Team Dynamo-Pavlov Uppsala

Paul Pettersson, Olle Gällmo,
Pahram Azimi, Rani Khalil, Martin Tillenius,
Arsenij Vodjanov, and Samuel Waxin

Department of Information Technology, Uppsala University
P.O. Box 337, S-751 05 Uppsala, Sweden.
Email: {paul.pettersson,olle.gallmo}@it.uu.se

Abstract. Team Dynamo-Pavlov of Uppsala is an effort at the Department of Information Technology at Uppsala University in Sweden, to establish a soccer team in the four legged league of Robocup. The core develoment team of the project is a group of 4th year computer science students taking a project course in the fall of 2002. In 2003 a smaller group of students have been working with the code to compete in German Open and Robocup 2003.

1 Introduction

Team Dynamo-Pavlov of Uppsala was founded in the fall of 2002, when a group of 25 undergraduate computer science students started their fourth-year programming project at the Department of Information Technology at Uppsala university. Their task was to complete a large programming assignments with the Sony AIBO model 210A as the hardware target platform. As one of the assignments, 13 students started the development of a Sony AIBO soccer team to compete in the legged league at German Open and Robocup 2003.

From the university's point of view, the aim of the project is give the students deeper knowledge in project management and methodology, and in the area of distributed system. A goal is to let the students take part of a large program development project from the early stages of planning, through the development phases of a software project, to application and systems maintenance. In the area of distributed systems, the students should get an opportunity to enter deeply into some aspect of a complicated distributed computer systems.

The main part of the project was conducted in the fall of 2002, when the students completed the original 15 credit point (75% of full time) project course. During the spring of 2003, the development continued in a smaller group of students, in two phases. First, to complete a version of the system to compete in German Open 2003 in Paderborn. Secondly, to improve the system to compete at Robocup 2003 in Padova. The team finished on 4th place in German Open, but did not qualify from group play in Robocup 2003.

To a large extent, this team description is based on documentation produced by students during the project. The rest of the paper is organised as follows: in the next section we describe the original system produced in the fall of 2002.

In section 3 we describe the improvements made during the spring of 2003 in preparation for German Open and Robocup.

2 System Overview

The system is divided into six modules. Each module run as a single OObject with message passing communication. In the following we give a brief description of each module. For a more detailed description, we refer the reader to [3].

2.1 Module Vision

The vision module is responsible for analysing the images taken by the camera in the AIBO's nose. Once a picture is taken, the AIBO will try to find key objects on the field, such as the ball, a goal or other players. When the image has been analysed, the gathered data is sent to the strategy module which will work out what to do with the information.

It also tries to determine distance to these objects as well as assertain its own position on the field. Localisation is done by triangulating its position with the six uniquely colored beacons around the field.

From AIBO's camera we get an image in YCrCb–color-space. This image is then run through a BitMask Classifier that gives us a new image with a more convenient color-space. The classifier strips off uninteresting colors, other colors are classified in the sense that we are told what kind of object this color can be associated to.

The new image is then given to an edge–detection function that returns a bitmap containing the edges of the given image. The image and the edge–bitmap are the base for the next step in our object detection, the flood-filling. The flood-fill–function is not only a flood-fill–algorithm though, it has evolved to an object–detector and as it fills an object in the image it gathers information about the object's height, width, color, position, area and perimeter. Very small objects are thrown away, as they contain too much uncertain information. Objects that seem to form a beacon are combined.

Distance. The distance from the robot to objects are all approximated based on the size of different properties of these objects. The size is expressed in number of pixels of each object in the picture. The approximation uses a mathematical function which is calculated from the relationship between object size and real distance.

– Ball - The size of the ball is based on its width. This is because it is common that the top of the ball is to bright to be recognized as the ballcolor, which is partly due to our lighting conditions. So the height of the ball will most of the time be non accurate, and hence the width is better suited to represent the size of the ball.

– Beacon - The size of each beacon is based on the distance between the center of the two beacon parts. We first used the height and width of each beacon part, but the distance d seems to give better distance approximations.
– Goal - The goal size is equal to its height, because the AIBO is more likely to see the whole height than the whole width, when there are other robots obstructing the view.

The distance approximation to objects is quite good at close range. But due to the poor resolution of the camera, the approximation doesn't give us distances that are accurate enough when object are far away from the robot. This inaccuracy sometimes makes the localization of the robot unsatisfactory, since it is based on the distances to visible beacons. The solution is to use triangulation. When the robot sees three landmarks, either beacons or goalposts, we use triangulation to recalibrate the distances to these landmarks. This will enhance the accuracy of the localization.

2.2 Module Strategy

The strategy module receives data continuously, evaluates the data and determines the behaviour to perform. This behaviour is then sent on to the motion coordinator module.

Behaviours. The strategy module has a set of commands that it can send to the motioncoordinator:

– GoToBall, *the AIBO goes to the position of the ball*
– GoToXY, *requires a point to be given*
– Aim, *requires an angle that the AIBO should spin*
– KickBall, *the AIBO kicks the ball*
– LockBall, *the AIBO lowers its frontlegs and head to lock the ball*
– PushBall, *the AIBO pushes the ball lightly*
– ScanField, *the AIBO scans the field without moving its body*
– SpinBody, *requires an angle that the AIBO should spin*
– MakeSave, *the AIBO makes a save*
– ContinuePrevious, *if the new calculated action is the same as the previous one, we don't send the command again, but send a ContinuePrevious instead.*
– Stop, *stops the current command*

Decision trees. The behaviour is determined using a decision tree that consists of a hardcoded binary tree. This tree looks slightly different depending on wheather the AIBO is a goalkeeper or an ordinary player.

2.3 Module Worldstate

The worlstate module is a map module that handles and stores the data received from the vision- and the communication module. The data received from the vision module is a message containing data collected by the vision module. The message is first converted into a so-called worldstruct. The worldstruct contains all objects allowed on the field. We then send a copy of the message to the strategy module.

The worldstate module does not only receive data from the vision module. Since the AIBOs cooperate and have a boss they also send messages, i.e their own worldstruct, to the boss. The messages are sent via the communication module. When the boss receives messages from the AIBOs it combines the data from the different players and broadcasts the new combined data to all the AIBOs. The idea with this is that each AIBO should have the same view of the gamestate.

2.4 Module Motion Coordinator

The motion coordinator module acts as an abstraction layer between the strategy module and the two motion modules (head– and leg– motion). A finite set of commands (with parameters) can be sent in sequence from module strategy to motion coordinator. The motion coordinator's task is to convert these abstract strategic actions into lower–level commands understood by the motion modules, and deliver these commands in sequence to the appropriate motion module.

2.5 Module Motion

The motion module is responsible for translating commands from the Motion-Coordinator module into movements. This includes walking, rotating, moving the head, kicking, saving etc.

Walking style. We have chosen to use a walking style known as *trot*. In this walking style two legs move simultaneously which makes the walking style quite fast.

The basic idea is that we specify a rectangle in the 3D-room in which each paw should move. We could then choose points in the rectangle and transfer the coordinates of those points into joint angles by using inverse kinematics.

The coordinate system is individual for each leg and origo is placed in the shoulder. This method can be used both for walking forward, backward, sideways (strafing) and rotating. We choose to divide the rectangle into six positions, one in each corner and another two at the bottom. This will ensure that the time in which the paw is lifted, is equal to the time in which the paw is dragged along the floor. All positions are relative to the AIBO's starting position.

2.6 Module Communication

The communication module handles network connections for the AIBO. It uses the ANT (abr. for OPEN–R Network Stack) for TCP/IP communication with other team members and our debug tool.

3 System Improvments

In this section, we describe how the system was improved for the competion at Robocup. For more a detailed description, we refer the readers to the reports [1, 2].

3.1 Vision

Color classification. The classification was done using limits in Y, U and V space. To be able to classify colors more precisely, we changed this into a color lookup table. Colors was downsampled to 18 bits, 6 bits per channel, and we introduced a table covering the entire color space. Colors was allowed to belong to more than one class.

We also added the possibility to our controller program to take a frame and mark which colors belonged to which class.

Object detection. The Sobel filter marked unwanted lines between an object and highlights on that object from the lights. We removed this and used floodfill to detect objects. Every second pixel on the screen was looked up in the color classification table. If the pixel belonged to exactly one class, a floodfill was performed staring in that pixel.

Localization. The vision system used to remember and report everything it has seen since the last time the Aibo walked, and if an object already seen was seen again in the next frame, the one with the largest area was used and the other thrown away.

We implemented a Kalman filter to get a mean of all the observations of an object, and also we introduced a limit so that an observed land mark only can be used for localization once, then it has to be observed again. This was because when three land marks are spotted, the localization would triangulate a position, and even if no new information is seen, these three land marks will be reported and used for triangulation each frame.

The position from localization was also filtered, and we allowed positioning using only two beacons again. When only one beacon is seen, we assume that our position is correct and use the position and this beacon to update our angle.

3.2 Communication

According to the project plan, communication module was to be degugged, and afterwards adapted for use with Sony's GameManager.

Existing communication code has been debugged and works, and some effort went into making it useful in actual game play. However, in the last few days before departure, we realized that the whole Communication module was unnecessary and, in fact, not even allowed to be used in RoboCup. The so called TCP Gateway that was mentioned in RoboCup rules was a misleading name of a Open–R module that allows TCP communication between robots via a host PC with a WLAN card.

Future work must therefore focus on making use of the TCPGateway library. Communication module currently acts as a sort of marshaller and encoder for all outgoing data. Current communication module may therefore still be employed for simplicity, i.e. used as a wrapper for TCPGateway–specific code.

A tip for improving execution stability is to employ full functionality of Open–R subject–observer architecture, that is, make full use of IsReady() function (and likes) to prevent internal buffer overflow and uncontrolled loss of data.

3.3 Motion

Kicks. After taking part in the German Open we decided that we could need some changes in our way to play so that we could be more effective. One of these changes was to develop some extra kicks so that the AIBO would not have to rotate itself and face the goal for it to shoot the ball towards the goal. Using the kicks that we had before the AIBO could only manage to shoot the ball straight ahead. We had two different kinds of kicks that did that, the NORMAL–KICK where the AIBO sat down to shoot the ball and the CHEST–KICK where the AIBO moved backward and then forward in a fast way hitting the ball with its chest.

After seeing some of the other team's kicks we decided to develop new kicks. We decided that we needed kicks that could shoot the ball in a 45 degree direction and 90 degree direction. We also decided to change the NORMAL–KICK since we had a slight problem with it during one of our games in Germany where the goalie was about to shoot the ball when it accidentally pushed the ball back with itself over the line allowing an own goal to be scored. The normal kick was modified so that the rear legs of the AIBO are pushed back allowing the AIBO to lie on its stomach before shooting the ball. This adjustment will not allow the AIBO to move its position before shooting.

We also developed a HEAD–KICK that shoots the ball in an interval between 75 and 90 degrees. The AIBO leans forward moving its head towards the other side from where the goal is. It then proceeds by swinging its head in the direction of the ball and simultaneously removing the front leg back causing the ball to be hit with power.

Another kick that was developed was the BSLAP–KICK. This kick was supposed to allow the AIBO to shoot the ball in a 45 degree direction. The interval for this kick was more between 45 degrees and 60 degrees. This kick goes as follows, the AIBO starts by lifting one of its front legs (depending on which direction the ball should be kicked), it then lifts the opposite rear leg, it then pushes with its other rear leg so that it gets a forward force, it then finishes by moving the front leg down causing a slap to the ball.

A similar kick to the BSLAP was developed, the WIDE–KICK. The AIBO leans its body backward lifting one of it front legs (depending on which direction the ball should be kicked). This movement causes the AIBO to nearly sit on its behind with one of its front legs in the air. It then proceeds by straightening up and continuing the movement forward while simultaneously moving the lift front leg towards the body. This causes the AIBO to hit the ball with power. The interval for this kick was between 45 degrees and 65 degrees.

Problems. Time–decreasing performance of kicks designed for angle ranges between 0 and 90 degrees. As time passes, some kicks begin to miss the ball completely without any changes being made to the code..

Possible reason: forward legs joints begin to loosen up after some time. Other teams (German Team in particular) reports similar problems.

Fixed by regressing to use forward or head kicks, limiting kick angle range to approximately 0 and 90 degrees, respectively. This caused large portions of Strategy code become obsolete and unused.

Proposed improvement: implement parametrized kicks that use the inverse kinematics (IK) engine, as opposed to the current implementation whith hard coded joint angles. This way, a few constants can be used to account for loose joints, and the kicks can become an integrated part of robot's motion, as well as become more precise, making overall gameplay more adaptive.

3.4 Strategy

Player Strategy. The main area of improvement lied in adapting the player strategy to the new types of kicks that were introduced. This involved some major changes to the way the decision tree looked, but no big changes to the core implementation of it.

Some time was spent to optimize scanning behaviour in an attempt to minimize the number of head–scans required to maintain knowledge of own absoute position. Steps were taken towards "remembering" previously seen landmarks, and roughly approximating own absolute position and direction based on seeing at most one landmark.

Additional code was written to make use of communicated boss orders. The players can perform a defensive, supporting or offensive action, depending on the command from the boss. It was noted that due to current implementation limitations, the resulting scheme deviated from the originally proposed. Rather than having independent agents make own decision based on a common set of data, we ended up with the boss analysing the data and sending out simple orders: Ignore Ball, Defend, Attack.

It turned out that the functions that were to merge sets of data from different robots into a common set were not used or not working as intended, and the only reliable thing left that the Strategy eventually received from the merged information was a simple boss order.

Goalie Strategy. The procedure behind the goalie strategy is based upon the system of a regular player. We did very little to change this process. The main modifications here were merely the behavior of the goalie.

Since we don't have absolute positioning at the moment, we had to divide the strategy tree into two branches. One in which the AIBO is in position, and the other one when it is not. We decide this based upon the actions taken, e.g. if a gotoBall() action is taken, the position mode is switched to `NOT_IN_POS`. This mode is the re–switched to `IN_POS` whenever the AIBO has found the goal and moved towards it. This happens when the mode is set to `NOT_IN_POS` and the ball is not visible, i.e. AIBO tries to locate goal.

If the goalie sees the ball it stops scanning, and tries to track the ball with its head. Whenever the ball moves sideways the goalie tries to strafe towards the ball location. This way it maintains a good position in the goal relative to the ball position in field. In every cycle, the ball position and ball relative distance is calculated. If the distance is less than a previous decided value, the goalie moves towards it to shoot it away from the goal. Since the task of the goalie is to defend goal, it just moves towards the ball and shoots it away without aiming towards a specific target.

If the ball is towards goal and the velocity is larger than a pre–decided value, the goalie makes a save. This is done by stretching out the arms sideways to protect as large area of the goal as possible.

The major problem we had with the goalie was basically the same as with the regular players. Since it didn't have an absolute position, it could never be sure of were in the field it was. Hence, most of the code is based upon calculations relatively to the camera. This caused in worst cases that the goalie was standing in an erroneous position, believing it was in front of the goal. Although the code for how to behave when absolute position is found exists in the behavior, it has never been used, nor tested.

Problems On–Sight. The following problems were found on–sight at Robocup in Padova:

– **High rate of kicks towards own goal.**
  Main reason: bugs in WorldState and PlayerStrategy code that attempted to minimize the number of head scans, by using previously seen data. Apparently too old data was being incorrectly classified as up to date.
  A problem that makes the above bugs particularly severe is the fact that there are other robots on the field that may obstruct vision. This is happening very often at exactly the worst time, since many robots tend to try getting the ball at the same time, obstructing each others line of sight.

Returned to the old scheme when the robot scans around after reaching the ball. Began working on a method that would allow determining the correct kick direction from seeing only one beacon, and remembering which beacons have been seen recently, and where, using a vector–weighting algorithm to determine optimal kick direction. This should prove a good solution if it was combined with the Kalman filtering on object positions (also implemented only partially). Complete Kalman filter implementation would likely involve rewriting the entire WorldState module.
One must remember however that any solution involving beacons will eventually have to go, since it is likely that beacons will be removed from the field in future tournaments.

– **Delays between different types of movement**
Assumed at first that the poorly optimized implementation of the IK engine (in LegMotionInterface) was the reason behind the delays. Rewrote parts of leg motion code to only calculate the immediate movement step ahead, not the entire sequence of steps for the whole command. The speed of motion increased, but the switching delay was still substantial.
Upon further analysis arrived at the conclusion that the real source of the problem is a design flaw in the decision hierarchy split–up between Strategy and Motion Coordinator. The entire idea of moving functionality from Motion Coordinator to Strategy turned out to be the reason for the delays. The Strategy module has become much less generic than it was before German Open. Complex MC commands such as GoToBall were reimplemented inside Strategy as sequences of simple walk, strafe and rotate commands. In a perfect world this wouldn't matter, but the CPU speed limitation and message passing delays presented a huge degradation of performance. The Strategy module's backbone is either too slow in general, or not optimized towards Open–R, i.e. too busy handling messages to and from the multiple subjects and observers that are connected to it.
This idea was tested by writing a simple version of a go to ball command completely within Motion Coordinator. As expected, no delays occured between different types of motion during the execution of this one Motion Coordinator command. Delays remained between execution of different MC commands, i.e. whenever Strategy was required both to produce – and issue – a new decision.
This may seem strange, since Strategy is currently traversing its decision tree about 25 times each second, without this causing any slowdowns to currently ongoing MC actions. Therefore, the slowdowns can be attributed to the bursts of Open–R messages to several modules at once, which occur each time a new decision is being made and has to be issued by the Strategy module.

References

1. Jens Alén, Rikard Hansson, Rani Khalil, Maria Olsson, Arsenij Vodjanov, Samuel Waxin, and Joakim Örtbrant. Improvements upon Dynamo Pavlov's code

for robotic soccer with Sony Four-legged in order to participate in German Open 2003. Technical report, Department of information technology, Uppsala university, 2003. Available as `http://user.it.uu.se/~paupet/gu/projdv02/-downloads/technical-report-go03.pdf`.

2. Pahram Azimi, Rani Khalil, Martin Tillenius, and Samuel Waxin. Dynamo pavlov robocup 2003 report. Technical report, Department of information technology, Uppsala university, 2003. Available as `http://user.it.uu.se/~paupet/gu/projdv02/-downloads/technical-report-rc03.pdf`.
3. Mikael Gransell, Rikard Hansson, Parham Azimi, Jens Alén, Frank Bjennmyr, Rikard Björklind, Arsenij Vodjanov, Rani Khalil, Alexander Simeonidis, Samuel Waxin, Jesper Bengtsson, and Maria Olsson. Main report dynamo pavolov. Technical report, Department of information technology, Uppsala university, 2003. Available as `http://user.it.uu.se/~paupet/gu/projdv02/downloads/-report-proj-fall02.pdf`.