International Journal of Information Acquisition
Vol. 5, No. 4 (2008) 1–17

World Scientific
www.worldscientific.com

IMPLEMENTATION OF A LOCALIZATION-ORIENTED HRI FOR WALKING ROBOTS IN THE ROBOCUP ENVIRONMENT

RENATO SAMPERIO
Logistics and Production Robotics
German Research Centre for Artificial Intelligence
Robert-Hooke-Str. 5, 28359 Bremen, Germany
renato.samperio@dfki.de

HUOSHENG HU* and DONGBING GU†
Department of Computing and Electronic Systems
University of Essex, Wivenhoe Park
Colchester CO4 3SQ, United Kingdom
**hhu@essex.ac.uk*
†*dgu@essex.ac.uk*

Received
Accepted

This paper presents the design and implementation of a human–robot interface capable of evaluating robot localization performance and maintaining full control of robot behaviors in the RoboCup domain. The system consists of legged robots, behavior modules, an overhead visual tracking system, and a graphic user interface. A human–robot communication framework is designed for executing cooperative and competitive processing tasks between users and robots by using object oriented and modularised software architecture, operability, and functionality. Some experimental results are presented to show the performance of the proposed system based on simulated and real-time information.

Keywords: Human–robot interface; GUI; walking robots.

1. Introduction

To design controllers for autonomous robots, software development platforms play an important role, especially before a prototype system is available. In general, a good software platform can provide simulated functions that speed up the development of different algorithms, including complex programming and huge data collection. There are mainly two types of software development platforms for robotics research. One is a simulated platform in which modeled robotic systems are developed for method testing. In many cases, a simulation is also based on assumptions where models can work as a base for a further robot implementation.

The second type of software development is related to the hybrid control, which partially relies on the real robotic system. The parameters of these systems are collected from robots sensors in an applied use case. The proposed working development achieves that if the algorithm works well under such a hybrid platform, it can also work well on real robots.

There are some human–robot interfaces (HRIs) that have been already developed for proposed robot control [Martin *et al.*, 2004], behavior design [Hugel *et al.*, 2005], vision-based applications [Lovell, 2004], and multi-activity platforms [Golubovic *et al.*, 2004]. These systems manage robot behaviors within independent functionality, and offer an instructional execution between users and robots [Liu and Hu, 2006]. In such cases, the robots reaction to environmental stimulus are designed by user criteria [Sangpetch, 2005]. Moreover, the robot can support an auto-designing behavior which assimilates user instructions referred to an initial trajectory [Ogata and Takahashi, 1994]. This type of control platform adapts to processing resources and communication schemes for real-time or remote robot control.

Our research work is based on Sony AIBO walking robots which presents some difficulties in the development of control software and sensing algorithms for these robots. Firstly, programming AIBO robots takes a long time and may have a risk to damage them. It would be much safer for such development cycle done on a hybrid experimental platform in which most of the work is done on a PC efficiently.

Secondly, the processor on the Sony AIBO robot is not powerful enough. Complex algorithms cannot be implemented on real robots in real-time. So it becomes necessary to exchange information with the robot during the development phase with an efficient transfer rate.

Thirdly, the current interface between the Sony robot and human operators is not friendly. Each time when the experiment is completed, all the results need to be downloaded from the memory stick manually, which is very time consuming for the user. With the proposed hybrid platform, the progress of experiments is displayed on the screen directly and any problem with the experiment can be seen immediately, i.e. a convenient way to develop control and vision algorithms [Samperio and Hu, 2006; Samperio and Hu, 2008].

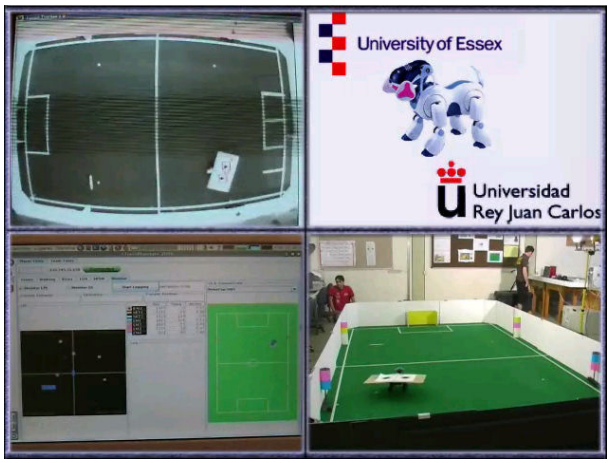

Fig. 1. HRI during a "Combined behavior" experiment with an overhead low quality tracker (top-left), GUI (bottom-left) and observed robot environment (bottom-right).

The rest of the paper is organized as follows. Section 2 describes the system design and a client–server scheme. Section 3 outlines the robot architecture from an embedding software perspective.

Simulated and tracking interfaces for localization support is detailed in Sec. 4. Experimental results are given in Sec. 5 to show the feasibility and performance of the proposed system. Finally, a brief conclusion and future work are described in Sec. 6.

2. The Proposal System

2.1. *System configuration*

The configuration of the proposed hybrid HRI is shown in Fig. 2. The user-robot interface manages robot localization information and user commands from a graphic user interface (GUI). The overhead VICON tracking system is used for evaluating robot positions.

The robot localization is receiving information from visual perception, motion, and behavior modules which continuously sends robot positioning information. In this case, the localization process is executed independently

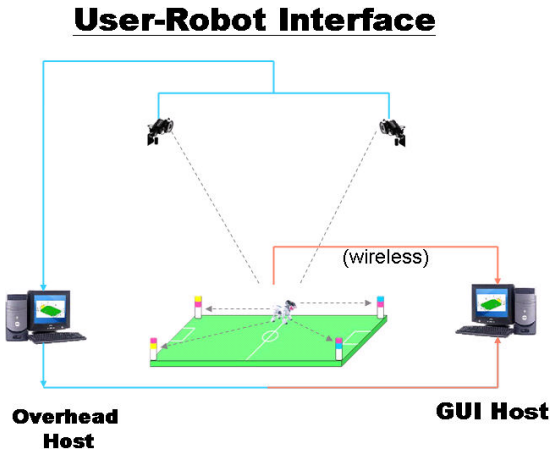

Fig. 2. The configuration of the proposed HRI system.

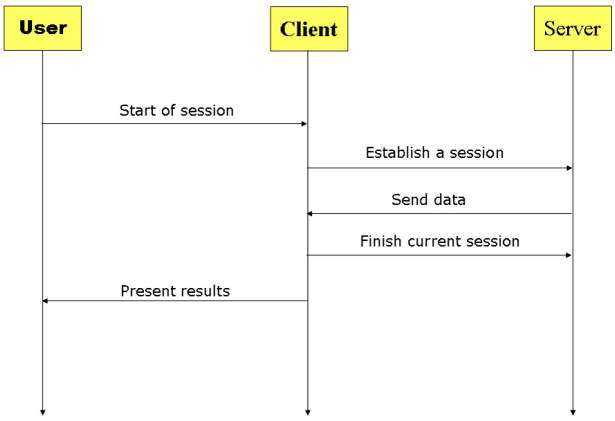

Fig. 3. Client–server task communication process.

from robot behavior and it shares computing resources. Therefore, users can control experiment execution by using a GUI tool that is composed by tasks for robot applications.

Asynchronously, the VICON overhead visual system for tracking robot pose and position is also part of our user-robot interface. The tracking results are used as a ground truth to evaluate robot self-localization results, which are sent to a GUI host PC for the purpose of further analysis.

This section describes a client–server scheme which works as a request-and-response service for human–robot communication. The design is implemented using a message exchange sequence where the server side is an information provider and client side is a service solicitor.

A wireless OpenR IPv4 stack is used for a TCP/IP robot communication process which is conformed by server service, GUI client interface, and a message. The service is used as a stand alone communication process among robot, overhead tracking, and user interaction. Proposed system also has an ignored time delay in transmission based on petition and response transactions. Therefore, the use case related to communication process is described in Fig. 3.

2.2. *Client–server module*

Initially, the client module starts a service session through a server petition. As soon as the client establishes a connection with the server, it states communication characteristics by providing a time stamp, transmission rate and message type, and priority. The transmission can continue until the client or server generates a halt in the service by way of a user command or an unexpected interruption. After all, the communication is finished once the information has been fully transmitted. The information process is contained in a message-object that is independent from any operative system and it is transmitted for communicating the GUI client and server interface. Moreover, it could be adapted to tolerate further modules for integrating any other functional environments.

2.3. *GUI — client*

On the client side is a GUI implementation which receives, sends, and presents information from services on-demand. It also provides robot operational commands for movement control, image processing, landmark, and localization tracking and behavior development. It has been implemented using a collection of Java API packages which operate according to user requirements.

As illustrated in Fig. 4, a GUI is implemented for evaluating robot localization and for monitoring support tasks which integrate an active visual robot with localization performance. These user-oriented tasks are asynchronously executed at any time by using command line messages.

The GUI is implemented into a thin client using a Java front view which can be used

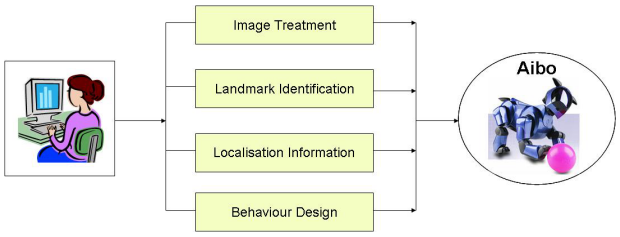

Fig. 4. User tasks design for localization process.

in any operating system and reusing code. Moreover, native Java packages are used for multi-threading, image treatment operations, distributed processing, and communication interfaces and contained into a client module. The implemented Java modules are as follows.

Image treatment: This module uses Java Advanced Imaging package for image manageability in a flexible, extensible and distributed image processing environment.

Networking: Java Net package is used for all messages sent in a wireless transmission form between robot commands, behavior design, and any other sensing device.

User interface: This is created with Java Swing package using a "look and feel" window screen navigation for visual calibration, behavior design, robot teleoperation, and positioning.

This client also coordinates experimental execution and integrates synthetic or real-time data processing for robot positioning. Moreover, modules execution is realized asynchronously by multi-task assigned services which allows to alternate information analysis. The characteristics of the client include:

(1) *Adaptability*: The client can be adapted for additional elements.
(2) *Multi-platform*: The client implements Java front view (JSwing).
(3) *Distributed processing*: The client uses multi-threading for sharing processing resources.
(4) *Code reusing*: The client implements interfaces with the same functionality, as telnet sessions for robot console communication.

2.4. *Server*

The server task is to alternate robot and overhead positioning information by using asynchronous ports for each service. On the one hand, the tracking service is used as a dedicated channel for sending robot positions or landmarks tracking from an overhead perspective. On the other hand, the robot position is sent by a TCP/IP implementation which follows an Apertos object-oriented schema.

On the robot side, the server acts as a robot service with reserved memory space for operations and variables, as described in [Martin *et al.*, 2004]. In such cases, the server provides networking routines for dealing with transmission delay and loss of information in an TCP/IP socket.

More specifically, the object in charge of any TCP/IP connectivity is an ANT entity which belongs to OpenR Network Toolkit. The server generates a dedicated communication channel for robot control and it formats information for client interactivity. In summary, the server provides information for robot movement control, image processing, observed features definition and, behavior control, as shown in Fig. 5.

The working platforms of the robot and tracking system are implemented in C^{++} for the TCP/IP communication routines. It also makes use of a object-oriented architecture for its functionality. The most representative classes for server implementation are related to a message generation and transmission.

In the robot's case, competition between the processes will cause a latency, which interferes directly with the robot's performance of tasks.

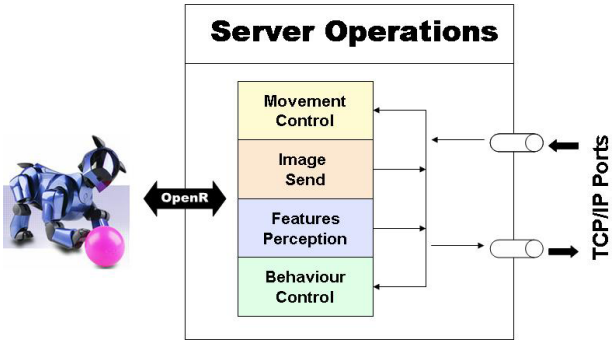

Fig. 5. Server operations.

Similarly, for the overhead tracking system service, the local control service sends information to the central GUI interface by transmitting an object implementation to a different port for each service. Both cases cause a depreciable delay of information.

2.5. *OpenR System*

OpenR is a Sony developed operating system that is based on an Apertos process structure. As mentioned in Yokote [2004], from a software design perspective and for robot programming are offered several advantages such as

Portability: It offers modularised and interchangeable dependencies working directly on parts, devices or processes.

Code sharing: It creates a more efficient reuse of code with standardized schemes of development.

Separation of policy from mechanism: It offers a dynamic changing of policy management as in the process scheduler.

Optimization: It allows an effective and constant redefining or optimizing algorithm evaluation.

Embedded trading: For an inefficient but portable class, elements can later be replaced with a more efficient, less portable, machine specific class.

Component testing: It presents case tests for evaluating and debugging multiple objects.

Adaptable interfaces: It can add new components by modifying object bindings from a name server.

Mutual exclusion and synchronization: It allows control of object mutual exclusion through a method definition and by providing protection to variable access.

OpenR uses an object message communication scheme for multi-task processing on a single-processor platform. The communication process initially requires an object definition for generating a synchronized functionality between its components and communications. Inside an OpenR object, many other objects interact and share processing resources by exchanging message content.

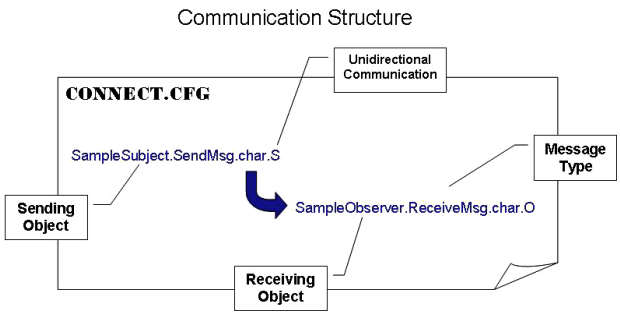

Fig. 6. Content specifications for Connect.cfg file.

For an inter-object communication, processed data is written in a shared memory section and is sent as an event notification. The communication process is divided into three stages: (a) Message construction and notification of sending, (b) Generation of shared memory and (c) Message reception. Each message object contains a *Subject* for handling the sending part and an *Observer* for encapsulating the receiving part.

The message is defined in a stub file called "connect.cfg". As shown in Fig. 6, this file syntax contains a subject and observer name, message type, and messaging structure. This communication process uses a unique character reference to identify unidirectional messages. Also, message objects are processed one message at a time. Extra messages generated are queued in a message stack awaiting the next processing notification.

The communication process is also described in Algorithm 1.

3. Robot Architecture

3.1. *Layered design*

A layered architecture is proposed for structuring information treatment, which enables easy design of independent and autonomous tasks with shared processing resources. A robot localization is adapted to a concurrent task list for further results analysis.

Each layer controls different robot skills from sensed information, an operative system structure, object containers for sensed traces, and robot behavior objects. From bottom up, layer capabilities are getting close to robot control

Algorithm 1. Communication process between subject and observer.

(Subject)	*(Observer)*
Require: Message $\{m\}$	
{*Initialise message to send.*}	
1: MessageReady($\{m\}$)	
	{*Inform message is ready.*}
2:	NotifyObservers($\{m\}$)
	{*Notify of data which can be used.*}
3:	$\{m\} \leftarrow$ NotifyData()
	{*Finish processing data and get ready for next message.*}
4:	$\{m\} \leftarrow$ AssertReady($\{m\}$)
{*Process another message.*}	
5: $\{m\} \leftarrow$ CreateMessage()	

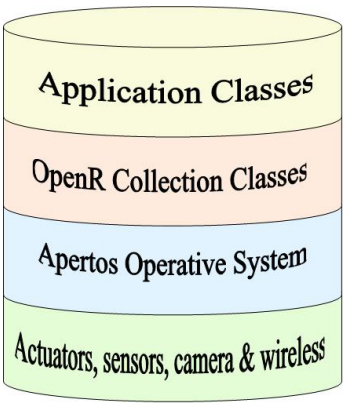

Fig. 7. Robot software layer design.

and from top down, layers create user applications. The four layered programming scheme is synchronized by a modularised architecture and a priority-based processing. The layers are illustrated in Fig. 7 and described as follows:

Actuators, sensors, and input/output: This first layer is in charge of controlling hardware devices. Aibo robot has a MIPS R7000 64bit RISC processor 576 MHz of clock speed with 512Mbit of SDRAM and 32 Mbit of flash memory. It handles 18 degrees of freedom for mouth, head, legs, ears and, tail. Also, it contains sensors for monitoring temperature, infrared distance, acceleration, and pressure on its chin and paws. All input/output devices include: a face, ear, head, and back LED; CMOS image camera of 350000 pixels; wireless IEEE 802.11b; memory stick slot; and miniature microphones and speaker.

Apertos operative system: The Apertos layer includes a distributed object-oriented operative system which uses a reflective architecture which assigns a specific use for each inherited object, and it is associated with a set of meta-objects with object semantics. A reflector offers meta-operations provided by meta-objects constituting a meta-space and it is defined by a class hierarchy or meta-hierarchy. This sort of reflective programming reuses existing reflectors whilst a meta-hierarchy checks object compatibility. Objects can also be changed by meta-space composition and their semantics modified. Hence, new object behavior can replace old ones using components of the operative system.

OpenR collection classes: This layer contacts the Apertos Operative System using the programming operations with *OVirtualRobotComm*, *OVirtualRobotAudio*, and *ANT* TCP/IP objects. The *OVirtualRobotComm* object provides a service for the *Effectors*, *Sensor*, and *OFbkImageSensor* objects. The *Effectors* object is in charge of gain and LED control via the *OCommandVectorData* structure. The *Sensor* object acquires information from sensor and joint values using an *OSensorFrameVectorData* structure. The *OFbkImageSensor* object manages camera input with an *OFbkImageSensor* structure and it can modify image quality and color segmentation. Finally, the *ANT* object is in charge of the endpoint buffers for input and output data.

Application classes: This layer contains programs for robot behavior and a localization

module. It is composed of three main modules: *AiboEssexControl*, *AiboEssexObserver*, and *AiboEssexComm*. The *AiboEssexControl* object performs contact with robot actuators and input/output devices as well as their required algorithms of treatment. The *AiboEssexObserver* object translates any sensed input data into positioning, such as robot odometry, pose, and position. The *AiboEssexComm* object is in charge of TCP/IP communication protocol, as well as of information formatting for transmission. Robot walking styles and head tracking; image analysis and feature detection; sound generation; and behavioral control; are all included as subtasks of the *AiboEssexControl* object.

This architecture integrates environmental information from robot sensors for obtaining localization results during a robot behavior execution. Moreover, these modules adapt reusable code for odometry measurements, image treatment, walking styles, playing behaviors, and networking communication.

3.2. *Localization module*

The localization module transforms sensed data from image perception and robot odometry into a stand alone localization process. On the one hand, landmarks are detected from image analysis to offer distance and angle references for robot positioning. On the other hand, robot odometry is used for approximating displacement by measuring the speed of the leg. The complete procedure is illustrated in Fig. 8 and the process is described in Algorithm 2.

The algorithm is:

(1) Initially, localization landmarks and robot odometry are generated independently from each other and used, respectively, by the *receiveOdometry* and *receiveLandmarks* methods. The *receiveOdometry* method creates suitable odometry values for the localization method which are encapsulated in *Odometry* object. The *receiveLandmarks* method organizes observed candidate landmarks into an *LandmarksSeen* object.
(2) Afterwards, the Robot generates a position using a localization method from the *Localize* object and follows an update–observe–predict Bayesian structure. From this stage, a positioning vector is represented by x, y, θ, γ which corresponds to x and y robot space coordinates, orientation angle, and a level of confidence, respectively.
(3) Then, information is formatted and transmitted for further local reuse in a method

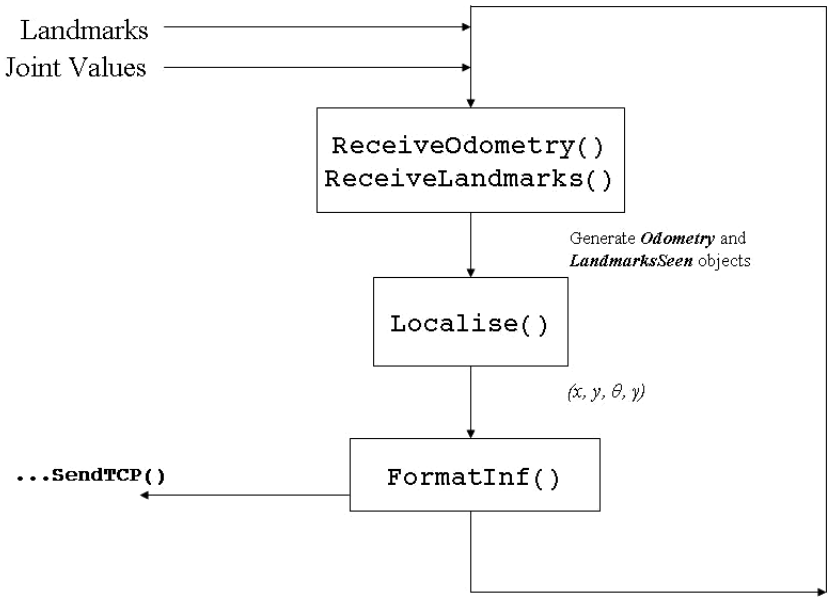

Fig. 8. Localization module.

Algorithm 2. Robot Localization algorithm.

Require: Initialise *Odometry* odometry
Require: Initialise *Landmark Map* landmarks
{*Predict and Update are asynchronous because images are obtained faster than odometry*}
1: **for all** *LocalisationCycle* i **do**
2: **if** odometry$_i$ $\exists$ **then** {*Predict phase generates a new state from robot odometry*}
3: $odometry_i$ = receiveOdometry(i)
4: $< x, y, \theta, \gamma > \leftarrow$ *LocalisationModule.* update($odometry_i$)
5: **end if**
6: **if** landmarks$_i$ $\exists$ **then** {*Update phase use landmark information for final state*}
7: $landmarks_i$ = receiveLandmarks(i)
8: $< x, y, \theta, \gamma > \leftarrow$ *LocalisationModule.* predict($landmarks_i$)
9: **end if**
10: formatInformation($< x, y, \theta, \gamma >$) {*Information is prepared for transmission*}
11: checkPositioning($< x, y, \theta, \gamma >$) {*State further control is applied*}
12: **end for**

called *FormatInf*. Also during this stage, information can be sent to the client TCP/IP service in *sendTCP()* if a petition is received.

(4) Lastly, state is weighted and validated according to localization method's policy during a *checkPositioning* method.

The localization module treats and exchanges any localization method which follows an update–observe–predict scheme of sensing integration. The output of this module is a positioning vector which can be used by simulated or robot localization objects.

3.3. *Behavioral control module*

A behavior module is implemented for relating robot execution to its environment. A behavior is a sequence of states which guides robot activities using the perceived environment as a stimulus for a robot activity [Hugel *et al.*, 2005]. The behavior module has a state machine as part of a list for assigned actions in which each state executes a robot task or set of tasks. The state machine is linked and evaluated by tasks required for sensing information in order to decide on a new state selection from assigned roles.

A behavior requires transitional state rules for executed states such as "*Initial*," "*Ready*," "*Play*," and "*Finish*" steps. The "*Initial*" step initializes robot objects and devices; "*Ready*" sets up robot devices and uploads object behavior configuration; "*Play*" step executes the current state; and the "*Finish*" step stops any service, destroys memory objects, and power off robot devices if necessary. Behavior execution is iterated until an explicit end behavior is reached or it receives an external stop command. The behavior module execution is illustrated in Fig. 9 and is described as follows:

(1) Firstly, any robot devices required for each behavior are initialized into an object configuration in the *InitBehavior()* method. This method includes an initial scanning of the internal and external robot status. A behavior can also execute states or meta-behavior states in which additional behavior states are included in this step.

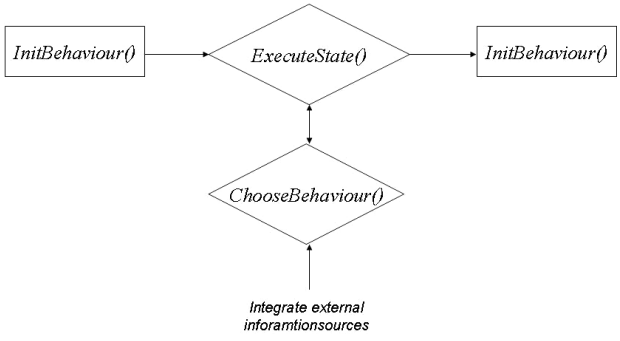

Fig. 9. Behavior control module sequential procedure.

(2) A *chooseState()* method evaluates the next behavior selection depending on transition conditions and the previous behavior state. Additionally, it is assigned a confidence value for evaluating the quality of transition state. This evaluation is based on combining input/output information that state consistency with the state to be executed.
(3) An *executeBehavior()* method realizes robot tasks which are executed as independent processes and it also shares robot computational resources for completing a state behavior.
(4) The *checkLocalization()* method obtains information from robot localization in order to be executed asynchronously and simultaneously to robot behavior.

The behavior state can be finished once the implied devices are released and the robot is put into a "*Stand By*" mode awaiting any action to be performed. Also, transition states are evaluated combining the accountability of behavior performance, speed of execution, and accomplishment of rules for state transition. Therefore, a transition with multiple choices is more likely to recall successful states than ones with a lower evaluation value. So, a useful state transition can be traced whenever behavior states are not being realized as expected.

4. Environmental interface

4.1. *Localization Simulator*

Even, method accuracy and effectiveness are evaluated for robot conditions by comparing localization values with a high precision tracking system. Initially, the simulator diagnoses method accuracy and effectiveness in a testing environment where localization algorithms are developed and evaluated with and without noise presence.

The first stage of localization analysis is based on a simulation of a moving robot playing with a ball. The mobile robot is simulated as a 2D holonomic movement composed by the speed of two-wheeled movement as illustrated in Fig. 10 and Eq. (1). Then, the prediction for robot positioning is represented in a football

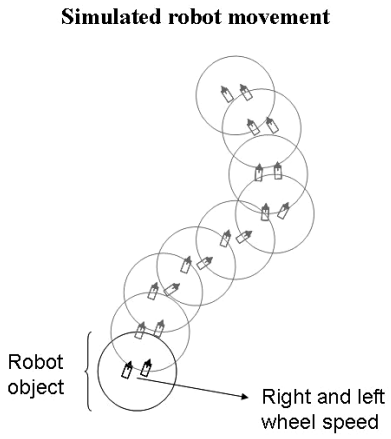

Fig. 10. Simulated holonomic movement.

pitch as the value where odometry velocities are obtained from the difference between the last and current position as in Eqs. (2) and (3).

$$V_r = Vx + Vy, \tag{1}$$

where V_r is the robot velocity, Vx the velocity in X, and Vy the velocity in Y. Each speed in X and Y orientation is, respectively, defined as

$$Vx = Vx_i + Vx_{i+1}, \tag{2}$$

$$Vy = Vy_i + Vy_{i+1}, \tag{3}$$

where the index i indicates current state and $i+1$ for next robot step as components for current robot mobility.

A 2D collision detection is adapted between robots, ball, and limiting walls, as is assigned to an initial speed and angle value. The initial speed is calculated from robot speed and a random kick speed value as illustrated by a ball kicking movement in Fig. 11 and Eq. (4).

$$\begin{pmatrix} \overrightarrow{V}_{\text{ball}} \\ \overrightarrow{\Theta}_{\text{ball}} \end{pmatrix} = \frac{\text{rand}(\overrightarrow{F_i}) \cdot t_i}{m} * \begin{pmatrix} \overrightarrow{V}_{\text{robot}} \\ \overrightarrow{\Theta}_{\text{robot}} \end{pmatrix}, \tag{4}$$

where $\overrightarrow{V}_{\text{robot}}$ is the speed which the robot is walking to the ball, $\overrightarrow{\Theta}_{\text{robot}}$ the angle related to the pitch which the robot has, $\overrightarrow{F_i}$ is the force that the robot kicks the ball only modified by other body (wall or robot) and reduced each time iteration t_i, m is the mass of the ball, $\overrightarrow{V}_{\text{ball}}$ is the resultant velocity, and $\overrightarrow{\Theta}_{\text{ball}}$ the resultant angle both of the ball.

The initial ball angle is used as the robot speed angle and is activated when the robot

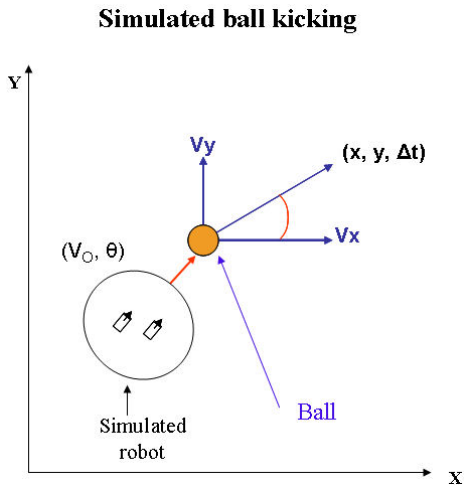

Fig. 11. Simulated ball kicking execution.

is close to the ball. The ball representation is divided according to time step approximations where each x and y ball coordinate depends on each speed component (V_x, V_y) as partial distances for every time step realized.

As in previously simulated applications [Liu and Hu, 2004], a three-layered object definition architecture is implemented for the localization process and user software interface. In this case, each layer has a specific purpose in architectural design, as shown in Fig. 12 and described below:

- *Robot object*: This layer encloses the physical modeling and a graphical representation of robots, ball, or landmark, as well as attributes such as dimensions, kinematics, and collisions.
- *Localization object*: This layer encapsulates the *Localization* algorithms with standardized input and output contents. The simulated odometry information is based on a two-wheeled robot model with variable noise speeds and simulated visible landmarks. Also, it offers a (x, y, and θ) set of values as the positioning interface.
- *GUI object*: This GUI has similar front-end characteristics to the localization monitor in the GUI architecture with options for controlling robot behavior, pace speed, and noise.

The simulator is used for testing localization algorithms and was built following a GUI architecture model. Even its capabilities do not hold

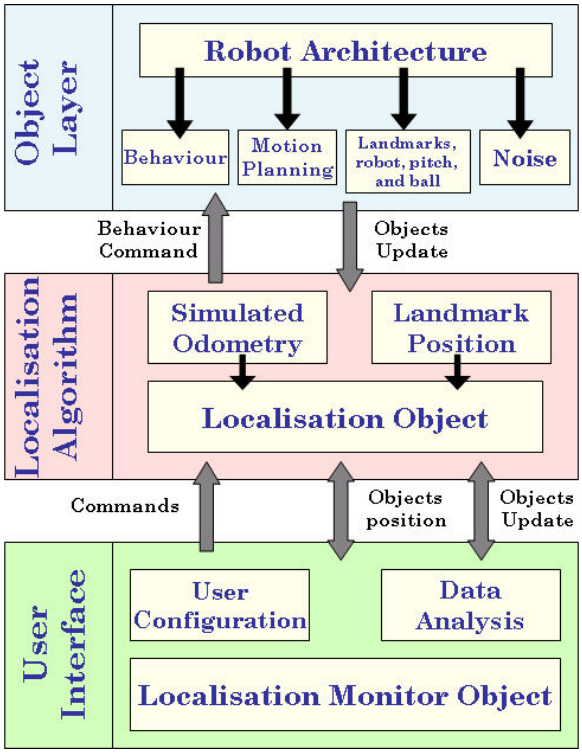

Fig. 12. Architecture of a task for a localization process.

visual and robot mobility, localization methodology, and algorithm implementation, it does follows an scalable robot architecture. Thus, the simulator is a developing platform for practical maintenance and to evaluate localization methods.

4.2. *Tracking system*

The overhead tracking system for recording robot positioning data is used in order to obtain a ground truth reference. The tracking system is a 3D optical VICON system which has less than 10 ms of latency, a positional accuracy of 0.1 mm, and an angular accuracy of 0.15°. It is composed of eight CMOS cameras that are positioned around robot space and at a height of 4.80 m. Also, it can track up to 50 and 150 markers that are made of light reflective material. Figure 13 shows the VICON tracking system used for obtaining the ground truth.

An object generated by the tracking interface is transmitted with robot heading, position, confidence of perception, time stamp, and if relevant a landmarks position. Robot tracking

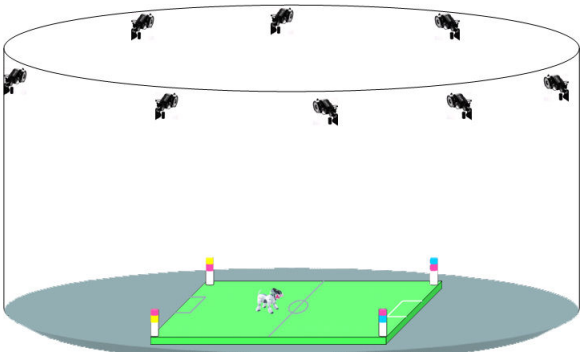

Fig. 13. The VICON tracking system for ground truth.

follows a five marker set placed in observable joints on the robot body. Independent robot poses and positions are obtained from this marker set as (x, y, θ, φ) which corresponds to robot 2D positions, heading, and confidence values. Then, similar values are obtained from a virtual robot mass centre as is described in Vicon and Woods [2004–2005] for tracking a mass skeleton.

4.3. *Complementary software*

The complementary software gives robot control and gives support for analysis of results in order to evaluate robot localization. This complementary software incorporates additional tasks as part of the proposed methodology, as shown in Fig. 14. The task make use of *color calibration* for image interpretation, *behavior designing* for robot environmental interaction, robot *movements construction* for adjusting robot movement online, and *localization results monitoring* for method evaluation.

The support software is described as follows:

(1) Part of this software create appropriate movements for kicks or walking styles and speeds in a specific interface.
(2) A behavior generation tool for a faster design and debug of robot control and its environment adaptability.
(3) Another section is a camera calibration used for color segmentation with a specific user interface. The camera calibration interface is used for generating appropriate color space values for environmental and camera conditions.
(4) Alternatively, an interface for localization monitoring informs and simulates positioning information. This information is combined with a overhead ground truth for achieving a more efficient method evaluation.

The modules provides a control tool for robot operability during experimental conditions. They also enable experiments with simulated or real-time data to be analyzed and replicated. The platform has been designed to work in Java for improving easy module adaptability from independently created modules.

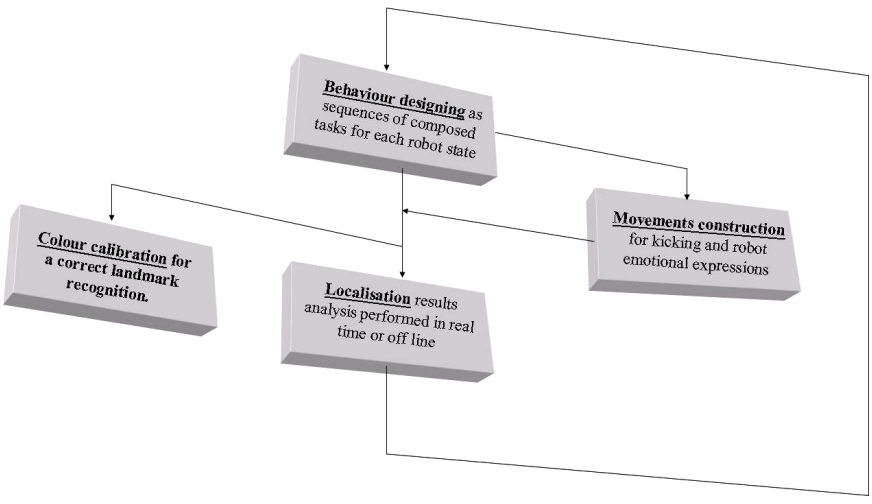

Fig. 14. Methodology of modeling a localization algorithm.

5. Experimental Results

In this section, Extended Kalman Filter (EKF), Fuzzy-Markov (FM), and Fuzzy-Markov-Kalman (FM-EKF) positioning methods are compared by using proposed HRI. For each experiment, we analyzed simulated and real-time experiments which evaluate localization performance compared with a ground truth generated by the overhead vision system. The experiments approach is described as follows:

(1) *Simple movements*: This stage is realized by a squared trajectory limited by landmark positions along football pitch. Then, the robot follows a trajectory in order to obtain a variety of visible points along the pitch.
(2) *Combined behaviors*: This experiment is composed by a playing session using a single robot which is constantly looking to score. Originally, it is evaluated by two dissimilar experiments: (a) robot reaching predefined points as accurately as possible and (b) a playing session for a robot to play a session alone and against an opponent.
(3) *Kidnapped robot*: This stage is realized in randomly sequences of kidnapping by time and pose. In each kidnap, the objective is to obtain information about where the robot is and how fast it can localize again.

The evaluation criteria for all experiments is defined by measuring parameters and relating obtained results with robot environment. Specifically, measured conditions for these experiments describe robot performance during execution as it is shown in Table 1 and described below:

- Robot mobility is the distance covered by robot for a complete experiment obtained from ground truth movement tracking. The total displacement from all experiments is of 186,190 mm.
- Time of execution is the time required for each experiment performance. The total time of execution for all the experiments is of 1170 s.

Table 1. Experimental conditions.

Experiment	Mobility	Time	Cycles
Simulated 1	12528.84	160.44	185
Real-time 1	78821.24	593.67	4878
Simulated 2	12499.48	68.90	268
Real-time 2	5770.73	38.68	372
Simulated 3	14514.38	38.06	125
Real-time 3	62055.79	270.36	2566

- Localization cycles is the complete execution of a correct and update step into localization module. The total amount for these experiments is of 8394 cycles.

The following experimental results were obtained from the comparison of localization methods realized in Samperio and Hu [2008] and presented in previous research work in Samperio and Hu* [2008]. Both real-time and simulated results are presented for showing the feasibility of methods design and execution whenever they were in use of the proposed HRI.

Figures 15 and 16 show the results from "Simple Movements" in which trajectories in simulated environment predict a noiseless representation of the real-time execution. For both experiments, the EKF method is considered the weakest of all the localization methods. Moreover, in the "Combined Behavior" experiment, the robot trajectories presents periods of low quality in localization as a consequence of additional behavioral head movements, as it is shown in Fig. 18.

In contrast, the simulated trajectory of Fig. 17 is conformed by a constant acquisition of environmental information. Lastly, the "Kidnapped robot" experiment was realized replicating sensed information from a real-time data into simulation with a modeled motion model. The results for both experiments are shown in Figs. 19 and 20.

The results are shown in trajectory maps obtained from sequences of localization cycles. Afterwards, each localization method can be evaluated with an independent ground truth positioning service asynchronously attached to the HRI. Therefore, proposed HRI architecture is capable of presenting visual information from

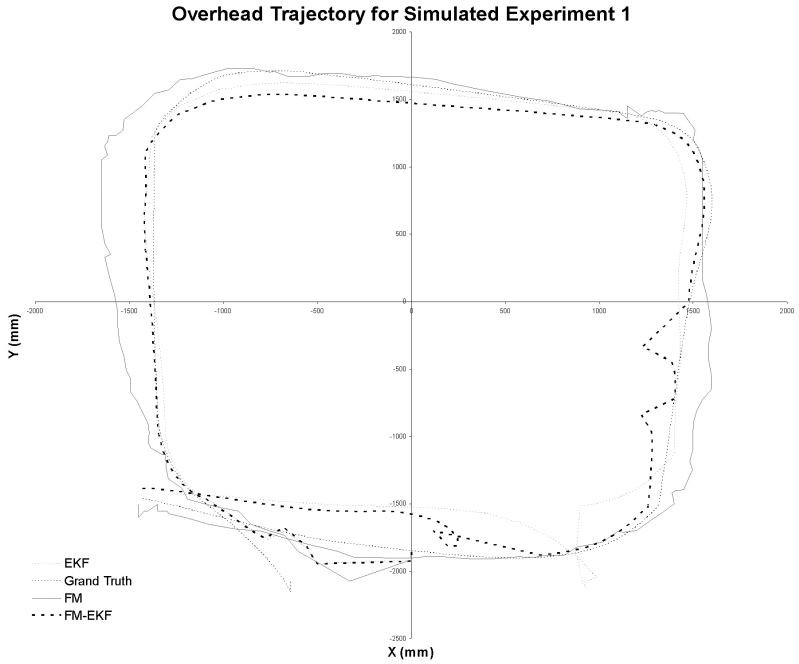

Fig. 15. Simulated experiment 1.

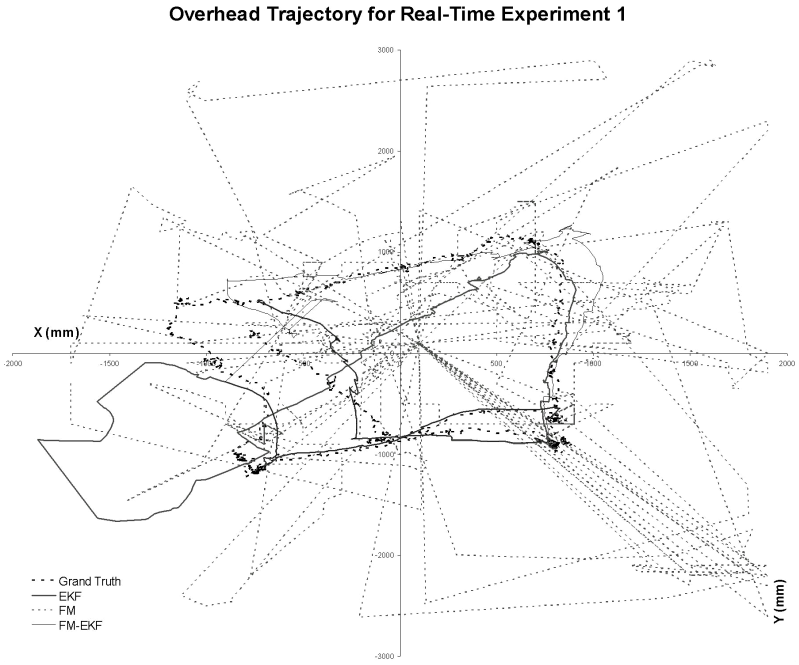

Fig. 16. Real-time experiment 1.

robot behavior operability and an analysis of measured parameters.

This HRI also evaluates the available positioning information obtained from robot visual perception. In this sense, we make use of experiments presented in Samperio [2008] to show the feasibility of proposed approach.

The visual information is also processed in order to implement an on the flight landmark model for detecting and classifying robot

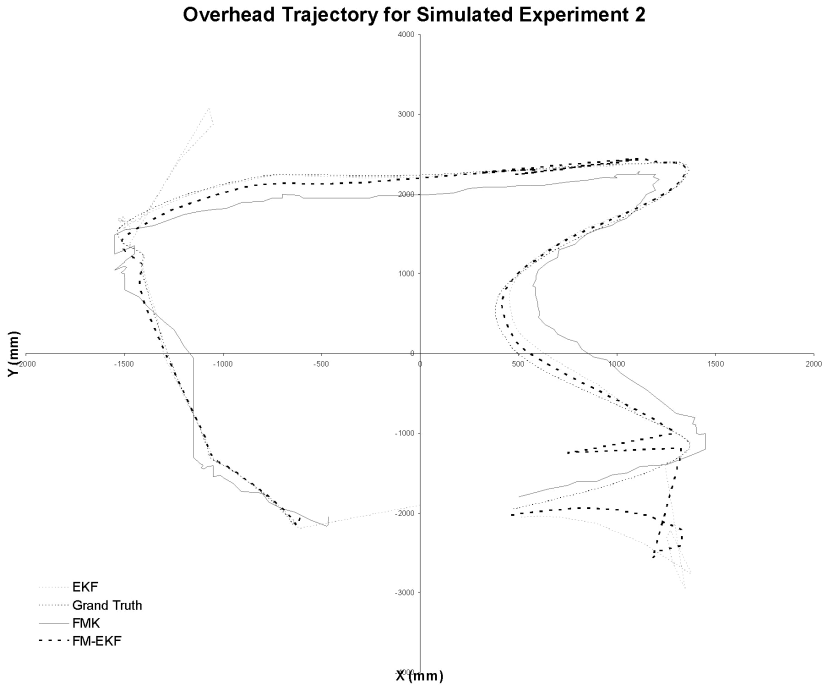

Fig. 17. Simulated experiment 2.

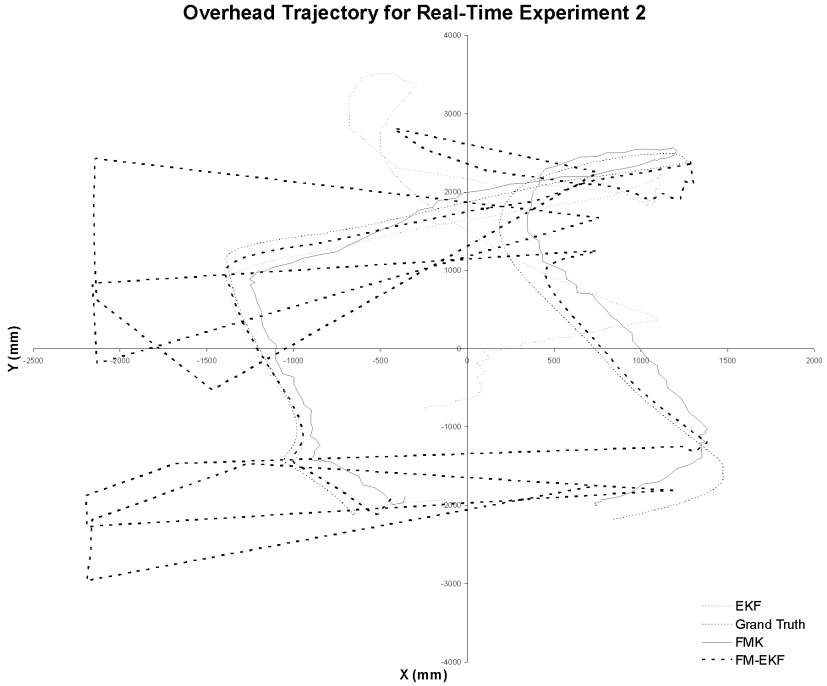

Fig. 18. Real-time experiment 2.

environmental features. Also, the HRI can redefine such landmark by adapting sensed information into a robot self-constructed map, and a further landmark definition.

Figure 21 shows a range of detected landmark poses from a static robot perspective that can be created in real-time. In this, map is possible to appreciate errors in landmark

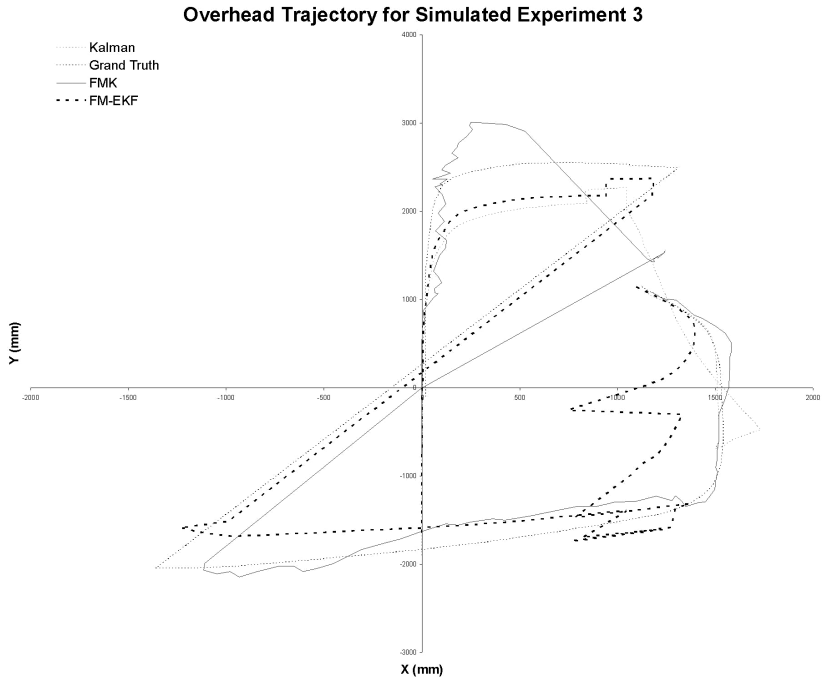

Fig. 19. Simulated experiment 3.

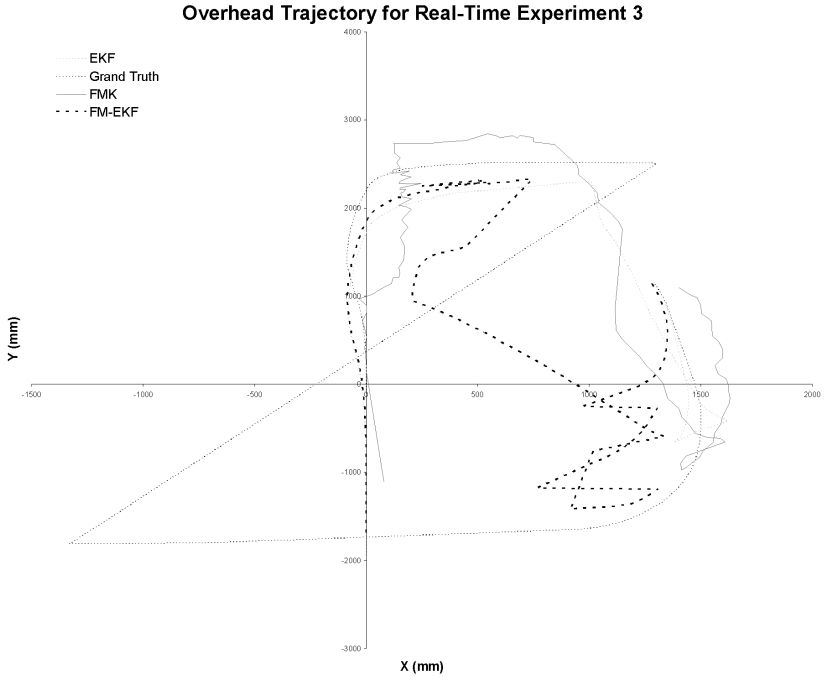

Fig. 20. Real-time experiment 3.

detection for a further visual calibration and a real-time perception analysis. Then in Fig. 22 is presented similar error detection when the robot follows a circular trajectory.

So on, the HRI is an adaptable control tool capable of presenting robot environment for further adaptable changes during experiment execution. Also, presented experiments

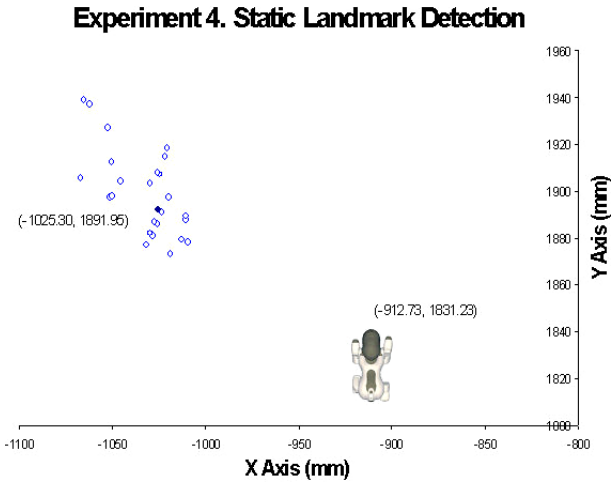

Fig. 21. Landmark detection from a static robot.

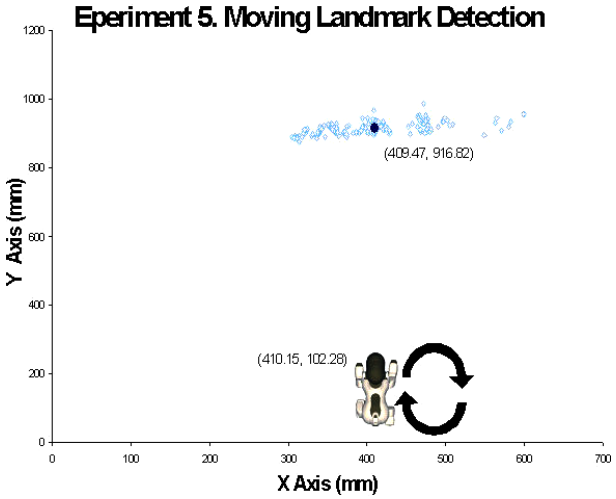

Fig. 22. Landmark detection from a moving robot.

demonstrate the robot-user interaction in an applied analysis methodology of robot localization.

6. Conclusions and Future Work

In this paper, we present an interactive HRI that is a useful tool for the development of robot localization algorithms in the RoboCup domain. It consists of a walking robot, an overhead visual tracking system, and a user interface. Such a system offers real-time control and evaluation facilities for localization analysis using a GUI as a central support tool. The overhead camera is used to provide ground truth to compare experimental results obtained from real robots with the ground truth.

The experimental results demonstrated that the proposed system is feasible and has good performance. Further research will focus on the development of an adaptable behavior-oriented architecture suitable for any sort of mobile robots with wireless communication, including suitable interfaces for user control, behavior design, robot interaction, and online analysis.

Acknowledgments

We would like to thank Dr Francisco Martin from Universidad Rey Juan Carlos, Dr Vicente Matellan from Universidad de Leon and Team-Chaos of the RoboCup four legged league for their programming work and the technical support from Essex University. Part of this research was also supported by the Mexican CONACyT government scholarship with reference number 178622.

References

Golubovic, D., Li, B. and Hu, H. [2004] "A hybrid software platform for sony aibo robots," in *IRoboCup 2003: Robot Soccer World Cup VII*, Vol. 3020/2004 (Springer, Berlin/Heidelberg), pp. 478–486.

Hugel, V., Amouroux, G., Costis, T., Bonnin, P. and Blazevic, P. [2005] "Specifications and design of graphical interface for hierarchical finite state machines," *Lecture Notes in Computer Science. RoboCup 2005: Robot Soccer World Cup IX*, Vol. 4020/2006 (Springer, Berlin/Heidelberg) 648–655.

Liu, J. and Hu, H. [2004] "3D simulation of autonomous robotic fishes," *International Journal of Automation and Computing* **1**, 42–50.

Lovell, N. [2004] "Real-time embedded vision system development using aibo vision workshop 2," *Proceedings of the Fifth Mexican International Conference in Computer Science*, pp. 268–274.

Martin, F., Gonzalez-Careaga, R., Canas, J. and Matellan, V. [2004] "Programming model based on concurrent objects for the aibo robo," *Proceedings of the XII Jornadas de Concurrencia y Sistemas Distribuidos*, 367–379.

V. MX and V. series Systems [2004–2005] *Polygon 3.1 Visualization and reporting tool. System*

Tutorial Revision 1.2., (Vicon Motion Systems Limited).

Ogata, H. and Takahashi, T. [1994] "Robotic assembly operation teaching in a virtual environment," *IEEE Trans. Robotics and Automation* **10**(3), 391–399.

Samperio, R. and Hu, H. [2006] "Kalman filter based localization for aibo walking robots," *Proceedings of the IEEE International Symposium on Robotics and Automation*, San Miguel Regla Hotel, Hgo., 25–28 August 2006.

Samperio, R., Hu, H., Martin, F. and Mantellan, V. [2008] "A Hybrid approach to fast and accurate localization for legged robots," *Robotica*.

Samperio, R. and Hu, H. [2008] "An interactive HRI for walking robots in RoboCup," *IEEE International Conference on Information and Automation.* 20–23 June, Hunan, China.

Samperio, R. [2008] "Visual-based localization for Aibo walking robot" *PhD. Thesis*, University of Essex.

Sangpetch, A. [2005] *Visualizing Robot Behavior with Self-Generated Storyboards*, (PhD thesis, Carnegie Mellon University), May 10.

Tzafestas, C., Palaiologou, N. and Alifragis, M. [2006] "Virtual and remote robotic laboratory: comparative experimental evaluation," *IEEE Transactions on Education* **49**, 360–369.

Yokote, Y. [1992] "The apertostreflective operating system: The concept and its implementation," *Conference on Object Oriented Programming Systems Languages and Applications*.

Huosheng Hu is a Professor in School of Computer Science and Electronic Engineering at the University of Essex, UK, leading the Human-Centred Robotics Group. He has held a number of research grants from the EPSRC, the Royal Society, EU, RAEng, as well as from industry. His research interests include behavior-based robotics, human-robot interaction, embedded systems, learning algorithms, pervasive computing, and service robots. He has published over 270 papers in journals, books, and conferences in these areas, and received a number of best paper awards.

He is one of the founding members of IEEE Robotics and Automation Society Technical committee on Networked Robots, a senior member of IEEE and ACM, and a member of IET and IAS. He is a member of the EPSRC Peer Review College. He has been a chair or committee member for many international conferences such as IEEE ICMA, IEEE ROBIO, IEEE IROS, RoboCup Symposiums, and IASTED RA, CA, and CI conferences. He currently serves as Editors-in-Chief for International Journal of Automation and Computing.

Dongbing Gu is a senior lecturer in School of Computer Science and Electronic Engineering at the University of Essex, UK. His current research interests include multi-agent systems, wireless sensor networks, distributed control algorithms, distributed information fusion, cooperative control, reinforcement learning, fuzzy logic and neural network — based motion control, and model predictive control. He has published over 80 papers in international journals and conferences. He has also served as the member of organising committees and programme committees for many IEEE conferences. He is the member of several IEEE technical committees and a senior member of IEEE.

Renato Samperio received his B.Sc. in Computer Science from Instituto Tecnologico de Estudios Superiores de Monterrey Campus Estado de Mexico, Mexico, in 2002. He is finishing his Ph.D. degree in Computer Science at University of Essex, UK. The research topic during his Ph.D. was "Visual based localization for Aibo walking robots."

He is also working as a researcher in the area of Logistics and Production Robotics at the German Research Centre for Artificial Intelligence (DFKI GmbH), in Bremen, Germany. His research interests include mapping, SLAM, robot navigation, computer vision, legged robots, mobile robotics, sensors fusion, and collaborative robotics.