Predictive Memory for an Inaccessible Environment

Mike Bowling mhb@cs.cmu.edu | Peter Stone pstone@cs.cmu.edu | Manuela Veloso veloso@cs.cmu.edu

Computer Science Department
Carnegie Mellon University
Pittsburgh PA 15213-3890

Abstract

Inaccessible and nondeterministic environments are very common in real-world problems. One of the difficulties in these environments is representing the knowledge about the unknown aspects of the state. We present a solution to this problem for the robotic soccer domain, an inaccessible and nondeterministic environment. We developed a predictive memory model that builds a probabilistic representation of the state based on past observations. By making the right assumptions, an effective model can be created that can store and update knowledge for even the inaccessible parts of the environment. Experiments were conducted to compare the effectiveness of our approach with a simpler approach, which ignored the inaccessible parts of the environment. The experiments consisted of using the memory models in a situation of a free ball, where two players are racing after the ball to be the first to pass it or kick it to one of their teammates or the goal. The results obtained demonstrate that this predictive approach does generate an effective memory model, which outperforms a non-predictive model.

1 Introduction

Real world systems have to contend with many complications which make it difficult to apply standard artificial intelligence techniques. These include an inaccessible environment, where the sensors are limited in what information they provide about the current state. Another complication is noise in the effectors of the agent. This makes the environment nondeterministic since the agent isn't sure of the effects of its actions.

Both of these complications are illustrated in the robotic soccer domain. In this domain the agents are players working cooperatively with their teammates to defeat their opponents. The agents in this domain have limited sensory information, which is given only for a limited view angle. Also, due to the nature of the game, there are unpredictable bounces and slips, which make the domain nondeterministic.

One aspect of the agent that is needed to overcome these complications in both the soccer domain and real world systems is an accurate memory model. In this paper we describe a model for the robotic soccer domain, which uses past sensory input and a probabilistic approach to maintain the model for even the inaccessible parts of the environment. Although players and the ball can move unpredictably when they are not in view, our memory model maintains reasonable estimates of their locations relative to the agent. Stationary objects, such as the goals, can be located much more precisely, even when the agent is not looking at them. Our memory model uses the locations of visible stationary objects to help determine the positions of those not in view.

The model is demonstrated using a soccer simulation program, which is explained in the following section. This is followed by a description of the memory model and the results of testing this model against a simpler one. The purpose of this paper is to allow others currently using or planning to use the Soccer Server system to duplicate our memory model for their own use.

2 Simulator

The memory model was created for use by client programs for Noda's soccer server [Noda, 1995]. The server provides a virtual field and simulates the motion of two teams of players and the ball. The players are controlled by separate client programs which connect to the server using UDP/IP sockets. The simulator also provides an interface for a coach, which can move and monitor the players to provide a facility for testing. This paper discusses the details of the client's interface to the server, since this determines the inputs to the memory model and the actions the client has available. The model described in this paper is designed and tested using version 1.x of the simulator. A newer version of the simulator, 2.x, has been made available. The differences in the sensory information provided by the two versions are given in this section. Calculations necessary to extend the memory model are provided in the appendix.

2.1 Sensory Information

The simulator sends sensory information to the client about the portion of the state that is visible or audible. This information comes in two forms: visual and auditory. Auditory information is just a message that was spoken and from what direction the message came from. This provides little information about the current state since there's no simple way to determine who spoke the message.

Visual information is provided as a list of objects with their relative distance and angle from the client. Information is only provided for objects within a 60 degree arc from the direction the client is facing. Objects within this arc will be referred to as visible objects, because their exact position is given by the simulator. Objects that the simulator provides information about are the ball, other players, the two goals, the four flags at the corners of the field, and the flags on the sides at mid-field. For players, the amount of information describing which player varies on how far the player is away. If the player is close then both the team and number is provided. If the player is further away then only the team is provided, and if very far away then no information is provided.

This is a very limited amount of information, making this domain highly inaccessible. There are many situations when most of the current state is not available to the client. If there were no memory model at all, then just turning away from the an object would cause the client to completely forget where the object was. For example, if the client receives a pass and then turns to look for the goal, it no longer knows where the ball is and won't be able to shoot. A memory model is essential for overcoming the inaccessibility of the environment.

Version 2.x of the simulator provides additional information to the client. At each step, the distance to the boundary directly ahead, along with the relative angle of the client's facing to that boundary line, is provided to the client. This information is in addition to the information described above. The memory model outlined in this paper equally applies to the newer simulator. The appendix provides calculations which use this added information to improve the memory model.

2.2 Client's Actions

The simulator receives actions from the client which describe what the client is doing. The possible actions for a client are *turn*, *dash*, *kick* and *say*. The *say* command cause the simulator to send an auditory message to the other players with the details of the message. The other three commands provides the client with the basic actions of playing soccer.

The *turn* command takes a parameter, which is the amount of the turn, in the range of -180 to 180. The result of the action is to cause the player to rotate approximately that amount. If the client is moving at the time the amount of rotation is reduced. The *dash* command takes a power parameter in the range of -30 to 100, and causes the player to accelerate by the specified amount in the direction the client is facing. Since a *dash* causes an increase in velocity, the effects of dash might continue a number of time steps after it was sent. The final command is *kick*, which takes a power parameter between 0 and 100, and an angle between -180 and 180. If the ball is close enough to the player it causes the player to kick the ball with the specified power in the specified angle relative to the current direction the client is facing.

The design of the simulator provides many difficulties that a client will need to deal with. The first of these is that commands sent to the client or from the client may be lost. There is no guarantee that any commands that were sent were ever executed. A client has to verify whether his commands were executed from the sensory information that it receives. Another difficulty is introduced with noise. Turn angles and kick angles aren't perfect, and moving objects don't always move in a straight line. Again, the client can't know this noise and would have to observe it through the sensory information it receives.

3 Memory Model

3.1 Structure of the Model

The memory model concentrates on the objects. There are two different classes of objects in the soccer domain that must be handled differently. There are stationary objects whose global position do not change, and there are mobile objects whose global position do change. Both of these classes of objects need to be handled differently. Mobile objects can move unpredictably when not in view, so it is not always possible to accurately determine where they are if they are not in view. However, with the proper memory model, stationary objects can be located even when not in view.

With velocity estimations, a crude estimate of mobile objects' positions can be maintained. Since they have their own motion, changes in their relative position could be caused by either the client's motion or the motion of the object. These two causes will need to be separated in order to keep an accurate estimate of a mobile object's velocity.

For each object, an estimation of its position is stored and updated. The position consists of its relative polar coordinates, the same information that the simulator sends to the client for visible objects. Since this stored position is an estimate and we want it to apply to objects that aren't visible, we need to store something that describes the accuracy of the estimation. We do this by storing a probability, which is the confidence in the estimate. When the memory receives position information from the simulator, we assign it a confidence of one. If the object is not currently visible then it will have a confidence of less than one.

Audio sensory information complicates this slightly. Since audio information gives us information on an object's direction but not its distance, it is possible to have a different confidence in the object's direction than its distance. So we must store separate probabilities for direction and distance.

3.2 Updating the Model

We now assume that we have an accurate model of the state, and we've received some sensory input from the simulator for the next time step. We have to update our model to account for this input. We divide this update into two stages depending on what caused the changes to the model. First, we update the model for internal changes, those that are caused by the client (e.g. turning and dashing have a large impact on the model since the positions that are stored are relative to the client.) Second, we update the model for external changes, those that are not directly caused by the client (e.g. the motion of the ball and other clients.)

Internal Changes

In trying to estimate the changes to the model caused by the actions of the client, we want to avoid depending on the actions the client sent to the simulator. There is a number of reasons for this. First, we can't be sure the simulator ever received the commands. They might have gotten lost or delayed in the network. Second, the simulator is nondeterministic. It adds some amount of noise to the client's actions so that the client's commands don't have exact effects. Finally, dashes continue to affect the client a number of steps after the command was given. In order to account for this we would have to replicate the physics model that the simulator is using. But this unnecessarily ties the client's performance to a specific simulator model. Instead, we want the client to observe the effects of its own actions, and update its memory model based on the objects that are visible.

There are three possibilities of how the model is updated, depending on how many stationary objects are currently visible. If two or more objects are in view, then there's enough information to triangulate the client's current global position and angle and use that to determine the relative position of all stationary objects. Because this can be a costly calculation, this is only done if the overall confidence in the model is below some threshold. The calculations used to perform the triangulation is given in appendix A.1.

The other end of the spectrum is when there are no visible stationary objects. This situation is rare in the robotic soccer domain and only occurs when the client is very close to the field bounds. In this case, we have absolutely no information from the simulator as to the effects of our actions. Instead, we resort to a crude estimation of the client's movement from the actions it performed. We assume that turns were exact and ignore dashes, since their effects are difficult to estimate. This doesn't result in a very accurate estimation, but this case is rare, and it does give a good enough prediction for the client to recover.

The above two cases are the extremes, and the majority of the time only one stationary object will be visible. In this case, we don't have enough information to get an exact measurement from triangulation, but we do have enough information that we don't have to make uninformed guesses. We know how one object's position changed because of the client's actions. From this, we need to estimate the movement of the client and then apply this to the other objects that aren't visible.

Since we don't have enough information to determine the change exactly, we'll have to make some assumptions, which will allow us to make an accurate estimation. The first assumption is that the client will always turn before dashing. This assumption is a good one, since we can actually control this in the client's behavior. The second assumption is that all motion is in the direction the client is facing. It turns out that the simulator being used obeys this assumption. The result of these two assumptions is that the object's current facing is the same direction as its last movement. This now allows us to compute the rotation and translation of the client.

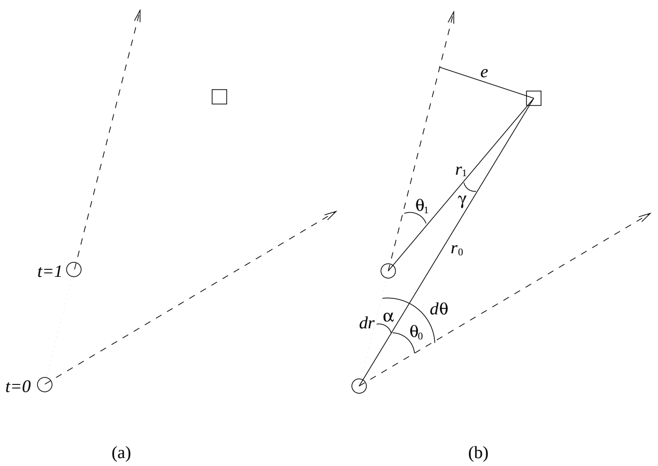


Figure 1: Estimating motion from the change in position of one object. (a) is an example position and facing for the client at two time steps, and the position of a marker to be used to calculate the client's rotation and translation. (b) this same diagram is labelled with the values used for the calculation.

From figure 1, we know the values θ_0, θ_1, r_0, and r_1. We want to calculate $d\theta$ and dr, which corresponds to the client's rotation and translation, respectively. By our assumptions, we know that the line segment, e, is shared by two triangles: one whose opposite vertex is the client at $t = 0$ and the other at $t = 1$. These assumptions allow the following calculation.

$$\begin{aligned} e &= r_1 \sin\theta_1 \\ \alpha &= \sin^{-1}\left(\frac{e}{r_0}\right) \\ \gamma &= \pi - ((\pi - \theta_1) + \alpha) = \theta_1 - \alpha \\ d\theta &= \alpha - \theta_0 \\ dr &= r_1 \frac{\sin\gamma}{\sin\alpha} \end{aligned}$$

The above calculation assumes that the object that we are using to estimate our movement has been visible for the previous two time steps. This assumption is not necessary. The same calculation can be carried out using the *estimation* for the position of the object in the previous time step. In this case there is a confidence in our estimation of the client's movement, which is just our confidence in the object's previous position.

Using this method of calculating the client's own rotation and translation, we can then update all other objects, including mobile objects, for the motion of the client. The calculations used to do this update is given in appendix A.2.

External Changes
We've described how the memory is updated for changes to the state caused by the client itself. There are also changes to the state caused by other players and momentum, which only affect the position of mobile objects

As mentioned above, the client can never be sure what a mobile object is doing when it is out of view. But in reality, when something moves out of our field of vision we can, for a period of time, be confident that the object is in the direction that it left our view. This is the kind of behavior that the memory model should have. In order to implement this, an estimation of the velocity of mobile objects needs to be stored. This is simply done by storing the difference in position between the current and previous time step. We factor out the change in relative position caused by the motion of the client itself using the calculation from the previous section. With an estimation of velocity for each object, we just assume that objects that aren't visible maintain the same velocity and update their position to account for it.

Overview
The procedure for updating the memory given some new sensory input is summarized below.

1. Determine client's own effect on the state. Method depends on the number of visible stationary objects.
   - None. Make a coarse guess based on the client's actions.
   - Two. Triangulate its global position on the field.
   - One. Estimate its effect on the state.
   (a) Use the change in the object's position to calculate the actual amount of rotation and translation.
   (b) Use the estimation to update the unseen objects in the model.
2. Update unseen mobile objects based on their last observed velocity.

4 Testing and Results

Two experiments were conducted to determine the effectiveness of this new memory model. The first experiment examined its advantages in the situation of a free ball, where two opposing players would race to gain control of a ball in the open field. The second experiment examined the accuracy of the model. In both experiments the predictive model was compared against a simple model that was being used previously. The tests were designed to verify that the memory model accurately maintains the positions of stationary objects relative to the client, even when they are out of view.

4.1 Simple Model

The memory model that was developed was compared with a very simple memory model, which ignores the the inaccessible aspects of the state. This memory stores the positions of all objects that are visible. In order for the model to be useful at all, the client must at least be able to turn away from an object (e.g. the ball) while looking for another object (e.g. the goal). This was done by adding the capability for the model to remember the position of objects when the client turns, by simply updating their position by the exact amount of the *turn*. If the client dashes then the positions of all the objects are forgotten and only the position of the newly visible objects are known. This is the simplest model for memory that provides enough functionality to be usable by the client.

4.2 Client Behavior

The behavior of the client for these tests was very simple. In essence, the client would chase after the ball, and upon reaching it would kick it at its goal. A low-level description of what the client would do at each time step is given below.

1. Query the memory for the position of the ball. If its not known then look for it.
2. If the ball is close enough to be kicked then:
   (a) Query the memory for the location of the goal. If its not known then look for it.
   (b) Kick the ball toward the goal.
3. If the ball is not close enough to be kicked then dash toward it.

Figure 2: The soccer field used by the simulator. The boxes mark out where the players and ball were placed for the free ball experiment.

4.3 Free Ball

The first test was to simply have two clients, one using the simple memory model and the other using the predictive model, compete for a free ball. The ball and players were independently placed in random positions on the field. The range of positions for the players and ball is shown graphically in figure 2. The distance of players to the ball ranged from about one fifth to a half of the length of the field. After placing the players, there was a few seconds delay to allow the players to face the ball. The players were then released and it was then recorded to which side of the field the ball was kicked. Since both players were trying to kick to opposite goals, this would measure which was more effective at gaining control of the loose ball.

It was expected that the predictive model would outperform the simpler approach, since the predictive model would not have to take the time to turn and face the goal, before shooting the ball. The results of 2500 trials is shown in table 4.4, and corresponds with our expectations.

4.4 Accuracy

A second test was conducted to verify that the predictive memory was keeping an accurate model of the field. This was done by allowing the client to perform the same loose ball trial, but without the opposing player. The client would then dash for the ball unhindered, and kick it at the goal. The accuracy of the shot was then measured as the distance of the ball as it crossed the goal line from the center of the goal. For this experiment, the noise factor in the motion of the ball was removed, so that the raw accuracy of the memory model could be measured. It was expected that the simple memory would be more accurate since it is facing the goal when it shoots, while the predictive memory won't have seen the goal since it started dashing towards the ball. The results, though, were surprisingly close. They are shown in table 4.4. The results are given as a fraction of the size of the goal mouth.

4.5 Results

From the results of these tests we conclude that the predictive memory model gives substantial gains in the client's awareness of the state of the environment. There is a tradeoff with accuracy, though, it is quite small. Testing was restricted to client behaviors that would work well for both memory models, in order to isolate the effects of the memory from the behavior. This restriction makes it difficult to show the real advantage of the predictive model, which is the possibilities for advanced behaviors.

5 Conclusion

The tests demonstrate that an effective and accurate memory model can be developed for inaccessible environments. The model uses a probabilistic approach and crucial assumptions to provide a predictive aspect to model the unseen parts of the state. The model considerably outperformed a simpler approach with only a small loss in accuracy. As mentioned before, even though this model was designed and tested using the older version of the simulator, equations are provided in the appendix that use the added information of the simulator. These equations, coupled with the descriptions throughout this paper, should allow other researchers to implement our memory model for use in the Soccer Server system.

Further research in this area could look at applying learning techniques to the predictive aspects of the model. The client could learn the effects of actions and apply it to situations where there is no information for prediction (i.e. situations when there are no markers visible.) Other research could focus on designing behaviors for the clients, which take advantage of the model.

Acknowledgements

This research is sponsored in part by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant number F33615-93-1-1330. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Wright Laboratory or the U. S. Government.

References

[Noda, 1995] Itsuki Noda. Soccer server : a simulator of RoboCup. In *Proceedings of AI symposium '95*, pages 29–34. Japanese Society for AI, December 1995.

A Auxiliary Calculations

A.1 Triangulation

$$
\begin{aligned}
(X_A, Y_A), (X_B, Y_B) &= \text{Global Cartesian coords. of two markers } (\mathit{known}) \\
(x_A, y_A), (x_B, y_B) &= \text{Relative Cartesian coords. of two markers } (\mathit{known}) \\
(r_A, \theta_A), (r_B, \theta_B) &= \text{Relative polar coords. of two markers } (\mathit{known}) \\
(X_C, Y_C) &= \text{Global Cartesian coords. of client } (\mathit{unknown})
\end{aligned}
$$

$$
\begin{aligned}
dx &= x_B - x_A \\
dy &= y_B - y_A \\
dX &= X_B - X_A \\
dY &= Y_B - Y_A \\
\theta &= \tan^{-1}\left(\frac{dy}{dx}\right) \\
\Theta &= \tan^{-1}\left(\frac{dY}{dX}\right) \\
\phi &= \theta + \Theta \\
X_C &= X_A + r_A \cos(\phi - \theta_A) \\
Y_C &= Y_A + r_A \sin(\phi - \theta_A)
\end{aligned}
$$

A.2 Update for Client's Actions

$$
\begin{aligned}
(r_0, \theta_0) &= \text{Old relative polar coordinates of object (known)} \\
(x_0, y_0) &= \text{Old relative Cartesian coordinates of object (known)} \\
(dr, d\theta) &= \text{Translation and rotation of client (known)} \\
(r_1, \theta_1) &= \text{New relative polar coordinates of object (known)}
\end{aligned}
$$

$$
\begin{aligned}
\theta_1 &= \theta_0 + d\theta \\
x_1 &= r_0 \sin \theta_1 \\
y_1 &= r_0 \cos \theta_1 + dr \\
r_1 &= \sqrt{x_1^2 + y_1^2}
\end{aligned}
$$

Results	Trials	Simple Memory	Predictive Memory
Free ball successes	2500	645	1855
Free ball percentage	2500	25.8%	74.2%
Accuracy (mean error)	1500	0.22	0.30

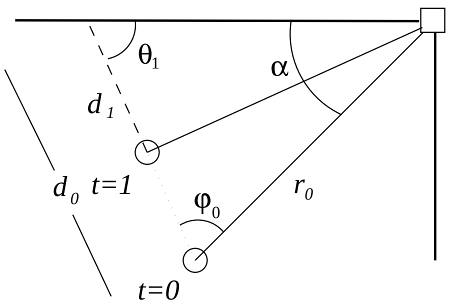
Table 1: The tabulated results. Accuracy is given as the fraction of the size of the goal mouth.

B Calculation for Version 2.x

Version 2.x of the Soccer Server provides some additional sensory input to the client. The client can always ignore the extra information and proceed as described above. However, it can use the additional sensory information to improve its memory model.

As sensory input, the client receives the relative angle of a boundary line. From this angle and the knowledge of which line, the global angle of the client can be computed and known exactly at any moment. This additional piece of information can be used to improve our model of the field. Using the same assumptions, we can now perform triangulation when only one marker is in view. Now the only case requiring estimation is when there are no markers in view.

When a marker is visible, the client's rotation and distance travelled can be calculated as follows. From the previous time step we know the relative position of the markers along with the distance and direction of the boundary line for the current and previous time step. Since we always can calculate global angle, the client's rotation is simple to compute, but distance travelled is more difficult. The diagram and associated calculation for distance is given below.



(r_0, ϕ_0)	$=$	Old relative polar coordinates of object on the boundary line (known)
θ_1	$=$	New relative angle of boundary line (known)
d_1	$=$	New distance to boundary line (known)
δ	$=$	Distance moved (unknown)

$$\begin{aligned} \alpha &= \pi - \theta_1 - \phi_0 \\ d_0 &= \frac{\sin \alpha}{\sin \theta_1} r_0 \\ \delta &= d_0 - d_1 \end{aligned}$$