Reinforcement Learning for Landmark-based Robot Navigation

Dídac Busquets, Ramon López de Màntaras, Carles Sierra
Artificial Intelligence Research Institute (IIIA)
Spanish Council for Scientific Research (CSIC)
Campus UAB, 08193 Bellaterra, Barcelona, Spain
{didac,mantaras,sierra}@iiia.csic.es

Thomas G. Dietterich
Oregon State University
Corvallis, OR, 97331 USA
tgd@cs.orst.edu

ABSTRACT

In previous work, we described a cooperative multi-agent system (MAS) for coordinating various subcomponents of a robot navigation system. In this system, manually-tuned bidding functions controlled the interaction between different agents to guide a robot to a desired target location. A difficulty with hand-tuning is that it is hard to handle situations involving complex tradeoffs. This paper explores the suitability of reinforcement learning for automatically tuning agents within an MAS to optimize a complex tradeoff. Based on high-fidelity simulations, we conclude that reinforcement learning can learn better bidding functions than our hand-coded procedures. The learned bidding functions yield higher probability of success, shorter navigation trajectories, and reduced use of expensive sensing actions. However, we also found that correct functioning of reinforcement learning required extensive experimentation with the state representation.

1. INTRODUCTION

Robot systems for navigating through unknown environments are a focus of research in many application areas including spacecraft (rovers for Mars and the Moon), hazardous materials handling, and search and rescue robotics. In these applications, the robot must be able to start in an unknown location and navigate to a desired target using visually-acquired landmarks. The specific scenario that we are studying assumes that there is a target landmark that the robot is able to recognize visually. The target is visible from the robot's initial location, but it may subsequently be occluded by intervening objects. The challenge for the robot is to acquire enough information about the environment (locations of other landmarks and obstacles) so that it can move along a path from the starting location to the target position. The robot should do this quickly but safely (i.e., without getting lost).

In this paper, we build upon our previously described architecture for outdoor landmark-based navigation [6]. The architecture is composed of three systems, as shown in Figure 1. Each system competes for two available resources: motion (direction and amount of movement) and vision (direction of gaze and complexity of image processing). The three systems have the following responsibilities. The Pilot is responsible for all motions of the robot legs or wheels. It selects these motions in order to carry out commands from the Navigation system and (independently) to avoid obstacles. The Vision system is responsible for identifying and tracking landmarks (including the goal landmark) and for detecting obstacles to support obstacle avoidance. The process of identifying landmarks requires comparing images taken before and after suitable robot motions, so the Vision system must ask the Pilot system to carry out certain motions to support perception. Finally, the Navigation system is responsible for choosing higher-level robot motions in order to move the robot to a specified goal. This requires requesting the Vision system to identify and track landmarks (in order to build a map of the environment) and requesting the Pilot to move the robot in various directions (in order to aid the Vision system or in order to move the robot toward the goal position or toward some intermediate target position).

From this brief description, two observations can be made. First, these three systems must cooperate to achieve the overall task of reaching the goal landmark position. The Pilot needs the Vision system to identify obstacles, and it needs the Navigation system to select a path to the goal. The Vision system needs the Pilot to move the robot in order to identify visually-stable landmarks. And the Navigation system needs both the Vision and Pilot systems in order to identify landmarks and move the robot to the goal. Second, the systems are also competing—there are some tradeoffs between them. For example, the Pilot and the Navigation system both compete for the Vision system. The Pilot needs vision for obstacle avoidance, while the Navigation system needs vision for landmark detection and tracking. Similarly, the Vision and Navigation systems both compete for the Pilot. Vision needs the Pilot to execute motions to help identify stable landmarks, while Navigation needs the Pilot to move the robot toward the goal.

To manage this cooperation and competition, we have chosen a bidding mechanism. Each system generates bids for the services offered by the Pilot and Vision systems. The service actually executed by each system depends on the winning bid at each point in time. We have manually written the bidding functions in order to obtain good performance from the combined system.

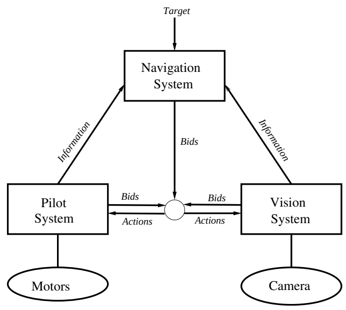

Figure 1: Robot architecture

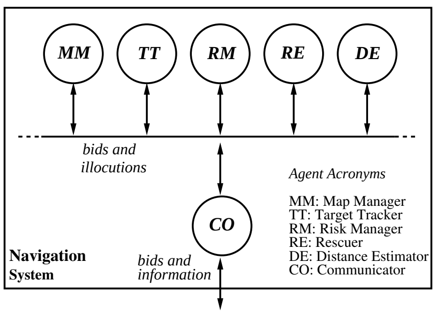

Figure 2: Original navigation system

The Navigation system itself is also implemented as a multiagent system. This system is composed of six agents with the following responsibilities: keep the target located with maximum precision and reach it (*Target Tracker*), keep the risk of losing the target low (*Risk Manager*), recover from blocked situations (*Rescuer*), keep the error in the distance to landmarks low (*Distance Estimator*), and keep the information on the map consistent and up-to-date (*Map Manager*). There is an additional agent, *Communicator*, which manages the communication between the Navigation system with the robot's other systems. As with the overall system, the Navigation system employs a bidding mechanism to coordinate these agents. Each agent bids for the action it wants the robot to perform. These bids are sent to the *Communicator* agent, which determines the winning action. The selected action is then sent as the Navigation system's bid for the services of the Vision and Pilot systems. Each action can involve a combination of requests to the Vision and the Pilot systems. The multiagent Navigation system is depicted in Figure 2.

For map representation and wayfinding, we employ an extension of Prescott's beta-coefficients system [14]. Prescott's model stores the relationships among the landmarks in the environment to build a map. The location of a landmark is encoded based on the relative locations (headings and distances) of three other landmarks. This relationship is unique and invariant to viewpoint. Once this relationship has been stored, the location of each landmark can be computed from the locations of the three landmarks encoding it, no matter where the robot is located as long as the robot can compute the heading and distance to each of the three landmarks.

As the robot explores the environment, it stores the relationships among the landmarks it sees. This creates a network of relationships among the landmarks in the environment. If this network is sufficiently-richly connected, then it provides a computational map of the environment. Given the headings and distances to a subset of currently-visible landmarks, the network can be employed to compute the locations of all of the remaining landmarks, even if they are currently not visible from the robot.

If there are landmarks associated with all known obstacles in the environment, then this network can also be employed to plan a path from the current location of the robot to the goal location. Because of obstacles, this path may not be a straight line from the current location to the goal. Instead, it typically consists of a sequence of intermediate targets, called diverting targets, such that if the robot travels from one diverting target to the next, it will eventually reach the goal.

Prescott's model assumes that the robot is able to measure the exact location of the landmarks in the environment. But this is not the case in our robot: the vision system gives only imprecise information about the location of the landmarks, and we cannot rely on the odometry of the robot, as it is also imprecise. To deal with this unavoidable imprecision, our extended model represents all of the network coordinates as fuzzy numbers and carries out all map computations using fuzzy arithmetic [4].

Each of the agents within the Navigation system has a bidding function that is controlled by a set of internal parameters. These parameters need to be tuned in order to achieve the best performance of the Navigation subsystem and of the overall system. Adjusting these parameters manually can be very difficult, particularly because of the tradeoffs confronting the top-level agents. An alternative to manual tuning is to employ machine learning techniques, specifically reinforcement learning (RL) methods [17]. In this paper, we describe some experiments to test the feasibility of applying reinforcement learning within this multiagent system.

Ideally, we would like to apply reinforcement learning to tune all of the parameters of all of the agents in the system. However, this is a very difficult problem, and it is not clear that reinforcement learning is the best solution at all levels of the system. We have chosen instead to focus on a particular learning problem within the Navigation system. Reinforcement learning is most needed and most appropriate in cases where there is a complex, quantitative tradeoff between behaviors. In such cases, manual tuning is difficult, and the quantitative criterion of maximizing expected reward, which is the goal of reinforcement learning, permits us to represent the tradeoff nicely.

Within the Navigation system, such a tradeoff exists between the *Target Tracker* agent, the *Risk Manager*, and the *Distance Estimator*. The *Target Tracker* wants to know the exact heading and distance to the target at all times. This can be achieved by pointing the camera at the target and moving toward the target. The *Risk Manager* wants to ensure that the robot is surrounded by a rich network of landmarks so that the robot does not get lost. This can be achieved by pointing the camera in various directions around the robot to identify and track landmarks. The *Distance Estimator* seeks to know accurate distances to each landmark. This can be achieved by pointing the camera at a landmark while moving the robot orthogonally to the direction of the landmark. In addition to this conflict between these agents, the Navigation system also must not monopolize the camera, because the Pilot needs to use the camera for obstacle avoidance as well.

In this paper, we propose to replace the *Target Tracker*, the *Risk*

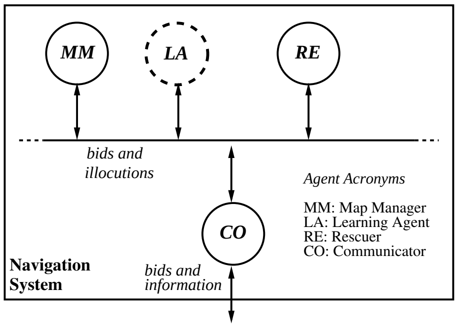

Figure 3: Modified navigation system, with the new agent

Manager, and the *Distance Estimator* with a new *Learning Agent* whose bidding function (known in reinforcement learning as the value function) will be tuned by reinforcement learning. We will formulate the reward function for this agent so that it is rewarded for reaching the current target location while minimizing the use of the camera. The two remaining agents have very different roles. The *Map Manager* maintains the beta-coefficient map, but does not bid on actions. The only remaining active agent is the *Rescuer*, which is responsible for the higher-level choice of diverting targets whenever the robot becomes blocked. This activity is better-implemented by path planning algorithms than by reinforcement learning, so we have not included the *Rescuer*'s responsibilities within the *Learning Agent*. The new architecture for the Navigation system is shown in Figure 3.

The rest of the paper describes our experiments with this learning agent, and it is organized as follows. Section 2 is devoted to relevant related work. Section 3 describes the task to be learned by the learning agent. The details of the reinforcement learning algorithm we have used are explained in Section 4. The experiments are presented in Section 5. Finally, the paper concludes with Section 6.

2. RELATED WORK

Since Brooks proposed the subsumption architecture [5], many other coordination mechanisms for robotic systems have been proposed, such as those by Maes [12] and Arkin [1]. Regarding multi-agent architectures, Liscano et al [9], Isik [10], and Stentz [16], among others, use hierarchical centralized architectures with arbitration to decide which activity takes control of the robot. Our approach, however, is completely decentralized, which means that the broadcast of information is not hierarchical. This approach is easier to program and is more flexible and extensible than centralized approaches. Furthermore, we propose a model for cooperation and competition between activities based on a simple bidding mechanism. A similar model was proposed by Rosenblatt [15] in CMU's DAMN project, in which voting was used to coordinate a set of modules to control the robot.

The map building approach used in our model is based on previous work by Prescott, who proposed a network model (the beta-coefficient coding system) that stores the spatial relationships among landmarks for robot navigation. By matching a perceived landmark with the network, the robot can find its way to a target provided it is represented in the network. Whereas Prescott's approach is quantitative, ours uses a fuzzy extension of the beta-coefficient coding system in order to work with fuzzy qualitative information about distances and directions. Levitt and Lawton [11] also proposed a qualitative approach to the navigation problem, but assume unrealistically accurate distance and direction information between the robot and the landmarks. Another qualitative method for robot navigation was proposed by Escrig and Toledo [8], using constraint logic. However, they assume that the robot has some a priori knowledge of the spatial relationship of the landmarks, whereas our system builds these relationships while exploring the environment.

3. THE TASK TO BE LEARNED

The task confronting the *Learning Agent* is to choose actions (for both motion and vision) in order to reach the current target location while minimizing the use of the camera. The current target location is determined by the *Rescuer*, and the *Map Manager* provides the map information needed by the *Learning Agent*. If the robot becomes blocked, the *Rescuer* will choose a new target (a diverting target), and then the *Learning Agent* will take control and choose actions to reach that new target. Once the diverting target is reached, the *Rescuer* may be able to set the current target to be the original goal, and then the *Learning Agent* will attempt to move to that target (and hence, solve the original task).

4. THE RL ALGORITHM

There are two general styles of reinforcement learning algorithms: Model-based and Model-free. Model-based algorithms learn a transition model $P(s'|s,a)$ for the environment, where s is the state of the environment at time t, a is an action to be executed, and s' is the resulting state of the environment at time $t+1$. Model-based algorithms also learn a reward model $R(s,a,s')$ which gives the expected one-step reward of performing action a in state s and making a transition to state s'. Once these models have been learned, then dynamic programming algorithms [3] can be applied to compute the optimal value function V^* and the optimal policy π^* for choosing actions.

In contrast, model-free methods (such as Q learning and SARSA(λ)) directly learn a value function V^* by repeatedly interacting with the environment without first learning transition or reward models. They rely on the environment to "model itself". For robot learning, however, model-free methods are impractical, because they require many more interactions with the environment to obtain good results. They make sense in simulated worlds where the cost of performing an action can be much less than the cost of storing the transition and reward models, particularly if the environment is evolving over time. But the cost of performing an experimental action with a real robot is very high.

Hence, for our experiments, we have chosen the model-based algorithm known as Prioritized Sweeping [13]. Prioritized Sweeping works as follows. In each time step, the learner observes the state s of the environment, chooses an action a, performs the action, receives a one-step reward r, and observes the resulting state s'. The learner then updates its estimate of $P(s'|s,a)$ and of $R(s,a,s')$ using the observed result state s' and the observed reward r. Finally, the learner performs the k most important Bellman backups to update its estimate of the value function V. A Bellman backup in state s is computed as follows:

$$V(s) := \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + V(s')].$$

This is essentially a one-step lookahead that considers all possible actions a and all possible resulting states s', computes the expected backed-up value of each a, and assigns the maximum such value to be the new estimate of V at state s.

Prioritized Sweeping maintains a maximizing priority queue of states in which it believes a Bellman backup should be performed. First, it performs a Bellman backup for the most recent state s. In each Bellman backup, it computes the change in the value $V(s)$ resulting from the backup:

$$\Delta(s) = \left| V(s) - \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + V(s')] \right|.$$

After performing the Bellman backup, Prioritized Sweeping considers all states s^- that are known predecessors of s, and computes the potential impact C of the change in $V(s)$ on the change in the value of s^- according to

$$C(s^-) = \sum_a P(s|s^-,a)\Delta(s).$$

It then places the state s^- on the priority queue with priority $C(s^-)$. Finally, Prioritized Sweeping performs $k-1$ iterations in which it pops off the state with the maximum potential impact, performs a Bellman backup in that state, and then computes the potential impact of that backup on all predecessor states. In our experiments, $k = 5$. (In our implementation, we actually use the state-action, or Q, representation of the value function rather than the state value function V. We have described the method using V in order to simplify the presentation.)

Prioritized Sweeping is essentially an incremental form of value iteration, in which the most important updates are performed first. Because every interaction with the environment is applied to update the model, Prioritized Sweeping makes maximum use of all of its experience with the environment. Prioritized Sweeping is an "off-policy" learning algorithm. During the learning process, any exploration policy can be employed to choose actions to execute. If the exploration policy guarantees to choose every action in every state several times, then Prioritized Sweeping will converge to the optimal action-selection policy. We employ ϵ-greedy exploration. In this form of exploration, when the robot reaches state s, it executes a random action with probability ϵ. With probability $1-\epsilon$, it executes the action that is believed to be optimal (according to the current value function V). Ties are broken randomly.

We represent both the transition model $P(s'|s,a)$ and the reward model $R(s,a,s')$ by three-dimensional matrices with one cell for each combination of s, s', and a. This technique will only work if the state and action spaces are small. There are two reasons for this. First, the tables must fit into memory. Second, the time required for learning is proportional to the number of cells in these tables, because the *Learning Agent* must experience multiple visits to each state s so that it can perform each action a several times and gather enough data to estimate $P(s'|s,a)$ and $R(s,a,s')$. Hence, the most challenging aspect of applying reinforcement learning is the proper design of the state representation.

4.1 State Representation

We want the *Learning Agent* to learn a general policy that works for any environment, independent of the locations of the landmarks and targets. Hence, our state representation must not directly employ the locations of the landmarks. Moreover, the robot cannot directly observe the complete state of the environment, which would include the location of the robot, all obstacles, and all landmarks! Instead, the task of the robot is to learn under conditions of incomplete knowledge about the locations of obstacles, landmarks, and targets.

State spaces that encode incomplete knowledge are known as "belief state spaces" [7]. The purpose of a belief state representation is to capture the current *state of knowledge* of the agent, rather than the current state of the external world. In our case, the learning agent is trying to move from a starting belief state in which it knows nothing to a goal belief state in which it is confident that it is located at the target state. Along the way, it seeks to avoid getting lost (which is a belief state in which it does not know its location relative to the target position).

To explain our state representation, we begin by defining a set of belief state variables. Then we explain how these are discretized to provide a small set of features each taking on a small set of values so that $P(s'|s,a)$ and $R(s,a,s')$ can be represented with small tables.

At any given point in time, the headings to all objects (landmarks and the target position) are divided into six sectors. The field of view of the robot is 60 degrees, so at any point in time, the robot can observe one sector, see Figure 4. For each sector, we represent information about the number of landmarks believed to be in that sector and the precision of our beliefs about their headings and distances.

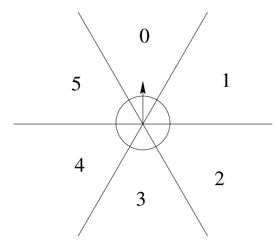

Figure 4: Division of environment in sectors. The arrow shows the direction in which the robot is facing.

Given these sectors, the following state variables can be defined:

- Distance to target, and its imprecision, $D(t), I_d(t)$
- Heading to target, and its imprecision, $H(t), I_h(t)$
- The landmarks in each sector, $L(s) = \{l_1, ..., l_{n_s}\}$
- Number of landmarks in each sector, $N(s) = \min(4, |L(s)|)$
- Average imprecision of landmarks in each sector, $\overline{I}(s) = \frac{1}{N(s)} \sum_{l \in Best(4,L(s))} I(l)$

We now explain each of these. The distance $D(l)$ to a landmark (or $D(t)$ to the target) is a fuzzy number in the range $[0,\infty]$. The heading to a landmark $H(l)$ (or $H(t)$ to the target) is a fuzzy number with range $[0, 2\pi]$. For each of these, the imprecision ($I_d(l)$ for distance, $I_h(l)$ for heading) is defined by taking the interval corresponding to the 70% α-cut of the fuzzy number (see Figure5).

The imprecision of a landmark is computed by combining the imprecision in the heading and in the distance as follows. First, $I_h(l)$ is normalized by dividing by its maximum value of 2π. Second, $I_d(l)$ is normalized by applying the hyperbolic tangent function, which maps it into the $[0, 1]$ interval. Finally, the two imprecisions are combined according to:

$$I(l) = \lambda \cdot \tanh(\beta \cdot I_d(l)) + (1-\lambda) \cdot \frac{I_h(l)}{2\pi}$$

where λ weighs the relative importance of the two imprecisions, and β controls how quickly the transformed I_d approaches 1. In our experiments, we set $\beta = 1$ and $\lambda = 0.2$.

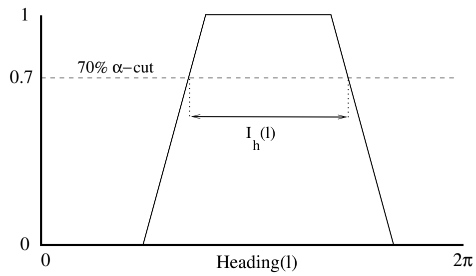

Figure 5: Computation of imprecision from a fuzzy number

We summarize the agent's knowledge of the landmarks in each sector by averaging the imprecision of the four most-precisely-known landmarks. The function $Best : N \times 2^L \rightarrow 2^L$ selects a subset, $B = Best(n, L)$, of a group of landmarks, $L = \{l_1, ..., l_m\}$, such that $|B| \leq n \wedge \forall_{l \in B} \forall_{l' \in L-B} I(l) \leq I(l')$. Having 4 landmarks in one sector is already very good, since only 3 landmarks are needed to use the beta-coefficient system network. Furthermore, we do not want these measures to be affected by bad landmarks when we have some that are good enough. That is why we use $Best(4, L(s))$ when computing $\overline{I}(s)$.

4.2 Features

After computing these state variables, we discretize them to define a small number of features each of which takes on a small number of values. We employ the following features:

- Target Distance, $D(t)$, discretized to 5 intervals.
- Target Location Imprecision: measure of imprecision on the location of the target, $I(t)$, discretized to 7 intervals.
- Landmark Count: average number of landmarks over the six sectors, $\overline{C} = \frac{1}{6}\sum_{s=0}^{5} N(s)$, discretized to 4 intervals.
- Landmark Imprecision: average imprecision of landmarks' locations in each sector, $\overline{I} = \frac{1}{6}\sum_{s=0}^{5} \overline{I}(s)$, discretized to 7 intervals.

This gives a total of 980 belief states.

4.3 Actions

Just as reinforcement learning requires careful design of the state space to ensure that it is compact, it also requires careful design of the action set to ensure that it is small but also sufficient for the robot to achieve its goals.

Physically, the robot is able to simultaneously perform two types of actions: *moving* actions and *looking* actions. Moving actions make the robot move in a given direction. Looking actions employ the camera to identify or track landmarks in the environment in specified sectors. The vision system can either search for new landmarks or re-acquire already-detected landmarks, but it is not able to do both things at the same time, because different image processing routines are required for each. In either case, however, the vision system returns the heading and distance to the landmarks it detects.

An additional constraint on the design of actions is that the vision system is most effective when the robot is moving in certain directions relative to the landmarks being observed.

Given these constraints, we have designed the following set of actions for the *Learning Agent*:

- Move Blind (MB): move toward the target (i.e., in the direction in which the target is *believed* to lie). Do not use the vision system.
- Move and Look for Landmarks (MLL): move toward the target. Point the camera in the sector that contains the fewest number of known landmarks, and look for new landmarks in this sector.
- Move Orthogonally to Target (MOT): move orthogonally to the direction of the target. Point the camera at the target and attempt to improve the precision of the heading and distance to the target.
- Move and Verify Landmarks (MVL): move toward the target. Point the camera to the sector with the maximum imprecision, $\overline{I}$, and attempt to re-acquire known landmarks and measure their heading and distance more accurately.
- Move and Verify Target (MVT): move toward the target. Point the camera at the target and attempt to re-acquire it and measure its heading and distance more accurately.

These actions should affect the state variables as follows. All actions except MOT will make the distance to the target decrease. MB will make all imprecisions grow. MLL should increase the number of detected landmarks. MOT should reduce the imprecision about the target's location, while MVL should reduce the overall imprecision. MVT will also reduce the imprecision of the target's location, but not as much as MOT. All actions require that the heading to the target is known (at least approximately). The heading is choosen as the center of the fuzzy interval for $H(t)$. If the heading is completely unknown, the center of this interval will be π. This causes the robot "pace" back and forth, turning 180 degrees (π radians) each time an action is executed.

We have assigned an immediate reward to each action to reflect the load on the vision system and the motion system. The rewards are negative, because they are costs. MB is the cheapest action, since it does not use the camera. It has a reward of -1. MVL and MVT produce a reward of -5, since they make moderate demands on the vision system. MOT gives a reward of -6, because it requires more motion in addition to the same image processing as MVL and MVT. Finally, MLL is the most expensive, with a reward of -10, because it must do extensive image processing to search for new landmarks and verify that they are robust to changes in viewpoint.

The system receives a reward of 0 when it reaches the target location. The reinforcement learning objective is to maximize the total reward. In this case, this is equivalent to minimizing the total cost of the actions taken to reach the target.

5. EXPERIMENTATION

We employ the Webots[1] simulator to perform our experiments. Figure 6 shows a picture of the simulated environment. This environment contains a set of landmarks (shown as black spots), one of

[1]Webots simulator, by Cyberbotics, www.cyberbotics.com

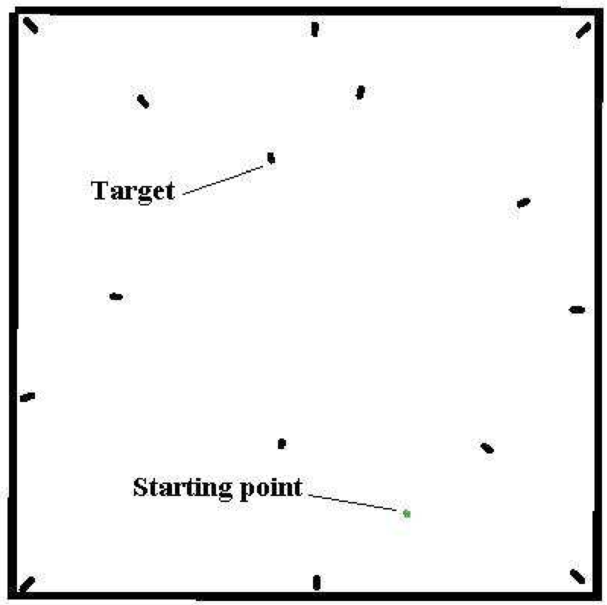

Figure 6: A picture of the simulated robot environment

which is designated as the target. There is also a wall that surrounds the region in which the robot is navigating. The robot is located at the point indicated by the "starting point". The landmarks are the only objects in the environment. There are no obstacles, as obstacle avoidance is handled by the Pilot system. However, the robot can be blocked by the landmarks or by the wall. In each trial, the robot starts at a random location in this environment, and it has to reach the target. The trial terminates under three conditions: (a) if the robot reaches the target (and is confident that it has reached the target), (b) if the robot takes 500 steps without reaching the target, or (c) if the robot is blocked. When the trial is finished, the next one begins with another random initial location for the robot.

5.1 Results

In order to see if the performance of the system improves after learning, we compare it with a hand-coded policy. The hand-coded policy uses the same discretized features as the learning algorithm:

```
if (Target Distance is high) then
    if (Landmark Count is low) then
        MLL
    else if (Landmark Imprecision is high) then
        MVL
    else
        if (Target Location Imprecision is high) then
            MOT
        else
            MB
        endif
    endif
else
    if (Target Location Imprecision is high) then
        if (Landmark Imprecision is high) then
            MVL
        else
            MVT
        endif
    else
        if (Target Distance is very low) then
            MVT
        else
            MB
        endif
    endif
endif
```

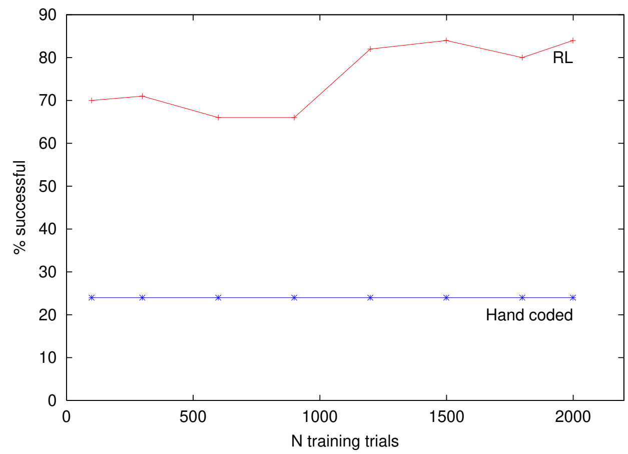

Figure 7: Number of successful test trials as a function of the amount of training.

where $high$, low and $very\ low$ are defined as follows:

Variable	Very Low	Low	High
Target Distance	< 1	≤ 2	> 2
Target Location Imprecision	–	< 5	≥ 5
Landmark Count	–	< 2	≥ 2
Landmark Imprecision	–	< 5	≥ 5

The reader should note that this hand-coded policy is not the same as the policy produced by the hand-coded bidding functions previously reported [6]. We have chosen this policy because it allows us to debug and test the *Learning Agent* separately from the rest of the multi-agent system.

The *Learning Agent* was trained for 2000 simulated trials. At regular intervals, the learned value function was tested by placing the robot in 99 randomly-chosen starting locations, running one trial from each location, and measuring the total reward, the total number of actions, and whether the robot succeeded in reaching the target position. The same set of 99 starting locations was employed in each testing period. The hand-coded policy was also evaluated on these 99 starting locations.

First, let us consider the fraction of successful trials. Figure 7 shows that even after only 99 trials, the *Learning Agent* is already out-performing the hand-coded policy. After 2000 trials, the *Learning Agent* succeeds in reaching the target in 84 of the trials, compared to only 24 for the hand-coded policy.

A second way of analyzing the performance of the *Learning Agent* is to compute the average reward per trial, the number of actions per trial, and the number of actions of each type. Table 1 displays this information after 2000 training trials. Each value is averaged

Table 1: Comparison of the *Learning Agent* (LA) and the hand-coded policy (HC) after 2000 training trials.

	Reward per trial	Actions per trial	MB	MOT	MVT	MVL	MLL
HC	–858	153.33	4.94	18.59	0.52	121.96	7.32
LA	–336	49.95	11.41	6.52	5.61	4.97	21.43

over five test runs. The only difference between test runs is the random number seed for the Webots simulator. We see that while the hand-coded policy receives an average of -858 units of reward, the learned policy only receives -336 units, which is a huge improvement. In addition, the *Learning Agent* on the average only requires 50 steps to terminate a trial (reach the goal, become blocked, or execute 500 steps) compared to 153 steps for the hand-coded policy. The *Learning Agent* never terminates because of reaching the 500-step limit.

Table 1 contains other interesting information. In particular, we see that the *Learning Agent* has learned to perform fewer MOT and MVL actions and more MB, MVT, and MLL actions. Note particularly that the learning agent is executing an average of 11.4 MB (Move Blind) actions per trial, compared to only 4.9 for the hand-coded policy. One of the goals of applying reinforcement learning was to find a policy that freed the camera for use by the low-level obstacle avoidance routines, and this is exactly what has happened. On the other hand, we were surprised to see that the *Learning Agent* chooses to execute the most expensive action, MLL, so often (21.4 times per trial, compared to only 7.3 times per trial for the hand-coded policy). Evidentally, it has found that a mix of MLL and MB gives better reward than the combination of MVL and MOT that is produced by the hand-coded policy. The *Learning Agent* spends much more time looking for new landmarks and much less time verifying the direction and distance to known landmarks.

6. CONCLUSIONS AND FUTURE WORK

The long-term goal of our research is to apply reinforcement learning to automatically learn bidding functions in a multi-agent system. In this paper, we have taken a step toward this goal by showing that reinforcement learning can learn to select actions to resolve a complex tradeoff that arises in our multi-agent robot control system. The results of our experiments show that the *Learning Agent* achieves a higher rate of success than the hand-coded policy in the Navigation multi-agent system. Furthermore, the *Learning Agent* accomplishes this while performing many fewer actions and while performing a different mix of actions. This underlines the value of employing reinforcement learning rather than attempting to resolve complex tradeoffs by hand-coding.

The next step in our work will be to integrate the *Learning Agent* into the overall multi-agent system and compare the performance of the *Learning Agent* with the performance of the hand-coded bidding functions described in our previous work [6]. This will require that we find a way to scale the learned value function so that it provides appropriate bids within the MAS. We may also find that the *Learning Agent* should place a higher or lower cost on using the camera than we employed in the experiments described here. We can easily change the reward function to reflect increased or decreased costs and then retrain the *Learning Agent*.

Additional future work will also test the robustness of these results to changes in the set of features and the set of actions. The current sets of features and actions reflects lessons from three previous design iterations. A challenge was to design features and actions that were well-matched in the sense that the actions should frequently lead to a change in the feature values so that reinforcement learning can detect progress in moving toward (or away from) the goal. If the actions operate at a finer grain than the features can represent, most actions appear to leave the state unchanged, and learning becomes impossible. This has been termed the "state-action deviation problem" [2]. It can be fixed by using more levels of discrete values for each feature, but this slows learning by requiring that the robot perform more interactions with the environment. Asada et al. suggest another solution in which self-loops (cases where the state is unchanged) are ignored in the transition model. We plan to evaluate the suitability of this approach in future experiments.

In designing the current set of actions, we sought to minimize the number of actions in order to keep the branching factor of the exploratory search small. However, a consequence of this is that most of the actions have "built-in" decisions. For example, MLL always looks for new landmarks in the sector with the fewest landmarks, and this might not always be best. However, we did not want to introduce six different "look-for-landmarks" actions, one for each sector, because this would increase the branching factor and make the initial exploration very slow. Nonetheless, we would like to study whether it might be possible to introduce such finer-grained actions after the *Learning Agent* has found a good initial policy. This might allow the learner to improve its performance further. This will be another focus of future work.

In conclusion, reinforcement learning is a promising method for tuning the behavior of agents in a multi-agent system. However, further research is needed to develop an engineering methodology for designing state and action representations that lead to successful reinforcement learning without the need for many design iterations.

7. ACKNOWLEDGEMENTS

This research has been supported by the Fulbright Joint Research Project and Plan Nacional Project DPI 2000-1352-C02-02. Dídac Busquets holds the CIRIT doctoral scholarship 2000FI-00191.

8. REFERENCES

[1] R.C. Arkin. Motor schema-based mobile robot navigation. *Int. J. Robotics research*, 8(4):92–112, 1989.

[2] Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 23:279–303, 1996.

[3] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, Belmont, MA, 1996.

[4] G. Bojadziev and M. Bojadziev. *Fuzzy sets, fuzzy logic, applications*, volume 5 of *Advances in Fuzzy Systems*. World Scientific, 1995.

[5] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, 1986.

[6] R. López de Màntaras C. Sierra and D. Busquets. Multiagent bidding mechanisms for robot qualitative navigation. In *Intelligent Agents VII. Lectures Notes in Artificial Intelligencee (Proceedings of ATAL-2000)*, pages 198–212. Springer, Verlag, 2001.

[7] Anthony R. Cassandra, Leslie Pack Kaelbling, and Michael L. Littman. Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 1023–1028, Cambridge, MA, 1994. AAAI Press/MIT Press.

[8] M. Teresa Escrig and F. Toledo. Autonomous robot navigation using human spatial concepts. *Int. Journal of Intelligent Systems*, 15:165–196, 2000.

[9] R. Liscano et al. Using a blackboard to integrate multiple activities and achieve strategic reasoning for mobile-robot navigation. *IEEE Expert*, 10(2):24–36, 1995.

[10] C. Isik and A.M. Meystel. Pilot level of a hierarchical controller for an unmanned mobile robot. *IEEE J. Robotics and Automation*, 4(3):242–255, 1988.

[11] T.S. Levitt and D.T. Lawton. Qualitative navigation for mobile robots. *Artificial Intelligence Journal*, 44:305–360, 1990.

[12] P. Maes. The dynamics of action selection. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI'89)*, pages 991–997, 1989.

[13] Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103, 1993.

[14] T.J. Prescott. Spatial representation for navigation in animats. *Adaptive Behavior*, 4(2):85–125, 1996.

[15] Julio Rosenblatt. Damn: A distributed architecture for mobile navigation. In *Proceedings of the 1995 AAAI SpringSymposium on Lessons Learned from Implemented Software Architectures for Physical Agents*. AAAI Press, March 1995.

[16] A. Stentz. The codger system for mobile robot navigation. In C.E. Thorpe, editor, *Vision and Navigation, the Carnegie Mellon Navlab*, pages 187–201, Boston, 1990. Kluwer Academic Pub.

[17] Rich Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, 1998.