Uppsala Underdogs — A Robot Soccer Project

Marianne Ahlford, Martin Andersson, Hanna Blomquist, Magnus Ekström,
Lars Ericsson, Johannes Gumbel, Anna Holmgren, Petter Holmberg,
Leonard Kia, Anton Lindström, Magnus Lyrberg, Shaman Mahmoudi,
Bengt Nolin, Jesper Sundien, Henrik Wallentin,
Olle Gällmo, Anders Hessel, Leonid Mokrushin, Paul Pettersson

Department of Information Technology, Uppsala University
P.O. Box 337, S-751 05 Uppsala, Sweden.
Email: {maah6328,maan2623,habl3140,maek5781,laer6919,
jogu2893,anho5110,peho6681,miki9353,anli6327,maly8595,
shma7099,beno9295,jesu3473,hewa4869}@student.uu.se,
{crwth,hessel,leom,paupet}@it.uu.se

Abstract. In this paper, we describe the four-legged soccer team Uppsala Underdogs developed by a group of 4*th* year computer science students at Uppsala University during the fall of 2004. The project is based on the experience from two similar previous projects. This year the emphasis of the project has been on distribution of data and on support for evaluation and reconfiguration of strategies. To support data distribution, a middleware has been developed, which implements a replication algorithm and provides a clean interface for the other software modules (or behaviors). To enable easy reconfiguration of strategies, an automata-based graphical description language has been developed, which can be compiled into code that uses the database and the lower level modules, such as tactics and positioning, to make decisions and control the robot. In addition, a graphical simulator has been developed in which the strategies can be evaluated.

1 Introduction

Team Uppala Underdogs is the result of a 4th year student project in computer science, running anually during the autumn semester at 75% of full time. The goal is to complete a large programming assignment with the Sony AIBO model ERS-210A for the four-legged robot league in RoboCup. This is the third time the project has focused on RoboCup soccer, but so far no student group have decided to continue the previous year's work. This is something we have attempted to change with a well-structured design and good documentation.

When constructing a robot soccer team there are a number of problems to consider:

- Positioning of the AIBO and other objects on the field
- Communication and data distribution
- Strategy and AI

To gather data for positioning, our main source is visual data from the AIBO camera. By analyzing the visual data, we are able to recognize known objects and position ourselves. We also rely on odometrical data collected during movement to continually update our estimated position. For a global estimation of the ball's position, we use information from the whole team to construct a mean value.

Communication is necessary for cooperation, and we handle communication through a distributed database containing data for all players. This allows every player to make a local decision based on global data, thus eliminating the need for a central decision-taking unit.

Having all robots sharing the same data gives a foundation for creating good strategies. We create strategies based on roles, describing actions. These roles are handed out during play and an AIBO's role may change depending on the state of the game. Roles and the role-deciding rules are all developed in a graphical user interface and read at run-time.

2 Preliminaries

2.1 RoboCup

The Sony Four-Legged Robot League is one of five leagues that currently form the RoboCupSoccer domain. In some leagues, the teams are required to design and build the robot hardware themselves. In the case of the Sony Four-Legged Robot League, all teams use the same robotic platform, manufactured by Sony. The robots operate autonomously, i.e. there is no external control, neither by humans nor by computers, during the game. Since all teams use the same hardware, the difference lies in the methods they devise to program the robots.

Each soccer team consists of four robots and a game consists of two 10-minute halves, separated by a 10-minute half-time break.

2.2 Tekkotsu

We decided not to use Sony's development platform OPEN-R [1] directly for development of our system, but to use the Tekkotsu [2] framework, developed at Carnegie Mellon University. By using a framework, we could speed up development and focus more on the computer science-related tasks we had at hand.

Tekkotsu is based on three concurrent processes: `MainObj`, `MotoObj` and `SoundPlay`. `SoundPlay` handles playback of digital sound on the AIBO, which is a feature we don't use. `MotoObj` handles the state of all joints of the AIBO, ensuring smooth movement of legs and head even during a high system load.

`MainObj` is where everything else is handled, and the heart of it is an event router. Programs made for the framework, called behaviors, are not able to run simultaneously since everything is in one system process, but switching between behaviors is possible with the event router. The Tekkotsu system and the behaviors generate events that behaviors listen for and act upon. Examples of events are when the AIBO camera has taken a new picture or when a button on the AIBO is pressed.

Since it is under development, not all things work as advertised, but overall it is a very useful framework which simplifies construction of the soccer team.

3 System Overview

On top of Tekkotsu we built our own system consisting of a number of behaviors, communicating via events. As stated above, none of these run in parallel but some of them are run when appropriate events happen. That leaves us without having to think about concurrent access to data.

3.1 Strategy

Strategies are designed as automata, using a language constructed for this purpose. We use TimesTool [3], a tool set featuring a graphical editor for timed automata, to develop our strategies. We chose this software because:

- It provides a nice graphical interface for creating and editing automata.
- It's free to use (through an educational software license).
- It came to our attention since it's developed at the Department of Information Technology at Uppsala University.

After creating the automata in TimesTool, they are saved in XML format. These XML files are then read by our Strategy behavior and translated into an internal representation that execute the given instructions.

We have chosen to design our strategies in this way mainly because the graphical representation of the automata gives a better overview than a text representation and because strategies can be changed easily without recompiling the system.

Strategy automata A minimum of three automata are required: *lineup*, *rolecaster* and at least one *role* automata.

The *lineup* is a description of which roles we want to use under different circumstances. The *rolecaster* tells each AIBO what role they have in the current game state. The *role* is the local strategy for each AIBO.

The XML file is parsed by the Strategy behavior, which sends orders to our Tactic behavior, which breaks down abstract instructions into a series of simpler instructions. When the Tactic behavior has done what is was ordered to do, it sends a message to the Strategy behavior that the order has been carried out. Data needed for strategic decisions is provided by our Database behavior and fetched when needed. This data includes the position of all AIBOs in the team and the global estimation of the ball's position. With this system it is possible to create very versatile strategies.

Current strategy Our current strategy consists four roles: *goalkeeper*, *defender*, *attacker* and *supporter*. All roles except *goalkeeper* are dynamically distributed among the other players depending on the current state. The role of the goalkeeper is fixed (in order to comply with the rules of the game) and no other AIBO can take it. In order to prevent all AIBOs from chasing the ball at the same time, ending up getting entangled with each other, only one of the players, the attacker, is allowed to approach the ball. The AIBO closest to the ball gets the *attacker* role. The attacker's main objective is to score goals, so it constantly tries to move towards the ball and shoot against the opponents' goal. The AIBO closest to its own goal gets the *defender* role. The defender positions itself between the ball and its own goal, on its own side of the field. The AIBO that does not fulfill the requirements for becoming either attacker or defender gets the *supporter* role. The supporter positions itself so that it is ready to take over the attacking role.

Supported elements in the language used to make strategies are:

- Predicates: *ballVisible*, *closestToBall*, *closeToGoal*, *closestToOwnGoal*, *timeoutReached*, *tacticDone*
- Actions: *kick*, *gotoBall*, *locateBall*, *makeSave*, *obstructBall*, *gotoPosition*
- Boolean operators: *AND*, *OR*, *NOT*
- Misc: *timeOut*

3.2 Tactic and Movement

Our Tactic and Movement behaviors provide the system with the compound movements and tactical decisions that the AIBO has to make in the field. These behaviors work closely together and supply functionality for carrying out the orders given by the Strategy behavior. Instructions are passed down in the following manner:

$Strategy \rightarrow Tactic \rightarrow Movement \rightarrow MotionManager$

At each module the orders are broken down into smaller steps suitable for the module at the level below. The Tactic behavior breaks down the abstract instructions given by the Strategy behavior into a series of simpler instructions that are sent to the Movement behavior. To execute the given commands, the Movement behavior communicates with MotionManager, which is a part of the Tekkotsu framework.

Whenever one of the behaviors has completed a task, an event is generated to notify the behavior giving the order that it has been carried out.

The main task of the Tactic behavior is to provide abstract functions for moving, locating the ball and kicking the ball. The Movement behavior provides the fundamental kinematics for the system. This is the walking functionality, the kicks and the head and tail movements.

Walking The Tactic behavior can be ordered to go to a specified position on the field. To accomplish this, the function estimates which walking style will get the AIBO to the destination in the shortest time and chooses that one. The functionality for walking is then provided by the Movement behavior, which supports walking specified with an arbitrary combination of forwards/backwards, sideways and rotational speeds. During the time the AIBO walks, odometrical data is sent to our Positioning module by the Movement behavior.

The Tactic behavior can also be told to go to the ball. A position near the ball such that the opponents' goal is on the opposite side, is then calculated. This will put the AIBO in a position where it is possible to make a good shot. It then uses the above functionality to get to this position.

Obstructing the ball is another functionality provided by the Tactic behavior. The position closest to the AIBO that lies between the ball and the own goal will then be calculated, whereupon the AIBO is then told to move.

Locating the ball When locating the ball, the AIBO rotates on the spot and turns the head back and forth in an attempt to spot the ball, regularly checking the database to know if the ball has been spotted. When it has, the AIBO stops its movements.

Kicking The Tactic behavior also provides a kicking functionality. Depending on the position of the AIBO relative to the goal of the opposing team, the appropriate kick is chosen from the kicks supplied by the Movement behavior. If the AIBO is standing at an angle in which it is deemed impossible to kick the ball towards the opponents' goal, no kick will be carried out.

3.3 Vision

Our Vision behavior implements image analysis for recognition and robot-relative positioning of objects on the field. The behavior uses the Tekkotsu vision pipeline (with small modifications) for low-level image analysis and replaces its high-level image analysis with more advanced algorithms.

The high-level image analysis is able to recognize and position field objects under lighting conditions well below the minimum requirements given in the 2004 RoboCup rules. It has been tested successfully below the required threshold of 500 lux with standard fluorescent ceiling lamps, using the color calibration tools provided by Tekkotsu. The lamps introduce a considerable amount of noise in the image due to the oscillation of the light, as compared to the more stable light of halogen lamps, commonly used in RoboCup soccer. This problem has been solved with extensions of the Tekkotsu vision pipeline.

Due to lab conditions it has also been impossible to surround the field with a uniformly colored wall, which introduces the possibility of objects outside the field similar to real field objects to be visible. The main strategy to avoid detecting such objects has been to estimate the height above the field of identified field objects and discard those that are not within some given accepted height limits. Identified field objects are balls, beacons, goalposts and other AIBOs.

Low-level image analysis Low-level image analysis is based on the Tekkotsu incorporation of the CMVision [4] package, developed as an undergraduate thesis by James Bruce at Carnegie Mellon University. CMVision translates the raw camera image into lists of color regions matching the calibrated colors. Every color region contains information about its properties, such as area, centroid and bounding box coordinates. The information has been extended for our purposes to also include information about the minimum and maximum coordinates of every scanline in the color region. This enables us to filter away some of the noise introduced by the lighting when doing high-level image analysis, and thereby getting more accurate results.

High-level image analysis The high-level image analysis analyzes identified color regions and tests them against a set of requirements that corresponds to properties of color regions belonging to field objects. Examples of such color region properties are size, shape and height above the field.

Using the properties of color regions identified as belonging to field objects, the camera-relative angle and distance to the field objects are calculated. The angles are then adjusted using the AIBO head angles, such that the calculated field object positions are relative to the AIBO body, with the origo located at the neck joint. Identified field objects are passed to the Positioning module, which uses them to estimate the absolute position of the AIBO on the field.

Field object recognition and positioning details Balls are identified by analyzing the largest orange color regions identified. The relative position of a non-occluded ball is calculated based on the width of the color region. Balls that are estimated not to be fully visible are positioned using radius calculations based on color region edge points, which are obtained through the extensions to CMVision.

Beacons are identified as two vertically adjacent color regions with approximately the same height/width proportions and sizes. First, the medians of the widths of the scanlines in the two color fields are calculated. Then, the distance is calculated using the mean of these two medians. This information is also obtained via the the extensions to CMVision and allows a considerably improved beacon positioning under poor lighting conditions and calibrations.

Goalposts are recognized by analyzing the largest color regions with the colors of the goals. The distance is calculated using the largest height of a specified part of the color region containing its left or right edge. The width of a goal color region is typically unusable for distance calculations, as goals are rarely empty and seen in their full width. This also makes it possible for one goal to be divided into multiple color regions, all which may be recognized as separate goals. It is up to the Positioning module to pick out the leftmost and rightmost goalposts out of all reported goalposts.

Other AIBOs are recognized by grouping together the largest color regions of the appriopriate colors with smaller color regions close to larger ones. The distance is calculated based on the area of the bounding box of such a group,

and is very error-prone due to the difficulty in recognizing other AIBOs properly. This information does not help the AIBO to position itself though and is only provided as a means for the AIBOs to detect temporary obstacles in the form of other AIBOs nearby. Currently, this information is not used by any other modules.

3.4 Positioning

Our Positioning module supplies the database with estimates of the current positions of the AIBO and the ball, as well as information of whether the ball is currently visible or not.

The Vision behavior supplies this module with information of objects the AIBO has seen. Given this information, a position can be computed using triangulation and trilateration. The Movement behavior supplies odometrical data that the Positioning module use to implement dead reckoning.

Monte Carlo Localization Because of the poor camera resolution and inaccurate object positioning during AIBO movement, the Vision behavior sometimes provides unreliable data. This can cause subsequently estimated positions to differ greatly from each other even when the AIBO has not moved. To solve this problem, some means of inertia needs to be introduced to the system.

The solution has been to implement a particle filtering method called Monte Carlo Localization (MCL), providing a positioning more tolerant of input data errors.

The basic principle of MCL is to keep a set of particles, each containing a possible position and a confidence value corresponding to how accurately this particular particle represents the correct position.

The particles are initialized with equal confidence values and containing randomly generated positions. Odometrical data from the Movement behavior are used to update the position of the particles according to the translation contained in the data. When a new position is calculated from field objects identified by the Vision behavior, the particles' confidence values are updated. The closer the particular particle's position is to the new position, the greater the confidence value will be, and vice versa. After any of these updates, a new absolute position is calculated from the particle set.

Resampling After a few MCL iterations, a majority of the particles will be depleted due to them drifting so far from the estimated position that their confidence values have become too small for them to contribute to the absolute position. To redeem this problem, we introduce a concept called resampling.

Rekleitis [5] refers to a measure of when to resample the particles: The effective sample size (ESS). When ESS drops below a threshold (experimentially obtained between 0.5-1.0 in our implementation) resampling must be applied to the particles as they are depleted.

The resampling method we have implemented is called *select with replacement resampling*. In this method, the confidence value of the particles will determine its chances to propagate further, and thus also to be duplicated more often. The higher the confidence value, the more likely the particle is to propagate further.

Maintaining the variance of distribution when resampling To maintain the variance of distribution among the particles, we create a small amount (in our case 10%) of randomly generated particles and let them propogate further.

The kidnapping/teleporting problem Instead of generating random particles on the entire field when resampling, we generate random particles within a region, with the center at the estimated position.

Calculating the absolute position The absolute position can be calculated in several different ways. We have implemented two of them. They are called the *best particle method* and the *weighted mean method*. The *best particle method* simply picks the particle with the highest confidence value, while the *weighted mean method* calculates a mean value of all particles, weighed by their individual confidence values.

The *weighted mean method* is generally better suited for real life applications and is thus our default method.

3.5 Distributed database

The Database behavior provides a central place in the AIBO system for storing data that is going to be globally available to other modules in the system. It provides a global communication channel between the internal modules and the other AIBOs that are participating in the soccer game. The Database behavior contains two database parts: A local part where all the data that the AIBO itself produces are stored, and a replicated part, which contains the data about the other playing AIBOs in the team. Which part to use depends on what data is to be stored. The reason for this design is to make it possible to dynamically connect and disconnect the AIBOs during a game.

Middleware We needed some way of storing data for localization and getting data from other AIBOs to make strategic decisions on. The solution to this is a middleware, which is a part of the Database behavior. The middleware's task is to setup and maintain connections to the other AIBOs.

The middleware is also responsible for the data distribution. It retrieves data from the other participating AIBOs and stores it in its own database. The distribution of data occurs in a predefined interval, currently 1 second. Each AIBO's middleware asks every other AIBO's middleware for their local data and stores this data in the replicated part of its database.

This enables other modules in the system to ask the Database behavior for data about another AIBO. Hence, strategic decisions can be made.

Data replication algorithm The data replication/retrieval algorithm works as follows: An AIBO's middleware detects that it is time to update the replicated data from the other AIBOs and sends requests to them for their data.

When data starts to arrive from the other AIBOs, it is stored in the replicated part of the database. A weighted value of the ball's position is created from the received data and stored in the local part of the database.

3.6 Simulator

Constructing strategies is an iterative process where things are constantly evaluated and fixed or discarded. We wanted to have some means of testing the strategies we came up with before the AIBOs were actually able to play soccer. Therefore we built a simulator whose main goal was to just run strategies in a simulated representation of the world. This would not only allow us to test strategies without running them on the AIBOs but also to run them faster (or slower) than real time.

The simulator uses the Strategy behavior almost as it is (there are only minor differences) and provides an environment for it. This includes stubs for the Database behavior, the Tactic behavior and so on. The simulator consists of a server doing the actual simulation, and a client, written in Java, that is used to view and control the simulation. This separation makes it possible to watch the simulation from multiple workstations over a network and connect and disconnect while a game is running.

Many simplifications have been made to the representation of the real world. We aimed at getting a good enough model to evaluate strategies, not to simulate every aspect of the AIBO and gameplay physics. There are other projects that do this much better than we could manage during the time of the project.

The simulator uses a 2-dimensional representation of the field since we felt that was enough to reach our goals. It is a frame-based model in which we evaluate the state of all objects for each time frame.

4 Evaluation and Testing

Initially we used CppUnit [6] to test our modules, but when we started integrating them with the Tekkotsu framework, this didn't work anymore.

Since we had eight AIBOs at our disposal, we could often test modules for real because it didn't bother anyone else. We also created debugging monitors for some of the modules, which presented that module's data in an intuitive way.

5 Conclusion

Tekkotsu By providing much of the low-level functionality that we would otherwise have been required to implement ourselves, Tekkotsu allowed us to move ahead to the more interesting high-level tasks much faster. This was especially

true for the Movement and Vision modules. The event handling made it very easy to develop modules separately and make them work together.

Unfortunately, most of the code from the first year of the project was unusable to us because it was not based on Tekkotsu. The second year's code used an older version of Tekkotsu than the one available to us at the start of the project. Furthermore, we wanted to take a more modular approach, changing the original Tekkotsu code as little as possible. Much of the code from the second year was merged with Tekkotsu's code.

We believe that our code would be much easier to reuse and/or extend, even with the use of future versions of Tekkotsu.

Tekkotsu presented some problems for the Database behavior though. The serialization used by Tekkotsu didn't work that well when we extended the implementation. This led to us having to make our own implementation, and the effort of extending Tekkotsu's implementation was wasted.

Strategy Even though TimesTool was not originally intended for this kind of usage, it worked well. The ability to present the AIBOs behavior visually, and being able to change the behavior by changing the automaton was neat.

The simulator could have been very useful for trying out strategies. Unfortunately it was developed in parallel with the rest of the project, and so couldn't be used until the very end.

Positioning Visual positioning exceeded expectations in terms of lighting conditions, but did not provide usable results during AIBO movement. Odometrical data could be used to compensate for this problem. Due to the absence of movement calibration, this data was too inaccurate to be useful for positioning. The result was that the estimated positions were not accurate enough to trigger the intended AIBO actions. It would have been very useful to have a proper calibration/monitoring tool, possibly integrated with the simulator, but development of one was unfortunately omitted in our project plan. This made it difficult to test the Strategy and Tactic behaviors on the actual AIBOs.

Tactic When implementing the Tactic behavior, we encountered some complications, one of which was to determine correct positions. Since most of the functionality in the Tactic behavior rely in great deal on the AIBO knowing where it is, it was nearly impossible to make things work properly with unreliable positioning. However, when the ball is seen, the position of the ball relative to the AIBO is often correct, which makes it possible for the AIBO to approach the ball successfully, as long as it is kept in sight.

References

[1] [AIBO SDE] official web site
`http://openr.aibo.com` (2005-03-04)

[2] Tekkotsu Development Framework for AIBO Robots
`http://www.tekkotsu.org` (2005-03-04)

[3] TimesTool
`http://www.timestool.org` (2005-03-04)

[4] CMVision
`http://www-2.cs.cmu.edu/~jbruce/cmvision` (2005-03-04)

[5] Ioannis Rekleitis. Cooperative Localization and Multi-Robot Exploration.
Ph.D. thesis, School of Computer Science, McGill University, Montreal, Quebec, Canada, 2003.
`http://www.cim.mcgill.ca/~yiannis/Publications/thesis.pdf` (2005-03-04)

[6] CppUnit Wiki
`http:\\cppunit.sourceforge.net` (2005-03-04)