Learning from Examples in Unstructured, Outdoor Environments

J. Sun* , **T. Mehta°** , **D. Wooden°** , **M. Powers***,

J. Rehg* , **T. Balch*** , **and M. Egerstedt°**

°School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332–0250
{tmehta,wooden,magnus}@ece.gatech.edu

*College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332–0250
{sun,mpowers,rehg,tucker}@cc.gatech.edu

Abstract

In this paper, we present a multi-pronged approach to the "Learning from Example" problem. In particular, we present a framework for integrating learning into a standard, hybrid navigation strategy, composed of both plan-based and reactive controllers. Based on the classification of colors and textures as either good or bad, a global map is populated with estimates of preferability in conjunction with the standard obstacle information. Moreover, individual feedback mappings from learned features to learned control actions are introduced as additional behaviors in the behavioral suite. A number of real-world experiments are discussed that illustrate the viability of the proposed method.

1 Introduction

Consider a situation in which a human operator is driving a robot through an unknown outdoor environment. One can typically expect the robot to stay on easily traversable regions, such as paths or flat ground, while staying away from dense vegetation and mud. Moreover, it is conceivable that the robot (through human control) reacts to distinctive features in the environment in particular ways, such as "turning left at the pine tree" or "staying 2 meters to the left of the creek". A natural objective is to have the robot mimic the human-operated behaviors and maneuvers in a completely autonomous manner. However, the problem is not that of reproducing the behavior at the level of trajectories (e.g. by simply storing the path that the human-operated robot took) but rather to *learn* at a behavioral level what features are important and how the human reacted to them. This is precisely the problem under investigation in this paper, i.e. the problem of determining how to autonomously control a mobile robot in such a way that it acts in a manner "similar" to the human operated example data.

The example scenario outlined in the previous paragraph implies that in order for the proposed "Learning from Example" program to be successful, learning has to take place at two different levels (learning based on *preferable regions* and learning based on *relevant features*) as well as focus on two different operational aspects of the navigation tasks (learning at the *perception* side, i.e. "What are the relevant features?", and learning at the *control* side, i.e. "How should the robot act given a particular feature?")

In this paper we will present an architecture for mobile robot navigation that supports the seamless integration of both of these two types of learning in a natural way. In particular, we will use an architecture in which a cost map is produced that captures both obstacle information (e.g. through terrain elevation) and preferability (classification of visual information based on color and texture cues). A global path is then planned through this map, producing a desired direction for the robot. However, robots typically have a configuration space of a higher dimension than is computationally tractable for global map-based path planning. Furthermore, the appearance of sudden obstacles close to the robot must be dealt with quickly. These two observations imply that the desired direction generated by the planner is just one of the factors that must be taken into account when controlling the robot. In typical fashion, we solve this problem by adding a number of low-level controllers (or behaviors), each dedicated to performing a particular task: e.g. avoiding obstacles, staying on smooth ground, or driving towards a known goal-location. These behaviors are combined with the planned path through a voting scheme, and this architecture is illustrated in Figure 1.

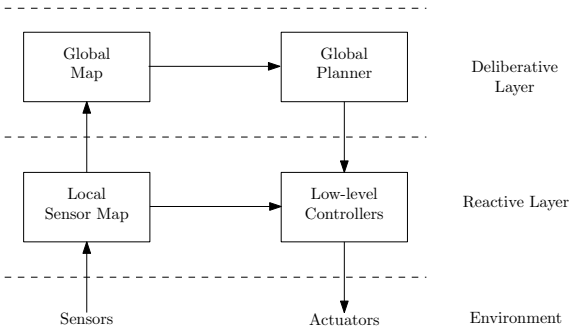


Figure 1: Standard Hybrid Control System Block Diagram.

The manner in which such an architecture can be combined with "Learning from Example" information is as follows:

- **Preferability:** The goal is to mimic the human operation from examples, by learning what terrain regions are preferred. For example, a road is as traversable as an open field, but is more preferable. Based on where the robot actually was driving during the example runs, the colors related to pixels over which the robot drove are associated with a "good" labelling, while colors over which the robot did not drive are associated with "bad". When running autonomously, the robot's cost map is populated with costs that are derived from not only stereo information, but also from the learned color based preferability classification. (This is the main issue in Section 4.)
- **Feature-Based Behaviors:** The predefined behavioral suite that dictates the motion of the robot will be augmented by another behavior, namely the *do-what-the-*

human-did behavior. This behavior will be treated as any other behavior by the arbitration mechanism. It is obtained by taking the example runs and storing relevant images and the corresponding velocities that the human operator had the robot execute in response to these images. During the autonomous runs, the *do-what-the-human-did* behavior searches the database of stored images for the best fit (or fits), and then sends the corresponding control action (or actions) to the behavioral arbitration module. (This is the topic of Section 5.)

The outlined "Learning from Example" problem is motivated by the DARPA LAGR (Learning Applied to Ground Robots) project, in which this problem is an explicit component. The outline of this paper is as follows. In Section 2, we recall some related work. In Section 3, we describe the system architecture in some detail. The reason why this is relevant to the problem under consideration in this paper is that an explicit goal with the proposed "Learning from Example" method is that it should integrate seamlessly with standard, effective navigation architectures in order for it to be a convincing and usable solution. Following this, in Sections 4 and 5, the actual machine learning processes are described, with Section 4 focusing on the *learning of preferable regions*, and Section 5 focusing on *learning how to act* based on images and high-level image features. Since the proposed methods have been deployed and tested extensively in real, outdoor environments, we report on some of the experimental results in Section 6, followed by the concluding remarks, in Section 7.

2 Related Work

Navigation in unstructured outdoor environments has received considerable interest during the last decade, arguably culminating with the DARPA Grand Challenge (DARPA, 2006). What makes outdoor environments particularly challenging is the inherent lack of *a priori* well-defined features. As such, robustness, adaptation, and flexibility become paramount concerns in that the system needs to satisfy the following constraints:

- **Robustness**: It must be able to handle a wide variety of features and terrain types. The classic division into traversable and non-traversable regions no longer holds because dense vegetation may or may not be traversable.
- **Adaptation**: It must be able to learn that certain types of terrain are preferable to others, e.g. based on color and texture cues, rather than mere obstacle-indicators like stereo-based elevation measurements.
- **Flexibility**: It must be modular in that a navigation architecture suitable for desert environments should still be useful in grassy terrain, possibly following a small change in the perception algorithms.

As such, this paper takes the point of view that learning techniques must be easily integrated into existing, standard navigation architectures in order to reach their full potential. We briefly relate our two learning strategies to previous work next.

Previous learning methods have been successfully applied to the robot navigation problem. For example, the ALVINN system (Pomerleau, 1989) used a neural network to learn steering

angle corrections for road-following. Other examples include the detection of open areas using learned visual features (T. M. Jochem, 1995; J. Michels, 2005). Other related work uses an online learning approach, where data acquisition and labelling is accomplished through the interaction between the robot and the environment. For example, (I. Ulrich, 2000) uses stereo sensing in the vicinity of the robot to provide the training data for a color-based classification to predict flat regions of the ground. A similar idea is pursued in (Thrun et al., 2006), where laser range data is used to drive the color-based classification of regions into drivable and undrivable areas - a key technique for their system, the winner of the DARPA Grand Challenge 2005. For outdoor robot navigation, (Kim et al., 2006) proposed a method to learn a direct mapping from image features to traversability, which is closely related to direct perception in affordance theory (Gibson, 1979). In contrast to this, non-direct mapping approaches have been proposed, but they require another layer of representation or logic functionality in order to decide the traversability of an area (C. Wellington, 2004). *Our work differs in the sense that besides avoiding non-traversable areas, we also enable the robot to select preferred regions among equally traversable terrain region candidates.* Knowledge of the fact that roads are more preferable than random flat ground is made possible through learning based on the human-operated example runs.

The learning methods described in the previous paragraph are ultimately going to be used in map-based applications in that traversable/preferable regions are identified and planned over. We complement this approach with the behavior-based approach, where we attempt to learn behaviors that closely resemble the motion demonstrated by the human operator. This approach has been increasingly used to identify complex biological behaviors (Egerstedt et al., 2005; Balch et al., 2006; Guillory et al., 2006; Webb, 2000). In (Egerstedt et al., 2005), behaviors were obtained from live ant data and used to simulate ant movements. In (Mehta and Egerstedt, 2005), new behaviors were learned as a combination of existing behaviors. In this paper, we learn behaviors using this approach of combining existing behaviors as well as learning appropriate mappings from observations to actions. The first approach is appropriate when relevant features are known so that behaviors exploiting these features can be designed. And, although not novel as a learning paradigm (see for example (N. Ratliff, 2006) in which Markov decision processes were employed for this purpose), our system is deployed in a truly unstructured environment in which neither features nor policies are crisp or well-defined. The design task involves combining these behaviors to achieve the desired overall behavior. Moreover, the proposed approach is quite general and can be applied when relevant features are not known in advance.

It should be stressed again that although a rich body of research exists on machine learning for robot navigation, *our explicit aim is to set the learning process up in such a way that they can be integrated seamlessly with established, stable control architectures.* The reason for this is that outdoor robotics is a challenging endeavor even in the absence of learning, and as such, it is beneficial to base the approach on established principles for outdoor navigation.

3 System Architecture

3.1 Robot Platform

The LAGR robot, depicted in Figure 2, possesses four color cameras, a front bump switch, a Garmin GPS receiver, and an internal inertial measurement unit. The cameras are paired together so that each pair can provide stereo depth maps with a range of approximately 6 meters. The robot's turning axis is centered on the front axle, with the back two unpowered wheels turning on casters. Its is 1.0 meters long, 0.6 meters wide, and 1.0 meters high, with a weight of 60 kg.

Figure 2: The LAGR Robot.

Mapping, control and planning processes are run on a single linux machine (2.0 Ghz, Pentium-M, 1 GB RAM), the *planning computer*. There are additionally three identical computers connected via ethernet: 2 for camera/stereo processing (called *eye computers*), and 1 for lower-level functions (e.g. radio-control interface, GPS/IMU integration). Additional detail of our system setup for the LAGR robot has been described in (Wooden, 2006).

3.2 Perception

Each of the two eye computers communicates with a pair of stereo cameras that collect color images. From these images, stereo depth maps are calculated. Given the known intrinsics and extrinsics of the cameras and the robot's global position, we transform a local terrain map onto a global coordinate frame. This information is subsequently communicated over the network to processes on the planning computer. On the same eye machines, the color camera images are used to compute estimates of preferability of points in the global frame. This process uses the appearance models of preferability learned offline from the example log files. The stereo and preferability processes run steadily at 4Hz.

Figure 3: Sample Camera Images.

3.3 Mapping and Planning

The planning computer receives the two streams of stereo and preferability information from both eye computers and incorporates it into global maps of terrain and preferability. This data is stored in grid cells of fixed size, with a resolution of 0.1 m. Under the assumption that the newest information is the most likely to be correct, previous information in a grid cell is overwritten with the new.

A separate map also stores the locations where the robot has encountered hits on its bumper, spikes in its motor amperes, and detections of wheel slippage.

Motion planning is executed over the global maps described above. The planning algorithm we use is either a heuristically improved A^* planner, or the combinatorial planner we described in (Wooden and Egerstedt, 2006). However, for the purposes of this discussion, the particular planner to be used is of no real consequence. The mapper passes on points which have been modified by the stereo/preferability/etc processes to the planner, which updates its planned path.

3.4 Behaviors

Reactive control is implemented in a manner heavily influenced by the DAMN architecture (Rosenblatt, 1997). In this implementation, individual behaviors – each designed for specific interests related to the robot's overall objective – are each given an allotment of "votes". The behaviors may cast these votes either *for* or *against* potential actions in a manner that works to achieve their particular goal. An arbitrator tallies the votes, choosing the action with the most support. This implementation utilizes straight-line paths at a resolution of 5 degrees around the robot as the action set. Similar implementations have been successfully deployed in several robotic navigation tasks (Williams et al., 2001; Rosenblatt, 2000).

As is true with many other behavior-based implementations, a potential danger in this architecture is the misallocation of each behavior's gain (or in this case its allotment of votes). If the behavior's voting weights are not properly balanced, one behavior's input may dominate the tally, either preventing the robot from achieving higher-level goals or allowing the robot to enter an undesirable state. Noting that this weighting is typically an empirical process and dependent on both implementation and the robot's environment, an additional layer of robustness has been added by supplementing the voting scheme with "vetoes" (R. Sukthankar, 1997). Each behavior, in addition to getting an allotment of votes to apply to the action set, is given the opportunity to veto each action in the set. The arbitrator respects the vetoes by disregarding any action that has been vetoed by any behavior, no matter how many votes it has garnered.

Strategically, vetoes are only used in cases when the robot faces "imminent danger". Of course, what qualifies as imminent danger is specific to each behavior. However the strength of this strategy lies in the fact that because a behavior needs only to consider imminent danger, complex calculations over the robot's configuration space that would be impossible in a full planner can be carried out. This allows the behaviors potentially to consider the full dynamics of the robot, including collision checking of rotations and the feasibility of maneuvers given the slope of the terrain.

When all of these building-blocks are combined, the result is a hybrid architecture both with respect to the control actions (through a combination of deliberative planning and local behaviors) and with respect to perception (through a combination of various visual cues.) The information obtained by the system at each time instant is depicted in the Georgia Tech LAGR Dashboard, as shown in Figure 5, where learned preferability, stereo depth maps, the current global cost map, and the planned path are displayed.

4 Color-Based Learning of Preferable Regions

4.1 From Example Data to Color Models

In this section, we explain our framework for learning a model of "preferable" vs. "non-preferable" terrains. In (Kim et al., 2006), we described a system that learns terrain traversability on the fly when the robot moves in the environment. However, there is a

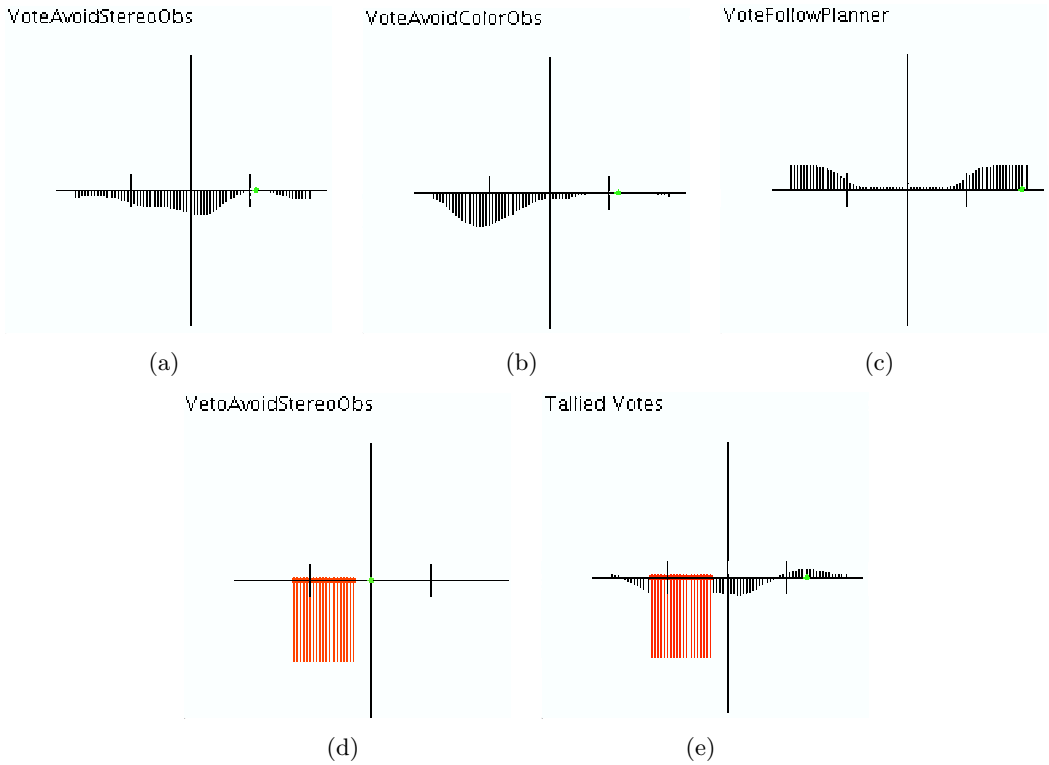


Figure 4: A graphical representation of the voting scheme employed to navigate the robot. The x axis of each plot represents an ego-centric angular distribution of possible paths around the robot from $-\Pi$ to $+\Pi$, 0 being in front of the robot. The y axis represents the relative preference of each path, according to the respective controller. Vetoes are drawn as large negative values. The last plot represents the sum of the votes provided by all the controllers. The largest non-vetoed value is chosen for action by the robot. In this example, the first behavior resists a stereo-perceived obstacles to the front and left of the robot. A color-based obstacle is perceived to the left. The plan tells the robot to go backwards, and the left is vetoed as a result of stereo obstacles. The tallied votes tell the robot to go to the right.

big difference between preference and traversability. While non-traversable terrains are certainly not preferred, traversable terrains are not always preferable. As an example, consider a robot driving to a goal point that is behind a large area of bushes. Assuming there exists a path around the bushes, the robot should ideally take that path, since the ground on the path is "consistently" traversable. The direct path through the bushes, although shorter, is less preferable. A more intelligent system should prefer the longer path, in order to achieve both efficiency and consistency. (A Robot that travels in less time is more efficient; the variance of the travelling time across runs determines consistency: large variance implies inconsistent performance and high risk.)

In this regard, it is natural to have a teacher that instructs the robot the concept of "preferable" terrains by way of providing training examples. Specifically, the teacher manually drives the robot in some similar environment, and the image sequences and robot state obtained by the cameras are stored in a log file. The learning procedure consists of two parts: data labelling and model learning. We use a *ground projected feature map* to obtain the

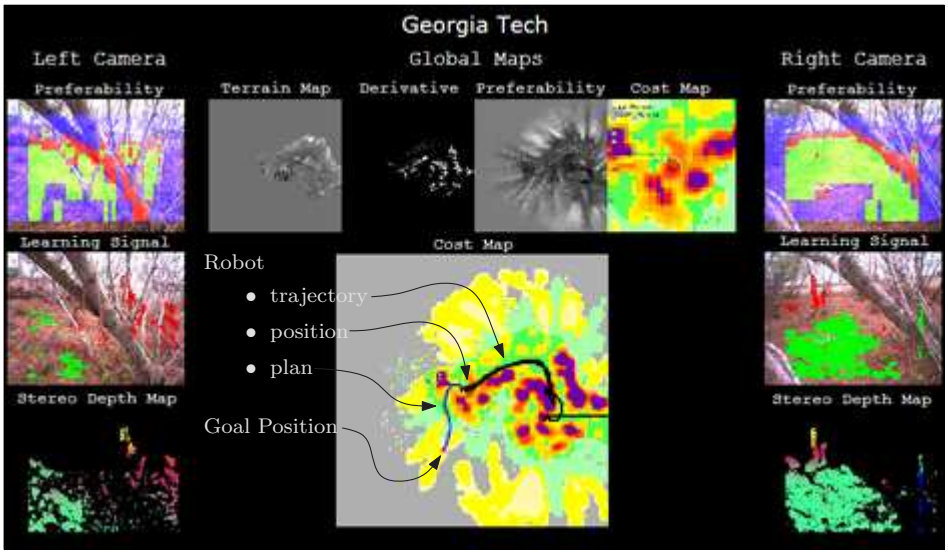
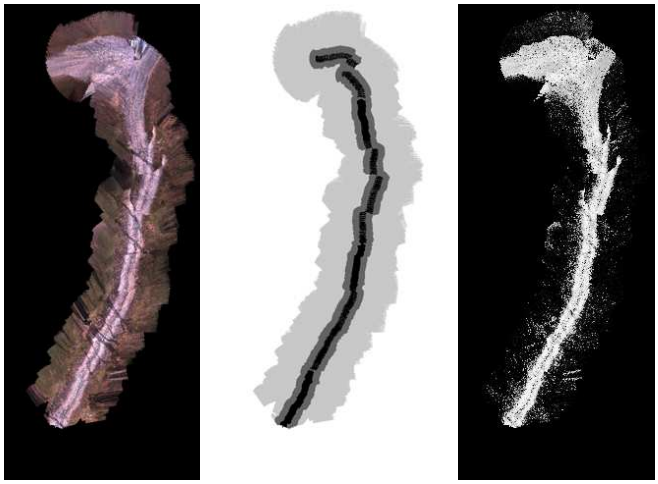


Figure 5: Sample Image of the Georgia Tech LAGR "Dashboard". The left and right columns contain information pertaining to the left and right stereo pairs, with the top figure depicting a classification of the image into preferable and non-preferable regions. The middle figure shows the learning signal used to establish preferability based on whether or not the depth map (lower figure) indicates the presence of an obstacle. The center column contains information relating to the planner, including a cost map and the resulting path from the current robot position to the goal location.

training data. For clarity, we explain our approach with a simplified *color-based* classifier. A benefit of a color-based classifier is that color of the terrain (as opposed to texture) does not change when it is observed at different distances. (Note that both the feature space and the classifier algorithm can be replaced for a different application.)

The main idea of the feature map is to project features of image patches to the terrain from which they come. For example, if we use RGB color as a feature, and we take each image patch to be exactly one pixel, then we are projecting image pixels from the camera to the ground plane. One such example is shown in figure 6(a). In practice, such a projection is either geometric or stereo-based. The depth map recovered by stereo cameras can correlate an image pixel to its 3D location relative to the robot, which can then be transformed into the global coordinates.

However, because stereo perception is limited to a short range (6 meters), regions of the camera images do not correspond to a any part of the stereo depth map. Nonetheless, distant portions of the image are informative: when the robot makes a turn to avoid objects in the distance, there is a good indication that they are non-preferable. In order to take advantage of these pixels in the learning process, we project these image pixels to the ground with a pure geometric projection. With the known camera intrinsics and extrinsics, we can project the image coordinates onto the ground plane. Because the ground plane assumption is invalidated for objects with non-zero height, we only apply this projection when there is no stereo information for that pixel in the image. To produce the feature map, at each



(a) (b) (c)

Figure 6: Learning with ground projected feature map (a) Color feature map (b) Mask of data labelling: black is positive, light grey is negative, dark grey is not used in training. (c) Encoded cost map classified with the learned model: lighter terrains are classified as more preferable

frame, we project the pixels onto the ground, and record the distance at which these pixels are observed. As new images are obtained when the robot moves, the feature map is over-written when the ground cell is observed at a *closer* distance. In this sense, such feature map is called a *minimum distance feature map*.

Once the feature map is obtained, we can label the training data and train a classifier off-line. However, there exists no automated labelling process that matches exactly the preferability of the human teacher. Thus, we adopt the following heuristics: the ground that is within a distance D_{close} of the robot's trajectory is considered to be positive, and negative data is taken from the ground more than a distance D_{far} from the robot's trajectory. In order to minimize mislabelling, we set D_{close} to be quite small (1.2 meters) and D_{far} to be relatively large (4 meters). Any data points between D_{close} and D_{far} are considered unknown and are not used for training (Figure 6(b)). Note that the distance information is immediately available from the construction of the feature map.

The training procedure is a standard two-class classification problem. Depending on the feature space that we pick, several machine learning techniques can be used, such as Boosting, SVM, decision trees, etc. In our case where the feature is the RGB color, we use a likelihood ratio test based on color histograms. Specifically, we build a $16 \times 16 \times 16$ RGB histogram for the positive and for the negative data sets. For each bin, we calculate the likelihood of $P(bin|class)$, where $class = \{0, 1\}$, with 1 positive and 0 negative. The likelihood ratio test is

$$\frac{P(bin|1)}{P(bin|0)} > T \tag{1}$$

for positive, where the threshold T is determined in training. In practise, such 0-1 binary classification is unnecessary, since the classification is to be integrated into a cost map based

(a) (b)

Figure 7: Predicting terrain preferability: (a) Camera color image (b) Classified cost of preferable terrain: lighter terrains are classified as more preferable

on additional channels of information (e.g. terrain variation). Thus, it would be better to provide a continuous cost function instead of a binary classification. We use

$$\frac{P(bin|0)}{p(bin|1) + p(bin|0)} \tag{2}$$

as the cost of learned preferability (as in Figure 6(c)).

With the learned terrain model, the robot is able to predict the preferable terrain with continuous camera images from both eyes (instead of a ground project feature map). For each frame, based on the features calculated in the image (e.g. color), we can calculate the cost of each pixel (figure 7). We then send the cost of each pixel and its relative location with respect to the robot to the control process, which is integrated with other cost measures such as stereo determined obstacles. Note that there is a lot of redundancy in the cost messages across frames, thus we limit the classification to a 4Hz frame rate. The overall learning and detection system is illustrated in Figure 8.

4.2 Integration of Learned Preferability in Cost Maps

The above section describes how camera images are transformed into preferability information in a coordinate frame centered on the robot. That is, a pixel's RGB color is mapped to preferability (in the range [0,1]), and the its coordinate in the image frame is projected into the planar coordinate frame immediately around the robot. This local information from this one camera image must next be projected into a global planar coordinate frame (e.g. based on GPS information), and incorporated with previous measurements of preferability.

4.2.1 Projection

When preferability estimates must be projected into the global coordinate frame, and no stereo information is present about the location of the pixel in that frame, the pixel is

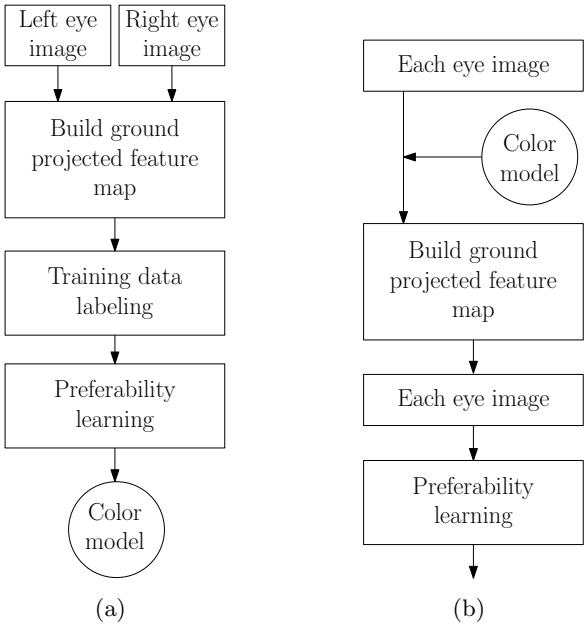


Figure 8: Learning block diagram (a) Learning preferability from example (b) Classifying preferability online

projected into the world under the (often false) assumption that the world is flat. This has the effect of spacing out pixels at the top of the image, as depicted in Figure 9.

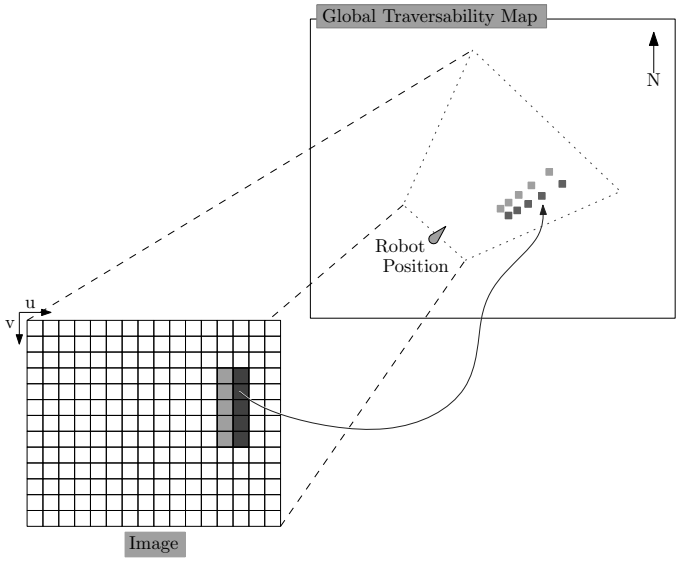


Figure 9: Illustration of Terrain Preferability Projection - from Camera Frame to Global Frame.

In order to ameliorate this ill-effect, image pixels are projected into the global map onto a corresponding $n \times n$ patch in the global frame, and n, the size of the patch, depends on the distance the projected patch is from the robot. In other words, the preferability estimate of a distant location is projected onto a large region. As a result, the preferability projection

resembles that shown in Figure 10.

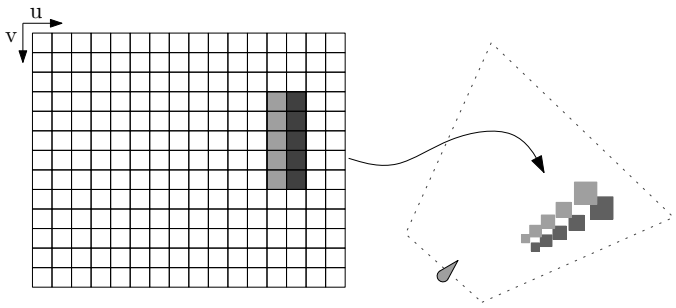


Figure 10: Proportional-Patch Projection.

Two notes should be made.

- The patches written into the global map are squares oriented to the global frame. That is, the patches are not rotated to match the yaw angle of the robot's global position.
- During this projection, some patches will overlap others and some "holes" are left unfilled.

Both items represent a simple way to perform the projection, in terms of computational and coding complexity, yielding an imperfect method. However, these imperfections are mitigated by the fact that the robot is constantly moving (by deliberate movement, as well as by localization noise), and that the way in which the patches are incorporated into the map is based on a low-pass filter.

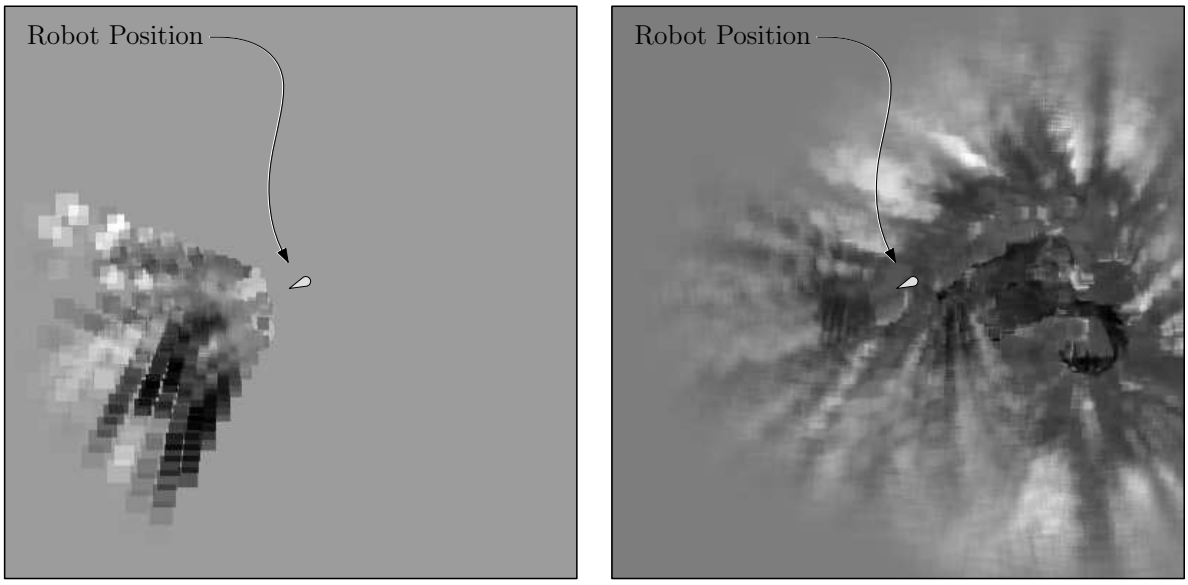
4.2.2 Incorporation

The local-to-global coordinate transformation is handled by a simple rigid-body rotation and translation, using the robot's current position and heading provided by an independent localization process. A new measurement of preferability at a pixel is incorporated with old measurements by means of a simple low-pass filter:

$$t(i,j) = \alpha\ t_{new} + (1-\alpha)\ t(i,j), \tag{3}$$

where i and j are indexes into the global map, $t(i,j)$ is the preferability stored at that location, and t_{new} is the new preferability estimate. The filter coefficient, α, depends on the distance the preferability estimate is from the robot at the time the measurement is made. We have used a simple linear mapping, where a distance of zero maps to $\alpha = 0.4$, and a distance of 20 meters maps to $\alpha = 0.0$. In other words, more distant estimates, which we expect to be less accurate than those made near to the robot, are given less influence in the map.

The figure below shows the global preferability map at different stages in a test run. Dark regions correspond to non-preferable areas, and light regions correspond to preferable areas. When the map is initially created, each cell is set to a value 0.5.



(a) (b)

Figure 11: Sample Terrain Preferability Images. (a) The global traversability early in a test run. (b) The map during the middle of a run.

5 High-Level Learning of Behaviors

5.1 Learning Behaviors from Known Features

In this section, we propose to solve the "Learn From Example" problem using experience (training data) to learn optimal weights as a basis for the behavioral voting scheme. We initially assume that some relevant features are specified or known, and we start by designing a set of behaviors to exploit these known features (e.g. "follow-path", "follow-path-to-goal", "turn-left-at-orange-fence"). These behaviors complement the initial set of behaviors which include "move-to-goal", "avoid-obstacles," etc. Now that an assortment of behaviors has been designed, the main challenge is to determine the weights (or allotment of votes) associated with each behavior. Clearly, the overall behavior will vary greatly based on the individual behavioral weights, i.e. the importance designated to each behavior. As mentioned earlier, it is desirable for these weights to be evenly distributed so that one behavior does not dominate arbitration. The overall objective is to select these behavioral weights such that the overall robot behavior closely resembles the behavior observed from the training data.

Formally, let Y be the set over which the measurements (e.g. images) can take on values and U be the set of control values. Each behavior is a mapping from the set of observations to the set of control values, i.e. $\kappa : Y \rightarrow U$. Given a set of such behaviors $K = \{\kappa_1, \kappa_2, \ldots, \kappa_p\}$, the overall robot behavior is given by some function of these behaviors and the weights associated with each behavior (see Section 5.3). The final control value (e.g. desired angular velocity) is given by $u = f(\kappa_1, \ldots, \kappa_p, \alpha_1, \ldots, \alpha_p)$, where $\alpha_i \in \mathbb{R}$ is the weight associated

with behavior κ_i. The problem under consideration in this section is find α such that:

$$\min_{\vec{\alpha}} \int_0^T L, \tag{4}$$

where $L = \|Q(u_t) - Q(u)\|$ is the instantaneous cost, $Q : U \rightarrow U_q$ is a finite precision quantization operator, and $\vec{\alpha} = (\alpha_1, \ldots, \alpha_n)^T$ is the control vector. In the instantaneous cost above, u_t represents the angular velocity from the training data and u is the output of our controller given the same observation. Note that we choose to quantized the angular velocity ($Q(u)$) before computing the error in order to avoid fitting noise, which may be the case if we attempt to minimize the true error between angular velocities. This problem occurs because angular velocity is not always a relevant measure of the actual desired behavior. For example, suppose we have two drivers manually driving a robot through the same course. One driver may be more aggressive and make sharper turns (higher angular velocities), while the less aggressive driver may make slower turns. However, both drivers successfully maneuver through the course and it is not clear that one approach is better than the other. In order to avoid this pitfall, we introduce a coarse quantization (e.g. straight, soft-left, hard-left, soft-right, hard-right) which helps us converge to a meaningful solution for α. Note that although angular velocities may not perfectly represent the human operator's decisions, it is all we have to go on.

We propose to solve this problem using an approach similar to the genetic algorithm (Mitchell, 1996). The basic idea is to quickly search the solution space (which here is $\mathbb{R}^p$) to find a good suboptimal solution. The general algorithm is given in Figure 12. We start with an initial guess of the control vector, which gives each behavior an equal number of votes. Next, we generate a set of possible solutions around this guess. For example, with at time equal to k and $p = 2$, $\vec{\alpha}^{(k)} = [1, 1]$, then $\mathbb{S}_k = \{[1-\delta, 1-\delta], [1-\delta, 1], [1-\delta, 1+\delta], [1, 1-\delta], [1, 1], [1, 1+\delta], [1+\delta, 1-\delta], [1+\delta, 1], [1+\delta, 1+\delta]\}$. Here δ encodes how far from the seed weights, the new candidate weights are allowed to be. Note that this is just one way to generate the set of possible solutions, there are many other acceptable approaches. Next, we evaluate each of these solutions with respect to the cost function defined above. Then we take the best candidate (e.g. the control vector in $\mathbb{S}$ with the lowest cost), and repeat the process by generating samples around this solution. The search terminates when either the best candidate is the same as the seed (e.g. $\vec{\alpha}^{(k+1)} = \vec{\alpha}^{(k)}$) or the cost difference between the best candidate and the seed is less than ϵ (e.g. $\left\|J(\vec{\alpha}^{(k+1)}) - J(\vec{\alpha}^{(k)})\right\| < \epsilon$). Note that the selection of δ may be critical in the convergence of this algorithm. If δ is too big, the solution may not be close to the optimal solution. On the other hand, the algorithm may take a long time to converge if δ is too small. The selection of an appropriate δ will vary based on the application.

5.2 Learning Behaviors from Unknown Features

In this section, we discuss learning methods when we do not have prior knowledge of relevant features. Here, the method described in the previous section is no longer applicable since we assumed that a given set of behaviors (and thus also features) would be used. However, we would still like to benefit from the training examples. We already have a rich set of behaviors defined with respect to particular features (e.g. "move-to-goal", "avoid-obstacles", "follow-

initialize

- Let vector $\vec{\alpha}^{(0)} = [1, 1, \ldots, 1]^T$, let $k = 0$

loop

- Generate a set of possible solutions $\mathbb{S}_k$ around $\vec{\alpha}^{(k)}$
- Evaluate each solution with respect to the cost defined above
- Let $\vec{\alpha}^{(k+1)}$ be the control vector with the minimum cost
- **if** $(\vec{\alpha}^{(k+1)} == \vec{\alpha}^{(k)}$ **or** $\left\|J(\vec{\alpha}^{(k+1)}) - J(\vec{\alpha}^{(k)})\right\| < \varepsilon)$ **exit**
  else k = k+1, **loop**

Figure 12: An algorithm for solving Eq (4) by approximation.

smooth-gradient", and others). We propose to augment these behaviors with additional behaviors that exploit the information learned from the training examples. To this end, we aim to learn/derive behaviors using the training data. In other words, we want to construct a mapping $\kappa : Y \rightarrow U$. In fact, we will construct two behaviors corresponding to each eye computer. The observation in this case will be the image taken from the left or right camera pair, while the control will be the desired angular velocity (quantized as in the previous section).

We propose to save all the distinct mappings from Y to U gathered from the training examples. Then at run-time, we can use these mappings as a "look-up" table (with the observation closest to the y's stored in our database of mappings) to determine the corresponding control action. A block diagram describing the off-line (learning) and run-time (execution) architecture is shown in Figure 13. Two immediate concerns of this approach may be storage space and computational cost. First, we have a limited amount of storage space, hence we cannot store several large images. Second, the computation cost must also be managed in order to be able to return control values every control cycle (4 Hz). In light of these concerns, as well as the fact that we do not know relevant features, we propose to significantly down-sample the images before we store them (the down-sampled images were 12×16 for our application). Also to reduce the size of the database, we require that a new mapping be added only if either

1. The mean squared error of the down sampled image with respect to the saved images is greater than some threshold, or
2. The mean squared error is less than the threshold, but the control action is different.

So we go through each of the training examples and save all the mappings (sampled image $\rightarrow$ desired angular velocity) as described above. Note the size of possible mappings (and as a result performance) will depend on the threshold variable. Thus, at the end of the learning phase, we will have a separate set of mappings for the left eye and the right eye.

Now at run-time, during each control cycle, an image is read from each eye. Next we compute the closest matched image from the related database (i.e. left or right). The control action corresponding to the closest matched image is sent back to the controller along with a

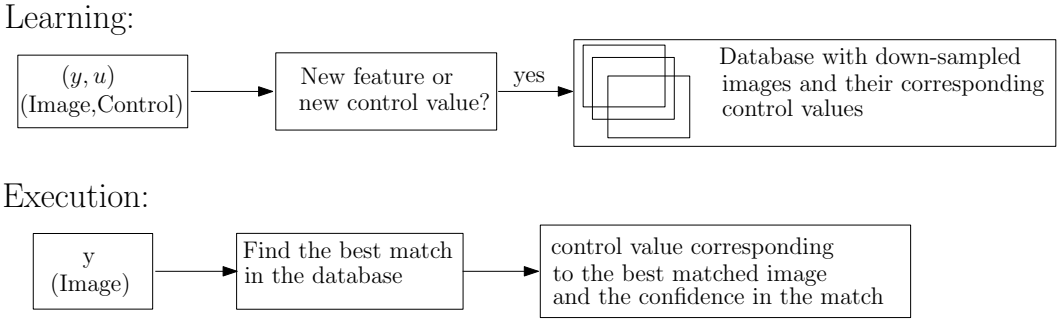


Figure 13: Block diagram depicting the learning and execution architecture.

confidence value representing the closeness of the match. The confidence value represents the error between the observed image and the matched image. Note that during each control cycle, two votes for the desired angular velocity are sent to the controller (one corresponding to each eye). These constitute votes from two different behaviors in our control architecture.

5.3 Integration of Learned and Predefined Behaviors

The learned behaviors are integrated with predefined behaviors in a manner consistent with the architecture described in Section 2. In the case of the behaviors learned from known perceptual features, the behaviors are allotted votes in proportion to their learned weights. In the case of behavior learned from unknown perceptual features, only one behavior is returned, which is given full weight. In both instances, the learned behaviors are combined with other pre-defined safety behaviors (e.g. *avoid-obstacles*) which are designed to keep the robot out of dangerous situations but not interfere with the learned behavior. Finally, the confidence returned by the classification algorithm is used to dynamically switch between the learned behavioral suite and a pre-defined default behavioral suite, designed with an appropriate, albeit naive, navigation strategy. That is, when the current situation is highly correlated with the learned cases, the learned behaviors control the robot. When the current situation is not highly correlated with the learned cases, the predefined default behaviors take over.

More formally, we have three sets of behaviors: a set of *learned* behaviors B^l, *default* behaviors B^d, and *safety* behaviors B^s. Each behavior outputs a vote distribution over the quantized angles from $-\pi$ to π. Behaviors within the sets above are combined according to equation

$$v^k = \mathcal{C}^k \sum_{b_i \in B^k} b_i w_i, \tag{5}$$

where w_i is the weight on the behavior b_i. The coefficient $\mathcal{C}^k$ is a confidence that scales the behavior suite B^k. The output v^k is a vector containing the behavior suite's vote distribution. The confidence associated $\mathcal{C}^l$ associated with the *learned* behavior suite, ranges from 0 (not confident) to 1 (very confident). For the *default* behavior suite, $\mathcal{C}^d = 1 - \mathcal{C}^c$. For the *safety* suite, $\mathcal{C}^s = 1$.

Finally, the behavior suites are combined as

$$V = \sum_{k \in K} v^k, \tag{6}$$

where K is set composed of the behavior suites: learned, default, and safety. The argmax of final vote distribution V determines the desired heading for the robot.

6 Experimental Results

Tests of the integrated system were performed at various test sites around the Atlanta, GA area. It should be stressed that all algorithms presented here have been extensively tested and evaluated within the LAGR monthly competition format (DARPA, 2005). Moreover, one outcome of these competitions is the difficulty with which one can objectively evaluate the relative merits of a system's individual components, meaning that the holistic system-wide viewpoint advocated in this paper might be necessary.

6.1 Color-Based Learning of Preferable Regions

Learning of preferable regions was tested at a vacant lot in Mableton, GA. The lot is primarily flat, but strewn with piles of landscaping waste (e.g. dead trees, bushes, and brush) creating a tangle of traversable "roads" among the large piles of obstacles, shown in Figure 14.

Figure 14: Test site for color-based learning of preferable regions.

Because of the tangled nature of the site's roads, there are potentially many acceptable routes between any two points at the site. For the test, start and goal points were chosen such that there existed a marginally traversable direct route to the goal and an easily traversable longer route. The robot was allowed to run the course several times without the benefit of learning to establish that the shorter, more difficult route was preferred by the naive system. Then, a path of cedar-wood mulch was laid along the longer, easier route. This mulch provided a distinguishable color-based signal to the robot, differentiating the two routes. A human operator then guided the robot along an entirely separate mulch path, providing a training set for the color-based preferability classifier.

In the next round of tests, the robot was able to classify the mulch path as the learned

concept, classifying it as very preferable, according to the learning strategy described in Section 4. In particular, the addition of a color-based learned preferable region module made the robot take a dramatically different (more reasonable) path to the goal. Figure 15 compares a raw camera image of the mulch path and a classification of the same image. The mulch path is clearly visible in the classified image as an area of high correlation to the learned preferability concept. The lower cost associated with this level of preferability caused the planner to lead the robot down the mulch path, to the goal. The final cost map created by the robot is shown in Figure 16. The mulch path is clearly visible as a region of low cost along the robot's trajectory.

Figure 15: Example raw and classified camera images from the robot following the mulch path. In the classified image, lighter colored regions correspond to terrain learned to be preferable.

6.2 Learning Behaviors from Known Features

Learning behaviors from known features was tested at the same site as color-based learning of preferable regions. This example corresponds to one of the LAGR tests, where the objective was for the robot to drive autonomously from the start position to a specified goal location while emphasizing the behavior learned from the provided positive training examples. The course was designed such that appropriate learning from training examples significantly simplified the navigation task. A relevant feature (namely a white path laid out using lime) was clearly identified. In light of this feature, we added "follow-path", "follow-path-to-goal", "follow-path-from-goal" and "avoid-path" behaviors to complement the existing "move-to-goal", "follow-smooth-gradient", "follow-free-space", "avoid-obstacles" and "veto-obstacles" behaviors. Then we used our algorithm to learn the optimal weights for each of these behaviors from the given training examples according to the method described in Section 5.1. (Note that this could just as well have been achieved with a color-based learning scheme, as discussed in the previous paragraphs.) We started by letting each behavior have equal weights and let $\delta = 0.1$. The algorithm converged with positive weights (0.1) for "follow-path", "follow-path-to-goal" and "follow-smooth-gradient" behaviors and zero weight for the other behaviors. Figure 17 shows sample shots from the test of this learned behavior in a course we constructed. The angular velocity was quantized to 5-levels (straight: $u_q = 0$ if

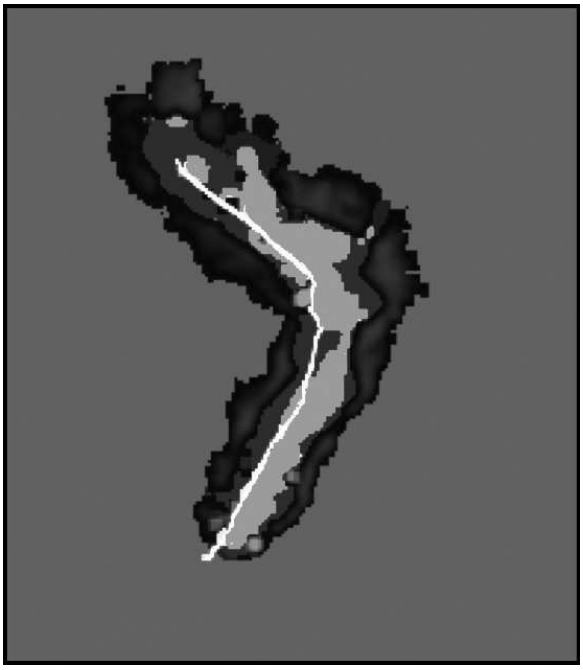


Figure 16: The final cost map from test on the mulch path. Lighter colored regions correspond to areas of lower cost (which, by extension, correspond to preferable terrain). The white line shows the robot's trajectory down the path. The location of the mulch path is clearly visible as a band of low cost along the robot's trajectory.

$-\frac{\pi}{8} \leq u \leq \frac{\pi}{8}$, soft-left: $u_q = -1$ if $-\frac{\pi}{2} \leq u < -\frac{\pi}{8}$, hard-left: $u_q = -2$ if $u < -\frac{\pi}{2}$, soft-right: $u_q = 1$ if $\frac{\pi}{8} < u \leq \frac{\pi}{2}$, hard-right: $u_q = 2$ if $u > \frac{\pi}{2}$).

(a) (b) (c)

Figure 17: Sample images from the test run navigating using the learned behaviors.

6.3 Learning Behaviors from Unknown Features

The more challenging problem of learning behaviors form unknown features based on the method of Section 5.2, was demonstrated using data acquired at a small park in Roswell, GA. Figure 18 shows the saved images along with the quantized desired angular velocity (represented by an arrow) from the left eye camera pair. (Note that the actual images stored are a lot smaller, but a larger image is shown for illustrative purposes.) The main idea here is that color and texture based region classification may not always be sufficient. Instead,

we learn directly over the images themselves, thus hoping to capture the relevant features in the environment.

This particular example corresponds to a training run at the test site. With our selected threshold, 8 mappings were stored from 463 possible mappings. Figure 19 shows some sample mappings during runtime. The image on the left is the actual image seen by the left-eye, the middle image is the down-sampled image, and the image on the right shows the best matched image along with the desired control action.

Here we repeat this process for a training run at a test site located at Stone Mountain, GA. In this case, we saved only 5 mappings from a possible 809 mappings. The results are shown in Figure 20, while Figure 21 shows some run-time mappings.

Figure 18: Example 1: LeftEye Database from data acquired in Roswell, GA. The database consists of 8 mapping from images to quantized desired angular velocity (represented by an arrow on each image).

Figure 19: Example 1: Sample run-time mappings during execution. The image on the left is the actual image seen by the left-eye, the middle image is the down-sampled image, and the image on the right shows the best matched image along with the desired control action.

7 Conclusions

In this paper we propose a two-pronged approach to the problem of learning how to maneuver a mobile robot from example data. In particular, we propose both to learn how to associate costs with different colors based on the observation that colors that the human operator drove over are more likely to be good than those that the operator did not drive over. This

Figure 20: Example 2: LeftEye Database constructed from data acquired in Stone Mountain, GA. The database consists of 5 mapping from images to quantized desired angular velocity (represented by an arrow on each image).

Figure 21: Example 2: Sample run-time mappings during execution. The image on the left is the actual image seen by the left-eye, the middle image is the down-sampled image, and the image on the right shows the best matched image along with the desired control action.

information is then used by a global planner to produce an optimal path from the robot to the goal.

Due to the fact that the robot's true configuration space is not planar, while the computational aspects of path-planning make planarity more or less a necessary assumption, the plan-based solutions are augmented by a number of local reactive behaviors. As part of the proposed "Learning from Example" program, behaviors are also learned by identifying relevant features with their corresponding control actions.

It should be stressed that we do not present any novel learning techniques *per se*, but rather show how learning can be integrated in a seamless and robust manner with existing, established navigation, perception, and control architectures. A number of experimental results show the effectiveness of the proposed approach.

Acknowledgments

This work was supported by DARPA through grant number FA8650-04-C-7131.

References

Balch, T., Dellaert, F., Feldman, A., Guillory, A., Isbell, C., Khan, Z., Pratt, S., Stein, A., and Wilde, H. (2006). How a.i. and multi-robot systems research will accelerate our understanding of social animal behavior. *Proceedings of the IEEE*.

C. Wellington, A. S. (2004). Online adaptive rough-terrain navigation in vegetation. In *IEEE Intl. Conf. on Robotics and Automation*.

DARPA (2005). http://www.darpa.mil/ipto/programs/lagr.

DARPA (2006). http://www.darpa.mil/grandchallenge.

Egerstedt, M., Balch, T., Dellaert, F., Delmotte, F., , and Khan, Z. (2005). What are the ants doing? vision-based tracking and reconstruction of control programs. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, Barcelona, Spain.

Gibson, J. J. (1979). *The Ecological Approach to Visual Perception*. Houghton Mifflin.

Guillory, A., Nguyen, H., Balch, T., and Isbell, C. (2006). Learning executable agent behaviors from observation. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*.

I. Ulrich, I. N. (2000). Appearance-based obstacle detection with monocular color vision. In *AAAI National Conference on Artificial Intelligence*, pages 866–871.

J. Michels, A. Saxena, A. Y. N. (2005). High speed obstacle avoidance using monocular vision and reinforcement learning. In *International Conference on Machine Learning*.

Kim, D., Sun, J., Oh, S. M., Rehg, J. M., , and Bobick, A. (2006). Traversability classification using unsupervised on-line visual learning for outdoor robot navigation. In *IEEE Intl. Conf. on Robotics and Automation*, Orlando, FL.

Mehta, T. and Egerstedt, M. (2005). *An Optimal Control Approach to Mode Generation in Hybrid Systems*.

Mitchell, M. (1996). *An introduction to genetic algorithms*. MIT Press, Cambridge, MA.

N. Ratliff, J. Bagnell, M. Z. (2006). Maximum margin planning. In *International Conference on Machine Learning*.

Pomerleau, D. (1989). Alvinn: An autonomous land vehicle in an neural network. *Advances in Neural Information Processing Systems*, pages 305–313.

R. Sukthankar, D. Pomerleau, C. T. (1997). A distributed tactical reasoning framework for intelligent vehicles. In *Proc. of Intelligent Systems and Manufacturing*.

Rosenblatt, J. (1997). Damn: A distributed architecture for mobile navigation. *Journal of Experimental and Theoretical Artificial Intell.*, 9(2):339–60.

Rosenblatt, J. (2000). Maximizing expected utility for optimal action selection under uncertainty. *Autonomous Robots*, 9(1):17–25.

T. M. Jochem, D. A. Pomerleau, C. E. T. (1995). Vision-based neural network road and intersection detection and traversal. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, pages 344–349.

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., , and Mahoney, P. (2006). Stanley, the robot that won the darpa grand challenge. *Journal of Field Robotics*.

Webb, B. (2000). What does robotics offer animal behaviour. *Animal Behaviour*, 60:545558.

Williams, S. B., Newman, P., Rosenblatt, J., Dissanayake, G., and Durrant-Whyte, H. (2001). Autonomous underwater navigation and control. *Robotica*, 19(5):481–496.

Wooden, D. (2006). A guide to vision-based map building. *Robotics and Automation Magazine*, 13:94–98.

Wooden, D. and Egerstedt, M. (2006). Oriented visibility graphs: Low-complexity planning in real-time environments. In *IEEE Conf. on Robotics and Automation*, pages 2354–59.