

2004

VISION-BASED URBAN NAVIGATION PROCEDURES FOR VERBALLY INSTRUCTED ROBOTS

KYRIACOU, THEOCHARIS

<http://hdl.handle.net/10026.1/2778>

<http://dx.doi.org/10.24382/3721>

University of Plymouth

All content in PEARL is protected by copyright law. Author manuscripts are made available in accordance with publisher policies. Please cite only the published version using the details provided on the item record or document. In the absence of an open licence (e.g. Creative Commons), permissions for further reuse of content should be sought from the publisher or author.

VISION-BASED URBAN NAVIGATION PROCEDURES FOR VERBALLY INSTRUCTED ROBOTS

by

THEOCHARIS KYRIACOU

A thesis submitted to the University of Plymouth
in partial fulfilment for the degree of

DOCTOR OF PHILOSOPHY

School of Computing, Communications and Electronics
Faculty of Technology

May 2004

Author: Theodoros Kyriacou

Thesis title: Vision-Based Urban Navigation Procedures for Verbally Instructed Robots.

Abstract

The work presented in this thesis is part of a project in instruction based learning (IBL) for mobile robots where a robot is designed that can be instructed by its users through unconstrained natural language. The robot uses vision guidance to follow route instructions in a miniature town model.

The aim of the work presented here was to determine the functional vocabulary of the robot in the form of “primitive procedures”. In contrast to previous work in the field of instructable robots this was done following a “user-centred” approach where the main concern was to create primitive procedures that can be directly associated with natural language instructions. To achieve this, a corpus of human-to-human natural language instructions was collected and analysed. A set of primitive actions was found with which the collected corpus could be represented. These primitive actions were then implemented as robot-executable procedures.

Natural language instructions are under-specified when destined to be executed by a robot. This is because instructors omit information that they consider as “commonsense” and rely on the listener’s sensory-motor capabilities to determine the details of the task execution. In this thesis the under-specification problem is solved by determining the missing information, either during the learning of new routes or during their execution by the robot. During learning, the missing information is determined by imitating the commonsense approach human listeners take to achieve the same purpose. During execution, missing information, such as the location of road layout features mentioned in route instructions, is determined from the robot’s view by using image template matching. The original contribution of this thesis, in both these methods, lies in the fact that they are driven by the natural language examples found in the corpus collected for the IBL project.

During the testing phase a high success rate of primitive calls, when these were considered individually, showed that the under-specification problem has overall been solved. A novel method for testing the primitive procedures, as part of complete route descriptions, is also proposed in this thesis. This was done by comparing the performance of human subjects when driving the robot, following route descriptions, with the performance of the robot when executing the same route descriptions. The results obtained from this comparison clearly indicated where errors occur from the time when a human speaker gives a route description to the time when the task is executed by a human listener or by the robot.

Finally, a software speed controller is proposed in this thesis in order to control the wheel speeds of the robot used in this project. The controller employs PI (Proportional and Integral) and PID (Proportional, Integral and Differential) control and provides a good alternative to expensive hardware.

Contents

Abstract.....	3
Contents.....	5
Nomenclature.....	10
Abbreviations	13
List of Figures	14
List of Tables.....	19
Acknowledgements	22
Author's Declaration	23
Chapter 1.....	26
1 Introduction.....	26
1.1 Instruction based learning for mobile robots	26
The aim of this PhD thesis	30
1.2 Challenges	31
1.3 Methodology and original contributions	32
1.4 The author's contribution to the IBL project.....	33
1.5 Thesis overview	35
Chapter 2	38
2 Literature review.....	38
2.1 Natural language in robotics.....	38
2.2 Segmentation of natural language into functional components.....	43

2.3	Road layout recognition for navigation	45
2.4	Localization and mapping.....	49
2.5	Discussion	50
Chapter 3		52
3	Experimental environment setup and robot design	52
3.1	The components of the Instruction Based Learning system.....	52
3.2	The miniature town model	54
3.3	The robot	56
3.3.1	Robot hardware	57
3.3.2	Robot software.....	59
3.3.2.1	The PID controller used for robot speed control.....	61
3.4	Programming platforms and developed software	69
3.5	Summary/Contributions.....	71
Chapter 4		73
4	Corpus based system design	73
4.1	Corpus collection procedure	74
4.2	Word analysis.....	77
4.3	The primitive procedures in route instructions	81
4.3.1	The primitive procedures extracted from the corpus and their representation	82
4.4	The “go” primitive.....	90
4.5	The “go_until” primitive.....	90
4.6	The under-specification of natural language and how it affects the functional specification of the primitive procedures.....	92

4.6.1	The use of default parameter values	93
4.6.2	The reference to the destination landmark.....	94
4.6.3	The multiple meanings of “go”	96
4.7	The concept of corpus based designed system.....	97
4.7.1	Contribution of the corpus to the natural language system design.....	97
4.7.2	Contribution of the corpus to the robot system design	98
4.8	Summary/Contribution	100
Chapter 5		103
5	The functional structure of the primitive procedures	103
5.1	The structure of primitive procedures and how it reflects the structure of spoken instructions	104
5.2	The use of low-level procedures	107
5.3	The verification of new procedures during their creation	109
5.4	The “prediction” and “action” functions of primitive procedures	111
5.5	Summary/Discussion	115
Chapter 6		118
6	Vision for robot navigation.....	118
6.1	Capturing and pre-processing the robot camera image.....	120
6.2	Detection of road features using image template matching	127
6.2.1	Template matching.....	128
6.2.2	Road feature templates	129
6.2.3	The template matching procedure	131
6.2.4	How template matching is used in primitive procedures	137

6.3	The use of a short-lived map.....	140
6.3.1	The matching of new road edge data on the “short-lived” map.....	146
6.4	Road surface detection.....	148
6.4.1	Colour filtering using the chromaticity vector	149
6.5	Road edge detection	154
6.6	The need for a state variable during robot navigation.....	155
6.7	Detection of non road-layout objects	156
6.8	Spatial references to landmarks.....	161
6.9	Summary/Contributions.....	166
Chapter 7		168
7	The evaluation of the primitive procedures	168
7.1	Results per route instruction	171
7.1.1	Human performance tests.....	172
7.1.2	Error descriptions.....	175
7.2	Results per primitive procedure call.....	181
7.3	Discussion	183
Chapter 8		188
8	Conclusions and future work.....	188
8.1	Conclusions.....	188
8.2	Future work.....	192
8.3	Final statement	197
Appendices		199
Appendix A		199

Appendix B..... 218

Appendix C..... 219

Appendix D 223

List of References/Bibliography 228

Copies of Publications 238

Nomenclature

(d, e)	Real world coordinates in metres with reference to the robot's location.
(x, y)	Pixel location in a digital image with reference to the top-left pixel in the image.
(x_c, y_c)	Pixel location of the robot's camera image centre.
$[C_r, C_g]$	Chromaticity vector of a pixel location in a digital image.
$[x, y, \phi]$	Location (in pixels) and orientation (in radians) of the template during template matching.
A_{ij}	Action performed by the robot which changes its state from S_i to S_j .
C_{mean}	Mean chromaticity value of the road surface.
e	Speed error measured in encoder pulses per second. This is the difference between the desired speed (left-wheel, right-wheel or differential speed) actual speed of the robot. It is the input to the corresponding speed controller.
e_{dr}	The differential wheel speed error measured in encoder pulses per second. It is the input to the PID (Proportional Integral and Differential) controller.
e_l	The left wheel robot speed error measured in encoder pulses per second. It is the input to the left PI (Proportional Integral) controller.
e_r	The right wheel robot speed error measured in encoder pulses per second. It is the input to the right PI (Proportional Integral) controller.
b	The distance between the robot's camera and the ground plane.
I_l	The input to the left robot motor in volts.
I_r	The input to the right robot motor in volts.

k_f	Optical distortion coefficient.
K_p	The proportional constant of the two PI (Proportional Integral) speed controllers of the robot.
K_p'	The proportional constant of the PID (Proportional Integral and Differential) differential speed controller of the robot.
m	Value of a pixel location in the road map image.
M	The road map image matrix.
m'	Value of the information mask of a pixel location in the road map image.
n	Value of a pixel location in the road edge image.
N	The road edge image matrix.
n'	Value of the information mask of a pixel location in the road edge image.
p	Pixel location in the overlapping area of the road edge image and the road edge map.
Q_1	Matching quality of the template on the road surface map image.
Q_{1min}	Minimum acceptable matching quality signifying a successful match during the template matching.
Q_2	Matching quality of new road edge information on the road edge map image.
Q_{2min}	Minimum acceptable matching quality signifying a successful match of the road edge image on the road edge map.
S_i	The state of the robot before the execution of an action.
S_f	The state of the robot after the execution of an action.
s_l	The robot's left wheel speed in encoder pulses per second.
s_{lr}	The robot's left wheel requested speed in encoder pulses per second.

s_r	The robot's right wheel speed in encoder pulses per second.
s_{rr}	The robot's right wheel requested speed in encoder pulses per second.
t	Value of a pixel location in the template image.
T	The template image matrix.
t'	Value of the information mask of a pixel location in the template image.
T_d'	The differential constant of the PID (Proportional Integral and Differential) differential speed controller of the robot.
T_i	The integral constant of the two PI (Proportional Integral) speed controllers of the robot.
T_i'	The integral constant of the PID (Proportional Integral and Differential) differential speed controller of the robot.
T_s	The sampling interval (in seconds) of the robot's speed controllers.
V_{all}	The robot's battery terminal voltage in volts.
Δs_a	The actual differential speed of the robot. It is measured in encoder pulses per second.
Δs_r	The requested differential speed of the robot. It is measured in encoder pulses per second.
α	Maximum vertical angle of view of the robot's camera in radians.
β	Maximum horizontal angle of view of the robot's camera in radians.

Abbreviations

DM	Dialogue Manager
DRS	Discourse Representation Structure
HSI	Hue, Saturation and Intensity
IBL	Instruction Based Learning
PI	Proportional and Integral
PID	Proportional Integral and Differential
PSL	Procedure Specification Language
RGB	Red, Green and Blue
RM	Robot Manager

List of Figures

Figure 1-1: (a) Miniature model town and (b) robot used in the IBL project.	27
Figure 1-2: Processing modules of the IBL system.	29
Figure 3-1: The components of the IBL system.	53
Figure 3-2: The miniature town model.	55
Figure 3-3: (a) The robot used in the IBL project (80x80x160mm) and (b) The robot-football robot (80x80x80mm).	56
Figure 3-4: The hardware components of the robot.	57
Figure 3-5: Example of the robot's view.	59
Figure 3-6: Open loop speed control of each of the robot's motors. Where I_l and I_r are the left and right inputs to the motors in volts and s_l and s_r are the left and right wheel speeds respectively.	61
Figure 3-7: Closed loop speed control system with a PI controller for each motor.	64
Figure 3-8: The complete robot speed control system.	65
Figure 3-9: Screenshot taken during the development of the primitive procedures. The top-left window shows the "video server's" interface. The video server is an application, which continuously captures the image "seen" by the robot's camera and saves that into a file when requested by another application. The remaining image windows (apart from the command line window at the bottom) are "image monitor" applications, each used to monitor the changes of an image file during execution time.	70
Figure 4-1: A top view of the miniature town model indicating the starting point E (common to both routes) and destinations P and H referred to in Table 4-1.	77

Figure 4-2: Number of distinct words discovered in the corpus as the number of instruction samples increases. The long line is for all groups considered. The shorter lines are for groups A, B and C taken in isolation. Curves are obtained by averaging 50 random sets comprising an increasing number of sample route descriptions.	79
Figure 4-3: The description in Table 4-5(a) illustrated on the map of the miniature town. The dotted red line shows the route that the user implies to the robot and the solid red line is the route he/she explicitly describes.....	86
Figure 4-4: Number of distinct primitive procedures discovered in the corpus as the number of instruction samples increases. The long line is for all groups considered. The shorter lines are for groups A, B and C taken in isolation. Curves are obtained by averaging 50 random sets comprising an increasing number of sample instructions.	88
Figure 5-1: (a) Flowchart of “rotate” primitive procedure and (b) flowchart of all other primitive procedures extracted form the corpus.....	105
Figure 5-2: Primitive procedures can be accessed by users via natural language whereas low-level procedures cannot.	108
Figure 5-3: Illustration showing how the prediction function in primitive procedures is used. Row (a) shows a case were the state of the robot after executing procedure P1 is consistent with the next procedure to be executed P2. Row (b) shows the case where there is an inconsistency between the state the robot is left in after executing P1 and the expected state for the next procedure to be executed. For a more detailed explanation of the figure see text below.	110
Figure 5-4: Procedural knowledge representation.	115
Figure 6-1: An example of a raw (unprocessed) robot camera image.....	118
Figure 6-2: (a) Raw camera image and (b) the same image after optical calibration.....	121

Figure 6-3: Illustration showing how inverse perspective transformation is performed on the robot's camera image. (a) Shows an example of an optically calibrated camera image, (b) shows the result when inverse perspective mapping is applied to (a). Note the missing information due to the sampling effect in (b). In (c) the missing information is interpolated using neighbouring pixels containing information. (d) is the part of (c) used by the primitive procedures for further processing.	123
Figure 6-4: Diagram showing the correspondence of pixels on the image plane with pixels on the ground (or road surface) plane.....	125
Figure 6-5: Template (d) of Table 6-1 is used to represent a range of possible right turnings at different angles to the main road.....	131
Figure 6-6: Illustration of one position of the template image on the road surface map while searching for the best matching position.	133
Figure 6-7: Illustration showing how the matching of the template is performed on the road surface map in order to save computational time. This starts with coarse scanning (a) of the pivot point (of the template) in the map (1 in every 4 pixels shown in grey colour). For the second step an area S_1 (shown magnified in (b)) is selected for a finer scan (1 pixel in 2) around the position that produced the best matching quality in the first scan. Each side of S_1 is equal to twice the scan step in (a). In the same way scan area S_2 is selected from (b). All pixels of S_2 are scanned in order to give the best possible matching position.....	136
Figure 6-8: The best matching position of the left turn template on the road surface map. Note how the pivot point of the template indicates the next waypoint of the robot.	137
Figure 6-9: Users u9 and u24 were asked to explain a route starting from the Hospital (H). Their first instruction referred to the t-junction the robot would meet.....	138

Figure 6-10: Illustration showing the “dead angles” of the robot.	141
Figure 6-11: Vector diagram showing the robot’s actual displacement vector “ d ”, which is the vector sum of the robot’s intended displacement “ i ” and the odometric error “ r ”.	142
Figure 6-12: Series of figures showing how the “short-lived” map is appended with new visual information and how, as a consequence, this localizes the robot. For an explanation of how this is done see text below.	144
Figure 6-13: Illustration of one position of the top view image on the map while searching for the best matching position.	146
Figure 6-14: (a) An example of a top view image and (b) its corresponding road filtered version. White pixels denote road areas and black pixels denote non-road areas.	148
Figure 6-15: Improvements in order to improve illumination constancy in the miniature model town.	150
Figure 6-16: Series of images illustrating how the road surface colour is bootstrapped. For a detailed explanation of how this is done see text below.	153
Figure 6-17: The high-low-high intensity profile kernel (magnified by a factor of 10) that is convolved with the top view image to discriminate the road edge lines.	154
Figure 6-18: (a) An example of a top view image and (b) its corresponding road edge image.	155
Figure 6-19: Examples of non road-layout objects mentioned in the corpus: (a) signed building, (b) unsigned building, (c) the bridge, (d) trees.	157
Figure 6-20: Examples of placing a coloured marker in front of objects to be able to locate them.	158
Figure 6-21: (a) An example of a camera image, (b) the corresponding top view image and (c) the top view image filtered for the colour of the marker of the landmark sought.	159

Figure 6-22: Illustration showing the location that the robot must reach to execute the instruction: “follow the road to Safeway” in comparison with the actual location of the “Safeway” building.	160
Figure 6-23: Illustration showing how the road waypoint representing a reference to a landmark is found.	161
Figure 6-24: (a) Camera view and (b) the corresponding top view before the “university” when the robot follows the instruction: “before reaching the university’s main door take the road to your right”. Notice that when the “university’s door” is visible, the “road to the right” is still within the robot’s view.	165
Figure 7-1: The diagram shows the occurrence of errors at different stages between the speaker giving route descriptions and the execution of these descriptions by (a) a human listener and (b) the robot. K_s , K_H and K_R represent the knowledge of the human speakers, the human listeners and the robot’s respectively. The diagram shows the crucial difference between cases (a) and (b): the ability of humans to do repair during the execution of the route instructions. This accounts for the higher success rates of humans.	186
Figure 8-1: A case when the user says “turn left” when he/she actually means “take the first exit off the roundabout”.....	194

List of Tables

Table 4-1: Examples of “short” and “long” route descriptions.	76
Table 4-2: Most frequent and least frequent user word in the corpus. The least frequent words were found only once in 96 route descriptions.	78
Table 4-3: Primitive procedures extracted from the collected corpus of route descriptions. ...	83
Table 4-4: Examples of primitive procedures extracted from the corpus and their corresponding primitive procedure calls.	84
Table 4-5: An example of a translation of a route description to its corresponding primitive calls. Row (a) shows the transcribed version of the route description u7_GC_CX. User 7 explains the route from Boots (C) to the Post-office (X) (see Figure 4-3). Row (b) shows the corresponding manual translation of the description to its primitive procedure calls.	85
Table 4-6: Parameter types and possible values they can take in primitive procedure calls.	89
Table 5-1: Examples of low-level procedures.	108
Table 5-2: Pseudo-code of the prediction function in primitive procedure modules.	113
Table 6-1: The templates used for the template matching method. Light grey colour indicates road-like areas and the black colour represents non-road areas. The templates shown are used to find: (a) straight road, (b) end of road, (c) left and (d) right turnings, (e) crossroad, (f) left and (g) right bends, (h) t-junction, (i) roundabout entry, (j) clockwise and (k) anti-clockwise curved road in roundabout, (l) left and (m) right roundabout exits, (n) left and (o) right 90-degree turns.....	130
Table 6-2: Pivot point (dot-centred circle) and direction vector (arrow) for some of the templates.	132

Table 6-3: Words in the corpus that indicate a relation between the robot and a landmark along with an explanation of the robot's final location with respect to the landmark's location.	164
Table 7-1: Route description success results during the development and evaluation of the primitive procedures. Note that the percentage values indicate the proportions of the executed route descriptions and not the total route descriptions.	172
Table 7-2: Route description success results for the evaluation set when executed by the robot and by the human subjects.	173
Table 7-3: Analysis of the 26 cases where the robot fails to reach its destination in the evaluation phase and comparison with the performance of human subjects in the same routes.	174
Table 7-4: Route descriptions and illustrations of the two cases where the robot fails to complete a route description in the evaluation set because of new primitive procedure calls. Note that the solid red line indicates the road described by the user in each case and the dashed red line indicates a route implied by the user.	175
Table 7-5: Route descriptions and illustrations of the two cases where the robot fails to complete a route description in the evaluation set because of primitive procedure failures.	177
Table 7-6: Route descriptions and illustrations of two (out of five) cases where the robot fails to complete a route description in the evaluation set because of ambiguities in the user's description.	178
Table 7-7: Route descriptions and illustrations of two (out of seventeen) cases where the robot fails to complete a route description in the evaluation set because of mistakes in the user's description.	180

Table 7-8: Primitive call success results during the development and evaluation of the primitive procedures. Note that the percentage values indicate the proportions of the executed primitive calls and not the total primitive calls.	181
Table 7-9: The robot fails to “see” the Boots (C) marker, at the end of the route, because when it turns the marker falls outside its field of view.	182

Acknowledgements

First of all, I would like to thank my supervisor Dr. Guido Bugmann for giving me the opportunity to work on such an ambitious and challenging project. Mostly I would like to thank him for his continuous support and guidance throughout the duration of my work. The valuable things I learned from him will accompany me throughout my professional life.

There are no words to express my gratitude towards my parents Kyriacos and Despo to whom I attribute only the best of who I am and will ever become. They taught me to respect and appreciate myself and always have been by my side to support and guide me. I feel honoured and privileged to be their son.

I would like to dedicate this thesis to my beloved wife and best friend Maria. Her continuous love and care made this work possible. Also this thesis is dedicated to the little one growing within her who, without knowing it, made possible for me to see the end of this work.

Author's Declaration

At no time during the registration for the degree of Doctor of Philosophy has the author been registered for any other University award.

This study was supported by the Engineering and Physical Sciences Research Council (EPSRC).

A programme of advanced study was undertaken, including one postgraduate course in Computational Modelling.

Relevant scientific seminars and conferences were regularly attended at which work was often presented; external institutions were visited for consultation purposes and several papers were prepared for publication.

Publications:

- Lauria S., Bugmann G., Kyriacou T., Bos J., Klein E., **Personal Robot Training via Natural-Language Instructions**, IEEE Intelligent Systems, 16:3, 2001, pp. 38-45.
- Bugmann, G., Lauria, S., Kyriacou, T., Klein, E., Bos, J., Coventry, K., **Using Verbal Instructions for Route Learning: Instruction Analysis**, Proceedings of TIMR 2001 (Towards Intelligent Mobile Robots), Manchester, 2001. Technical Report Series, Department of Computer Science, Manchester University, ISSN 1361-6161. Report number UMC-01-4-1.

- Lauria, S., Bugmann, G., Kyriacou, T., Klein, E., **Instruction Based Learning: how to instruct a personal robot to find HAL**, Proceedings of the 9th European Workshop on Learning Robots, EWLR-9, Prague, Czech Republic, September 2001, pp. 15-24.
- Lauria, S., Bugmann, G., Kyriacou, T., Bos, J., Klein, E., **Converting Natural Language Route Instructions into Robot Executable Procedures**, Proceedings of the 2002 IEEE International Workshop on Robot and Human Interactive Communication (ROMAN '02), Berlin, Germany, 2002, pp. 223-228.
- Lauria, S., Bugmann, G., Kyriacou, T., Klein, E., **Mobile Robot Programming Using Natural Language**, Robotics and Autonomous Systems, 38 (3-4), 2002, pp. 171-181 (ISSN 0921-8890).
- Kyriacou, T., Bugmann, G., Lauria, S., **Vision-Based Urban Navigation Procedures for Verbally Instructed Robots**, Proceedings of the 2002 IEEE/RSJ International Conference in Intelligent Robots and Systems (IROS 2002) EPFL, Lausanne, Switzerland, October 2002, pp. 1326-1331.
- Robinson, P., Bugmann, G., Kyriacou, T., Culverhouse, P., Norman, M., **MIROSOT: A teaching and learning tool**, Proceedings of the 2002 FIRA Robot World Congress, edited by Korea Robot Soccer Association, 2002, pp. 309-314 (ISBN: 89-86522-47-0-93560).
- Kyriacou, T., Bugmann, G., Lauria, S., **Vision-Based Urban Navigation Procedures for Verbally Instructed Robots**, Proceedings of TIMR 2003 (Towards Intelligent Mobile Robots), Bristol, 2003.

Meetings and Conferences attended:

- European Robotics Research Network (EURON) Kick-off Meeting, Gran Canaria, Spain, January 2001.
- Towards Intelligent Mobile Robots (TIMR 2001). 3rd British Conference on Autonomous Mobile Robotics and Autonomous Systems, Manchester, United Kingdom, April 2002.
- International Conference on Intelligent Robots and Systems (IROS 2002), Lausanne, Switzerland, September 2002.
- Towards Intelligent Mobile Robots (TIMR 2003). 4th British Conference on (Mobile) Robotics, Bristol, United Kingdom, August 2003.

Summer schools attended:

- EPSRC/BMVA Summer School in Computer Vision, Surrey, United Kingdom, June 2001.
- European Summer School on Mobile Robot Navigation (organized by the European Robotics Research Network – EURON), Lausanne, Switzerland, September 2001.

Signed 

Date 1-5-2004

Chapter 1

1 Introduction

The work presented in this thesis is part of an EPSRC funded project¹ in Instruction Based Learning (IBL) for mobile robots. The first section of this chapter gives a short introduction of the IBL project in order to set the scene for the work presented here. Section 0 describes the main aim of this thesis followed by section 1.2, which explains the main challenges presented in achieving the aim. Section 1.3 presents the methodology followed in order to achieve the aim of the thesis and gives a brief summary of the original contributions to knowledge made by this work. Section 1.4 specifies how the author of this thesis contributed to the Instruction Based Learning group project. Finally, section 1.5 gives an overview of the work presented in this thesis.

1.1 Instruction based learning for mobile robots

The idea behind the IBL project is that future robots will need to adapt to the special needs of their users and to their environment. It is likely that programming by natural language will be a key method enabling computer language-naïve users to instruct their robots. The project attempts to investigate the issues involved in building a robotic system able to learn from

¹ Grants GR/M90023 and GR/M90160.

verbal instructions. The proposed methodology is tested in the restricted domain of route instructions.

For the purposes of the project a miniature model town and a robot with an onboard video camera were built (Figure 1-1).

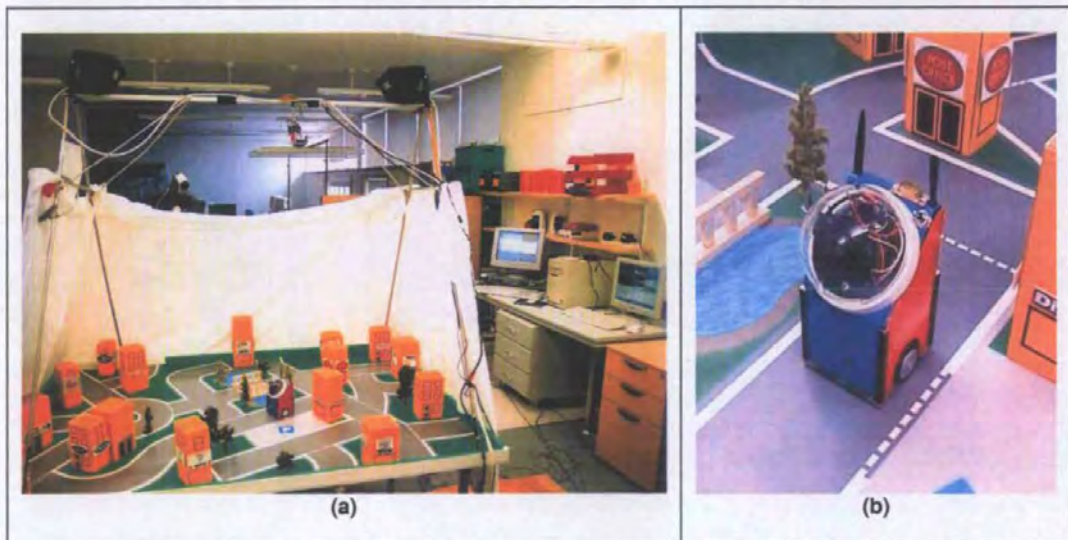


Figure 1-1: (a) Miniature model town and (b) robot used in the IBL project.

The robot is able to navigate in the miniature town following natural language route descriptions given by a human user. As example of the interaction between the user and the robot, consider the following scenario:

The robot is at the Museum and its user wants it to go to “Boots”. The user starts by asking the robot to go to “Boots” by first calling it and then giving the instruction.

The discourse between the user and the robot is as follows:

User: “Robot”

Robot: “Yes”

User: “Go to Boots.”

Robot: “How do I go there?”

User: “Take the third turning to the left...”

Robot: “Next instruction please”

User: “...follow the road to the roundabout...”

Robot: “Next instruction please”

User: “...take the third exit off the roundabout...”

Robot: “Next instruction please”

User: “...take the first right...”

Robot: “Next instruction please”

User: “...Boots will be on your left after the road bend.”

At this point the robot starts to navigate in the miniature town in order to reach

“Boots” following the route descriptions given by the user. The robot informs the user when it reaches the destination by saying:

Robot: “OK, it’s done”.

The architecture of the IBL system is comprised of several functional processing modules shown in Figure 1-2.

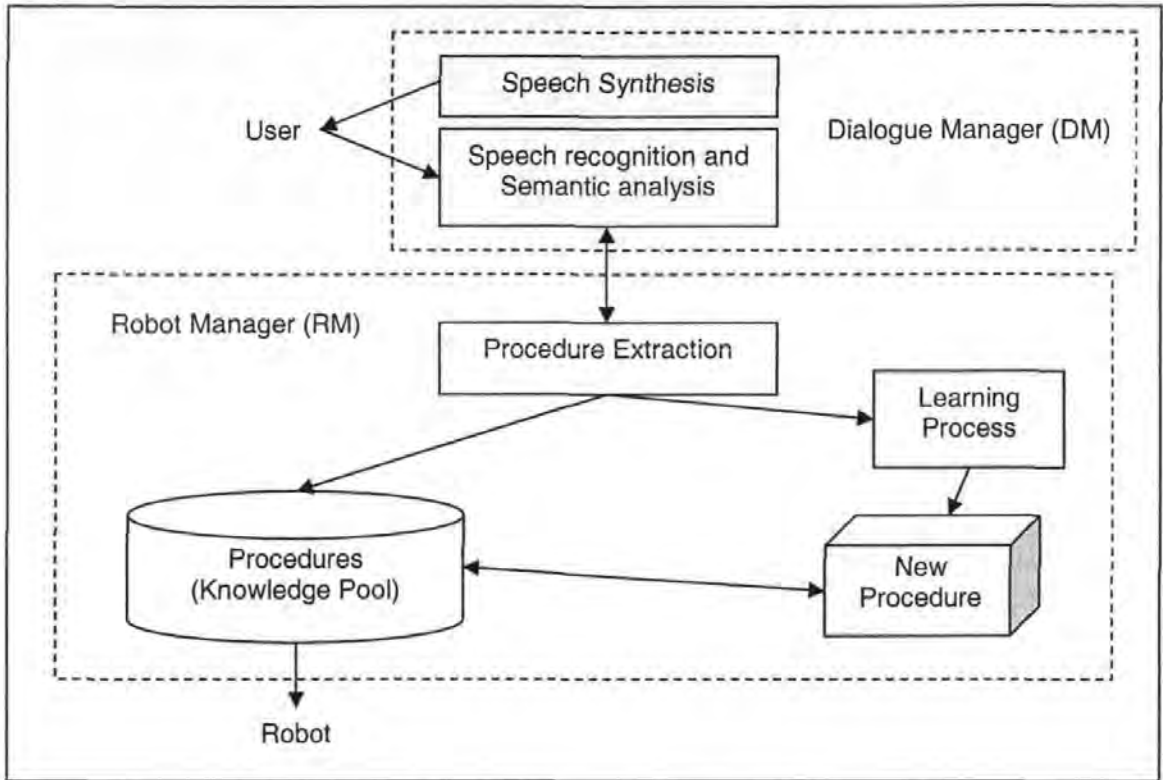


Figure 1-2: Processing modules of the IBL system.

These are divided into two major units: the Dialogue Manager (DM) and the Robot Manager (RM). The Dialogue Manager is a bi-directional interface between the Robot Manager and the user, either converting speech input into a Discourse Representation Structure or DRS (this is a semantic speech representation explained in more detail in [Traum et. al., 1999]), or converting requests from the Robot Manager into dialogues with the user. Its detailed function is described in [Lauria et. al., 2001].

The robot manager deals with the dialogue manager's output and also with the learning and execution of the commands from the user. Its function is to map the semantic representation produced by the dialogue manager into robot executable procedures in the knowledge pool. This mapping is done using a meta-language called PSL (Procedure Specification Language),

which is described in [Lauria et. al., 2002]. If the mapping from the semantic representation of the user's instructions to procedures in the knowledge pool of the system is successful, the corresponding procedures that would cause the robot to do what the user asked are executed. On the other hand, if a requested action does not exist in the knowledge pool, the robot manager initiates a learning process to learn it and then execute it. New procedures can only be composed from previously learned ones in the knowledge pool. This implies that an initial set of "primitive" procedures must exist in the knowledge pool before the robot starts learning for the very first time.

The aim of this PhD thesis

The aim of this PhD work was to determine and implement the primitive procedures of the robot used in the Instruction Based Learning project.

The primitive procedures reflect those actions that users expect that the robot knows how to perform without any further explanation. As an example, consider the action "turn", which is one of the primitive procedures implemented in this project (see chapter 4). Users assume that the robot knows how to turn. However, a quite complex and precise set of sub-actions need to be executed in order to achieve a turn (image processing, robot wheel speed and distance control etc.). This thesis work was concerned with finding what actions users expect the robot to know when it begins its "life" and also with determining the underlying program code that would produce the desired (by the user) behaviour in each case.

1.2 Challenges

An important aspect of the Instruction Based Learning system is that it is designed using a user-centred rather than a robot-centred approach. This means that it is able to cope with unconstrained spoken language instructions as a human would. This design target presented the challenge of creating a robotic system that can be used by humans without requiring prior training. A robot-centred approach on the other hand would be to build a robot and then train human users to use its special language in order to be able to communicate with it. This latter approach would, of course, contradict the main idea of this project.

Primitive procedures must, therefore, directly correspond to actions found in natural language. To result in the robot performing the requested (by the user) action, no more information should be required than that provided in natural language.

Natural language instructions are underspecified when it comes to them having to be executed by a robot. For example consider the case when a user instructs the robot to “take the left turning”. In this case there is no indication that the user means the “first” left turning. More importantly, there is no information as to where the left turning is or how a “left turning” looks like. These are vital bits of information, which are required by the robot in order to be able to successfully execute the above instruction. It is therefore essential for the validity of the Instruction Based Learning concept that it is possible to design primitive procedures that can cope with this under-specification in natural language. This issue is crucial to the development of the Instruction Based Learning system and it is of central importance to the development of the primitive procedures described in this thesis.

1.3 Methodology and original contributions

To achieve the aim of this thesis, a corpus of human-to-human route instructions was initially collected and later analysed in order to determine what “built-in” knowledge and ability human speakers expect their listeners to possess for following route instructions. This user-centred approach used to determine the functional vocabulary of the robot constitutes the main contribution of this thesis.

The corpus collected was transcribed and then analysed for its word and functional content. A method was developed for writing primitive procedures that are robust in coping with the under-specification problem of natural language route instructions. This method involves the determination of missing information in route instructions either during the time when the robot is being instructed by the user or during the time when the robot is executing the route instructions. To determine the missing information during instruction-time the corpus route instructions as well as the robot’s environment to which these referred to were studied in order to determine what human instructors considered as “commonsense” and therefore omitted in their instructions. This information was then used as default in the primitive procedures when it was not explicitly given by the instructors. During the execution of route instructions the robot uses artificial vision in order to determine missing information such as the location of road layout features mentioned in route instructions. The method of template matching is used in order to detect the sought features by trying to match template images (representing the sought features) to the robot’s view. The original contribution of this thesis with regard to this method lies in the fact that the design and use of the image templates was

purely influenced by the content of the collected corpus of route instructions. This was done in order to follow the user-centred design methodology.

A method is proposed in order to evaluate the performance of the primitive procedures created when they are executed as part of complete route descriptions. This was done by inviting human subjects to drive the robot following route descriptions. The performance of the human subjects was then qualitatively compared to that of the robot's for each route description. The differences found allowed for important conclusions to be drawn and gave indications for the course of future work.

In order to reliably control the speed of the robot used in this project an elaborate robot speed control system is proposed in this thesis. This is based two PI (Proportional and Integral) and one PID (Proportional, Integral and Differential) controllers in order to control the speed of each wheel and the differential speed of the robot.

1.4 The author's contribution to the IBL project

The following is an outline of the work contribution of the author of this thesis to the Instruction Based Learning project:

- Miniature town design and implementation (see section 3.2).
- Robot software and hardware integration. The existing robot hardware was integrated with a video camera, video transmitter (see section 3.3.1) and electronic circuits to enable easier battery charging while an external power supply was powering the robot.

Likewise, existing robot software was greatly changed to suit the robot performance requirements for this project. The most significant of these changes included the design and implementation of a software controller in order to control the speed of the robot in a reliable manner (see section 3.3.2).

- Assistance in corpus collection. The author was involved in all aspects of the corpus collection such as the preparation for each route description according to the corpus collection protocol, the briefing of subjects prior to the beginning of the experiments and the recording of route descriptions. Also the author was involved in the corpus collection of dialogues by taking the place of the robot in the dialogues and producing the responses that the robot would.
- Corpus lexical and functional analysis. The results of these analyses are presented in sections 4.2 and 4.3. The functional analysis was fundamental to the determination of the primitive procedures that needed to be implemented for the robot in this project.
- Primitive procedure implementation. This is the main part of the work described in this thesis.
- Assistance in collecting data for testing of the IBL system. The author of this thesis was involved in setting up the test environment for human subjects to drive the robot following route descriptions given to them. Also assistance was offered during the experiment in order to ensure adherence to the experimental protocol.
- Testing of the primitive procedures. This was done by evaluating the performance of the primitive procedures developed in this project both on an individual basis and as part of complete route instructions.

Apart from the above, it is also important to note that the author took part and contributed in all aspects of the Instruction Based Learning project during all the meetings and discussions with the other members of the project.

1.5 Thesis overview

This section gives a brief description of the contents of each chapter in this thesis.

Chapter 2 presents a review of previous work related to this thesis. It mainly summarises work in the three areas related to this project. These are:

1. The analysis of natural language, in the specific context of route descriptions, in order to determine the functional components of the task domain,
2. Methods for road layout recognition for navigation.
3. Robot localization and mapping.

Chapter 3 describes the experimental environment setup used in this project. The setup includes the miniature model town, the robot and two PCs, which act as the “remote brain” of the robot.

Chapter 4 explains how the corpus of natural language route instructions was collected and later analysed. The results from the word and functional analysis are also presented. The functional analysis revealed the primitive procedures that need to be created as part of the robot’s “built-in” knowledge. The functional analysis also revealed the issues concerned with

implementing a natural language system. The work in this thesis focuses on one of these issues. This is the “under-specification” of natural language. In this chapter three methods are proposed to overcome this problem.

Chapter 5 explains how the program code in primitive procedures is organized and the reasons behind this organization. The chapter mainly describes the need of a “prediction function” associated with each procedure in the knowledge base of the robot. This function is used during the learning of new procedures in order to verify that the series of route instructions given by the user can be executed. The “prediction function” is used in order to identify errors either in the user’s route description or in the processing of the route description by the system.

Chapter 6 describes how image template matching is used for determining the location of landmarks mentioned in route descriptions. The same chapter explains how the robot creates a small map of its immediate locality, called “short-lived” map. This map is used for two purposes:

1. To be able to determine the odometric errors of the robot and
2. In order to be able to retain in memory visual information close to it, which go out of its view as it navigates.

Chapter 7 describes how the primitive procedures developed in this project were evaluated.

Chapter 8 presents the original contributions of this work and gives suggestions for future work on this project.

Chapter 2

2 Literature review

2.1 Natural language in robotics

Programming human helper robots to perform even simplest of tasks is a time consuming and complex process. This is impossible to achieve for computer language naïve user. For such helper robots to be useful, a method of communication must be used between humans and robots that will not demand special technical and programming abilities from humans. This observation is made, among many texts, in [Crangle and Suppes, 1994] where it is stated that for efficient human-robot interaction the user should not have to become a programmer, or rely on a programmer. Also the user should not need to learn specialized technical vocabularies to request an action from a robot. In [Hausser, 2001] the difficulty of using a computer language for most potential users is stressed. The three main reasons given are: (a) users are not familiar with the operations of the machine, (b) the expressions of the programming language are different from those of every day language and (c) the use of the programming language requires great precision. This last point is also mentioned in [Huffman and Laird, 1993], which says that computer language procedures must be specified in complete detail and that no steps may be omitted or abstracted.

It is likely that natural language will be a key method for computer language naïve users to program their robots in the future. In [Herzog, 1995] it is stressed that natural language is a natural communication medium for humans and so systems, which are able to communicate in natural language meet human needs much better. In [Allen et. al., 1996] human subjects preferred to use the speech interface when given the choice between that and a keyboard interface in order to communicate with a natural language system.

In the last few years there have been numerous attempts to create robots that communicate with humans using natural language. In [Fong et. al., 2001] a robotic system is presented, which can engage into spoken dialogues with humans in order to request information that would assist it during navigational tasks. The robot initiates a dialogue with a human only when it is faced with a problem during navigation. The robot can produce a limited amount of messages (around 30) to its human user to which it expects simple answers such as “yes”/“no” or a numerical value. These messages are pre-defined by the creators of the robot by studying the possible problems that the robot could face during its navigational task. Furthermore, users are expected to have a certain amount of expertise regarding the robot and the environment in order to collaborate usefully with the robot. Here, the robot’s navigational task is already programmed by a robot expert and in effect, natural language dialogue is only used to make a selection between 2 or more alternatives of an already written program code. In [Spiliotopoulos et. al., 2001] a mobile robotic assistant for hospitals is described. The robot can be instructed (using natural language) to deliver a medicine or a message to a specific room or patient. Its users can also request database information such as the phone or room number of a patient. A simple state-based dialogue management technique is used. The robot leads the dialogue by asking specific questions for the user to answer. In this way the system is

made robust to speech recognition errors and also fast in its responses because it only needs to do simple language processing. However, the system restricts its user to a specific dialogue frame that can be uncomfortable for him.

In the robotic systems described above the dialogue structure is predefined. The robots lead the dialogue and their users are expected to respond with specific information. Users are constrained by a limited choice of answers. While these systems are robust because it is easier to recognize the user's utterances, they are limited as far as dialogue flexibility. In [Allen et. al., 1996] it is said that it is a fundamental requirement for natural spoken dialogue systems that the user should not be constrained in what can be said. Furthermore the above systems cannot learn new tasks. This means that they would be unable to adapt to the needs of individual users but rather, users have to understand how the robots work and how to use them.

Systems that can learn new tasks are described, among other, in [Huffman and Laird, 1993], [Asoh et. al., 2001], [Bischoff and Jain, 1999] and [Torrance, 1994]. In [Huffman and Laird, 1993] a virtual mobile robotic arm is instructed (using interactive natural language dialogue) to perform simple actions such as object displacement. The robotic agent can learn new actions composed of a sequence of known actions or sequences. The user is restricted to using simple imperative sentences such as: "Pick up the red block", "Move to the yellow table", "Move the arm above the red block" etc. In [Asoh et. al., 2001] an office robot is described. The robot communicates in natural language but understands five tasks (or task frames): database query, database update, person identification, navigation and person calling. Each task frame requires several necessary parameters (slots to be filled in the task frame) before it can be executed.

The dialogue manager of the system tries to identify these parameters from the user's utterance. A similar system is described in [Bischoff and Jain, 1999]. Here, the robot can understand several actions that are stored in the robots memory as prototypes with a single verb to describe the underlying action and several parameters (mandatory or not). An attempt is made to cover all possibilities of an action mentioned in natural language by creating more than one prototype for the action where each prototype has a different number and configuration of parameters. The system chooses the right word elements from the user's utterance in order to satisfy a prototype and then execute its associated action. In [Torrance, 1994] an indoors mobile robot is described, which follows typewritten instructions in natural language. The robot recognizes several types of instructions, some causing the robot to move and others are changing or interrogating its state. Although not explicitly mentioned, it seems that the system tries to match particular instruction templates on the user's text input and then uses the key words to modify its state variables.

In the above cases the user is constrained in using a closed set of template natural language instructions. These template instructions were designed with none or very little investigation of the structure of unconstrained natural language in the context used. Rather, a robot-centred approach was used, i.e. the template instructions were tailored according to the specifications of the robotic system being used. As with computer language programming, this requirement means that a user must be trained in order to be able to use the system.

An attempt for a user-centred approach was made in [Green and Severinson-Eklundh, 2001] where an indoor mobile robot, capable of natural language communication, is described. "Wizard of Oz" experiments were conducted during the project in order to investigate the

behaviour of subjects when speaking to the robot. The “Wizard of Oz” technique is described in [Dahlback et. al., 1993]. With this technique parts of an interactive system (that are not yet developed) are substituted by humans who try to mimic the behaviour of the future system. The purpose of this is to be able to study the responses of the users of the system. In [Green and Severinson-Eklundh, 2001] these experiments were not used to examine the structure and functional content of naturally spoken language. The tasks that the robot is able to perform were decided and implemented before the Wizard of Oz experiments. The experiments were used to determine the lexicon of the context, the states that the human-robot dialogue can take (question, answer, repair etc.) and the type of feedback a user requires from the robot but not what the user will need from such a robot and in what way he/she is likely to ask for it. This is therefore another example of a robot-centred approach in order to determine the functional vocabulary of the robot. Users of the robot would need to be trained on how to command the robot using a restricted set of commands that are predetermined by the robot’s creators.

In this thesis the functional vocabulary of the robot is determined using a user-centred approach. To achieve this, a corpus of route instructions was collected from prospective users of the robot. The corpus was then analysed in order to determine the nature of the instructions users give to the robot in the route description context.

2.2 Segmentation of natural language into functional components

Previous work on natural language analysis has shown that human free speech can be segmented into “atomic” units for the purposes of simplifying the analysis of long utterances and therefore enable the better understanding of their structure. For example in [Schleidt and Kien, 1997] it is found that human speech can be segmented into small speech units lasting a few seconds each. Speech units are defined as the “meaningful blocks” or “semantic phrases” where the meaning of a string of sounds is clear to an observer.

Based on the above findings, in order to determine the speech units in the route description context, the corpus collected for this project was manually segmented using task related criteria. Task oriented segmentation looks into the action described by the speech segments. It was expected that the speech module developed for the IBL (Instruction Based Learning) system would eventually produce the same result using syntactic and prosodic cues. A syntactic segmentation approach looks into the arrangement of words in order to establish grammatically sound sequences. A prosodic speech segmentation approach uses cues such as voice pitch, intonation, loudness, rhythm and stress in order to determine the limits of speech segments.

Previous examples of task related segmentation in the context of route descriptions are found in [Denis, 1997] and [Fraczak, 1995]. In [Denis, 1997] a corpus of route descriptions was analysed in order to investigate how humans externalise their understanding of space. For this, it was necessary to segment the route descriptions given by the subjects into small segments

called “chunks”. A chunk was defined as the speech unit that provides the smallest possible piece of information. In [Fraczak, 1995] a text-to-image translator is described, which translates route descriptions into sketches. A corpus of route descriptions was segmented into sequences and connections. Sequences are speech units describing action or introducing a landmark and connections are single words connecting two sequences. No indication of the size of sequences is given but from the examples mentioned they seem to be the smallest possible units of meaningful speech.

Following a similar approach as in the above mentioned work, in order to segment the collected route descriptions in this project a single criterion was used to define a speech segment: a speech segment is the smallest unit of speech that describes a single action.

However, speech segments cannot be directly associated with primitive procedures, i.e. a primitive procedure cannot be written to represent each speech unit found in the corpus. This is because the actions described by speech units do not always specify a final state, which is essential for the robot in order to execute the action. As an example consider the utterance:

“follow the road until you reach the post-office”

Although “follow the road” describes one action, it cannot be executed by the robot because it contains no information as far as when to stop following the road. Because of this, more than one speech units need to be combined into “functional components”. As described in section 4.3, in the example given above the whole utterance is considered as a functional

component in order to encompass the piece of information that gives the final state (i.e. to reach the post-office).

2.3 Road layout recognition for navigation

All primitive procedures in a route description scenario cause the robot to navigate on the road or confirm its position in relation to other landmarks. Therefore, the robot must be able to determine the road layout in order to navigate effectively and also be able to locate landmarks relative to the road.

This section gives examples of past work done in order to recover the road layout from an image observed by a camera onboard a moving vehicle. Methods use road surface features and/or road edge features to discriminate the road from the image. In [Waxman et. al., 1987] the components of a system used for visual land vehicle navigation are described. In this system the road is discriminated by its edges, which are approximated by pairs of parallel line segments. The line segments are found using edge detection and then selecting those pairs of lines that are parallel (in the real world) and which intersect at a vanishing point in the image. In [DeMenthon and Davis, 1990] a method is presented for reconstructing a 3D road model from a single image. As a first step, edge detection is performed on the image. The algorithm tries to find points lying on the edges of the road making reasonable hypotheses on the shape of the road, which add enough constraints to make the problem solvable. Such hypotheses are for example that the road is of uniform width, the road does not tilt sideways (zero-bank constraint) and that road edges are approximately parallel at opposite points. The aim is to calculate a possible centreline path along the road for a moving vehicle. In [Kaske et. al., 1997]

a vision-based method is suggested for finding the road edges in country roads where the edges are not very clear. This is done using statistical information from the image such as energy, homogeneity and contrasts that, when considered in combination, give distinctive results at the edges of the road. The location of the road edges, found in previous images that are taken as the robot moves are translated using the vehicle's motion vector to find the expected new positions. These are used with the results from the current image to eliminate false positives. In [Sayd et. al., 1998] a method is described to determine the location of a vehicle on a non-marked road. In order to extract the road surface from the image seen by a camera on the vehicle a small area in front of the vehicle (bottom of the image) is sampled. Assuming that the sample falls on the road surface, pixels in the image are classified as road or non-road pixels depending on how close their luminance is compared to the luminance of the pixels in the sample area. The vehicle location is then calculated by finding the road sides from the extracted road area assuming constant road width. In [Wilson and Dickson, 1999] an algorithm is suggested for tracking the boundaries of the road. The road boundaries are modelled as chains of line segments. The line segments are found incrementally starting from the bottom of the image (road close to vehicle) and moving towards the top (road far from the vehicle). The algorithm uses the endpoint of the previously found boundary segment (pivot point) as the start of the next segment. The slope of the new segment is determined by rotating a rectangular window with one of the smaller sides fastened at the pivot point and counting the number of edge pixels it encloses at each angle step. The angle that gives the maximum edge pixels is the slope of the next road boundary segment. The method suggested fails when the road boundary becomes discontinuous. The algorithm may also fail if the road surface where the initial search is done contains scars or other markings. In [Wang et. al., 2000] the Catmull-Rom spline algorithm is used to create curve models of the road edges. The

fitness of the models is then tested on the real image seen by the vehicle. The control points (of each road edge) used for the spline algorithm are selected from the edge-filtered image by first assuming that the road plane is flat and the road is of uniform width. A pair of points is taken to lie on opposite sides of the road if the tangents to the line segments they belong to, meet at the same point (vanishing point) on the horizon. After the application of the spline algorithm the two curve models (one for each side of the road) are superimposed on the edge-filtered image to test their fitness.

All of the methods mentioned above deal with the case of a straight or curved road extending in front of a vehicle. These methods are effective in cases where a vehicle needs to keep in the middle of a road lane when following a highway for example. However, they are unsuitable in more complex urban environments where the vehicle needs to turn at an intersection or a t-junction, use a roundabout etc. Further, all methods require that both edges of the road are visible in the image (though not necessarily continuous in all cases) to be able to recover the road.

Methods to recognize intersections on the road were proposed in [Jochem et. al., 1996] and [Crisman and Thorpe, 1993]. In [Jochem et. al., 1996] a neural network based vision system used for vehicle navigation is described. The neural network is first trained with images of straight road. During normal operation the image from the camera onboard the vehicle is used to produce images that could have been seen by “virtual cameras” positioned at different locations in front of the vehicle. The neural network uses the “virtual images” as input and produces a confidence measure indicating the presence of a straight road segment in them. By knowing the location of the virtual camera in relation to the actual camera, the system can

calculate the position of a straight road segment when the virtual camera image produces a high confidence value. In order to model an intersection the system requires the knowledge of either the position of the intersection (where the road branches meet), to determine its precise layout, or the layout of the intersection (i.e. the angles of the branches), to find its position. In the case of the IBL system, such a-priori information is not available in route instructions.

In [Crisman and Thorpe, 1993] dynamic model-building and matching are applied on a road surface likelihood image to determine the layout of the road. The road surface likelihood image is obtained by first clustering pixels of similar colour in the image (both on and off-road) and then using Bayesian estimation to classify the pixels in each cluster as road or non-road. The classification method uses information from the previous image analysis. The intersection detection method effectively finds intersections spurring from a straight road but would fail to find an intersection on a curved road or an exit from a roundabout for example.

In the case of a robot following natural language route instructions, there is a need for a method that will efficiently use the information provided in the route instructions in order to successfully cause the robot to navigate on the road. Such a robot must have knowledge of the geometry of the road layout features mentioned in route instructions, be able to find them and also be able to use them as instructed. It is also important that the representation of road layout features that the robot will have in its memory is generic enough to represent all variations of a road layout feature that are classified under the same name in route instructions.

The well known method of image template matching is used in this thesis to locate landmarks mentioned in route instructions. The road layout features mentioned in route descriptions are represented by template images, which are generic enough to cover all variations of the road feature in the robot's environment. The location of a road feature is found by matching the associated template in the robot's visual field.

2.4 Localization and mapping

The robot in this project needs to retain in memory visual information close to its immediate locality, which is "seen" by its camera but goes out of view as the robot follows the road. This information is retained by creating a local map (called "short-lived" map) of the robot's environment on which previously seen and current visual data coexist. In order to merge previous visual data with current a simplified method of image "mosaicing" is used. This method is described, among many, in [Unnikrishnan and Kelly, 2002a]. Image mosaicing for mapping is a method that uses a series of images taken, for example by a moving robot, which overlap such that each image contains a portion of the scene in common with both the image before and after it in the sequence. The aim is to create a map of the environment traversed by the robot by successfully linking the sequence of images. The great challenge presented to this task is loop closure in cyclic environments (see [Unnikrishnan and Kelly, 2002b] and [Gutmann and Konolige, 1999]). Loop closure requires the successful link of the first and last of a sequence of images taken when the robot's trajectory closes a loop. This problem arises because every time one image is linked to the next in the sequence a matching error is introduced due to image noise. This error increases with the number of images in the sequence and therefore with the distance travelled by the robot.

In this project, a “short-lived” map is created as the robot moves. The map is called “short-lived” because it only displays visual information in the last 2 images (previous and current) captured by the robot. Previously recorded visual information is lost when it falls out of the boundaries of the map as the robot moves. The main reason for this is that the robot in this project does not need to build a complete map of its environment in order to use it for future navigation. As mentioned earlier in section 1.1, only the procedure corresponding to the route description followed by the robot is saved in memory and can be called later to achieve the same navigational task. Here, therefore, the error due to image linking is present only once in the “short-lived” map and it is small enough to be safely neglected.

2.5 Discussion

The main observation from studying previous work on of natural language instructed robots is that previous attempts to create such robots are limited in the sense that their users are constrained to a pre-determined functional vocabulary. This functional vocabulary is decided by the creators of the robots without considering how the potential robot users would instruct the robot naturally. Thus, users are limited to a constrained form of natural language, which they must learn in order to instruct the robots. This approach of determining the functional vocabulary of the robot is called “robot-centred” approach.

In this project a purely “user-centred” approach is followed to determine the tasks that the robot is able to perform in the selected context. The idea is that the user should not require to learn how to use the robot and thus be restricted to using the robot’s functional vocabulary,

but he/she should be able to communicate with the robot in the same manner as with a human.

Chapter 3

3 Experimental environment setup and robot design

3.1 The components of the Instruction Based Learning system

For the purposes of the project an experimental environment was build that takes the space of a small room. The setup comprises mainly of a miniature town model, the robot and two host computers (Figure 3-1), which act as the “remote brain” for the robot.



Figure 3-1: The components of the IBL system.

The advantage of using a robot with remote brain architecture is that it does not require huge on-board computing and hence can be small, fitting the dimensions of the environment (see [Inaba et. al., 2000]). A controlled laboratory environment was chosen to apply the proposed methodology of the project instead of the real world environment. Apart from financial and time constraints the main reason for this was to deliberately exclude much of the complexity the real world presents and focus on the main idea of the project, which is to study the nature of spoken route instructions in order to determine how they can be converted into robot-executable programs.

The simple design of the miniature town model provided enough examples, in the collected corpus, to allow for reasonable conclusions that would apply to the more realistic robot environment. Users, during the corpus collection, reported that they felt comfortable with the setup and spoke as they would in a realistic environment. Route descriptions of course lacked the richness of landmark references or actions that would apply to the more complex real

world. Nevertheless the corpus collected was enough to enable the development of a methodology to determine the robot's primitive procedures using the user-centred approach.

The contribution of the author of this thesis to the experimental environment setup of the IBL project included the following:

1. The design and creation of the miniature model town environment.
2. The integration of a video camera and video transmitter on an existing robot-football robot in order to be used in this project.
3. The modification of the existing robot code in order to achieve better robot control.

In the following sections of this chapter each of the above contributions is described in detail.

3.2 The miniature town model

The layout of the miniature town model was designed using CorelDraw graphic design software. The model is flat and its dimensions are 170cm x 120cm (Figure 3-2).



Figure 3-2: The miniature town model.

The model town is a much simplified version of the real world for reasons explained earlier. Nevertheless, the road layout contains turnings at various angles, t-junctions, crossroads, dead ends, curves, y-splits, and a roundabout. The model also contains non road-layout landmarks like trees, a lake, a bridge, and buildings. Most of the buildings in the miniature town contain signs with names mostly taken from the real world (like Dixons, Derrys, the post-office etc). This was done to help people giving route instructions to the robot to easily identify these landmarks. For simplicity, the same colour was used for all the buildings in the miniature town.

3.3 The robot

The robot used in this project is a modified robot-football robot² (Figure 3-3). The main added components are a video camera and a video transmitter.

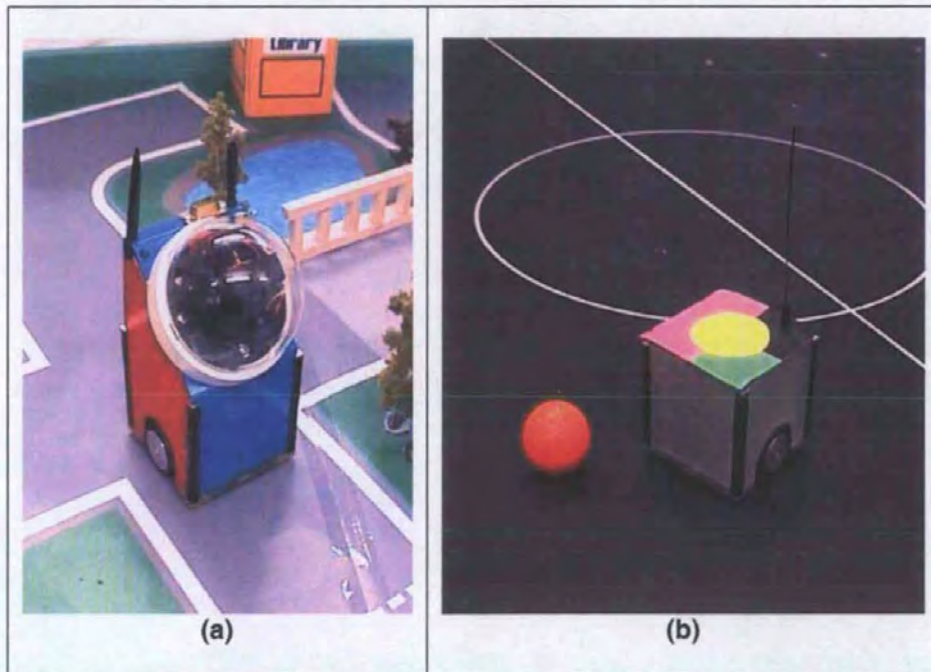


Figure 3-3: (a) The robot used in the IBL project (80x80x160mm) and (b) The robot-football robot (80x80x80mm).

² Provided by Merlin Systems (<http://www.merlinsystemscorp.com>).

3.3.1 Robot hardware

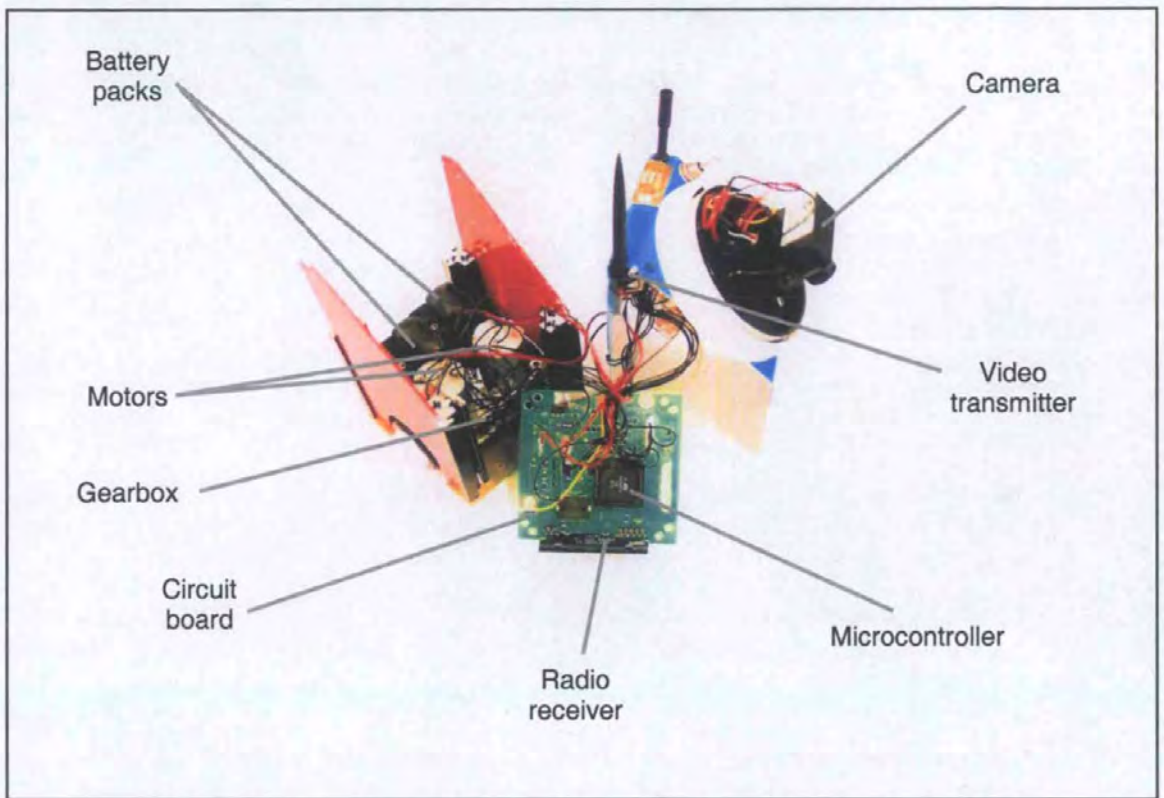


Figure 3-4: The hardware components of the robot.

Figure 3-4 shows the various hardware components of the IBL robot.

The robot has two independent motors driving each of its two wheels. The speed of the motors is geared down by a ratio of 16:1 using worm gears. Two optical encoder sensors are used to read the distance covered by each wheel of the robot. To achieve a greater distance resolution the encoders are placed so as to measure the angular speed of the motor shafts rather than that of the robot wheel axes. In effect, the distance resolution of each wheel is 1238 encoder pulses per metre traversed by the wheel.

A circuit board on which a microcontroller³ is a central processing unit controls all the functions on the robot. The main components on this circuit board apart from the microcontroller are a dual full-bridge driver⁴, which powers the two motors, and an 869 MHz radio receiver⁵, which receives navigation commands for the robot.

The robot is powered by eight Ni-MH (Nickel-Metal Hydride) rechargeable batteries of 700mAh (energy capacity) each. They are connected in series in order to create a cell of 9.6 Volts and 700mAh energy capacity.

Added to the robot-football robot, in order to use it in the IBL project, are a CCD colour TV camera⁶ (628 x 582 pixels) and a 2.4 GHz video transmitter⁷. A host computer acquires the images captured by the CCD camera onboard the robot through a wireless video link and via a TV capture card. An example of such image is shown in Figure 3-5.

³ Atmel AT90S8515

⁴ L298

⁵ Radiometrix RX3

⁶ Model MINI-C20A purchased at Allthings Sales and Services (<http://www.allthings.com.au>).

⁷ Model TX-MOD3 purchased at Allthings Sales and Services (<http://www.allthings.com.au>).



Figure 3-5: Example of the robot's view.

From these images the necessary information is extracted in order to decide the next waypoint for the robot. The robot is then sent the appropriate navigation command via the radio channel in order to reach this waypoint.

3.3.2 Robot software

The code that controls all functions of the robot is written in C computer language on a PC. It is then compiled, converted into machine language and finally downloaded into the 8K non-volatile flash memory of the microcontroller. The robot's microcontroller program works as follows: the host computer sends information to the robot in frames (or transmission strings) via radio link. Each frame contains (apart from synchronisation and error detection bits) the speed and distance to be covered by each of the two wheels. The robot controls the two wheels independently to cover the requested distance while moving with the requested speed. In this way the robot can perform any manoeuvre, per received command, where the speed of

each wheel is constant during the manoeuvre. Examples of such manoeuvres are motions on straight lines and arcs and on-the-spot rotations.

The robot program is interrupt driven. There are 4 types of interrupts that can occur:

1. Left wheel optical encoder sensor interrupt.
2. Right wheel optical encoder sensor interrupt.
3. UART (Universal Asynchronous Receiver/Transmitter) interrupt.
4. Interrupts from the two wheel speed controllers.

Whenever either of the two optical sensor interrupts occurs the variable that contains the distance covered by the corresponding wheel is incremented. Both variables are set to 0 when a new command is received by the robot. This is done because every command instructs the robot to perform a new manoeuvre starting from a position relative to the one it is when it receives the command. The robot does not record odometric information for more than one manoeuvre.

The UART interrupt occurs when a byte is received through the radio link. The interrupt service routine stores the byte in a string and checks if the string's length is equal to the expected command length. If not enough information is received to complete a command, the service routine does nothing, otherwise the received string is processed to extract speed and distance information for the two wheels. It then sets the appropriate microcontroller registers to cause the requested speed and resets the left and right distance variables to 0.

It must be noted here that the robot does not incorporate a radio transmitter in order to send data to the host computer. This means that there is no way for the robot to signal the end of an execution of a motion command to the host computer. To overcome this limitation, the host computer calculates the execution time of every command sent to the robot thus predicting when the robot will have finished executing it and will be ready to receive the following command. The execution time of a robot command can be calculated from the speed and distance values sent to the robot.

3.3.2.1 The PID controller used for robot speed control

The control diagram in Figure 3-6 shows how the speed of each wheel of the robot is controlled in some robot football robots.

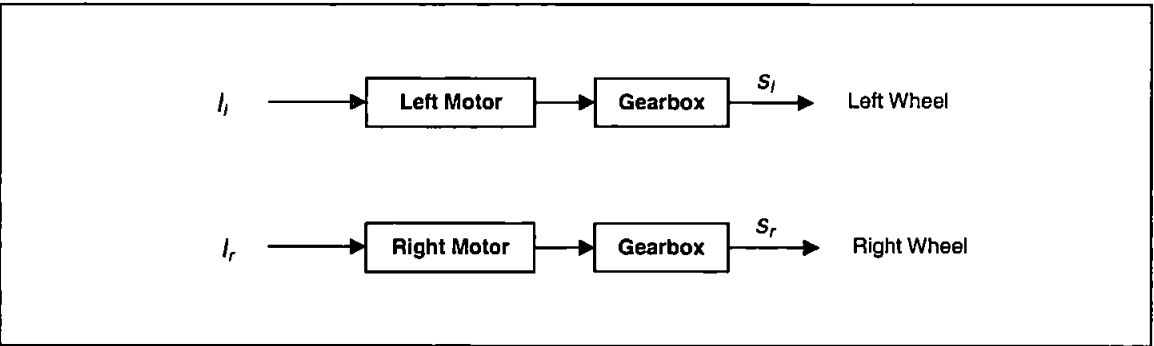


Figure 3-6: Open loop speed control of each of the robot's motors. Where I_l and I_r are the left and right inputs to the motors in volts and s_l and s_r are the left and right wheel speeds respectively.

This is an open loop control scheme for each individual wheel speed. A value corresponding to each motor voltage input is sent to the robot and it is applied to the respective motor until the requested distance is covered by the wheel attached to that motor through the gearbox.

There are two problems with the above method of control:

1. Factors such as the state of the robot's battery and frictional forces will affect the speed of each motor at a constant voltage input. This means that the speed of each motor depends on external factors apart from the level of input voltage.
2. Even if the two motors used for the robot are of the same model and manufacturer, small differences between them such as the resistance of their armature windings will result in a speed difference between the two motors when they are subjected to the same input voltage.

The effects of both problems above are minimal when the motors are driven at their rated voltage. At this input the robot is capable of accelerating up to speeds of 1.5 metres per second but this is not desirable because of heavy wheel slipping that gives false readings of distance covered by the robot. Furthermore, the duration of slipping of each wheel may be different depending on its grip with the ground. This often causes the robot to manoeuvre in the wrong direction. In robot football the motors are driven at their rated voltage. Any errors produced due to speed differences between the two wheels and slipping are quickly corrected because the robot is sent navigation commands several times per second.

For the purposes of the IBL project the robot is required to run at speeds not more than 10cm per second in order to avoid slipping. This low speed requirement means that the motors of the robot must be powered with less than 10% of their rated voltage. At this low input the load-speed characteristic of the motors is not linear and this causes unreliable

transient behaviour. For a successful manoeuvre of the robot, both wheels must complete their assigned distances but also they must maintain their assigned speeds reliably throughout the manoeuvre. This is not possible with the control system shown in Figure 3-6 at such low power input to each motor.

A popular solution, in order to overcome the problems of open loop systems such as the one shown in Figure 3-6, is to use a PID (Proportional, Integral and Differential) controller algorithm. The algorithm takes as input an error e , which is the difference between the desired output value and the actual output value of a system and produces the input to the system. The objective of the controller is to match the output of the system with the desired output. The PID algorithm is composed of three terms:

1. A term that is proportional to the error e .
2. A term that is proportional to the integral (or the sum of the previous values) of the error.
3. A term that is proportional to the rate of change of the error.

Each of the three terms plays a different role in the controller. For example the differential term (number 3 above) suppresses sudden departures of the output from its desired value. It is not necessary to have all three terms present in the controller. A controller can only have the proportional and integral terms for example in which case it is called PI controller. For a detailed explanation of the PID controller as well as implementation guidelines for real-time systems see [Bennet, 1994].

Two PI controllers are used to accurately control the speed of each wheel of the robot eliminating errors due to the robot's battery charge level and friction. The error inputs to each controller (e_l and e_r) are the differences between the requested (or desired) speeds (s_l and s_r) and the actual speeds (s_l and s_r) of the left and right wheels respectively. The outputs of the controllers (I_l and I_r) are the corresponding voltage inputs to the left and right motors. Figure 3-7 shows the closed loop speed control diagram of each motor.

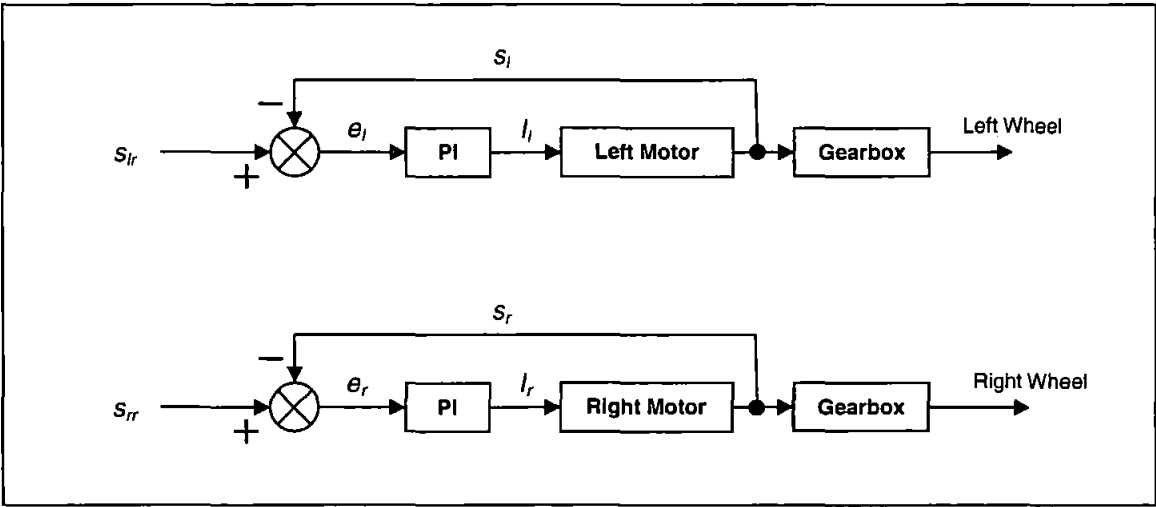


Figure 3-7: Closed loop speed control system with a PI controller for each motor.

The input to each motor is calculated by its corresponding PI controller whenever a wheel speed controller interrupt occurs. Notice that the speed of each wheel at the time of the interrupt is required for the calculation and not the distance covered by the wheel, which is physically measured at the motor shaft of the wheel (see section 3.3.1). The speed of each wheel is determined by dividing the distance travelled since the last interrupt by the interrupt's interval time. The interval of this interrupt is time critical and therefore one of the microcontroller's timers is used to produce it.

For every robot manoeuvre there is a difference between the left and right wheel speed. This difference is 0 when the robot is following a straight line and greater or less than 0 when the robot is performing a curve. It is crucial that this difference is maintained throughout the manoeuvre if the robot is to reach its target location successfully. To guarantee this, a PID controller is added to the system shown in Figure 3-7. The complete speed controller of the robot is shown in Figure 3-8.

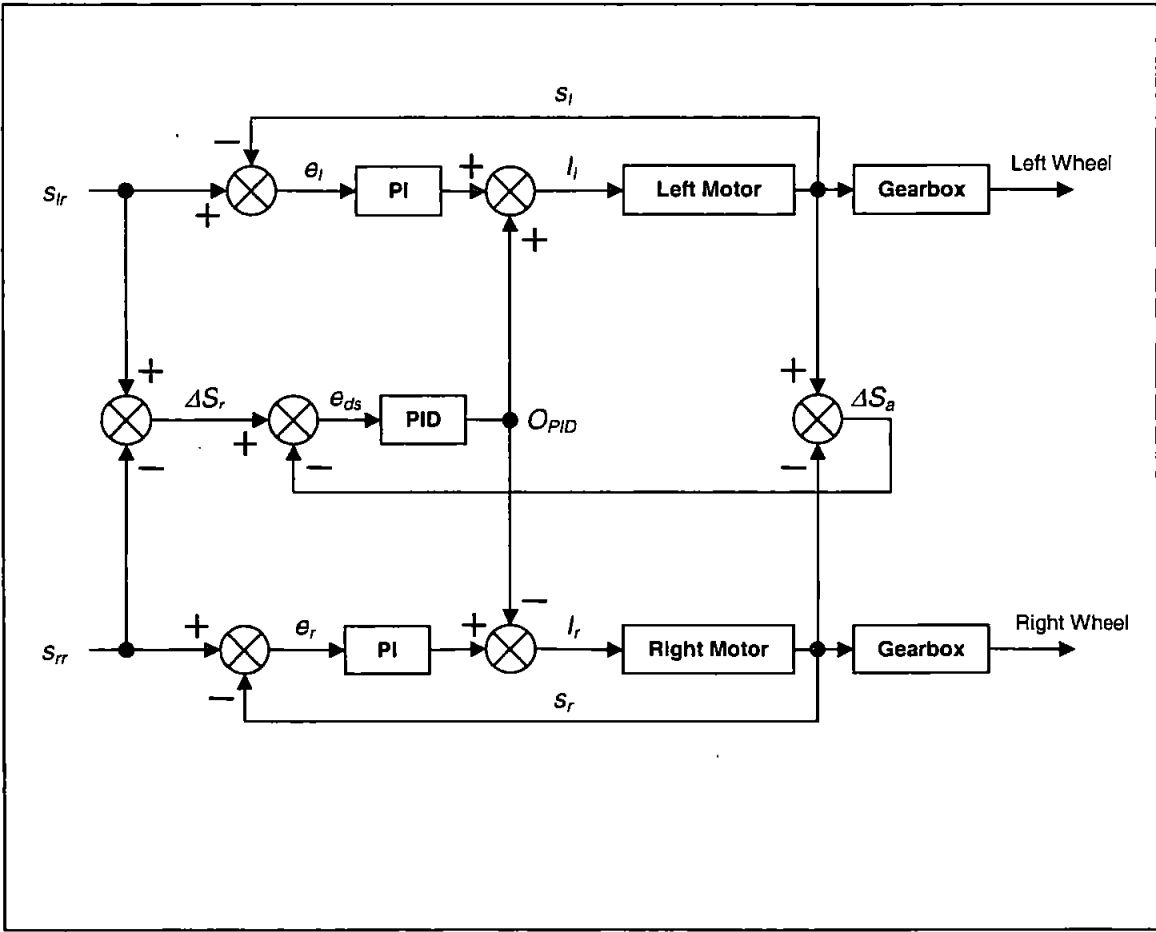


Figure 3-8: The complete robot speed control system.

As Figure 3-8 shows, the error input (e_{ds}) to the PID controller is the difference between the requested speed difference ΔS_r , and the actual speed difference ΔS_a between the two wheels.

The output of the PID controller contributes to the PI outputs to produce the motor inputs (I_l and I_r) in such a way so as to maintain the requested speed difference throughout the robot's manoeuvre.

To determine the parameters of the PI and PID controllers of the system, first the sampling interval T_s of the wheel speed controllers interrupt was selected. There are two points to consider when selecting a discrete-time controller's sampling interval:

1. It should not be too small. This is because digital computations have finite resolution limited to the length of the digital system's floating point number. As the sampling interval decreases, the change in the result of the controller's output between successive samples becomes less than the resolution of the system and thus information is lost (see [Leigh, 1992]).
2. It should not be too big because loss of information will occur due to the sampling effect (see [Bennet, 1994]). Nyquist's sampling theorem states that the sampling interval should at least be twice as fast as the highest frequency of the fastest changing signal in the PID controller's calculation. However, in practice a much higher sampling rate (more than 10 times the maximum frequency) is used.

Following the empirical rules suggested in [Bennet, 1994] the sampling interval for all three controllers was chosen to be 0.065536 seconds. This period is a multiple of the microcontroller's clock period and it is chosen among several possible preset periods given by the microcontroller's manufacturer.

The following set of equations show how the inputs to the left and right motors are calculated each time a wheel-speed controller interrupt occurs:

$$\begin{aligned}
 I_l &= K_p \left[e_l(n) + \frac{1}{T_i} \sum_{k=1}^n e_l(k) T_s \right] + O_{PID} \\
 I_r &= K_p \left[e_r(n) + \frac{1}{T_i} \sum_{k=1}^n e_r(k) T_s \right] - O_{PID} \\
 O_{PID} &= K_p' \left[e_d(n) + \frac{1}{T_i'} \sum_{k=1}^n e_d(k) T_s + T_d' \left(\frac{e_d(n) - e_d(n-1)}{T_s} \right) \right] \\
 e_d &= \Delta S_r - \Delta S_l
 \end{aligned} \tag{3-1}$$

where T_s is the sampling interval of the control system in seconds. The variables $e_l(n)$ and $e_r(n)$ represent the values of left and right speed errors respectively at some time interval nT_s , where n is an integer. The variable $e_d(n)$ is the error in the differential speed at the same time interval. Constants K_p and T_i are the proportional and integral constants of the PI controllers and K_p' , T_i' and T_d' are the proportional, integral and differential constants of the PID controller. All speed values are in wheel-speed-encoder pulses per second.

Note that the input ranges from 0 to 255 and corresponds to a voltage range from 0 to V_{cell} Volts where V_{cell} is the robot's battery terminal voltage.

The PI and PID controller constants were found experimentally using a procedure similar to the one suggested in [Braunl, 2003]. This procedure, adapted for the control system presented in Figure 3-8, is as follows:

1. The desired operating speed of the robot was selected to be 10 cm/sec.
2. To determine the PI controller parameters only one of the two wheels was operated.
The integral control of the PI controller of the wheel and the PID controller were turned off. K_p was increased until oscillation in the speed of the wheel occurred.
3. K_p was divided by 2.
4. T_i was decreased from a large number until oscillation occurred.
5. T_i was multiplied by 2.
6. Both wheels were set to operate with PI control and the parameters derived thus far.
7. K_p' was increased until oscillation in the differential speed of the wheels occurred, i.e. the robot oscillating between the left and right directions while moving forward.
8. K_p' was divided by 2.
9. K_d' was increased while observing the behaviour of the differential speed while changing the operating speed by approximately 5%. A value of K_p' was chosen to give a damped response.
10. T_i' was decreased from a large number until oscillation occurred.
11. T_i' was multiplied by 2.

The following values were obtained with the above procedure and by doing minor adjustments to achieve optimum performance:

$$K_p = 0.15$$

$$T_i = 0.6$$

$$K_p' = 0.05$$

$$T_i' = 0.6$$

$$T_d' = 0.5$$

3.4 Programming platforms and developed software

The Linux⁸ operating system was used to develop the primitive procedures. The primitive procedures are written in the Python⁹ programming language augmented by vision routines written in C. Python is an interpreted language that is well suited for this project since it is desired to create new program code from verbal instructions. Python generates programs in the form of scripts that can be executed immediately without an intermediate compilation step.

Although the Python Imaging Library (PIL) could be used to implement all image processing routines described in this thesis, it is very slow and for this reason all vision routines were written in C and compiled in order to be used as python functions. Details of how to extend the Python language with C language functions can be found in [Chun, 2001].

Primitive procedures cannot pass information between each other directly. This is because they are independent programs called in sequences. It is only when one primitive procedure finishes execution that another can be started. For this reason all data (including image files) is passed between the procedures by saving it to files. This does not slow the system because any data saved is quickly accessed while it is still in the PC's cache memory and so no time is lost

⁸ "Redhat" variant, Version 7.3

⁹ Python for Linux, Version 1.5.2 (<http://www.python.org>)

A video of a test-run, which shows how the development interface is run, is included in the CD accompanying this thesis (see Appendix C).

3.5 Summary/Contributions

The main contribution of the work presented in this chapter is the development of a complex software control system that is used to control and coordinate the wheel speeds of the robot in this project. The control system, comprising of two PI controllers to control each wheel's speed and a PID controller to control the differential speed, presented a particular challenge in this project because of the extremely unreliable behaviour of the motors driving each wheel when these were run at a small fraction of their rated voltage. The control system implemented provided an alternative to using expensive high-specification motors incorporating gearboxes in order to produce the same speed reliability and odometric accuracy required for the purposes of this project.

Similar (compared to the one presented in this chapter) control systems for differential-drive of non-holonomic robots are presented in [Jones et. al., 1999] and more recently in [Braunl, 2003]. In their work a proportional (P) controller was used to control the speed of each individual wheel and integral (I) controller was used to coordinate the differential speed of the two wheels.

Also in this chapter a method was described for monitoring the execution of primitive procedures without the need to incorporate extra code in them in order to achieve this

purpose. This is done by “tapping” into (or monitoring) the data exchanged between the primitive procedures while they are executed. An interesting feature of this method is that by switching off all the data monitoring applications, the execution-speed performance of the primitive procedures can be quickly established.

Chapter 4

4 Corpus based system design

As discussed in section 2.1, previous attempts to implement a natural language interface to robots mainly used a robot-centred approach for determining the functional lexicon of the robot. The functional lexicon of the robot was created by predicting what a user in the context would to ask the robot and in what way. Users, therefore, had to be constrained to a certain extend when speaking, taking care to conform to the robot's particular syntax and to include all the necessary parameters of the action they requested.

In this project, an effort was made not to constrain the user so that any user not previously trained to speak to the robot would be able to do so. In order to follow such a user-centred approach it was necessary to investigate how users speak when giving route instructions, what information they provide and what they omit as commonsense. The robot should be able to use the information given in the route descriptions and determine the information omitted in them, without discomforting its user with questions if it is to qualify as a useful human assistant.

To determine the functional vocabulary of the route description context, a corpus of route descriptions was collected from 24 subjects. Details of the corpus collection procedure are

given in section 4.1. The collected route descriptions were recorded and later transcribed for analysis. The corpus was analysed for its word and functional content. The method and results of these analyses are presented in sections 4.2 and 4.3.

The corpus of route descriptions collected in this project was split in two sets in order to enable development and later the evaluation of the primitive procedures (see chapter 7). For completeness, sections 4.2 and 4.3 present the results of the word and functional analyses of the complete corpus.

Section 4.6 explains the various cases in the corpus where natural language route instructions are missing information that is vital for the robot in order to execute the requested task. The methods proposed in this thesis for determining the missing information are also described.

The collected corpus of route instructions contributed in different ways to the main parts of the Instruction Based Learning system. Sections 4.7.1 and 4.7.2 explain how it contributed to the development of the natural language system design (dialogue manager) and the robot system design (robot manager) respectively.

4.1 Corpus collection procedure

To collect linguistic and functional data specific to route learning, 24 subjects were recorded as they gave route instructions to the robot in the miniature town environment. Subjects were divided into 3 groups of 8. The first two groups (A and B) were told that the robot was remote-controlled and that, at a later date, a human operator would use their instructions to

drive the robot to its destination. Subjects were told this so that they would speak as naturally as they would if they instructed a human. It was also specified that the human robot-operator would be located in another room, seeing only the image from the wireless on-board video camera. This was specified to induce the subjects into using spatial references accessible by the vision software. Subjects were also told to use previously defined routes whenever possible, instead of re-explaining them in detail. Each subject had 6 routes to describe among which 3 were “short” and 3 were “long”. Each long route included a short route. This was done to reveal the type of expressions used by the subjects in order to link taught procedures with primitive ones. Groups A and B received the same routes to describe, but with the sequence of “short” and “long” route inverted. This would reveal the difference between a fully detailed route, and a route with reference to a short route inserted. Again the question is one of how procedure insertion is handled by subjects (see Table 4-1 for examples of short and long route descriptions).

The first two groups (A and B) used totally unconstrained speech, to provide a performance baseline. It is assumed that a robot that can understand these instructions as well as a human operator would represent the ideal standard. Each subject described 6 routes having the same starting point and six different destinations. Starting points were changed after every two subjects. A total of 96 route descriptions were collected from these two groups.

A third group of 8 subjects (group C) had the same routes to describe as group A, but were forced into a simplified dialogue with an operator to produce shorter chunks of descriptions. It is known that it is very difficult for NL processing tools to correctly segment an uninterrupted stream of words into sentences. Therefore, corpus group C was thought to be

more representative of utterances in the eventual user-robot dialogue. Subjects in this group were told that an operator next door was taking notes. A researcher pretended to do so and interrupted the subjects (using a microphone) when they used chunks that were too long. He acted as if he understood all the instructions and did not initiate repair dialogues.

Table 4-1 shows an example of the same two “short” and “long” routes instructed by a subject in group A and a subject in group C.

Group A (Monologues)		
Short (u11_GA_EP)	User:	"okay take your first right and continue down the street past Derry's past Safeway and your parking lot the car park will be on your right"
Long (u11_GA_EH)	User:	"okay once you pass the car park er take your first right and then again take your first right and the hospital will be right in front of you"
Group C (Dialogues)		
Short (u4_GC_EP)	Wizard:	"could you tell me how to get to the car park please"
	User:	"okay you'll take the first right from where you are now past Derry's then Safeway"
	Wizard:	"Yes"
	User:	"you'll pass another road on the left and the car park's on the right from there"
	Wizard:	"thank you"
Long (u4_GC_EH)	Wizard:	"could you tell me how to go to the hospital please"
	User:	"okay you need to go back towards the car park"
	Wizard:	"Yes"
	User:	"past the car park take the first right"
	Wizard:	"I'm sorry after I pass the car park"
	User:	"you take the right after the car park"
	Wizard:	"Yes"
	User:	"and then another right again"
	Wizard:	"Yes"
	User:	"and you'll be moving towards the hospital on the end of that road"
	Wizard:	"thank you"

Table 4-1: Examples of “short” and “long” route descriptions.

The table shows a short route from the Grand Hotel (E) to the Car Park (P) and a long route from Grand Hotel (E) to the Hospital (H) (see Figure 4-1) given under monologue condition (group A) and dialogue conditions (group C). The wizard is a human operator mimicking verbal feedback that could be given by the robot.



Figure 4-1: A top view of the miniature town model indicating the starting point E (common to both routes) and destinations P and H referred to in Table 4-1.

The sound file and transcribed version of each route description collected in this project can be found on the CD accompanying this thesis (see Appendix C).

4.2 Word analysis

To provide an initial estimate of the task vocabulary, the data from all three groups were merged. The number of distinct words was counted in the set of 144 instructions collected from the three groups. Morphology was not taken into account, i.e. “travels” and “travel” were counted as different words. The vocabulary of the users was found to contain 336 different words, from a total of 6634 words in the combined corpus. The most frequent word was found 753 times and 96 words were used only once, i.e. by only one subject in a single route instruction (Table 4-2).

Most frequent		Least frequent (found only once)
Word	Count	
the	753	access, actually, already, amount, angle, any, apologise, area, arrive, bears, been, beige, bends, black, blocks, both, branch, carrying, centre, certainly, currently, diagonally, doors, double, en, ends, entering, exits, far, feel, five, fork, forty-five, half, has, here, hope, house, instruct, it'd, its, leave, leaving, lines, looks, make, means, moment, more, moving, now, only, or, order, outside, paper, park's, passing, please, post-offices, quadrangle, quarters, queens, reaching, recalling, robot, say, says, set, seventy, sharp, sixty, skyscraper, soon, starts, still, storey, taken, tesco, tesco's, thankyou, thing, think, thirty, too, travel, travels, trip, turned, uh-huh, upon, went, what, while, wiggles, without
and	263	
on	253	
you	233	
to	212	
left	188	
right	178	
go	168	
take	137	
a	132	

Table 4-2: Most frequent and least frequent user word in the corpus. The least frequent words were found only once in 96 route descriptions.

A complete list of the words found in the corpus along with their occurrence is presented in Appendix D.

To determine if the corpus collection had led to a complete sampling of the task vocabulary, the average number of distinct words was plotted as a function of the number of collected instructions. This is shown in Figure 4-2.

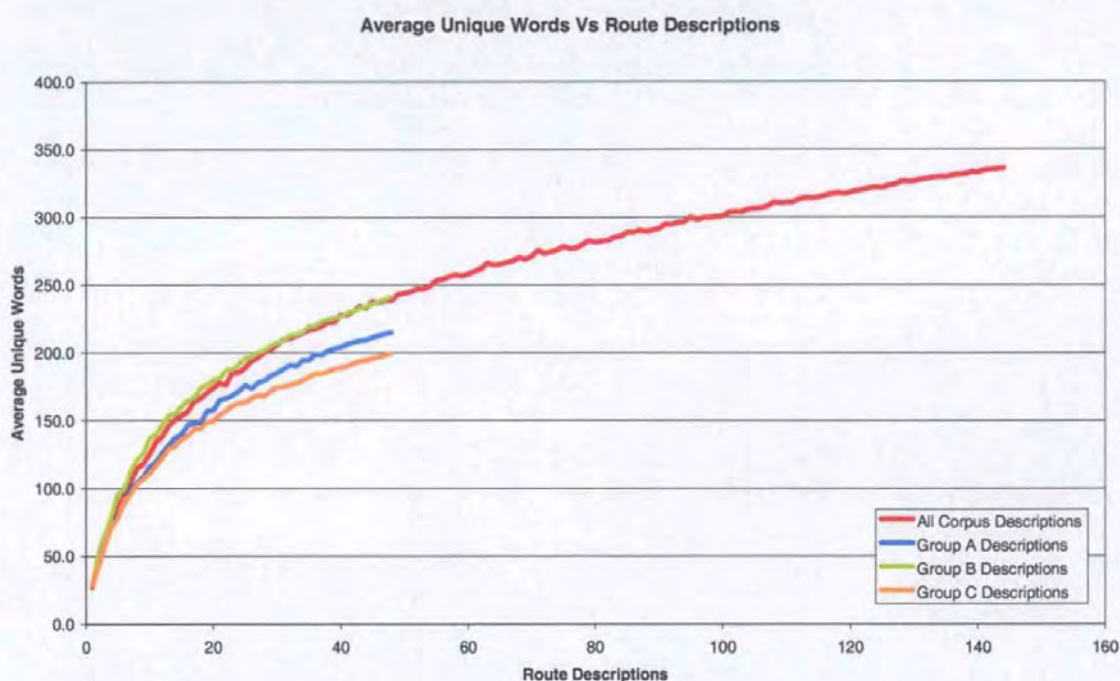


Figure 4-2: Number of distinct words discovered in the corpus as the number of instruction samples increases. The long line is for all groups considered. The shorter lines are for groups A, B and C taken in isolation. Curves are obtained by averaging 50 random sets comprising an increasing number of sample route descriptions.

Figure 4-2 shows that the number of distinct words is still rising at the end of the curve, indicating that more new words would be found if more route instructions had been collected. This behaviour is similar in other spoken language task domains. Some examples are found in [Zue, 1997] where corpora from different domains are collected and analysed for their lexical content. This was done as part of discussing the issues involved in human-computer spoken language interfaces. In the same paper it is mentioned that new words will be encountered by the speech recognition system no matter how large the training corpus is.

The slope of the curve representing all corpus descriptions in Figure 4-2 indicates that a new user might say on average one out-of-vocabulary word in each route description. To

determine what type of new word might be expected, each route instruction was compared to the corpus of all other instructions. The result is that the new words are all among the 96 least frequently used words listed in Table 4-2.

The dialogue group (group C) tended to use less distinct words as shown in Figure 4-2.

Therefore, future experiments may reveal an improved speech recognition performance in dialogue conditions.

New words (i.e. words spoken by the user, which are unknown to the speech recognition system) may present a problem in that they will be either recognised wrongly or not recognised at all and thus present the danger of changing the meaning of what the user said. There are three possible scenarios when a user utters a new word:

1. The word is a key word in the specification of the route description. In this case the robot will fail without being able to detect the problem.
2. The word is not a key word. In this case its presence does not alter the specification of the route description and the robot never “sees” its effect in the route description.
3. The word is misrecognised in such a way so that the meaning of the instruction appears wrong to the speech system. In this case the speech system can initiate a repair dialogue with the user in order to clarify what has been said or to bias the user to explain in a different way.

Starting such a repair dialogue with the user can be a very complex process. At present a simple dialogue is initiated with the user when the recognition confidence of the speech

system falls below a certain limit. This dialogue usually involves the system replying either with: “Repeat that please.”, in which case the user must repeat the instruction, or with “Did you mean ...”, in which case the user can only answer “yes” or “no”.

4.3 The primitive procedures in route instructions

In order to find the primitive procedures the robot should have in its memory when it starts its life, the corpus of route instructions collected was first segmented into its “functional components”. These functional components were then represented by primitive procedures written in computer language code.

The methodology followed to segment the route descriptions into their functional components was based on the definition of the functional component. Two rules were followed:

1. Functional components should describe a single action and
2. They must have a defined initial and final state.

The first rule makes sure that the most elementary actions that constitute a route description are considered to be its functional components. The second rule comes from implementation constraints. In the example utterance:

“follow the road until you reach the post-office”

Although “follow the road” can be considered as one action there is no information in it to suggest when to stop following the road. In this case the whole example utterance is considered as a functional component in order to encompass the piece of information that gives the final state (i.e. reaching the post-office).

Section 4.3.1 describes how the functional components extracted from the corpus are represented by robot procedures called “primitive procedures”. Primitive procedures are computer language procedures that control the robot. A primitive is called for every functional component found in the route description and this causes the robot to execute the action(s) specified by the functional component. Primitive procedures accept key words from the functional components as parameters.

4.3.1 The primitive procedures extracted from the corpus and their representation

Functional components found in the corpus are organized into groups describing similar actions. For example the primitive procedures: “take the first left turn” and “take the second left turn” have little difference in their implementation in robot executable code. Similarly with “follow the road to the post-office” and “follow the road to the library”. To avoid duplication of code, parameterised primitive procedures were written to represent groups of functional components found in the corpus rather than the individual components themselves. Different combinations of parameters are initialised in each primitive procedure call to represent each functional primitive found in the corpus.

The complete list of primitive procedures extracted from the collected corpus of route instructions is presented in Table 4-3. The rightmost column indicates the number of occurrences of the associated functional group in the corpus.

	Primitive Procedure	Occurrence
1	follow_road (relation_1, ordinal_1, object_1, relation_2, object_2)	234
2	turn (ordinal_1, relation_1, object_1, relation_2, object_2)	192
3	location (object_1, relation_1, ordinal_1, object_2='road', object_3, destination_1)	161
4	exit_roundabout (ordinal_1, relation_1, object_1)	36
5	go (relation_1, object_1)	30
6	go_until (object_1, relation_1, object_2)	9
7	enter_roundabout (direction_1, relation_1, object_1)	8
8	cross (object_1, relation_1, object_2)	3
9	rotate (relation_1, object_1)	2
10	take_road (relation_1, object_1)	2
11	exit_object (object_1)	1
12	park (relation_1, object_1)	1
13	bear (relation_1, object_1)	1

Table 4-3: Primitive procedures extracted from the collected corpus of route descriptions.

An explanation of each primitive is given in Appendix A along with detailed specifications of its parameters and the values they can take.

Table 4-4 shows some examples of functional components taken from the corpus and the corresponding primitive procedure call that should be executed to produce the requested action.

Functional primitive	Primitive procedure call
"take the first left"	turn (relation_1='left_of', object_1='self')
"take the second turning on your left hand side"	turn (ordinal_1='second', relation_1='left_of', object_1='self')
"keep walking past the lake on your right hand side"	follow_road (relation_1='past', object_1='lake', relation_2='right_of', object_2='self')
"exit the roundabout at the third exit"	exit_roundabout (ordinal_1='third')
"the library is on your left"	location (object_1='library', relation_1='left_of', object_2='self', destination_1='library')

Table 4-4: Examples of primitive procedures extracted from the corpus and their corresponding primitive procedure calls.

See Table 4-6 for an explanation of parameter types and the values they can take in a primitive procedure.

In the same manner all the corpus route descriptions were manually "translated" into their corresponding primitive procedure calls.

Each transcribed route description file was associated to a file with primitive procedure calls corresponding to the functional components in the route description. An example of such translation is given in Table 4-5.

(a)	"from the roundabout take the first exit on the left continue straight over the crossroads continue over the bridge continue straight over the second crossroads the post office should be on your right" (u7_GC_CX)
(b)	<pre> exit_roundabout(ordinal_1='first') follow_road(relation_1='over', object_1='crossroads') follow_road(relation_1='over', object_1='bridge') follow_road(relation_1='over', ordinal_1='second', object_1='crossroads') location(object_1='post_office', relation_1='right_of', object_2='self', destination_1='post_office') </pre>

Table 4-5: An example of a translation of a route description to its corresponding primitive calls. Row (a) shows the transcribed version of the route description u7_GC_CX. User 7 explains the route from Boots (C) to the Post-office (X) (see Figure 4-3). Row (b) shows the corresponding manual translation of the description to its primitive procedure calls.

All translation files of the corpus can be found on the CD accompanying this thesis (see Appendix C).



Figure 4-3: The description in Table 4-5(a) illustrated on the map of the miniature town. The dotted red line shows the route that the user implies to the robot and the solid red line is the route he/she explicitly describes.

This translation had to be made manually during the development of the IBL system for two reasons:

1. To enable the development and evaluation of the primitive procedures (see chapter 7).
2. To provide a performance baseline for the evaluation of the Robot Manager.

Although the translation of the corpus was done manually for all functional components in each route description, it was tried to the best possible extent to produce the translation that the final system would produce for the same route descriptions. This was achieved by continuously changing the translation of the corpus throughout the duration project based on feedback being received during the development of the dialogue manager and robot manager modules. Continuous reviewing of the translation of the corpus was being done until prior to

the evaluation of the system. The complete final translation of each route in the corpus can be found on the CD accompanying this thesis (see Appendix C).

As with the task vocabulary, to determine if the corpus collection had led to a complete sampling of the primitive tasks in our route description context, the average number of primitive procedures was plotted as a function of the number of collected route descriptions. Figure 4-4 shows that the number of distinct procedures is increasing with the number of sampled route descriptions. In the beginning there is a steep rate of increase of new primitive procedures but as more route instructions are considered this rate decreases. It can be speculated, by looking at the curve representing all the corpus descriptions in Figure 4-4, that the functional vocabulary of the robot is not completely determined by the collected corpus. The slope of the curve at 144 route descriptions seems to suggest that on average one new procedure is likely to be discovered in every approximately 40 route descriptions. A similar observation was made in section 4.2 with the rate of increase of new words for the robot (see Figure 4-2). There, it was discovered that on average one new word would be discovered for each route description. The issue of new primitive procedures appearing during the lifetime of the robot is very crucial to the design of instruction based robots. This issue is discussed further in the conclusion of this thesis (see section 8.2).

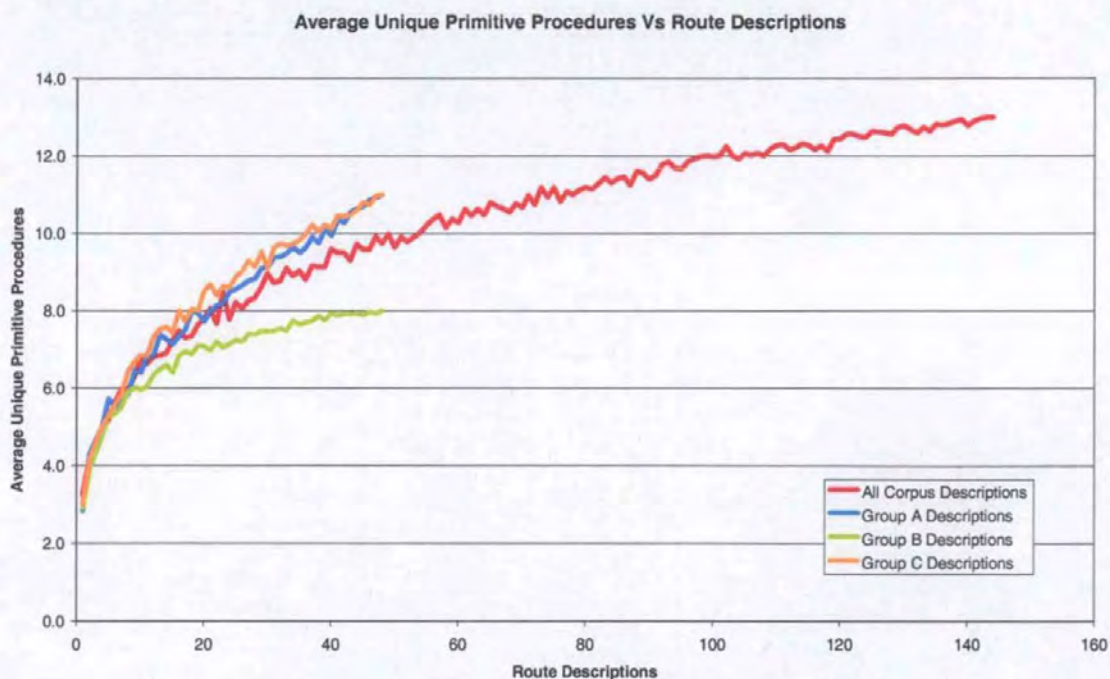


Figure 4-4: Number of distinct primitive procedures discovered in the corpus as the number of instruction samples increases. The long line is for all groups considered. The shorter lines are for groups A, B and C taken in isolation. Curves are obtained by averaging 50 random sets comprising an increasing number of sample instructions.

Primitive procedures accept several types of parameters. These are listed and explained in Table 4-6 along with examples of values they can take in primitive procedure calls.

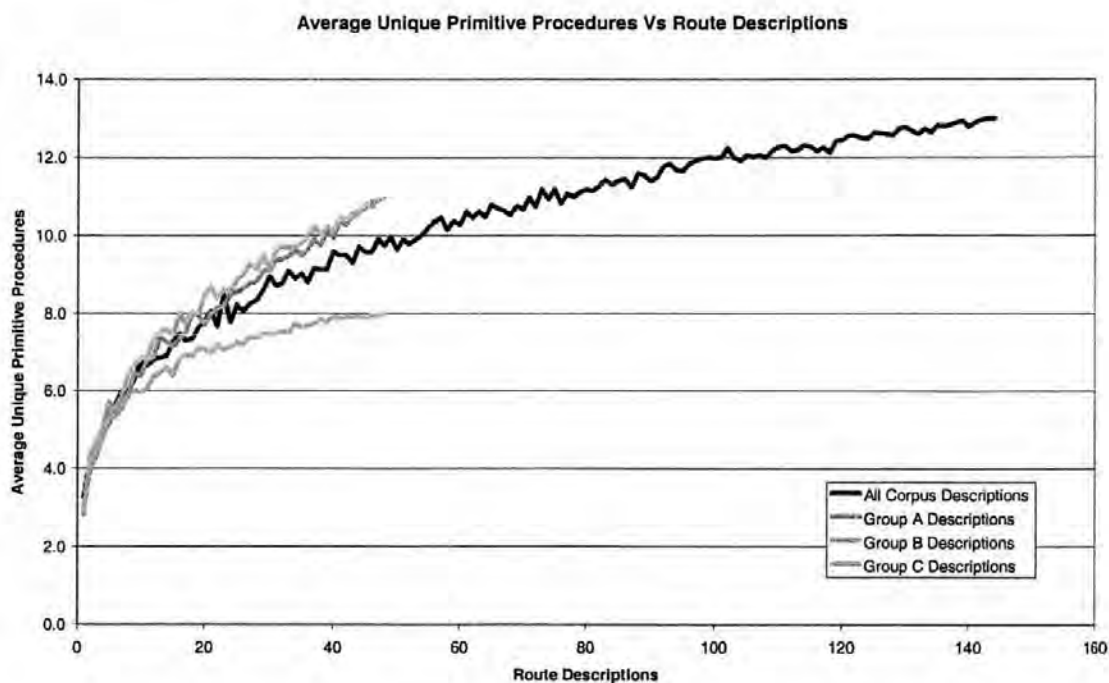


Figure 4-4: Number of distinct primitive procedures discovered in the corpus as the number of instruction samples increases. The long line is for all groups considered. The shorter lines are for groups A, B and C taken in isolation. Curves are obtained by averaging 50 random sets comprising an increasing number of sample instructions.

Primitive procedures accept several types of parameters. These are listed and explained in Table 4-6 along with examples of values they can take in primitive procedure calls.

Parameter name	Description	Examples of possible values
object	Takes the names of objects used as landmarks in route descriptions. This parameter can also take the value 'self' in cases when users indirectly refer to the robot.	'self', 'hospital', 'house', 'junction', 'lake', 'left_turning', 'bend', 'bridge', 'building', 'car_park', 'corner', 'derrys'
relation	It describes the relation of a mentioned object to another object or the robot. Always accompanied by one or more 'object' parameters.	'after', 'at', 'between', 'by', 'in_front_of', 'left_of'
ordinal	Specifies the order of one of many similar objects usually along the same road.	'first', 'second', 'third'
destination	Special case which is used because of system constraints (see section 4.4). Used only in the "location" primitive to specify whether a mentioned object is the destination of the route description.	'boots', 'car_park', 'grand_hotel', 'hospital', 'library', 'museum'

Table 4-6: Parameter types and possible values they can take in primitive procedure calls.

Parameter names end with a number in the primitive procedure headers to avoid confusing two or more parameters with the same name. The specific significance of each individual parameter for every primitive procedure is explained in Appendix A.

It is important to note that, in order to follow a purely “user-centred” approach in determining the functional vocabulary of the IBL system, the 13 primitive procedures, the parameters they can accept and all the possible values each parameter can take are derived solely from the corpus of route instructions collected for this project.

4.4 The “go” primitive

The “go” primitive is called when users refer to complete previously learned routes. Examples are:

“go to the post office” (u12_GA_EG)

“go to the roundabout” (u2_GC_MC)

“go to the roundabout mentioned previously” (u13_GA_CE)

“go to boots” (u19_GB_EC)

When the system learns a new route, it saves all the procedures of the new route in a script file called “go_<point A>__<point B>.py” where point A is the name of the starting landmark and point B is the name of the destination landmark. When the system encounters an instruction such as “go to the post office” in the beginning of a route description, say, from the library to the museum, it first searches the knowledge base to find a file called “go_library__post_office.py”. If the file is found, information from it is used in the new file being created (“go_library__museum.py”). How and which pieces of information from previous knowledge is extracted and used is described later in sections 5.3 and 5.4. If, in the example given, “go_library__post_office.py” is not found in the knowledge base then a new learning process is started to create it and then use it.

4.5 The “go_until” primitive

The primitive “go_until” refers to a previously learned route but only up to a certain point in that route. It is used either because the user intends to divert the robot onto another route or

because the destination is simply along the previous route but before that route's destination.

Examples in the corpus where such reference is made are:

“okay you'll need to pass the train station again as you did going to the post office and you'll see the university as you go onto the roundabout” (u4_GC_EW)

“erm head as though you're going towards the post office so you go over the bridge but instead of carrying straight on take a right” (u6_GC_CM)

“okay from the crossroads continue on straight ahead take the next right”
(u7_GC_CM)

“okay head towards the grand hotel but just before you get there the safeway is on your right hand side” (u10_GA_MD)

“recalling our previous destination was the grand hotel and we passed safeways en route just before derry's” (u23_GB_HD)

“right if you go exactly the same way towards the queens pub as before erm as you go over the bridge as you go past the t junction the post office will be there on your right” (u6_GC_CX)

This is a more complicated case than the “go” primitive because now the previously learned route must be partly used up to a landmark specified by the user. To add to the problem, this

landmark is not always apparent from the instructions in the previously learned file (e.g. “from the crossroads...”). This is either because when the file was created, information about the landmark was not inserted by the system, because it was not deemed relevant to the route, or because the landmark was not mentioned by the user at all.

Two methods for solving this problem were discussed during this project but were not implemented due to time constraints. These are presented as part of future work in Chapter 8.

4.6 The under-specification of natural language and how it affects the functional specification of the primitive procedures

Spoken route instructions can be very abstract often lacking information that is assumed by the instructor as commonsense. However, the missing information can sometimes be vital to the success of the robot in executing a route description. In most cases the human listener automatically infers the missing information. Alternatively he/she can engage in a dialogue with the speaker in order to request a more explicit version of the instruction. For the Instruction Based Learning system, starting such a clarification dialogue with the user can be a very complex process.

It is important therefore, for the system, to try to infer, to the extent possible, any implicit information in the user’s instructions. Since the system is lacking the cognitive power and experience of the human listener the only way to determine how to resolve such cases, is to

study the corpus of route instructions collected during this project for those cases where users omit the same information and expect the same action from the robot.

This section shows how the under-specification of natural language route instructions influenced the functional implementation of the primitive procedures. The sub-sections below present the three cases where missing information in route instructions, that would otherwise cause the robot to fail, is inferred by the system.

4.6.1 The use of default parameter values

Some parameters of primitive procedures can take default values when a call to these procedures does not initialize these parameters. An example of such an occurrence is when a user says: “take a left” actually meaning “take the first left turn”. The action to “take” is first mapped here to the “turn” primitive by the robot manager (see [Lauria et. al., 2002]) and then the parameters “relation_1” and “object_1” are initialised to “left_of” and “self” to reflect the information of direction passed from the user. However, the specifications of the “turn” primitive procedure require at least the “ordinal_1” parameter to be initialised too. This parameter is given a default value “first” because in all cases of the corpus when the turn instruction was used without specifying the ordinal of the turning, the first turning was implied. The default values of parameters used in the primitive procedures are listed in each primitive’s specification in Appendix A.

4.6.2 The reference to the destination landmark

When humans describe a route they continuously refer to landmarks. It has been observed by studying the collected corpus of route instructions that they always refer to the destination when that is reached in the description. This reference can be explicit or implicit. Examples of explicit references to the destination are:

“safeways is the next building on your right hand side” (u1_GA_MD)

“and on the right hand side opposite the lake is the car park” (u5_GC_EP)

“and the museum will be on your right” (u13_GA_CM)

However, references to the destination have no particular difference when compared with references to other landmarks. Some examples are:

“you got pc world on your right” (u20_GB_EC)

“you ve got a car park on your right” (u20_GB_EG)

“walk up few metres and then you see the huge tall building on your left”

(u22_GB_CL)

“there is a lake on the left hand side” (u1_GA_MY)

The robot’s actions are different when the destination is mentioned than when any other landmark is mentioned and therefore a different section of code should be executed for each case. In the first attempts to functionally analyse the corpus in order to determine the primitive procedures, two distinct primitive procedures called “destination” and “location”

where allocated to each case. “destination” was to be called when the landmark reference utterance mentioned the destination and “location” was expected to be called when the utterance mentioned any other landmark specific to the route description. This posed a problem to the robot manager’s design because, as can be seen from the examples above, there is no indication, from the landmark reference or the utterance structure, of whether these refer to the destination or any other landmark. The only way to solve this problem was for the robot manager keep in memory the destination landmark throughout the route description. Remember that in the beginning of a discourse between the user and the robot the user asks the robot to “go to the <landmark>” in which case the destination is always explicit. This is when the robot manager stores the landmark’s name in memory. After that, every time the user mentions a landmark, in his/her route description, this would be compared with the destination landmark and if they are the same, the actions for destination specification are called, otherwise the actions for location specification are called for execution. It was decided that this choice would be made at the primitive procedure level and eventually only one primitive procedure called “location” was used for this purpose (see Table 4-3). This procedure has one parameter called “destination_1” that is always initialized with the name of the destination landmark stored in the robot manager’s state. Every time the “location” primitive is called the “location_1” parameter, which indicates the landmark mentioned in the user’s utterance, is compared with the “destination_1” parameter and depending on the result the appropriate course of action is taken.

A further complication to this problem is that sometimes the final destination reference is not always explicit, i.e. the name of the destination is not mentioned. Examples of such references are:

“take the first left and continue round and you should see it” (u7_GC_CD)

“take the first right and it should be on your left” (u8_GC_HL)

At the present moment the dialogue manager cannot always resolve that “it” refers to the destination landmark that was mentioned early in the dialogue. When the system fails to attribute the reference to the destination it passes an unresolved reference error to the robot manager. In these cases the system fails to recognise that there is a mention to the destination.

4.6.3 The multiple meanings of “go”

A problem arises when the user says for example “go to the train station” when the train station is ahead of the robot on the same road. In this case the user actually means “follow the road to the train station”. The robot manager, therefore, considers three possibilities when a “go to <landmark>” utterance is spoken by the user:

1. The route to <landmark> was explained in a previous description and the associated file exists in the knowledge base.
2. The route to <landmark> was not explained earlier by the user but the user mistakenly assumes the robot knows how to get there. In this case a new learning procedure must be started.
3. <landmark> is ahead along the road. The “follow_road” primitive should be used instead of the “go” primitive.

To resolve the problem, the robot manager searches the knowledge base to find the previously learned route. The outcome distinguishes between possibility 1 (a previously learned route is found) and possibilities 2 and 3 (a previously learned route is not found). If a previously learned route is not found the proper course of action would be to start a clarification dialogue with the user to resolve the issue (whether the user meant 2 or 3 above). At the present moment engaging in a dialogue with the user to resolve such issues is part of future work and therefore it is not implemented. Rather, in such a case, the robot manager selects the most probable interpretation between cases 2 and 3 above.

4.7 The concept of corpus based designed system

4.7.1 Contribution of the corpus to the natural language system design

The collected corpus of route instructions contributed in two ways to the development of the dialogue manager:

1. It determined the lexicon of the selected context and
2. It provided the syntax that humans use when giving spoken route descriptions to a robot.

The speech recognition system used in this project is speaker-independent, i.e. it can recognise any human voice without it being trained with that voice. One of the major factors affecting the success rate of speech recognition systems, which are speaker-independent, is the number of different words they can recognize. As this number increases, the speech recognition error

rate increases exponentially. To be effective to an acceptable level of recognition, a speech recognition system must have a lexicon of, at most, a few hundred words. To keep within the bounds of this limitation the speech system must be able to dynamically change the size and content of its lexicon based on the context or theme of the dialogue. In this project the context is that of route descriptions and so the lexicon of the dialogue manager was constrained only to those words found in such context. This set of words was directly derived from the collected corpus (see section 4.2).

4.7.2 Contribution of the corpus to the robot system design

The collection of the route description corpus contributed in two ways to the development of the robot manager component of the IBL system:

1. It indicated the type and structure of the primitive procedures that would need to be created for the robot and
2. It indicated the objects that the robot should be able to recognise in the miniature model town.

As mentioned earlier in this thesis, the primitive procedures are those procedures that the robot will need to have in its knowledge base when it begins its “life”. These are the tasks that a user, in the route description context, will not explain in detail. Take for example the frequent occurrence of the instruction: “turn left”. The users did not explain in their route instructions how to turn left but assumed that the human who would at some point drive the robot knew how to do it. The program, therefore, which causes the robot to perform a left

turn, had to be created before the robot started learning new routes. The “turn” procedure is one of the primitive procedures (see section 4.3).

Humans continuously use landmarks in their route descriptions. Often these landmarks are used as essential parts of the route they are describing (“turn left after the post-office”, “at the crossroads take a right”, “the hospital will be in front of you” etc). Sometimes landmarks are also used as a reassurance that the robot is on the right track (“you will see a lake on your left”, “you will pass by the library”, “there will be some trees on your right” etc). The robot needs to be able to identify these landmarks when following route instructions using vision as its only sensing ability. Information as to how the crossroads or the library looks like, or how to search the visual field for such landmarks is assumed to be known by the robot and thus such ability should be pre-programmed into it. Also knowledge related to the nature of the landmarks themselves should exist in the robot’s memory. For example the instruction: “pass the crossroad” would require the robot to do something quite different from the similar instruction: “pass the post-office”.

The landmarks found in the corpus included road layout features such as crossroads, turnings, t-junctions, the roundabout exits, signed or unsigned buildings, the bridge, the roundabout, the lake and trees. In this project only road layout features are identified by the robot. All other landmarks mentioned by the users are identified by placing a coloured strip next to them on ground plane (see sections 6.2 and 6.3).

4.8 Summary/Contribution

This chapter explained how the corpus collected was analyzed in order to determine the functional lexicon for the robot used in this project. The primitive procedures presented in Table 4-3 were derived solely from the corpus thus ensuring a purely “user-centred” approach to the design of the IBL system. It is unlikely for a roboticist to have intuitively determined these primitive procedures without studying how humans give route instructions. This is because, not only the primitive procedures would have to be determined, but also the different ways humans call each primitive action through natural language. The primitive structure should be made flexible enough to accommodate this. Consider for example the following three utterances:

“at the roundabout, take the second exit”

“enter the roundabout in a clockwise direction and take the second left turn”

“turn left in the roundabout, take the second exit”

All three user utterances are accepted and will result in exactly the same actions by the robot in the IBL system because users in the corpus have used utterances similar to these in order to instruct the robot to use the roundabout. This diversity among users could only be revealed by studying the corpus collected in this project. In the primitive procedures the different natural language forms of the same action are accommodated by the use of different procedure parameter combinations and/or values. The possible parameter combinations and values are determined by studying the corpus.

It is possible, after the development of the system, that new (i.e. unseen in the corpus) forms of primitive actions will appear in natural language instructions, which will not be covered by the possible parameter combinations and values determined in the corpus. This problem is similar to the one of altogether new primitive actions appearing after the completion of the system, which is mentioned below.

The work described in this chapter revealed three problems that are important for the future design of IBL robots. These are:

1. The probability of new primitive functions arising in route instructions after the development of the system (see section 4.3.1).
2. The cases when users make partial use of previously learned procedures while explaining new procedures (see section 4.5).
3. The under-specification of natural language (see section 4.6).

It has been shown in section 4.2 that the IBL system will be faced with new words after its completion. This has been observed also in previous work on speech recognition systems such as for example in [Zue, 1997]. In this thesis we also show that in the same way, new primitive functions can appear after the completion of the system.

This chapter also explained how users refer to a part of a previously explained route while explaining a new route. This reference is made implicitly by only mentioning a landmark at the point where the robot is supposed to stop following the previously explained route.

Determining the landmark referred to is not always apparent from the instructions in the previously learned procedure. This problem has never been documented previously.

Cases 1 and 2 above are discussed further in chapter 8.

This thesis focuses on the natural language under-specification problem. In this chapter three methods were proposed in order to determine missing information in natural language route instructions during the learning of new procedures (see Section 4.6). Section 6.2 explains how the well known method of image template matching is used in order to determine missing information during the execution of route instructions.

Chapter 5

5 The functional structure of the primitive procedures

In this chapter the functional organization of the code in primitive procedures and the reasons behind it are explained. Section 5.1 presents the generic flowchart followed by all primitive procedures found in the corpus. In Section 5.2 the need for lower level procedures is explained. These low-level procedures are called by the primitive procedures at the highest level. In Section 5.3 it is explained how the structure of the primitive procedures enables linking of a series of procedures in order to form a new “learned” procedure. Finally Section 5.4 describes the use of a “prediction function”, which is created in all primitive and new procedure files and it is used for verification and error detection in the user’s description during the “learning” stage. The prediction function predicts whether the primitive, when called during the execution of the route description, will execute without any inconsistencies due to wrong or missing parameters passed to it.

5.1 The structure of primitive procedures and how it reflects the structure of spoken instructions

With the exception of one primitive extracted from the corpus, the structure of the primitive procedures reflects the human cognitive process when following a route instruction. This process incorporates a “search-and-act” loop that exits when a terminating condition is met. The terminating condition is always associated with finding a landmark (the target). As an example consider the route instruction: “take the second turning to the left”. The target landmark is the second left turning. The search-and-act loop involves searching for the turning and moving along the road until it is found. When the second left turning is found the robot moves to where the roads meet and rotates left in order to face the new direction (target associated actions).

The exception to the loop-structure described above is the primitive “rotate”. This primitive was used only twice in the corpus (Table 4-3) and it simply causes the robot to rotate about itself 180 degrees. Therefore its execution is represented by a single pre-defined action.

Figure 5-1 shows the flowchart followed by all primitive procedures.

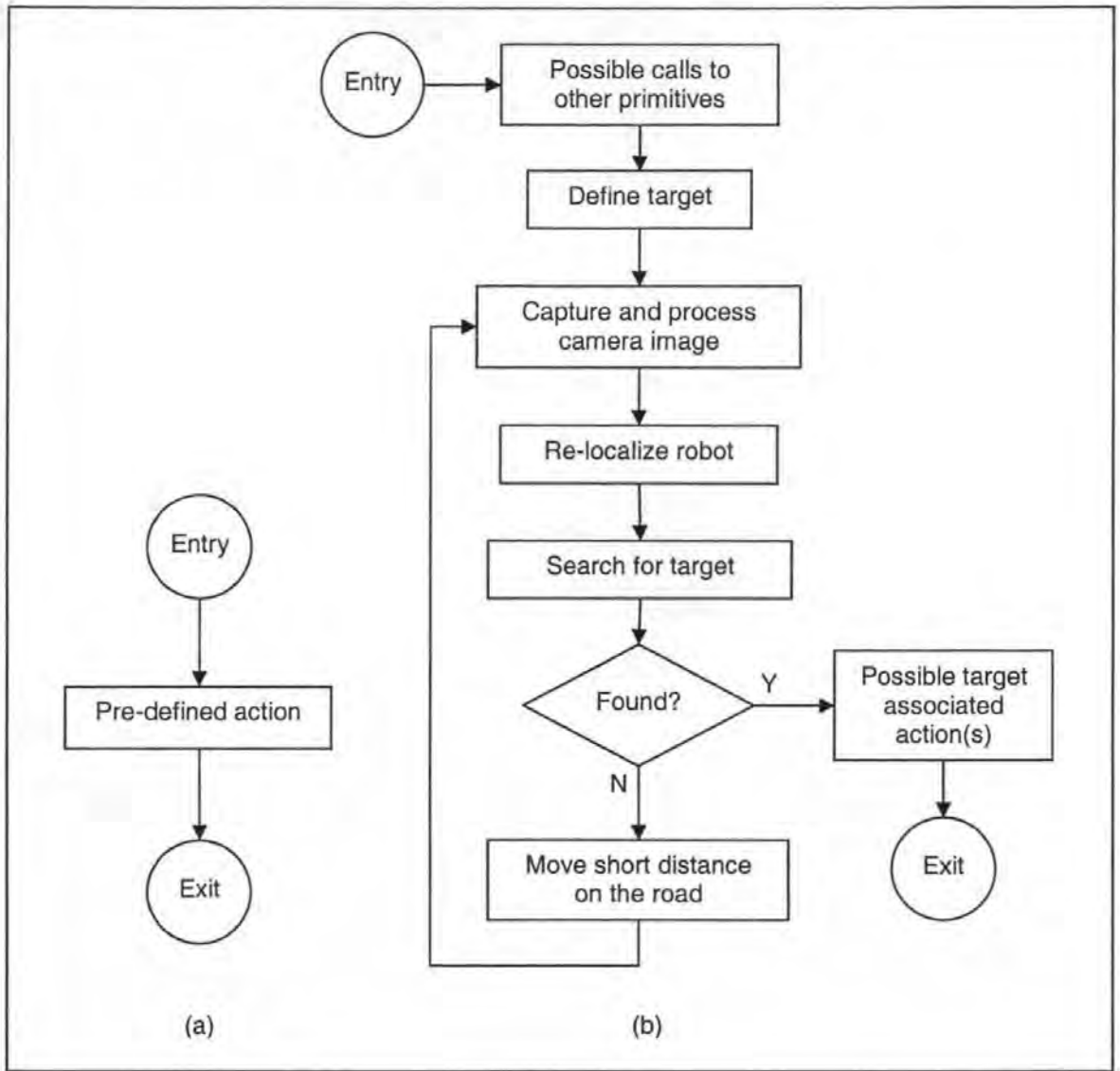


Figure 5-1: (a) Flowchart of “rotate” primitive procedure and (b) flowchart of all other primitive procedures extracted from the corpus.

The remainder of this section explains in detail the flowchart illustrated in Figure 5-1(b).

In the beginning, the target is defined based on the combination and values of parameters passed to the primitive procedure. The search-and-act loop is then entered. This consist of capturing and processing an image from the camera on-board the robot, using the new visual

data to re-localize the robot in order to determine and account for the odometric error and then searching for the target landmark in the field of view. If the target landmark is not found the robot moves along the road for a short distance before re-starting the loop. If the sought landmark is found, a set of target associated actions are performed and execution is then passed to the next primitive procedure. A detailed description of each block of the flowchart in Figure 5-1 is given in Chapter 6.

Primitive procedures can call other primitive procedures in the beginning of their body. At first glance, this may be thought to conflict with the definition “primitive” but this flexibility is only allowed to reduce the complexity of the system. Take for example the case when a user says: “after the library turn right” as part of a route description. This has to be considered as one functional component of the route description according to the definition of the functional component given in section 4.3. However, to execute the above instruction two primitive procedures are actually called that correspond to: “follow the road until the library” and “take the (first) right turn”. The mapping to the two primitive procedures can be done in two ways:

1. The robot manager can call the two primitive procedures individually. For example:

```
follow_road(relation_1="after", object_1="library")  
turn(relation_1="right_of", object_1="self")
```

2. The robot manager can call the “turn” primitive only with the necessary parameters so that this primitive would then call the “follow_road” primitive from within itself. For example:

```
turn(relation_1="right_of", object_1="self", relation_2="after",  
      object_2="library")
```

It was decided that it was easier to follow the second approach in such cases because it was more difficult for the robot manager to determine from the Discourse Representation Structure (DRS) the implicit action to “follow the road until the library” in the example given above.

The Python program code of all primitive procedures is included on the CD accompanying this thesis (see Appendix C).

5.2 The use of low-level procedures

It quickly became apparent during the implementation of the primitive procedures that these required to call low-level procedures from within their body. These were called “low-level” because they perform specific actions (more fundamental than that of the primitive procedure) and each can be used by more than one primitive procedure. These low-level procedures are not accessible directly by the human user (Figure 5-2), i.e. no functional speech segment of the route instructions could be directly mapped to them.

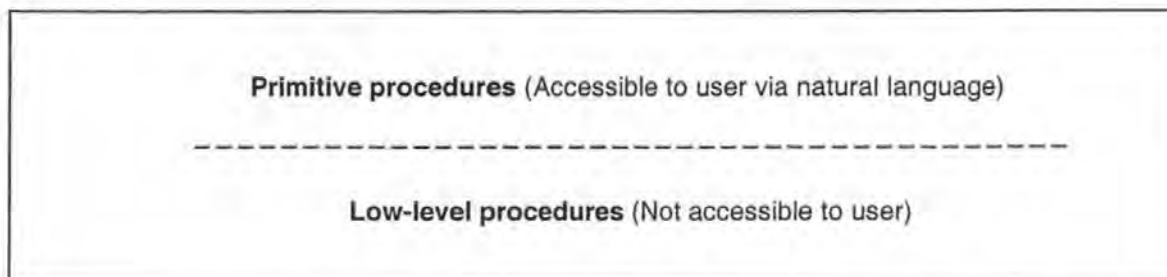


Figure 5-2: Primitive procedures can be accessed by users via natural language whereas low-level procedures cannot.

The low-level procedures needed to be created to avoid duplication of code in the implementation of the primitive procedures.

Some examples of low-level procedures are shown in Table 5-1:

<code>capture_image()</code>	Captures an image from the camera on-board the robot and saves it to an image file to be used for future processing.
<code>optical_calibration()</code>	Corrects optical distortion on the captured camera image.
<code>produce_world_perspective()</code>	Applies the inverse perspective mapping transform to obtain the top view of the robot's view. The top view image is saved to an image file.
<code>found_object(object)</code>	Scans the robot's view in order to find object. Returns true or false. If true (object is found) the object's location is also returned.
<code>move_to(x, y, θ)</code>	Causes the robot to move to a specified location relative to its own and once there turn at a specific angle.
<code>short_move_on_road()</code>	Causes the robot to follow the road for a short distance. This distance is short enough so that no visual information is lost.

Table 5-1: Examples of low-level procedures.

5.3 The verification of new procedures during their creation

As mentioned earlier, new procedures are learned by combining previously learned procedures (mostly primitive procedures) from the knowledge base of the robot. When the user describes a new procedure as a sequence of actions, it is important for the robot to verify if this sequence is executable before it saves the sequence into memory. The approach used in this project is to associate each procedure with a triplet $S_i A_{ij} S_j$ with properties similar to productions in SOAR (see [Laird et. al., 1987]). The state S_i is the pre-condition for action A_{ij} . It defines what conditions must be satisfied by the robot's state for action A_{ij} to be possible. The state S_j is the final state, resulting from the action A_{ij} applied to the robot's state. For a sequence of actions to be realisable, the final state of one action must be compatible with the pre-condition of the next one. To enable this verification, the robot must be able to "imagine" the consequence of an action. For that purpose, a "prediction" function is associated with each primitive action, and with each newly created procedure. This is described in more detail in the following section. Figure 5-3 illustrates the use of the prediction function during verification of the consistency of the sequence of instructions from the user.

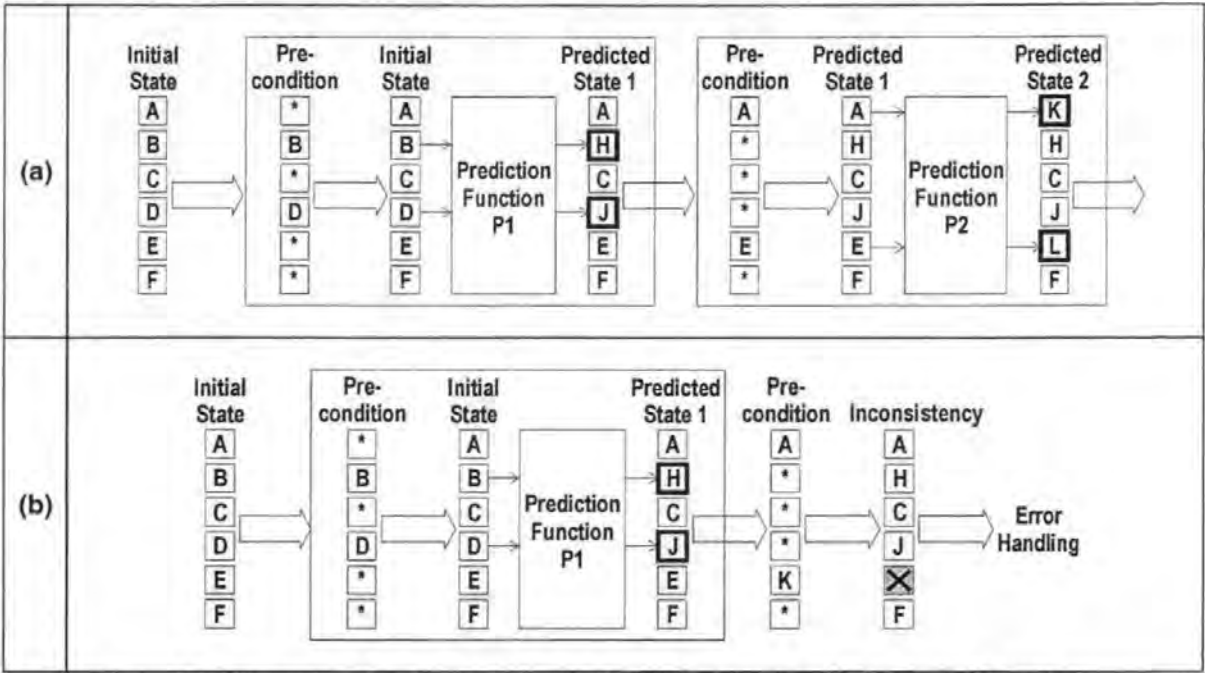


Figure 5-3: Illustration showing how the prediction function in primitive procedures is used. Row (a) shows a case where the state of the robot after executing procedure P1 is consistent with the next procedure to be executed P2. Row (b) shows the case where there is an inconsistency between the state the robot is left in after executing P1 and the expected state for the next procedure to be executed. For a more detailed explanation of the figure see text below.

For each procedure there is a prediction function that transforms a state vector into its future value (Figure 5-3(a)). The function first determines if the input state satisfies the minimal criteria ("pre-condition") to enable the procedure to be executed. An action is executable only if selected elements of the state vector have required values. If this is the case, the next state is predicted and processed by the prediction function associated with the next procedure in the instruction. Each action modifies certain components of the state vector, and leaves the others unchanged. If the predicted state produced by one procedure does not allow the next procedure to be executed, an error handling process is initiated (Figure 5-3(b)).

Any inconsistencies detected may mean that the user has made a mistake in his/her description or the system misinterpreted what the user said. In this case a simple dialogue with the user is started to clarify the problem.

5.4 The “prediction” and “action” functions of primitive procedures

Every primitive procedure and newly created procedure is composed by a “prediction” and an “action” functions. Both these functions exist in the procedure’s module file and take the same parameters with one exception: an extra parameter called “state” is passed only to the prediction function. This state is modified and returned by the prediction function.

As explained in the section 5.3 the prediction function of a procedure is used to predict the future state of the robot, given its current state, when the procedure is executed. Also the prediction function can detect any inconsistencies between the state of the robot and the expected state, which is required for the successful execution of the procedure. During the learning of new procedures, when an instruction given by the user is mapped to a procedure in the knowledge pool of the system, the prediction function of that procedure is called with the current “virtual” state of the robot in order to check whether this state satisfies the pre-condition for the procedure to execute. If no inconsistencies arise, the new virtual state is returned by the prediction function and the procedure call is added to the new procedure file. This is repeated with the new instruction of the user until he/she finishes his/her description. If there is an inconsistency, however, the prediction function returns an error message to the robot manager indicating the problem. In this case the system attempts to rectify the problem

in a simple way by responding to the user with: “I did not understand what you said” expecting the user to repeat.

The prediction function of every primitive procedure makes three checks every time it is called:

1. Parameter combination check.
2. Parameter value check (for every parameter passed to the procedure).
3. State check.

The checks are made in the order presented above and if any inconsistency occurs along the way, the prediction function returns without checking for any further inconsistencies. Table 5-2 shows the pseudo-code of the prediction function indicating the three checks.

```

prediction(state, parameter_1, parameter_2, ..., parameter_n)
{
    /// Parameter combination check //////////////////////////////////////
    if passed_parameters_list not in valid_parameter_combinations
    {
        return parameter_combination_error
    }

    /// Parameter value check //////////////////////////////////////
    if parameter_1 not in parameter_1_accepted_values
    {
        return [parameter_value_error, parameter_1]
    }

    if parameter_2 not in parameter_2_accepted_values
    {
        return [parameter_value_error, parameter_2]
    }

    .
    .
    .

    if parameter_k not in parameter_k_accepted_values
    {
        return [parameter_value_error, parameter_k]
    }

    /// State check //////////////////////////////////////
    list_of_valid_states = [state_1, state_2, ..., state_k]

    if state not in list_of_valid_states
    {
        return [state_error, state]
    }

    .
    .
    .

    return predicted_state
}

```

Table 5-2: Pseudo-code of the prediction function in primitive procedure modules.

The parameter combination check makes sure that the combination of parameters passed to the primitive is one of the allowed combinations (see Appendix A for the allowed parameter

combinations of each primitive procedure). Once the first check is passed, the second check makes sure that the value of every parameter passed to the procedure is among the allowed values for that parameter for the specific procedure (see Appendix A for the allowed parameter values of each primitive procedure). Finally, the state check verifies that the virtual state of the robot (predicted by the previous procedure) is among the valid states compatible with the action to follow. All primitive procedures have their own list of valid states. The state of the robot must be the same as one of the members in this list in order for the state check to be successful. For example, consider the case when a user says: “carry on to the end of the street” and then he/she continues by saying: “follow the road to...” In this case the state check will fail because the current state value in the virtual state variable will be “end_of_road” after the execution of:

```
follow_road(relation_1="to", object_1="end_of_road")
```

that corresponds to the first utterance of the user. This state value will not be among the valid states after the execution of the above “follow_road” primitive call since the robot at the end of the road (whether this is a dead-end or a t-junction) does not have a road ahead of it.

The action function in the procedure’s module is the one containing the commands which, when executed, cause the robot to perform the action instructed by the user. The different operations that take place in the action function of the primitive procedures are explained in detail in chapter 6.

The prediction and action functions of a new (learned) procedure are composed from the prediction and action functions of its constituent procedures. This is shown diagrammatically in Figure 5-4.

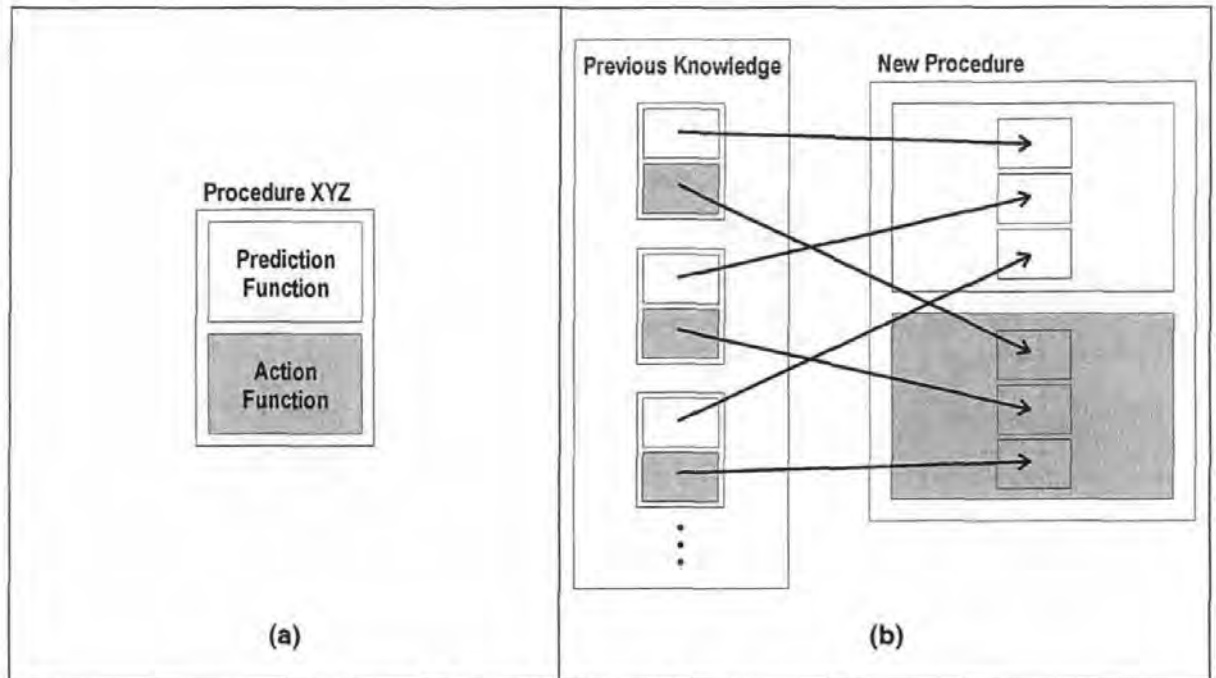


Figure 5-4: Procedural knowledge representation.

The contribution of the author of this thesis to the synthesis of new (learned) procedures in the IBL (instruction based learning) system was to create the prediction and action functions in each primitive procedure file in such a way so as to enable the straightforward copying of the program code into the new procedure files.

5.5 Summary/Discussion

In section 5.2 the requirement for low-level procedures in primitive procedures is described. These were created in order to achieve code efficiency and reusability. Apart from that

though, the appearance of such low-level procedures revealed a more important fact: it was the level of specialisation of the human users within the context that determined how fundamental, to the robot, primitive procedures are. For example, if all human subjects invited for the corpus collection were roboticists then the primitive procedures would probably refer to more fundamental tasks such as those listed in Table 5-1.

The existence of tasks at different levels is mentioned in [Lueth et. al., 1994] where a dialogue interface for a robot performing mechanical assembly tasks is explained. The main difference with the work presented here is that in [Lueth et. al., 1994] all the tasks that the robot can perform are accessible to the user via the natural language interface. This is to allow a more elementary control of the robot. As with previous approaches described in section 2.1 the complete task vocabulary of the robot described is determined using a robot-centred approach.

The use of a prediction function in primitive procedures (described in section 5.4) provides a mechanism for the robot to prevent failure in the execution of a route description before the execution starts and while it still has the attention of the user. Errors in the execution of a route description can occur because of four reasons:

1. When the user makes a mistake in the route description.
2. When the user is ambiguous.
3. When the user does not provide enough information for the execution of an instruction.
4. When the system wrongly recognizes what the user said.

It is important to realize that the prediction function will not always detect errors because of the above reasons. This is because the outcome of the above cases can sometimes result in a valid procedure call that will produce an action not intended by the user.

Chapter 6

6 Vision for robot navigation

The robot used in this project uses a camera as the only sensor of its environment. Images from the camera (see example in Figure 6-1) are sent by wireless video link to a PC, which processes them to extract information specific to the route description that the robot is following.



Figure 6-1: An example of a raw (unprocessed) robot camera image.

The robot camera image is used in three ways:

1. For robot localization.
2. To establish the location and orientation of objects mentioned in route descriptions.
3. To establish the location of the next waypoint that must be reached by the robot while it is moving towards its final destination.

To achieve the above tasks several pre-processing operations are performed on the raw camera image first. These are described in section 6.1.

Landmarks referred to in route descriptions are categorized in two groups for the purposes of this project:

1. Road layout features (such as turnings, crossroads, t-junctions etc) and
2. Non road-layout objects (such as trees, buildings, the bridge etc).

Section 6.2 explains how road layout features are found in the robot's view using the well known method of image template matching.

Although originally planned, the duration of this project did not allow for the development of image processing routines, which would recognize non road-layout objects mentioned in the natural language route descriptions. Section 6.7 explains how non road-layout landmarks mentioned in route descriptions are detected using a coloured marker placed next to them.

In order to be able to determine and account for the odometric error introduced every time the robot moves (localization) and also to keep in memory important visual information previously “seen” by the robot, which afterwards falls outside its field of view, a “short-lived” map of the environment is created as the robot moves. The “short-lived” map is described in section 6.3.

In sections 6.4 and 6.5 it is explained how road surface and the road edge information are extracted from the robot’s view. Road surface information is used in the template matching process and road edge information is used in the creation of the “short-lived” map.

Finally section 6.8 describes how spatial references to landmarks are used for successful robot navigation.

6.1 Capturing and pre-processing the robot camera image

Two successive operations are performed on the raw camera image after it is captured by the robot:

1. Optical calibration and
2. Inverse perspective mapping.

Optical calibration is applied to the raw camera image to correct the distortion produced due to the optics of the camera lens. Figure 6-2 shows the same image before and after optical calibration.



Figure 6-2: (a) Raw camera image and (b) the same image after optical calibration.

The mathematical formula that provides the corrected location of a pixel after optical calibration is taken from [Faugeras, 1993]. It is reproduced here below:

$$\begin{aligned} x' &= x_c + (x - x_c) \left[1 + k_1 \left[(x - x_c)^2 + (y - y_c)^2 \right] \right] \\ y' &= y_c + (y - y_c) \left[1 + k_1 \left[(x - x_c)^2 + (y - y_c)^2 \right] \right] \end{aligned} \quad (6-1)$$

Where (x, y) are the coordinates to be corrected, (x_c, y_c) are the coordinates of the camera's image centre and k_1 is the optical distortion coefficient. The experimental method followed to obtain k_1 is similar to the one explained in [Koay, 2002]. In short, this is done by placing a calibration pattern (such as a mesh of known dimensions) at a known distance in front of the camera so that the camera's optical axis is perpendicular to the calibration pattern's plane. After determining the camera's image centre coordinates x_c, y_c (this is where no distortion occurs on the captured image), the value of the optical distortion coefficient k_1 is found by

trial-and-error. This is done by changing the value of k , until the corrected image displays the calibration pattern undistorted. For the images captured by the camera in this project k , was found to be 6.5E-6.

Inverse perspective mapping is applied to the optically calibrated camera image to produce a top view (or “eagle’s eye view”) of the scene that the robot is facing. However, this is a pseudo-view because it can only show true geometrical information of objects existing only on one plane. For the purposes of this project, the road surface plane is chosen to be consistent with the true top view of the scene. All three-dimensional objects appear distorted in this view (Figure 6-3(b)).

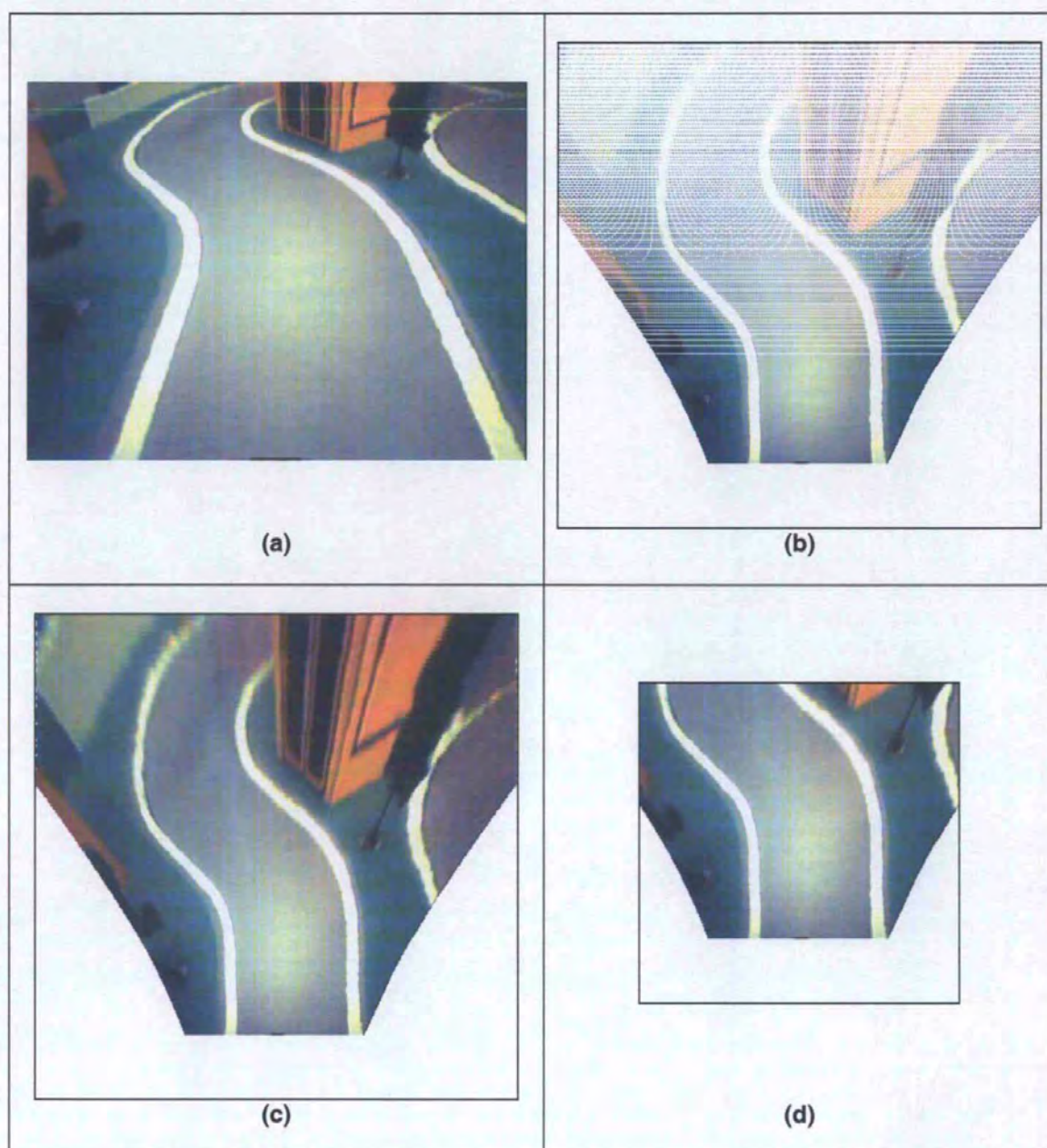


Figure 6-3: Illustration showing how inverse perspective transformation is performed on the robot's camera image. (a) Shows an example of an optically calibrated camera image, (b) shows the result when inverse perspective mapping is applied to (a). Note the missing information due to the sampling effect in (b). In (c) the missing information is interpolated using neighbouring pixels containing information. (d) is the part of (c) used by the primitive procedures for further processing.

This distortion occurs because the inverse perspective transform takes as input a two-dimensional image (the camera image). Therefore only one plane (out of the infinite possible) can be selected to be consistent with the real world. In this project the road plane is selected for this purpose. The following equations describe the inverse perspective mapping function:

$$d = h \cdot \tan \left(\frac{\pi}{2} - \theta + \arctan \left(\frac{(y_c - y) \cdot \sqrt{2 \cdot (1 - \cos(\alpha))}}{I_h \cdot \cos\left(\frac{\alpha}{2}\right)} \right) \right) \quad (6-2)$$

$$e = \frac{x - x_c}{I_w} \cdot 2 \cdot \sqrt{h^2 + d^2} \cdot \tan\left(\frac{\beta}{2}\right)$$

Where (d, e) are the world coordinates of a point represented by the camera image coordinates (x, y) . Angle θ is the tilt angle of the camera, (x_c, y_c) are the coordinates of the camera image centre, α and β are, respectively, the camera's vertical and horizontal maximum angles of view and h is the distance between the camera and the ground plane.

Notice that the height and inclination of the robot camera are the two parameters that define which plane (in the real world) will be geometrically consistent with the top view image after the transform is applied.

Considering the road surface plane, notice that pixels in the lower part of the camera image correspond to visual information in the plane closer to the robot. Likewise, pixels at the top part of the camera image correspond to visual information further away from the robot (Figure 6-4).

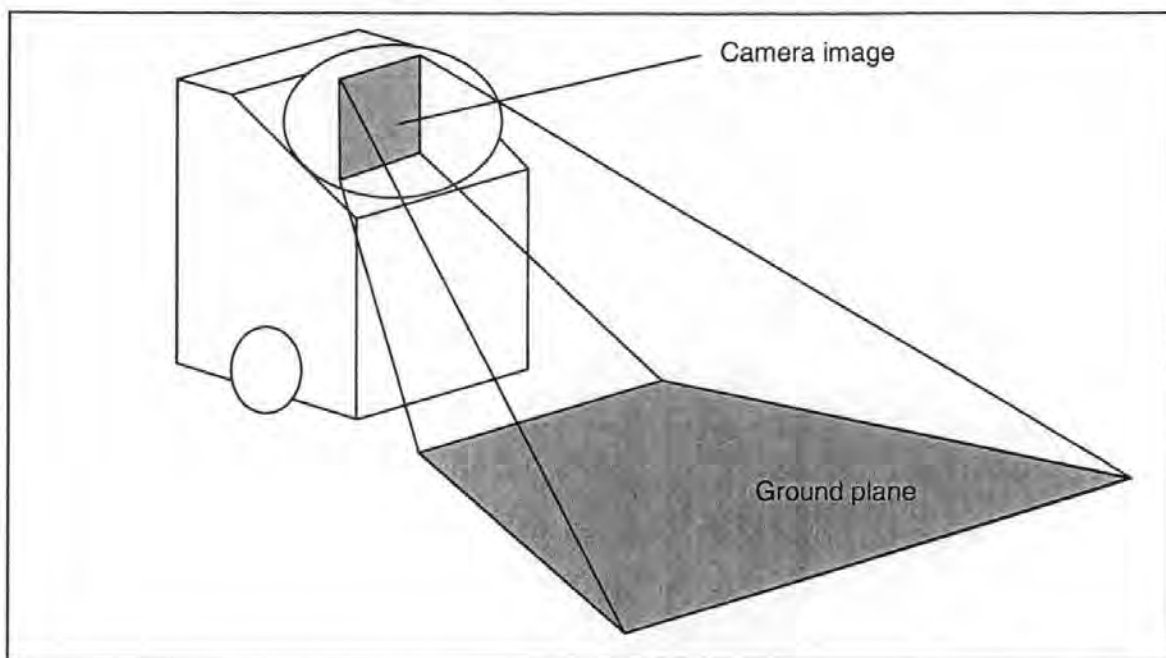


Figure 6-4: Diagram showing the correspondence of pixels on the image plane with pixels on the ground (or road surface) plane.

Because of the digital nature of the camera image, the inverse perspective mapping transform does not have information for areas of the world in-between pixels of the camera image. Due to this reason, discontinuities appear in the resulting top view image (see Figure 6-3(b)).

Notice that these discontinuities are larger at the top of the image. This is because the corresponding world distance between two adjacent pixels in the camera image is larger at the top of the camera image than at the bottom.

In order to improve the top view image, the discontinuous areas are interpolated using information from their most adjacent pixels that contain visual information. The result of this interpolation is shown in Figure 6-3(c). Figure 6-3(c) is likely to be a more “faithful” reproduction of the real world in areas closer to the robot rather than further away from it.

This is because the sampling at the bottom of the top-view image is less severe than at the top and therefore any interpolations at the bottom of the image are likely to be more accurate.

Using the whole of Figure 6-3(c) would take a lot of computational time to perform future image processing. Furthermore, the robot does not require visual data as far as Figure 6-3(c) extends since it will not be able to move that far in one go because of its accumulating odometric error. Therefore a smaller section of Figure 6-3(c) is used for the purposes of further image processing. This has dimensions 100x100 pixels and it is the lowest part of Figure 6-3(c). This is shown in Figure 6-3(d). The scale of this is 0.003330m/pixel. This means that the robot uses only visual information consistent with its plane of motion and its useful view extends as far as 0.333m away from its position. From this point onwards this image matrix will be referred to as the “top view image”.

The two pre-processing steps described above (i.e. optical calibration and inverse perspective transform) are always performed on every new image captured by the robot. In order to save computational time equations 6-1 and 6-2 are used to produce a “transformation lookup matrix”. This matrix is of the same size as the top view image and contains in each element the coordinates of the pixel in the raw image that needs to be copied to the corresponding location in the top view image to achieve both pre-processing steps.

6.2 Detection of road features using image template matching

In this section, the method for locating road layout features mentioned in route descriptions is described. Such features include left/right turnings, t-junctions, roundabout entries/exits, road ends, road bends and crossroads.

In order for the robot to navigate successfully to its destination, it needs precise location information of any landmarks mentioned in the route description. In most cases the robot must move to the landmark's location and perform a manoeuvre. For example to "turn left at the crossroads" the robot needs to move to the centre of the crossroads and rotate left to face the new direction.

In the example given above the precise location of the crossroads is not explicitly given by the user and therefore it has to be determined by the robot during the execution of the road instruction. To do this, the robot needs to first identify the road feature (the crossroads) and then select a point on the feature to navigate to (centre of the crossroads). Then it needs to perform the requested action (rotate left).

To identify road layout features in this project a simple form of template matching is used. Section 6.2.1 explains briefly what is template matching and refers to previous research in this area. Section 6.2.2 presents the templates of the template matching method used in this thesis, section 6.2.3 explains the template matching procedure and section 6.2.4 describes how the

template matching method is used in the primitive procedures to achieve successful robot navigation.

6.2.1 Template matching

Template matching is a method that falls under the broader scope of image matching or image registration. Image matching is a well researched field spanning over the last forty years (see [Rosenfeld, 1969], [Niblack, 1986], [Jain, 1989] and [Gonzales and Woods, 1992]). The aim of image matching techniques is to obtain a measure of similarity (or difference) between two images. One of the images (usually called the reference image) is geometrically transformed so that each point in it can be mapped to a point in the other image. This transformation can involve rotation, translation and scaling of the reference image. For each such transformation a similarity (or difference) value is calculated based on the properties of the overlapping regions in the two images.

The method of calculation of the similarity (or difference) measure between the two images depends on several factors such as the area of application of the image matching operation, the available computing power, the required precision of the result, the available image information etc.

A comprehensive review of image matching techniques and examples of the wide spectrum of applications where the different approaches are used can be found in [Brown, 1992] and [Aschwanden, 1992].

Template matching is one variation of image matching. In template matching the reference image (or template image) is an image of an object of interest, which is sought in the main image. A sufficiently good match of the template in the main image reveals the presence (and location) of the object represented by the template in the main image (see [Rosenfeld and Kak, 1982], [Pearson, 1991] and [Pratt, 1991]).

In this thesis template matching is used to locate road layout features in the robot's view. The templates used in this method are pre-constructed images of road layout features. These are presented in the following section.

6.2.2 Road feature templates

The templates used in this thesis are binary images (indicating road and non-road regions) of local road surface features drawn at the same scale as the short-lived map. Fifteen templates are used by the primitive procedures. They are shown in Table 6-1.

 (a)	 (b)	 (c)	 (d)	 (e)
 (f)	 (g)	 (h)	 (i)	 (j)
 (k)	 (l)	 (m)	 (n)	 (o)

Table 6-1: The templates used for the template matching method. Light grey colour indicates road-like areas and the black colour represents non-road areas. The templates shown are used to find: (a) straight road, (b) end of road, (c) left and (d) right turnings, (e) crossroad, (f) left and (g) right bends, (h) t-junction, (i) roundabout entry, (j) clockwise and (k) anti-clockwise curved road in roundabout, (l) left and (m) right roundabout exits, (n) left and (o) right 90-degree turns.

In searching for a road layout feature mentioned in a route instruction the associated template image is matched against the road surface map of the top view. The road surface map is a binary image showing road and non-road regions in the front vicinity of the robot (Section 6.3 explains how the road surface map is created). A good matching position of the template on the road surface map provides the location and orientation of the road feature, which are vital for the successful execution of the route instruction.

Note from Table 6-1 that some template shapes are more generic than others thus covering a range of possible road layout features described the same way in natural language route instructions. For example the right turning template (Table 6-1, image (d)) represents a range of right turnings from approximately 45 to 135 degrees to the direction of the road. This is illustrated in Figure 6-5 for three angles (45, 90 and 135 degrees to the main road).

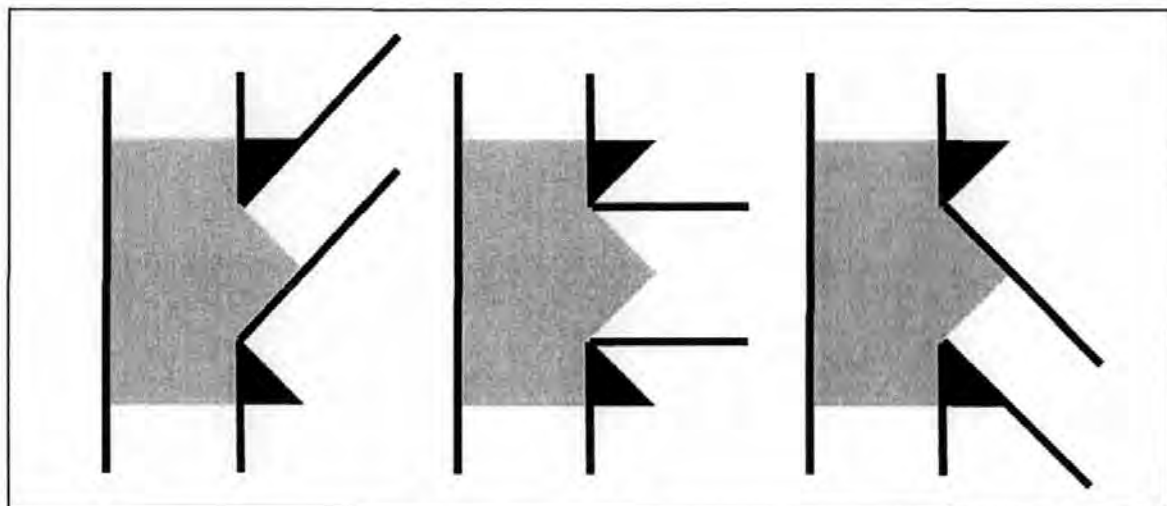


Figure 6-5: Template (d) of Table 6-1 is used to represent a range of possible right turnings at different angles to the main road.

Where the thick black lines in the figure represent the edges of the road.

The following section describes how the template images are used by the primitive procedures in order to locate the road layout features mentioned in route instructions.

6.2.3 The template matching procedure

Each template is associated with a pivot point and a “new direction” vector. These are shown for some templates in Table 6-2.

			
(a)	(b)	(c)	(d)

Table 6-2: Pivot point (dot-centred circle) and direction vector (arrow) for some of the templates.

Translation and rotation of the template during the matching process is done with reference to its pivot point. The pivot point of the matching template is mapped into real-world coordinates and this becomes the next waypoint for the robot. The direction vector indicates to the robot the direction it must turn to, after reaching the template waypoint, in order to keep facing the road ahead.

While searching for the best matching position, the road edge image is displaced and rotated (vector $[x,y,\phi]$) so that its pivot point scans the road surface map image. Figure 6-6 illustrates one position of the template on the map image during the matching process. The road layout feature sought in the example is the left turning.

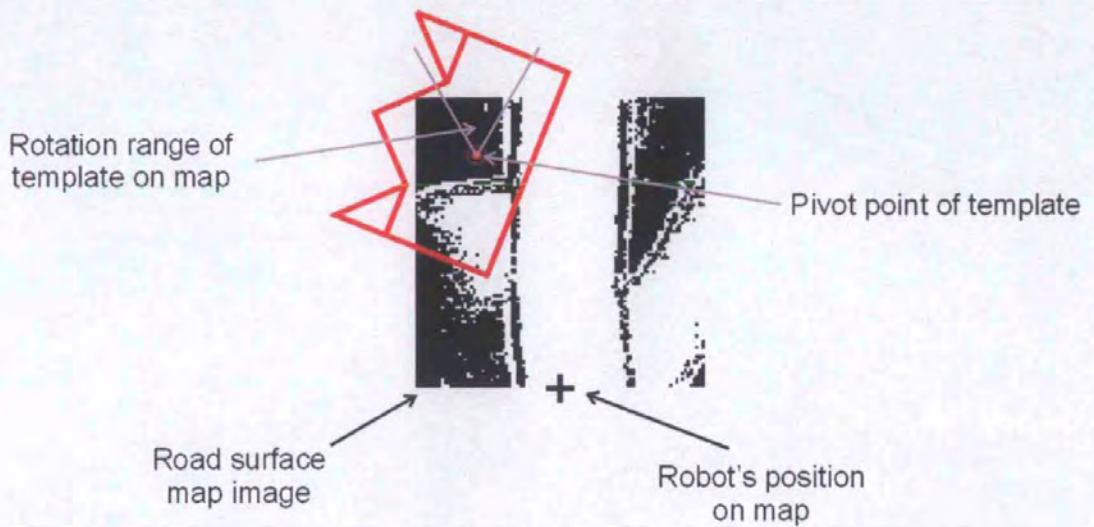


Figure 6-6: Illustration of one position of the template image on the road surface map while searching for the best matching position.

Note that only the outline (in red colour) of the associated template is shown for clarity.

The matching process for each location and orientation (vector $[x, y, \phi]$) of the template on the road surface map produces a matching quality Q_i . Variable Q_i is made from the sum of two ratios:

1. The score, which is the sum of the matching road and non-road pixels in the two images divided by the number of template pixels falling onto areas of the map where information is available and
2. The confidence factor, which is the fraction of template area falling onto areas of the map with road surface information.

Both, the score and confidence terms, are required to give an indication of how good a matching position is. The score gives an indication of how “well” one image matches on the

other and the confidence gives an indication of how much image area (compared to the total area of the template image) was considered to obtain the score. The bigger the area considered, the more believable is the score value.

This is expressed formally by the following equation:

$$Q_i(x, y, \phi) = \frac{\sum_{p \in T(x, y, \phi) \cap M} XOR(m_p, t_p)}{\sum_{p \in T(x, y, \phi) \cap M} NOR(m'_p, t'_p)} + \frac{\sum_{p \in T(x, y, \phi) \cap M} NOR(m'_p, t'_p)}{\sum_{p \in T} t_p} \quad (6-3)$$

$m, m', t, t' \in \{0,1\}$

Where p is a pixel location in the overlapping area of the template and road surface map images. Variables m and t are values of pixels in the road surface map image M and template image T respectively. Value 0 denotes no road, and value 1 denotes road. $T(x, y, \phi)$ is the template image translated by (x, y) and rotated by ϕ . Variables m' and t' are the information masks of the map and template images where 0 denotes the presence of information (mask is off) and 1 denotes no information (mask is on). The binary functions XOR (exclusive or) and NOR (not or/inverse or) are used in equation 6-3 to avoid more complicated algebraic expressions. This has been possible here because the images operated upon and their information masks are binary.

The best matching position and orientation of the template is the one where Q_i is maximum. Equation 6-3 ensures that, for two configurations with equal score, the one with highest confidence is the winner.

This is essentially no different than computing the sum of the absolute (or square) of the differences between overlapping pixels in the two images and normalising over the overlapping area between the two images. The aim would then be to find a template position that would give a minimum rather than a maximum score. This method is described in [Rosenfeld, 1969].

Because template matching is a costly operation as far as computer processing time is concerned different methods exist to locate the best template matching position in a more efficient manner. In this thesis a simple “hill-climbing” method described in [Rosenfeld and Kak, 1982] is used to speed up the matching process. The method requires that the correlation between the template and the map image contains relatively smooth and broad maxima. In other words the matching quality between neighbouring template transformations in the (x, y, ϕ) space should vary in a relatively gradual manner. In order to find the best matching position of the template, a crude search is performed initially using coarse steps of position and rotation of the template on the map. The search is then refined for a more accurate determination of the position and orientation of the best matching position. This is illustrated in Figure 6-7.

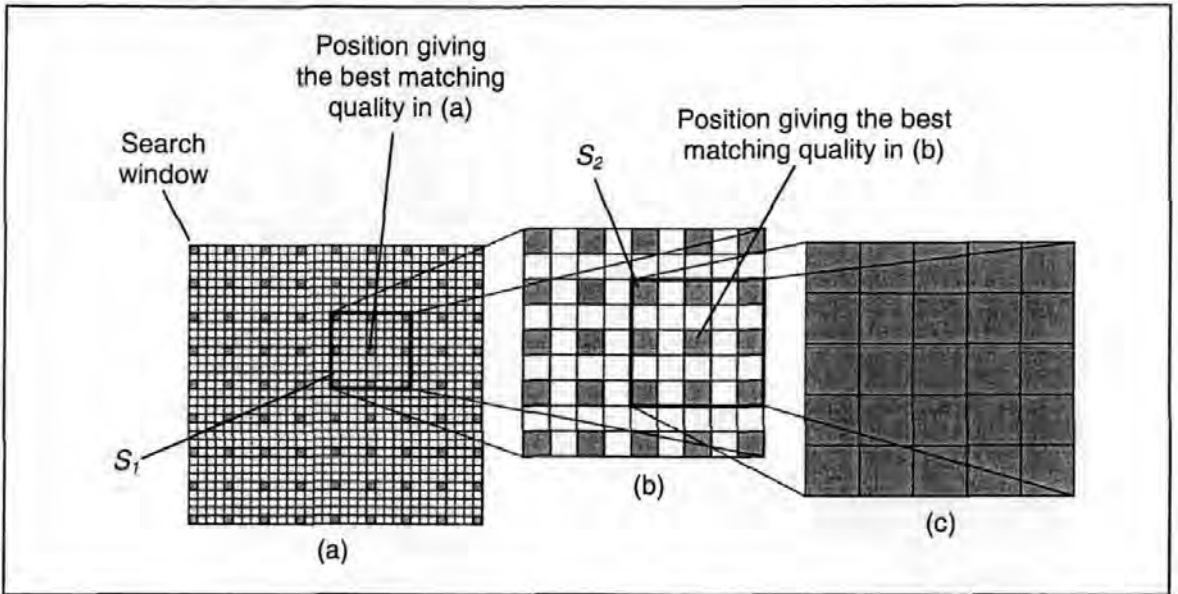


Figure 6-7: Illustration showing how the matching of the template is performed on the road surface map in order to save computational time. This starts with coarse scanning (a) of the pivot point (of the template) in the map (1 in every 4 pixels shown in grey colour). For the second step an area S_1 (shown magnified in (b)) is selected for a finer scan (1 pixel in 2) around the position that produced the best matching quality in the first scan. Each side of S_1 is equal to twice the scan step in (a). In the same way scan area S_2 is selected from (b). All pixels of S_2 are scanned in order to give the best possible matching position.

Note that the saving in computational time for the example shown in Figure 6-7 is 88%.

Every template is associated with a minimum quality Q_{1min} . If the template matching quality Q_i is less than Q_{1min} then it is assumed that a road feature associated with the template does not exist in the robot's view. In the case illustrated in Figure 6-6 the best matching position gives a matching quality above Q_{1min} indicating the presence of a left turning. The best position of the template is shown in Figure 6-8.

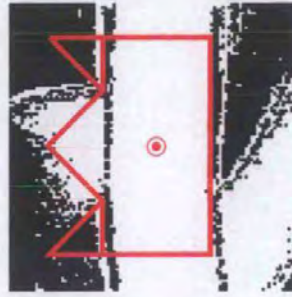


Figure 6-8: The best matching position of the left turn template on the road surface map. Note how the pivot point of the template indicates the next waypoint of the robot.

6.2.4 How template matching is used in primitive procedures

As described in section 5.1 the first task in the “action” function of every primitive procedure (except the primitive “rotate”) is to define the target (Figure 5-1(b)). This is always a landmark mentioned by the user which, when found by the robot, the primitive exit its “search-and-act” loop and passes execution to the next primitive. This section explains how landmark searching is achieved for road layout features.

In every iteration of the “search-and-act” loop the robot searches for the target landmark in its view. If this is not found, the robot moves a short distance along the road and repeats the loop sequence again. If the target landmark is found the robot performs any possible target-associated action(s) (see section 5.1) and the primitive procedure exits. The landmark to be sought in the “search-and-act” loop and its associated actions are decided in the beginning of the primitive procedure from the parameters passed to the primitive. For example, when the user says: “turn left at the t-junction”, the target is the t-junction. The search-and-act loop will exit if a successful matching position for the t-junction template (Table 6-1(h)) is found. The target associated actions are to:

1. Move to the waypoint mapped to by the template's pivot point.
2. Turn to the new direction according to the template's direction.
3. Rotate 90 degrees to the left.

Different users in the corpus sometimes used the same words to refer to two different road layout landmarks. For example in the following corpus segments:

“take a left turn at the junction” (u9_GC_HC)

“take a left turn” (u9_GC_HD)

“turn left” (u24_GB_HD)

users were actually instructing the robot to turn left at a t-junction (see Figure 6-9). Note that their instructions could have equally been valid for a left turning.



Figure 6-9: Users u9 and u24 were asked to explain a route starting from the Hospital (H). Their first instruction referred to the t-junction the robot would meet.

For this reason, depending on the user's instruction and therefore the parameters passed to a primitive procedure, a landmark sought in the primitive may be represented by a set of possible templates instead of just one. In the examples given above it is not clear whether user's u9 and u24 refer to a turning or a t-junction and therefore the target in the "turn" primitive would involve finding a good matching position for either template (c) or template (n) of Table 6-1. Each template would have its own associated actions, which would be executed if the template succeeds.

When, after searching its view, the robot does not find the landmark it is looking for, it follows the road it is on for a short distance before it searches again. Following the road is also a task that involves template matching. For this purpose, the robot uses one of three templates depending on its state, i.e. with reference to Table 6-1: if the robot is on a straight or slightly curved road, template (a) is used, if the robot is in the roundabout in a clockwise direction, template (j) is used and if the robot is in the roundabout going round in the anti-clockwise direction, template (k) is used to follow the road. The template associated actions with these templates are simply to:

1. Move to the waypoint mapped to from the template's pivot point.
2. Turn to the new direction according to the template's direction vector.

A video example showing the execution of the route instruction "take the second left" can be found on the CD accompanying this thesis (see Appendix C). The example illustrates how

template matching takes place in real-time and also how localization and mapping (described in the following section) occurs.

6.3 The use of a short-lived map

A short-lived map is a map of the immediate vicinity of the robot that is updated as the robot moves in its environment. The map records previously seen visual information that goes out of view as the robot moves. The dimensions of the map are 100 x 100 pixels (i.e. same size as the top view image). The robot's position on the map is always in the middle of the bottom edge and facing the top of the map (i.e. same as in the top view image). As the robot moves, the map is translated and rotated to maintain this frame of reference. In the process, elements of the map that reach its edge will disappear, thus the term "short-lived".

Two versions of the short-lived map are used, the first shows areas of the road surface and the second shows the road edges. These versions are constructed using road surface and road edge information filtered out from the top view. Details of how the road surface and road edge images are obtained are given in section 6.4 and 6.5 respectively.

The purpose of constructing a "short-lived" map is twofold:

1. To be able to determine the odometric errors of the robot and
2. To compensate for the "dead angles" of the robot. These are the areas close to the robot that fall outside its field of view due to the position and inclination of the camera (see Figure 6-10).

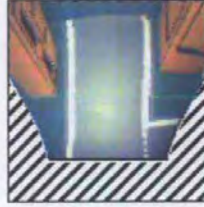


Figure 6-10: Illustration showing the “dead angles” of the robot.

The line-shaded region in the figure indicates an area falling outside the field of view of the robot. Visual information in this area is important because it is close to the robot’s position. In the example the right turning cannot be properly seen.

Every time the robot moves, it needs to know its new location compared with the previous one. This is because it records and maintains in its memory the position and orientation of landmarks (with reference to its own position and orientation) found earlier and which may not be present in its view when it reaches its new location. The only way of knowing the position of these landmarks is by adding to their position vector (position of a landmark relative to the robot) the robot’s motion vector (position of the robot’s next waypoint relative to the robot’s current position) each time the robot moves, a process otherwise described as “dead-reckoning”. The odometric error of the robot must also be taken into account for this calculation to be accurate. This error is the difference between the motion vector and the actual robot’s displacement vector (see Figure 6-11).

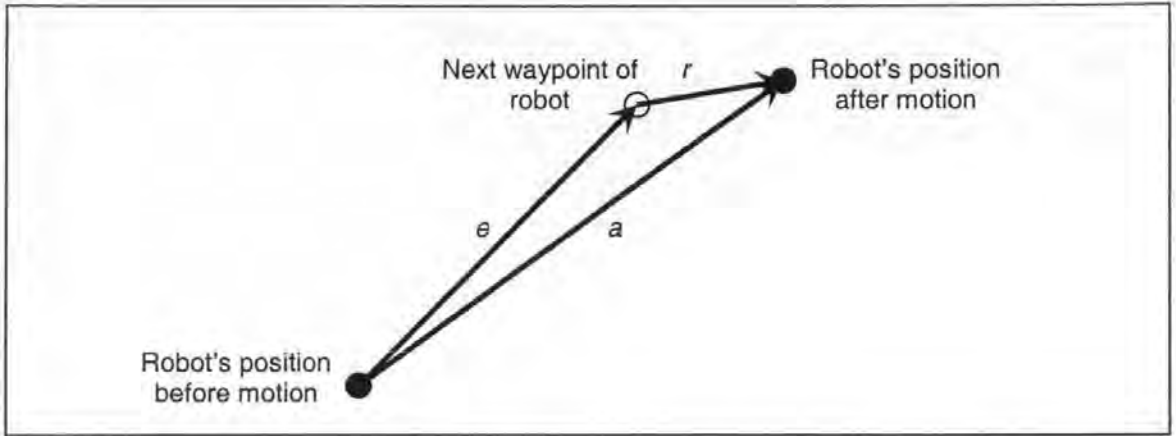


Figure 6-11: Vector diagram showing the robot's actual displacement vector " a ", which is the vector sum of the robot's intended displacement " e " and the odometric error " r ".

To be able to determine the odometric error the robot moves a short distance every time in order to maintain in its visual field information that will enable it to connect its new view with the previous one. A successful connection indicates the actual position of the robot (after motion has taken place) that is used to calculate the odometric error.

The location (relative to the robot) of important landmarks is only maintained in memory for as long as a primitive procedure call is executed. The reason why the location of landmarks is memorized is only to avoid recognizing previously found landmarks of the same type and considering them as new. For example when the user instructs the robot to "take the second turning to the left", after the first left turning is found and although the robot moves a short distance along the road, the first turning can still be visible in the map. Unless its location is "remembered" there is a danger of recognizing it again, this time as the second left turning, which would cause the robot to perform the wrong action. After the execution of the "turn" primitive procedure in the example given above, there is no need to remember the locations of the two left turnings and therefore they are cleared from memory.

When appending new visual information to the map, only the road edge information is used. This is because the road edge is geometrically consistent between top view images (because it is on the road plane) but also because the road edge allows for better accuracy in the matching of new information on previous information on the map. The position where the new road edge view matches best on the previous road edge view is used to append new road surface information on the previously seen road surface data. The complete process of how this is done is illustrated in Figure 6-12 and described below.

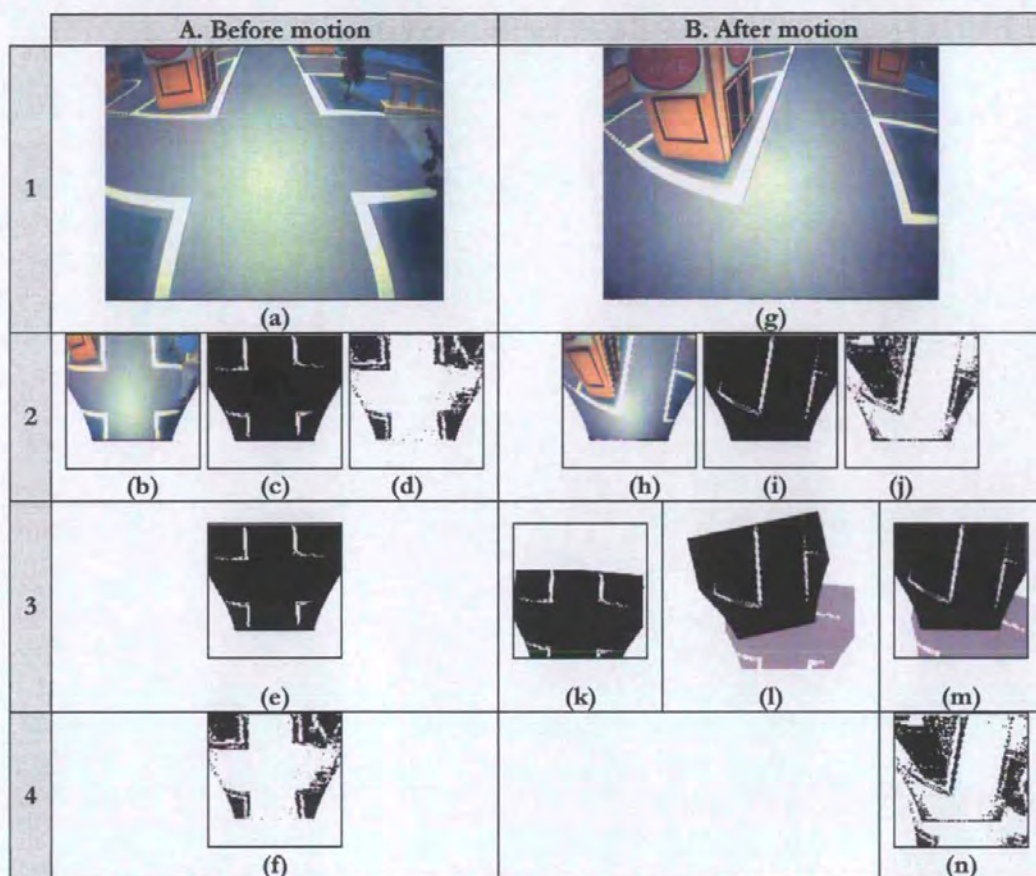


Figure 6-12: Series of figures showing how the “short-lived” map is appended with new visual information and how, as a consequence, this localizes the robot. For an explanation of how this is done see text below.

Figure 6-12 shows the state of the map before and after the robot moves (columns A and B respectively). For these two moments in time, row 1 shows the raw camera image seen by the robot, row 2 shows the corresponding top view, the road edge image and the road surface image of the top view, row 3 shows the state of the road edge map and row 4 shows the state of the road surface map. Each image in the figure is produced or manipulated in time in the order indicated by the small letter under the image.

Before the robot moves, image (a) is captured and from its top view (b) the road edge and road surface information is extracted (images (c) and (d) respectively). Assuming this is the first time the robot is going to move, the map contains no information and therefore the road edge and surface information is simply pasted on the respective edge and surface map versions (images (e) and (f) respectively). After the robot moves a new camera image is captured (g) and from it the top view is produced (h) from which the new road edge and surface information is extracted (images (i) and (j) respectively). This time the map is translated by the motion vector sent to the robot so as to reflect the estimated new position of the robot. This produces image (k). The difference between this estimated position and the actual position of the robot is the error vector r (shown in Figure 6-11). The new road edge information (i) is then matched against the shifted road edge map (k). This matching process is explained in section 6.3.1. The position where the new road edge image (i) matches best on the road edge map (k) in the example of Figure 6-12 is shown in image (l). Note that the brightness of the two images is changed for clarity. The odometric error of the robot is equal to the displacement and rotation of image (i) in order to match on image (k). This error vector is then added to image (k) to produce the actual robot's position in the road edge map (image (m)). Finally, the road edge version of the map is translated and appended with the new road surface information so as to reflect the actual position of the robot (image (j)) in the road surface map.

6.3.1 The matching of new road edge data on the “short-lived” map

Matching of the road edge image on to the road edge map is done with reference to the robot's position on the top view image. As mentioned earlier this point falls outside of the field of view of the robot camera. For the purposes of the matching operation described in this section, this point will be called here the “pivot point” of the road edge image. While searching for the best matching position, the road edge image is displaced and rotated (vector $[x, y, \theta]$) so that its pivot point scans the map image in a search window (Figure 6-13).

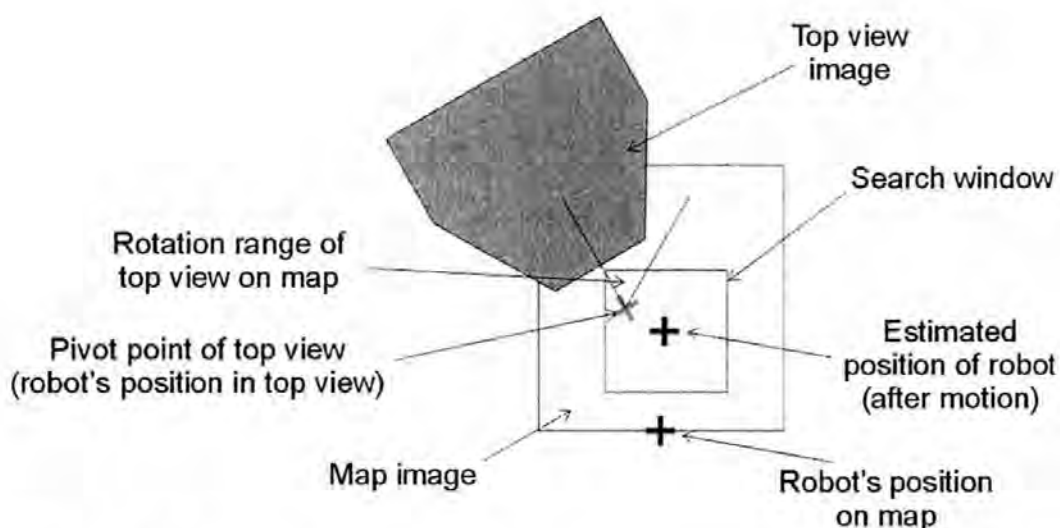


Figure 6-13: Illustration of one position of the top view image on the map while searching for the best matching position.

A matching quality Q_2 describes how the road edges of the two images overlap for each position of the road edge image on the road edge map. Q_2 is made from the sum of two ratios:

1. The score, which is the matching road edge pixels in the intersecting area of the two images divided by the number of road edge pixels in the map image and
2. The confidence factor, which is the fraction of the road edge image area falling onto areas of the map containing information.

This is formally expressed by the following equation:

$$Q_2(x, y, \theta) = \frac{\sum_{p \in N(x, y, \theta) \cap M} AND(m_p, n_p)}{\sum_{p \in N(x, y, \theta) \cap M} m_p} + \frac{\sum_{p \in N(x, y, \theta) \cap M} NOR(m'_p, n'_p)}{\sum_{p \in N} n_p} \quad (6-4)$$

$m, m', n, n' \in \{0, 1\}$

Where p is a pixel location in the overlapping area of the two images. m and n are values of pixels in the road edge map image M and road edge image N respectively. Value 0 denotes no road edge, and value 1 denotes road edge. $N(x, y, \theta)$ is the road edge image of the top view translated by (x, y) and rotated by θ . m' and n' are the information masks of the map and road edge images where 0 denotes the presence of information (mask is off) and 1 denotes no information (mask is on). The best matching position and orientation of the road edge image is the one where Q_2 is maximum. Equation 6-4 ensures that, for two configurations with equal score, the one with highest confidence has the best match quality.

To save computation time and limit the risk of matching the new top view at the wrong location, the search is limited to a small window defined on the map around the expected position of the robot. The range of rotation of the top view for each match position is limited

to a small angle θ . The size of the search window and angle θ are set taking into consideration the maximum odometric error of the robot.

To avoid false matching of the road surface image on the road surface map in cases where road surface data do not overlap, either because of bad image quality or because the robot has previously rotated at a large angle and lost all previous information from its map, a matching quality threshold Q_{2min} is used to set a minimum accepted matching between the two images. If the matching quality Q_2 is less than Q_{2min} the new road edge image is appended on the road edge map at the robot's estimated position thus neglecting the odometric error.

6.4 Road surface detection

A pre-requisite of the template matching method (described in section 6.2) is a road filtered version of the top view image. This is simply a binary image showing only road and non-road information. An example of a top view image and its corresponding road filtered version is shown in Figure 6-14.

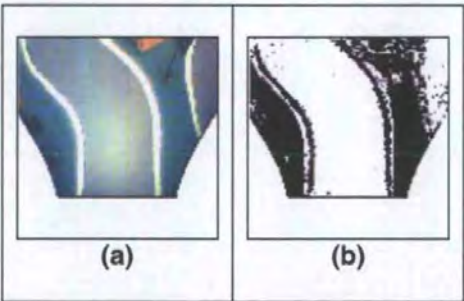


Figure 6-14: (a) An example of a top view image and (b) its corresponding road filtered version. White pixels denote road areas and black pixels denote non-road areas.

To produce Figure 6-14(b) from Figure 6-14(a) a simple colour filtering technique is used.

This is explained in the following section.

6.4.1 Colour filtering using the chromaticity vector

The colour of the road on the miniature model town is a uniform shade of grey. It does not however appear uniform or consistent in the camera image, and subsequently in the top view image, because of several reasons. These are:

1. Automatic white balance and aperture control of robot camera.
2. Casting of shadows from other objects.
3. Casting of colour shadows from objects and the robot.
4. Changing colour and intensity of natural sun light entering through the windows of the lab during the day.
5. Changing light conditions in the lab.

The white balance and aperture control of robot camera are changed automatically by the camera's circuit depending on the composition of the image. White balance is the method used by the camera to calibrate the colour values of its output image. This is done using a reference colour from the image and interpolating to find the other values. The problem arises when the reference colour changes with the composition of the image.

Aperture control is used in order to maintain constant the average intensity of the image output by the camera. This again creates a problem because changes in the intensity of a pixel are reflected in its RGB values.

The effects on the apparent road colour differences were decreased significantly by using two halogen light sources (300W each) diffused by the white ceiling of the lab on top of the miniature town model. Also, external sun light and lab lighting was blocked as much as possible using white sheet screens on the three sides of the model town (Figure 6-15).

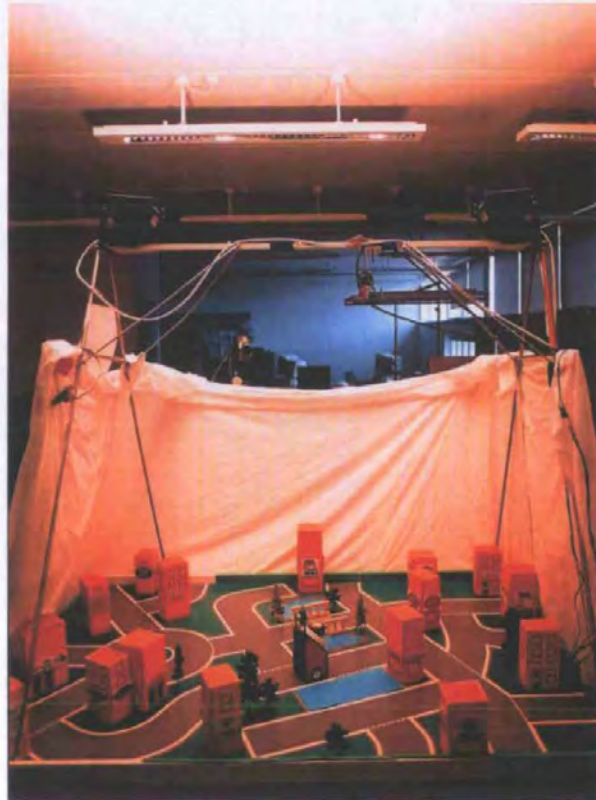


Figure 6-15: Improvements in order to improve illumination constancy in the miniature model town.

Simple chromaticity filtering was then be used to discriminate the colour of the road.

Chromaticity is an intensity invariant description of colour. It is a two-dimensional vector $[C_r, C_g]$ derived from the RGB colour coordinates by the following equation:

$$C(R, G, B) = \begin{bmatrix} C_r \\ C_g \end{bmatrix} = \begin{bmatrix} \frac{R}{B} \\ \frac{G}{B} \end{bmatrix} \quad (6-5)$$

The chromaticity (or normalized RGB) colour space was used instead of the HSI (Hue, Saturation, Intensity) space because of practical difficulties. Grey colour (i.e. $R=G=B=X$ where X is in the range $[0, 255]$) is represented in the HSI space by $H=0$, $S=0$ and $I=X$. This is confirmed by the following equations (taken from [Ballard and Brown, 1982]), which are used to convert a colour representation from the RGB space to the HSI space:

$$\begin{aligned} H &= \cos^{-1} \left(\frac{(R-G) + (R-B)}{2\sqrt{(R-G)^2 + (R-B)(G-B)}} \right) \\ S &= 1 - \frac{3}{R+G+B} (\min(R, G, B)) \\ I &= \frac{1}{3} (R+G+B) \end{aligned} \quad (6-6)$$

Although the colour of the road was grey, in practice, image noise and other reasons mentioned at the beginning of this section, caused changes to the three channels of the RGB representation so that the relationship $R=G=B$ (for the road colour) was broken in the image captured by the camera. It was therefore not possible to detect the existence of the road

colour using the criterion $H=0$, $S=0$ and $I=X$ as hue was greater than 0 when the relationship $R=G=B$ was not true.

Filtering to produce Figure 6-14(b) from Figure 6-14(a) is done by setting to white ($RGB=[255, 255, 255]$) all pixels on the top view image whose chromaticity falls within a range of values about a mean value C_{mean} . These are considered to be the road pixels. All other pixels are considered to be non-road pixels and are changed to black ($RGB = [0, 0, 0]$). C_{mean} is a value that is initially set to $[1.0, 1.0]$ (chromaticity of grey) but continuously changed, each time a new road filtered image is produced, to the average chromaticity value of the road pixels found in the image, thus bootstrapping it to any changes to the apparent colour of the road. Figure 6-16 illustrates how bootstrapping takes place between two successive image captures.

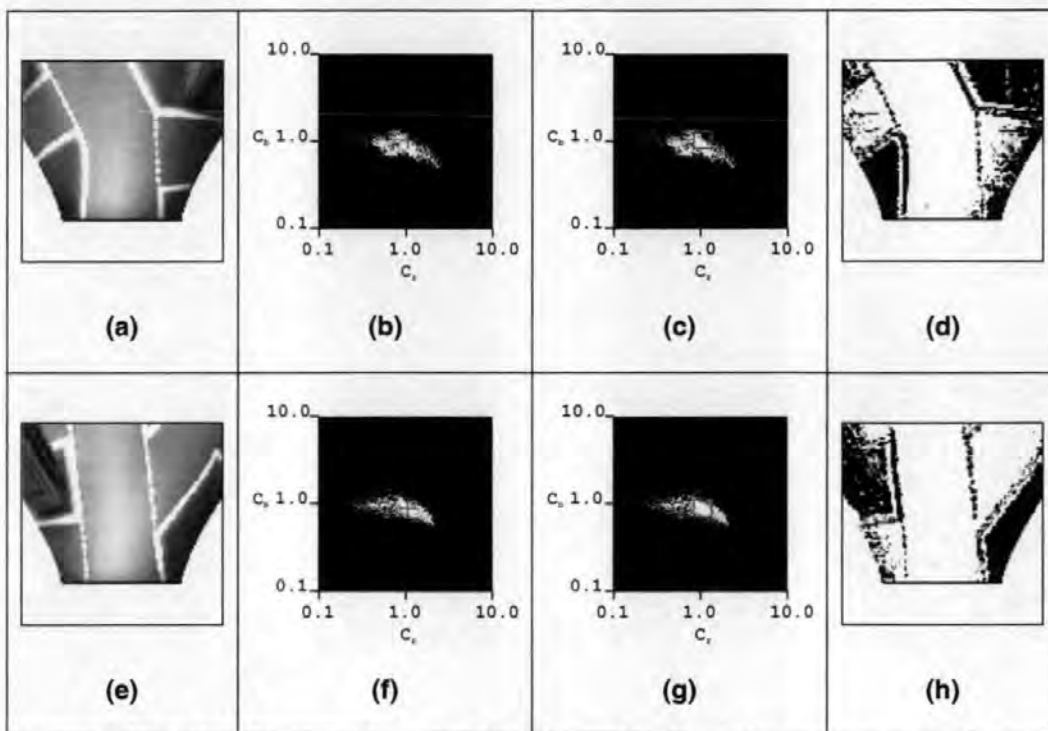


Figure 6-16: Series of images illustrating how the road surface colour is bootstrapped. For a detailed explanation of how this is done see text below.

Figure 6-16 shows the chromaticity distributions of two successive top view images I_n (Figure 6-16(a)) and I_{n+1} (Figure 6-16(b)). With reference to Figure 6-16, images (b), (c) and (f), (g) show the chromaticity distributions of I_n and I_{n+1} respectively. The cross in (c) indicates the average chromaticity $C_{mean(n-1)}$ of the road pixels found during the road surface filtering of top view image I_{n-1} (i.e. the one used before (a)). The square in the (c) indicates the chromaticity range of values, which are considered as road colour in I_n based on their proximity to $C_{mean(n-1)}$. Likewise the cross in (f) indicates the average chromaticity value $C_{mean(n)}$ of the road pixels found in top view image I_n (i.e. the white coloured pixels in (d)). In the next top view image (I_{n+1}), the range of chromaticity values considered as road surface in the image is shown by the square in (g). These values are based on their proximity to $C_{mean(n)}$.

Notice that from one top view image to the next, the chromaticity region specifying the road colour in the top view image shifts, thus following changes in the apparent road colour.

6.5 Road edge detection

The road edge information is used to accurately append new visual information on “short-lived” map (explained in section 6.3).

Road edge information is extracted from the top view image using an illumination-invariant approach similar to the one suggested in [Broggi, 1995]. This approach discriminates the white lines (road markings) along each side of the road by convolving a two-dimensional low-high-low intensity image (shown in Figure 6-17) with the top view image.

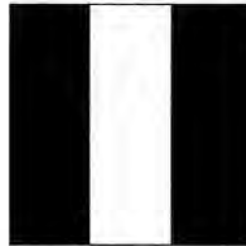


Figure 6-17: The high-low-high intensity profile kernel (magnified by a factor of 10) that is convolved with the top view image to discriminate the road edge lines.

The high intensity span of the image is equal to the width of the road markings in the top view image. The image in Figure 6-17 is convolved with the top view image at 8 different angles in the range from 0 to 180 degrees (i.e. every 22.5 degrees) in order to find the road edges at all orientations. A pixel in the top view image is considered to be a road-edge pixel if the

convolution with the kernel in Figure 6-17 is above a minimum value for at least 1 and at most 2 angles of the kernel. These are called “positive convolution results”. This ensures that only lines are detected in the top view image. An example of the resulting road edge image is shown in Figure 6-18.

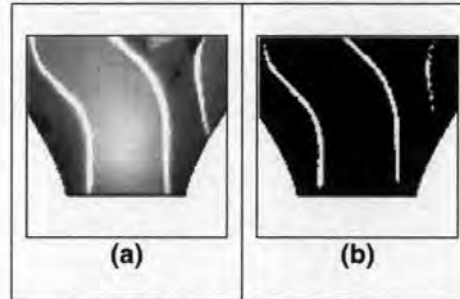


Figure 6-18: (a) An example of a top view image and (b) its corresponding road edge image.

The combination of the convolution kernel angle step and minimum/maximum positive convolution results was determined experimentally while trying to achieve minimum computational time (less than 100ms) for producing the edge image and satisfactory detection of the road edges in order to be used in later processing (see section 6.3).

6.6 The need for a state variable during robot navigation

Because primitive procedures exist in separate module files they cannot directly pass information between each other when they are executed. This posed a problem in cases found in the corpus when, while executing a route description, the course of action of one primitive depended on the state at which the robot was left in by the previous primitive. For example, by “take the first exit on your left”, users sometimes mean “take the first left exit off the roundabout” but they can also mean “take the first left exit off the road you are on”. The two

meanings require different actions to be performed as the robot uses different templates to identify a roundabout exit and a road exit (or turning). Therefore, information as to whether the robot was instructed to enter a roundabout in a previous instruction is required to decide which course of action the “turn” primitive must take in the above example.

To overcome this problem a file is used as a way of exchanging state information between primitive procedures. This file is interrogated by state-dependent primitive procedures upon entry and modified by all primitive procedures upon exit to reflect the state the robot is left in after the execution of each primitive.

6.7 Detection of non road-layout objects

This section explains how the location of non road-layout objects is found. Examples of non road-layout objects in the miniature model town are buildings, trees, the bridge etc. (see examples in Figure 6-19).

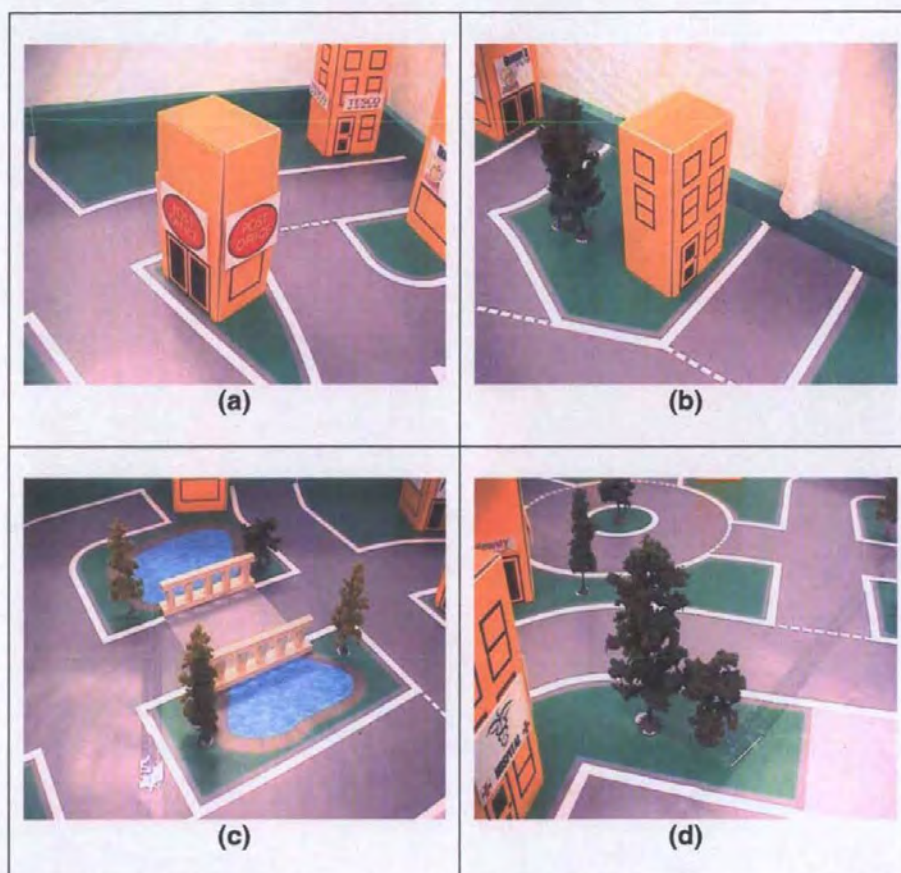


Figure 6-19: Examples of non road-layout objects mentioned in the corpus: (a) signed building, (b) unsigned building, (c) the bridge, (d) trees.

To enable testing of the primitive procedures, identification of such objects was simplified by placing a coloured strip of known dimensions (9×2 cm) next to them (Figure 6-20).

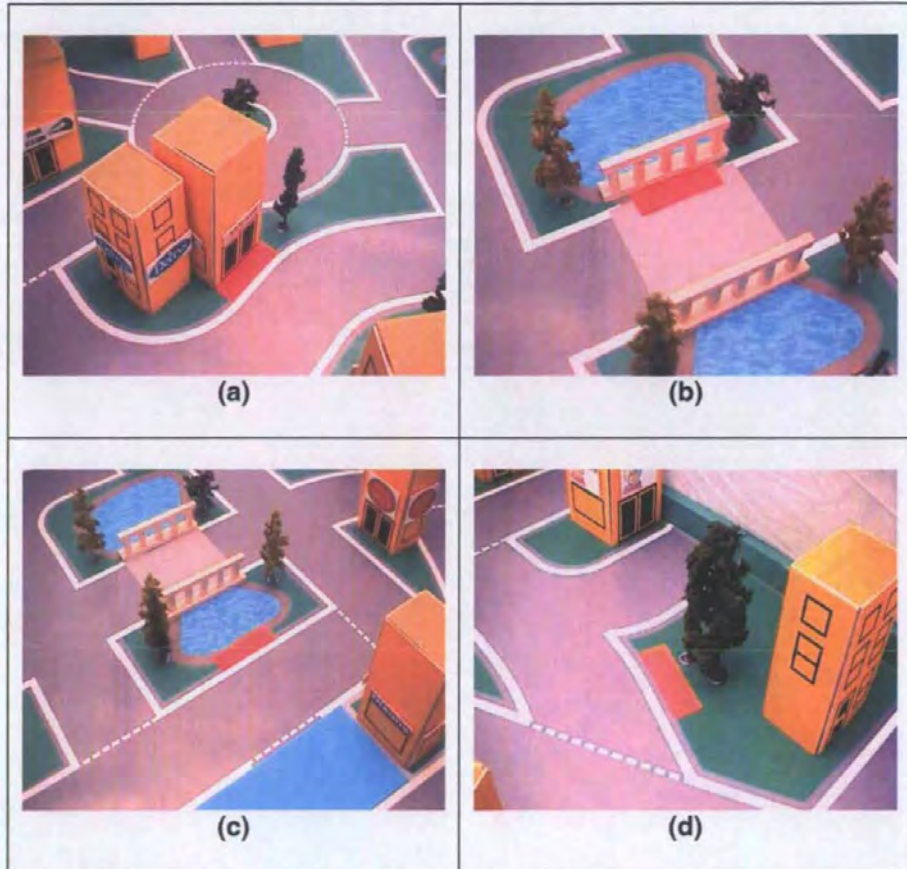


Figure 6-20: Examples of placing a coloured marker in front of objects to be able to locate them.

When a user refers to a non road-layout landmark, the target in the primitive procedure is associated with finding the landmark's colour strip instead of the landmark itself. The template matching method described in section 6.2.3 is used to locate the coloured marker. The only template used in this case is a rectangle with the dimensions of the landmark's marker on the top view image. This template is matched against the top view image filtered for the colour of the marker (Figure 6-21).

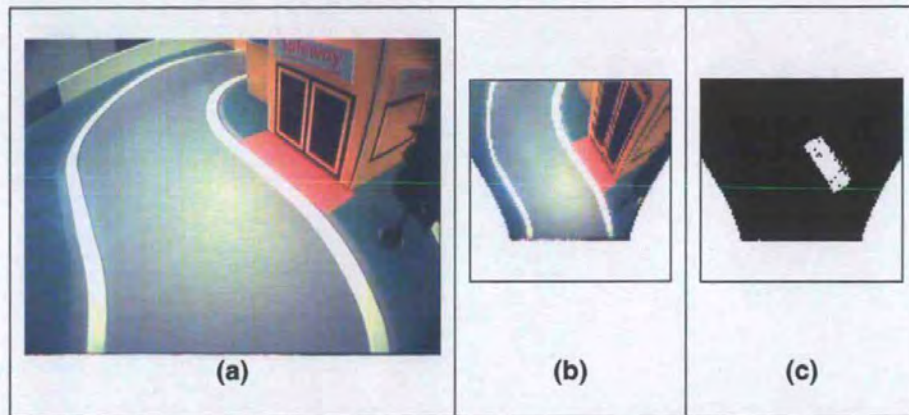


Figure 6-21: (a) An example of a camera image, (b) the corresponding top view image and (c) the top view image filtered for the colour of the marker of the landmark sought.

Recall in section 6.2.3 that in the case of road layout feature landmarks, the pivot point of the winner template is projected onto the real world to become a waypoint for the robot. In the case of non road-layout objects a different step needs to be taken, after finding the object's marker, in order to establish the waypoint of the robot. In this step a position on the road next to the landmark must be determined and used as a waypoint. This is because when users refer to non road-layout objects, as a navigational landmark in a route instruction, they are actually referring to their projected location on the road. This is a point in an area of road closest to the object as viewed by the robot when approaching the object. For example, in the utterance "follow the road to safeway" the location of "safeway" is different from the location that the robot must move to in order to execute the task correctly (Figure 6-22).

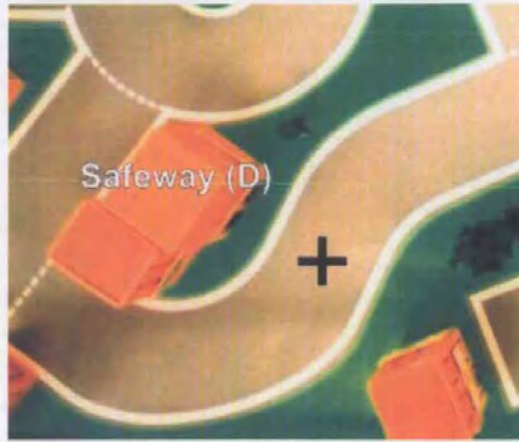


Figure 6-22: Illustration showing the location that the robot must reach to execute the instruction: “follow the road to Safeway” in comparison with the actual location of the “Safeway” building.

This is different from the case when a user says: “follow the road to the crossroad”.

To find the road point representing the reference to the landmark, the number of road pixels on the two long sides of the coloured marker is compared. The side found to contain the most road pixels is taken to be the road side and the road waypoint representing the landmark is found on that side on a line normal to the long side of the marker and at a distance equal to half the road width (Figure 6-23).

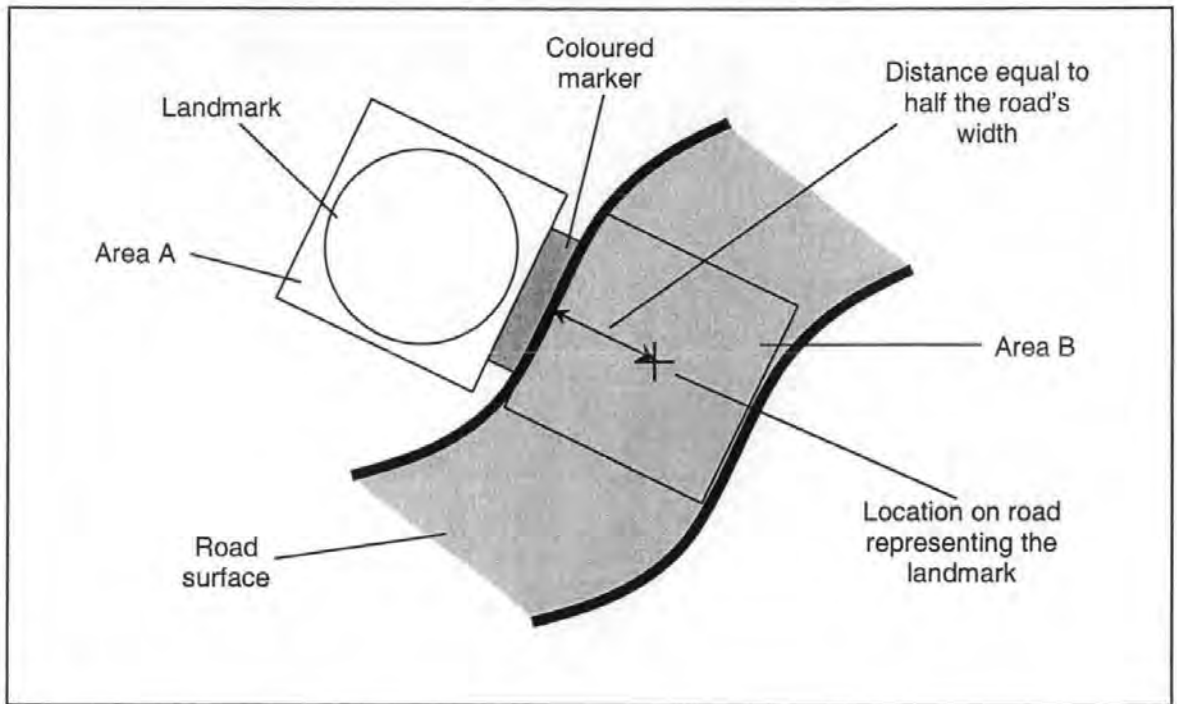


Figure 6-23: Illustration showing how the road waypoint representing a reference to a landmark is found.

The pixel regions considered on the two sides of the marker are square with side dimensions equal to the width of the road in the top view image.

6.8 Spatial references to landmarks

In almost all cases in the corpus, subjects used a robot-centred frame of reference when they mentioned other landmarks in their route descriptions. Two types of landmark references were met in the corpus:

1. References implying an instruction to the robot to move in relation to a landmark. For example: “continue forwards until you come to a junction” (u9_GC_HL), and

2. References that were used to inform the robot of a landmark's location. For example:
"you got pc world on your right" (u20_GB_EC).

The second case above is further resolved in two cases:

- 2a. The landmark mentioned is not the final destination. This type of reference is usually used to assure the robot that it is following the correct road and
- 2b. The landmark mentioned is the final destination of the robot.

In all cases, the robot needs to locate the landmark referenced by the user. Locating a landmark means establishing a road position, which will represent the landmark. This is achieved as described in sections 6.2 and 6.3 depending on the type of landmark (road layout feature or non road-layout feature).

In case 1, after locating the landmark, the robot needs to move in order to satisfy the relation between itself and the landmark. The robot's action to move is part of the target associated actions in the primitive procedure (see sections 5.1 and 6.2.4). In the example utterance: "continue forwards until you come to a junction", once the junction is found and its road location is established the robot moves to that location.

In case 2a the user only informs the robot of a landmark it will meet along the road as a confirmation that the robot is on the right track. No action is taken by the robot once the landmark is found in this case. In fact, there is no problem in completely discarding such references as users always use them to refer to landmarks along the road that the robot is

following. An exception to such references is when the user refers to the destination landmark (case 2b). In this case the robot uses the reference to move to the destination landmark even though there is no explicit instruction from the user to do so. For example in the corpus route description u1_GA_MD the destination is Safeway (D) and therefore the final utterance: “Safeways is the next building on your right hand side” is actually treated as the instruction: “follow the road to Safeways which is the next building on your right hand side”.

In Table 6-3 all the words in the corpus indicating a relation between the robot and a landmark are listed along with examples of their occurrence. An explanation of the action of the robot once the landmark is found is also given. Note that this action is only performed when the reference falls in categories 1 and 2b above.

Relational expression	Example form the corpus	Robot's action
across	"across the bridge" "across the crossroads"	The robot moves on the landmark's road location.
after	"after you pass the university" "take a left hand turn after the post office"	The robot moves on the landmark's road location.
along	"tescos a short way along that road" "you will find the hospital a short way along that road"	The robot moves on the landmark's road location.
at	"exit the roundabout at the third exit" "you arrive at the car park"	The robot moves on the landmark's road location.
before	"before reaching the university main door take the road to your right" "just before the post office on the left hand side turn left"	No action from the robot once the landmark is found.
onto	"across the crossroads onto a roundabout" "go straight onto the roundabout"	The robot moves on the landmark's road location.
over	"again turn left over the bridge" "come to another junction if you go straight over that"	The robot moves on the landmark's road location.
past	"past the university of Plymouth" "past pc world"	The robot moves on the landmark's road location.
to	"come to a junction" "follow on to the roundabout"	The robot moves on the landmark's road location.
towards	"you ll need to go forward towards the roundabout"	The robot moves on the landmark's road location.
until	"forwards er until you come to tescos" "continue forwards until you come to a junction"	The robot moves on the landmark's road location.
in front of	"it should be right in front of you" "in front of you is er a crossroads"	The robot moves on the landmark's road location.
opposite	"the car park is directly opposite"	The robot moves on the landmark's road location.
left of	"the post office is directly on your left hand side" "and then you reach boots which is on your left"	The robot moves on the landmark's road location.
right of	"pass a park on your right" "safeways should be on your right"	The robot moves on the landmark's road location.

Table 6-3: Words in the corpus that indicate a relation between the robot and a landmark along with an explanation of the robot's final location with respect to the landmark's location.

Note that when the user instructs the robot to move "after", "past", "over" or "across" a landmark the robot actually stops on the landmark's road location and not past it as suggested. This is because there is no information as to how far past the landmark it should move before it stops.

In very few cases in the corpus the relation “before” is used. One such example is in u16_GA_HC: “before reaching the university main door take the road to your right”. This is illustrated in Figure 6-24.

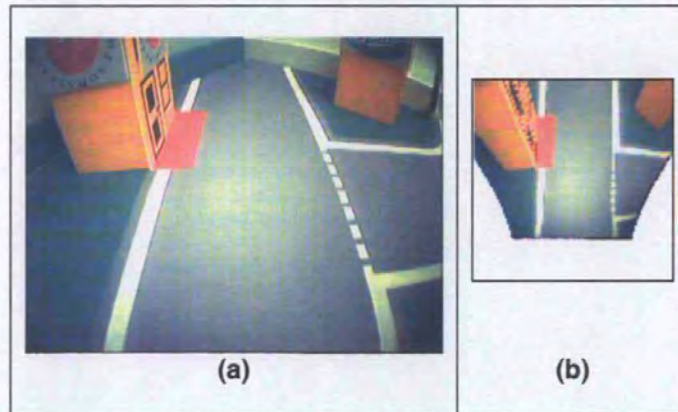


Figure 6-24: (a) Camera view and (b) the corresponding top view before the “university” when the robot follows the instruction: “before reaching the university’s main door take the road to your right”. Notice that when the “university’s door” is visible, the “road to the right” is still within the robot’s view.

The problem here is that the order in which the two landmarks (the “university’s main door” and the “road to the right”) are referenced in the route description is different to the order they will appear to the robot as it moves along the road. The robot needs to locate the further away landmark before it can start searching for the closer one. This is why, in this example for instance, as soon as the “university’s main door” is found the robot does not move any further and starts searching for the “road to the right”. In all such cases found in the corpus the two landmarks mentioned were close enough to avoid the danger of passing one while searching for the other.

6.9 Summary/Contributions

This chapter described how image template matching is used in the robot's primitive procedures in order to detect road layout features mentioned in route descriptions. The interesting feature of the method proposed here lies not in the image processing technique itself (as template matching is a well researched area) but in the fact that it is driven by the content of natural language instructions. The templates used in each search operation are derived from the natural language instruction given by the user. Furthermore, the method proposed here follows a user-centred approach in that the template images shown in Table 6-1 were created solely by studying the collected corpus of route instructions and the environment that these referred to.

A simplified method of "image mosaicing" is also described in this chapter in order to create a "short-lived" map of the robot's immediate locality. The odometric error of the robot can be determined as a result of appending new visual information on the map every time the robot moves to a new position. The odometric error is used to re-localize the robot after dead reckoning is used to estimate its position and thus re-localize the position, with reference to the robot, of any significant landmarks stored in the robot's memory. Apart from re-localizing the robot the "short-lived" map is used to compensate for the "dead-angles" of the robot.

As mentioned in section 2.4 there is an error produced in the image-sequence mosaicing process due to image noise. This problem creates a particular challenge in cases where continuous incremental mapping of cyclic environments is required because it increases with the number of images in the sequence and therefore with the distance travelled by the robot.

The map created as the robot moves in this project only displays visual information in the last 2 images captured by the robot. For this reason the error due to linking the two images is present only once in the map. As mentioned in [Unnikrishnan and Kelly, 2002a] the maximum value of this error is equal to the world distance represented by 1 pixel in the map image. In this project the maximum error is 3.33mm. This value is small when compared with the size of landmarks in the miniature town and can be safely neglected.

Chapter 7

7 The evaluation of the primitive procedures

The IBL system is comprised of several functional units. Errors can occur at different stages from the time the user speaks to the time the robot starts to follow the route instructions.

These errors fall into four main categories:

1. Speech recognition errors (caused by the Dialogue Manager).
2. Grammatical and syntactical analysis errors (caused by the Dialogue Manager).
3. Errors in the translation from the DRS (Discourse Representation Structure) to primitive procedures or previously learned procedures (caused by the Robot Manager).
4. Errors during the execution of the route instructions (caused by the primitive procedures).

This chapter is concerned with the last type of errors, in the list above, which are related to the work presented in this thesis.

To determine the performance of the individual components and eventually the complete IBL system when faced with new route instructions, the collected corpus was split in two equal sets. One to be used for the development of the system (development set) and the other for its evaluation (evaluation set). Appendix B lists the route descriptions in each of the two sets. In this chapter, the method for evaluating the primitive procedures is described.

To develop and test the primitive procedures two steps were followed:

1. Finalization of the specifications of the primitive procedures using the development set of the corpus.
2. Testing of the performance of the primitive procedures with the evaluation set of the corpus without changing their specifications.

As mentioned earlier in section 4.3.1, after the collection of the corpus and the initial specification of the primitive procedures, each route description collected was manually translated into its primitive procedure calls. An example of one such translation is given in Table 4-5.

For the purpose of developing and evaluating the primitive procedures, it was assumed that for every route description the robot would have no prior knowledge, i.e. the robot's knowledge pool would only contain the primitive procedures. As mentioned in section 4.1, during the corpus collection procedure each subject gave six route descriptions and was encouraged to refer to previously explained routes if he/she wanted. Any references to previously explained routes by the users were discarded during the development and

evaluation phases and the robot was placed at a point where the route description was explicit. An example of such a case is the route description in Table 4-5(a) and illustrated in Figure 4-3. Figure 4-3 shows that the user assumed that the robot knew the way to the roundabout because he/she explained this in an earlier route description. During the primitive procedure testing of route description u7_GC_CX the robot was placed in front of the roundabout (where the solid red line begins in Figure 4-3) and the translation in Table 4-5(b) was executed.

The aim during the development step of the primitive procedures was to execute each manually translated route description in the development set until the robot performed successfully in all cases. The success of the robot was considered as per primitive procedure call and not per successful route (whether the robot did or didn't reach the destination) although both results are presented here (sections 7.1 and 7.2). This is because route descriptions could be unsuccessful because of wrong or ambiguous route instructions given by subjects. This is further discussed in section 7.1.

For the testing phase each translated route description in the corpus evaluation set was executed. During this phase no changes were made to the specifications of the primitive procedures. Success results were recorded at route description level and at the primitive procedure level. These are presented, along with a description of their significance in the evaluation process, in sections 7.1 and 7.2 respectively.

Video sequences showing the robot executing two route descriptions (u13_GA_CL and u19_GB_EG) in the evaluation set can be found on the CD accompanying this thesis (see Appendix C).

7.1 Results per route instruction

A route description was considered to be successful if, after the execution of the associated manual translation file, the robot reached its destination. During the development and testing of the primitive procedures, the robot could fail to reach its destination because of one of two reasons:

1. Either due to a primitive procedure failing or
2. Due to a wrong or ambiguous route description given by the subject.

During the development phase of the primitive procedures, route failures due to the primitive procedures were cause for modifying the programs of the primitive procedures. However, any failures due to wrong or ambiguous descriptions could not be corrected and were executed until the point where the user's mistake or ambiguity occurred. During the evaluation phase no modifications were made to the primitive procedures. Any route failures either due to the primitive procedures or the descriptions of the users were simply recorded. The errors that occurred during this phase are described in section 7.1.2.

Table 7-1 presents the route success results for the development and evaluation sets when each route was executed from the manual translation files.

	Development set	Evaluation set
Total route descriptions	72	72
Executed route descriptions	70	71
Successful	39 (55.7%)	45 (63.4%)
Unsuccessful	31 (44.3%)	26 (36.6%)

Table 7-1: Route description success results during the development and evaluation of the primitive procedures. Note that the percentage values indicate the proportions of the executed route descriptions and not the total route descriptions.

As it will be further discussed in this section and section 7.2, failures in route descriptions are mostly due to the route descriptions being wrong or ambiguous and not because of primitive procedures failing. For this reason the two figures (44.3% for the development set and 36.6% for the evaluation set) in Table 7-1 should not be compared because the route descriptions in each of the two sets were selected at random.

Two routes in the development set and one route in the evaluation set were not executed by the robot because the corpus subjects in these cases referred to the destination as being along a previously explained route. Following the same procedure for testing as with all route descriptions, that would mean simply placing the robot at the destination without needing to execute any primitive procedure call.

7.1.1 Human performance tests

To find out how humans would perform in driving the robot while following the route instructions in the evaluation set and thus to set a performance baseline for the system, 12 subjects were invited to drive the robot. Each subject was allocated 6 route instructions at

random from the evaluation set. Subjects were allowed to listen to each route instruction as many times as they wished and they were asked to take notes for each route description. They were then asked to drive the robot to its destination for each of the six routes using only their notes. They drove the robot using a keyboard while being seated in front of a PC by looking only at the camera image of the robot. During the experiments subjects were asked to “think aloud” both when listening to the route descriptions and later, when driving the robot. This was done so that any doubts or inferences they were making could be recorded. A camcorder was used to record all the experiments to allow for a more careful assessment of the results at a later stage.

Table 7-2 presents the performance of the robot compared to that of the human subjects for the routes in the evaluation set.

	Robot	Human subjects
Total route descriptions	72	72
Executed route descriptions	71	72
Successful	45 (63.4%)	60 (83.3%)
Unsuccessful	26 (36.6%)	12 (16.7%)

Table 7-2: Route description success results for the evaluation set when executed by the robot and by the human subjects.

Table 7-3 further categorizes the 26 cases where the robot fails in the development phase and compares the corresponding performance of the human subjects in the same route descriptions.

Reason for robot's failure to reach destination	Number of robot failures in category	Corresponding failures of human subjects
New primitive procedure call	2	0
Failure of primitive procedure	2	1
Ambiguous route descriptions	5	1
Wrong route descriptions	17	8

Table 7-3: Analysis of the 26 cases where the robot fails to reach its destination in the evaluation phase and comparison with the performance of human subjects in the same routes.

As Table 7-3 shows, human subjects were more successful in driving the robot while following the route descriptions in the evaluation set. The reason behind this is that humans were able to make inferences and assumptions in order to clear ambiguities or correct mistakes in route instructions given in the corpus. This happened, in some occasions, because subjects were able to see the destination landmark before following all instructions in the route description and thus they were able to clarify any ambiguities in the last route instructions. In other cases, the layout of the road ahead would suggest what the instructor possibly meant and this would enable the human driver to make corrections “on the fly” while driving the robot, rather than strictly following the written notes he/she took during listening to the route instructions. Finally, if no visual clue was able to correct any ambiguous instructions, subjects followed one (the most likely to be correct) of two or more possible alternatives, which led to them successfully reaching the destination.

In the following section each category of failures by the robot (shown in each row of Table 7-3) is described in more detail. Route failure examples taken from each category are also illustrated and explained.

7.1.2 Error descriptions

In 2 cases in the development set users requested a new action (not previously requested in the development set) from the robot. The route description given by the user and an illustration of the route in each case are presented in Table 7-4.

(a) u5_GC_EH	(b) u6_GC_CL
“er when you arrive at the car park if you cross the car park and turn right you turn right and you will find the hospital a short way along that road in front of you”	“okay erm go to the junction with the university of plymouth opposite you the building and pc world on your left get to the roundabout with the tree in the middle take the first exit on the left erm carry straight on you ll go over a bridge erm you ll come to another junction if you go straight over that you ll have the post office on your right if you take the first right after the post office and bear round to the left as the road travels round there you ll and the queens pub will be on your left”
	

Table 7-4: Route descriptions and illustrations of the two cases where the robot fails to complete a route description in the evaluation set because of new primitive procedure calls. Note that the solid red line indicates the road described by the user in each case and the dashed red line indicates a route implied by the user.

In the first case (Table 7-4(a)) the user asked the robot to “cross the car park”. In this case the “cross” primitive procedure is called with its “object_1” parameter initialized to “car_park”.

Because this value is not among the allowed values "object_1" can take in the specific primitive, the procedure returns a "parameter value error" when called.

In the second case (Table 7-4(b)), the user asked the robot to "bear round to the left" at the y-junction. In this case no primitive procedure exists to accommodate this action because it is first met in the evaluation set.

Finding new or unmet primitive procedures in the evaluation set was expected after the functional analysis of the corpus (described in section 4.3.1). The graph of Figure 4-4 shows that on average two new primitive procedures appear between 72 descriptions (size of development set) and 144 descriptions.

In two occasions the robot fails to reach its destination because of primitive failure (Table 7-3). In the first case (illustrated in Table 7-5(a)) the subject asks the robot in succession to pass two landmarks, which are opposite each other: "go past derrys and the grand hotel". The "follow_road" primitive procedure, which is called twice, fails to find the "grand hotel" because once it is at "derrys" the "grand hotel" is not in the visual field of the robot as seen by the illustration in Table 7-5(a).

(a) u19_GB_EH	(b) u24_GB_HL
“okay you want to go to the hospital and to get there you want to do the first right down the road from where you are to go past derry s and the grand hotel and then you want to work your way round the bendy road there all the way to the end past the car park on your right until you get to a t junction again and then this time you want to turn right and then carry down on down the road and the first on the right er you come to another junction and the first on the right you want to er turn into and then go all the way down that road past the car park again from the other side and then you should see right at the end the hospital big grey building”	“move forward to the white dotted line turn left turn first exit right turn first exit right stop by queens pub left end”
	

Table 7-5: Route descriptions and illustrations of the two cases where the robot fails to complete a route description in the evaluation set because of primitive procedure failures.

The second case of primitive failure is presented in Table 7-5(b). In this case a wrong matching of the left turn template (shown in the illustration of Table 7-5(b)) causes the robot to follow the wrong road after the left turn.

In 5 route descriptions of the corpus users gave ambiguous instructions that caused the robot to fail to reach its destination (Table 7-4). It has to be noted here that the term “ambiguous” is subjective to the method of translation of the route descriptions concerned. Remember that

this translation was done manually for each route description while trying, to the best possible extent, to imitate the translation that the final system (dialogue manager and robot manager) would produce for the same route descriptions. Further discussion on this matter can be found in section 7.3.

Two such cases are presented in Table 7-6 for route descriptions u7_GC_CX and u24_GB_HW.

(a) u7_GC_CX	(b) u24_GB_HW
“from the roundabout take the first exit on the left continue straight over the crossroads continue over the bridge erm continue straight over the second crossroads the post office should be on your right”	“move forward to white dotted line turn left take er move forward to crossroads turn left move forward to roundabout go clockwise two sixty degrees two sixty degrees exit er by turning left move forward to first building on left stop”
	

Table 7-6: Route descriptions and illustrations of two (out of five) cases where the robot fails to complete a route description in the evaluation set because of ambiguities in the user’s description.

In case (a) of Table 7-6, by the “second crossroads” the user actually means the one right after the first crossroads he/she mentions. However, the second reference to crossroads is

translated to a primitive call without any consideration to any previous mention of crossroads and thus it is assumed that the user refers to a third crossroads after the “exit on the left”.

In Table 7-6(b) the instruction after the robot reaches the crossroads is to “turn left”.

Although the user meant “turn left (at the crossroads)” this is treated as “turn left (after the crossroads)” because “move forward to crossroads” and “turn left” qualify as two independent functional segments. During execution the robot passes the crossroads and then starts looking for a left turning.

Finally the robot fails to reach the destination in 17 occasions because of wrong route descriptions (Table 7-4). These are cases where the user was clearly mistaken in at least one of his/her instructions while explaining the route to the destination. Two such examples are presented in Table 7-7.

(a) u6_GC_CX	(b) u9_GC_HL
"right if you go exactly the same way towards the queens pub as before erm as you go over the bridge as you go past the t junction the post office will be there on your right"	"the queens pub erm if you er go forwards er continue forwards until you come to a junction er then er take a sort of right turn by dixon's erm carry on up there and then take a right that should take you to the queens pub okay"
	

Table 7-7: Route descriptions and illustrations of two (out of seventeen) cases where the robot fails to complete a route description in the evaluation set because of mistakes in the user's description.

In Table 7-7(a) the user mentions a t-junction after the bridge when he/she probably meant a crossroad.

In Table 7-7(b) when the user mentions the "junction" this is taken to be the right turn and not the t-junction, which was probably implied. This is because the word "junction" was used in the corpus to mean t-junctions, turnings or crossroads and thus the robot was programmed to recognize any of the three when looking for a "junction". The robot in this case fails when it turns into the first right turning but it would also have failed even if the robot turned right at the t-junction since the user instructed a turn in the wrong direction.

7.2 Results per primitive procedure call

The results presented in 7.1 are more important in the evaluation process of the complete IBL system since they provide the success rate of the primitive procedures as they appear in route descriptions.

This section shows how successful was the robot in the development and evaluation sets in the execution of the individual primitive calls of each route description. These results were more useful in the development and evaluation of the primitive procedures at an atomic level, i.e. given the right initial conditions (robot state and correct primitive call initialisation) whether the primitive procedure would execute correctly. These results do not take into account the human error and therefore primitive procedure calls, which occur at or after a user's error or ambiguity in the route description, were not executed. Table 7-8 presents these results for the development and evaluation sets.

	Development set	Evaluation set
Total primitive calls	336	344
Executed primitive calls	227	218
Successful	224 (98.7%)	214 (98.2%)
Unsuccessful	3 (1.3%)	2 (0.9%)
New primitive procedures or primitive parameter combinations	-	2 (0.9%)

Table 7-8: Primitive call success results during the development and evaluation of the primitive procedures. Note that the percentage values indicate the proportions of the executed primitive calls and not the total primitive calls.

During the development phase of the primitive procedures out of the 336 primitive calls in the development set translation files, 227 were executed. These primitive calls were continuously executed (as part of their route descriptions) while the primitive procedures were

being changed to achieve the best possible performance. At the end of the development phase, 3 primitive calls were still unsuccessful because of the same reason: the inability of the robot to detect an object located just after a curve because of the position and inclination of the robot camera. One such example is illustrated in Table 7-9.

“go to the roundabout take the third exit then take the first right and boots is on the left hand side” (u2_GC_MC)


Table 7-9: The robot fails to “see” the Boots (C) marker, at the end of the route, because when it turns the marker falls outside its field of view.

This occurs because, as mentioned previously in section 6.3, areas close to the front of the robot fall outside its field of view. This visual information can be estimated if it has been recorded previously (see section 6.3). However, in the above case the marker of the destination could not be seen when the robot took the right turn (after the roundabout) because it was hidden by the Boots building. When the robot got close to Boots the marker was already out of its field of view.

During the evaluation phase out of a total of 344 primitive calls in the associated translation files 218 were actually evaluated. Out of these 214 performed successfully. The 4 cases where primitive calls failed are explained in section 7.1 and illustrated in Table 7-4 and Table 7-5.

7.3 Discussion

In 7% (5 out of 71) of route descriptions in the evaluation set the robot failed because one or more of the instructions in these route descriptions were classed as “ambiguous”. As mentioned in section 7.1.2 this characterization is based on the criteria on which the translation from natural language route descriptions into primitive procedure calls was made. The translation was done by hand and in doing so the performance of the dialogue manager and robot manager in dealing with such ambiguous cases in the corpus was considered. One such route description (mentioned in section 7.1.2) is u7_GC_CX:

“from the roundabout take the first exit on the left continue straight over the crossroads continue over the bridge erm continue straight over the second crossroads the post office should be on your right” (u7_GC_CX).

The ambiguity arises when the user refers to a “second crossroads” when actually it is the first crossroads from where the robot will be when executing the instruction: “continue straight over the second crossroads”. The user refers to this ordering to stress that there are two crossroads in succession along the road. The system fails to recognise this because each instruction in the route description is treated independently from what was said prior to it.

Humans do not find situations like this ambiguous because they can resolve such references by comparing them to what was said earlier in the route description. Similar behaviour can be simulated by the robot by analysing the route description given by the user before executing it in order to detect and resolve such references. Alternatively the system can ask the user what he/she meant by asking him a question in the form of: "Did you mean ... or ...?".

A more detailed observation of Table 7-4 shows that the human robot-drivers were more able to correct ambiguities rather than mistakes in the descriptions of the instructors in the corpus. Also in the cases where a new primitive procedure or the wrong execution of a primitive causes the robot to fail the human subjects succeed. Note that in one case where the robot fails due to a wrong execution of a primitive procedure the human subject fails (by coincidence) because he/she omitted to record on paper important information when he/she was listening to the route description.

Human subjects failed to reach the destination in 12 route descriptions (Table 7-3) but only 10 of those (Table 7-4) are the same as the routes in which the robot failed. This means that in two cases human subjects failed when the robot succeeded. In both these cases the human robot-driver failed because of wrongly following the instructions in the route description.

The evaluation of primitive procedures on an individual primitive procedure call basis (section 7.2) shows that the code developed for the primitive procedures is successful in almost all cases. Less than 1% of primitive calls failed because the user requested a primitive action, which was not known by the system. This was expected since the graph in Figure 4-4 predicted that the system will be faced by approximately one new primitive in every 35 route

descriptions (slope of curve representing the complete corpus at 72 route descriptions). The issue of new primitive procedures arising after the system is completed is discussed in Chapter 8.

Table 7-8 shows that less than 2% of primitive procedures fail because the robot tries to find landmarks already past its field of view. The possibility of using a pan/tilt camera or simultaneous searching for more than one landmark at any one time while the robot follows the road, are two methods that could solve this problem.

Figure 7-1 summarizes the observations made in this chapter.

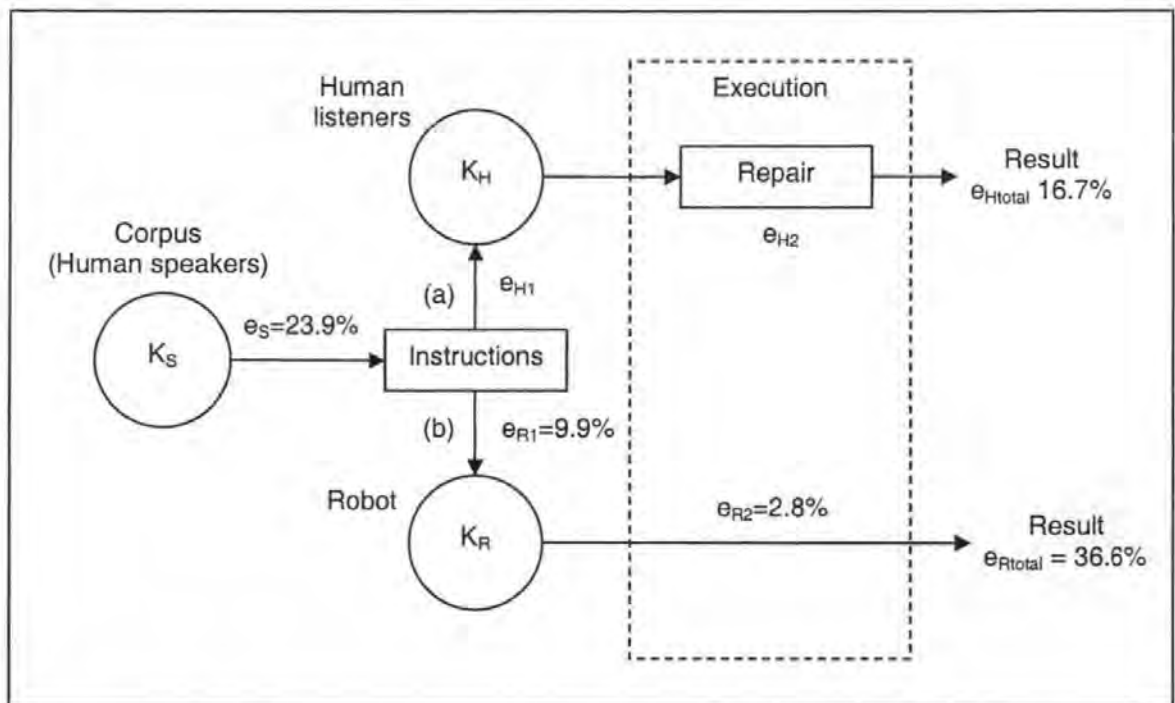


Figure 7-1: The diagram shows the occurrence of errors at different stages between the speaker giving route descriptions and the execution of these descriptions by (a) a human listener and (b) the robot. K_S , K_H and K_R represent the knowledge of the human speakers, the human listeners and the robot's respectively. The diagram shows the crucial difference between cases (a) and (b): the ability of humans to do repair during the execution of the route instructions. This accounts for the higher success rates of humans.

The figure shows that human speakers give instructions, they can make mistakes. In this project, users in the evaluation set of the corpus were wrong in 23.9% of the route descriptions. When humans listen to these instructions, they, also make mistakes in recording or memorizing the instructions. This is shown by e_{H1} in Figure 7-1. In a similar way the robot listener makes mistakes. In the robot's case (assuming perfect speech recognition and analysis) these are due to not recognizing new primitive procedures in the speaker's instructions and due to ambiguities in the instructions. In this project 9.9% of the evaluation set descriptions failed because of these errors. During execution of the route descriptions human listeners can

realize and repair mistakes of the speakers and of their own “on the fly”. Even though they can make mistakes (e_{H2}) in following the correct road sometimes their success owing to the ability to repair far exceeds that of the robot, which lacks this ability.

Therefore, the robot can increase the chances of succeeding in the execution of a route description if it uses repair during execution time. Repair and the ability of the robot to initiate dialogue with its users during the learning of new routes in order to resolve possible mistakes or ambiguities in the route descriptions are discussed as part of future work in Chapter 8.

Chapter 8

8 Conclusions and future work

8.1 Conclusions

This PhD thesis presented the work done as part of a project in Instruction Based Learning for mobile robots. The aim of this work was to determine and implement the primitive procedures that a natural language instructed robot following route descriptions would require to have in its knowledge pool.

The main contribution of this thesis to knowledge lies in the “user-centred” approach taken for determining the functional vocabulary of the robot. The aim was to create a robot that could be instructed by its human users without them needing to be previously trained on how to do so. Previous work in the field of instructable robots required the users of the robots to learn precise lexical and functional vocabularies with which to instruct the robots. These vocabularies were predetermined by the creators of the robots thus following a “robot-centred” approach. Although such approaches have succeeded in creating a faster and simpler communication method between users and robots, in comparison with formal programming methods, they nevertheless still constrained the users to a great level of formality and precision. Furthermore, the amount of training that the users would require prior to using the

robot would increase in proportion to the amount and complexity of the tasks that the robot would be required to perform. This of course would defy the purpose of creating a natural language instructed robot to be used by computer language naïve users.

The aim of the Instruction Based Learning project was to create a robot that would be able to accept instructions from its users as a human would. Such a robot would need to be able to deal with the imprecision of the spoken natural language medium. Furthermore, such a robot would need to be able to accept variations in spoken instructions that would result in similar actions thus being able to deal with the versatility of natural language.

In order to determine the nature of spoken natural language route instructions following a “user-centred” approach a corpus of route instructions was collected in the beginning of the project from 24 different human subjects. The subjects were asked to give route instructions to the robot as they would to a fellow human ensuring, in this way, that the instructors would produce unconstrained natural language utterances. The implementation of the Instruction Based Learning system was solely based on the results of the analysis of the corpus collected.

The study of the corpus natural language instructions exposed three main problems that are important for the design of Instruction Based Learning robots. These are:

1. The under-specification of natural language, which is a known problem.
2. The probability of new primitive functions arising in route instructions after the development of the system. This is known for words but it has been seen here to appear for primitive functions.

3. The cases when users make partial use of previously learned procedures when explaining new procedures. This problem has never been documented previously.

This thesis focused on the natural language under-specification problem. Cases 2 and 3 above are discussed in section 8.2 as part of future work.

Two methods are proposed in this thesis for determining the missing information in natural language route instructions. This information is vital for the robot in order to be able to execute the route instructions successfully. The methods proposed can determine the missing information during the learning of new procedures and during their execution.

During learning time the missing information is determined by imitating the commonsense approach of human listeners in order to achieve the same purpose. What human instructors consider as commonsense and therefore omit in their spoken route instructions is determined by studying examples of such occurrences in the corpus collected in this project. Necessary parameters in primitive procedures are then allowed to take default values whenever these values are not explicitly provided in the natural language instructions.

During the execution of primitive procedures missing information, such as the precise location and orientation of landmarks mentioned in the route instructions, is determined using the method of image template matching. In this thesis the method focuses on the determination of the location of road-layout features, which are mentioned in route instructions. The significant contribution in this method lies in the fact that it is driven by the

natural language instructions both in the design of the templates used and in their selection during the execution of a primitive procedure.

The primitive procedures developed during the work presented in this thesis were evaluated both on an individual primitive procedure call basis and also as part of route instructions. The evaluation of individual primitive procedure calls showed very good results (more than 98% success). This shows that the natural language under-specification problem can be solved with the methods proposed in this thesis.

A novel method was followed in order to test the primitive procedures as part of complete route descriptions. During this method, human subjects were invited to drive the robot following route descriptions while being provided with the same visual information as the robot. Their performance was later analysed and compared with the robot's performance after executing the same route descriptions. This evaluation method allowed a direct comparison between the human listener and the robot as far as the execution of route descriptions. The results of this comparison provided an important difference between the humans and the robot in this context: humans can repair errors (made by the instructor or themselves) during the execution of route descriptions by using past experience and reasoning. This is an important issue that will need to be taken into consideration in the future in order to improve the performance of the current system. Some ideas on possible future work on this matter are presented in the following section.

A software controller is proposed in this thesis in order to reliably and accurately control the wheel speeds of the robot used in this project. The controller described comprises of one PI

(Proportional and Integral) speed controller for each wheel of the robot and a PID (Proportional, Integral and Differential) controller to control the differential speed of both wheels during each robot manoeuvre. The design and implementation of this software control scheme provided an alternative to an expensive and more time-consuming hardware solution in order to achieve the same purpose.

8.2 Future work

During the evaluation of the primitive procedures, when the success of route descriptions was considered, results showed that the robot managed to successfully navigate to its destination in 63.4% of the cases. Less than 3% of these failures are due to primitive procedures failing. The remaining failures were due to the following reasons:

1. Errors in route descriptions.
2. Ambiguities in route descriptions.
3. Failures due to the users requesting primitive actions, which did not exist in the robot's knowledge base (see point 2 in previous section).

In order to improve the performance of the robot, future work in instruction based learning needs to be focused on the ability of the robot to start a dialogue with the user during the learning of a new procedure in order to resolve ambiguities or mistakes in the user's instructions. Initiating a dialogue with the user is a complex task because a question to the user needs to be formulated properly by the system in order to address the problem and at the same time the system should be able to handle the response from the user.

A mistake in a user's instruction can only be detected by the robot if it causes an error in the primitive call or if the instruction cannot be executed because of an incompatibility between the previous or the next instruction. In these cases the error will be detected by the prediction function of the primitive procedure corresponding to the action suggested by the user. A possible solution could be to formulate questions for each possible error that the prediction function can return in order to prompt the user to rectify his/her mistake.

Ambiguities in the user's instructions can be detected by studying the corpus for cases of similar instructions with multiple meanings. In such cases the user can be given a choice between the different meanings in the form of a question. Two examples of ambiguous instructions met in this project are:

1. When users order landmarks in succession to previously mentioned landmarks of the same type for example: "go to the crossroads, ..., at the second crossroads..." when it is ambiguous whether the last crossroads is the second or the third from the beginning of the example and
2. When users say, for example: "go to the post-office" when it is ambiguous whether the post-office is on the road ahead or whether they assume that the robot knows the route to the post-office.

Both cases can be resolved by giving a choice to the user in the form of a question: "Did you mean ... or ...?".

In Chapter 7 it was concluded that the human ability to repair mistakes and ambiguities in route descriptions, while executing them, accounted for the difference in execution success between humans and the robot. The ability to repair can be incorporated in primitive procedures in some cases such as for example when the layout of the road is used as a clue to indicate what the instructor meant in ambiguous situations. An example of such a case is when the user says: “turn left” when he/she actually means “take the first exit off the roundabout” in the part of a route illustrated in Figure 8-1.



Figure 8-1: A case when the user says “turn left” when he/she actually means “take the first exit off the roundabout”.

In such a case a low matching quality of the left turning template (Table 6-1(c)) or the straight road template (Table 6-1(a)) at the entrance of the roundabout could make the robot check whether the user meant “left at the roundabout” by matching the roundabout-entry template (Table 6-1(i)). If the roundabout-entry template is successful the “turn” primitive call can be changed to an “exit_roundabout” primitive call.

It was found in Chapter 4 that the robot's task vocabulary can never be complete in practice. New primitive procedures are likely to appear in the user's route instructions after the completion of the system regardless of the size of the corpus used to develop the system. When a user instructs a new primitive action there are two possibilities for the speech recognition system:

1. The system will misrecognize the command for another action.
2. The system will reject the command because of low speech recognition confidence.

In the first case, it is not possible for the system to realize that an error has taken place and this will most likely result in the execution of the wrong action by the robot. In the second case the system can ask the user a question such as: "did you mean ... or is this a new action?" an answer to which can cause the system start learning the new action. However, this only solves half the problem of the new primitive procedures because in order to describe a primitive action the user will need to refer to low-level robot procedures that are not accessible to him via natural language (see section 5.2). Therefore, the problem of new procedures appearing after the IBL system is completed is a challenging one to which a solution is still being discussed.

Also in Chapter 4 it was found that some users referred partially to previously learned routes when explaining new routes. One such example is:

“okay you ll need to pass the train station again as you did going to the post office and you ll see the university as you go onto the roundabout” (u4_GC_EW)

Were the previously explained route was from the Grand Hotel to the post-office and the new route, being explained, is from the Grand Hotel to the university. Two possible solutions to the partial re-use of previously learned routes were discussed during this project. These are:

1. To solve the problem during the learning of the new route by requesting an explicit route description from the user for the part of the route he/she is referring to. In the example above the system could respond to the user by saying: “Please explain the route to the roundabout.”
2. To solve the problem during execution time by starting the execution of the previously learned route and at the same time searching for the landmark mentioned by the user where the diversion/termination is to occur (i.e. using concurrent processing). When the diversion/termination point is found, execution of the recalled route should be terminated. In the example given above the system will start executing the known route from the Grand Hotel to the post-office while concurrently searching for the roundabout. When the roundabout is found execution of the known route is terminated.

The first solution suggested solves the problem at the expense of the user since he/she would have to re-explain part of a previously described route. Implementation of the second solution is transparent to the user but may require significant changes in the structure of the primitive procedures in order to be able to achieve the concurrency described above.

8.3 Final statement

One of the most important aspects in the creation of human helper robots, which will be useful in environments other than just the industrial floor and which can be used by people not necessarily possessing any programming or engineering knowledge, is their ability to communicate with humans. This issue is as important as the functionality of the robot since a robot that can do a complex task is not useful if its user cannot instruct it to do so!

Humans prefer the medium of spoken natural language more than any other (writing, signalling etc) in order to convey information to their fellow humans. This is because speech is more efficient, fast and requires less effort. It is therefore likely that in the future humans will instruct their robots using spoken natural language.

When it comes to a robot, understanding spoken natural language is complex task. This is because spoken natural language has no formal structure and it appears to be incomplete and ambiguous. A human speaker assumes that the human listener will be able to resolve this complexity using “commonsense”. Furthermore, the listener is expected to be able to engage in a dialogue with the speaker in order to resolve any remaining ambiguities. These are issues that need to be addressed in the case of a robot listener.

In this thesis a “user-centred” method is proposed for creating the primitive procedures of a robot following route descriptions. It has been shown how the under-specification problem of natural language route instructions can be solved.

It is hoped that the work presented here can provide a start towards solving the grater problem of unconstrained natural language dialogue between humans and their helper robots.

Appendices

Appendix A

In this appendix the specifications of each primitive procedure are presented in separate pages.

Primitive procedure:

CROSS

Description:

Instructs the robot to cross the road to an object (usually the car park) ahead or to just cross to the opposite road at a crossroads for example.

Parameters:

Parameter name	Description	Possible values	Default value
object_1	The object to cross.	'car_park', 'road'	None
relation_1	Preposition.	None, 'to'	None
object_2	The object to cross the road to.	None, 'car_park'	None

Parameter combinations:

Combination	Example
object_1	"cross the road"
object_1, relation_1, object_2	"across the road to the parking lot"

Primitive procedure:

enter_roundabout

Description:

Instructs the robot to enter the roundabout in a specific direction.

Parameters:

Parameter name	Description	Possible values	Default value
direction_1	Direction in which to follow the roundabout.	None, 'clockwise'	None
relation_1	Direction to turn to right after entering the roundabout.	None, 'left_of', 'right_of'	None
object_1	Object to which relation_1 refers to.	None, 'self'	None

Parameter combinations:

Combination	Example
direction_1	"start going around the roundabout"
	"go over the roundabout"
relation_1, object_1	"turn right at the roundabout"

Primitive procedure:

exit_object

Description:

Instructs the robot to exit from a place. Usually used for exiting the car park.

Parameters:

Parameter name	Description	Possible values	Default value
object_1	Object to exit from.	'car_park'	None

Parameter combinations:

Combination	Example
object_1	"if you're in the parking space go out <hesitation>erm</hesitation> basically straight on into the following road"

Primitive procedure:

exit_roundabout

Description:

Instructs the robot to take an exit off the roundabout. If the robot has not entered the roundabout then it follows the road until it meets the roundabout, enters it turning left (clockwise around the roundabout) and takes the designated exit.

Parameters:

Parameter name	Description	Possible values	Default value
ordinal_1	Order of exit to take.	None, 'first', 'second', 'third'	None
relation_1	Preposition associated with object_1.	None, 'at', 'left_of', 'past', 'right_of'	None
object_1	Object with reference to which the roundabout exit is specified.	None, 'self', 'train_station', 'university'	None

Parameter combinations:

Combination	Example
object_1, relation_1	"take the exit with a train station on your right"
object_1, ordinal_1, relation_1	"take the second first after the university"
ordinal_1	"take the third exit"
	"take your first exit on the right at the roundabout"
	"take the second left off the roundabout"

Primitive procedure:

follow_road

Description:

Instructs the robot to move forward on the road until a specified location.

Parameters:

Parameter name	Description	Possible values	Default value
relation_1	Preposition associated with object_1.	None, 'across', 'after', 'along', 'around', 'at', 'in_middle_of', 'onto', 'over', 'past', 'to', 'towards', 'until'	'to'
ordinal_1	Order of object_1 along the road.	None, 'first', 'second', 'third'	'first'
object_1	Object with reference to which the location the robot should stop (following the road) is specified.	'bend', 'bridge', 'broken_line', 'building', 'car_park', 'corner', 'crossroads', 'derrys', 'dixons', 'dotted_line', 'end_of_road', 'exit', 'grand_hotel', 'house', 'junction', 'lake', 'museum', 'pc_world', 'pizza_hut', 'pond', 'post_office', 'queens_pub', 'right_turning', 'roundabout', 'safeway', 't_junction', 'tesco', 'train_station', 'tree', 'turning', 'university', 'water'	None
relation_2	Preposition associated with object_2.	None, 'after', 'at', 'by', 'in_front_of', 'left_of', 'opposite', 'right_of', 'to'	None
object_2	Object with reference to which the location of object_1 is specified.	None, 'bridge', 'post_office', 'roundabout', 'safeway', 'self'	None

Parameter combinations:

Combination	Example
object_1, ordinal_1, relation_1	"go straight down that road and to the next junction" "come to a junction" "go over the bridge" "just past pizza hut"
object_1, object_2, ordinal_1, relation_1, relation_2	"walk straight ahead past the post-office which is on your right-hand side"

Primitive procedure:

location

Description:

Specifies the location of an object. If the object is the destination the robot moves to it otherwise the robot stops as soon as it locates the object.

Parameters:

Parameter name	Description	Possible values	Default value
object_1	Object whose location is described by this primitive.	'boots', 'bridge', 'building', 'car_park', 'crossroads', 'derrys', 'grand_hotel', 'hospital', 'junction', 'lake', 'library', 'museum', 'pc_world', 'pizza_hut', 'pond', 'post_office', 'queens_pub', 'safeway', 'safeways', 'tesco', 'tree', 'university'	None
relation_1	Preposition associated with object_2	None, 'across', 'after', 'along', 'at', 'before', 'end_of', 'in_front_of', 'left_of', 'on', 'onto', 'opposite', 'past', 'right_of'	'along'
ordinal_1	Order of object_2 along the road.	None, 'first', 'second'	None
object_2	Object with reference to which the location of object_1 is specified.	None, 'bend', 'bridge', 'building', 'car_park', 'corner', 'derrys', 'end_of_road', 'exit', 'left_turning', 'right_turning', 'road', 'roundabout', 'safeway', 'self', 'street', 't_junction', 'tesco', 'train_station', 'turning', 'university'	'road'
destination_1	The destination of the route description of which this primitive is part.	'boots', 'car_park', 'grand_hotel', 'hospital', 'library', 'museum', 'post_office', 'queens_pub', 'safeway', 'safeways', 'tesco', 'university'	None

Parameter combinations:

Combination	Example
object_1, relation_1, object_2, destination_1	"boots is on your left" "safeway is on your right-hand side" "the hospital will be right in front of you" "boots is on the left-hand side of the street" "the hospital is at the end of the street" "the post-office is just across the street" "pc world is on the corner" "the university is opposite the train station"
object_1, relation_1, ordinal_1, object_2, destination_1	"the museum is after two further turnings on the right and it is on the right-hand side"

Primitive procedure:

bear

Description:

Instructs the robot to take one of the two directions at a y-junction.

Parameters:

Parameter name	Description	Possible values	Default value
relation_1	Direction to take at the y-junction.	'left_of'	None
object_1	Object to which relation_1 refers to.	'self'	None

Parameter combinations:

Combination	Example
relation_1, object_1	"bear round to the left"

Primitive procedure:

park

Description:

Instructs the robot to park either on/by a specific location.

Parameters:

Parameter name	Description	Possible values	Default value
relation_1	Preposition specifying the position to park with reference to object_1.	'in_centre_of'	None
object_1	Object associated with relation_1.	'quadrangle'	None

Parameter combinations:

Combination	Example
relation_1, object_1	"park in the center of the <hesitation>er</hesitation> quadrangle that's in front of you"

Primitive procedure:

rotate

Description:

Instructs the robot to rotate about itself.

Parameters:

Parameter name	Description	Possible values	Default value
relation_1	Preposition specifying the extent of rotation with reference to object_1.	'around'	None
object_1	Object associated with relation_1.	None, 'self'	'self'

Parameter combinations:

Combination	Example
relation_1, object_1	"you need to turn round" "assuming you turned round"

Primitive procedure:

take_road

Description:

Instructs the robot to take a road in view. Usually used when the robot is at an intersection and needs to get on an opposite road.

Parameters:

Parameter name	Description	Possible values	Default value
relation_1	Preposition specifying the road to take in relation to object_1.	'opposite', 'right_of'	None
object_1	Object associated with relation_1.	'self'	None

Parameter combinations:

Combination	Example
relation_1, object_1	"take the road that's opposite you"

Primitive procedure:

turn

Description:

Instructs the robot to take a turn from the current road.

Parameters:

Parameter name	Description	Possible values	Default value
ordinal_1	Order of the turning to take along the road.	None, 'first', 'second', 'third'	'first'
relation_1	Preposition specifying the direction of the turning in relation to object_1.	'left_of', 'right_of'	None
object_1	Object associated with relation_1.	'self'	None
relation_2	Preposition specifying the position of the turning in relation to object_2.	None, 'after', 'around', 'at', 'before', 'by', 'into', 'past'	None
object_2	Object associated with relation_2.	None, 'bridge', 'car_park', 'crossroads', 'derrys', 'dixons', 'junction', 'pc_world', 'post_office', 'roundabout', 't_junction', 'university'	None

Parameter combinations:

Combination	Example
ordinal_1, relation_1, object_1	"you take a left" "turn right" "take the first right" "take your first right"
ordinal_1, relation_1, object_1, relation_2, object_2	"take a sort of right turn by dixon's" "take a left-hand turn after the post-office" "take a left turn at the junction" "turn left at the post-office" "take a right at the crossroads" "take the second right after you reach the post-office" "at the university take the first right" "carry along until you find the car park and turn right into it"

Primitive procedure:

go_until

Description:

Instructs the robot to use part of a previously explained route.

Parameters:

Parameter name	Description	Possible values	Default value
object_1	Final destination of known route.	'grand_hotel', 'museum', 'post_office', 'safeway', 'university'	None
relation_1	Preposition specifying the diversion point in relation to object_2.	'at', 'before', 'over', 'past'	None
object_2	Object associated with relation_1.	'bridge', 'derrys', 'grand_hotel', 'train_station', 'university'	None

Parameter combinations:

Combination	Example
object_1, relation_1, object_2	"okay you ll need to pass the train station again as you did going to the post office and you ll see the university as you go onto the roundabout" "erm head as though you re going towards the post office so you go over the bridge but instead of carrying straight on take a right" "okay from the crossroads continue on straight ahead take the next right" "okay head towards the grand hotel but just before you get there the safeway is on your right hand side" "recalling our previous destination was the grand hotel and we passed safeways en route just before derrys" "right if you go exactly the same way towards the queens pub as before erm as you go over the bridge as you go past the t junction the post office will be there on your right"

Primitive procedure:

go

Description:

Instructs the robot to execute a previously explained route.

Parameters:

Parameter name	Description	Possible values	Default value
object_1	Destination of previously explained route.	'bridge', 'car_park', 'grand_hotel', 'post_office', 'queens_pub', 'roundabout', 'safeway', 'tesco', 'university'	None

Parameter combinations:

Combination	Example
object_1	"go to the post office" "go to the roundabout" "go to the roundabout mentioned previously" "go to boots"

Appendix B

This appendix lists the route descriptions in the development set and the evaluation set.

Development set route descriptions

u1_GA_MD	u6_GC_CM	u12_GA_EP	u16_GA_HW	u20_GB_EW
u1_GA_MW	u6_GC_CP	u12_GA_EW	u17_GB_MY	u21_GB_CD
u1_GA_MY	u7_GC_CD	u13_GA_CD	u17_GB_ME	u21_GB_CM
u2_GC_MC	u7_GC_CE	u13_GA_CE	u18_GB_MC	u21_GB_CP
u2_GC_MW	u7_GC_CL	u13_GA_CL	u18_GB_MD	u21_GB_CX
u2_GC_MY	u7_GC_CM	u13_GA_CM	u18_GB_ME	u22_GB_CD
u2_GC_MZ	u7_GC_CP	u14_GA_CD	u18_GB_MW	u22_GB_CE
u3_GC_ME	u8_GC_HD	u14_GA_CE	u18_GB_MY	u22_GB_CL
u3_GC_MY	u8_GC_HL	u14_GA_CL	u19_GB_EC	u23_GB_HC
u4_GC_EP	u9_GC_HW	u14_GA_CM	u19_GB_EG	u23_GB_HD
u4_GC_EW	u10_GA_MD	u14_GA_CP	u19_GB_EP	u23_GB_HG
u5_GC_EG	u10_GA_ME	u15_GA_HD	u20_GB_EC	u24_GB_HC
u5_GC_EPbis	u11_GA_EH	u15_GA_HE	u20_GB_EG	
u5_GC_EW	u12_GA_EG	u15_GA_HG	u20_GB_EH	
u5_GC_EX	u12_GA_EH	u16_GA_HG	u20_GB_EP	

Evaluation set route descriptions

u1_GA_MC	u6_GC_CD	u10_GA_MW	u15_GA_HW	u21_GB_CL
u1_GA_ME	u6_GC_CE	u10_GA_MY	u16_GA_HC	u22_GB_CM
u1_GA_MZ	u6_GC_CL	u10_GA_MZ	u16_GA_HD	u22_GB_CP
u2_GC_MD	u6_GC_CX	u11_GA_EC	u16_GA_HEbis	u22_GB_CX
u2_GC_ME	u7_GC_CX	u11_GA_EG	u16_GA_HL	u23_GB_HE
u3_GC_MC	u8_GC_HC	u11_GA_EP	u17_GB_MC	u23_GB_HL
u3_GC_MD	u8_GC_HE	u11_GA_EW	u17_GB_MD	u23_GB_HW
u3_GC_MW	u8_GC_HG	u11_GA_EX	u17_GB_MW	u24_GB_HD
u3_GC_MZ	u8_GC_HW	u12_GA_EC	u17_GB_MZ	u24_GB_HE
u4_GC_EC	u9_GC_HC	u12_GA_EX	u18_GB_MZ	u24_GB_HG
u4_GC_EG	u9_GC_HD	u13_GA_CP	u19_GB_EH	u24_GB_HL
u4_GC_EH	u9_GC_HE	u13_GA_CX	u19_GB_EW	u24_GB_HW
u4_GC_EX	u9_GC_HG	u14_GA_CX	u19_GB_EX	
u5_GC_EC	u9_GC_HL	u15_GA_HC	u20_GB_EX	
u5_GC_EH	u10_GA_MC	u15_GA_HL	u21_GB_CE	

Appendix C

This appendix presents the contents of the CD accompanying this thesis.

Note that each corpus route description file is given a name which contains the subject's number, the subject's group (see section 4.1) and the starting location and destination in the miniature model town. For example:

u4_GC_EH

refers to the route description of subject 4 who is in group C. The subject was asked to describe the route from the Grand Hotel (designated with the letter E) to the Hospital (designated with the letter H). The designations of each landmark in the miniature model town are illustrated in the map image file "miniature_model_town_map.jpg" which can be found on the CD accompanying this thesis.

The following paragraphs explain the contents of each file/directory in the accompanying CD.

Directory "corpus_sound_recordings":

Contains the sound files of the route descriptions collected during the corpus. The file format of all sound files is "wav".

File "corpus_transcriptions.txt":

A single file containing all transcriptions of the corpus. The format of the file is DOS "txt".

Directory “corpus_translations”:

Contains the manual translation files of the corpus route descriptions (sets of initialized primitive procedure calls corresponding to each route description in the corpus). The format of the files is “py” (python source files). They can be viewed in a UNIX-based text editor or MS Windows Word.

Directory “video_examples”:

Contains 4 “mpg” (MPEG) video files:

- **“take_the_second_left_1.mpg”:** Shows the robot executing the route instruction: “take the second left”.
- **“take_the_second_left_2.mpg”:** Shows the computer screen when executing the same route instruction as above.
- **“u13_GA_CL.mpg”:** Shows the execution of the corpus route description with the same name.
- **“u19_GB_EG.mpg”:** Shows the execution of the corpus route description with the same name.

Note 1: The video files showing the robot navigating in the miniature model town had several 5-second intervals removed from them while the robot was stationary. This time delay was deliberately inserted, after the robot executes a motion command, in order to allow the camera image to settle. The upset in the video image while the robot moves occurs because of the large current drawn by the robot motors.

Note 2: The video sequences shown by “template_matching_2.mpg” and “template_matching_2.mpg” were recorded at different times so the actual position of the robot at any moment does not necessarily correspond in the two runs.

File “miniature_model_town_map.jpg”:

Annotated map image of the miniature model town. The file format is JPEG.

Directory “code”:

The explicit program code developed during the work described in this thesis. The directory contains among other files the primitive procedure modules. A brief explanation of the main files is given below:

- “bear.py”, “cross.py”, “enter_roundabout.py”, “exit_object.py”, “exit_roundabout.py”, “follow_road.py”, “location.py”, “park.py”, “rotate.py”, “take_road.py”, “turn.py”: Primitive procedure modules (python source files).
- “*.pyc”: Python byte-compiled files.
- “various.py”: Contains flags and values used throughout the primitive procedures.
- “average_chromaticity_data”: Stores the chromaticity vector of the road surface colour.
- “calculate_transform_data.py”: Produces the “transformation look-up matrix” (see section 6.1).
- “capture.py”: Saves the robot camera image into a file.
- “capture_and_process.py”: Same as “capture.py” but in addition it executes the pre-processing and road edge/road surface filtering steps (see sections 6.1, 6.4 and 6.5).

- **“*.c” and “*.h”**: C source and header files containing the image processing routines.
- **“*.o”**: Object files resulting from the compilation of the C sources.
- **“make_wrappers”**: Executable. Compiles all C sources with “python_wrappers.c” to produce “cfunctionsmodule.so” which is a python extension module. By importing this module in python scripts the C functions can be run as python functions.
- **“video_server.py”**: When run, it continuously captures and displays the robot’s camera image. Any request to save a camera image to file is made to this process.
- **“robot_server.py”**: Manages communication of motion commands to the robot. Any request to move the robot is made to this application.
- **“simulation_server.py”**: Runs a simple simulation in order to test the primitive procedures. The appropriate flag must be set in “various.py” so that primitive procedures direct video capture and navigation commands to this server instead of the video and robot servers.

Appendix D

The table below shows all the words found in the corpus collected for the IBL project. The words are presented in alphabetical order along with their occurrence in the corpus.

a	132	bear	2	clockwise	5
able	3	bears	1	come	46
about	4	been	1	continue	45
access	1	before	15	corner	11
across	18	beginning	2	cross	2
actually	1	beige	1	crossing	2
after	14	bend	18	crossroads	35
again	27	bending	4	currently	1
ahead	9	bends	1	degrees	7
all	14	bendy	3	derry's	17
almost	3	between	2	destination	11
along	14	big	2	diagonally	1
already	1	bit	12	did	2
amount	1	black	1	direction	7
an	4	blocks	1	directly	5
and	263	boots	17	discussed	2
angle	1	both	1	dixons	11
another	10	bottom	2	do	7
any	1	branch	1	door	3
apologise	1	bridge	48	doors	1
are	6	building	28	dotted	6
area	1	buildings	3	double	1
around	10	but	8	doubling	2
arrive	1	by	7	down	54
as	23	can	2	en	1
assuming	2	car	22	end	37
at	42	carry	25	ends	1
back	9	carrying	1	entering	1
basically	4	center	1	er	84
be	38	certainly	1	erm	71

exactly	2	head	5	library	6
exit	42	here	1	like	4
exits	1	hit	14	line	7
facing	2	hope	1	lines	1
far	1	hospital	13	little	12
feel	1	hotel	27	looks	1
few	5	house	1	lot	3
find	16	hundred	2	main	12
first	72	hut	8	make	1
five	1	i	6	means	1
follow	13	if	26	meet	3
following	4	immediate	6	mentioned	3
for	6	immediately	6	middle	4
fork	1	in	36	moment	1
forty-five	1	instead	4	more	1
forward	44	instruct	1	move	13
forwards	73	into	3	moving	1
from	35	is	85	museum	6
front	13	it	21	need	6
further	3	it'd	1	next	10
get	19	its	1	ninety	3
go	168	it's	15	now	1
going	68	junction	32	of	84
gone	3	just	28	off	11
got	18	keep	59	oh	2
grand	26	lake	6	ok	3
grey	4	leave	1	okay	62
half	1	leaving	1	on	253
has	1	left	188	once	5
have	7	left-hand	31	one	6

only	1	quarters	1	slight	4
onto	3	queens	1	slightly	21
opposite	13	queen's	14	small	2
or	1	reach	13	so	19
order	1	reaching	1	some	3
other	6	recall	3	soon	1
our	3	recalling	1	sorry	6
out	2	right	178	sort	5
outside	1	right-hand	26	space	2
over	40	road	109	start	5
paper	1	roads	2	starts	1
park	26	robot	1	station	10
parking	7	round	29	still	1
park's	1	roundabout	81	stop	17
pass	17	route	6	storey	1
passed	3	safeway	19	straight	61
passing	1	safeways	16	street	14
past	60	same	2	sure	4
pc	14	say	1	take	137
pizza	8	says	1	taken	1
place	2	second	41	taking	2
please	1	see	17	tesco	1
plymouth	13	set	1	tescos	11
pond	4	seventy	1	tesco's	1
post-office	57	sharp	1	thankyou	1
post-offices	1	short	2	that	48
previous	4	should	24	that's	6
previously	4	side	58	the	753
pub	16	sixty	1	then	97
quadrangle	1	skyscraper	1	there	32

there'll	2	up	9	your	127
there's	2	upon	1	you're	20
thing	1	very	2	yourself	3
think	1	walk	15	you've	11
third	20	walking	5	yuh	2
thirty	1	want	19	yup	2
this	12	was	2		
though	2	water	2		
three	4	way	18		
till	5	we	7		
time	6	well	2		
t-junction	13	went	1		
to	212	were	2		
too	1	what	1		
towards	12	when	14		
train	10	where	10		
travel	1	which	16		
travels	1	while	1		
tree	3	white	7		
trees	6	wiggles	1		
trip	1	will	44		
turn	109	with	12		
turned	1	without	1		
turning	28	work	2		
turnings	2	world	14		
two	6	yards	3		
uh-huh	1	yeah	7		
um	10	yes	6		
university	39	you	233		
until	53	you'll	40		

List of References/Bibliography

- Allen, J. F., Miller, B. W., Ringger, E. K., Sikorski, T., **Robust Understanding in a Dialogue System**, Proceedings of the 34th Annual Meeting of the Association of Computational Linguistics (ACC), June 1996, pp. 62-70.
- Aschwanden, P., Guggenbuhl, W., **Experimental Results from a Comparative Study on Correlation-Type Registration Algorithms**, Proceedings of the 2nd International Workshop on Robust Computer Vision, Wichmann, Karlsruhe, 1992, pp. 268-289.
- Asoh, H., Vlassis, N., Motomura, Y., Asano, F., Hara, I., Hayamizu, S., Itou, K., Kurita, T., Matsui, T., Bunschoten, R., Krose, B., **Jijo-2: An Office Robot That Communicates and Learns**, IEEE Intelligent Systems, Volume 16, No. 5, September/October 2001, pp. 46-55.
- Badler, N. I., Bindiganavale, R., Allbeck, J., Schuler, W., Zhao, L., Lee, S. J., Shin, H., Palmer, M., **Parameterized Action Representation and Natural Language Instructions for Dynamic Behaviour Modification of Embodied Agents**, AAAI Spring Symposium 2000, Stanford, California, 2000, pp. 6-10.
- Ballard, D.H., Brown, C.M., **Computer Vision**, Prentice-Hall, 1982.
- Barnard, K., **Computational Colour Constancy: Taking Theory into Practice**, MSc Thesis, School of Computing Science, Simon Fraser University, August 1995.
- Batavia, P., Singh, S., **Obstacle Detection Using Adaptive Color Segmentation and Color Stereo Homography**, Proceedings of the IEEE International Conference on Robotics and Automation, IEEE, May 2001, pp. 126-131.
- Bennet, S., **Real-Time Computer Control: An Introduction**, second edition, Prentice Hall, 1994.
- Bertozzi, M., Broggi, A., **GOLD: A Parallel Real-Time Stereo Vision System for Generic Obstacle and Lane Detection**, IEEE Transactions on Image Processing, Volume 7, No. 1, January 1998, pp. 62-81.

- Bischoff, R., Graefe, V, **Machine Vision for Intelligent Robots**, LAPR Workshop on Machine Vision Applications, Makuhari/Tokyo, November 1998, pp. 167-176.
- Bischoff, R., Jain, T., **Natural Communication and Interaction with Humanoid Robots**, Proceedings of the Second International Symposium on Humanoid Robots, Tokyo, October 1999, pp. 121-128.
- Braunl, T., **Embedded Robotics: Mobile Robot Design and Applications with Embedded Systems**, Springer, 2003.
- Broggi, A., Berte, S., **Vision-Based Road Detection in Autonomous Systems: A Real-Time Expectation-Driven Approach**, Journal of Artificial Intelligence Research 3, 1995, pp. 325-348.
- Broggi, A., **Robust Real-Time Lane and Road Detection in Critical Shadow Conditions**, Proceedings of the IEEE International Symposium on Computer Vision, Coral Gables, Florida, November 1995, pp. 353-358.
- Brøndsted, T., Dalsgaard, P., Larsen, L. B., Manthey, M., Mc Kevitt, P., Moeslund, T. B., Olesen, K. G., **CHAMELEON: a general platform for performing intellimedia**, The Eight International Workshop on the Cognitive Science of Natural Language Processing, Galway, Ireland, August, 1999, pp. 110-122.
- Brown, L.G., **A Survey of Image Registration Techniques**, ACM Computing Surveys, 24(4), December 1992, pp. 325-376.
- Bruckner B. K., Röfer, T., Carmesin, H. O., Müller, R., **A Taxonomy of Spatial Knowledge for Navigation and its Application to the Bremen Autonomous Wheelchair**, Freksa, C., Habel, C., Wender, K.F. (Eds.), Spatial Cognition I - An interdisciplinary approach to representing and processing spatial knowledge, Springer, Berlin, 1998, pp. 373-397.
- Cen, M., **Image Processing for Outdoor Path Layout Analysis**, MSc Thesis, School of Computing, University of Plymouth, United Kingdom, September 2000.
- Chang, P., Krumm, J., **Object Recognition with Colour Co-occurrence Histograms**, IEEE Conference on Computer Vision and Pattern Recognition, Fort Collins,

- CO, June 1999, pp. 2498-2504.
- Chun, W.J., **Core Python Programming**, Prentice Hall, 2001.
- Crangle, C. E., Suppes, P., **Language and Learning for Robots**, Stanford University, Stanford: CSLI Press, 1994.
- Crangle, C., **Conversational Interfaces to Robots**, *Robotica*, Volume 15, 1997, pp. 117-127.
- Crisman, J. D., Thorpe, C. E., **UNSCARF, A Colour Vision System for the Detection of Unstructured Roads**, Proceedings of the 1991 IEEE International Conference on Robotics and Automation, Sacramento, California, April 1991, pp. 2496-2501.
- Crisman, J.D., Thorpe, C.E., **SCARF: A Color Vision System that Tracks Roads and Intersections**, *IEEE Transactions on Robotics and Automation*, Volume 1, Issue 1, February 1993, pp. 49-58.
- Dahlback, N., Jonsson, A., Ahrenberg, L., **Wizard of Oz studies - why and how**, *Knowledge-Based Systems*, Volume 6, Number 4, 1993, pp.258-266.
- Deans, M. C., Hebert, M., **Invariant filtering for simultaneous localization and mapping**, *IEEE International Conference on Robotics and Automation*, Vol. 2, 2000, pp. 1042-47.
- DeMenthon, D., **A Zero-Bank Algorithm for Inverse Perspective of a Road from a Single Image**, Proceedings of the IEEE International Conference Robotics and Automation, Raleigh, NC, April 1987, pp.1444-1449.
- DeMenthon, D., Davis, L. S., **Reconstruction of a Road by Local Image Matches and Global 3D Optimization**, Proceedings of the IEEE International Conference on Robotics and Automation, 1990, pp. 1337-1342.
- Denis, M., **The description of routes: A cognitive approach to the production of spatial discourse**, *CPC*, 16:4, 1997, pp.409-458.
- Faugeras, O., **Three-Dimensional Computer Vision: A Geometric Viewpoint**, MIT Press, 1993.

- Fong, T., Thorpe, C., Baur, C., **Collaboration, Dialogue and Human-Robot Interaction**, 10th International Symposium of Robotics Research, Springer-Verlag, Lorne, Victoria, Australia, November 2001.
- Fraczak, L., **From route descriptions to sketches: a model for a text-to-image translator**, 33rd Annual Meeting of the Association for Computational Linguistics (ACL-95), Student Session, MIT, Cambridge, USA, 1995, pp. 299-301.
- Fua, P., Brechbuhler, C., **Imposing Hard Constraints on Soft Snakes**, European Conference on Computer Vision (ECV 1996), Cambridge, England, April 1996, pp. 495-506.
- Gapp, K. P., **Object Localization: Selection of Optimal Reference Objects**, Conference On Spatial Information Theory (COSIT), 1995, pp. 519-536.
- Gonzales, R.C., Woods, R.E., **Digital Image Processing**, Addison-Wesley, 1992.
- Graefe, V., Bischoff, R., **Vision-Guided Intelligent Robots**, International Conference on Mechatronics and Machine Vision in Practice, Nanjing, September 1998, pp. 21-26.
- Green, A., **C-Roids: Life-like Characters for Situated Natural Language User Interfaces**, Proceedings of 10th IEEE International Workshop on Robot and Human Interactive Communication, ROMAN2001, Bordeaux/Paris, France, September 2000. pp. 140-145.
- Green, A., Hüttenrauch, H., Norman, M., Oestreicher, L., Eklundh, K. S., **User Centered Design for Intelligent Service Robots**, Proceedings of 9th IEEE International Workshop on Robot and Human Interactive Communication, ROMAN2000Osaka, Japan, September 2000, pp. 161-166.
- Green, A., Severinson-Eklundh, K., **Task-oriented Dialogue for CERO: a User-centered Approach**, In Proceedings of Ro-Man'01 (10th IEEE International Workshop on Robot and Human Communication), Bordeaux, Paris, September 2001, pp. 146-151.
- Gutmann, J. S., Konolige, K., **Incremental Mapping of Large Cyclic Environments**,

- Proceedings of Computational Intelligence in Robotics and Automation (CIRA), 1999, pp. 318-325.
- Hausser, R., **Foundations of Computational Linguistics: Human-Computer Communication in Natural Language**, 2nd Edition, Berlin, Springer, 2001.
- Herzog, G., **From Visual Input to Verbal Output in the Visual Translator**, Technical Report 124, Universitat des Saarlandes, VITRA, July 1995.
- Herzog, G., Wazinski, P., **Visual TRANslator: Linking Perceptions and Natural Language Descriptions**, Artificial Intelligence Review, 8(2), 1994, pp. 175-187.
- Hu, Z., Uchimura, K., **Action-Based Road Horizontal Shape Recognition**, SBA Contole & Automacao, Volume 10, No. 2, 1999, pp. 83-88.
- Huffman, S. B., Laird, J. E., **Learning procedures from interactive natural language instructions**, Machine Learning: Proceedings of the Tenth International Conference, P. Utgoff, editor, June 1993, pp 143-150.
- Huffman, S., **Instructable autonomous agents**, PhD Dissertation, University of Michigan, 1994.
- Inaba M., Kagami S., Kanehiro F., Hoshino Y., Inoue H., **A platform for robotics research based on the remote-brained robot approach**, International Journal of Robotics Research, 19:10, 2000, pp. 933-954.
- Jain, A.K., **Fundamentals of Digital Image Processing**, Prentice-Hall, 1989.
- Jochem, T. M., Pomerleau, D. A., Thorpe, C. E., **Vision Based Intersection Navigation**, Proceedings of the 1996 IEEE Symposium on Intelligent Vehicles, September 1996, pp. 391-396.
- Jochem, T., Pomerleau, D., Thorpe, C., **Vision-Based Neural Network Road and Intersection Detection and Traversal**, IEEE Conference on Intelligent Robots and Systems (IROS '95), Vol. 3, August, 1995, pp. 344 - 349.
- Jones, J., Flynn, A., Seiger, B., **Mobile robots-From Inspiration to Implementation**, 2nd Edition, AK Peters, Wellesley MA, 1999.
- Kalinke, T., Tzomakas, C., von Seelen, W., **A Texture based Object Detection and an**

- Adaptive Model-based Classification**, Proceedings of the IEEE Intelligent Vehicles Symposium '98, Stuttgart, Germany, October 1998, pp. 341-346.
- Kalman, R., **A New Approach to Linear Filtering and Prediction Problems**, Transactions of the ASME – Journal of Basic Engineering, Vol. 82, Series D, pp. 35-45.
- Kaske, A., Wolf, D., Husson, R., **Lane Boundary Detection Using Statistical Criteria**, Proceedings of International Conference on Quality by Artificial Vision, QCAV'97, Le Creusot, France, 1997, pp. 28-30.
- Koay, K. L., **Intelligent Vision-Based Navigation System**, PhD Thesis, University of Plymouth, November 2002.
- Laird J.E., Newell A. and Rosenbloom P.S., **Soar: An architecture for general Intelligence**, Artificial Intelligence, 33:1, 1987, pp.1-64.
- Lauria S., Bugmann G., Kyriacou T., Bos J., Klein E., **Personal Robot Training via Natural-Language Instructions**, IEEE Intelligent Systems, 16:3, 2001, pp. 38-45.
- Lauria, S., Bugmann, G., Kyriacou, T., Bos, J., Klein, E., **Converting Natural Language Route Instructions into Robot Executable Procedures**, Proceedings of the 2002 IEEE International Workshop on Robot and Human Interactive Communication (ROMAN '02), Berlin, Germany, 2002, pp. 223-228.
- Leigh, J.R., **Applied Digital Control**, second edition, Prentice Hall, Englewood Cliffs, NJ, 1992.
- Lippmann, R. P., **Speech Recognition by Machines and Humans**, Speech Communication 22, 1997, pp.1-15.
- Lopes, S. L., Teixeira, A., **Human-Robot Interaction through Spoken Language Dialogue**, Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2000, pp. 528-534.
- Lueth, T. C., Laengle, T., Herzog, G., Stopp, E., Rembold, U., **KANTRA: Human-Machine Interaction for Intelligent Robots Using Natural Language**, 3rd

- IEEE Int. Workshop on Robot and Human Communication, RO-MAN'94, Nagoya, Japan, 1994, pp. 106-111.
- Maass, W., **From visual perception to multimodal communication: Incremental route descriptions**, *AI Review Journal*, 8(2-3), 1994, pp. 159-174.
- Maass, W., Baus, J., Paul, J. **Visual grounding of route descriptions in dynamic environments**, *Proceedings of the AAAI Fall Symposium on Computational Models for Integrating Language and Vision*, MIT Press, Cambridge, 1995a.
- Maass, W., **How spatial information connects visual perception and natural language generation in dynamic environments: Towards a computational model**, *Spatial Information Theory: A Theoretical Basis for GIS. Proc. of the Int. Conference COSIT'95*, Semmering, Austria, 1995b, pp. 223-240.
- Marchant, J. A., Onyango, C. M., **Shadow-invariant classification for scenes illuminated by daylight**, *Journal of the Optical Society of America A (JOSA A)*, Vol. 17, Issue 11, November 2000, pp. 1952-1961.
- Niblack, W., **An Introduction to Digital Image Processing**, Prentice/Hall International, 1986.
- Nicolescu, M. N., Mataric, M. J., **Learning and Interacting in Human-Robot Domains**, *IEEE Transactions on Systems, Man and Cybernetics – Part A: Systems and Humans*, Volume 31, No. 5, September 2001, pp. 419-430.
- Onillon, V., Bugmann, G., Simpson, A., Nurse, P., **Artificial vision for micromouse**, *Research Report NRG-95-05*, School of Computing, University of Plymouth, Plymouth, United Kingdom, 1995.
- Pearson, D., **Image Processing**, McGraw-Hill, 1991.
- Pratt, W.K., **Digital Image Processing**, John Wiley and Sons, 1991.
- Ringger, E. K., **A Robust Loose Coupling for Speech Recognition and Natural Language Understanding**, *Technical Report 592*, The University of Rochester, Computer Science Department, Rochester, New York, September 1995.
- Rosenfeld, A., **Picture Processing by Computer**, Academic Press, 1969.

- Rosenfeld, A., Kak, A.C., **Digital Picture Processing**, Vol. I and II, Academic Press, Orlando, FL, 1982.
- Ross, R. J., Kelly, R., **Social Robotics: The Need for Audio & Language Interaction**, Technical Report, Department of Computer Science, University College Dublin, Ireland.
- Sayd, P., Chapuis, R., Aufrere, R., Chausse, F., **A Dynamic Vision Algorithm to Recover the 3D Shape of a Non-Structured Road**, Proceedings of the 1998 IEEE International Conference on Intelligent Vehicles, 1998, pp. 80-86.
- Schleidt, M., Kien, J., **Segmentation in Behaviour and what it can tell us about Brain Function**, Human Nature, Volume 8, No. 1, 1997, pp. 77-111.
- Smith, S. M., **ALTRUISM: Interpretation of three-dimensional information for autonomous vehicle control**, Engineering Applications of Artificial Intelligence, 8(3), 1995, pp. 271-280.
- Snaith, M., Lee, D., Probert, P., **A Low-Cost System Using Sparse Vision for Navigation in the Urban Environment**, Image and Vision Computing, Volume 16, 1998, pp. 225-233.
- Spyropoulos, C., **Image Processing for Robot Navigation in Urban Environment**, MSc Thesis, School of Computing, University of Plymouth, United Kingdom, September 1999.
- Spiliotopoulos, D., Androutsopoulos, I., Spyropoulos, C. D., **Human-Robot Interaction Based on Spoken Natural Language Dialogue**, European Workshop on Service and Humanoid Robots (servicerob 2001), Santorini, Greece, 2001.
- Stopp, E., Gapp, K. P., Herzog, G., Laengle, T., Lueth, T. C., **Utilizing Spatial Relations for Natural Language Access to an Autonomous Mobile Robot**, KI-94: Advances in Artificial Intelligence, Springer, Berlin, Heidelberg, 1994, pp. 39-50.
- Takeuchi, Y., Hebert, M., **Finding Images of Landmarks in Video Sequences**, Proceedings of the IEEE Conference on Computer Vision and Pattern

- Taylor, H. A., Tversky, B., **Perspective in Spatial Descriptions**, Journal of Memory and Language, 35, 1996, pp. 371-391.
- Theobalt, C., Bos, J., Chapman, T., Romero, A. E., Fraser, M., Hayes, G., Klein, E., Oka, T., Reeve, R., **Talking to Godot: Dialogue with a mobile robot**, Proceedings of 2002 IEEE/RSJ International Conference on Intelligent Robots and System, 2002, pp. 1338-1343.
- Thorpe, C., Hebert, M., Kanade, T., Shafer, S., **Vision and navigation for the Carnegie-Mellon Navlab**, IEEE Transactions on Pattern Analysis and Machine Intelligence, 10(3), 1988, pp. 362-373.
- Torrance, M. C., **Natural Communication with Robots**, MSc Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, February, 1994.
- Traum, D., Bos, J., Cooper, R., Larsson, S., Lewin, I., Matheson, C., Poesio, M., **A model of dialogue moves and information state revision**, Trindi Report D2.1, 1999.
- Turk, M. A., Morgenthaler, D. G., Gremban, K. D., Marta, M., **VITS: A Vision System for Autonomous Land Vehicle navigation**, IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 10, No. 3, May 1988, pp. 342-361.
- Unnikrishnan, R., Kelly, A., **A Constrained Optimization Approach to Globally Consistent Mapping**, Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems, EPFL, Lausanne, Switzerland, October 2002b, pp. 564-569.
- Unnikrishnan, R., Kelly, A., **Mosaicing Large Cyclic Environments for Visual Navigation in Autonomous Vehicles**, Proceedings of the IEEE Conference on Robotics and Automation (ICRA), May 2002a, pp. 4299-4306.
- Wang, Y., Shen, D. D., Teoh, E. K., **Lane Detection Using Catmull-ROM Spline**, Proceeding of IEEE International Conference on Intelligent Vehicles, IV'98, Germany, October 1998, pp. 51-57.

- Wang, Y., Shen, D., Teoh, E. K., **Lane Detection Using Spline Model**, Pattern Recognition Letters, 21(8), July 2000, pp. 677-689.
- Wang, Y., Teoh, E. K., **Lane Detection Using B-Snake**, Proceedings of the IEEE International Conference on Intelligence, Information and Systems, Hayatt Regency, Bethesda, November, 1999, pp. 438-443.
- Waxman, A. M., Lemoigne, J. J., Davis, L. S., Srinivasan, B., Kushner, T. R., Liang, E., Siddalingaiah, T., **A Visual Navigation System for Autonomous Land Vehicles**, IEEE Journal of Robotics and Automation, Volume 3, Issue 2, April 1987, pp. 124-141.
- Wilson, M. B., Dickson, S., **Poppet: A Robust Boundary Detection and Tracking Algorithm**, BMVC99 (British Machine Vision Conference 1999), pp. 352-361.
- Worrall, A. D., Ferryman, J. M., Sullivan, G. D., Baker, K. D., **Pose and Structure Recovery using Active Models**, Proceedings of the British Machine Vision Conference (BMVC), Birmingham, United Kingdom, 1995, pp. 137-146.
- Yang, J., Ozawa, S., **Recovery of 3-D Road Plane Based on 2-D Perspective Image Analysis and Processing**, IEICE Trans. Fundamentals, Volume E79-A, No. 8, August 1996, pp. 1188-1193.
- Zue, V., **Conversational interfaces: Advances and challenges**, In Proceedings of Eurospeech, Rhodes, Greece, 1997, pp. 9-14.

Copies of Publications

Personal Robot Training via Natural-Language Instructions.

Stanislao Lauria, Guido Bugmann¹, Theocharis Kyriacou, Johan Bos*, Ewan Klein*

Centre for Neural and Adaptive Systems, School of Computing, University of Plymouth
Drake Circus, Plymouth PL4 8AA, United Kingdom.

*Institute for Communicating and Collaborative Systems, Division of Informatics, University of Edinburgh, 2
Buccleuch Place, Edinburgh EH8 9LW, Scotland, United Kingdom.

<http://www.tech.plym.ac.uk/soc/staff/guidbugm/ibl/index.html>

19/3/2001

Abstract

Future domestic robots will need to adapt to the special needs of their users and to their environment. Programming by natural language will be a key method enabling computer language-naïve users to instruct their robots. Its main advantages over other learning methods are speed of acquisition and ability to build high level symbolic rules into the robot. This paper describes the design of a practical system that uses unconstrained speech to teach a vision-based robot how to navigate in a miniature town. The robot knows a set of primitive navigation procedures that the user can refer to when giving route instructions. A particularity of this project is that the primitive procedures are determined by analysing a corpus of route instructions. It is found that functions primitives natural to the user, such as "turn left after the church" are very complex procedures for the robot, involving visual scene analysis and local route planning. Thus, to enable natural user-robot interaction, a high-level of intelligence needs to be built into "primitive" robot procedures. Another finding is that the set of primitive procedures is likely not to be closed. Thus, on time to time, a user is likely to refer to a procedure that is not pre-programmed in the robot. How best to handle this is currently investigated. In general, the use of Instruction-Based Learning (IBL) imposes a number of constraints on the design of robotics systems and knowledge representation. These issues are developed in the paper and proposed solutions described.

1. Introduction

Intelligent robots must be capable of action in reasonably complicated domains with some degree of autonomy. This requires adaptivity to a dynamic environment, ability to plan and also speed in the execution. In the case of helper robots, or domestic robots, the ability to adapt to the special needs of their users is crucial. As most users are computer-language-naïve, they cannot personalise their robot using standard programming methods. Indirect methods, such as learning by reinforcement or learning by imitation, are also not appropriate for acquiring user-specific knowledge. For instance, learning by imitation does not enable the acquisition of rules such as "IF-THEN". Learning by reinforcement is a lengthy process that is best used for refining low-level motor controls, but becomes impractical for complex tasks. Further, both methods do not readily generate knowledge representations that the user can interrogate. An alternative method, learning from verbal instructions is explored in this paper.

Instruction-based learning (IBL) has several potential advantages. Natural language can express rules and sequence of commands in a very concise way. Natural language uses symbols and syntactic rules and is well suited to interact with robot knowledge represented at the symbolic level. It

¹ To whom correspondence should be addressed.

has been shown that learning in robots is much more effective if it operates at the symbolic level (Cangelosi and Harnad, 2001). This is to be contrasted with the much slower learning at the level of direct sensory-motor associations.

Previous work on verbal communication with robots has mainly focused on issuing *commands*, i.e. activating pre-programmed procedures using a limited vocabulary (e.g. IJCAI'95 office navigation contest). Only a few research groups have considered *learning*, i.e. the stable and reusable acquisition of new procedural knowledge. An inspiring project was Instructo-SOAR (Huffman & Laird, 1995). This system used textual input into a simulation of a manipulator with a discrete state and action space. Another investigation (Crangle and Suppes, 1994) used voice input to teach displacements within a room and mathematical operations, but with no reusability. In (Torrance, 1995), textual input was used to build a graph representation of spatial knowledge. This system was brittle due to place recognition from odometric data and use of IR sensors for reactive motion control. Knowledge acquisition was concurrent with navigation, not prior to it. The present work aims at using unconstrained language in a real-world robotic application. More recent projects with related scope are CARL (Seabra Lopes and Teixeira, 2000) and HERMES (Bischoff and Jain, 1999).

The use of IBL has system-wide repercussions on the design of a robot control system. (i) The robot must be able to convert utterances in natural language into internal symbols that the robot understands. By understanding we mean here that there is a correspondence between symbols and actions or real-world objects. Thus primitive functions associated with symbols must be provided. (ii) The robot must be able to distinguish a command to be executed immediately from an instruction to be memorized. This requires context resolution at the natural language processing level. (iii) The robot must be able to verify that the instruction can be converted into an executable procedure. This requires an internal representation of consequences of actions. (iv) In the case of translation errors at any level of user-robot communication process, the robot must be able to inform the user and engage in a repair dialogue. This requires sophisticated dialogue management. (v) Finally the robot must be able to execute a command while listening to the user, and must cope with interruptions and inappropriate answers to its requests. This requires a carefully designed system architecture.

This paper describes initial steps and considerations towards a practical realisation of an IBL system. The experimental environment is that of a miniature town in which a robot provided with video camera executes route instructions. The robot has a set of pre-programmed sensory-motor action primitives, such as "turn left" or "follow the road". The task of the user is to teach the robot new routes by combining action primitives using unconstrained speech. That task should reveal all the constraints described above, and enable testing of the developed methodology.

Section 2 clarifies how symbol-level description and low-level sensory motor action procedures are integrated. The proposed representation of procedural knowledge is also described. In section 3, natural language processing is described. In section 4, the system architecture is described.

One of the problems to consider is the selection of action primitives. This is done here by analyzing recorded route instructions, and establishing a list of actions that are natural to users. Section 5 describes this process and presents the result of this investigation. One of them is that the list of primitives may not be a closed one. The implications of that and other findings is discussed in section 6, along with the question of how the proposed system compares to other approaches.

2. IBL model

2.1 Symbolic learning

In IBL, verbal instructions given by the user are converted into new internal program code that represents new procedures. Such procedures become part of a pool of procedures that can then be reused to learn more and more complex procedures. Hence, the robot becomes able to execute increasingly complex tasks.

This process starts with a predefined initial knowledge. This “innate” knowledge consist of primitive sensory-motor procedures with names, such “turn left”, “follow the road” (the choice of these primitives is explained in section 2.3 and 5). The names constitute the “symbols”, and the pieces of computer program that controls the execution of corresponding procedures are called “actions” (Figure 1A). As each symbol is associated with an action, it is said to be “grounded”.

When a user explains a new procedure to the robot, say a route from A to B that involves a number of primitive actions, the IBL system creates a new name for the procedure, and writes a new piece of program code that executes that procedure, and links the code with the name (see section 2.2 for details). The code refers to primitives actions by name. It does not duplicate the low-level code defining theses primitives. For that reason, the new program can be seen as a combination of symbols rather than a combination of actions (figure 1B). As all new procedures are constructed from grounded primitives, they become grounded by inheritance and are thus “understandable” by the system when referred to in natural language.

When explaining a new procedure, the user can also refer to old procedures previously defined by himself. In that way the complexity of the robot's symbolic knowledge increases (fig. 1C).

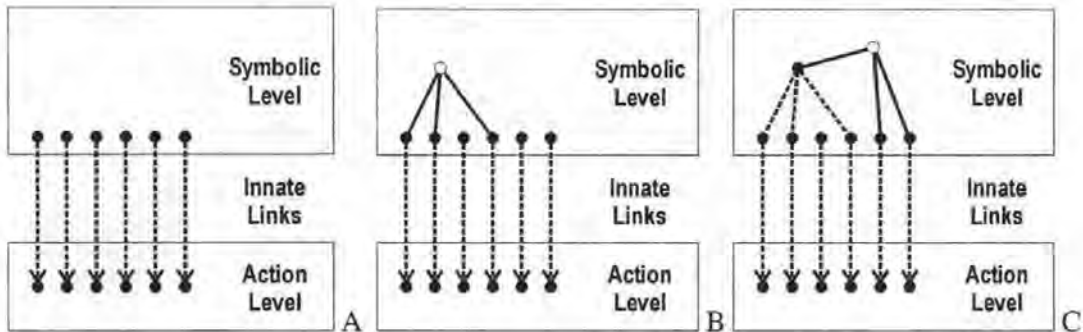


Figure 1. Symbolic learning. (A) is a schematic representation of the initial system, comprising symbols associated with pre-programmed (innate) primitive action procedures. In (B) the user has defined a new procedure (open circle) as a combination of symbols. The new symbol is grounded because it is a construct of grounded symbols. In (C), the user has defined a new procedure that combines a procedure previously defined by himself with primitive action procedures.

2.2 Knowledge representation

The internal representation needs to support three functions: (i) formal modeling of NL route descriptions; (ii) internal route planning for determining whether a given route description is sufficiently specified; and (iii) the generation of procedures for navigation at execution time. These three functions require different representations that will be described in turn.

(i) The utterances of the user are represented using the discourse representation structure (DRS) (section 3). This is then translated into symbols representing procedures or is used to initiate internal functions such as execution of a command or learning of a series of commands (section 4).

(ii) When the user describes a route as a sequence of actions, it is important for the robot to verify if this sequence is executable. The approach proposed here associate each procedure with a triplet $S_i A_{ij} S_j$ with properties similar to productions in SOAR (Laird et al, 1987). The state S_i is the *pre-condition* for action A_{ij} . It defines which components of the input state vector need to have specific values for the action A_{ij} to be possible. The state S_j is the final state, resulting from the action A_{ij} applied to the input state. For a sequence of actions to be realisable, the final state of one action must be compatible with the pre-condition of the next one. To enable this verification, the robot must be able to “imagine” the consequence of an action. For that purpose, a PREDICTION function is associated with each primitive action, and with each newly created procedure. Figure 2 illustrates the use of the prediction function during verification of the consistency of the sequence of instructions

from the user. It should be noted that this process also helps detecting some of the errors in natural language processing.

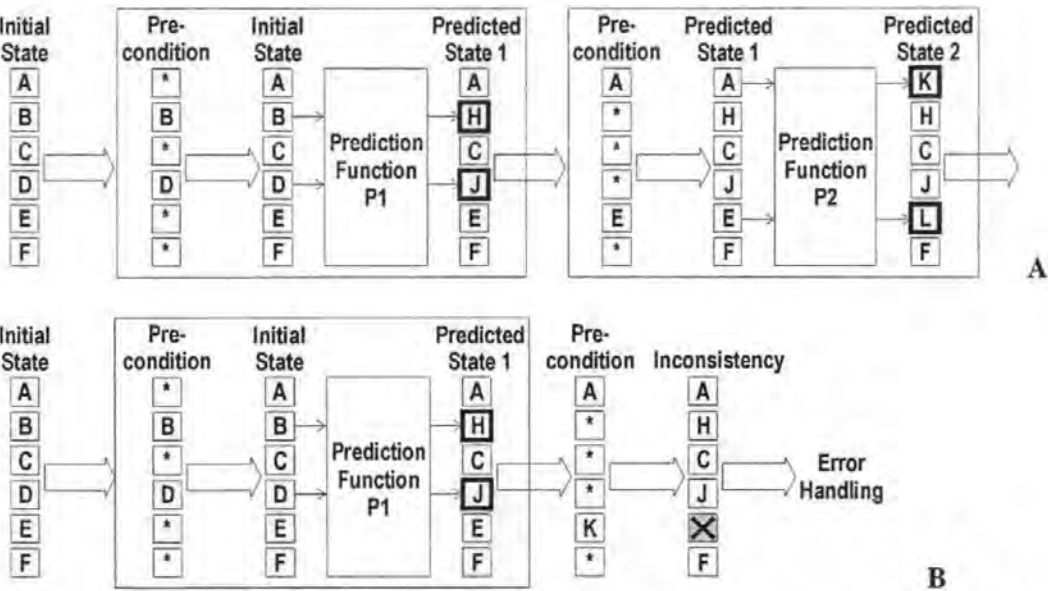


Figure 2. Route instruction verification. (A) For each procedure there is a prediction function that transforms a state vector into its future value. The function first determines if the input state satisfied the minimal criteria (“pre-condition”) to enable the procedure to be executed. An action is executable only if selected elements of the state vector have required values. If this is the case, the next state is predicted and processed by the prediction function associated with the next procedure in the instruction. Each action modifies certain components of the state vector, and leaves the others unchanged. (B) If the predicted state produced by one procedure does not allow the next procedure to be executed, an error handling process is initiated. (Note: the “initial state” in the text corresponds to the “current state” in the figure).

(iii) When a robot executes a command, it executes a piece of program code that contains the sequence of primitive procedures to be executed. Thus, a key part of IBL is the generation of a program code. This is enabled by the use of a scripting language (section 4). This program is called the ACTION function. Both ACTION and PREDICTION functions are physically located in the same file that contains all information specific to a procedure. This is schematised in figure 3.

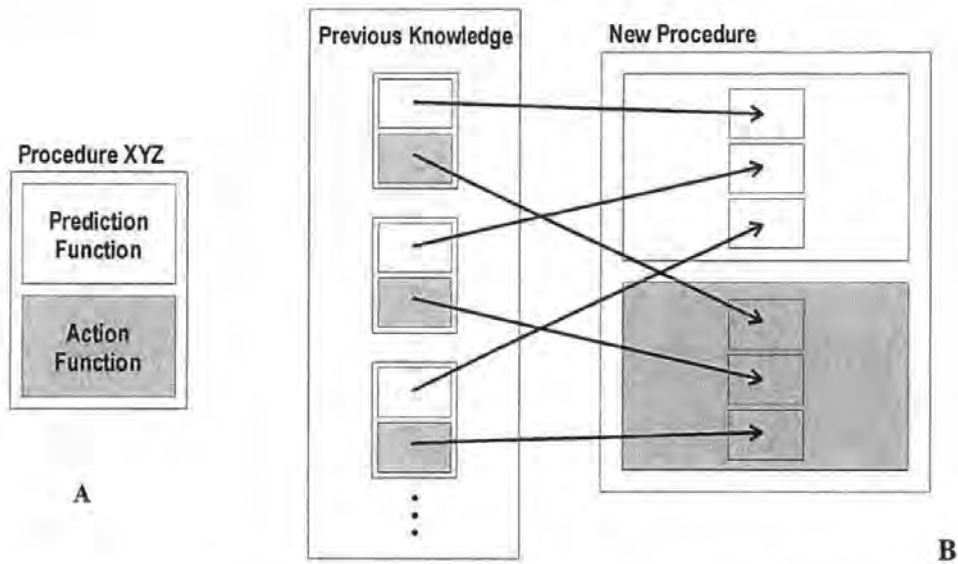


Figure 3. Procedural knowledge representation. (A) A procedure file contains an ACTION function that causes the physical displacement of the robot, and a PREDICTION function that calculates the future state of the robot resulting from the action. The ACTION is used during execution of a command, and the PREDICTION is used for consistency checking during the learning process. (B) An instruction by the user results in a “New Procedure” file being written. In this file, the actions components of the requested primitive procedures are combined (in the form of function calls) to create the new ACTION function, and the prediction components are combined to create the new PREDICTION function. This includes an additional procedure-specific pre-condition.

2.3 Sensory-Motor primitives

Sensory-motor primitives are defined as actions that users usually refer to in unconstrained speech. These are not low-level robot control actions, and often involve complex processing and planning. A task such as “approach that building at the end of the street” is a typical action that users ask the robot to do at the end of a route instruction, when the goal is in sight (section 5). It is a complex action involving visual detection of a building and of its entrance, its localisation in relation to the street, and planning of a route along the street. All this is easy for a human, but in many ways stretches the limits of robot “intelligence”.

The use of relatively high-level primitives allows to accept underspecified natural-language commands as executables procedures, and thus simplifies the mapping from natural language expressions to robot procedures. It also gives the robot some autonomy in the execution of commands, as the execution details depend on the local conditions.

3. Natural Language Processing

3.1 The Dialogue Move Engine

The ongoing dialogue between user and robot is represented by a discourse representation structure (DRS) proposed by Discourse Representation Theory (Kamp & Reyle 1993). New utterances yield DRSs (see section 3.2 below), which update the DRS of the dialogue, following the recent information state approach to dialogue processing (Traum et al. 1999). Context-sensitive expressions (such as pronouns and presupposition triggers) are resolved with respect to this DRS. Finally,

utterances of the robot are realized by generating prosodically annotated strings from DRSs and feeding these to a synthesizer.

Using semantic representations for modeling the dialogue is motivated by the need to perform inferences in order to let the robot make "intelligent" responses. Inferences are required to resolve ambiguities present in the user's input (being of scopal, referential, or lexical nature), to detect the speech act associated to an utterance (e.g., did the user answer a question or has a new issue been raised?), to plan the next utterance or action, and to generate naturally sounding utterances (e.g., by distinguishing old from new information within an utterance). Inference are actually carried out using off-the-shelf theorem provers, by translating DRSs to first-order logic (cf. Blackburn et al. 1999).

3.2 Speech Recognition

Speech recognition and semantic construction are integrated into one component. The basic idea is to use off-the-shelf speech recognition, and to use a grammar that is linguistically motivated and domain independent. The grammar not only consists of rules that determine the syntactic structure of utterances, but also features semantic rules that specify how semantic representations (underspecified DRSs) are built in a compositional way.

The current prototype implementation uses Nuance² tools for speech recognition. The initial grammar is a unification-based phrase structure grammar, which is compiled into GSL, the Grammar Specification Language supported by Nuance's technology. This compilation involves removing left-recursive rules within the grammar, as well as replacing features and their possible values for syntactic category symbols, as GSL neither support left-recursive rules nor a feature-value system. As a consequence, the language models for the speech recognition are huge, but still feasible for small lexicons (a few hundred words in the case of IBL).

The semantic operations are compiled-out in GSL as well, and each word in the lexicon is associated with a semantic representation. As a result, the output of the speech recognizer is directly a semantic representation, in our case an underspecified DRS, and another step of processing (such as parsing and semantic construction) is not required. Hence, by compiling our linguistic grammar into GSL, we short-cut the parsing and semantic construction process into a single component.

4. System Architecture

The architecture is comprised of several functional processing modules (figure 4). These are divided into two major units: the Dialogue Manager (DM) and the Robot Manager (RM).

The DM and the RM are designed as two different processes based on asynchronous communication protocols. These processes run concurrently on different processors. In this way, the system can handle, at the same time, both the dialogue aspects of an incoming request from the user (i.e. speech recognition and semantic analysis, or detection of a "stop" command) and the execution of a previous user request (i.e. check if the request is in the system knowledge domain, and execute vision-based navigation procedures).

² <http://www.nuance.com>

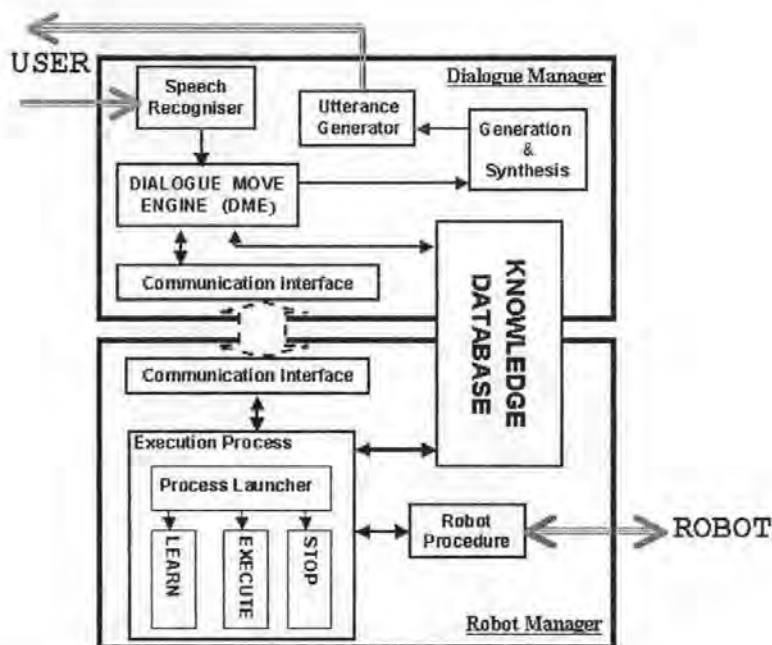


Figure 4. IBL system's architecture (see text for description).

Two aspects are essential with this concurrent-processes approach. Firstly, to define an appropriate protocol between the two processes. Secondly, to define an appropriate architecture for the RM and DM allowing the two processes to both communicate with each other while performing other tasks. At present a communication protocol based on sockets and context-tagged messages is evaluated.

Moreover, the system must also dynamically adapt itself to new user requests or to new internal changes, by being able to temporarily suspend or permanently interrupt some previous activity. For example the user may want to prevent the robot crashing against a wall and must therefore be able to stop the robot while it is driving towards the wall. Hence, the importance of a concurrent approach where the system constantly listens to the user while performing other tasks. Further, the system must be able adjust task parameters if necessary.

The Dialogue Manager is a bi-directional interface between the Robot Manager and the user, either converting speech input into a semantic representation, or converting requests from the Robot Manager into dialogues with the user. Its components are run as different processes communicating with each other via a blackboard architecture.

The RM must be able to concurrently listen to messages from to the DM and process them, and send requests to the DM. For this reason a multi-threads approach has been used. The communication interface is a process that launches a message-evaluation thread "Execution Process" (fig. 4) and then resumes listening to the DM. The execution process then starts an appropriate thread for executing a command, or places a tagged message on a message board if the message is part of a dialogue in a specific thread, e.g. learning a route.

The Robot Manager is written using the scripting language Python³ and C. An important feature of scripting languages is their ability to write their own code. For instance, a route instruction given by the user will be saved by the Robot Manager as a Python script that then becomes part of the procedure set available to the robot for execution or for future learning.

³ <http://www.python.org>

5. Corpus Collection and Data Analysis

To evaluate the potential and limitations of IBL, a real-world instructions task is used, that is simple enough to be realisable, and generic enough to warrant conclusions that hold also for other task domains. A simple route scenario has been selected, using real speech input and a robot using vision to execute the instructed route (see 5.1 below for more details). The first task in the project is to define the innate actions and symbols in the route instruction domain. For this reason, a corpus of route descriptions has been collected from students and staff at the University of Plymouth. In section 5.2 and 5.3 corpus collection and data analysis are presented.

5.1 Experimental Environment.

The environment is a miniature town covering an area of size 170cm x 120cm (figure 5). The robot is a modified RobotFootball robot⁴ with a 8cm x 8cm base (figure 6A). The robot carries a CCD colour TV camera⁵ (628 (H) x 582 (V) pixels) and a TV VHF transmitter. Images are processed by a PC that acquires them via with a TV capture card⁶ (an example of such image is shown in figure 6B). The PC then sends motion commands by FM radio to the robot. During corpus collection, the PC is also used to record instructions given by subjects.

The advantage of a miniature environment is the ability to build a complex route structure in the limited space of a laboratory. The design is as realistic as possible, to enable subjects to use expressions natural for the outdoor real-size environment. Buildings have signs taken from real life to indicate given shops or utilities such as the post-office. However, the environment lacks some elements such as traffic lights that may normally be used in route instructions. Hence the collected corpus is likely to be more restricted than for outdoor route instructions.

The advantage of using a robot with a remote-brain architecture (Inaba et al., 2000) is that the robot does not require huge on-board computing and hence can be small, fitting the dimensions of the environment.

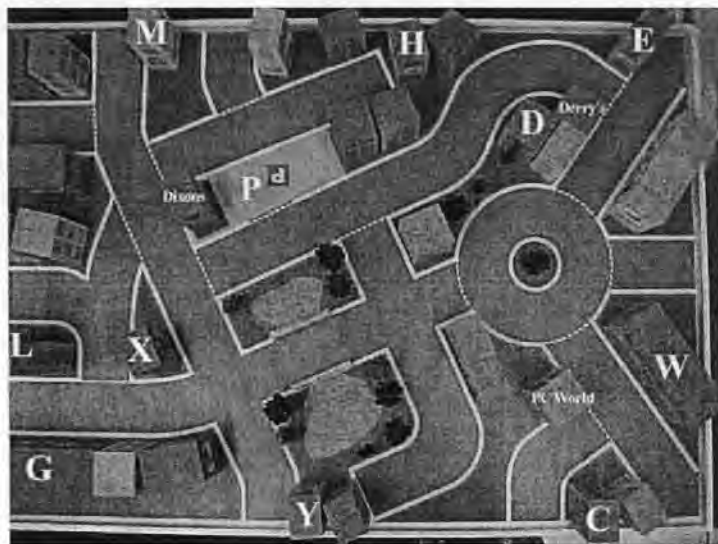


Figure 5. Miniature town in which a robot will navigate according to route instructions given by users. Letters indicate the destinations and origins of various routes used in the experiment.

⁴ Provided by Merlin Systems (<http://www.merlinsystemscorp.com/>)

⁵ Provided by Allthings Sales and Services (<http://www.allthings.com.au/>)

⁶ TV Card: Hauppauge WinTV GO



Figure 6. A. Miniature robot (base 8cm x 8cm). B. View from the on-board colour camera.

5.2 Collection of a corpus of route instructions

To collect linguistic and functional data specific to route learning, 24 subjects were recorded as they gave route instructions to the robot in the environment. Subjects were divided into three groups of 8. The first two groups (A and B) used totally unconstrained speech, to provide a performance baseline. It is assumed that a robot that can understand these instructions as well as a human operator would represent the ideal standard. Subjects from group C were induced in producing shorter utterances by a remote operator “taking notes”. Subjects in groups A and B were told that the robot was remote-controlled and that, at a later date, a human operator would use their instructions to drive the robot to its destination. It was specified that the human operator would be located in another room, seeing only the image from the wireless on-board video camera. This induced subjects to use a camera-centred point of view relevant for robot procedure primitives. Subjects were also told to reuse previously defined routes whenever possible, instead of re-explaining them in detail. Each subject had 6 routes to describe among which 3 where “short” and 3 where “long”. The long routes included a short one, so that users could refer to the short one when describing the long one, instead of re-describing all segments of the short one (figure 1c). This was to reveal the type of expressions used by users to link taught procedures with primitive ones. Each subject described 6 routes having the same starting point and six different destinations. Starting points were changed after every two subjects. A total of 144 route descriptions were collected. For more details about collection and analysis of the corpus see (Bugmann et al. 2001).

5.3 Corpus Analysis: The functional vocabulary

The aim of the corpus analysis is to twofold. First, to define the vocabulary used by the users in this application, in order to tune the speech recognition system for an optimal performance in the task. Secondly, to establish a list of primitive procedures that users refer to in their instructions. The aim is to pre-program these procedures so that a direct translation from the natural language to grounded symbols can take place. In principle, if the robot does not know a procedure, the user could teach it. However, this is a process that we wish to avoid at this stage of the project, as discussed in section 6. Hereafter, we report on the functional analysis of the corpus (Groups A and C merged. Group B not included at this point in time). The reader interested in the task vocabulary can refer to (Bugmann et al., 2001).

The functional vocabulary is a list of primitive navigation procedures found in route descriptions. The initial annotation of instructions in terms or procedures, as reported here, is somehow subjective, and influenced by two considerations. (i) The defined primitives will eventually be produced as C-Programs. It was hoped that only a few generic procedures would have to be written. Therefore, the corpus has been transcribed into rather general procedures characterised by several parameters (table 1). (ii) An important issue is knowledge representation. According to the

SAS representation discussed in section 2, the executability of primitives can only be evaluated if their initial and final states are defined. Subjects however rarely specified explicitly the starting point of an action and sometimes did not define the final state in the same utterance. Nevertheless, it was assumed that the system would be able to infer the missing information from the context. Therefore, procedures without initial or final state were considered to be complete, and were annotated as such. The specifications of primitive procedures is likely to evolve during the project.

This analysis methodology differs slightly from the one in (Denis, 1997). In our analysis, there are no statements describing landmarks, as these are made part of procedures specifications, and consequently there are also no actions without reference to landmarks. Even when a subject specified a non-terminated action, such as "keep going", it was classified as "MOVE FORWARD UNTIL", assuming that a termination point would be inferred from the next specified action. The list of actions found in the route descriptions of groups A and C is given in table 1.

	Count	Primitive Procedures
1	178	MOVE FORWARD UNTIL [(past over across) <landmark>] [(half_way_of end_of) street] [after <number><landmark> [left right]] [road_bend]
2	118	TAKE THE [<number>] turn [(left right)] [(before after at) <landmark>]
3	94	<landmark> IS LOCATED [left right ahead] [(at next_to left_of right_of in_front_of past behind on opposite near) < landmark >] [(half_way_of end_of beginning_of across) street] [between <landmark> and <landmark>] [on <number> turning (left right)]
4	49	GO (before after to) <landmark>
5	32	GO ROUND ROUNDABOUT [left right] [(after before at) <landmark>]
6	27	TAKE THE <number> EXIT [(before after at) <landmark>]
7	9	FOLLOW KNOWN ROUTE TO <landmark> UNTIL (before after at) <landmark>
8	3	STATIONARY TURN [left right around] [at from <landmark>]
9	1	TAKE THE ROAD in_front
10	1	PARK AT <location>
11	1	CROSS ROAD
12	1	EXIT [car_park park]

Table 1. Primitive navigation procedures found in the route descriptions collected from groups A and C. Procedure 3 is used by most subjects to indicate the last leg of a route, when the goal is in sight.

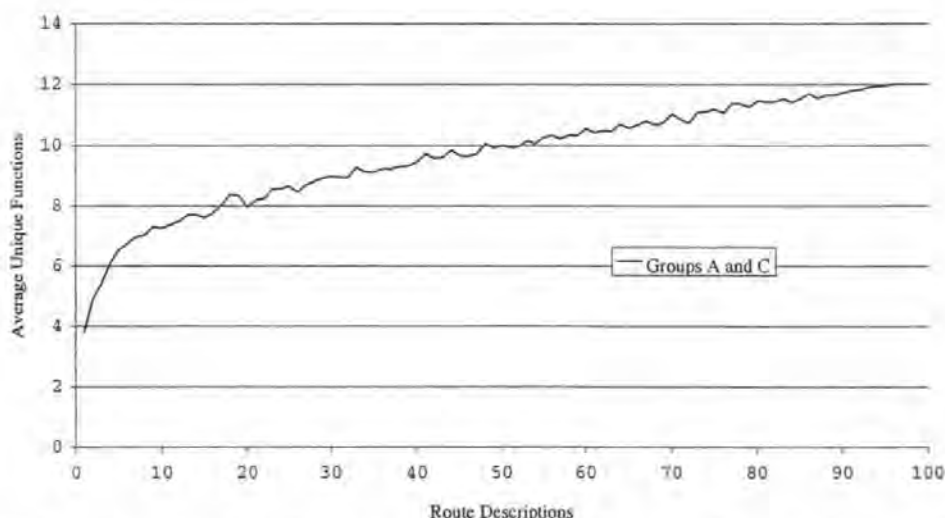


Figure 5. Average number of unique procedures as a function of the number of collected route instructions. The curve is obtained by averaging over 96 sets comprising a random selection of n route descriptions. The number n is shown on the x-axis of the graph. The slope of the curve indicates that, on average, one new function will be added to the functional lexicon for every 25 additional route instructions collected.

Figure 5 shows that the number of distinct procedures is increasing with the number of sampled instructions, but at a rate much smaller than the number of distinct words reported in (Bugmann et al., 2001). Here we discover on average one new procedure for every 25 route instructions, while with words, we discovered in average one new word for each instruction. New procedures typically are the least frequent in table 3.

6. Practical Implications

Teaching a route to a robot using natural language is an application of a more general instruction-based learning methodology. The corpus-based approach described here aims at providing users with the possibility of using unconstrained speech, whilst creating an efficient natural language processing system using a restricted lexicon. It is found that the functional vocabulary is small, containing only 12 primitives (although that number may vary with the annotation method). From a roboticist's point of view, route navigation could probably be achieved with a smaller number of primitives. However, when accepting spontaneous speech, a wider variety of functions must be expected.

An important finding is that the functional vocabulary is not closed. Hence, at some point in the robot's life, the user will have to teach it new primitives (e.g. "cross the road") or reformulate its instructions. To enable learning, the robot must possess a larger set of primitives, which correspond to lower level robot actions. For instance, the user may wish to refer to a number of wheel turns in its instruction. An example of such instructions is found in (Seabra Lopes, 2000). With our approach, this would require the collection of a new corpus to determine the necessary additional primitive procedures. Another solution may lie in an appropriate dialogue management to suggest a reformulation of the instruction. It is expected that with the corpus-based method used here, the frequency of such "repair dialogues" will be minimised. An open question is the detection of new functions in the user's utterance, as the lexicon may not contain the required vocabulary.

The approach to robot control described may be seen as an attempt to integrate the good properties of Behaviour-based control (Brooks, 1991) and classical AI. Behaviour-based control is an

effective method for designing low-level primitives that can cope with real-world uncertainties, and AI has developed effective tools for symbol manipulation and reasoning. However, the system differs in several ways from both methods. Here, the corpus defines what symbols and primitives to use. Consequently, some of the primitives are rather complex functions, involving representations of the environment and planning. These are not always compatible with the representation-less philosophy of behaviour-based systems. On the AI side, the system does not use the full range of reasoning capabilities offered by systems such as SOAR. There are no other aims in symbolic processing than verifying the consistency of instructions, and the construction of new procedure specifications. In particular, planning at the symbolic level is not needed at this stage of the project. Instead, planning is performed by the user, and the plan is communicated to the robot using natural language. This limits the autonomy of the robot, but also improves the safety of its use, as unpredictable behaviour is limited.

Other hybrid architectures integrating Behaviour-based systems and AI have been investigated as possible solutions to the symbol-grounding problem (Malcom 1995, MacDorman 1999, Tani 1996, Toshihiro et al. 1999). This problem is one of maintaining the coherence between representations used to reflect upon actions and events, and the stream of sensory information produced by a dynamic environment (Harnad, 1990). This problem can be avoided if the reasoning process itself depends in some way on its relation to the world, or, in other words, if the development of the internal categories and their transformations depends on external interactions. It has been proposed that truly sentient robots should have learning abilities such that dynamically changing external events and results of own action are allowed to constrain abstract reasoning.

The system developed by (Malcom, 1995) operated in a relatively well-ordered and predictable world in which complex tasks had to be achieved. But since the symbol system could only operate under internal syntactic constraints the grounding problem was not really addressed. In the system suggested by (MacDorman, 1996; Tani, 1996) the development of the internal categories and their transformations depended on external interactions. However, there was no human interaction and the grounding could not be modified by a non-experienced user. Whereas in (Matsui et al. 1999), the system could learn new actions through natural language dialogues but only while the robot was performing them (i.e. it could only learn a new route from A to B while it was actually moving from A to B and dialoguing with the user).

In the IBL system described here, learning operates purely at the symbolic level, hence it can be done prior to performance. The ability to predict future states enables to engage in a verification dialogue before execution errors occur. If environmental conditions change such that an instruction is not valid anymore, this can be detected from the mismatch between the expected result and the actual one. Learning however is not autonomous. The system requires interaction with a human user to learn new symbols and their meaning. This simplifies the design of the robot due to the transfer of part of the cognitive load to the user. Future experiment will reveal if this approach results in effective and socially acceptable helper robots.

Acknowledgement: This work is supported by EPSRC grants GR/M90023 and GR/M90160. The authors are grateful to Angelo Cangelosi and Kenny Coventry for enlightening discussions.

References:

- Blackburn P., Bos J., Kohlhasse M. and de Nivelles H. (1999). Inference and Computational Semantics. In: Third International Workshop on Computational Semantics (IWCS-3), Tilburg, The Netherlands.
- Bischoff, R. Jain T. (1999). Natural Communication and Interaction with Humanoid Robots. Second International Symposium on Humanoid Robots. Tokyo, Japan, October 1999, pp. 121-128.
- Brooks, R. A. (1991) Intelligence without representation, *Artificial Intelligence*, 47, 139-159

- Bugmann G., Lauria S., Kyriacou T., Klein E., Bos J. and Coventry K. (2001) "Using Verbal Instruction for Route Learning", Proc. of 3rd British Conference on Autonomous Mobile Robots and Autonomous Systems: Towards Intelligent Mobile Robots (TIMR'2001), Manchester, 5 April.
- Cangelosi A., Harnad S. (2001) The adaptive advantage of symbolic theft over sensorimotor toil: Grounding language in perceptual categories. *Evolution Communication*, (in press)
- Crangle C. and Suppes P. (1994) Language and Learning for Robots, CSLI Lecture notes No. 41, Centre for the Study of Language and Communication, Stanford, CA.
- Denis M. (1997) "The description of routes: A cognitive approach to the production of spatial discourse", *CPC*, 16:4, pp.409-458.
- Harnad, S. (1990) The Symbol Grounding Problem. *Physica D* 42: 335-346.
- Huffman S.B. and Laird J.E. (1995) "Flexibly Instructable Agents", *Journal of Artificial Intelligence Research*, 3, pp. 271-324.
- Inaba M., Kagami S., Kanehiro F., Hoshino Y., Inoue H. (2000) "A platform for robotics research based on the remote-brained robot approach", *International Journal of Robotics Research*, 19:10, pp. 933-954.
- Kamp H. and Reyle U. (1993): *From Discourse to Logic*. Kluwer.
- Laird J.E., Newell A. and Rosenbloom P.S. (1987) "Soar: An architecture for general Intelligence" *Artificial Intelligence*, 33:1, pp.1-64.
- Malcom C. M. (1995), The SOMASS system: a hybrid symbolic and behaviour-based system to plan and execute assemblies by robot. In J. Hallam, et al. (Eds), *Hybrid problems and Hybrid solutions* pp 157-168. Oxford: ISO-press.
- MacDorman, K. F. (1999). Grounding symbols through sensorimotor integration. *Journal of the Robotics Society of Japan*, 17(1), 20-24.
- Seabra Lopes, L. and A.J.S. Teixeira (2000) Human-Robot Interaction through Spoken Language Dialogue, *Proceedings IEEE/RSJ International Conf. on Intelligent Robots and Systems*, Japan.
- Tani J. (1996) Model based learning for mobile robot navigation from the dynamical system perspective 1996 *IEEE Trans. Sys. Man Cybernetics*, Part B, 26:3, pp 421- 436
- Traum, D., J. Bos, R. Cooper, S. Larsson, I.Lewin, C. Matheson and M. Poesio (1999): A model of dialogue moves and information state revision. *Trindi Report D2.1*. Available from <http://www.ling.gu.se/research/projects/trindi>

Using Verbal Instructions for Route Learning: Instruction Analysis.

Guido Bugmann, Stanislao Lauria, Theocharis Kyriacou, Ewan Klein*, Johan Bos*, Kenny Coventry[†]

Centre for Neural and Adaptive Systems, School of Computing, University of Plymouth

[†]Department of Psychology, University of Plymouth,
Drake Circus, Plymouth PL4 8AA, United Kingdom

*Institute for Communicating and Collaborative Systems, Division of Informatics, University of Edinburgh, 2 Buccleuch Place, Edinburgh EH8 9LW, Scotland, United Kingdom
<http://www.tech.plym.ac.uk/soc/staff/guidbugm/ibl/index.html>

5/4/2001

Abstract

Future domestic robots will need to adapt to the special needs of their users and to their environment. It is likely that programming by natural language will be a key method enabling computer language-naïve users to instruct their robots. This paper describes initial steps and considerations towards the design of Instruction-Based Learning (IBL) systems. The proposed methodology is to be tested in the restricted domain of route instructions with real speech input and a real mobile robot using vision for navigation. Users will use unconstrained speech within a restricted domain-specific lexicon determined by analysing a corpus of route instructions. This will maximise speech recognition performance. The robot will possess an appropriate set of primitive procedures that correspond to procedures found in route instructions. Based on 96 route instructions, it is found that the task vocabulary contains approximately 270 words, but is not closed. It increases at an average rate of one new word for every new route instruction. However, there are large inter-individual differences and 58% of instructions contain no out-of-vocabulary words. The functional vocabulary is found to include 12 different procedures, and is also not closed. It increases at an average rate of one new procedure for every 25 instructions.

1. Introduction

Future domestic robots will be required to perform tasks that manufacturers cannot pre-program. For instance, making tea to the taste of the user, or fetching a book in a specified room. Such tasks require knowledge about the layout of the home of the user and on his/her preferences. Other tasks will need to be performed at a given time of the day or if certain conditions are met. In this paper we concentrate on navigation tasks, for instance the ones performed by an autonomous wheelchair carrying his/her user to a desired destination. The problem addressed is here the one of how a user, with no programming skills, could interact with the robot to modify its internal program.

The question of how robots could learn from their users has been investigated so far along two main routes, learning by imitation [Billard et al., 1998] and learning by reinforcement [e.g. Perez-Urbe and Hirsbrunner, 2000]. However, both methods have limited scope. For instance, learning by imitation does not enable the acquisition of rules such as "IF-THEN". Learning by reinforcement is a lengthy process that is best used for refining low-level motor control, but becomes impractical for complex tasks. Both methods do not readily generate knowledge representations that the user can interrogate. This paper focuses on another form of learning, by verbal instruction, that has proven its effectiveness in human learning [Bloom, 1984], but has received relatively little attention in robotics.

Previous work on verbal communication with robots has mainly focused on issuing *commands*, i.e. activating pre-programmed procedures using a limited vocabulary (e.g. IJCAI'95 office navigation contest). Only a few research groups have considered *learning*, i.e. the stable and reusable acquisition of new procedural knowledge. [Huffman & Laird, 1995] used textual input into a simulation of a manipulator with a discrete state and action space. [Crangle and Suppes, 1994] used voice input to teach displacements within a room and

[†] Proc. TIMR 01 – Towards Intelligent Mobile Robots, Manchester 2001.

Technical Report Series, Department of Computer Science, Manchester University, ISSN 1361 – 6161.
Report number UMC-01-4-1. <http://www.cs.man.ac.uk/csonly/cstechrep/titles01.html>

mathematical operations, but with no reusability. In [Torrance, 1995], textual input was used to build a graph representation of spatial knowledge. This system was brittle due to place recognition from odometric data and use of IR sensors for reactive motion control. Knowledge acquisition was concurrent with navigation, not prior to it.

Key criteria for the design of practical instruction-based learning (IBL) systems are seen here as: 1. Handling of natural speech, with its variations, underspecifications and errors in speech recognition; 2. Handling real world continuous state spaces with uncertainty and noise; 3. Incremental learning, with new instructions reusing previously taught procedures; 4. User-friendly and effective dialogue management by the robot.

Satisfying these criteria imposes numerous inter-linked constraints on the system architecture, the robot control design and the natural language processing component of an IBL system. In order to explore these effects, a simple route learning task has been selected, using real speech input and a robot using vision to execute the instructed route. The interaction scenario and the architecture of the proposed IBL system are outlined in section 2. Speech recognition, natural language understanding and dialogue management are described in section 3. The miniature experimental environment and the robot are described in section 4. As a first step toward designing a system that can handle unconstrained speech we have collected a corpus of data on unconstrained instructions given by users. The corpus collection procedure and the analysis of the data are described in section 5. Analysis is done along two lines, i) specifying the lexicon, **grammar rules, dialogue acts and ontology (really ???)** and ii) determining a list of primitives navigation procedures referred to in the instructions. The implications of the findings are discussed in section 6.

2. IBL concept: Interaction Scenario and Architecture.

2.1 Concept

The aim of the IBL project is to develop a system that converts verbal instructions into internal program code. Procedures learnt from the user become part of a pool of procedures that can be reused to learn more and more complex procedures. Hence, the robot becomes able to execute increasingly complex tasks.

To evaluate the potential and limitations of IBL, a real-world instructions task should be used, that is simple enough to be realisable, and generic enough to warrant conclusions that hold also for other task domains. To be generic, the task should require the learning of the *three fundamental* components of computer programs: Sequence, Selection and Repetition. These components are found in route instructions. First, a route is a sequence of route-segments. Secondly, although decisions are rarely part of route instructions (e.g. "if the road is blocked take this other one"), they are implicit in the execution of all segments. For instance, "take the first left" is to be translated in programming terms into "IF you are not at the intersection yet, THEN keep moving towards it. ELSE: do the left turn". Thirdly, explicit repetitions do occur in route instructions ("turn left 2 times"), and are also implicit in all segments which, as in the example above, requires a repetition of procedures ("keep moving until ...").

In terms of user-robot interaction, a typical learning process would start with the user asking the robot to perform a given task. If the robot lacked information about the task, it would ask for clarification or may ask the user to explain the task step by step. The ensuing dialogue constitutes the core of "instruction-based learning".

Due to the nature of the users, a requirement of the project is the use of unconstrained speech. In terms of vocabulary, this means that the user is allowed to use the words that are natural to him. However, using a restricted lexicon improves the performance of a speech recogniser. It is planned here to use a restricted lexicon that matches the one naturally used by users, so as to allow unconstrained speech. In terms of navigation procedures, the user is allowed to construct route using functional primitives that are natural to him. It is planned to provide the robot with pre-programmed counterparts to these primitives, to enable a seamless conversion of verbal route instructions into programs. Corpus analysis along these lines is described in section 5. Another dimension of unconstrained speech is dialogue management. The user should be free to initiate or terminate dialogue moves at will. For instance, the user should be able at any time to interrupt a process in the robot, by issuing the command "stop", or leave a learning dialogue to issue a new command. This requires a flexible dialogue management but also a purpose-designed system architecture.

2.2 System Architecture

The architecture is comprised of several functional processing modules (figure 1). These are divided into two major units: the Dialogue Manager (DM) and the Robot Manager (RM).

The DM and the RM are designed as two different processes based on asynchronous communication protocols. These processes run concurrently on different processors. In this way, the system can handle, at the same time, both the dialogue aspects of an incoming request from the user (i.e. speech recognition and semantic analysis, or detection of a "stop" command) and the execution of a previous user request (i.e. check if the request is in the system knowledge domain, and execute vision-based navigation procedures).

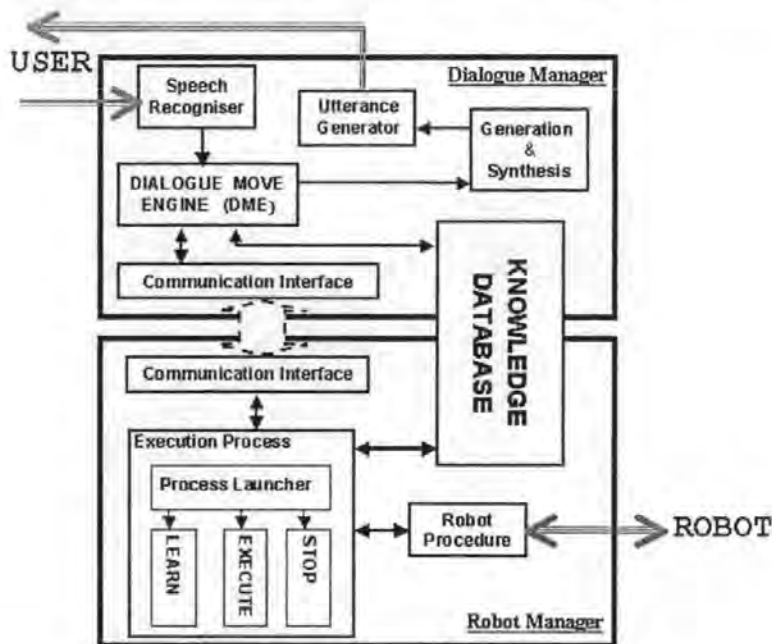


Figure 1. IBL systems architecture.

Two aspects are essential with this concurrent-processes approach. Firstly, to define an appropriate protocol between the two processes. Secondly, to define an appropriate architecture for the RM and DM allowing the two processes to both communicate with each other while performing other tasks. At present a communication protocol based on sockets and context-tagged messages is evaluated.

Moreover, the system must also dynamically adapt itself to new user requests or to new internal changes, by being able to temporarily suspend or permanently interrupt some previous activity. For example the user may want to prevent the robot crashing against a wall and must therefore be able to stop the robot while the robot is driving towards the wall. Hence, the importance of a concurrent approach where the systems constantly listens to the user while performing other tasks and at the same time be able to adjust the task if necessary.

The Dialogue Manager is a bi-directional interface between the Robot Manager and the user, either converting speech input into a semantic representation, or converting requests from the Robot Manager into dialogues with the user. Its components are run as different processes communicating with each other via a blackboard architecture.

The RM must concurrently listen/send requests from/to the DM and try to execute them. For this reason a multi-threads approach has been used. The communication interface is a process that only launches a message evaluation thread "Execution Process" and resumes listening to the DM. The execution process then starts an appropriate thread for executing a command, or places a tagged message on a message board if it is a part of a dialogue in a specific thread, e.g. learning a route. The characteristic of this approach is that all processes in the RM are sharing common memory stack so that threads can be started and paused, depending on the user's input.

The Robot Manager is written using the scripting language Python¹ and C. An important feature of scripting languages is their ability to write their own code. For instance, a route instruction given by the user will be saved by the Robot Manager as a Python script that then becomes part of the procedure set available to the robot for execution of for future learning.

¹ <http://www.python.org>

3 Natural Language processing and dialogue management.

3.1 The Dialogue Move Engine

The ongoing dialogue between user and robot is represented by a discourse representation structure (DRS) proposed by Discourse Representation Theory (Kamp & Reyle 1993). New utterances yield DRSs (see section 3.2 below), which update the DRS of the dialogue, following the recent information state approach to dialogue processing (Traum et al. 1999). Context-sensitive expressions (such as pronouns and presupposition triggers) are resolved with respect to this DRS. Finally, utterances of the robot are realized by generating prosodically annotated strings from DRSs and feeding these to a synthesizer.

Using semantic representations for modeling the dialogue is motivated by the need to perform inferences in order to let the robot make "intelligent" responses. Inferences are required to resolve ambiguities present in the user's input (being of scopal, referential, or lexical nature), to detect the speech act associated to an utterance (e.g., did the user answer a question or has a new issue been raised?), to plan the next utterance or action, and to generate naturally sounding utterances (e.g., by distinguishing old from new information within an utterance). Inference are actually carried out using off-the-shelf theorem provers, by translating DRSs to first-order logic (cf. Blackburn et al. 1999).

3.2. Speech Recognition

Speech recognition and semantic construction are integrated into one component. The basic idea is to use off-the-shelf speech recognition, and to use a grammar that is linguistically motivated and domain independent. The grammar not only consists of rules that determine the syntactic structure of utterances, but also features semantic rules that specify how semantic representations (underspecified DRSs) are built in a compositional way.

The current prototype implementation uses Nuance² tools for speech recognition. The initial grammar is a unification-based phrase structure grammar, which is compiled into GSL, the Grammar Specification Language supported by Nuance's technology. This compilation involves removing left-recursive rules within the grammar, as well as replacing features and their possible values for syntactic category symbols, as GSL neither support left-recursive rules nor a feature-value system. As a consequence, the language models for the speech recognition are huge, but still feasible for small lexicons (a few hundred words in the case of IBL).

The semantic operations are compiled-out in GSL as well, and each word in the lexicon is associated with a semantic representation. As a result, the output of the speech recognizer is directly a semantic representation, in our case an underspecified DRS, and another step of processing (such as parsing and semantic construction) is not required. Hence, by compiling our linguistic grammar into GSL, we short-cut the parsing and semantic construction process into a single component.

4 Experimental Environment and task.

The environment is a miniature town covering an area of size 170cm x 120cm (figure 2). The robot is a modified RobotFootball robot³ with a 8cm x 8cm base (figure 3A). The robot carries a CCD colour TV camera⁴ (628 (H) x 582 (V) pixels) and a TV VHF transmitter. Images are to be processed by a PC that acquires them via with a TV capture card⁵ (an example of such image is shown in figure 3B). The PC will then sends motion commands by FM radio to the robot. During corpus collection, the PC is also used to record instructions given by subjects.

The advantage of a miniature environment is the ability to build a complex route structure in the limited space of a laboratory. The design is as realistic as possible, to enable subjects to use natural expressions for the outdoor real-size environment. Buildings have signs taken from real life to indicate given shops or utilities such as the post-office. However, the environment lacks some elements such as traffic lights that may normally be used in route instructions. Hence the collected corpus is likely to be more restricted than for outdoor route instructions.

The advantage of using a robot with a remote-brain architecture [Inaba et al., 2000] is that the robot does not require huge on-board computing and hence can be small, fitting the dimensions of the environment.

² <http://www.nuance.com>

³ Provided by Merlin Systems (<http://www.merlinsystemscorp.com/>)

⁴ Provided by Allthings Sales and Services (<http://www.allthings.com.au/>)

⁵ TV Card: Hauppauge WinTV GO

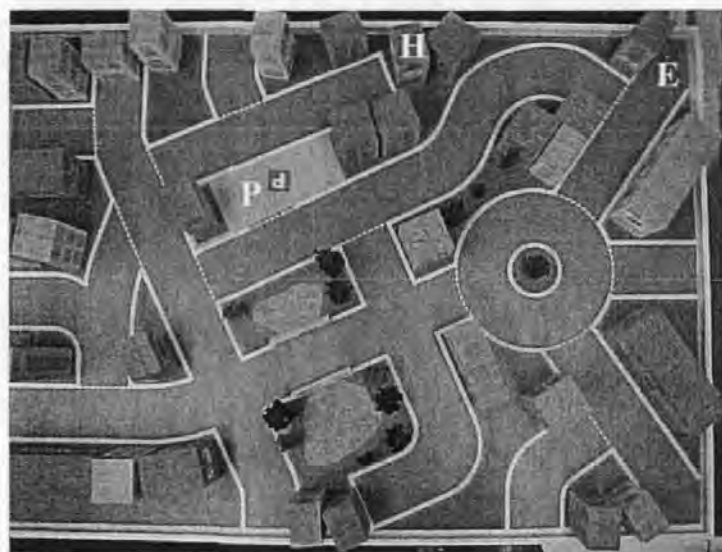


Figure 2. Miniature town in which a robot will navigate according to route instructions given by users.

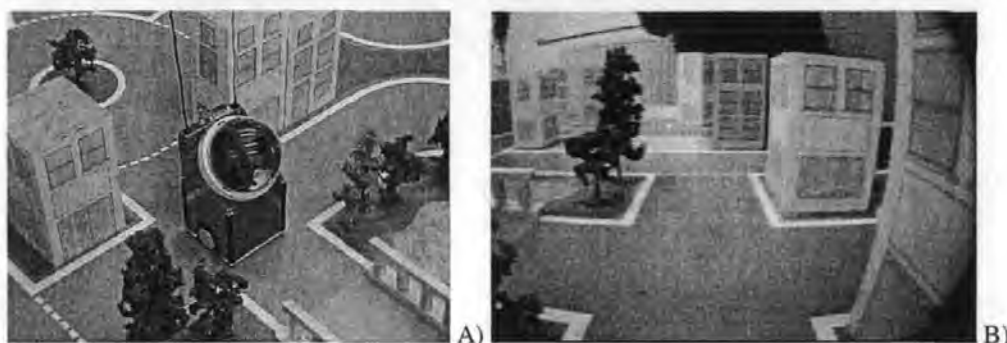


Figure 3. A. Miniature robot (base 8cm x 8cm). B. View from the on-board camera.

5 Corpus collection and data analysis

5.1 Data collection

To collect linguistic and functional data specific to route learning, 24 subjects were recorded as they gave route instructions to the robot in the environment. Subjects were divided into three groups of 8. The first two groups (A and B) were told that the robot was remote-controlled and that, at a later date, a human operator would use their instructions to drive the robot to its destination. It was specified that the human operator would be located in another room, seeing only the image from the wireless on-board video camera. This was specified to induce the subject into using spatial references accessible by the future vision software. Subjects were also told to reuse previously defined routes whenever possible, instead of re-explaining them in detail. Each subject had 6 routes to describe among which 3 were "short" and 3 were "long". The long routes included a short one, so that users could refer to the short one when describing the long one, instead of re-describing all segments of the short one. This was to reveal the type of expressions used by users to link taught procedures with primitive ones. Groups A and B received the same routes to describe, but with the sequence of "short" and "long" route inverted. This would reveal the difference between a fully detailed route, and a route with reference to a short route inserted. Again the question is the one of how procedure insertion is handled by subjects.

The first two groups (A and B) used totally unconstrained speech, to provide a performance baseline. It is assumed that a robot that can understand these instructions as well as a human operator would represent the ideal standard. Each subject described 6 routes having the same starting point and six different destinations. Starting points were changed after every two subjects. A total of 96 route descriptions were collected.

A third group of 8 subjects (C) had the same routes to describe as group A, but were forced into a simplified dialogue with an operator to produce shorter chunks of descriptions. It is known that it is very difficult for NL

processing tools to correctly segment an uninterrupted stream of words into sentences. Therefore, corpus C may be more representative of utterances in the eventual user-robot dialogue. Subjects in this group were told that the operator next door was taking notes. A researcher pretended to do so and interrupted the subjects (using a microphone) when they used chunks that were too long. He acted as if he understood all the instructions and did not initiate repair dialogues. The analysis performed so far covers group A and group C. Table 1 shows an example of the same two "short" and "long" routes instructed by a subject in group A and a subject in group C. The instructions were transcribed in XML using the Transcriber⁶ software.

Monologue	
Short	User: okay take your first right and continue down the street past Derry's past Safeway and your parking lot the car park will be on your right
Long	User: okay once you pass the car park er take your first right and then again take your first right and the hospital will be right in front of you
Dialogue	
Short	Wizard: <i>could you tell me how to get to the car park please</i>
	User: okay you'll take the first right from where you are now past Derry's then Safeway
	Wizard: <i>yes</i>
	User: you'll pass another road on the left and the car park's on the right from there
Long	Wizard: <i>thank you</i>
	Wizard: <i>could you tell me how to go to the hospital please</i>
	User: okay you need to go back towards the car park
	Wizard: <i>yes</i>
	User: past the car park take the first right
	Wizard: <i>i'm sorry after i pass the car park</i>
	User: you take the right after the car park
	Wizard: <i>yes</i>
	User: and then another right again
	Wizard: <i>yes</i>
	User: and you'll be moving towards the hospital on the end of that road
	Wizard: <i>thank you</i>

Table 1: Example of instructions for a short route from E to P and a long route from E to H (see figure 2) given under monologue condition (group A) and dialogue condition (group C). The wizard is a human operator mimicking verbal feedback that could be given by the robot.

5.2 Analysis of the Task Vocabulary

To provide an initial estimate of the task vocabulary, the data from group A and C were merged. The number of distinct words was counted in the set of 96 instructions given. Morphology was not taken into account, i.e. "travels" and "travel" were counted as different words. The vocabulary of the users was found to contain 269 different words, from a total of 4020 word in the combined corpus A and C. The most frequent words were found up to 491 times and 67 words were used only once (table 2), i.e. only one subject used a particular word in a single route instruction.

To determine if the corpus collection had led to a complete sampling of the task vocabulary, the average number of distinct words was plotted as a function of the number of collected instructions. Figure 4 shows that the number of distinct words is still rising at the end of the curve, indicating that more new words would be found if more route instructions were collected. This behaviour is similar in other task domains [Zue, 1997]. The slope of the curve in figure 4 indicates that a new user might say on average one out-of-vocabulary word in each instruction.

To determine what type of new word might be expected, each route instruction was compared to the corpus of all other instructions. The result is that the new words are all among the 69 least frequently used words. Table 2 shows that these are not necessarily "unusual" words. The question of how the understanding of an instruction might be affected by the absence of such words from the vocabulary will be investigated further.

The dialogue group tended to use less distinct words (figure 3) and tended to produce less "out-of-vocabulary" words (table 3). Therefore, future experiments may reveal an improved speech recognition performance in dialogue conditions.

⁶ <http://www ldc.upenn.edu/mirror/Transcriber/>

Most Frequent		Least frequent
Word	Count	Order, here, doors, onto, robot, well, center, moving, moment, thank you, lot, park's, actually, its, carrying, able, tesco, sharp, turned, leave, arrive, branch, taking, while, crossing, hundred, taken, double, bears, area, ninety, instruct, turnings, feel, apologize, thirty, or, place, amount, leaving, time, blocks, diagonally, there's, say, currently, what, reaching, travels, some, bear, bends, says, means, quadrangle, exits, like, forty-five, set, now, half, five, very, only, uh-huh, certainly, tesco's, paper, quarters, soon, move
The	491	
And	166	
On	162	
You	125	
To	123	
Take	117	
Left	114	
Right	108	
Go	97	
Your	92	

Table 2: Most frequent and least frequent user word in the corpus. The least frequent words were found only once in 96 route descriptions.

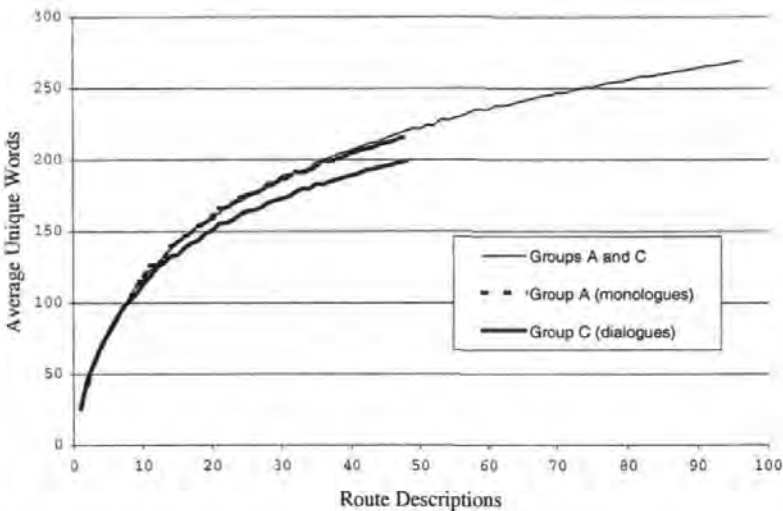


Figure 4. Number of distinct words discovered in the corpus as the number of instruction samples increases. The long line is for group A and C pooled. The shorter lines are for groups A and C taken in isolation. Curves are obtained by averaging over 48 random sets comprising an increasing number of sample instructions. The slope of the long curve indicates that, on average, one new word is added to the vocabulary for every additional route instruction collected.

Another way to look at the problem of out-of-vocabulary words, is to determine how many instructions actually contain new words. The result is that 58% of instructions had no new words (60% of those in group A, and 56% of those in group C) implying that more than half of the instructions would be recognised perfectly by a speech recognition system based on the current vocabulary. Among the remaining 42% of instructions, 65 % had only one new word and 35% had between 2 and 6 new words.

Subject:	2	3	4	5	6	7	8	9	1	10	11	12	13	14	15	16
Group:	d	d	d	d	d	d	d	d	m	m	m	m	m	m	m	m
Nb. New words	2	3	10	5	5	0	0	2	2	3	3	1	1	19	10	3

Table 3: Number of new words used by each subject in their 6 route instructions. The group of the subjects is indicated by m = monologue (A) or d = dialogue (C).

There was also a significant inter-subject variability (table 3). Some subjects used less than two new words in their 6 descriptions, while others were blessed with a particularly rich vocabulary and produced several new words in each one of their instruction. This is not necessarily a blessing when it comes to interacting with a robot. However, some of the "new" words counted here were morphological variations of known words, and a speech recognizer would have recognized them. In general, not more than one sentence per instruction contains a truly out-of-vocabulary word. Hence, it is expected that situations where repair is needed will not be unbearably frequent, but they are likely to affect most users.

5.3 Analysis of the Functional Vocabulary.

The functional vocabulary is a list of primitive navigation procedures found in descriptions. The initial annotation of instructions in terms of procedures, as reported here, is somehow subjective, and influenced by two considerations. 1. The defined primitives will eventually be produced as C-Programs. It was hoped that only a few generic procedures would have to be written. Therefore, the corpus has been transcribed into rather general procedures characterised by several parameters (table 4). 2. An important issue is knowledge representation. A route is to be represented as a graph, constituted of a continuous chain of primitives. For that purpose, all primitives must be consistent with a standard " $S_i A_{ij} S_j$ " representation (Initial state S_i , final state S_j and linking action A_{ij}). For a route description to be accepted as complete and executable, the initial state of each procedure must correspond to the final state of the previous one.

Subjects however rarely specified explicitly the starting point and it was assumed that the system would need to be able to infer the starting point from previous action specifications. Therefore, procedures without starting points were considered complete, and were annotated as such. The specifications of primitive procedures is likely to evolve during the project.

This methodology differs from the one used in [Denis, 1997]. Denis converted each instruction into a propositional format. For instance "You will arrive at a wooden bridge that you must cross" is converted into:

1. ARRIVE AT(YOU, BRIDGE); 2. WOODEN(BRIDGE); 3. CROSS(YOU, BRIDGE)

Statements in this format were grouped into four classes: "prescribing action" (e.g. "turn left"), "prescribing actions with reference to a landmark" (e.g. number 3 above), "introducing landmarks" (e.g. "there is a tree to your left"), "describing landmarks" (e.g. number 2 above) and "Commentaries" (e.g. "the route will take about 5 min.").

In our analysis, there are no statements describing landmarks, as these are included in the termination points and there are no actions without reference to landmarks, as robot procedures need a defined termination point. Even when a subject specified a non-terminated action, such as "keep going", it was classified as "MOVE FORWARD UNTIL", assuming that a termination point would be inferred from the next specified action. The list of actions found in the descriptions of groups A and C is given in table 4.

	Count	Primitive Procedures
1	178	MOVE FORWARD UNTIL [(past over across) <landmark>] [(half_way_of end_of) street] [after <number><landmark> [left right]] [road_bend]
2	118	TAKE THE [<number>] turn [(left right)] [(before after at) <landmark>]
3	94	<landmark> IS LOCATED [left right ahead] [(at next_to left_of right_of in_front_of past behind on opposite near) <landmark>] [(half_way_of end_of beginning_of across) street] [between <landmark> and <landmark>] [on <number> turning (left right)]
4	49	GO (before after to) <landmark>
5	32	GO ROUND ROUNDABOUT [left right] [(after before at) <landmark>]
6	27	TAKE THE <number> EXIT [(before after at) <landmark>]
7	9	FOLLOW KNOWN ROUTE TO <landmark> UNTIL (before after at) <landmark>
8	3	STATIONARY TURN [left right around] [at from <landmark>]
9	1	TAKE THE ROAD in_front
10	1	PARK AT <location>
11	1	CROSS ROAD
12	1	EXIT [car_park park]

Table 4. Primitive navigation procedures found in the route descriptions collected from groups A and C. Procedure 3 is used by most subjects to indicate the last leg of the route, when the goal is in sight.

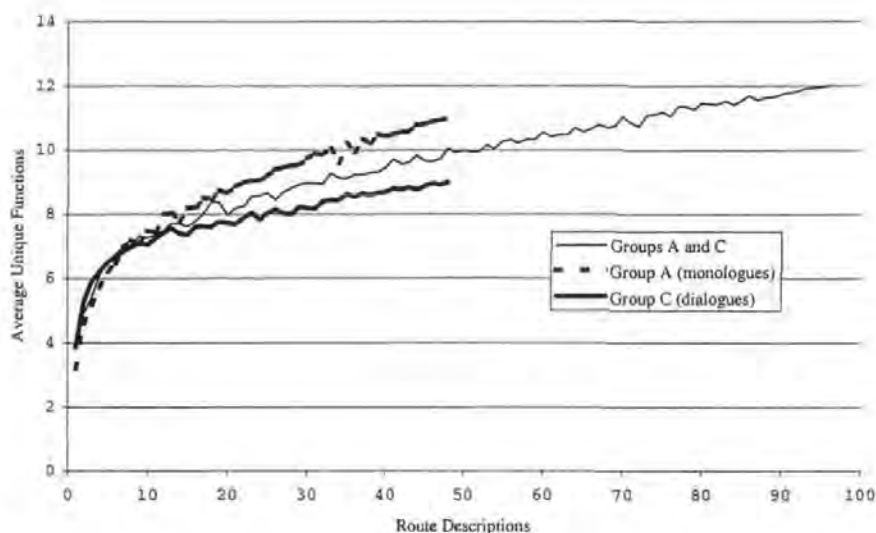


Figure 5. Average number of unique procedures as a function of the number of collected route instructions (Curves calculated as in figure 4.).

Figure 5 shows that the number of distinct procedures is increasing with the number of sampled instructions, but at a rate much smaller than the number of distinct words seen in the previous section. Here we discover on average one new procedure for every 25 route instructions, while with words, we discovered in average one new word for each instruction (figure 4). New procedures typically are the least frequent in table 4.

6 Discussion

Teaching a route to a robot using natural language is only one application of a more general instruction-based learning methodology. The approach described here aims at providing users with the possibility of using unconstrained speech, whilst creating an efficient natural language processing system using a restricted lexicon. The preliminary analysis of the lexicon shows however that out-of vocabulary errors are to be expected. This is a well known problem in the domain of speech recognition, but it is a rather new observations on the functional side. From a roboticist's point of view, route navigation can be achieved with a rather small number of primitives. However, in spontaneous speech, a wider variety of functions must be expected.

We are attempting to give the user the freedom to reply or not to reply to a query, to control when given dialogues are to take place and to interrupt the robot at will. This created interesting constraints on the design of the system's architecture. In particular it calls for a solution using multi-threads with shared memory. Experiments will reveal how effective this solution is.

The results in section 5.2 indicate that when working with a limited vocabulary, it is unavoidable that unknown words are going to be used by users. This is the price to pay for having a reasonably robust speaker independent recogniser. In current speech recognition systems, such words would either be ignored or replaced with the most likely word in the lexicon. Limited research has gone into speech recognisers that would signal that some sound is likely to be a new word and learn the new word [Zue, 1997; Asadi et al, 1991]. When working with large vocabularies, out-of-vocabulary words are less likely to occur, but word recognition errors then occur due to the larger search space. Thus in any case, error spotting and repair mechanisms need to be built into an IBL system.

Word recognition errors can be revealed in the DM when they cause ungrammatical sentences. The RM can also detect word errors when they lead to unknown tasks being requested. The last stage of error spotting is to ask the user to confirm a task just before execution. Overall, error spotting and repair is not a simple problem, and experiments will be needed to understand how best to approach it.

The functional vocabulary is rather small. It includes navigation procedures and cognitive⁷ procedures. An important finding is that the functional vocabulary is not closed. Hence, at some point in the robot's life, the

⁷ "cognitive" denotes here actions that manipulate knowledge as opposed to actions that move the robot.

user will have to teach it new primitives (e.g. "cross the road"). Future work will have to determine what additional set of primitives are needed by the robot to understand instructions explaining how to "cross the road". Another issue is the identification of new functions, as the lexicon may not contain the required words.

7 Conclusion

The project described in this paper is aimed at exploring IBL for a limited class of functions: routes descriptions. Hence steps were taken to pre-program all other functions necessary for constructing route descriptions. A corpus of instructions was analysed to determine the list of words that the speech recognition system should recognise. Similarly a list of primitive procedure was established to ensure that the robot would be able to execute the navigation procedures forming the instructions. However, the initial results presented here show that neither the lexicon nor primitive procedures are likely to form closed sets. Ideally, an IBL system should therefore also be capable of acquiring new words, and users should be given the possibility to teach new primitive procedures. Unfortunately, the former is beyond the capabilities of current speech recognition systems. As for learning new primitives procedures, this would require a new set of more primitive procedures to be combined via user instructions. Whether it will be possible to explore this during the project is unclear. To allow IBL to operate despite these limitations, it is likely that a crucial role will be played by dialogue management.

Acknowledgement: This work is supported by EPSRC grants GR/M90023 and GR/M90160.

References:

- Asadi A., Schwartz R. and Makhoul J. (1991) "Automatic modelling for adding new words to a large vocabulary continuous speech recognition system", Proc. ICASSP, pp. 305-308.
- Billard A., Dautenham K. and Hayes G. (1998) "Experiments on human-robot communication with Robota, an imitative learning and communication doll robot", Contribution to Workshop "Socially Situated Intelligence" at SAB98 conference, Zurich, Technical Report of Centre for Policy Modelling, Manchester Metropolitan University, CPM-98-38. (<http://www.cpm.mmu.ac.uk:80/cpmrep38.html>)
- Blackburn P., Bos J., Kohlhasse M. and de Nivelle H. (1999). Inference and Computational Semantics. In: Third International Workshop on Computational Semantics (IWCS-3), Tilburg, The Netherlands.
- Bloom B.S. (1984) "The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring", *Education Researcher*, 13:6, pp. 4-16.
- Crangle C. and Suppes P. (1994) Language and Learning for Robots, CSLI Lecture notes No. 41, Centre for the Study of Language and Communication, Stanford, CA.
- Denis M. (1997) "The description of routes: A cognitive approach to the production of spatial discourse", CPC, 16:4, pp.409-458.
- Huffman S.B. and Laird J.E. (1995) "Flexibly instructable agents", *Journal of Artificial Intelligence Research*, 3, pp.271-324.
- Inaba M., Kagami S., Kanehiro F., Hoshino Y., Inoue H. (2000) "A platform for robotics research based on the remote-brained robot approach", *International Journal of Robotics Research*, 19:10, pp. 933-954.
- Kamp H. and Reyle U. (1993): From Discourse to Logic. Kluwer.
- Perez-Urbe A. and Hirsbrunner B., "Learning and Foraging in Robot-bees", SAB2000 Proceedings Supplement Book, Meyer, Berthoz, Floreano, Roitblat and Wilson (Eds), Published by International Society for Adaptive Behavior, Honolulu (to appear). <http://www-iiuf.unifr.ch/~aperezu/robot-bees/> also <http://www-iiuf.unifr.ch/~aperezu/robotreinfo.html>
- PYTHON: <http://www.python.org>
- Torrance M.C. (1994) Natural Communication with Robots, MSc Thesis submitted to MIT Department of Electrical Engineering and Computer Science, January 28, 1994.
- Traum, D., J. Bos, R. Cooper, S. Larsson, I. Lewin, C. Matheson and M. Poesio (1999): A model of dialogue moves and information state revision. Trindi Report D2.1. Available from <http://www.ling.gu.se/research/projects/trindi>
- Young S.J. (2000) "Probabilistic Methods in Spoken Dialogue Systems", *Phil. Trans. Royal Society A*, 358: 1769, pp. 1389-1401 (<http://citeseer.nj.nec.com/386391.html>)
- Zue, V. (1997) "Conversational interfaces: Advances and challenges", In Proc. Eurospeech, pages 9-14, Rhodes, Greece. (<http://citeseer.nj.nec.com/78849.html>).
-

Instruction Based Learning: how to instruct a personal robot to find HAL.

Stanislao Lauria, Guido Bugmann¹, Theodoris Kyriacou, Ewan Klein*

Centre for Neural and Adaptive Systems, School of Computing, University of Plymouth
Drake Circus, Plymouth PL4 8AA, United Kingdom.

*Institute for Communicating and Collaborative Systems, Division of Informatics, University of Edinburgh, 2
Buccleuch Place, Edinburgh EH8 9LW, Scotland, United Kingdom.

<http://www.tech.plym.ac.uk/soc/staff/guidbugm/ibl/index.html>

Abstract

Future domestic robots will need to adapt to the special needs of their users and to their environment. Programming by natural language will be a key method enabling computer language-naïve users to instruct their robots. Its main advantages over other learning methods are speed of acquisition and ability to build high level symbolic rules into the robot. This paper describes the design of a practical system that uses unconstrained speech to teach a vision-based robot how to navigate in a miniature town. The robot knows a set of primitive navigation procedures that the user can refer to when giving route instructions.

Since the user is likely to refer to a procedure that is not pre-programmed in the robot, the system must be able to learn it. This paper investigates how to make the learning process possible. In particular, a method is proposed to fasten the choice of an initial set of primitives to the natural human speech chunking. Moreover, the use of Instruction-Based Learning (IBL) imposes a number of constraints on the design of robotics systems and knowledge representation. These issues are developed in the paper and proposed solutions described.

1. Introduction

Intelligent robots must be capable of action in reasonably complicated domains with some degree of autonomy. This requires adaptivity to a dynamic environment, ability to plan and also speed in the execution. In the case of helper robots, or domestic robots, the ability to adapt to the special needs of their users is crucial. The problem addressed here is the one of how a user could instruct the robot to perform tasks which manufacturers cannot

completely program in advance. In this case the system will not work at all if it cannot learn.

Such learning requires interaction and collaboration between the user and the robot. But, as most users are computer-language-naïve, they cannot personalise their robot using standard programming methods. Indirect methods, such as learning by reinforcement or learning by imitation, are also not appropriate for acquiring user-specific knowledge. For instance, learning by reinforcement is a lengthy process that is best used for refining low-level motor controls, but becomes impractical for complex tasks. Further, both methods do not readily generate knowledge representations that the user can interrogate.

Instruction-Based Learning (IBL), which uses unconstrained speech, has several potential advantages. Natural language can express rules and sequences of commands in a very concise way. Natural language uses symbols and syntactic rules and is well suited to interact with robot knowledge represented at the symbolic level. It has been shown that learning in robots is much more effective if it operates at the symbolic level (Cangelosi and Harnad, 2001). This is to be contrasted with the much slower learning at the level of direct sensory-motor associations.

Chunking, sequencing and repair are the aspects, related to natural language interactions, shaping the design of IBL systems discussed here. Chunking is a principle that applies to the communication of information. Chunking is meant here as the human characteristic to divide, during explanations, tasks into sub-tasks, so that all information should be presented in small 'basic' units of actions. As shown in (Miller 1956), chunking is done spontaneously by humans and consequently the system must be on the same 'wavelength' as the user in order to be successful. This means establishing for the robot the appropriate

¹ To whom correspondence should be addressed.

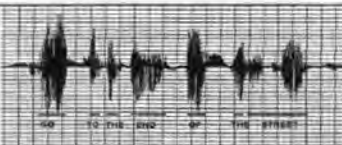
Analysis			Repair
	Speech recognition		
↓	Tagging	Go/VB to/TO the/DT end/NN of/IN the/DT street/NN	↑
↓	Syntactic Parsing	[VG go] [to to] [NG the end of the street]	↑
↓	Semantic Analysis	Robot (x), end_of_street (y), request_go (x, y)	↑
↓	Functional Mapping	Goto("end_of_street")	↑
↓	Robot program	Until found(end_of_street) follow_the_road()	↑

Table 1. From speech to action. The various steps involved in the transformation of a user command into the corresponding action are shown here.

prerequisites for the conversion of cognition, carried in chunks, into the form of procedures. In a robot involved with navigation tasks, a fundamental prerequisite is that the system must possess a set of pre-programmed procedures related to the very basic chunks used in route instruction situations. Moreover, since in the learning process the user does not express his requirement with a single chunk, the system must be able to sequence the chunks correctly. For example, in a sequence of instructions given by the user, the final state of an action may not correspond to the expected state for the next action. In this case, the system would not be able to perform its task due to the missing chunk. For this reason, it is necessary to define a proper internal knowledge representation allowing the system to detect the missing information. In this way, the system would be able to make predictions about future events so that the problem can be solved while the system is still interacting with the user.

Finally, the system not only has to pay attention to user knowledge and dialogue goals, but it also has to adapt its dialogue behaviour to current limitations of the user's cognitive processing capabilities. Assistance is then expected from the system, so that the interaction may naturally flow over the course of several dialogue turns. Moreover, a dialogue manager should take care of identifying, and recovering from, speech recognition and understanding errors.

This paper describes initial steps and considerations towards a practical realisation of an IBL system. The experimental environment is that of a miniature town in which a robot provided with video camera executes route instructions. The robot has a set of pre-programmed sensory-motor action primitives, such as "turn left" or "follow the road". The task of the user is to teach the robot new routes

by combining action primitives. That task should reveal all the constraints described above, and enable testing of the developed methodology.

In the next section the IBL architecture implications due to chunking, sequencing and repair are discussed and how the rest of this paper is organized is also specified.

2. The big picture: from verbal utterance to robot action

With IBL, the system must convert verbal instructions given by the user into procedures containing internal program code controlling the robot sensors/actuators. It is during the learning process that such procedures are created and become part of a pool of procedures that can then be reused to learn more and more complex procedures. In this way the robot becomes able to execute increasingly complex tasks based on a set of pre-programmed primitives.

The closer the correspondence between primitives and chunks expressing the very basic actions (such as "turn left") is, the less difficult the learning is, since, in this way, the interaction between the user and system is kept to the minimum. For this reason, it is necessary to select these primitives that corresponds as closely as possible to the action expressed in the chunks.

Then, there is the problem of handling the chunks. In table 1, an example is given showing the various steps necessary to transform a user chunk into a robot action. First, the robot must be able to perform some speech recognition tasks in order to convert speech into text. After that, some syntactic parsing and semantic analysis is carried out. Then at the functional mapping level, the system must be able to transform the user utterance into internal

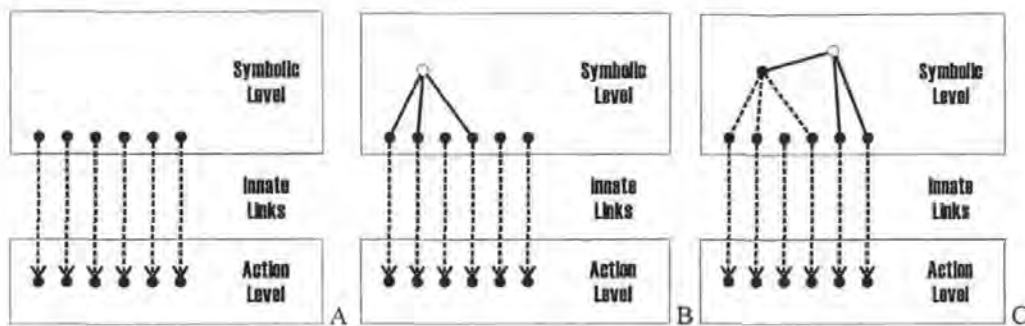


Figure 1. Symbolic learning. (A) is a schematic representation of the initial system, comprising symbols associated with pre-programmed (innate) primitive action procedures. In (B) the user has defined a new procedure (open circle) as a combination of symbols. The new symbol is grounded because it is a construct of grounded symbols. In (C), the user has defined a new procedure that combines a procedure previously defined by himself with primitive action procedures.

symbols that the robot can understand. By understanding we mean here that there is a correspondence between symbols and actions or real-world objects. In this way, the appropriate procedure can be called to act on the sensors/motors accordingly to the user intentions.

This multi-step approach has system-wide repercussions on the design of a robot control system. For example, the robot must be able to distinguish a command to be executed immediately from an instruction to be memorized. This requires context resolution at the natural language processing level. Moreover, the robot must be able to verify that the instruction can be converted into an executable procedure. It requires an internal representation of consequences of actions and the ability to verify the correct action sequencing. The robot must also be able to execute a command while listening to the user, and must cope with interruptions and inappropriate answers to its requests. This requires carefully designed system architecture. Some of the aspects discussed here are presented in more detail in the next sections. In particular, section 3 clarifies how symbol-level description and low-level sensory motor action procedures are integrated. The proposed representation of procedural knowledge is also described. In section 4 the system architecture is described.

The problems of considering the appropriate selection of action primitives is described in section 5 by analyzing recorded route instructions, and establishing a list of actions that are natural to users. The results of this investigation are also discussed. One of them is that the list of primitives may not be a closed one. The implications of that and other findings is discussed in section 6, along with the question of how the proposed system compares to other approaches. The conclusion follows in section 7.

3. IBL model

3.1 Symbolic learning

The learning process is based on predefined initial knowledge. This "innate" knowledge consists of primitive sensori-motor procedures with names, such as "turn left", "follow the road" (the choice of these primitives is explained in sections 3.3 and 5). The name is what we call here a "symbol", and the piece of computer program that controls the execution of the corresponding procedure is called the "action" (Figure 1A). As each symbol is associated with an action, it is said to be "grounded".

When a user explains a new procedure to the robot, say a route from A to B that involves a number of primitive actions, the IBL system, on the one hand, creates a new name for the procedure, and, on the other hand, writes a new piece of program code that executes that procedure and links the code with the name (see section 3.2 for details). The code refers to primitive actions by name. It does not duplicate the low-level code defining these primitives. For that reason, the new program can be seen as a combination of symbols rather than a combination of actions (figure 1B). As all new procedures are constructed from grounded primitives, they become also grounded by inheritance and are "understandable" by the system when referred to in natural language.

When explaining a new procedure, the user can also refer to old procedures previously defined by himself. In that way the complexity of the robot's symbolic knowledge increases (fig. 1C).

3.2 Knowledge representation

The internal representation needs to support three functions: (i) formal modeling of NL route descriptions; (ii) internal route planning for determining whether a given route description is sufficiently specified; and (iii) the generation of

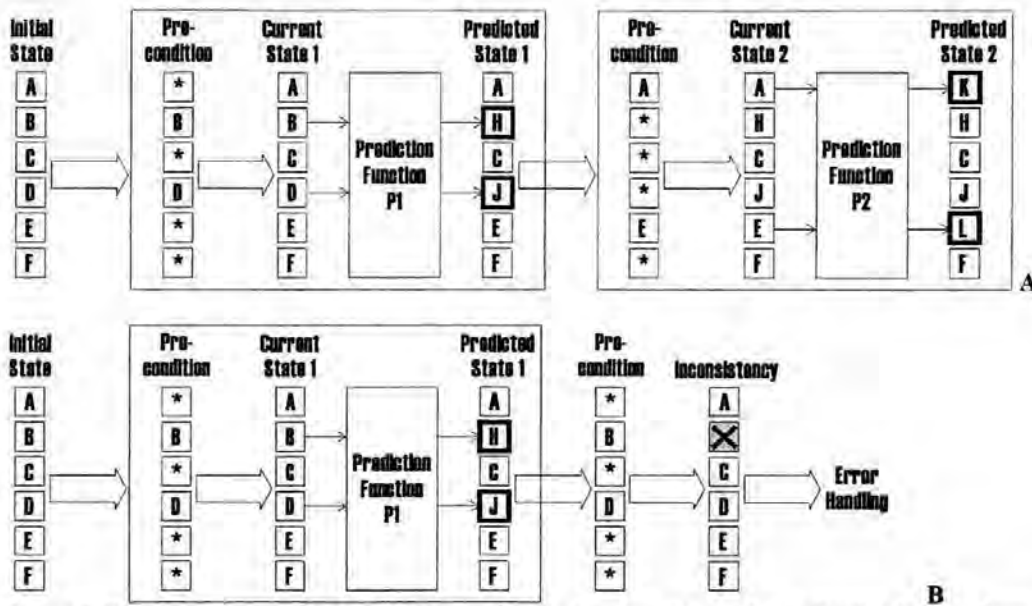


Figure 2. Route instruction verification. (A) For each procedure there is a prediction function that transforms a state vector into its future value. The function first determines if the input state satisfied the minimal criteria ("pre-condition") to enable the procedure to be executed. An action is executable only if selected elements of the state vector have required values. If this is the case, the next state is predicted and processed by the prediction function associated with the next procedure in the instruction. Each action modifies certain components of the state vector, and leaves the other unchanged. (B) If the predicted state produced by one procedure does not allow the next procedure to be executed, an error handling process is initiated. (Note: the "initial state" in the text corresponds to the "current state" in the figure).

procedures for navigation at execution time. These three functions require different representations that will be described in turn.

(i) The utterances of the user are represented using the discourse representation structure (DRS) (Bugmann2001). This is then translated into symbols representing procedures or is used to initiate internal functions such as execution of a command or learning of a series of commands (section 4).

(ii) When the user describes a route as a sequence of actions, it is important for the robot to verify if this sequence is executable. The approach proposed here associate each procedure with a triplet $S_i A_{ij} S_j$ with properties similar to productions in SOAR (Laird et al, 1987). The state S_i is the initial state in which the action A_{ij} can take place. It is the pre-condition for action A_{ij} . The state S_j is the final state, resulting of the action of A_{ij} applied to the initial state (figure 2 clarifies the difference between "initial state" and "pre-condition"). For a sequence of actions to be realisable, the final state of one action must be compatible with the pre-condition of the next one. To enable this verification, the robot must be able to "imagine" the consequence of an action. For that purpose, a PREDICTION function is associated with each primitive action, and with each newly created procedure. Figure 2 illustrates the use of the prediction function during verification of the

consistency of the sequence of instructions from the user. It should be noted that this process also helps detecting some of the errors in natural language processing.

(iii) When a robot executes a command, it executes a piece of program code that contains the sequence of primitive procedures to be executed. Thus, a key part of IBL is the generation of a program code. This is enabled by the use of a scripting language (section 4). This program is called the ACTION function. Both ACTION and PREDICTION functions are physically located in the same file that contains all information specific to a procedure. This is schematised in figure 3.

3.3 Sensory-Motor primitives

Sensory-motor primitives are defined as actions that users usually refer to in unconstrained speech (chunking). These are not low-level robot control actions, and often involve complex processing and planning. A task such as "approach that building at the end of the street" is a typical action that users ask the robot to do at the end of a route instruction, when the goal is in sight (section 5). It is a complex action involving visual detection of a building and of its entrance, its localisation in relation to the street, and planning of a route along the street. All this is easy for a human, but in many ways stretches the limits of robot "intelligence".

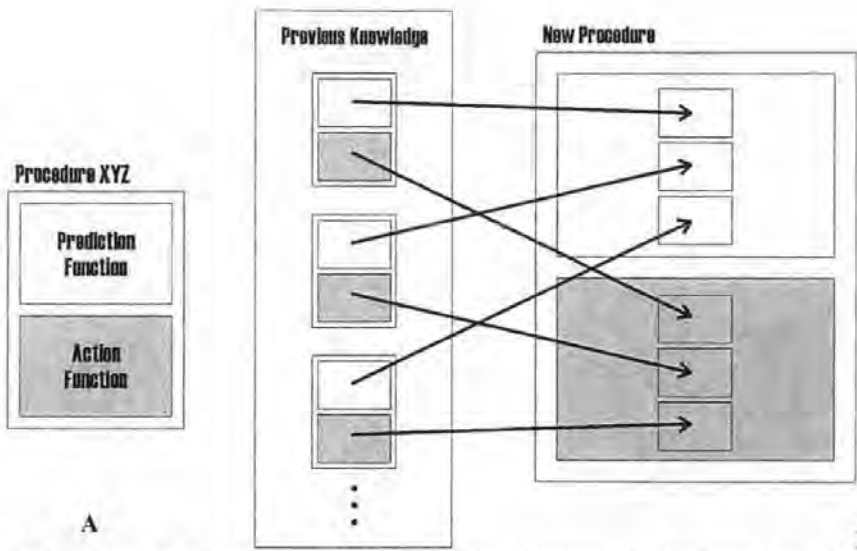


Figure 3. Procedural knowledge representation. (A) A procedure file contains an ACTION function that causes the physical displacement of the robot, and a PREDICTION function that calculates the future state of the robot resulting from the action. The ACTION is used during execution of a command, and the PREDICTION is used for consistency checking during the learning process. (B) An instruction by the user results in a "New Procedure" file being written. In this file, the actions components of the requested primitive procedures are combined (in the form of function calls) to create the new ACTION function, and the prediction components are combined to create the new PREDICTION function. This includes an additional procedure-specific pre-condition.

We see here that, by setting the boundaries between the symbolic level and the action level to be the same as the one found in natural language, the symbolic level processing has been simplified, but at the cost of an increased complexity of "low-level" procedures. These give the robot some autonomy in the execution of commands, as the execution details depend on the local conditions.

4. System Architecture

The architecture is comprised of several functional processing modules (figure 4). These are divided into two major units: the Dialogue Manager (DM) and the Robot Manager (RM).

The DM and the RM are designed as two different processes based on asynchronous communication protocols. These processes run concurrently on different processors. In this way, the system can handle, at the same time, both the dialogue aspects of an incoming request from the user (i.e. speech recognition and semantic analysis, or detection of a "stop" command) and the execution of a previous user request (i.e. check if the request is in the system knowledge domain, and execute vision-based navigation procedures).

Two aspects are essential with this concurrent-processes approach. Firstly, to define an appropriate communication protocol between the two processes. Secondly, to define an appropriate

architecture for the RM and DM allowing the two processes to both communicate with each other while performing other tasks. At present a communication protocol based on sockets and context-tagged messages is evaluated.

Moreover, the system must also dynamically adapt itself to new user requests or to new internal changes, by being able to temporarily suspend or permanently interrupt some previous activity. For example the user may want to prevent the robot crashing against a wall and must therefore be able to stop the robot while the robot is driving towards the wall. Hence, the importance of a concurrent approach where the system constantly listens to the user while performing other tasks and at the same time being able to adjust the task if necessary.

The Dialogue Manager is a bi-directional interface between the Robot Manager and the user, either converting speech input into a semantic representation, or converting requests from the Robot Manager into dialogues with the user. Its components are run as different processes communicating with each other via a blackboard architecture. The RM must concurrently listen/send requests from/to the DM and try to execute them. For this reason a multi-threads approach has been used. Its communication interface is a process that only launches a message evaluation thread "Execution Process" and resumes listening to the DM. The execution process then starts an appropriate thread for executing a command, or

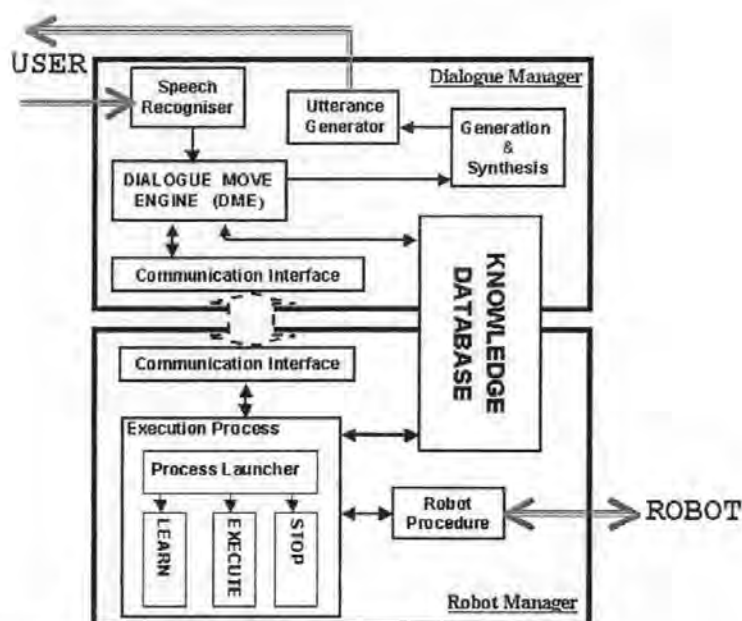


Figure 4. IBL system's architecture (see text for description).

places a tagged message on a message board if it is a part of a dialogue in a specific thread, e.g. learning a route. The characteristic of this approach is that all processes in the RM are sharing a common memory stack so that threads can be started and paused, depending on the user's input. At the moment, the Execution Process component is implemented with the Process Launcher controlling only the Learning and Execution modules since the Stop component is in an early stage of development. The Robot Manager is written using the scripting Python² language and C. An important feature of scripting languages is their ability to write their own code. For instance, a route instruction given by the user will be saved by the Robot Manager as a Python script that then becomes part of the procedure set available to the robot for execution or for future learning.

5. Corpus Collection and Data Analysis

To evaluate the potential and limitations of IBL, a real-world instructions task is used, that is simple enough to be realisable, and generic enough to warrant conclusions that hold also for other task domains. A simple route scenario has been selected, using real speech input and a robot using vision to execute the instructed route (see 5.1 below for more details). The first task in the project is to define the innate actions and symbols in the route instruction domain. For this reason, a corpus

of route descriptions has been collected from students and staff at the University of Plymouth. In section 5.2 and 5.3 corpus collection and data analysis are presented.

5.1 Experimental Environment

The environment is a miniature town covering an area of size 170cm x 120cm (figure 5). The robot is a modified RobotFootball robot³ with an 8cm x 8cm base (figure 6A). The robot carries a CCD colour TV camera⁴ (628 (H) x 582 (V) pixels) and a TV VHF transmitter. Images are processed by a PC that acquires them via with a TV capture card⁵ (an example of such image is shown in figure 6B). The PC then sends motion commands by FM radio to the robot. During corpus collection, the PC is also used to record instructions given by subjects.

The advantage of a miniature environment is the ability to build a complex route structure in the limited space of a laboratory. The design is as realistic as possible, to enable subjects to use expressions natural for the outdoor real-size environment. Buildings have signs taken from real life to indicate given shops or utilities such as the post-office. However, the environment lacks some elements such as traffic lights that may normally be used in route instructions. Hence the collected corpus is likely to be more restricted than for outdoor route instructions. The advantage of using

³ Provided by Merlin Systems
(<http://www.merlinsystems.com/>)

⁴ Provided by Allthings Sales and Services
(<http://www.allthings.com.au/>)

⁵ TV Card: Hauppauge WinTV GO

² <http://www.python.org>

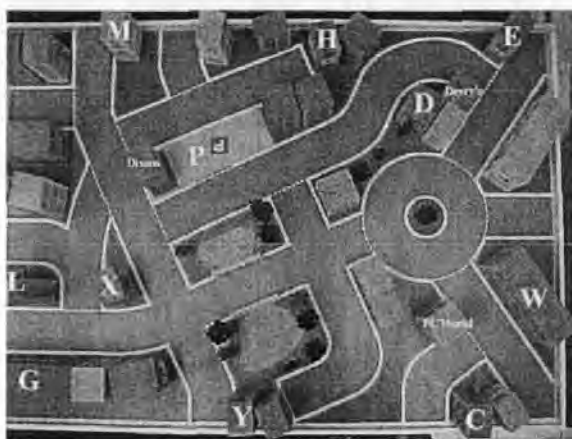


Figure 5. Miniature town in which a robot will navigate according to route instructions given by users. Letters indicate the destinations and origins of various routes used in the experiment.

a robot with a remote-brain architecture (Inaba et al., 2000) is that the robot does not require huge on-board computing and hence can be small, fitting the dimensions of the environment.

5.2 Collection of a corpus of route instructions

To collect linguistic and functional data specific to route learning, 24 subjects were recorded as they gave route instructions to the robot in the environment. Subjects were divided into three groups of 8. The first two groups (A and B) used totally unconstrained speech, to provide a performance baseline. It is assumed that a robot that can understand these instructions as well as a human operator would represent the ideal standard. Subjects from group C were induced in producing shorter utterances by a remote operator "taking notes".

The other two groups (A and B) were told that the robot was remote-controlled and that, at a later date, a human operator would use their instructions

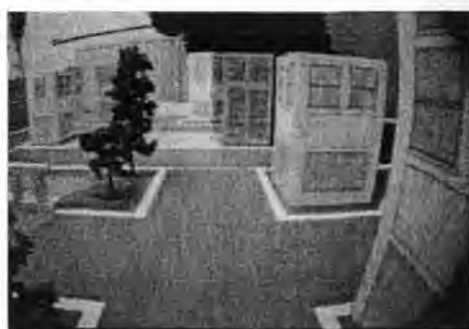
that the human operator would be located in another room, seeing only the image from the wireless on-board video camera. This induced subjects to use a camera-centred point of view relevant for robot procedure primitives. Subjects were also told to reuse previously defined routes whenever possible, instead of re-explaining them in detail. Each subject had 6 routes to describe among which 3 were "short" and 3 were "long". The long routes included a short one, so that users could refer to the short one when describing the long one, instead of re-describing all segments of the short one. This was to reveal the type of expressions used by users to link taught procedures with primitive ones. Each subject described 6 routes having the same starting point and six different destinations. Starting points were changed after every two subjects. A total of 144 route descriptions were collected. For more details about collection and analysis of the corpus see (Bugmann et al. 2001).

5.3 Corpus Analysis: The functional vocabulary

The aim of the corpus analysis is to twofold. First, to define the vocabulary used by the users in this application, in order to tune the speech recognition system for an optimal performance in the task. Secondly, to establish a list of primitive procedures that users refer to in their instructions. The aim is to pre-program these procedures so that a direct translation from the natural language to grounded symbols can take place. In principle, if the robot does not know a procedure, the user could teach it. However, this is a process that we wish to avoid at this stage of the project, as discussed in section 6. Hereafter, we report on the functional analysis of the corpus. The reader interested in the task vocabulary can refer to (Bugmann et al., 2001). The functional vocabulary is a list of primitive navigation procedures found in route



A)



B)

Figure 6 A. Miniature robot (base 8cm x 8cm). B. View from the on-board colour camera.

to drive the robot to its destination. It was specified

descriptions.

	Count	Primitive Procedures
1	308	MOVE FORWARD UNTIL [(past over across) <landmark>] [(half_way_of end_of) street] [(after <number><landmark> [left right]) [road_bend]
2	183	TAKE THE [<number>] turn [(left right)] [(before after at) <landmark>]
3	147	<landmark> IS LOCATED [left right ahead] [(at next_to left_of right_of in_front_of past behind on opposite near) <landmark>] [(half_way_of end_of beginning_of across) street] [between <landmark> and <landmark>] [on <number> turning (left right)]
4	62	GO (before after to) <landmark>
5	49	GO ROUND ROUNDABOUT [left right] [(after before at) <landmark>]
6	42	TAKE THE <number> EXIT [(before after at) <landmark>]
7	12	FOLLOW KNOWN ROUTE TO <landmark> UNTIL (before after at) <landmark>
8	4	TAKE ROADBEND (left right)
9	4	STATIONARY TURN [left right around] [at from <landmark>]
10	2	CROSS ROAD
11	2	TAKE THE ROAD in front
12	2	GO ROUND <landmark> TO [front back left_side right_side]
13	1	PARK AT <location>
14	1	EXIT [car_park park]

Table 2. Primitive navigation procedures found in the route descriptions collected from groups A and C. Procedure 3 is used by most subjects to indicate the last leg of a route, when the goal is in sight.

The initial annotation of instructions in terms of procedures, as reported here, is somehow subjective, and influenced by two considerations. (i) The defined primitives will eventually be produced as C and Python Programs. It was hoped that only a few generic procedures would have to be written. Therefore, the corpus has been transcribed into rather general procedures characterised by several parameters (table 2). (ii) An important issue is knowledge representation. According to the SAS representation discussed in section 3.2, the executability of primitives can only be evaluated if their initial and final states are defined. Subjects however rarely specified explicitly the starting point of an action and sometimes did not define the final state in the same utterance. Nevertheless, it was assumed that the system would be able to infer the missing information from the context. Therefore, procedures without initial or final state were considered to be complete, and were annotated as such. The specifications of primitive procedures are likely to evolve during the project.

This analysis methodology differs slightly from the one in (Denis, 1997). In our analysis, there are no statements describing landmarks, as these are made part of procedures specifications, and consequently there are also no actions without reference to landmarks. Even when a subject specified a non-terminated action, such as "keep going", it was classified as "MOVE FORWARD UNTIL", assuming that a termination point would

be inferred from the next specified action. The list of actions found in the route descriptions of groups A and C is given in table 2. Figure 7 shows that the number of distinct procedures is increasing with the number of sampled instructions, but at a rate much smaller than the number of distinct words reported in (Bugmann et al., 2001). Here we discover on average one new procedure for every 38 route instructions, while with words, we discovered in average one new word for each instruction. New procedures typically are the least frequent in table 2.

6. Discussions

Teaching a route to a robot using natural language is an application of a more general instruction-based learning methodology. The corpus-based approach described here aims at providing users with the possibility of using unconstrained speech, whilst creating an efficient natural language processing system using a restricted lexicon. It is found that the functional vocabulary is small, containing only 12 primitives (although that number may vary with the annotation method). From a roboticist's point of view, route navigation could probably be achieved with a smaller number of primitives. However, when accepting spontaneous speech, a wider variety of functions must be expected.

An important finding is that the functional vocabulary is not closed. Hence, at some point in

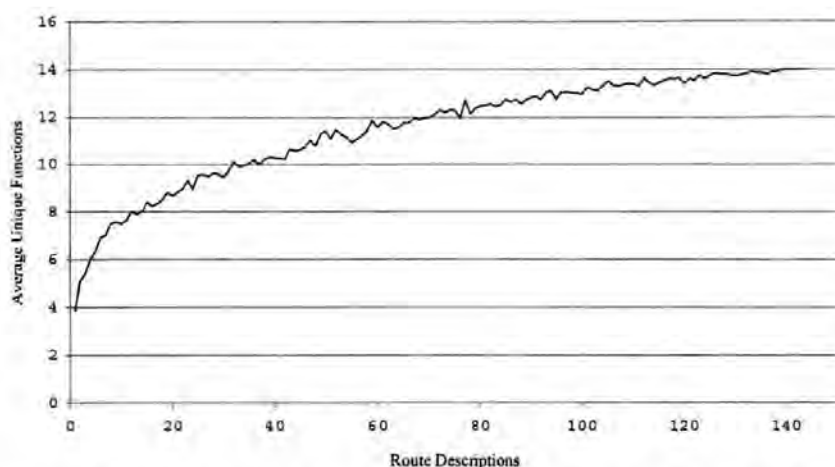


Figure 7. Average number of unique procedures as a function of the number of collected route instructions. The curve is obtained by averaging 50 sets comprising a random selection of n route descriptions. The number n is shown on the x -axis of the graph. The slope of the curve indicates that, on average, one new function will be added to the functional lexicon for every 38 additional route instructions collected.

the robot's life, the user will have to teach it new primitives (e.g. "cross the road") or reformulate its instructions. To enable learning, the robot must possess a larger set of primitives, which correspond to lower level robot actions. For instance, the user may wish to refer to a number of wheel turns in its instruction. Examples of such instructions are found in (FLAKEY) and (Seabra Lopes, 2000). With our approach, this would require the collection of a new corpus to determine the necessary additional primitive procedures. Another solution may lie in an appropriate dialogue management to suggest a reformulation of the instruction. It is expected that with the corpus-based method used here, the frequency of such "repair dialogues" will be minimised. An open question is the detection of new functions in the user's utterance, as the lexicon may not contain the required vocabulary.

The approach to robot control described may be seen as an attempt to integrate the good properties of Behaviour-based control and classical AI. Behaviour-based control is an effective method for designing low-level primitives that can cope with real-world uncertainties, and AI has developed effective tools for symbol manipulation and reasoning (for a more detailed discussion about hybrid systems see for example Malcom (1995)). However, the system differs in several ways from both methods. Here, the corpus defines what symbols and primitives to use. Consequently, some of the primitives are rather complex functions, involving representations of the environment and planning. These are not always compatible with the representation-less philosophy of behaviour-based

systems. On the AI side, the system does not use the full range of reasoning capabilities offered by systems such as SOAR. There are no other aims in symbolic processing than verifying the consistency of instructions, and the construction of new procedure specifications.

Other previous work on verbal communication with robots has mainly focused on issuing commands, i.e. activating pre-programmed procedures using a limited vocabulary (e.g. IJCAI'95 office navigation contest). Only a few research groups have considered learning, i.e. the stable and reusable acquisition of new procedural knowledge. (Huffman & Laird, 1995) used textual input into a simulation of a manipulator with a discrete state and action space. (Crangle and Suppes, 1994) used voice input to teach displacements within a room and mathematical operations, but with no reusability. In (Torrance, 1995), textual input was used to build a graph representation of spatial knowledge. This system was brittle due to place recognition from odometric data and use of IR sensors for reactive motion control. Knowledge acquisition was concurrent with navigation, not prior to it. Whereas in (Matsui et al. 1999), the system could learn new actions through natural language dialogues but only while the robot was performing them (i.e. it could only learn a new route from A to B while it was actually moving from A to B and dialoguing with the user).

In the IBL system described here, learning operates purely at the symbolic level; hence it can be done prior to performance. The ability to predict future states enables to engage in a verification

dialogue before execution errors occur. If environmental conditions change such that an instruction is not valid anymore, this can be detected from the mismatch between the expected result and the actual one. Learning however is not autonomous. The system requires interaction with a human user to learn new symbols and their meaning. This simplifies the design of the robot due to the transfer of part of the cognitive load to the user. Future experiment will reveal if this approach results in effective and socially acceptable helper robots.

7. Conclusions

The project described in this paper is aimed at exploring IBL for route descriptions. It has been discussed how the design of the IBL system is adapted to natural human behaviour. Indeed, both the vocabulary matches to the unconstrained user language and the functional primitives built into the robot are determined from actions natural to the users. This defines an architecture open to spontaneous user interventions, unexpected replies and errors. Nevertheless, user-friendliness is not a prior specification, but a consequence of practical constraints. Indeed, robots without learning will not achieve specific tasks (such as finding HAL) and a system without adapted vocabulary causes too many errors. Similarly, explaining tasks is beyond the cognitive capabilities of users without high level primitives and, like with HAL, a robot that listens only when it decide to do so would be out of control. So far, the speech recognition part is in an early stage of development while the DRS part is operational for a limited number of examples, and that work is in progress to improve the coverage of corpus. However, it is found that the functional vocabulary is small, containing only 12 primitives (although that number may vary with the annotation method). The full transformation from NL utterances into procedures has been tested with dummy primitives (i.e. preprogrammed robot displacements). Programs for the proper sensory-motor primitives are currently under development. This will then allow further testing of the IBL concept.

However the initial results presented here show that neither the lexicon nor primitive procedures are likely to form closed sets. Ideally, IBL system should therefore also be capable of acquiring new words, and users should be given the possibility to teach new primitive 'innate' procedures. Unfortunately, the former is beyond the capabilities of current speech recognition systems. As for learning new primitives procedures, this would require a new set of more primitive procedures to

be combined via user instructions. Whether it will be possible to explore this during the project is unclear. To allow IBL to operate despite these limitations, it is likely that dialogue management will play a crucial role.

Acknowledgement: This work is supported by EPSRC grants GR/M90023 and GR/M90160. The authors are grateful to A. Cangelosi and K. Coventry for enlightening discussions.

References:

- Bugmann G., Lauria S., Kyriacou T., Klein E., Bos J. and Coventry K. (2001) "Using Verbal Instruction for Route Learning", Proc. of 3rd British Conf. on Auton. Mobile Robots and Autonom. Systems: Towards Intelligent Mobile Robots (TIMR'2001), Manchester, 5 April.
- Cangelosi A., Harnad S. (2001) The adaptive advantage of symbolic theft over sensorimotor toil: Grounding language in perceptual categories. *Evolution Communication*. (in press)
- Crangle C. and Suppes P. (1994) Language and Learning for Robots, CSLI Lecture notes No. 41, Centre for the Study of Language and Communication, Stanford, CA.
- Denis M. (1997) "The description of routes: A cognitive approach to the production of spatial discourse", *CPC*, 16:4, pp.409-458.
- FLAKEY:
www.ai.sri.com/people/flakey/integration.html
- Huffman S.B. and Laird J.E. (1995) "Flexibly Instructable Agents", *Journal of Artificial Intelligence Research*, 3, pp. 271-324.
- Inaba M., Kagami S., Kanehiro F., Hoshino Y., Inoue H. (2000) "A platform for robotics research based on the remote-brained robot approach", *International Journal of Robotics Research*, 19:10, pp. 933-954.
- Laird J.E., Newell A. and Rosenbloom P.S. (1987) "Soar: An architecture for general Intelligence" *Artificial Intelligence*, 33:1, pp.1-64.
- Malcom C. M. (1995), The SOMASS system: a hybrid symbolic and behaviour-based system to plan and execute assemblies by robot. In J. Hallam, et al. (Eds), *Hybrid problems and Hybrid solutions* pp 157-168. Oxford: ISO-press.
- Matsui T., Asoh H., Fry J., et al. (1999) Integrated Natural Spoken Dialogue System of Jijo-2 Mobile Robot for Office Services, <http://citeseer.nj.nec.com/matsui99integrated.html>
- Miller G. (1956) 'The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity Processing Information'. *The Psychol. Review*, v. 63, p. 81-97
- Seabra Lopes, L. and A.J.S. Teixeira (2000) Human-Robot Interaction through Spoken Language Dialogue, *Proceedings IEEE/RSJ International Conf. on Intelligent Robots and Systems, Japan*.
- Torrance M.C. (1994) Natural Communication with Robots, MSc Thesis submitted to MIT Dept of Electrical Engin. and Comp. Science

Converting Natural Language Route Instructions into Robot Executable Procedures

S. Lauria T. Kyriacou G. Bugmann
Robotic Intelligence Laboratory
School of Computing
University of Plymouth
Drake Circus, Plymouth PL4 8AA
United Kingdom

J. Bos E. Klein
ICCS
Division of Informatics
University of Edinburgh
Buccleuch Place, Edinburgh EH8 9LW
Scotland, United Kingdom.

Abstract

Humans explaining a task to a robot use chunks of actions that are often complex procedures for robots. An instructable robot needs to be able to map such chunks to existing pre-programmed primitives. We investigate an architecture for spoken dialogue systems able to extract executable robot procedures from user instructions. A suitable representation of the dialogue is introduced, then a Procedure Specification Language (PSL) is described that allows to extract from the semantic representation of the dialogue the robot executable procedures and their parameters

1 Introduction

This paper presents an semantically based approach for human-robot dialogue understanding, as part of a project than envisages "Instruction-Based Learning" (IBL) [5], where robots acquire user-specific skills based on verbal instructions given by the user. In particular, we will focus on mapping the human language commands to for the robot executable instructions, using an intermediate semantic representation.

Our IBL system operates according to the following scenario. A user engages in a dialogue with the robot, where spoken instructions are mapped to semantic representations, natural language ambiguities are resolved, and the functional parameters are extracted from that representation [9]. The robot, having a database with previously learned tasks at its disposal, will now either perform the given instruction (if it knows how to do it), or if the task is unknown, ask the user to explain how to perform the task. The user then explains the task step by step. At the end of this learning process, the robot will have built a new procedure that becomes part of its knowledge base.

The requirements of natural language understanding induce the internal model of a route as a sequence

of high-level task specifications (primitives). Hence it is necessary to provide the robot with a set of pre-programmed primitives corresponding to action chunks referred to by users. For more details about these aspects see [7].

A typical example in our scenario is the following (example u8_GC_HD extracted from the IBL corpus):

Instructor: Go to the post office!

Robot: How do I get to the post office?

Instructor: Er head to the end of the street.
Turn left. Take the first left. Er go right down
the road past the first right and it's the next
building on your right.

One of the issues in the project is the mapping from action chunks used in natural language to actions executable by the robot for both of the possible situations: either the system already knows how to perform a request, or it has to learn how to perform it. The first case corresponds to a successful mapping from the semantic analysis of the request to a sequence of executable robot actions. The second case corresponds to the creation of a sequence of executable robot actions for the unknown request through a user-robot dialogue.

Previous approaches to interpreting natural language instructions for mobile robots assume a application specific semantic representation [4]. However, we argue that there is a need for a domain-independent intermediate representation. This representation captures the meaning of the dialogue between user and robot and is used to resolve ambiguities inherent in natural language (for instance the reference of the pronoun it the example above). In addition, we use an application specific mapping from the intermediate representation to obtain robot executable scripts. Using this extra layer results in an overall system that

is much easier to adapt the robot to new scenarios or tasks.

The paper is structured as follows. First we introduce the intermediate semantic representations known as Discourse Representation Structures (Section 2). In Section 3 we present the Procedure Specification Language (PSL) used for the interpretation of the DRS. Section 4 illustrates the conversion of basic program components found in verbal instructions into robot-executable procedures. In Section 5 ongoing work covering the reuse of previously explained routes is discussed.

2 Understanding Natural Language Instructions

We will use Discourse Representation Structures (DRSs) to represent the meaning of the dialogue between user and system. There are three reasons that motivate this choice of formalism. First and foremost, DRT is a well understood framework and covers a wide variety of linguistic phenomena [6, 11]. These phenomena include context-sensitive expressions such as pronouns and presuppositions. To our knowledge, there is no other semantic formalism that comes close to the empirical coverage of DRT. Second, there now exist computational implementations that provide means to extend existing linguistic grammars with DRS-construction tools, and there are efficient algorithms available that implement Van der Sandt's presupposition projection algorithm for DRT [2]. Third, there is a direct link between DRT and first-order logic—there is a translation from DRSs to formulas of first-order logic that behaves linear on the size of the input [1].

2.1 Representing Instructions

DRT was initially designed to deal with texts, so we will use an extension of standard DRT that enables us to cope with instructions such as given in the example above. This extension introduces *actions* and modal operators into the DRS-language.

Let us first define the syntax of the DRS language. Basic DRSs have two components: a set of *discourse referents*, and a set of *conditions* upon those referents. Discourse referents stand for objects mentioned in the course of the dialogue. Conditions constrain the interpretation of these discourse referents. More formally, DRSs and merge of DRSs are defined in the usual way:

Syntax of DRSs:

1. If $\{x_1, \dots, x_n\}$ is a set of discourse referents, and $\{\gamma_1, \dots, \gamma_m\}$ is a set of DRS-conditions, then the ordered pair $\langle \{x_1, \dots, x_n\}, \{\gamma_1, \dots, \gamma_m\} \rangle$ is a DRS;

2. If B_1 and B_2 are DRSs, then so is $(B_1 \oplus B_2)$.

Following Lascarides [8], we extend the DRS language with action terms. Atomic action terms are identified by the δ -operator. Complex action-terms are composed out of other action terms by either ; (sequence) or | (free choice).

Syntax of DRS-action-terms:

1. If B is a DRS, then δB is a DRS-action-term;
2. If A_1 and A_2 are DRS-action-terms, then so are $(A_1; A_2)$ and $(A_1 | A_2)$.

The DRS-condition subsume those of standard DRT. Further we have the modal operators $\Box$ and $\Diamond$ (clauses 3 and 6), hybrid DRS-conditions formed by discourse referents and DRSs (clause 5), and the command operator (clause 7).

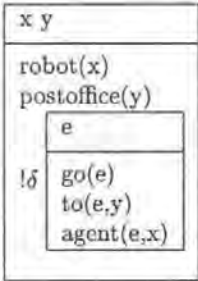
Syntax of DRS-conditions:

1. If R is a relation symbol for an n -place predicate and $x_1, \dots, x_n$ are discourse referents then $R(x_1, \dots, x_n)$ is a DRS-condition;
2. If x_1 and x_2 are discourse referents, then $x_1 = x_2$ is a DRS-condition;
3. If B is a DRS, then $\neg B$, $\Box B$, $\Diamond B$ are DRS-conditions;
4. If B_1 and B_2 are DRSs, then $B_1 \vee B_2$, $B_1 \Rightarrow B_2$ are DRS-conditions;
5. If x is a discourse referent and B a DRS, then $x:B$ is a DRS-condition;
6. If A is a DRS-action-term, and B a DRS, then $[A]B$ and $\langle A \rangle B$ are DRS-conditions;
7. If A is a DRS-action-term then $!A$ is a DRS-condition.

One of the theoretical motivations behind the internal structure of DRSs is the analysis of pronouns and other anaphoric expressions. Pronouns are interpreted in DRT by binding a previously introduced *accessible* discourse referent. Accessibility is governed by the way DRSs are nested into each other, and hence narrows down the choice of an antecedent in the process of pronoun resolution.

2.2 Example Representations

We will now illustrate the formal syntax definition by giving some examples that show how instructions can be modelled. We will use the more convenient box notation for DRSs in the examples that follow. Recall that we use the δ -operator to form action-DRSs from DRSs and the $!$ operator to express that an action is commanded. So, the directive Go to the post office! translates to the following DRS:



This DRS states that the plan in the actual world contains the action for the robot to go to the post office. Semantically, actions relate two possible worlds: the world (or state) in which the action is issued, and the world in which the effects of the action hold. Because actions themselves can be of complex nature, we will use an additional world that describes what constitutes the action. This enables us to reason about possible outcomes of actions (not only for the purpose of planning, but also to verify that the resulting states are desired) and to check the preconditions of actions.

2.3 Interpreting Instructions

One way to interpret DRSs is to translate them to ordinary first-order logic. This is the approach that Bos & Oka follow [3], and they use classical first-order theorem provers and model builders to automate inference. The translation they use is based on the relational translation for modal logic to first-order formulas and essentially similar to the standard translation from DRT to first-order logic [6], extended with rules to deal with the modal operators and DRS-action-terms. The example DRS above would get the following translation in first-order logic:

$$\exists w \exists x \exists y (\text{possible_world}(w) \wedge \text{robot}(w, x) \wedge \text{postoffice}(w, y) \wedge \exists v \exists a (\text{action}(w, a, v) \wedge \exists e (\text{go}(a, e) \wedge \text{to}(a, e, x) \wedge \text{agent}(a, e, y))))$$

Note that the translation increases the arity of all predicates symbols with one, where the additional argument position denotes a possible world. A minimal first-order model satisfying this formula (and further

background knowledge in the form of meaning postulates describing the pre-conditions and effects of actions) could contain the following information:

```
D={d1,d2,d3,d4,d5,d6,d7,d8}
F(possible_world)={d1,d2,d3}
F(robot)={{(d1,d4),(d2,d4),(d3,d4)}}
F(postoffice)={{(d1,d5),(d2,d5),(d3,d5)}}
F(action)={{(d1,d2,d3)}}
F(go_from_to)={{(d2,d4,d6,d5)}}
F(at_loc)={{(d1,d4,d6),(d3,d4,d5)}}
```

Bos & Oka [3] actually employ automated model builders to generate models of these kind, and use these to extract actions for the dialogue manager. Since models are essentially flat structures without recursion, they are easy to process. For instance, all quantification and boolean structures are explicit in models. This makes models ideal to function as a database-lookup table to find out whether there are actions to be performed by the system.

However, the state-of-the-art in automated model building is not in a stage yet where it leads itself easily to integration in efficient implementations. Although the model building methods performs well for examples up to a few utterances, in general the instructions in the IBL corpus are much larger than that and sometimes reach ten to twenty utterances in a learning dialogue. Therefore we use an alternative rule-based method to extract executable primitives from DRSs. This technique is much more efficient and will be presented in the next section.

3 Procedure Specification Language

The internal representation of a route is a sequence of high-level task specifications (primitives). For this reason a production-rule based approach has been used to interpret the DRS as a sequence of procedure names.

The Procedure Specification Language (PSL) provides a common interchange language to describe resources. A list of robot executable procedures are extracted from the DRS and saved as a new procedure -the result of Instruction Based Learning-. The PSL provides the skeletal syntax used to compose the procedure names and the required parameters.

The PSL terms are either *special characters* or regular *string literals*, where string literals are made of sequences of characters excluding the special characters. The complete set of special characters that cannot appear as part of a string literal is:

& | # % \$ ->

These characters can only be used for the special syntactic forms described in the above RSL syntax overview

The core syntax of the PSL syntax is the rule $a \rightarrow b$. Rules associate the *condition* a on the left of the special syntax $\rightarrow$ with the string on the right of $\rightarrow$ corresponding to the robot procedure. For example, the rule

```
event(X)&go(X)&to(X,Z)&$landmark(Z)->
go(prepare='to',landmark=$landmark(Z)) (1)
```

will generate the procedure

```
go(prepare='to',landmark='postoffice')
```

from the DRS in Section 2.2.

In each PSL rule, the condition is a conjunction of terms separated using the special syntax $\&$, where each term can be a one or two place predicate symbol, a variable predicate, a variable action. In (1) $event(X)$ is an example of a one place predicate while $to(X,Z)$ is a two place predicate. The upper case symbol in parenthesis (i.e. X for $event(X)$) is the variable associated with the predicate. In the example in Fig 1, only the $event, go, to$ predicates with the same value for X can be considered to satisfy the condition for rule (1).

A variable predicate is indicated as the special symbol $\$$ followed by a string literal. A variable predicate indicates a class of possible predicates. In the rule example (1), $\$landmark(Z)$ specifies that the predicate must be of landmark type.

A list containing all the predicate belonging to each class defined must be included with the PSL. The syntax to specify a class and all the members belonging to it is:

```
class_name: predicate_1|predicate_2|...|...
```

where *class_name* is the string literal indicating the class and $predicate_1|predicate_2$ is the list of terms $predicate_1, predicate_2$ belonging to the class separated by the special symbol $|$.

An action predicate is indicated as the special symbol $\#$ followed by a predicate pointing to an action. For example, an action predicate can indicate an action to be executed while executing another action (for example *sure from the hospital er go forwards until you come to dioxons* extracted from *u9_GC_HW* in the IBL corpus)

The end of both a rule and a class list is indicated by the special syntax $\%$. The list of the defined class is preceded by the string $\#parameters\%$, while the list of the rules is preceded by the string $\#rules\%$.

The PSL rule based approach facilitates the interpretation of a user command into a call to a procedure with the correct parameter associated to it. The

introduction of parametrised primitives allows it to generalise the use of the procedure. For instance, the procedure designed for *turn left after the tree* should also work if the value *tree* for the parameter *landmark* is replaced by the value *church*. It is also possible to pass different combinations of parameters to the primitive procedure.

While, as explained in more detail in [9], the choice of the initial set of primitives is corpus based (that is it has been driven by the way users express themselves), both the parameter combinations and the interpretation of predicates into a parameter value is mainly robot driven as explained in more detail in [7].

The PSL rule syntax allows to establish the desired mapping between the predicates from the DRS representation and their interpretation into the correct value for the correct parameter. For example the user utterances: *turn right* and *take a right* should produce the same procedure call despite being represented as two different types of events in the DRS (i.e. as a *turn* and a *take* action respectively). Table 1 shows two possible rules allowing to obtain the same procedure call for both utterances.

```
event(X)&turn(X)&in(X,Z)&$direction(Z)->
turn($direction(Z))
```

```
event(X)&take(X)&$direction(Z)->
turn($direction(Z))
```

Table 1: PSL rules. Example of two rules mapping different symbolic representation of an action into the same procedure.

Not all the information present in the DRS is used in detecting the condition components of the rule. One aspects still not yet fully implemented is the use of negation in the condition part of the rule. This would allow the designer to exclude undesired combination of predicates to be mapped into a rule.

4 Basic Program Components

A requirement in Instruction Based Learning is that components such as conditionals, loops, sequences found in instructions are correctly converted into robot executable procedures. Utterances containing conditional expressions have not been found in this corpus. Here, instructions consist mainly of sequences and loops.

4.1 Sequences

Since the order of the actions in the utterance is preserved by the DRS, extracting a properly ordered sequence of primitives and building the corresponding

procedure code is straight forward. For instance, the pseudocode for the user explanation (example extract from u22_GB_CD in the IBL corpus):

Instructor: er you have to take right and then again the first right

is shown in Table 2.

```
def action():
    ....
    take(direction='right')
    take(direction='right',ordinal='first')
    ....
```

Table 2: Sequential Instructions. Pseudocode for the sequence of procedures obtained from a sequence of user actions

4.2 Loops

References to loops where an action has to be executed a fixed number of times are not found in the corpus. However, while-loops and do-until-loops are frequently found. These can either be explicit or implicit.

Implicit while-loops are found in actions such as in the example extracted from u22_GB_CD in the IBL corpus:

Instructor: er you have to take right

This action implicitly requires from the robot to search for the landmark *right_turning* while following the road. Such implicit loops are handled inside pre-programmed procedures (e.g. `take(direction='right',ordinal='first')`. See [7] for more details.

However, an action can be explicitly described as loop by the user in utterances such the one from u20_GB_EP extracted from the IBL corpus:

Instructor: ..keep turning right until you ve got the grand hotel on your left..

With the PSL it is possible to define a suitable rule which allows to introduce the loop explicitly on the right hand side of it. For example, the rule in Table 3 will produce the pseudocode in table 4 for the utterance from u20_GB_EP. As a result, everytime the program in Table 4 is called the loop will be executed.

```
event(X)&turn(X)&in(X,Z)&$direction(Z)
&until(X,C)&#proposition(C) ->
while !(#proposition(C)):
    turn(direction=$direction(Z))
```

Table 3: Loop.Example of a rule extracting an explicit while-loop.

```
def action():
    ....
    while !(near(landmark='grand_hotel')):
        turn(direction='right')
    ....
```

Table 4: Loop. Pseudocode for the utterance u20_GB_EP.

5 Reusing Previously Learnt Procedures

One of the key features of IBL is the reuse of previously explained procedure as part of explanations of new more complex procedure. In the corpus of route instructions, this takes the form of previously explained routes being reused in later route explanations. Three possible ways of reusing previous routes can be found in the corpus. In the first case, the user explicitly refers to the use of the whole of a previously explained route followed by a series of actions. The following example is extracted from u12_GA_EG in the IBL corpus:

Instructor: go to the post office at the post office turn left take a right at the crossroads tesco's is on the left hand side of the street

In the second case, the user still explicitly refers to a previously explained route. However, this time the route has to be used only partially since at a given point (e.g. a landmark) a diversion is introduced by the user. The following example u6_GC_CM is extracted from the IBL corpus:

Instructor: right erm head as though you re going towards the post office so you go over the bridge but instead of carrying straight on take a right carry on down that road until it bears round to the right slightly and at the end of the road the museum is there

In the third case, the user does not explicitly refer to a previously explained route, but only refers to a landmark used in it. Thus that route has to be inferred. The following u13_GA_CL example is extracted from the IBL corpus:

Instructor: go to the bridge mentioned previously continue over the crossroads immediately after the bridge and follow the road to its end on your right you ll find the queens pub

In all these cases the system must be able to correctly link previously learnt sequences of actions with new instructions. As explained in more detail in [10], during the execution of a sequence of actions, the final state of the robot after an action must be compatible with the initial state of the next action. As a consequence, the recalled procedure has to be *tailored* so that the next procedure can be successfully executed.

For example in the utterance u12.GA_EG, the first action recalls the procedure to go to the post office, which ends with the robot facing the entrance of the post office, but this is not a suitable initial state for the next action (i.e turn left). So for the robot to succeed, the procedure `go_postoffice()` should not be executed entirely. However, to determine which elements of a previously learnt sequence must be kept is not an easy problem.

A solution to this problem could be the rule in Table 5. In this case the procedure `go_postoffice()` is executed until the condition `near(landmark='postoffice')` is verified, where `near(landmark='postoffice')` is a vision based procedure that check whether the robot is near the post office.

```
event(X)&go(X)&to(X,Y)&postoffice(Y)->
while !(near(landmark='postoffice')):
    go_postoffice()
```

Table 5: Linking Sequences. The procedure `go_postoffice()` is executed until the robot is near the post office.

Then, the part of the procedure `go_postoffice()` which drives the robot into a position facing the post office, will not be executed. Note that this implies a concurrent execution of the two procedures.

Future work will cover the resolution of the problem above also for more complex cases such as the u13_GA_CL example and will evaluate the efficacy of the various components presented in this paper in converting Natural Language instructions into robot procedures.

Acknowledgments

This work is supported by EPSRC grants GR/M90023 and and GR/M90160.

References

- [1] Patrick Blackburn, Johan Bos, Michael Kohlhase, and Hans de Nivelle. Inference and Computational Semantics. In Harry Bunt, Reinhard Muskens, and Elias Thijssen, editors, *Computing Meaning*, volume 2, pages 11–28. Kluwer, 2001.
- [2] Johan Bos. Implementing the binding and accommodation theory for anaphora resolution and presupposition projection. *Computational Linguistics*, to appear.
- [3] Johan Bos and Tetsushi Oka. An Inference-based Approach to Dialogue System Design. In *Proceedings of Coling 2002*, 2002.
- [4] C. Crangle and P. Suppes. *Language and Learning for Robots*. CSLI Lecture Notes 41. Chicago University Press, Stanford, 1994.
- [5] <http://www.tech.plym.ac.uk/soc/staff/guidbugm/ibl/index.html>.
- [6] Hans Kamp and Uwe Reyle. *From Discourse to Logic; An Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and DRT*. Kluwer, Dordrecht, 1993.
- [7] Theodoris Kyriacou, Guido Bugmann, and Stanislaw Lauria. Personal Robot Training via Natural-Language Instructions. In *Proceedings of IROS 2002*, 2002. to appear.
- [8] Alex Lascarides. Imperatives in Dialogue. In *Proceedings of the 5th International Workshop on Formal Semantics and Pragmatics of Dialogue (BI-DIALOG)*, pages 1–16, Bielefeld, Germany, 2001.
- [9] Stanislaw Lauria, Guido Bugmann, Theodoris Kyriacou, Johan Bos, and Ewan Klein. Training Personal Robots Using Natural Language Instruction. *IEEE Intelligent Systems*, pages 38–45, September/October 2001.
- [10] Stanislaw Lauria, Guido Bugmann, Theodoris Kyriacou, and Ewan Klein. Mobile Robot Programming Using Natural Language. *Robotics and Autonomous Systems*, pages 171–181, 2002.
- [11] Rob A. Van der Sandt. Presupposition Projection as Anaphora Resolution. *Journal of Semantics*, 9:333–377, 1992.

Mobile Robot Programming Using Natural Language.

Stanislao Lauria, Guido Bugmann¹, Theodoris Kyriacou, Ewan Klein*

Centre for Neural and Adaptive Systems, School of Computing, University of Plymouth
Drake Circus, Plymouth PL4 8AA, United Kingdom.

*Institute for Communicating and Collaborative Systems, Division of Informatics, University of Edinburgh, 2
Buccleuch Place, Edinburgh EH8 9LW, Scotland, United Kingdom.

<http://www.tech.plym.ac.uk/soc/staff/guidbugm/ibl/index.html>

KEYWORDS: Natural Language, Human-robot dialogue, mobile robots learning, corpus collection, route-description.

Abstract

How will naive users program domestic robots? This paper describes the design of a practical system that uses natural language to teach a vision-based robot how to navigate in a miniature town. To enable unconstrained speech the robot is provided with a set of primitive procedures derived from a corpus of route instructions. When the user refers to a route that is not known to the robot, the system will learn it by combining primitives as instructed by the user. This paper describes the components of the Instruction Based Learning architecture and discusses issues of knowledge representation, the selection of primitives and the conversion of natural language into robot-understandable procedures.

1 Introduction

Intelligent robots must be capable of action in reasonably complicated domains with some degree of autonomy. This requires adaptivity to a dynamic environment, ability to plan and also speed of execution. In the case of helper robots, or domestic robots, the ability to adapt to the special needs of their users is crucial. The problem addressed here is one of how a user could instruct the robot to perform tasks which manufacturers cannot completely program in advance. In such case the system would not work at all if it cannot learn.

Such learning requires interaction and collaboration between the user and the robot. But, as most users are computer-language-naïve, they cannot personalise their robot using standard programming methods. Indirect methods, such as learning by reinforcement or learning by imitation, are also not appropriate for acquiring user-specific knowledge. For instance, learning by reinforcement is a lengthy process that is best used for refining low-level motor controls, but becomes impractical for complex tasks. Further, both methods do not readily generate knowledge representations that the user can interrogate.

Instruction-Based Learning (IBL), which uses unconstrained speech, has several potential advantages. Natural language can express rules and sequences of commands in a very concise way. Natural language uses symbols and syntactic rules and is well suited to interact with robot knowledge represented at the symbolic level. It has been shown that learning in robots is much more effective if it operates at the symbolic level [2]. This is to be contrasted with the much slower learning at the level of direct sensory-motor associations.

Chunking, sequencing and repair are the aspects, related to natural language interactions, shaping the design of IBL systems discussed here. Chunking is a principle that applies to the communication of information.

Chunking is meant here as the human characteristic to divide, during explanations, tasks into sub-tasks so that all information should be presented in small 'basic' units of actions. As shown in [12], chunking is done spontaneously by humans and we expect that conversions from natural language instruction to robot program will be facilitated if the robot knows a set of primitive procedures corresponding to the action-chunks natural to the user.

Regarding repair, natural language explanations are notoriously underspecified, and the robot must be able to verify the consistency of the acquired program. For example, in a sequence of instructions given by the user, the final state of an action may not correspond to the expected state for the next action. In this case, the system

¹ To whom correspondence should be addressed.

would not be able to perform its task due to a missing chunk. For this reason, it is necessary to define a proper internal knowledge representation allowing the system to detect the missing information. In this way, the system would be able to make predictions about future events so that the problem can be solved while the system is still interacting with the user.

The system not only has to pay attention to user knowledge and dialogue goals, but it also has to adapt its dialogue behaviour to current limitations of the user's cognitive processing capabilities. Assistance is then expected from the system, so that the interaction may naturally flow over the course of several dialogue turns. Finally, a dialogue manager should take care of identifying, and recovering from, speech recognition and understanding errors.

This paper describes initial steps and considerations towards a practical realisation of an IBL system. The experimental environment is that of a miniature town in which a robot provided with video camera executes route instructions. The robot has a set of pre-programmed sensory-motor action primitives, such as "turn left" or "follow the road". The task of the user is to teach the robot new routes by combining action primitives. That task should reveal all the constraints described above, and enable testing of the developed methodology.

The closer the correspondence between primitives and chunks expressing the very basic actions (such as "turn left") is, the less difficult the learning is, since, in this way, the number of repair dialogue between the user and system is kept to the minimum. For this reason, it is necessary to select these primitives that corresponds as closely as possible to the action expressed in the chunks (see section 4).

A complete IBL requires several steps to transform a spoken chunk into a robot action (Table 1). First, the system must be able to convert speech into text. After that, some syntactic parsing and semantic analysis is carried out. Then at the functional mapping level, the system must be able to transform the user utterance into internal symbols that the robot can understand. By understanding we mean here that there is a correspondence between symbols and actions or real-world objects. In this way, the appropriate procedure can be called to act on the sensors and motors according to the user intentions.

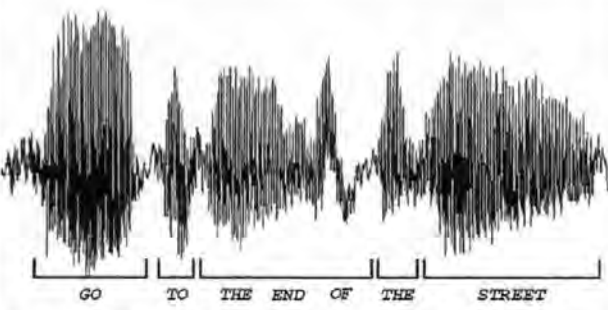
Analysis			Repair
	Speech recognition		
↓	Tagging	Go/VB to/TO the/DT end/NN of/IN the/DT street/NN	↑
↓	Syntactic Parsing	[VG go] [to to] [NG the end of the street]	↑
↓	Semantic Analysis	Robot (x), end_of_street (y), request_go (x, y)	↑
↓	Functional Mapping	Goto("end_of_street")	↑
↓	Robot program	Until found(end_of_street) follow_the_road()	↑

Table 1. From speech to action. The various steps involved in the transformation of a user command into the corresponding action are shown here.

Section 2 clarifies how symbol-level description and low-level sensory motor action procedures are integrated. The proposed representation of procedural knowledge is also described. In section 3 the system architecture is described.

The problems of considering the appropriate selection of action primitives is described in section 4 by analyzing recorded route instructions, and establishing a list of actions that are natural to users. The results of this

investigation are also discussed. These implications and other findings are discussed in section 5, along with the question of how the proposed system compares to other approaches. The conclusion follows in section 6.

2 The IBL model

2.1 Symbolic learning

The learning process is based on predefined initial knowledge. This "innate" knowledge consists of primitive sensori-motor procedures with names, such as "turn left", "follow the road" (the choice of these primitives is explained in sections 2.3 and 4). The name is what we call here a "symbol", and the piece of computer program that controls the execution of the corresponding procedure is called the "action" (Figure 1A). As each symbol is associated with an action, it is said to be "grounded".

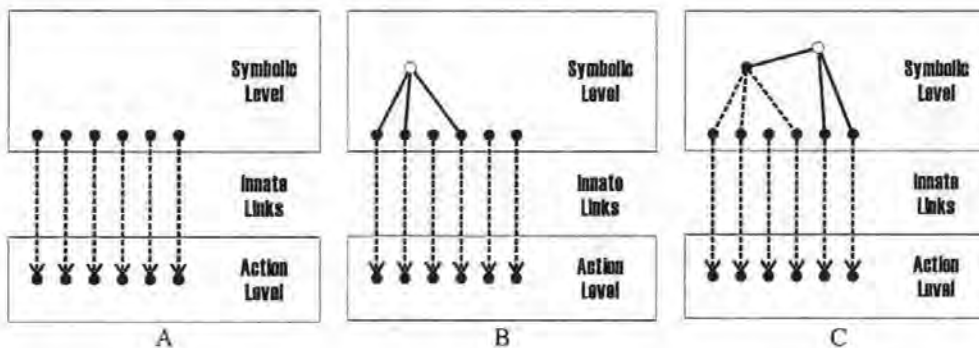


Figure 1. Symbolic learning. (A) is a schematic representation of the initial system, comprising symbols associated with pre-programmed (innate) primitive action procedures. In (B) the user has defined a new procedure (open circle) as a combination of symbols. The new symbol is grounded because it is a construct of grounded symbols. In (C), the user has defined a new procedure that combines a procedure previously defined by himself with primitive action procedures.

When a user explains a new procedure to the robot, say a route from A to B that involves a number of primitive actions, the IBL system, on the one hand, creates a new name for the procedure, and, on the other hand, writes a new piece of program code that executes that procedure and links the code with the name (see section 2.2 for details). The code refers to primitive actions by name. It does not duplicate the low-level code defining these primitives. For that reason, the new program can be seen as a combination of symbols rather than a combination of actions (figure 1B). As all new procedures are constructed from grounded primitives, they become also grounded by inheritance and are "understandable" by the system when referred to in natural language. When explaining a new procedure, the user can also refer to old procedures previously defined by himself. In that way the complexity of the robot's symbolic knowledge increases (fig. 1C).

2.2 Knowledge representation

The internal representation needs to support three functions: (i) formal modeling of NL route descriptions; (ii) internal route planning for determining whether a given route description is sufficiently specified; and (iii) the generation of procedures for navigation at execution time. These three functions require different representations that will be described in turn.

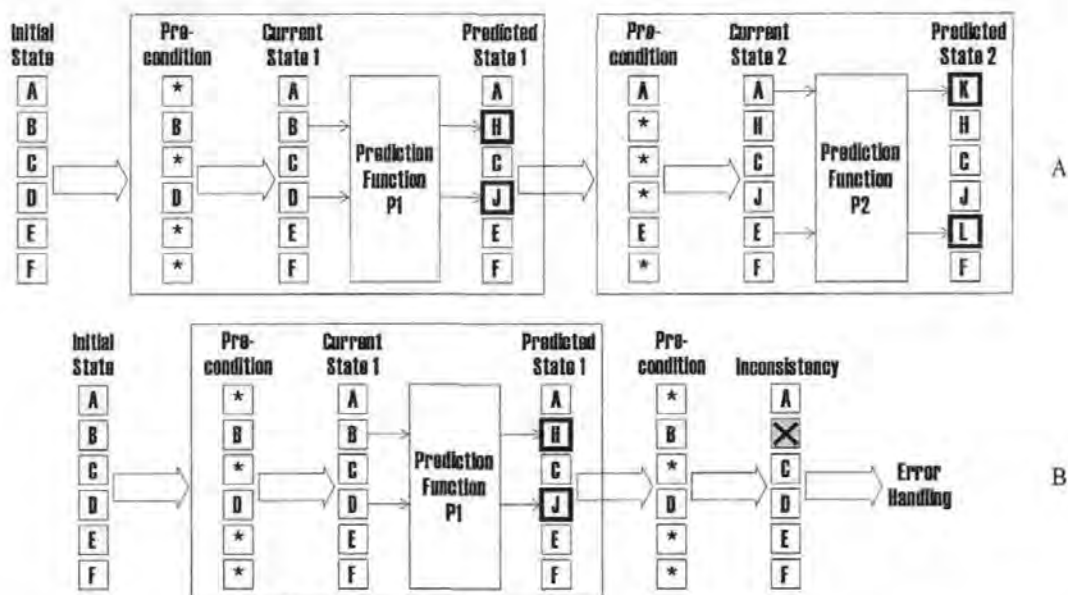


Figure 2. Route instruction verification. (A) For each procedure there is a prediction function that transforms a state vector into its future value. The function first determines if the input state satisfied the minimal criteria ("pre-condition") to enable the procedure to be executed. An action is executable only if selected elements of the state vector have required values. If this is the case, the next state is predicted and processed by the prediction function associated with the next procedure in the instruction. Each action modifies certain components of the state vector, and leaves the other unchanged. (B) If the predicted state produced by one procedure does not allow the next procedure to be executed, an error handling process is initiated. (Note: the "initial state" in the text corresponds to the "current state" in the figure).

- (i) The utterances of the user are represented using the Discourse Representation Structure (DRS) [9]. This is then translated into symbols representing procedures or is used to initiate internal functions such as execution of a command or learning of a series of commands (section 3).
- (ii) When the user describes a route as a sequence of actions, it is important for the robot to verify if this sequence is executable. The approach proposed here associate each procedure with a triplet $S_i A_{ij} S_j$ with properties similar to productions in SOAR [8]. The state S_i is the initial state in which the action A_{ij} can take place. It is the pre-condition for action A_{ij} . The state S_j is the final state, resulting of the action of A_{ij} applied to the initial state (figure 2 clarifies the difference between "initial state" and "pre-condition"). For a sequence of actions to be realisable, the final state of one action must be compatible with the pre-condition of the next one. To enable this verification, the robot must be able to "imagine" the consequence of an action. For that purpose, a PREDICTION function is associated with each primitive action, and with each newly created procedure. Figure 2 illustrates the use of the prediction function during verification of the consistency of the sequence of instructions from the user. It should be noted that this process also helps detecting some of the errors in natural language processing.

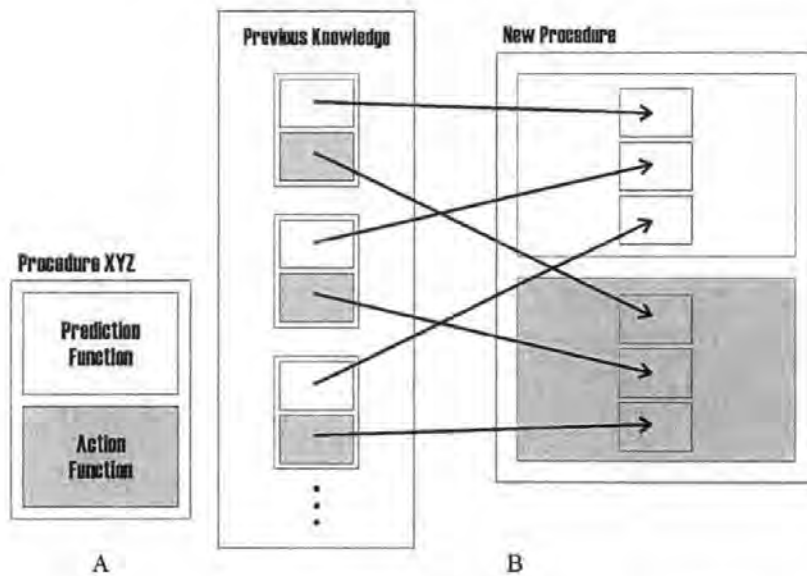


Figure 3. Procedural knowledge representation. (A) A procedure file contains an *ACTION* function that causes the physical displacement of the robot, and a *PREDICTION* function that calculates the future state of the robot resulting from the action. The *ACTION* is used during execution of a command, and the *PREDICTION* is used for consistency checking during the learning process. (B) An instruction by the user results in a "New Procedure" file being written. In this file, the actions components of the requested primitive procedures are combined (in the form of function calls) to create the new *ACTION* function, and the prediction components are combined to create the new *PREDICTION* function. This includes an additional procedure-specific pre-condition.

(iii) When a robot executes a command, it executes a piece of program code that contains the sequence of primitive procedures to be executed. Thus, a key part of IBL is the generation of a program code. This is enabled by the use of a scripting language (section 3). This program is called the *ACTION* function. Both *ACTION* and *PREDICTION* functions are physically located in the same file that contains all information specific to a procedure. This is schematised in figure 3.

2.3 Sensory-Motor primitives

Sensory-motor primitives are defined as action-chunks that users usually refer to in unconstrained speech. These could be low-level procedures referring, for example, to robot wheel turns, distance vectors etc. or they can be high-level procedures like for example "turn left after the church" or "take the second exit off the roundabout". In natural language route instructions, low-level specification of actions generally does not appear. Instead, higher-level procedures are mentioned which will have to be pre-programmed and thus become the sensory-motor primitives in this context.

In this project we have defined primitives as procedures which take parameters. For example the action "take the second right after the post-office", maps to the primitive *turn* with parameters *second*, *right*, *after* and *post-office*. It is then a matter of correctly mapping user utterances to the right primitives and passing the right parameters to them.

3 System Architecture

The architecture is comprised of several functional processing modules (figure 4). These are divided into two major units: the Dialogue Manager (DM) and the Robot Manager (RM).

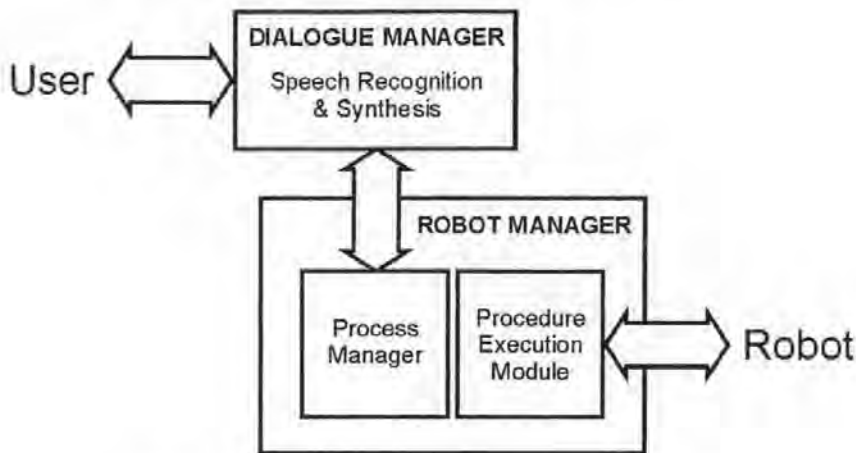


Figure 4. IBL system's architecture (see text for description).

The Dialogue Manager is a bi-directional interface between the Robot Manager and the user, either converting speech input into a DRS semantic representation [16], or converting requests from the Robot Manager into dialogues with the user. Its components are described in [9].

The RM deals with the DM's output and also with the learning and execution of the commands from the user. As shown in figure 4 the RM includes two modules: the Process Manager (PM) and the Procedure Execution Module (PEM). The PEM is responsible for carrying out the commands by the user. It executes procedures called by the Process Manager module.

The PM transforms the semantic representation produced by the DM into the internal language of the robot that includes learning and execution functions. Mapping symbols from the DRS onto the corresponding entities in the internal representation allows converting user requests into robot procedures with the right parameters. When successful, the PM starts the appropriate process either to execute the requested task by a call to the PEM or alternatively to build a new user-defined procedure explained by the user. When such mapping is not successful the RM must inform the DM, which starts a clarification dialogue with the user. Such mapping process is supported by a new specification language that expresses the relations between the symbols used in the DRS and the corresponding primitives. Thus to introduce new primitives, it is sufficient for the designer of an IBL system to change the grammar of the specification language without having to recompile any of the RM modules.

The Robot Manager is written using the Python² scripting language. C language extensions to Python are also used in case where speed is a constraint (for example in vision routines). An important feature of scripting languages such as Python is their ability to write their own code. For instance, a route instruction given by the user will be saved by the Robot Manager as a Python script that then becomes part of the procedure set available to the robot for execution or future learning.

It is important that the RM must listen to the DM and try to process its output but at the same time it should be able to send messages to the DM. The DM and the RM are designed as two different processes based on asynchronous communication protocols. These processes run concurrently on different processors. In this way, the system can handle, at the same time, both the dialogue aspects of an incoming request from the user (i.e. speech recognition and semantic analysis) and the execution of a previous user request (i.e. check if the request is in the system knowledge domain, and execute vision-based navigation procedures).

Two aspects are essential with this concurrent-processes approach. The first is to define an appropriate communication protocol between the two processes. The second is to define an appropriate architecture for the RM and DM allowing the two processes to both communicate with each other while performing other tasks. At present the use of context-tagged messages within a communication based on the Open Agent Architecture (OAA) [13] is evaluated.

Moreover, the system must also dynamically adapt itself to new user requests or to new internal changes, by being able to temporarily suspend or permanently interrupt some previous activity. For example the user may want to prevent the robot crashing against a wall and must therefore be able to stop the robot while the robot is

² <http://www.python.org>

driving towards the wall. Hence, the importance of a concurrent approach where the system constantly listens to the user while performing other tasks and at the same time is able to adjust the task if necessary.

4 Corpus Collection and Data Analysis

To evaluate the potential and limitations of IBL, a real-world instructions task is used, that is simple enough to be realisable, and generic enough to warrant conclusions that hold also for other task domains. A simple route scenario has been selected, using real speech input and a robot using vision to execute the instructed route (see 4.1 below for more details). The first task in the project is to define the innate actions and symbols in the route instruction domain. For this reason, a corpus of route descriptions has been collected from students and staff at the University of Plymouth. In section 4.2 and 4.3 corpus collection and data analysis are presented.



Figure 5. *Miniature town in which a robot will navigate according to route instructions given by users. Letters indicate the destinations and origins of various routes used in the experiment.*

4.1 Experimental Environment

The environment is a miniature town covering an area of size 170cm x 120cm (figure 5). The robot is a modified RobotFootball robot³ with an 8cm x 8cm base (figure 6A). The robot carries a CCD colour TV camera⁴ (628 (H) x 582 (V) pixels) and a TV VHF transmitter. Images are processed by a PC that acquires them via with a TV capture card⁵ (an example of such image is shown in figure 6B). The PC then sends motion commands by FM radio to the robot. During corpus collection, the PC is also used to record instructions given by subjects.

³ Provided by Merlin Systems (<http://www.merlinsystemscorp.com/>)

⁴ Provided by Allthings Sales and Services (<http://www.allthings.com.au/>)

⁵ TV Card: Hauppauge WinTV GO



Figure 6 A. Miniature robot (base 8cm x 8cm). B. View from the on-board colour camera

The advantage of a miniature environment is the ability to build a complex route structure in the limited space of a laboratory. The design is as realistic as possible, to enable subjects to use expressions natural for the outdoor real-size environment. Buildings have signs taken from real life to indicate given shops or utilities such as the post-office. However, the environment lacks some elements such as traffic lights that may normally be used in route instructions. Hence the collected corpus is likely to be more restricted than for outdoor route instructions. The advantage of using a robot with a remote-brain architecture [7] is that the robot does not require huge on-board computing and hence can be small, fitting the dimensions of the environment.

4.2 Collection of a corpus of route instructions

To collect linguistic and functional data specific to route learning, 24 subjects were recorded as they gave route instructions for the robot in the environment. Subjects were divided into three groups of 8. The first two groups (A and B) used free flow speech, to provide a performance baseline. It was assumed that a robot that can understand these instructions as well as a human operator would represent the ideal standard. Subjects from group C were induced in producing shorter utterances by a remote operator taking notes.

The groups A and B were told that the robot was remote-controlled and that, at a later date, a human operator would use their instructions to drive the robot to its destination. It was specified that the human operator would be located in another room, seeing only the image from the wireless on-board video camera. This induced subjects to use a camera-centred point of view relevant for robot procedure primitives and to use expressions proper for human communication. Subjects were also told to reuse previously defined routes whenever possible, instead of re-explaining them in detail. Each subject had 6 routes to describe among which 3 were "short" and 3 were "long". The long routes included a short one, so that users could refer to the short one when describing the long one, instead of re-describing all segments of the short one. This was to reveal the type of expressions used by users to link taught procedures with primitive ones. Each subject described 6 routes having the same starting point and six different destinations. Starting points were changed after every two subjects. A total of 144 route descriptions were collected. For more details about collection and analysis of the corpus see [1]

4.3 Corpus Analysis: The functional vocabulary

The aim of the corpus analysis is twofold. First, to define the vocabulary used by the users in this application, in order to tune the speech recognition system for an optimal performance in the task. Secondly, to establish a list of primitive procedures that users refer to in their instructions. The aim is to pre-program these procedures so that a direct translation from the natural language to grounded symbols can take place. In principle, if the robot does not know a primitive procedure, the user could teach it. Hereafter, we report on the functional analysis of the corpus. The reader interested in the task vocabulary can refer to [1]. The functional vocabulary is a list of primitive navigation procedures found in route descriptions.

The initial annotation of instructions in terms of procedures, as reported here, is somehow subjective, and influenced by two considerations. (i) The defined primitives will eventually be produced as C and Python Programs. It was hoped that only a few generic procedures would have to be written. Therefore, the corpus has been transcribed into rather general procedures characterised by several parameters (table 2). (ii) An important issue is knowledge representation. According to the SAS representation discussed in section 2.2, the

executability of primitives can only be evaluated if their initial and final states are defined. Subjects however rarely specified explicitly the starting point of an action and sometimes did not define the final state in the same utterance. Nevertheless, it was assumed that the system would be able to infer the missing information from the context. Therefore, procedures without initial or final state were considered to be complete, and were annotated as such. The specifications of primitive procedures are likely to evolve during the project.

	Count	Primitive Procedures
1	308	MOVE FORWARD UNTIL [(past over across) <landmark>] [(half_way_of end_of) street] [(after <number> <landmark> left right)] [road_bend]
2	183	TAKE THE <number> turn [(left right)] [(before after at) <landmark>]
3	147	<landmark> IS LOCATED [left right ahead] [(at next_to left_of right_of in_front_of past behind on opposite near) <landmark>] [(half_way_of end_of beginning_of across) street] [between <landmark> and <landmark>] [on <number> turning (left right)]
4	62	GO (before after to) <landmark>
5	49	GO ROUND ROUNDABOUT [left right] [(after before at) <landmark>]
6	42	TAKE THE <number> EXIT [(before after at) <landmark>]
7	12	FOLLOW KNOWN ROUTE TO <landmark> UNTIL (before after at) <landmark>
8	4	TAKE ROADBEND (left right)
9	4	STATIONARY TURN [left right around] [at from <landmark>]
10	2	CROSS ROAD
11	2	TAKE THE ROAD in front
12	2	GO ROUND <landmark> TO [front back left_side right_side]
13	1	PARK AT <location>
14	1	EXIT [car_park park]

Table 2. Primitive navigation procedures found in the route descriptions collected from groups A and C. Procedure 3 is used by most subjects to indicate the last leg of a route, when the goal is in sight.

This analysis methodology differs slightly from the one in [4]. In our analysis, there are no statements describing landmarks, as these are made part of procedures specifications, and consequently there are also no actions without reference to landmarks. Even when a subject specified a non-terminated action, such as "keep going", it was classified as "MOVE FORWARD UNTIL", assuming that a termination point would be inferred from the next specified action. The list of actions found in the route descriptions of groups A and C is given in table 2. It has been shown in [9] that the number of distinct procedures is increasing with the number of sampled instructions, but at a rate much smaller than the number of distinct words. Here we discover on average one new procedure for every 38 route instructions, while with words, we discovered in average one new word for each instruction. New procedures typically are the least frequent in table 2.

5 Discussions

Teaching a route to a robot using natural language is an application of a more general instruction-based learning methodology. The corpus-based approach described here aims at providing users with the possibility of using unconstrained speech, whilst creating an efficient natural language processing system using a restricted lexicon. As mentioned in section 2.3, primitives are quite complex procedures. Section 4.3 describes how the primitives were extracted from a corpus recorded by a group of people, mostly students, from various fields of study. They spoke freely to the robot using human-like expressions and therefore the primitives extracted from what they said reflect the amount of "knowledge" naive users would expect the robot to have. The level of complexity of the primitives therefore depends, not only on the nature of the natural language application but also on its users and their expectations of the robot. If the subjects of our corpus were robot engineers, for example, and were told that the robot does not know how to move or turn prior to their route instructions they may have produced a different corpus from which different primitives would have been extracted.

An important finding in [9] was that functional vocabulary is not closed. Hence, at some point in a robot's life, the user may have to teach it new primitives. For that purpose, the robot would need to possess an additional set

of low level primitives, which correspond to lower level robot actions. Examples of such primitive learning are found in [5] and [14]. With our approach, this would require the collection of a new corpus to determine the necessary additional primitive procedures. Another solution could lie in an appropriate dialogue management to suggest a reformulation of the instruction. It is expected that with the corpus-based method used here, the frequency of such "repair dialogues" will be minimised. An open question is the detection of new functions in the user's utterance, as the lexicon may not contain the required vocabulary.

The approach to robot control described may be seen as an attempt to integrate the good properties of Behaviour-based control and classical AI. Behaviour-based control is an effective method for designing low-level primitives that can cope with real-world uncertainties, and AI has developed effective tools for symbol manipulation and reasoning (for a more detailed discussion about hybrid systems see for example [10]). However, the system differs in several ways from both methods. Here, the corpus defines what symbols and primitives to use. Consequently, some of the primitives are rather complex functions, involving representations of the environment and planning. These primitives are not always compatible with the representation-less philosophy of behaviour-based systems. On the AI side, the system does not use the full range of reasoning capabilities offered by systems such as SOAR. There are no other aims in symbolic processing than verifying the consistency of instructions, and the construction of new procedure specifications.

Other previous work on verbal communication with robots has mainly focused on issuing commands, i.e. activating pre-programmed procedures using a limited vocabulary. Only a few research groups have considered learning, i.e. the stable and reusable acquisition of new procedural knowledge. [6] used textual input into a simulation of a manipulator with a discrete state and action space. [3] used voice input to teach displacements within a room and mathematical operations, but with no reusability. In [15] textual input was used to build a graph representation of spatial knowledge. This system was brittle due to place recognition from odometric data and use of IR sensors for reactive motion control. Knowledge acquisition was concurrent with navigation, not prior to it. Whereas in [11], the system could learn new actions through natural language dialogues but only while the robot was performing them (i.e. it could only learn a new route from A to B while it was actually moving from A to B and dialoguing with the user).

In the IBL system described here, learning operates purely at the symbolic level; hence it can be done prior to performance. The ability to predict future states enables to engage in a verification dialogue before execution errors occur. If environmental conditions change such that an instruction is not valid anymore, this can be detected from the mismatch between the expected result and the actual one. Learning however is not autonomous. The system requires interaction with a human user to learn new symbols and their meaning. This simplifies the design of the robot due to the transfer of part of the cognitive load to the user. Future experiment will reveal if this approach results in effective and socially acceptable helper robots.

The design of an IBL system requires, as expected, specialists in NL processing and *speech recognition*, as well as specialists in artificial vision and robot control. Here we found that significant work was also required in extracting from the semantic representation of the user's utterance the corresponding robot-executable procedures. It is hoped that this process will be simplified in the future by using the new specification language currently developed as part of the project.

6 Conclusions

In this paper, it was noted that domestic robots, which cannot learn from their users will be of limited use. The Instruction-Based Learning method (IBL) has been presented in the special case of route instructions. A key task in an IBL system is the translation from Natural Language (NL) instructions to robot-understandable procedures. The corpus-based approach has been proposed here to optimise such translation. It defines a task domain specific lexicon and set of primitives. This results in the implementation of a constrained language and limited task capabilities. However, it is expected that within a given task domain this will maximise the use of spontaneous speech and NL conversion efficiency. Only 14 primitives have been, but these are complex robotics procedures, involving visual search and planning. We believe that this is required to ensure efficient communication with a naive user. But the set probably is not closed. In other words, users at some time are likely to refer to primitives for which there is not preprogrammed counterpart in the robot's repertoire. It is likely that the dialogue management will play a key role in handling such situations.

Acknowledgement: This work is supported by EPSRC grants GR/M90023 and GR/M90160. The authors are grateful to A. Cangelosi and K. Coventry for enlightening discussions.

References:

- [1] Bugmann G., Lauria S., Kyriacou T., Klein E., Bos J. and Coventry K. (2001) "Using Verbal Instruction for Route Learning", Proc. of 3rd British Conf. on Auton. Mobile Robots and Autonom. Systems: Towards Intelligent Mobile Robots (TIMR'2001), Manchester, 5 April.
- [2] Cangelosi A., Harnad S. (2001) The adaptive advantage of symbolic theft over sensorimotor toil: Grounding language in perceptual categories. *Evolution Communication*. (in press)
- [3] Crangle C. and Suppes P. (1994) *Language and Learning for Robots*, CSLI Lecture notes No. 41, Centre for the Study of Language and Communication, Stanford, CA.
- [4] Denis M. (1997) "The description of routes: A cognitive approach to the production of spatial discourse", *CPC*, 16:4, pp.409-458.
- [5] FLAKEY: www.ai.sri.com/people/flakey/integration.html
- [6] Huffman S.B. and Laird J.E. (1995) "Flexibly Instructable Agents", *Journal of Artificial Intelligence Research*, 3, pp. 271-324.
- [7] Inaba M., Kagami S., Kanehiro F., Hoshino Y., Inoue H. (2000) "A platform for robotics research based on the remote-brained robot approach", *International Journal of Robotics Research*, 19:10, pp. 933-954.
- [8] Laird J.E., Newell A. and Rosenbloom P.S. (1987) "Soar: An architecture for general Intelligence" *Artificial Intelligence*, 33:1, pp.1-64.
- [9] Lauria S., Bugmann G., Kyriacou T., Bos J., Klein E. (2001) "Personal Robot Training via Natural-Language Instructions", *IEEE Intelligent Systems*, 16:3, pp. 38-45.
- [10] Malcom C. M. (1995), The SOMASS system: a hybrid symbolic and behaviour-based system to plan and execute assemblies by robot. In J. Hallam, et al. (Eds), *Hybrid problems and Hybrid solutions* pp 157-168. Oxford: ISO-press.
- [11] Matsui T., Asoh H., Fry J., et al. (1999) Integrated Natural Spoken Dialogue System of Jijo-2 Mobile Robot for Office Services, <http://citeseer.nj.nec.com/matsui99integrated.html>
- [12] Miller G. (1956) 'The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity Processing Information'. *The Psychol. Review*, v. 63, p. 81-97
- [13] Open Agent Architecture <http://www.ai.sri.com/~oaa/>
- [14] Seabra Lopes, L. and A.J.S. Teixeira (2000) Human-Robot Interaction through Spoken Language Dialogue, *Proceedings IEEE/RSJ International Conf. on Intelligent Robots and Systems, Japan*.
- [15] Torrance M.C. (1994) *Natural Communication with Robots*, MSc Thesis submitted to MIT Dept of Electrical Engin. and Comp. Science.
- [16] Traum, D., J. Bos, R. Cooper, S. Larsson, I. Lewin, C. Matheson and M. Poesio (1999) A model of dialogue moves and information state revision. Trindi Report D2.1. www.ling.gu.se/projekt/trindi/publications.html

	Stanislaw Lauria is currently a research fellow at the University of Plymouth. He received a Laurea in Physics from the Università di Napoli and a PhD degree in Cybernetics from the University of Reading. He has been research fellow at the University of Reading. His research interests are in the area of Neural Networks, Artificial Intelligence and robot vehicles. He can be contacted at stasha@soc.plym.ac.uk
	Guido Bugmann is a senior research fellow in the University of Plymouth's School of Computing, where he develops vision-based navigation systems for robots and investigates biological planning and spatial memory. He previously worked at the Swiss Federal Institute of Technology in Lausanne, NEC's Fundamental Research Laboratories in Japan and King's College London. He has three patents and more than 90 publications. Bugmann studied physics at the University of Geneva and received his PhD in physics at the Swiss Federal Institute of Technology in Lausanne. He is a member of the Swiss Physical Society, the Neuroscience Society, and the British Machine Vision Association. Contact him at the Centre for Neural and Adaptive Systems, School of Computing, University of Plymouth, Drake Circus, Plymouth PL4 8AA, UK; gbugmann@soc.plym.ac.uk.
	Theodoris Kyriacou is currently studying for a Ph.D. in Instruction Based Learning for Mobile Robots in the School of Computing of the University of Plymouth. He earned his B.Eng. (Honours) degree in Electronic Engineering Systems from the University of Sheffield in 2000.
	Ewan Klein is a reader in the Division of Informatics at the University of Edinburgh and director of Natural Language Research at Edify Corporation. His research interests include computational approaches to phonology, syntax, and semantics; multimodal interfaces; natural language specification of hardware design; and dialogue with intelligent systems. He received a BS in social and political science and a PhD in formal semantics from the University of Cambridge and an MS in general linguistics from Reading University

Vision-Based Urban Navigation Procedures for Verbally Instructed Robots

Theocharis Kyriacou, Guido Bugmann, Stanislaw Lauria

Robotic Intelligence Laboratory, School of Computing, University of Plymouth, Plymouth, United Kingdom,
<http://www.tech.plym.ac.uk/soc/staff/guidbugm/robolab/robolab.html>

Abstract

Humans who explain a task to a robot, or to another human, use chunks of actions that are often complex procedures for robots. An instructable robot needs to be able to map such chunks to existing pre-programmed primitives. We investigate the nature of these chunks in an urban visual navigation context and describe the implementation of one of the primitives: "take the n^{th} turn right/left". This implementation requires the use of a "short-lived" internal map updated as the robot moves along. The recognition and localisation of intersections is done using task-guided template matching. This approach takes advantage of the content of human instructions to save computation time and improve robustness.

1. Introduction.

This work is part of a project on "Instruction-Based Learning" (IBL) where robots acquire user-specific skills based on verbal instructions given by the user. One of the issues in the project is the mapping from action chunks used in natural language to actions executable by the robot. The approach used here is to provide a set of pre-programmed primitives corresponding to action chunks referred to by users. This facilitates the mapping from the semantic analysis of the spoken input to a sequence of executable robot actions.

In an earlier part of this project, subjects were invited to speak to a small robot in a miniature town (Figure 1) and to explain to it how to navigate between two landmarks. The instructions were recorded, transcribed and analysed to identify chunks of actions [1]. These chunks were grouped into about 15 "primitive" procedures that are listed and discussed in section 2.

In implementing such primitives, one can take advantage of the content of verbal instructions to minimize computational time and improve robustness, for instance by limiting visual search to those features mentioned in the instructions. The example of the navigation primitive "take the n^{th} left turn" is detailed in section 3. This primitive requires counting of landmarks and imposes the use of a "short-lived" internal map of the environment. Detection of road features, such as

intersections and turnings, is implemented by matching feature templates in the internal map.

The requirements of natural language understanding induce the internal representation of a route as a sequence of high-level task specifications (primitives). This is in principle very robust against environmental variations, provided that the primitives handle these appropriately.

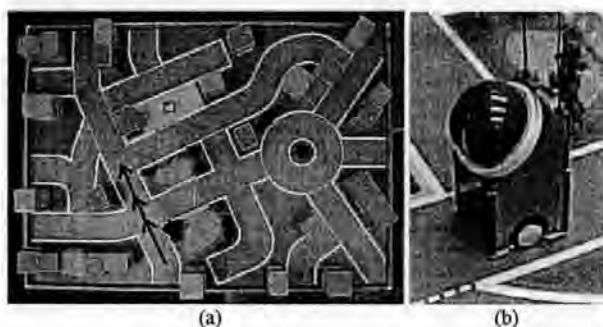


Figure 1: The miniature town (a) and robot with an 8x8 cm base (b). The marked path on the town image refers to the route followed by the robot in the example given in section 3.5 (where each arrowhead denotes a waypoint).

2. Action chunks in verbal instructions and corresponding primitives.

A corpus of route descriptions was collected from 24 subjects in the miniature town environment. Each subject was asked to give 6 route descriptions from a starting location (different for each subject) to a different destination each time. Subjects were instructed to assume the robot's point of view during their instructions. This seems to have worked since no description used a survey view.

A total of 144 route descriptions were analysed for their functional components (action chunks). These are the smallest possible units of information that compose each route instruction. Analysis was done by hand and all action chunks were then categorised into groups. The functional analysis revealed 15 functional groups (table 1). A similar approach for chunking of route descriptions was used by [2] and [3].

Table 1: Functional primitives extracted from the corpus.

	Primitive	Description
1	<code>go(description_1, landmark_1, preposition_1, description_2, landmark_2)</code>	Instructs the robot to follow a known route (with known starting point and destination).
2	<code>location_is(description_1, landmark_1, direction_1, preposition_1, description_2, landmark_2, description_3, landmark_3, ordinal_1)</code>	Specifies a location.
3	<code>destination_is(description_1, landmark_1, direction_1, , preposition_1, description_2, landmark_2, description_3, landmark_3, ordinal_1)</code>	Indicates the destination landmark.
4	<code>go_until(description_1, landmark_1, preposition_1, description_2, landmark_2)</code>	Follow known route to a landmark until a specified location in the route.
5	<code>exit_roundabout(ordinal_1, preposition_1, description_1, landmark_1)</code>	Take a specified exit from a roundabout.
6	<code>turn(ordinal_1, direction_1, preposition_1, description_1, landmark_1)</code>	Take a specified turn off a road.
7	<code>follow_road_until(preposition_1, description_1, landmark_1)</code>	Move forward until a certain location.
8	<code>rotate(direction_1, extend_1, around_1)</code>	Rotate to a certain extend.
9	<code>exit_from(description_1, landmark_1)</code>	Exit from a place, usually used for the car park.
10	<code>cross_to(description_1, landmark_1)</code>	Instructs the robot to cross the road to a landmark.
11	<code>enter_roundabout(direction_1)</code>	Enter the roundabout in a specific direction.
12	<code>park(preposition_1, description_1, landmark_1)</code>	Park on, or close, to a certain landmark.
13	<code>take_road(preposition_1, description_1, landmark_1)</code>	Take a road in view.
14	<code>goto_side(preposition_1, description_1, landmark_1)</code>	Go round a landmark to one of its sides.
15	<code>fork(direction_1)</code>	Follow a one of the two branches of a fork (Y split).

The number of functional groups found is subjective as it depends on the annotation method. Here, the annotation was done with two objectives in mind: 1. Produce parameterised primitives that generalize the description found in the corpus. For instance, the procedure designed for "turn left after the tree" should also work if "tree" is replaced by "Church". 2. An important issue is knowledge representation. Route following is a continuous chain of actions. When, as in this case, a route is represented as a sequence of primitives, the initial state of the robot in each primitive must be consistent with the final state in the previous primitive. Therefore, all actions referred to by subjects were assumed to have an initial and a final state. Subjects however rarely specified explicitly the starting point of an action and sometimes did not define the final state in the same utterance. It was assumed that the IBL system would be able to retrieve missing information from the context. For instance, when a subject specified a non-terminated action, such as "keep going", it was classified as "follow the road until", assuming that a termination point would be inferred from the next specified action.

Not all functional primitives in table 1 are purely navigation tasks. For example "go" consists mainly of retrieving from memory the list of primitives

corresponding to a given route, and "location is" specifies spatial relations between landmarks. In contrast, "destination is" is found at the end of a route description to indicate the location of the goal. The robot needs then to find its way to that location.

3. Implementation example: The primitive "turn()".

3.1. The parameter combinations for the "turn" primitive.

Four different combinations of parameters can be passed to the "turn" primitive procedure (table 2). The program implementing the primitive executes a different sequence of operations depending on the combination of parameters passed. For each of the four-parameter combinations a dedicated sub-routine is called in the primitive procedure. The next section shows the pseudo-code for the second case in the table above.

3.2. Pseudo-code for the case "take the n^{th} turn left/right".

In the first step of the pseudo-code in table 3, the templates selected for this case represent straight or

Table 2: Different combinations of parameters of the “turn” primitive procedure. The examples indicate some of the values that the parameters can take.

Parameter combination	Example
<code>turn(direction_1)</code>	“Take a right turn” “Turn left”
<code>turn(ordinal_1, direction_1)</code>	“Take the second left turn”
<code>turn(direction_1, preposition_1, landmark_1)</code>	“Turn left after the post-office”
<code>turn(ordinal_1, direction_1, preposition_1, description_1, landmark_1)</code>	“Take the second turning after Tesco’s” “Turn left at the library” “Turn right after the tall blue building”

curved road segments and intersections of various angles (section 3.4).

Table 3: Pseudo-code for case “take the n^{th} turn left/right”. The resulting sequence of displacements is illustrated in section 3.5 for n (ordinal_1) = 2 and direction_1=left.

Define set of road features (templates – see section 3.4) to look for. Loop: { • Capture and process road image. • Update internal map & localize robot (see section 3.3). • Find best matching template in the map. • Execute procedure (e.g. robot motion) associated with the winning template. }
--

Templates are mapped to road-like areas in the top view projection of the image captured by the camera. The template with the best match will determine the action to be performed next. For example the templates for straight or curved road will cause the robot to move further along the road. The intersection templates can have one of two actions associated with them: 1. either cause the robot to move to the centre of the intersection and rotate in the direction of turn or 2. just move ahead along the road. The first action is associated with the intersection templates when approaching the n^{th} intersection. In this case the robot takes the turn and the loop is exited so that execution is passed to the primitive associated with the next chunk in the route description. The second action is associated with the intersection templates until (but excluding) the n^{th} intersection. In these cases the robot carries on following the road. In this procedure, the robot must keep track, not only of the number of intersections passed but also of their location. When an intersection is identified, its location is compared against a record of previously found intersections and if a relatively close match is not found, it is considered to be a new intersection.

Intersection locations are recorded in the egocentric reference frame of the robot. Each time the robot moves these are updated to reflect their new relation to the robot. To perform this updating, the robot must know by how much it has moved since the last image was taken. In our purely vision-based system, this is done by tracking the displacements of landmarks in the image, using a “short-lived” feature map, as described in

section 3.3

3.3. Short-lived map

A short-lived map serves several purposes during the robot’s navigation. It is used to compensate for the dead angles of the robot by recording visual information, to keep track of landmark locations (like the intersections in the example of the previous section), and for resolving spatial relationships between a landmark and the road (e.g. to define a road area “after” a building).

The map is constructed progressively as the robot moves using road surface and road edge information filtered out from the top view of the scene (this is illustrated in figure 3). This view is produced by applying a perspective transform to the camera image.

Road surface information is extracted from the top view image using chromaticity information. Chromaticity is an intensity-invariant two-dimensional vector describing colour. The two components of the vector are the ratios of red to blue and green to blue components of the RGB vector. A road surface likelihood image is constructed by assigning a value to each pixel location in the original image, which is proportional to the Euclidian distance between its chromaticity vector and a reference chromaticity vector. The reference vector is obtained by calculating the average chromaticity of a sample of road area in the first image along the route. To increase computation speed, a threshold is applied on the road surface likelihood image resulting in a binary image with either road-like or non road-like areas.

For road edge extraction, an illumination-invariant approach similar to the one suggested in [4] is used to discriminate the white lines (road markings) along each side of the road. This is done by effectively convolving a two-dimensional low-high-low intensity mask with the original image with the high intensity span of the mask being at least equal to the width of the road markings in the top view image. Again, the resulting image is thresholded to obtain a binary image. Column B of figure 3 shows examples of road edge and road-like surface images.

After the execution of each motion command, the map view is translated according to the expected motion vector of the robot, so reflecting its new expected pose in the environment. The difference between the expected and actual location of the robot is due to

motion errors. These are corrected by finding the best matching pose of the new top view in the map, then by translating the map again to reflect the actual position of the robot. The matching process uses only the road edge image rather than the road surface image because edges are robust features of the image and allow a more precise matching. To save computational time and limit the risk of matching the new view at the wrong location, not all the map is searched but only a limited area defined around the expected position of the robot in the map. To further improve speed, a crude search is performed initially using coarse steps of position and orientation. The search is then refined for a more accurate determination of the position and orientation (match vector). The resulting match vector is used to paste the road surface image of the top view on the map and to translate the map. The map is termed "short-lived" because it is only maintained for a limited area around the robot's position (e.g. the size of images in columns C and D of figure 3). These steps are illustrated in section 3.5. The road surface likelihood map is built for the purposes of template matching which is described in the following section.

3.4. Road-feature templates

Templates are binary images of local road features drawn at the same scale as the short-lived map of road-like areas (Figure 2).

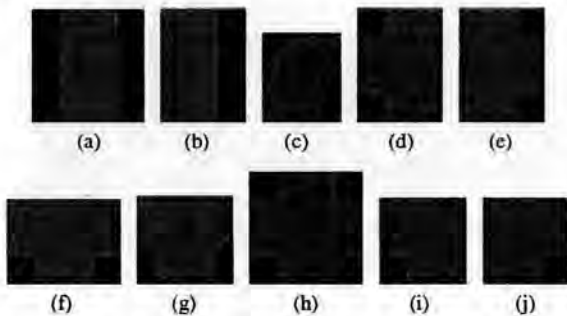


Figure 2: Examples of templates for: road following (a,b,c,i,j), for intersection detection (d,e,f,g,h). The light grey areas indicate road-like areas and the darker grey areas represent non-road areas.

For each road navigation task a subset of the available templates (corresponding to the task) is selected. For example, in the case of the action chunk: "take the second turn right" (illustrated in the following section) templates a, b and e are selected, for following the road and for detecting the intersection. The selected templates are continuously matched against the road surface map for each new image captured.

The matching process for each location and orientation of the template on the map produces a match quality measure made of the sum of two ratios: 1. the score, which is the sum of the matching road and non-road

pixels in the two images divided by the number of template pixels falling onto areas of the map where information is available from previous images and 2. the confidence factor, which is the fraction of the template area falling onto areas of the map with information. The best match of the template is the one where the sum of these two components is maximum. When the best vectors of all candidate templates are found, the "winner" template is selected as the one whose vector is associated with the best match. As with the map building, not all of the map is searched for the best match of the templates and the search is initially coarse, then refined. The position and orientation of the winning template defines the next motion command sent to the robot.

3.5. Example

Figure 3 shows the successive states of the short-lived maps and images processing results as the robot executes the instruction chunk: "take the second turning to the right". The path followed by the robot is marked on figure 1a. Each arrowhead indicates the points where the robot finishes an action and captures a new image to determine the next action. In step 1 there is no information on the edge map and so the edge and surface top views are simply pasted (in the egocentric reference frame) on the edge and surface maps respectively. In successive steps, this initial map is progressively shifted backwards and eventually rotated. Column D of figure 3 shows the best matching template in each step. Step 5 shows the resulting map after the rotation of the robot at the second right turn.

4. Concluding comments

Two aspects of natural language instructions influence the method proposed here for navigation in our urban model environment: Their division into action chunks and their under specified nature.

Each chunk can be considered as a search-and-act loop which exits when a condition is met. Primitives were written to reflect this. Like chunks, primitives loop until a condition is met. They have an initial state and a final state and only when their final state is reached, execution passes to the next primitive. The initial state of the next primitive must be consistent with the final state of the previous primitive to ensure consistency of execution.

In natural language, task specification is very abstract. For example in: "take the second turn right", the absolute locations of the intersections, their orientations or shapes are not given. These pieces of information must be retrieved in-situ by the robot to successfully complete the task. This is achieved here by the use of local road-feature templates that enable to recover orientation and shape information. Robustness is

achieved on one hand by defining very general template shapes and on the other hand by limiting visual search to those salient features selected by the

instructor.
To localize road features, the use of road surface information is deemed more robust than edge

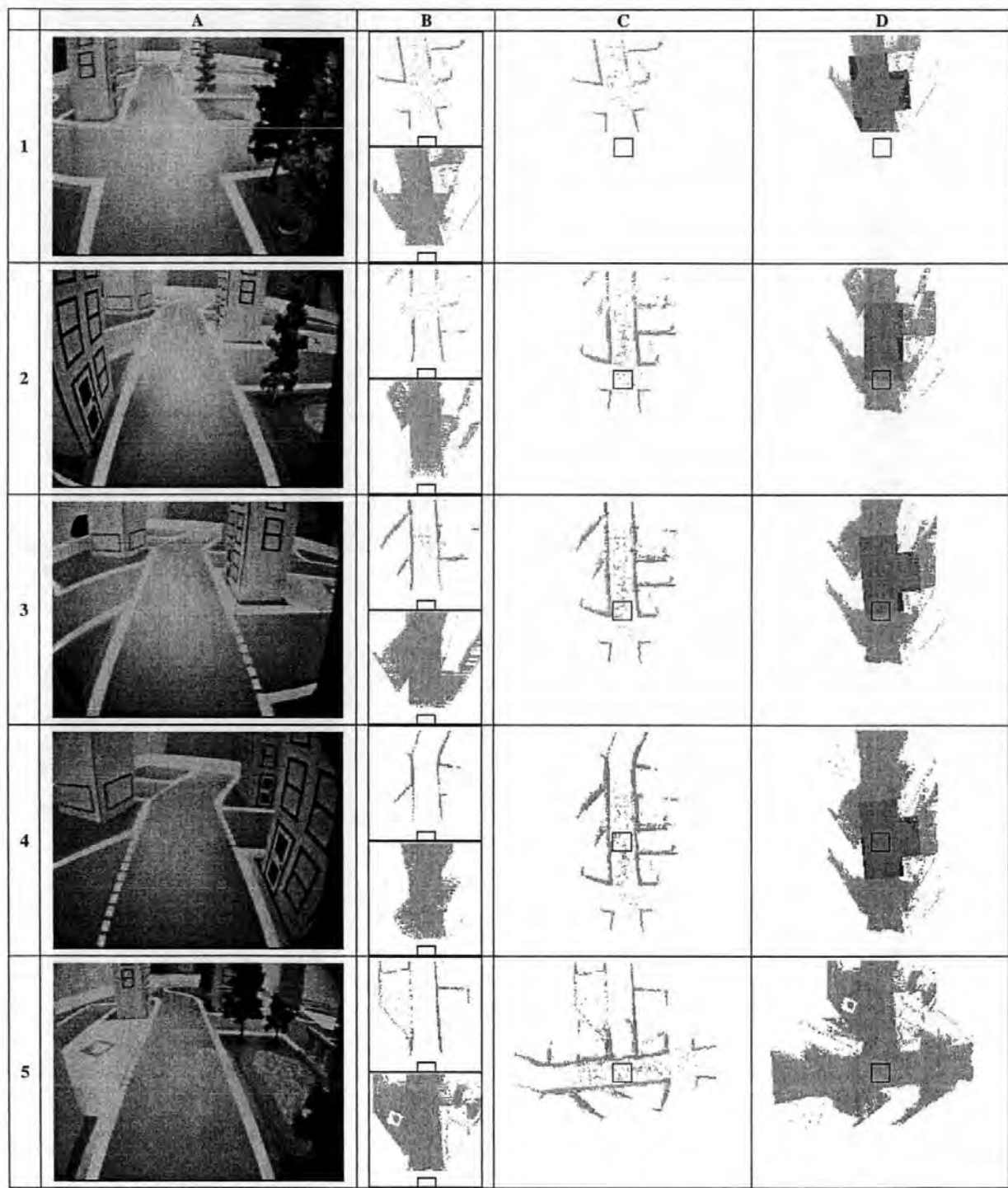


Figure 3: Step-by-step illustration of the execution of "take the second turning to the right". The execution is completed in five steps with the corresponding images at each step shown in the rows of the figure. Column A shows the camera view, column B shows the road edge and road surface images of the top view. In column C the road edge map is displayed and in column D the road surface map is shown. The best match position and orientation of the winning template for the step is also shown superimposed on the road surface map. Note also the indication of the position of the robot (black outline) in all the top view and map images.

information. A template has a good chance to match correctly even if road areas are partially missing, e.g. due to occlusion. For instance, in figure 3, image 1D, the "right turn" template matches at the correct location although the trees prevent full recovery of the road in the filtered image.

Most of the methods suggested in the past to recover the road layout from road images deal with the case of a straight or curved road extending in front of a vehicle, but without any turns, intersections, splits, roundabouts etc. [5], [6], [7], [8], [9] and [10]. These methods require that both sides of the road are visible (though not necessarily continuous) in the image to be able to recover the road. These methods are effective in cases where a vehicle needs to stay in the middle of a road lane when following a highway for example, but they are unsuitable in more complex urban environment.

Methods to recognize intersections on the road were proposed by [11] and [12]. In [11] a previously trained neural network is used to distinguish the road. The method lacks precision because of the neural network approach used and fails to accurately determine the location and orientation of the road. Furthermore, the method suggested for modelling a road intersection required the knowledge of either the position of the intersection, to determine its precise layout, or the layout, to find its position. A priori information for an intersection is also available in our case, through the natural language, but this is not absolute as far as the intersection's location or complete as far as its layout.

In [12] dynamic model building and matching are applied on a road surface likelihood image to determine the layout of the road. This method effectively finds intersections spurring from a straight road but would fail to find an intersection on a curve or an exit from a roundabout for example. Furthermore, the suggested method attempts to reconstruct the whole intersection. A strength of our method is that only the necessary road features (for the completion of the task in hand) are sought in the map, thus saving computational time and improving on system robustness.

Other landmarks (buildings, trees, lake, bridge etc.) of our model town are mentioned in the corpus and will need to be located to enable following the instructions. Ongoing work is addressing the problem of landmark recognition, the resolution of spatial relations between landmarks, and those between landmarks and the robot as sometimes mentioned in the corpus. The solutions to these problems are not expected to modify the navigation methodology described here but will rather merge with it.

In a real urban environment the template-based method should still work. The main issue is the segmentation of the scene into navigable and non-navigable areas.

Finally, an interesting property of such a system is that it has all the perceptual components required to robustly learn a route from experience (e.g. by following a human guide) in terms of reportable action chunks

rather than in terms of odometric measurements.

Applications of this work include intelligent helper robots such as autonomous wheelchairs for indoor/outdoor applications.

Acknowledgements: This work is supported by EPSRC grants GR/M90023.

5. References

- [1] Lauria S., Bugmann G., Kyriacou T., Bos J., Klein E., **Personal Robot Training via Natural-Language Instructions**, IEEE Intelligent Systems, 16:3, 2001, pp. 38-45.
- [2] Fraczak, L., **From route descriptions to sketches: a model for a text-to-image translator**, ACL-95 Student Session, MIT, Cambridge, USA, 1995.
- [3] Denis, M., **The description of routes: A cognitive approach to the production of spatial discourse**, CPC, 16:4, 1997, pp.409-458.
- [4] Broggi, A., **Robust Real-Time Lane and Road Detection in Critical Shadow Conditions**, Proc. of the IEEE International Symposium on Computer Vision, Coral Gables, Florida, November 1995, pages 353-358.
- [5] Waxman, A. M., Lemoigne, J. J., Davis, L. S., Srinivasan, B., Kushner, T. R., Liang, E., Siddalingaiah, T., **A Visual Navigation System for Autonomous Land Vehicles**, IEEE Journal of Robotics and Automation, Volume 3, Issue 2, April 1987, pp. 124-141.
- [6] DeMenthon, D., Davis, L. S., **Reconstruction of a Road by Local Image Matches and Global 3D Optimization**, Proc. of the IEEE International Conference on Robotics and Automation, 1990, pp. 1337-1342.
- [7] Kaske, A., Wolf, D., Husson, R., **Lane Boundary Detection Using Statistical Criteria**, Proc. of International Conference on Quality by Artificial Vision, QCAV'97, Le Creusot, France, 1997, pp. 28-30.
- [8] Sayd, P., Chapuis, R., Aufrere, R., Chausse, F., **A Dynamic Vision Algorithm to Recover the 3D Shape of a Non-Structured Road**, Proc. of the 1998 IEEE International Conference on Intelligent Vehicles, 1998, pp. 80-86.
- [9] Wilson, M. B., Dickson, S., **Poppet: A Robust Boundary Detection and Tracking Algorithm**, BMVC99 (British Machine Vision Conference 1999), pp. 352-361.
- [10] Wang, Y., Shen, D., Teoh, E. K., **Lane Detection Using Spline Model**, Pattern Recognition Letters, 21(8), July 2000, pp. 677-689.
- [11] Jochem, T. M., Pomerleau, D. A., Thorpe, C. E., **Vision Based Intersection Navigation**, Proc. of the 1996 IEEE Symposium on Intelligent Vehicles, September 1996, pp. 391-396.
- [12] Crisman, J.D., Thorpe, C.E., **SCARF: A Color Vision System that Tracks Roads and Intersections**, IEEE Transactions on Robotics and Automation, Volume 1, Issue 1, February 1993, pp. 49-58.

MIROSOT as a Teaching & Learning Tool

P. Robinson, G. Bugmann*, T. Kyriacou*, P. Culverhouse & M. Norman

Department of Communication & Electronic Engineering,
University of Plymouth, Drake Circus, Plymouth
Devon PL4 8AA, England

(Tel: 044 (0)1752 232572; Fax: 044 (0) 1752 232583; E-mail: probinson@plymouth.ac.uk)

*School of Computing, University of Plymouth, Drake Circus, Plymouth,
Devon, PL4 8AA, England

(Tel: 044 (0)1752 232566; Fax: 044 (0) 1752 232780; E-mail: gbugmann@plymouth.ac.uk)

Abstract: *This paper shows Mirobot league robot football successfully employed as an effective teaching and learning tool on a multidisciplinary, undergraduate engineering programme. An overview of robot football development at Plymouth is presented. Low cost Mirobot technology, designed in undergraduate project laboratories is described. The evolution of robot construction techniques, from Lego bricks through bent sheet metal boxes to a flexible, robust, inexpensive modular structure, is discussed. An off-the-shelf dc motor/gearbox/shaft encoder unit, controlled from an Atmel AT90f 8515 RISC processor and designed for Mirobot use is shown to have a wide range of mobile robots applications. One example of both high-level and low-level Mirobot control software is presented. Difficulties experienced developing the technology in an undergraduate student environment are discussed. Student feedback is reported to be excellent.*

Keywords: Mirobot, robot football, multidisciplinary, education

1. Introduction

The last decade has seen major changes in the engineering industry. Internet technologies, mobile communications and intelligent automation and robotics have all contributed to this change. Often undergraduate engineering programmes have failed to respond to changed circumstances. In the UK many prospective 18-year-old undergraduate students believe engineering courses are boring, stifle creativity and suppress originality. As a result relatively few able students wish to study engineering. It was against this background that the first undergraduate course in the UK in Robotics & Automated Systems (RAS) was developed at the University of Plymouth [1]. The aim of the course is to produce system integrators comfortable working across the boundaries of traditional engineering disciplines. It also aims to allow student flair, originality and imagination to blossom [2]. Robots, in their many guises, are excellent examples of integrated engineering systems. Robot football provides a focus for many disparate engineering disciplines, which together constitute the modern information society, and is one of a number of enabling technologies empowering RAS students in their own learning development. An added bonus is that Mirobot builds on the popularity of contemporary BBC TV programmes such as Robot Wars [3] and Techno Games [4].

The Mirobot project at the University of Plymouth came about as a result of a meeting in early 1997 at the IEE (Institution of Electrical Engineers), London. Dr Jeff Johnson of the Open University (OU) addressed the Robotics Committee of the IEE requesting help in the development of robot football in the UK. The author was the committee member tasked with investigating the request and reporting back to the committee. Later that

year Dr Johnson decided to build a Mirobot team and take it to the 1997 Mirobot championships in Taejeon, Korea. This team was built very rapidly using Lego motors and bricks for the body and a transputer based GM8104 graphics board as the frame grabber. Experience gained during the 1997 competition encouraged Dr Johnson and the author to co-operate in developing a joint OU/University of Plymouth team to represent England at the 1998 Mirobot World Championships in Paris. Professor Kim, from KAIST (Korean Advanced Institute of Science & Technology) provided further encouragement when he visited England in late 1997. During his visit a demonstration Mirobot competition was held at the OU, using Korean robots, and televised by the BBC.

In the six months leading up to the 1998 competition much development work was completed. Four final year Plymouth RAS undergraduate students worked on various aspects of the system design. A bent sheet aluminium chassis replaced the Lego bricks. Inexpensive dc motors incorporating gearboxes and optical encoders were bolted to the chassis. On-board electronics remained substantially unchanged and the vision system interface again used the GM8104 transputer board. It was apparent that the students had enjoyed the challenge of robot football and a decision was made to integrate robot football fully into the undergraduate RAS curriculum.

Critical assessments of the team performance in Paris led to a redesign of all parts of the system. Again RAS students completed much of the work. The transputer board was replaced by a Matrox Meteor II interface card, new robot bodies allowing in-situ removal of side panels were built and, most importantly, the on-board electronics were redesigned and constructed with the help of Merlin Systems Corporation, a company closely associated with

the University of Plymouth. The new, surface mount, electronics board is based upon the Atmel AT90f 8515 RISC processor with 8K of flash RAM memory. This new board provided improved functionality with reduced size. Fig. 1 shows a much used control board. Extra inputs for sensors are provided and the ability to transmit as well as receive UHF radio signals is included. Essentially this remains the board in use today.

At any one time there are many active versions of the various elements of the system. One student will be working on the communications problems whereas another may be testing PID algorithms on the robot. Several versions of software will be in development simultaneously. Often this creates project management difficulties for supervising academic and technical staff. Keeping track of the many system sub elements progressing in parallel can be difficult. Therefore a standard system is maintained and only upgraded by responsible staff. The most recent standard Plymouth Mirobot system is commercially available from Merlin Systems Corporation Ltd [5]. Many Merlin systems have been supplied to universities both in the UK and abroad.

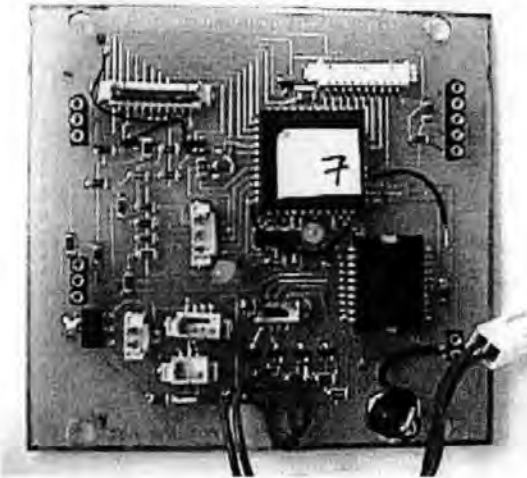


Fig. 1 – The Atmel control board

UK universities working with Mirobot robot football technology include Plymouth, Wales, the OU, Salford, Queens, Strathclyde and Essex. Dr Hu's research group at the University of Essex have developed their own Mirobot team and are actively researching in the field of multi-agent systems [6]. Meanwhile Dr Johnson at the OU is looking at, among other things, complexity inherent in-group and swarm behaviour [7].

Attempts to broaden the appeal of Mirobot robot football by the creation of a UK Mirobot league based in universities have been unsuccessful due to a lack of sponsorship. Both industrial and government sponsorship is being sought but so far without success. However, robot football remains popular with the media and regular request are made to stage competitions for both TV and technology exhibitions. In June 2001 the Plymouth team demonstrated Mirobot robots for a week at Earl's Court, London as part of the BBC Tomorrow's World Live

Exhibition. Many hundreds of school children had the opportunity to directly control robot footballers, via a PC keyboard, playing against autonomous players. They found it difficult to believe that no one was controlling the opposition, especially as they inevitably lost the match. All the children interviewed were captivated by this application of technology and, as a result, many left the exhibition seriously considering a career in engineering.

2. Mechanical Structure

Mirobot mechanical construction techniques at Plymouth have evolved through about seven generations. As stated above the initial 1997 design used standard Lego bricks and motors. This was followed by a variety of bent metal bodies working on the lotus flower principle. Basic design

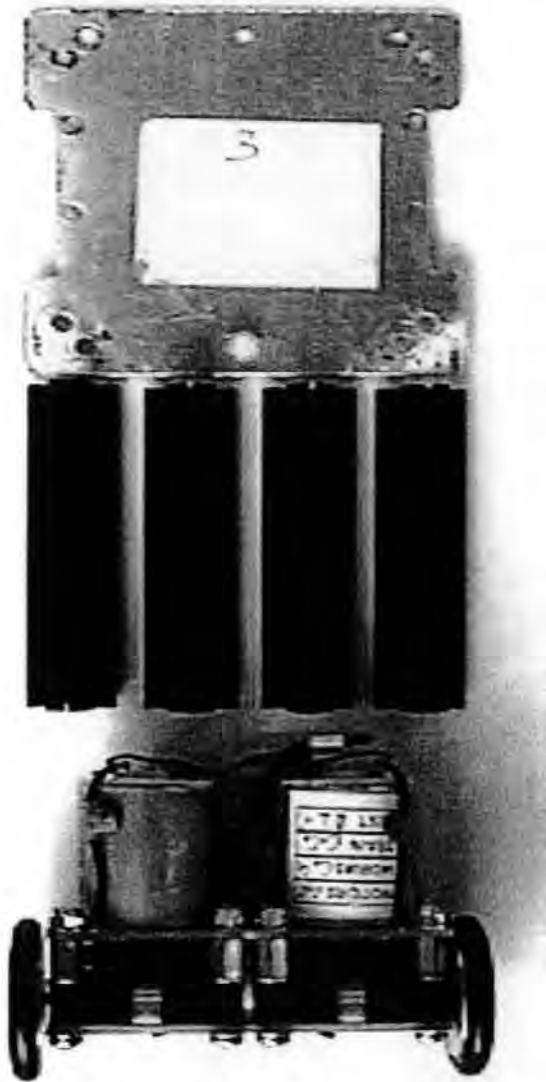


Fig. 2 Mechanical parts showing the base, pillars and motor/gearbox units

specifications included low cost, simplicity of build, i.e. no special machining required, ruggedness, good access and internal space for eight standard AA batteries. Access

proved to be a problem with the bent metal bodies. The present modular structure, Fig. 2, meets all the specifications to an acceptable degree and may be easily constructed by an undergraduate student without the need for special tools or skills.

Four extruded aluminum metal posts, cut from commonly available 2-meter lengths, screwed to an aluminum base plate constitute the basic robot skeleton. The control board, Fig. 1, is screwed to the other end of the posts to provide the top of the cube. Each side of the cube is made from sheet aluminum with cut outs for the wheel and ball apertures as required. The sides simply slot into the appropriate grooves of the mounting posts. Rigidity, robustness and ease of construction are combined with good accessibility to the batteries and internal parts.

High specification, small dc motors suitable for powering a Mirosoft robot tend to be expensive. Use of the Swallow matched motor/gearbox/shaft encoder units, at a unit price of about \$45, has provided a cost effective alternative [8]. At maximum efficiency, i.e. 6400 rpm, 6V and 0.57A, each 4.5-12V dc Mabuchi RC-280SA-20120 motor supplies a torque of 29 gm.cm. Good acceleration with speeds in excess of one meter/second is therefore possible. One drawback is that these motors tend to be electrically very noisy, especially when running at high speed. This may cause problems with false interrupts to the on-board micro-processor.



Fig.3 The robot skeleton

An assembled robot, minus its side panels is shown in Fig.3. The on/off switch is placed at the front above the ball capture aperture. Behind the motors there is sufficient room for 2*4 AA rechargeable battery packs. A simple nylon, half spherical skid is screwed to base at the front and back to provide stability. Because of the small wheel diameter, O-ring tyres and simple skids there is little frictional torque helping to guide the robot in a straight-line path. There is a natural tendency therefore for the robot to veer from a straight-line path.

The control board is mounted upside down and Velcro is used to fix the team shirt colours to the top. The RX module is plugged into the bottom of the board and may be easily changed from 418 MHz to 433 MHz as required. During matches there is a tendency for the RX module to be dislodged. Care must be taken to ensure that it is securely fixed.

3. System Software

The system software is written in C++, using where appropriate MFC facilities. This effectively means that the GUI (graphical user interface) & frame grabber are operated using MFC libraries. Figure 1 shows the overall software structure. At the centre is the match object through which all other objects are linked.

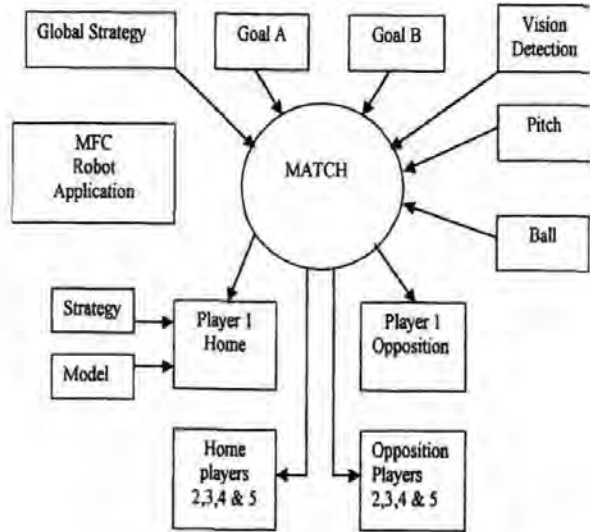


Fig. 4 System software structure

All the players and the ball are derived from a base class called *GameObject*. *GameObject* provides a range of services including default draw functions, automatic history recording and simple linear prediction. Any object within the system can have an arbitrary set of property variables (called a property set) which can be manipulated at runtime. The strategy property associated with each player can be used to point to a DLL which implements the strategy interface e.g. there is a goalkeeper DLL that determines the behaviour of the goalie, another DLL that is specific to a striker etc.

The global strategy allows property swaps so that a player strategy can be plugged in dynamically at run time, e.g. a goalkeeper can be changed to a striker as required. These properties may include player position, location restrictions, zoning and individual strategies. Model variables are common to all players. Dynamic modeling remains at best a good approximation. Inherent non-linearities such as friction inevitably lead to some uncertainty in model parameters. Notwithstanding these limitations a simplified model may be built around three global variables, namely; the dominant time constants, (maximum acceleration and deceleration) and maximum velocity.

4. Object Locations and Recognition

There has been much experimentation with the shape, size and colour of shirts. Presently the player's shirts are divided into equal sized quadrants. One quadrant is used for the team colour whereas the colour of the diagonally opposite quadrant identifies the player. The vision software searches a grabbed frame image for 'blobs', i.e. objects of interest, namely the players and the ball. In order to avoid identifying transient objects as blobs each blob must pass a fitness test. In practice if an object meets a min/max, x/y pixel size and is a non-pitch colour then it is a blob. After being identified as a blob the programme then attempts to link closely associated blobs.

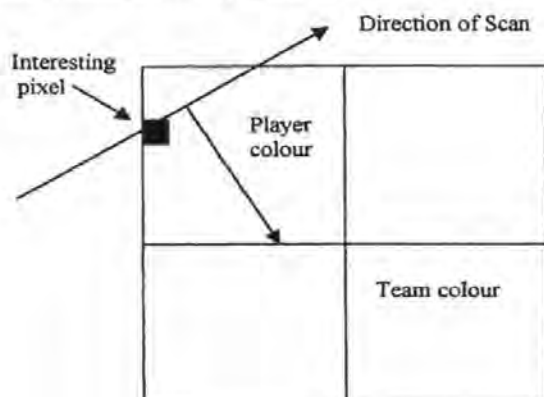


Fig. 5 Object location

The stored image is scanned until the first interesting pixel is found, Fig. 5. An interesting pixel is one with a hue value above the black pitch threshold intensity. Separation between RGB values is also tested. Once an interesting pixel is located the right hand scan is continued until a pitch pixel is found. The distance from the discovered interesting pixel to the next pitch pixel is bisected and a new scan started from the centre of the measured length and orthogonal to it. The new scan continues until another interesting pixel, i.e. one of a different colour, is detected. This procedure is repeated three times and a fitness test applied to see if a blob has been located. Orthogonal scanning provides both the approximate dimensions and centre point of the coloured object. As each scan progresses average values for R,G & B values are calculated. The result is used to calculate a hue value which when compared to a look-up table, identifies the specific colour. Four outcomes are possible., namely;

1. It is an opposing player.
2. It is a home team colour
3. It is a specific home team player.
4. It is the ball.

After storing the location and colour of this object the scan continues on its original path until all valid blobs have been located.

Linking blobs is the next stage. Blobs in close proximity, and identified as either home team colours or player identity, are assumed to be linked, i.e. they are part of the same robot shirt. This allows both the position and

orientation of the individual player to be determined. Problems can arise if two or more home players are next to each other. In the worst case scenario the orientation of a player may be in error by 180 degrees. During matches this effect can sometimes be observed but is usually self-correcting insofar as the confusion is eliminated immediately the robots separate.

5. The Robot Control System

Many robot control systems have been designed and tested. One such system is shown in Fig.6. where wheel two control is a mirror image of wheel one control. This particular robot control system responds to four basic commands, namely the required movement distance and speed for each wheel, i.e. D1, S1 and D2, S2. Usually there is no need to control the distance moved during a Mirobot game. However distance traveled is an interesting test of the low-level robot controller efficiency. In practice the basic unit of measurement is a single pulse from the shaft encoder situated between the motor and gearbox (G/B). Resolution calculated from 8 holes per encoder disk, 16:1 gear ratio and a 32mm diameter wheel is approximately 0.785 mm, i.e. more than adequate for requirements of MIROSOT competition.

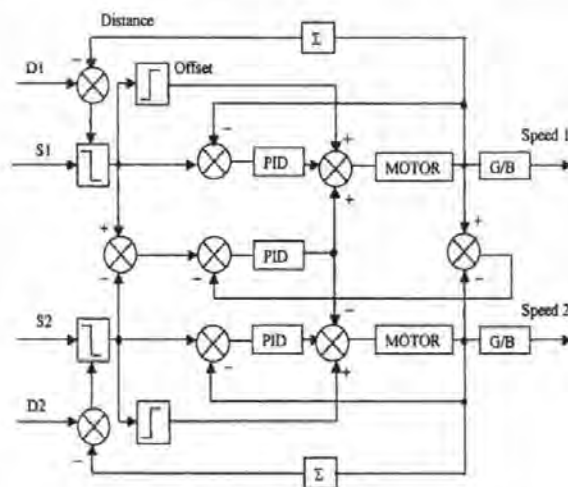


Fig.6 Robot control system block diagram

Each robot control system is subject to a variety of bench mark tests. These may consist of straight line/fixed distance movements, circles, squares and figures of eight. Because of inherent nonlinear problems achieving accurate control has proved to be difficult, especially in the case of a stand alone robot commanded to travel in a straight line for a fixed distance. Fixed distance movement requires four input commands, i.e. D1, S1 and D2, S2. For straight line movement D1 and S1 are equal to D2 and S2 respectively. Before the motor speed command can be activated it must pass through a normally-on switch controlled by the distance travelled circuit. Distance travelled is measured by summing the relevant wheel shaft encoder pulses and comparing the result with the input demand. When the two are equal the normally-on input speed switch is turned off thereby deactivating the motor. Providing the demand

distance has not been achieved $S1$ is compared to the actual motor speed and resulting error processed by a conventional PID controller.

Various techniques were used for setting the PID constants, including the classical Ziegler Nichols method. In practice empirical methods were found to provide the best results. Accurate, fast response, low speed control was found to be particularly difficult due to the relatively high amount of stiction in the motor gearbox drive unit. Speed demand is provided by a value in the range 0-255. During initial system testing there was found to be a dead band in the range 0-30, i.e. the minimum reliable speed was 8 cm/second. Attempts to cure this by increasing the integral speed of the PID controller led to typical limit cycle, oscillatory type of behaviour being superimposed on the behaviour of interest. A fixed value off-set, switched on by the presence of a speed input command and added to the input speed signal, Fig. 6, solved this problem.

Further practical difficulties result from the different characteristics of the motor/gearbox combinations. If identical signals are applied at both inputs it is invariably discovered that one wheel will accelerate at a different rate to the other thereby causing the robot to veer from the demanded straight line and move in an arc. A third PID controller, common to both wheels, helps overcome this problem. An error signal, generated from the difference between the demanded input speeds $S1$ and $S2$, is compared to an error signal based upon the actual difference between the wheels speeds. The two error signals are compared and the result processed by the third PID controller, the output of which is added, with appropriate polarity, to both motor drive signals. This causes the characteristics of the two motors to be drawn into line, resulting in straight line travel being achieved. The third controller also helps compensate for wheel spin due to skidding. When wheel spin occurs signals are also added, as appropriate, to $D1$ and/or $D2$ in order to ensure a correction is made to the sum of distance travelled measurements. Other practical problems encountered include noise spikes during high speed operation. This noise is interpreted as valid feedback resulting in a lower operational speed than would be expected.

6. The Student Experience

Final year undergraduate individual projects are scheduled for one day a week over a six month period. At the end of this time a comprehensive individual project report is submitted and each students attends two oral examinations (vivas). Every project is allocated a budget of about \$75 although this may be increased with the permission of the academic supervisor. It is clear that a student working alone would be unable to develop a complete Mirobot robot football system, both the complexity and cost are prohibitive. However over the last five years a great deal of development has been completed on a wide range of robot football technology. The individual student, looking to complete a robot football project, is able to build on this substantial body of work. In effect robot football is the focus for a very wide range of student project activities.

This is illustrated in Fig. 7. Some of the robot football links are obvious, such as wheeled robots, vision systems, UHF radio and control. However other links are not so clear. If, for example, a student is interested in say object oriented programming then robot football provides a ready application platform. On the other hand if AI (artificial intelligence) is the topic of interest then the multi-agent strategy and behaviour aspect of robot football provides an excellent application. Individual robots may be constructed quickly and cheaply using the standard format thereby freeing students to concentrate on their research area of interest. Alternatively components of the technology, e.g. the control card, may be used in a different application.

Over 50 final year engineering students have completed undergraduate projects linked to robot football activities [9]. Enthusiasm is such that many first and second year students work with the robot football group in addition to their scheduled programmes. Spin-off activities have proved to be beneficial to both staff and students. A second year digital system design module, ELEC212B, is based upon the Mirobot robot described above. Students study the mechanical and electronic design aspects of the robot. They then programme the robot in C, via the serial port of a PC, to perform a series of specified manouvres. Students benefit from hands-on experience and immediately see the effect of their programmes in the real, i.e. laboratory, world. Another module delivered to final year robotics students, CONT312A, concentrates on the mathematical analysis and modelling of mobile robots. Again the Mirobot robot is central to this work.

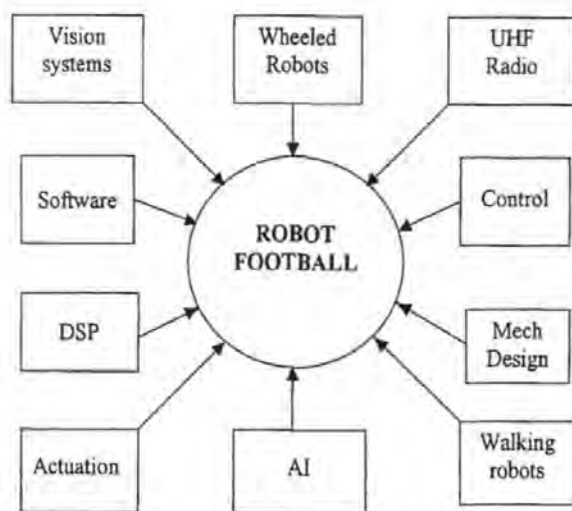


Fig. 7 Robot football technology

Trips to national and international competitions are very popular. Undergraduate students have taken part in Mirobot competitions in Manchester, London, Paris, Germany and Dubai. These visits constitute a well-earned reward for their hard work within the robot football group. An added benefit is that the students return full of enthusiasm and knowledge gained from discussions with other competitors.

From a staff perspective robot football complements many well established research interests in areas such as mobile robots, vision systems and control. Undergraduate project students working on various aspects of robot football technology may gain first hand experience of research problems by working closely with established academic staff. This can be an excellent method of recruiting the next generation of researchers. The student is able to gain experience in an area of interest while the academic staff member has the opportunity to assess the student's suitability as a full-time post-graduate researcher following an MPhil or PhD programme.

Up to date there have been no full-time, post-graduate researchers working on the Plymouth Mirobot system. From year to year academic and technical staff have provided continuity as one student cohort graduates and the next arrives. Because of severe demands elsewhere these staff can only devote a small part of their time to the project. Opportunities for advancing Mirobot technology to the best international standards have therefore been very limited. It is hoped in the near future to obtain funding to appoint full-time, post-graduate researchers to the team. If successful rapid progress can be expected in some of the more interesting areas such as team strategy and co-operative behaviour.

7. Conclusion

An inexpensive, robust Mirobot robot has been designed specifically for undergraduate project work. Mechanical construction requires no special tools or skills and is within the capability of all students on the Robotics & Automated Systems programme.

Robot football has proved to be an excellent teaching and learning tool. Two modules, one at the second year level another at the third year level, use the designed robot and its underpinning technology as the focus for study and experimentation. Large numbers of final year undergraduate students have chosen individual projects associated with robot football. Student feedback is overwhelmingly positive.

It was shown that many young people are fascinated with the idea of autonomous, intelligent robot teams playing football. This helps dispel the boring image of engineering and encourages more young people to study technological subjects at university.

References

- [1] Robinson, P. "A New Robotics Degree - The Plymouth Experience" Proceedings of the IEE Colloquium on Robotics in Education, pp. 1-8, 1995.
- [2] Robinson, P. "Robotics Education and Training: A Strategy for Development" Industrial Robot Journal, Vol.23, No.2, pp. 4-12, April 1996
- [3] <http://www.robotwars.co.uk>
- [4] <http://www.bbc.co.uk/science/robots>
- [5] <http://www.merlinsystems.com>

[6] B.Li, E. Smith, H. Hu and L. Spacek, "A Real-Time Visual Tracking System in Robot Soccer Domain", Proceedings of EUREL Robotics 2000, Salford, April 2000

[7] Johnson, J.H. & Sugisaka, M. "Complexity Science for the Design of Swarm Robot Control Systems" IECON-2000, IEEE Int. Conf. on Industrial Electronics, Control and Instrumentation, pp. 695-700, Nagoya, Japan, 22-28 October 2000

[8] <http://www.swallow.co.uk>

[9] <http://www.tech.plym.co.uk/robofoot>

Vision-Based Urban Navigation Procedures for Verbally Instructed Robots

Theocharis Kyriacou, Guido Bugmann, Stanislaw Lauria

Centre for Neural and Adaptive Systems, School of Computing, University of Plymouth
Drake Circus, Plymouth PL4 8AA, United Kingdom

<http://www.tech.plym.ac.uk/soc/staff/guidbugm/ibl/index.html>

Abstract

When humans explain a task to be executed by a robot they decompose it into chunks of actions. These form a chain of search-and-act sensory-motor loops that exit when a condition is met. In this paper we investigate the nature of these chunks in an urban visual navigation context, and propose a method for implementing the corresponding robot primitives such as "take the n^{th} turn right/left". These primitives make use of a "short-lived" internal map updated as the robot moves along. The recognition and localisation of intersections is done in the map using task-guided template matching. This approach takes advantage of the content of human instructions to save computation time and improve robustness.

1. Introduction.

This work is part of a project on "Instruction-Based Learning" (IBL) where robots acquire user-specific skills based on verbal instructions given by the user. One of the issues in the project is the mapping from action chunks used in natural language to actions executable by the robot. The approach used here is to provide a set of pre-programmed primitives corresponding to action chunks referred to by users.

This facilitates the mapping from the semantic analysis of the spoken input to a sequence of executable robot actions.

In an earlier part of this project, subjects were invited to speak to a small robot in a miniature town (Figure 1) and to explain to it how to navigate between two landmarks. The instructions were recorded, transcribed and analysed to identify chunks of actions (Lauria et al., 2001). These chunks were grouped into about 15 "primitive" procedures that are listed and discussed in section 2.

To successfully implement the primitives, the robot needs to manipulate spatial knowledge and keep track of its own position. This is achieved by using a "short-lived" internal map of the environment (section 3). Another requirement, the detection of road features such as intersections and turnings is implemented by matching local road feature templates on the internal map (section 4).

In implementing such primitives, one can take advantage of the content of verbal instructions to minimize computational time and improve robustness, for instance by limiting visual search to those features mentioned in the instructions. As an example, the execution of the navigation primitive "take the n^{th} left/right turn" is described in section 5.

The requirements of natural language

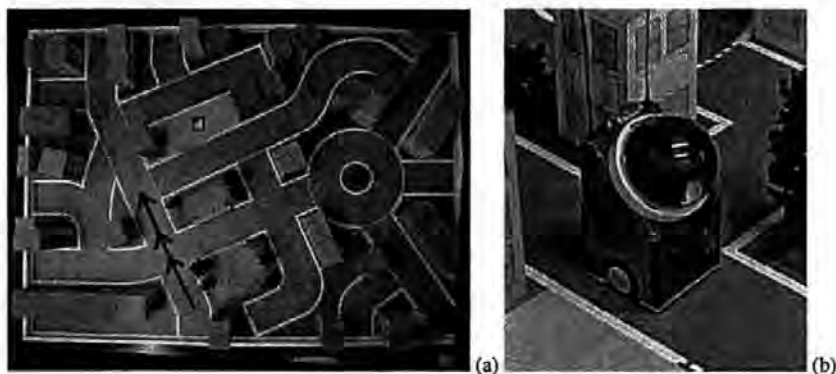


Figure 1: (a) The miniature town. The marked path on the town image refers to the route followed by the robot in the example given in section 5 (where each arrowhead denotes a waypoint). (b) The robot with an 8x8 cm base. The robot's wireless colour video camera sends images to a PC for processing and the PC sends motion commands by radio to the robot.

understanding induce the internal representation of a route as a sequence of high-level task specifications (primitives). This is in principle very robust against environmental variations, provided that the primitives can handle these appropriately. In this paper we focus on the detection of visual features of the road layout in a way that should be robust enough to allow traversal of complete routes.

2. Action chunks in verbal instructions and corresponding primitives.

A corpus of route descriptions was collected from 24 subjects in the miniature town environment. Each subject was asked to give 6 route descriptions from a starting location (different for each subject) to a different destination each time. A total of 144 route descriptions were analysed for their functional components (action chunks). The functional analysis revealed 15 functional groups (table 1).

This number is subjective as it depends on the annotation method. Here, the annotation was done with two objectives in mind: 1. Produce parameterised primitives that generalize the description found in the corpus. For instance, the procedure designed for “turn left after the tree” should also work if “tree” is replaced by “Church”. 2. An important issue is knowledge representation. Route following is a continuous chain of actions. When, as in this case, a route is represented as a sequence of primitives, the initial state of the robot in each primitive must be consistent with the final state in the previous primitive. Therefore, all actions referred to by subjects were assumed to have an initial and a final state. Subjects however rarely specified explicitly the starting point of an action and sometimes did not define the final state in the same utterance. It was assumed that the IBL system would be able to retrieve missing information from the context. For instance, when a subject specified a non-terminated action,

	Primitive	Description
1	<code>go(description_1, landmark_1, preposition_1, description_2, landmark_2)</code>	Instructs the robot to follow a known route (with known starting point and destination).
2	<code>location_is(description_1, landmark_1, direction_1, preposition_1, description_2, landmark_2, description_3, landmark_3, ordinal_1)</code>	Specifies a location.
3	<code>destination_is(description_1, landmark_1, direction_1, , preposition_1, description_2, landmark_2, description_3, landmark_3, ordinal_1)</code>	Indicates the destination landmark.
4	<code>go_until(description_1, landmark_1, preposition_1, description_2, landmark_2)</code>	Follow known route to a landmark until a specified location in the route.
5	<code>exit_roundabout(ordinal_1, preposition_1, description_1, landmark_1)</code>	Take a specified exit from a roundabout.
6	<code>turn(ordinal_1, direction_1, preposition_1, description_1, landmark_1)</code>	Take a specified turn off a road.
7	<code>follow_road_until(preposition_1, description_1, landmark_1)</code>	Move forward until a certain location.
8	<code>rotate(direction_1, extend_1, around_1)</code>	Rotate to a certain extend.
9	<code>exit_from(description_1, landmark_1)</code>	Exit from a place, usually used for the car park.
10	<code>cross_to(description_1, landmark_1)</code>	Instructs the robot to cross the road to a landmark.
11	<code>enter_roundabout(direction_1)</code>	Enter the roundabout in a specific direction.
12	<code>park(preposition_1, description_1, landmark_1)</code>	Park on, or close, to a certain landmark.
13	<code>take_road(preposition_1, description_1, landmark_1)</code>	Take a road in view.
14	<code>goto_side(preposition_1, description_1, landmark_1)</code>	Go round a landmark to one of its sides.
15	<code>fork(direction_1)</code>	Follow a one of the two branches of a fork (Y split).

Table 1: Functional primitives extracted from the corpus.

such as “keep going”, it was classified as “follow the road until”, assuming that a termination point would be inferred from the next specified action.

Not all functional primitives in table 1 are purely navigation tasks. For example “go” consists mainly of retrieving from memory the list of primitives corresponding to a given route, and “location is” specifies spatial relations between landmarks. In contrast, “destination is” is found at the end of a route description to indicate the location of the goal. The robot needs then to find its way to that location.

Different combinations of parameters can be initialised for each of the primitives. Not all possible combinations may be valid though. For each of the valid parameter combinations a dedicated sub-routine is called in the primitive procedure. Table 2 shows the four different combinations of parameters that can be passed to the “turn” primitive procedure. Section 5 describes how the sub-routine for the second case in the table is implemented.

In most primitive procedures the robot needs to navigate to a visually identified target location on the road. For example, in the “turn” primitive, regardless of the combination of parameters passed, the robot eventually needs to identify a specific turn, move to it and rotate to face the new direction. Our method to discriminate components of the road layout and navigate to them is described in the following sections.

3. Short-lived maps and self-localization

A short-lived map is a map of the immediate vicinity of the robot that is updated as the robot moves in its environment. The map records previously seen visual information which go out of view as the robot moves. The robot’s position is always centred on the map and facing towards the top of the map. As the robot moves the map is translated and rotated to maintain this frame of reference. In the

process, elements of the map that reach its edge will disappear. Thus the term “short-lived”.

The purpose of constructing such a map is to compensate for the dead angles of the robot (areas in the immediate locality of the robot which fall outside the visual field), to keep track of landmark locations and road layout features such as intersections and for resolving spatial relationships between a landmark and the road (e.g. to define a road area “after” a building).

Two versions of the short-lived map are used in this paper, the first represents the position of the road surface and the second represents the position of road edges. These maps are constructed using road surface and road edge information filtered out from the top view of the scene. This view is produced by applying a perspective transform to the camera image. This transform assumes a flat ground plane. Figures 2b and 2f show the top views of 2a and 2e respectively. This section describes how the road edge map is used to align new visual information with the ones existing in the map. The use of the road surface map for the detection of road features is described in section 4.

Road edge information is extracted from the top view image using an illumination-invariant approach similar to the one suggested in (Broggi, 1995). This approach discriminates the white lines (road markings) along each side of the road by effectively convolving a two-dimensional low-high-low intensity mask with the original image. The high intensity span of the mask is equal to the width of the road markings in the top view image. To increase computation speed in later stages, a threshold is applied on the road edge image resulting in a binary image with either road-edge or non road-edge pixels. Figures 2c and 2g are examples of thresholded road edge images of the top views in figures 2b and 2f respectively.

The top view of the scene is aligned with the map using the following steps. After the completion of each motion command, the map is translated

Parameter combination	Example
<code>turn(direction_1)</code>	“Take a right turn” “Turn left”
<code>turn(ordinal_1, direction_1)</code>	“Take the second left turn”
<code>turn(direction_1, preposition_1, landmark_1)</code>	“Turn left after the post-office”
<code>turn(ordinal_1, direction_1, preposition_1, description_1, landmark_1)</code>	“Take the second turning after Tesco’s” “Turn left at the library” “Turn right after the tall blue building”

Table 2: Different combinations of parameters of the “turn” primitive procedure. The examples indicate some of the values that the parameters can take.

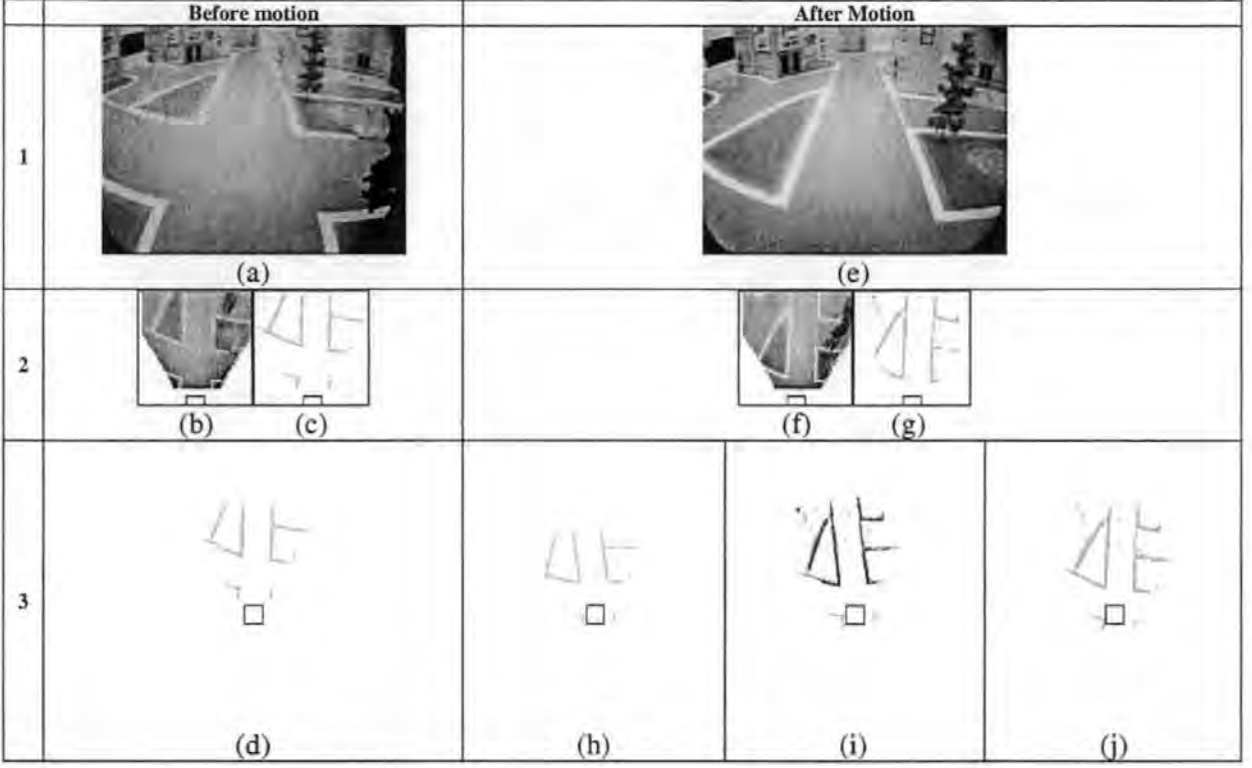


Figure 2: Camera view (row 1), top view (row 2) and road edge map (row 3) prior to robot motion (a to d) and after robot motion, showing expected position (h), best match of new top view on map (new view shown darker) and (i) and actual position (j) after translating map.

according to the motion command sent to the robot, so reflecting its new expected pose in the environment (figure 2h). Any difference between the expected and actual location of the robot in the map results from possible motion errors. These are corrected by using a newly captured image and by finding where the road edge image of the new top view best fits on the map. The map is then translated again to reflect the actual position of the robot.

The robot's position with regards to the top view image is a point outside the field of view called here "the pivot point" (figure 3). While searching for the best match, the top view image is displaced and rotated (vector $[x, y, \phi]$) so that its pivot point scans the map image.

The match quality Q_1 describes how the edges of the two images overlap. Q_1 is made of the sum of two ratios: 1. the score, which is the matching road

edge pixels in the intersecting area of the two images divided by the number of road edge pixels in the map image and 2. the confidence factor, which is the fraction of the top view area falling onto areas of the map containing information. This is formally described by Equation 1, where p is a pixel location in the overlapping area of the two images. m and n are values of pixels in the road edge map image M and road edge image of the top view N respectively. Value 0 denotes no road edge, and value 1 denotes road edge. $N(x, y, \phi)$ is the road edge image of the top view translated by (x, y) and rotated by ϕ . m' and n' are the information masks of the map and road edge images where 0 denotes the presence of information (mask is off) and 1 denotes no information (mask is on). a and b are weighting constants which bias the relative importance of respectively the score and the confidence of the match. At the moment we are using

$$Q_1(x, y, \phi) = a \frac{\sum_{p \in N(x, y, \phi) \cap M} AND(m_p, n_p)}{\sum_{p \in N(x, y, \phi) \cap M} m_p} + b \frac{\sum_{p \in N(x, y, \phi) \cap M} NOR(m'_p, n'_p)}{\sum_{p \in N} n_p} \quad (\text{Equation 1})$$

$m, m', n, n' \in \{0, 1\}$

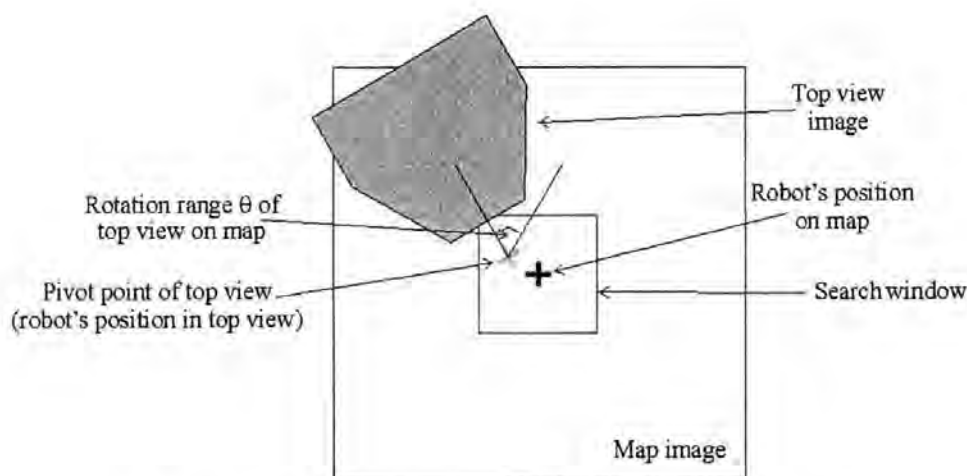


Figure 3: Illustration of one match position of the top view image on the map.

$a=1$ and $b=1$. The best matching position and orientation of the road edge image of the top view is the one where Q_1 is maximum. Equation 1 ensures that, for two configurations with equal score, the one with highest confidence has the best match quality.

To save computation time and limit the risk of matching the new top view at the wrong location, the search is limited to a small window defined on the map around the expected position of the robot. The range of rotation of the top view for each match position is limited to a small angle θ . The size of the search window and angle θ reflect the precision of the motion of the robot. To further improve speed, a crude search is performed initially using coarse steps of position and rotation of the top view on the map. The search is then refined for a more accurate determination of the position and orientation (match vector). An example of matching a new top view on the map after the robot's motion is illustrated in figure 2.

The matching process uses only the road edge image rather than the road surface image because edges are robust features of the image and allow a more precise matching. The resulting match vector is used to paste the road surface image of the top view on the map and to translate both versions of the map.

Road surface information is extracted from the top view image using chromaticity information. Chromaticity is an intensity-invariant two-dimensional vector describing colour. The two components of the vector are the ratios of red to blue and green to blue components of the RGB vector. A road surface likelihood image is constructed by assigning a value to each pixel location in the original image, which is proportional to the Euclidian distance between its chromaticity vector and a reference

chromaticity vector. The reference vector is obtained by calculating the average chromaticity of a sample of road area in the first image along the route. As with the road edge version of the map, a threshold is applied on the road surface likelihood image resulting in a binary image with either road-like or non road-like areas. Examples of the road surface images of the top view and map are shown in Columns B and D (respectively) of figure 6.

The road surface version of the map is used to locate local features of the road layout using templates. This is described in the following section.

4. Road-feature templates

Templates are binary images of local road surface features drawn at the same scale as the short-lived map. Some examples are shown in Figure 4.

For each road navigation task a specific subset of the available templates is selected with its associated action. For example, in the case of the task: "take the second turn right" (illustrated in the following section) templates a, c and g are selected, for following the road and for detecting the right turn. The selected templates are continuously matched against the road surface version of the map for each new scene captured and the robot navigation sequence associated with the "winner" template is executed. In this way template matching is interleaved with short motion sequences.

Like in map building, the matching process for each location and orientation (vector $[x, y, \phi]$) of the template on the map produces a match quality measure Q_2 . Here the score term of Q_2 is the sum of the matching road and non-road pixels in the two images divided by the number of template pixels

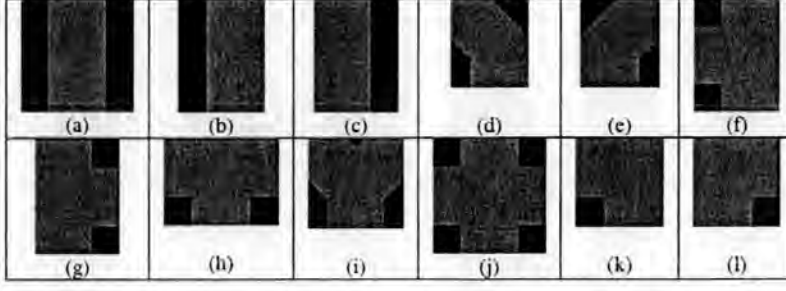


Figure 4: Examples of templates for: road following (a,b,c, d, e), for intersection detection (f,g,h, i, j, k, l). The light grey areas indicate road-like areas and the darker grey areas represent non-road areas.

falling onto areas of the map where information is available and the confidence factor, like the map building, is the fraction of the template area falling onto areas of the map with information. This is formally described by Equation 2, where p is a pixel location in the overlapping area of the two images. m and t are values of pixels in the road surface map image M and template image T respectively. Value 0 denotes no road, and value 1 denotes road. $T(x,y,\phi)$ is the template image translated by (x,y) and rotated by ϕ . m' and t' are the information masks of the map and template images where 0 denotes the presence of information (mask is off) and 1 denotes no information (mask is on). a and b are weighting constants which bias the relative importance of respectively the score and the confidence of the match. We are using $a=1$ and $b=1$. The best matching position and orientation of the template is the one where Q_2 is maximum. Equation 2 (like equation 1) ensures that, for two configurations with equal score,

the one with highest confidence is the winner.

Each template is associated with a pivot point. This is denoted by a dot-centred circle in figure 5 for some of the templates.

Translation and rotation of the template during the matching process is done with reference to this point. The pivot point of the winner template becomes a waypoint for the robot. Each template has a set of associated "search centres" (denoted by the numbered crosses in figure 5) which will define the template search window in the next image captured. The search window is defined around one of these centres (projected onto the map after the match) depending on the action associated with the winner template. If for example the action is to turn left at the crossroad, search centre 1 of the crossroad template (figure 5d) will be used to define the search window of the next template. As with map building, the search for the best match vector is initially coarse and then refined (section 3).

$$Q_2(x, y, \phi) = a \frac{\sum_{p \in T(x,y,\phi) \cap M} \text{XOR}(m_p, t_p)}{\sum_{p \in T(x,y,\phi) \cap M} \text{NOR}(m'_p, t'_p)} + b \frac{\sum_{p \in T(x,y,\phi) \cap M} \text{NOR}(m'_p, t'_p)}{\sum_{p \in T} t_p} \quad (\text{Equation 2})$$

$m, m', t, t' \in \{0,1\}$

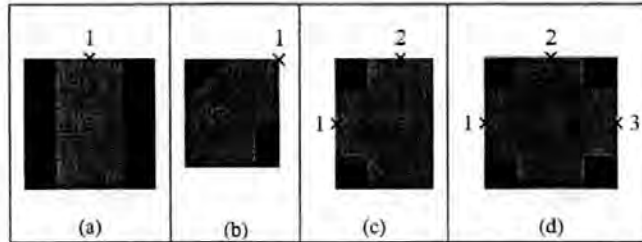


Figure 5: Pivot point and search centres for the crossroad template.

Define set of road features to look for. Loop: { Capture and process road image. Update internal map & localize robot. Find best matching template in the map. Execute procedure (e.g. robot motion) associated with the winning template. } }
--

Table 3: Pseudo-code for case “take the n^{th} turn left/right”. The resulting sequence of displacements is illustrated in section 3.5 for n (ordinal_1) = 2 and direction_1=left.

5. Example: “take the n^{th} turn left/right”.

In this section the instance of the “turn” primitive is described when the ordinal and direction parameters are passed to it (second case in table 2). Table 3 shows the pseudo-code of the sub-routine of the primitive that is called in this case.

In the first step of the pseudo-code in table 3, the templates selected for this case represent straight or curved road segments and intersections of various angles. In the loop, the selected templates are matched on the road surface map. The template with the best match in each iteration of the loop will determine the action to be performed next. For example, the templates for straight or curved road will cause the robot to move further along the road. The intersection templates can have one of two actions associated with them: 1. either to cause the robot to move to the centre of the intersection and rotate in the direction of the turn or 2. just move ahead along the road (head in the centre of the intersection but without rotation at the end). The first action is associated with the intersection templates when approaching the n^{th} intersection. In this case the robot makes the turn and the loop is exited so that execution is passed to the primitive associated with the next chunk in the route instruction. The second action is associated with the intersection templates until (but excluding) the n^{th} intersection. In these cases the robot carries on following the road.

In this procedure, the robot must keep track, not only of the number of intersections passed but also of their locations. When an intersection is identified, its location is compared against a record of previously found intersections and if a relatively close match is not found, it is considered to be a new intersection.

Intersection locations are recorded in the egocentric reference frame of the robot. Each time the robot moves these are updated to reflect their new position relative to the robot. To perform this updating, the robot must know by how much it has

moved since the last image was taken. In our purely vision-based system, this is done by tracking the displacements of landmarks in the image, using the short-lived feature map described in section 3.

Figure 6 shows the successive states of the short-lived maps and images processing results as the robot executes the road navigation task: “take the second turning to the right”. The path followed by the robot is marked in figure 1a, where each arrowhead indicates the points where the robot finishes an action and captures a new image to determine the next action. In step 1 there is no information on the road edge map and so the road edge and road surface top views are simply pasted (in the egocentric reference frame) on the road edge and road surface maps respectively. In successive steps, this initial map is progressively shifted backwards and eventually rotated. Column D of figure 6 shows the best matching template in each step. Step 5 shows the resulting map after the rotation of the robot at the second right turn.

6. Concluding comments

Two aspects of natural language instructions influence the method proposed here for navigation in our urban model environment: Their division into action chunks and their under-specified nature.

Each chunk can be considered as a search-and-act loop which exits when a terminating condition is met. Primitives were written to reflect this. Primitives are like Lego bricks that the user can combine in any sequence through his verbal instructions. It is therefore important that the terminating condition of each primitive is an initial condition that other primitives can handle. This issue is investigated further in this project.

In natural language, task specification is very abstract. For example in: “take the second turn right”, the absolute locations of the intersections, their orientations or shapes are not given. These pieces of information must be retrieved in-situ by the robot to successfully complete the task. This is achieved here by the use of local road-feature templates that enable

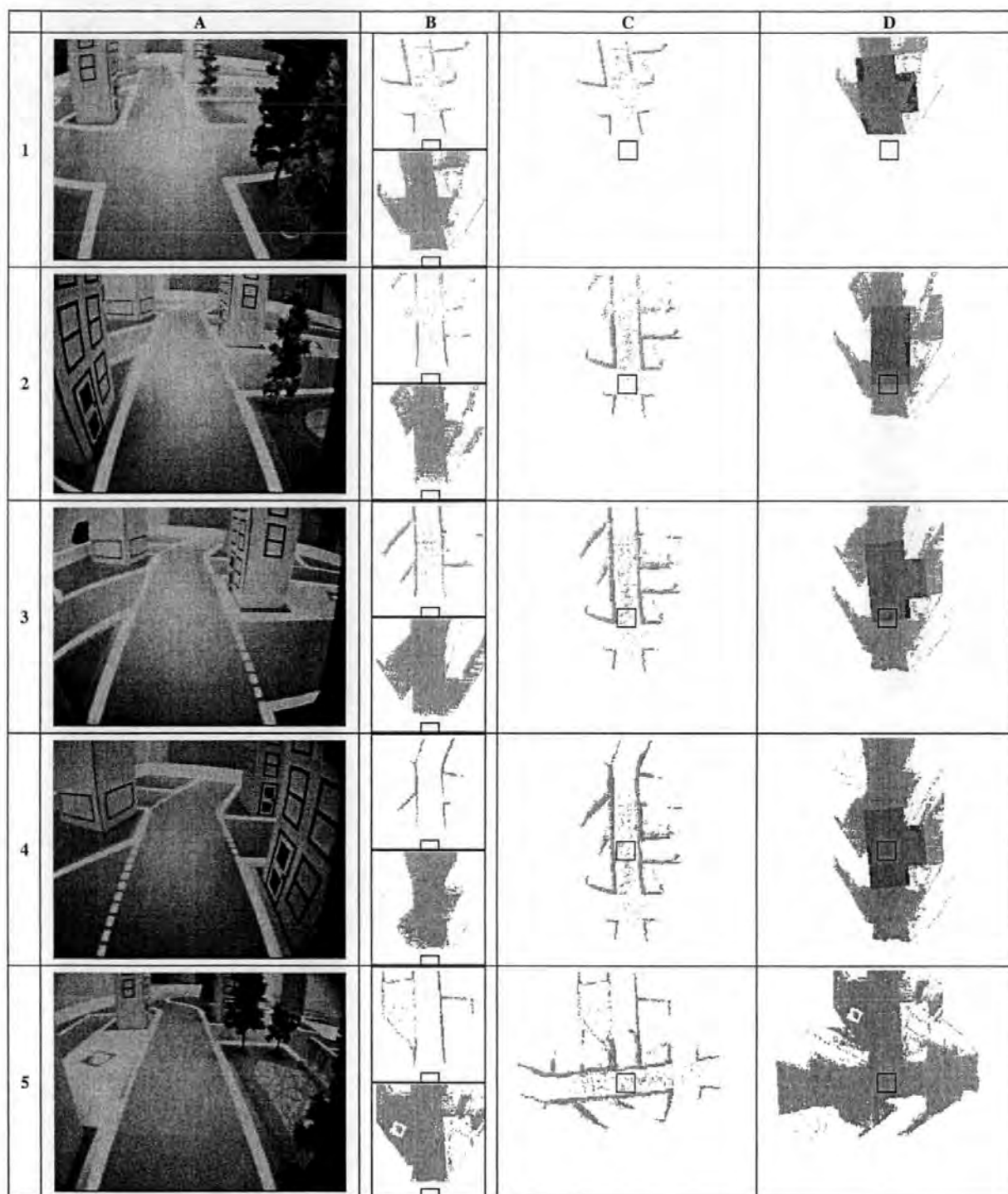


Figure 6: Step-by-step illustration of the execution of "take the second turning to the right". The execution is completed in five steps with the corresponding images at each step shown in the rows of the figure. Column A shows the camera view, column B shows the road edge and road surface images of the top view. In column C the road edge map is displayed and in column D the road surface map is shown. The best match position and orientation of the winning template for the step is also shown superimposed on the road surface map. Note also the indication of the position of the robot (black outline) in all the top view and map images.

to recover orientation and shape information. Robustness is achieved on the one hand by defining very general template shapes and on the other hand by limiting visual search to those salient features selected by the instructor.

To localize road features, the use of road surface information is deemed more robust than edge information. A template has a good chance to match correctly even if road areas are partially missing, e.g. due to occlusion. For instance, in figure 6 (column D, row 1) the "right turn" template matches at the correct location although the trees prevent full recovery of the road in the filtered image.

Most of the methods suggested in the past to recover the road layout from road images deal with the case of a straight or curved road extending in front of a vehicle, but without any turns, intersections, splits, roundabouts etc. (Waxman et al., 1987), (DeMenthon and Davis, 1990), (Kaske et al., 1997), (Sayd et al., 1998), (Wilson et al., 1999) and (Wang et al., 2000). These methods require that both sides of the road are visible (though not necessarily continuous) in the image to be able to recover the road. These methods are effective in cases where a vehicle needs to stay in the middle of a road lane when following a highway for example, but they are unsuitable in more complex urban environment.

Methods to recognize intersections on the road were proposed by (Jochem et al., 1996) and (Crisman and Thorpe, 1993). In (Jochem et al., 1996) a previously trained neural network is used to distinguish the road. The method lacks precision because of the neural network approach used and therefore fails to accurately determine the location and orientation of the road. Furthermore, the method suggested for modelling a road intersection required the knowledge of either the position of the intersection, to determine its precise layout, or the layout, to find its position. Some a-priori information on the intersection is also available in our case, through the natural language instruction.

In (Crisman and Thorpe, 1993) dynamic model building and matching are applied on a road surface likelihood image to determine the layout of the road. This method effectively finds intersections spurring from a straight road but would fail to find an intersection on a curve or an exit from a roundabout for example. Furthermore, the suggested method attempts to reconstruct the whole intersection. A strength of our method is that only the necessary road features (for the completion of the task in hand) are sought in the map, thus saving computational time and improving on system robustness.

Although the template matching method is applied here in a simplified model of the world, the same navigation principle could be applied for instructable vehicles navigating in the real world. The simplicity of the robot's environment in this project helps to set aside the problems of extracting road/non-road information from complex real-world scenes and focuses on determining the road layout once this information is found. Other information, absent in our model but present in the real world could give clues which could contribute towards a statistical measure suggesting the road layout ahead of the instructable vehicle. Such clues could be for example the direction of motion of other road vehicles, the alignment of buildings, road signs etc.

Other landmarks (buildings, trees, lake, bridge etc.) of our model town are mentioned in the corpus and will need to be located to enable following the instructions. Ongoing work is addressing the problem of landmark recognition, the resolution of spatial relations between landmarks, and those between landmarks and the robot as sometimes mentioned in the corpus. The solutions to these problems are not expected to modify the navigation methodology described here but will rather merge with it.

Finally, an interesting property of such a system is that it has all the perceptual components required to robustly learn a route from experience (e.g. by following a human guide) in terms of reportable action chunks rather than in terms of odometric measurements.

Acknowledgements: This work is supported by EPSRC grant GR/M90023.

References

- Lauria S., Bugmann G., Kyriacou T., Bos J., Klein E., Personal Robot Training via Natural-Language Instructions, *IEEE Intelligent Systems*, 16:3, 2001, pp. 38-45.
- Broggi, A., Robust Real-Time Lane and Road Detection in Critical Shadow Conditions, *Proceedings of the IEEE International Symposium on Computer Vision*, Coral Gables, Florida, November 1995, pages 353-358.
- Waxman, A. M., Lemoigne, J. J., Davis, L. S., Srinivasan, B., Kushner, T. R., Liang, E., Siddalingaiah, T., A Visual Navigation System for Autonomous Land Vehicles, *IEEE Journal of Robotics and Automation*, Volume 3, Issue 2, April 1987, pp. 124-141.
- DeMenthon, D., Davis, L. S., Reconstruction of a

- Road by Local Image Matches and Global 3D Optimization, Proceedings of the IEEE International Conference on Robotics and Automation, 1990, pp. 1337-1342.
- Kaske, A., Wolf, D., Husson, R., Lane Boundary Detection Using Statistical Criteria, Proceedings of International Conference on Quality by Artificial Vision, QCAV'97, Le Creusot, France, 1997, pp. 28-30.
- Sayd, P., Chapuis, R., Aufrere, R., Chausse, F., A Dynamic Vision Algorithm to Recover the 3D Shape of a Non-Structured Road, Proceedings of the 1998 IEEE International Conference on Intelligent Vehicles, 1998, pp. 80-86.
- Wilson, M. B., Dickson, S., Poppet: A Robust Boundary Detection and Tracking Algorithm, BMVC99 (British Machine Vision Conference 1999), pp. 352-361.
- Wang, Y., Shen, D., Teoh, E. K., Lane Detection Using Spline Model, Pattern Recognition Letters, 21(8), July 2000, pp. 677-689.
- Jochem, T. M., Pomerleau, D. A., Thorpe, C. E., Vision Based Intersection Navigation, Proceedings of the 1996 IEEE Symposium on Intelligent Vehicles, September 1996, pp. 391-396.
- Crisman, J.D., Thorpe, C.E., SCARF: A Color Vision System that Tracks Roads and Intersections, IEEE Transactions on Robotics and Automation, Volume 1, Issue 1, February 1993, pp. 49-58.

Copyright © by Theocharis Kyriacou 2004

This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without the author's prior consent.