The Brown University Robocup 2006 Four-Legged League Team Report

Ethan Leland (Captain)
Odest Chadwicke Jenkins (Faculty Advisor)
Brendan Dickenson
Dan Grollman
Mark Moseley
Brown University

May 2006

Abstract

This paper describes the design and implementation of the first four-legged RoboCup team at Brown University and lessons learned from our first year of RoboCup competition. The paper provides background about our team, outlines the overall structure to our implementation architecture, and describes each component of this architecture in more detail. The strengths and weaknesses of each module are discussed. Based on these observations, specific improvements are proposed for improvements to realize in future years of the Brown University RoboCup Team

1 Introduction

The Brown University RoboCup Team began the challenge of trying to field a four-legged team in the Fall of 2005. Most of our work in 2005 focused on defining a direction for the team, scaling the learning and infrastructure curves, and exploring previous work by teams in the 4-legged RoboCup

League. Learning the Open-R API and teaching it to our team took much of our time. Work was started on learning about the motion model and implementing a world model, and we had the dogs at least walking by the semester break. In December, Chad (along with Meinolf Sellman) secured a Salomon Grant entitled "RobAuCon – Autonomous Control for Robots from Demonstration" to fund of purchase of robots, equipment, and travel. Additionally, this proposal laid out a principal direction for the team, learning robot control policies from demonstration. We started to learn the rules of four-on-four robotic soccer, and thinking about the challenges of the game such as handling and shooting the ball, avoiding the other team, and avoiding the penalties that can be called during game play.

In February of 2006, that the four permanent team members got involved in the project and the code used in competition actually started to get written. By April of 2006, we had a team up and running to participate in the US Open, which was hosted at Georgia Tech in Atlanta, GA. In the three months that we intensely worked on the project, we were able to accomplish much of what we set out to do. However, there were still some things that we did not have time to implement, and there were many compromises that we were forced to make with our team. By the time of the competition we had the fundamentals of finding the ball, moving to it, and kicking it in the general direction of the opponent's goal all working, which we were very happy with. We spent a great deal of time on localization of the robots in hopes of implementing a crude version of team coordination, but were not able to localize sufficiently for making control decisions online.

A great deal about RoboCup was learned while competing in the 2006 US Open. We have gathered many ideas as to how to approach the problems that will persist for future competitions. Simply watching other teams play against each other was a wonderful learning experience, which taught us what kinds of things are important to field a more successful team. Many of the ideas we have for next year's competition will be discussed later on in this paper.

As a first year team, we made the decision not to write everything from scratch, but to borrow some parts from other teams who have released their code from past years. We would like to thank those teams for helping us make competition possible. This paper will explain exactly what code came from which team in more detail later on. Section 2 explains the system that

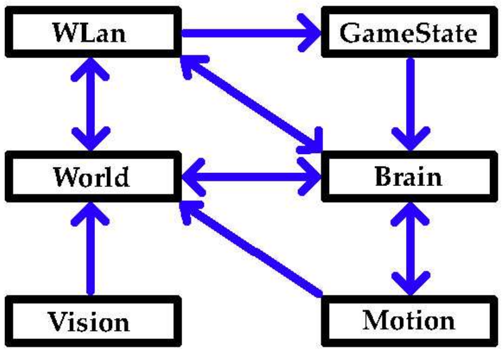

Figure 1: This is a representation of our system this year. Each box represents on module compiled into a .BIN file, and the blue arrows represent messages being passed between modules using shared memory.

we used for the US Open competition, and section 3 explains many of the problems we encountered with it. Section 4 explains what we would like to do with our team next year, and section 5 concludes the paper.

2 System Overview

When initially designing our system, we wanted an architecture that was very modular and could be broken up into different pieces, each of which could be worked on separately. We decided on the design in Figure 1, which is comprised of six main modules which were compiled into the following .BIN files:

MOTION.BIN: Receives and executes all the motion and LED commands that the system requires.

VISION.BIN: Captures the images from the color camera and processes each image, determining what objects are currently seen.

WORLD.BIN: Maintains a particle filter representing the world, updating it each time it receives new information from the Vision module.

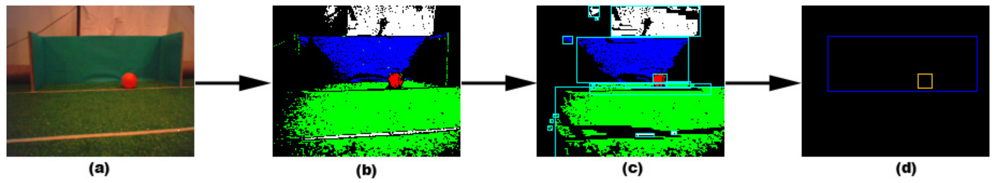

Figure 2: This is our vision pipeline. The images taken from the color camera would be converted to RGB (a), then segmented into the known color classes (b). Blobs of like colors would be detected (c), and from these blobs objects are highlighted (d).

GAMESTATE.BIN: Deals with state of the robot in the game (initial, playing, penalized, etc.) and updates the current state using the buttons and/or wireless signals.

WLAN: Uses the wireless radio to send and receive all the information used to communicate with other robots and the game controller.

BRAIN.BIN: Makes all the decisions of what the robot should be doing at any given time.

The communication between all of the different modules is done through a common packet sending class, using a header file to define each packet and what information it will contain. These packets are sent using shared memory and the OPEN-R API.

2.1 Vision

We borrowed the guts of our vision system from the MetroBots [5], which we used because we felt that it would provide a good structure and we could change pieces of it as necessary. Also, the MetroBots included a calibration tool which proved very helpful. The MetroBots vision system processes images using the following pipeline.

First, after converting the image from YUV to RGB, each pixel in the image is classified as either not an important color, or one of the following color classes: blue, yellow, pink or orange. To this we added the colors white and green in hopes that we would eventually detect the lines on the field.

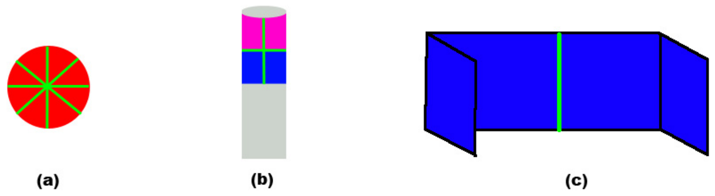

Figure 3: These pictures show how we determined the distance to the ball (a), the markers (b), and the goal (c). The green lines represent the measurements used in equations 2-4.

Each color is defined simply as a range of values in the red, green, and blue channels. This segmentation yields a 2D array of pixel values, which is then scanned through twice. First, neighboring pixels of like colors are joined. Second, these groups are merged into blobs of like color. These blobs of color are then used to identify the known objects, and the proximity of different colored blobs is used to verify that the object is seen. For example, the pink-over-blue marker is only detected if we find a blob of pink adjacent to a blob of blue.

Once an object is discovered, the distance and angle to the object is calculated using methods that we implemented ourselves. After experimenting with a number of geometric methods for calculating distance, we finally decided to use the focal length of the camera and the area in pixels that the object takes up in the picture. Our method uses the horizontal and vertical viewing angles of the camera, which are given in the robot's reference guide, [12] along with the known real world sizes of objects to derive a mapping from number of pixels in an image to a distance. We decided upon this method when looking though the code of Carnegie Mellon University [7]. Here are the equations to find the horizontal (HFL) and vertical (VFL) focal lengths of the camera, which are treated as constants in our code.

$$HFL = \frac{\frac{Image\ width\ in\ pixels}{2}}{tan(\frac{horizontal\ viewing\ angle}{2})} \quad VFL = \frac{\frac{Image\ height\ in\ pixels}{2}}{tan(\frac{vertical\ viewing\ angle}{2})} \tag{1}$$

We use slightly different equations for each type of object due to the special properties of each object and how we see them from the robot's view.

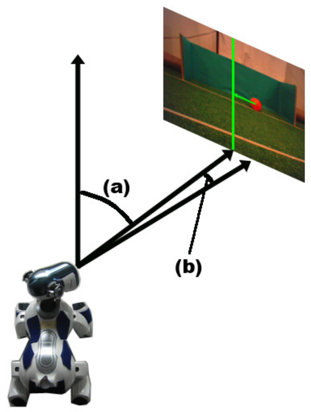

Figure 4: This diagram shows how we calculate the angle to a given object detected in an image. Value (a) is the pan of the head, and (b) is the angle between the center of the image and the object detected, in this case the ball. See equation 5 for how these values are used.

For the markers on the sides of the field, which are always seen as rectangles because the marker posts are round, it is easy to calculate the area each color takes up in any given image. The area is simply the width in pixels times the height in pixels of the blob of color detected. Then, to find the distance in millimeters to a marker, we use the following equation.

$$Distance\ to\ Marker = VFL \times VLF \times \frac{real\ world\ area\ in\ mm}{area\ in\ image\ in\ pixels} \tag{2}$$

The goals are slightly different because it is rare that we see the whole goal straight on. Therefore, instead of using the width and the height, we use only the height.

$$Distance\ to\ Goal = VFL \times \frac{real\ world\ goal\ height\ in\ mm}{height\ in\ image\ in\ pixels} \tag{3}$$

Since we understood that the ball is the most important object to identify, we try to get sub-pixel accuracy for the edges of the ball in any given picture. This is achieved by using four different scan lines across the ball at different angles and averaging them to determine the diameter of the ball. This is similar to a technique that the German Team uses [6]. We also tried scanning only in the horizontal or vertical directions, then choosing the maximum scan line as the diameter, but found it less effective.

$$Distance\ to\ Ball = HFL \times VFL \times \frac{real\ world\ ball\ area\ in\ 2D}{\pi \times (\frac{diameter\ of\ ball\ in\ image}{2})^2} \tag{4}$$

The same calculation is done to find the angle to any of these objects. First, the center of the object is found and the object's horizontal distance in pixels from the center of the image is determined. This is multiplied by half the horizontal viewing angle (HVA) of the camera to find the angle at which the camera observes the object. The angle of the head relative to the body (pan) is added to this to determine the final angle of the object.

$$\theta\ to\ Object = .5 \times HVA \times dist\ from\ center\ of\ image + camera\ pan \tag{5}$$

The final output of the vision system at each frame is simply an array of the objects that were detected, along with the angle and distance at which the object was observed. The angles are relative to the robot's body alignment, and can be positive or negative if the object is to the left or the right.

2.2 World

At the core of our world model is an Adaptive Monte Carlo Localization (AMCL) particle filter. [14, 11] This is different from a standard MCL particle filter because it varies the number of particles in the system. At all times a confidence value is maintained that represents how well distributed the particles are over the field. The idea here is that when all the particles are close together, the robot is well localized and does not need to keep track of as many particles, thus reducing processing time. This confidence value is based off of the standard deviation of the particles in the filter as well as how many objects are seen and how long ago the last object was observed.

The particle filter is updated each time that information is received from vision, and can be partially or fully reinitialized on the brain's request. For

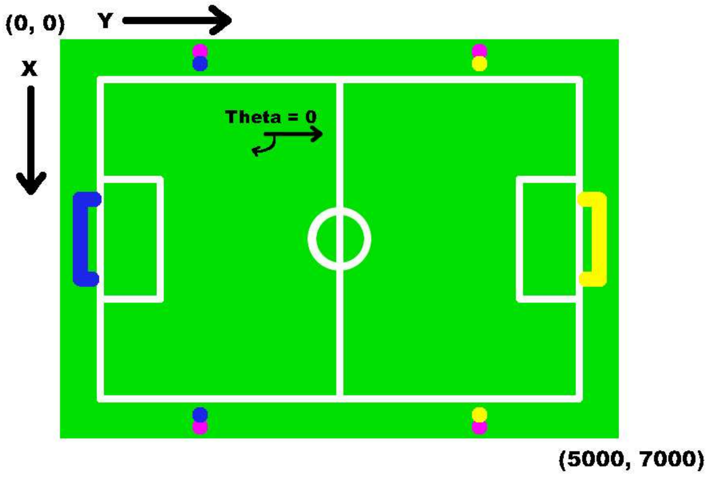

Figure 5: This is the coordinate system that we used for our world model. Our representation is 5m x 7m, and our (x,y) coordinates represent the distances in mm from the left and top of the field respectively. We always assume that our home goal is on the left, and to switch our teams color, we swap the positions of the four markers and the two goals on the field.

example, when a dog has just come out of a penalty the whole particle filter is reinitialized, and after the robot has executed a kick the ball's coordinates are reinitialized. Also, the particle filter takes into account odometry values that are calibrated for various walking parameters and are sent directly to the world model from motion each time a motion is executed.

The world model maintains a view of the world according to the robot, which included the robot's own location and orientation, and the location of the ball. These locations and orientation are relative to a static field shown in Figure 5. The output of the world model is simply the location and orientation of the robot, and the location of the ball. Furthermore, we decided to pass the information about the ball straight through from vision when the ball was within the camera's field of view, which allowed for faster reactions to the ball during actual game play.

2.3 Motion

The motion model that we used was largely based off of the University of Pennsylvania's motion code. [3] We decided that we would not have time to generate our own walking gait, and instead invested our time in exploring and porting UPenn's code to work with ours. Their gait is the result of the use of machine learning techniques to maximize the speed of the dog, and inverse kinematics to allow the dog to move in any direction at any time, including changing direction when it is in mid-stride. For more details, see [3, 9] This system worked well for us, although we could not achieve the same results that were described in their paper. Our maximum speed was approximately 295mm/sec. However, even this number is not entirely accurate, because the material that we used as a field in our lab was very different from what was used in Atlanta.

Also included in the code borrowed from the University of Pennsylvania were .MOT files which describe special action such as kicks and getup routines that we used during competition. We experimented with a number of different kicks, and implemented a system that chose the best kick for the situation that the robot's world model predicted. This choosing process was based off a number of factors including distance and angle to the target goal. Unfortunately, our world model was so inaccurate that we decided only to use one kick and try to trap the ball and then aim the robot at the target goal before executing the kick. The trapping of the ball is dealt with in the brain, and will be described further in the next section.

2.4 Brain

The brain that we used in competition was much different from what we envisioned the brain would be when we started this project. We first describe our control strategy in an ideal situation, followed by what was actually used in competition. In fact, we were able to implement some interesting and potentially useful ideas in the brain, but a number of factors kept us from successfully deploying these ideas in practicality. Primarily, the fact that localization mostly unsuccessful prevented us from implementing the team play we originally envisioned . This of course was a compound problem that will be further discussed in section 3.

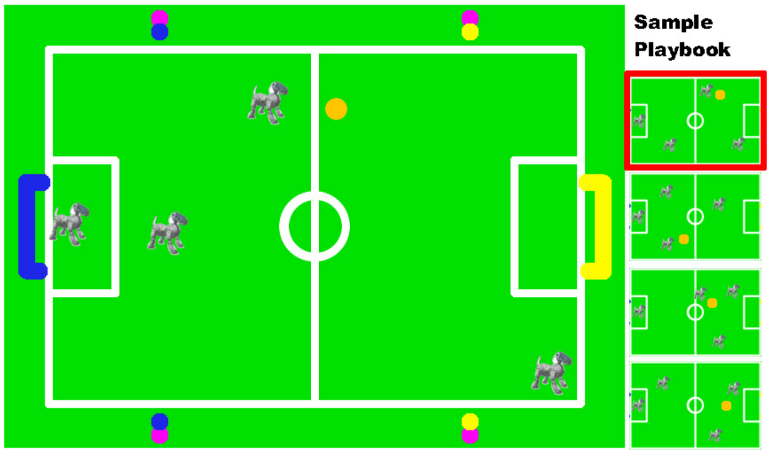

Figure 6: This is a graphical representation of the nearest neighbor search performed by our brain. The playbook (list of possible plays) is shown on the left, with the closest matching play highlighted. The role that each robot is to be assigned is stored in the playbook.

The team play that we implemented was based on a paper on Skills, Tactics, and Plays that was published by Carnegie Mellon University [1]. This paper describes a hierarchy of that has basic skills such as moving and kicking at the bottom. These skills are used by more advanced tactics such as 'move to ball' and 'shoot on goal', which are in turn part of a play that is being executed by the whole team simultaneously. An example of a play broken down into its base parts is shown in Figure 6. To decide which play should be running at any given time, a simple nearest-neighbor search was implemented that looks through a playbook (list of play descriptions) and the appropriate play is chosen. This playbook search was designed to be running on the goalie, who would distribute roles (tactics) to each of the other robots, then allow the play to run for a specified amount of time before choosing a new play. The skills and tactics that we implemented are shown in Table

Skills	Tactics
Move	Go to Ball
Turn To Ball	Shoot Ball on Enemy Goal*
Kick	Pass
Head Sweep	Go to (x,y)
Trap Ball	Search for Ball*
Point Head at Ball	Defend Goal

Table 1: Above are the Skills and Tactics used by the Plays in the playbook. The Tactics that we used in competition are marked with an *.

2. In competition, we actually did use the skills and tactics hierarchy, but there were no plays running, and only a very small subset of the tactics were actually used.

The tactic that was used the most was 'Shoot ball on enemy goal.' This tactic was a finite state machine that involved moving to the ball, trapping it, turning until the robot could see the enemy goal, and then executing a shot. If the enemy goal was not seen after three seconds, the robot would have to shoot anyway, since one rule of RoboCup is that no robot can hold the ball for longer than three seconds. Moving to the ball involved strafing and turning towards the ball at the same time which took advantage of the robust UPenn motion system. Trapping the ball was accomplished by bringing the head down over the ball and attempting to open the mouth. If the mouth could not open all the way, the ball was considered to be trapped. Finally, the robot would turn in the direction that it thought the goal was (according to the world model), and shoot if it saw the goal or if it had been holding the ball for almost three seconds. The only other tactic that was used was the 'Look around' tactic, which searched for the ball when its location was unknown.

There was also another brain that we used in competition that made decisions based solely on learning from demonstration concepts. Using joysticks to control the robots, humans played soccer against each other or other teams to train this brain. The training involved creating a mapping directly from segmented images captured to actions. During competition, this brain would perform a nearest-neighbor search comparing the current segmented image from the vision system to a database of saved images and perform the action correlated to the nearest match.

3 Results and Known Issues

Overall, we were enormously proud of the results that just a few of us were able to achieve in such a short amount of time. We were able to compete in the US Open, and even scored a goal in competitive play, and tied that game 1-1! Of course, there were certainly a number of issues that we knew to be problems and would have liked to fix given more time. Many of these issues did not come up until we actually started competing against other teams, and the descriptions below are mainly the result of the knowledge we acquired while watching competitions and talking to other teams. These other teams were extremely receptive to our questions and helped us determine exactly what was going wrong.

3.1 System Architecture

The one system-wide issue that we were constantly dealing with was timing. In an event driven system such as ours, timing is key, and we did not have a sufficient way of dealing with it. The vision would run each time it received a frame from the camera, the localization would update when it received vision data, and the brain would only execute when it received a new world model. This seems ok on the surface, but we were never sure of exactly how quickly these events occurred, and thus did not have the best time-relative information in the world model, thus restricting us from making the best time-sensitive decisions in the brain. This was all a result of our decision to value modularity of our code above efficiency. The result was a very robust system which did not handle the information it received from the world in the most efficient manor. A potentially better system architecture will be described later, in section 4 Now we will describe a few issues we had with each module.

3.2 Vision

The vision system we chose turned out to be slightly buggy, and we ended up fixing a few memory leaks that we discovered just days before the competition. However, the far larger problem that we encountered was our inability to easily deal with changes in light across different competition spaces. We found ourselves constantly calibrating our color space, and could never get something that worked as well as we would have liked. We also ran into problems with the camera such as motion blur, camera calibration, and too strong of a blue channel which had us seeing the blue goal all over the place. One method we used to reduce the amount of blue we detected was to segment for the color green before blue so that pixels that were close to both green and blue were classified as blue. Also, since the blue we saw was mainly in the corners, we decided to ignore any blue goals that were detected in the corners of the image.

The lack of well segmented images made distance and angle calculations especially hard. When the edged of an object do not show up in the segmented image, the object is thought to be further away than it is. When standing in one goal looking directly at the other, we would get fluctuations of more than 1m in the distance to the far goal which was 5.4m away, and the angle would vary by up to 30 degrees in either direction.

3.3 World

When tested on fake, perfectly accurate data, our particle filter and world model worked very well. Unfortunately, there was a great deal of noise among all the inputs to the particle filter that, when compounded, rendered the whole localization system unusable. The main source of our error was the vision data. The lack of good color segmentation made accurate object recognition very rare, which really hurt localization. Another large source of error was in our odometry estimates. We calculated odometry through timed observations which made it difficult to maintain a high degree of accuracy when moving during gameplay, especially when we combined actions such as strafing and turning at the same time.

Once we realized just how noisy the vision data that we were getting actually was, we implemented the idea of having an interpreter to filter out some of the more noisy data. This interpreter compares the new data arriving from vision to a short history of data that has been recently received. If the new data is too different from the historical data, it is discarded.

If we had known about all these various sources of error before writing the particle filter, there may have been some ways of dealing with it well. We could have put all of the particles on an arc around an object that was being observed. We could have used dead reckoning to the markers to improve the goalies sense of the world. However, it was extremely difficult to do these things because we did not have a final working version of the vision system until only days before the competition.

3.4 Motion

Only after watching the actually competitions did we discovers issues with our motion. Simply put, we always operated within one set of motion parameters, while other teams change their stances and gaits on the fly. As a result, we had issues trapping the ball, seeing the goals while the ball was trapped, and were often out-maneuvered by other teams. Also, due to our larger timing issues, we would occasionally call motion commands too quickly. This would cause our robots to appear to spasm because they were executing one motion command before the last one finished. To fix this in competition, we added slight pauses in different parts of our architecture.

4 The Future

We have a number of ideas for how to improve our system for future years, and we recognize that next year is going to bring major rewrites to almost all of our current system. Once again, almost all of what is described in this section is the direct result of what we learned while attending the RoboCup competition in Atlanta, as well as looking through other teams' reports afterwards.

4.1 System Architecture

First and foremost, we have decided upon a slightly different overall design for our system. In brief, we would like to combine the vision, world, and brain modules all into a single .BIN file. Doing this would allow for far easier communication between these modules as well as much more exact timing for the decision-making pipeline. Also, this would reduce the amount of shared memory that is used in our system in general.

Another design issue with our current system is that we open and access different sensors and joints in different places. In the future we will consolidate this into one .BIN file that incorporates all of the motion, LEDs, and sensors into one module. The final module in our future system will be in charge of the wireless radio and the gamestate, again simply combining two of our existing modules.

With careful coding, this new system can still be as modular as the current system, and will solve some of the problems that we have described in this paper.

4.2 Vision

This is one place that we plan on focusing a great deal of effort on for next year's team. One idea that we will be looking into more is the concept of object recognition through feature point detection. We have written a basic SIFT routine that will be ported to the robots and are interested to see if this will work. [10] If we decide to use more traditional techniques, we will need to rewrite our vision code from scratch.

Color segmentation of the images from the camera is very important in RoboCup, and we need to focus our effort in this area first. We will no longer be working in the RGB color space, since YUV is the native color space of the robots, and the conversion takes unnecessary time. The German Team uses an novel method that uses the TSL color space as described in [4] which we are considering. They also use neighboring color values to help classify a pixel's color. These techniques will help up segment much better, making the remaining vision problems much easier. To define our color classes, we are going to write a real-time interactive video utility that captures the images seen by a robot and displays them on a screen. Our vision system will be running on this video, so the user will be able to see how the robot currently views the world while they make changes to the color classes.

Once we have the segmented images, we would like to only perform one pass to find the color blobs, and then perform object detection. From there, we will use techniques similar to what we used this year to determine distance and angle to each object.

Another scheme that many teams use [7, 6] deals with detecting the horizon line. Using the horizon, we could rule out objects seen in places they cannot be, and look for other features such as lines only below the horizon. Line detection using Hough Transforms [2] are going to be a must for next year, and will help our localization a great deal. For now, we will stay with our distance calculations, as they are quite robust and worked well this year.

4.3 World

To improve our world model, we will be investigating the use of a Rao-Blackwelized [8] particle filter to replace our AMCL method. We feel that this will be able to better keep track of where the robot is over longer periods of time, and possibly even allow us to successfully autonomously place the robots in the correct positions for kickoff. We plan to share world model information between robots as well, which will allow for group ball localization and better team play, and even help robots know where they are themselves.

4.4 Motion

As of now, we are thinking that we will continue to use the motion model written by UPenn, for implementing a whole new walking gait could amount to too much work when there are a lot of other things to work on. However, there are still many changes to be made. We need a more robust motion system that can change gaits on the fly and implement different kicks in different situations. Also we need more accurate odometry to send to the world model while different actions are being executed. We will also be looking into some new kicks of our own, and moving the trap-ball motion from the brain to the motion module.

4.5 Brain

The new brain will be heavily based on what was described in the first part of section 2.4. Of course, there will me some more skill and tactics that will be implemented, as well as a full playbook. However, what the brain will do is heavily dependent on how well our other ideas are implemented, and we cannot yet predict the full extent of our future brain. There are some things that we do know:

- We need to try to get fewer penalties called on us
- We need the goalie to move and possible rush at an oncoming offender
- Keeping one defensive robot near the home goal is important

4.6 Higher Level Language

Many of the other teams competing in RoboCup use a higher level text-based language such as Perl or Lua for behaviors and decision making, which is something we are considering as well. However, in order to use these languages an interpreter would have to be written to run on the robots. This interpreter would have to make all the necessary function calls to the lower level Open-R commands such as motion and camera calls. The higher level language would then be used to handle all the data that is received from the robot, acting as the brain. There potential advantages to using one of these languages are that the behavior code would become cleaner and more modular. The robot and interpreter would act more as a platform to work on instead of a whole system to be built at the same time. All the behavior code would be in text files which could be quickly and easily modified and reloaded on the robot instead of recompiling lots of code. Also, a behavior simulator would be much easier to write for computers that could also help speed up the development of the brain, and new behaviors could be written as necessary during competition.

The main downside to using a higher level language is the implementation time. In our first year we wanted to get something to work, which we did. Next year we already have a number of changes and rewrites to do, and we might not have the time or resources to build this into our system.

5 Conclusion

Overall, we were very proud of what we were able to accomplish this year, and are very excited to start rebuilding our team for next year. Now that we have seen competition, we have a very good understanding of what is important, and where to start our new team. Vision is going to be the first thing that we have to write, starting with color segmentation and then horizon and line detection. Next we would like to improve the motion code, adding on the fly parameter changes as well as new kicks, some of which will use the head. From there, we will need to rewrite localization, which will allow us to start working on the brain. To help us speed up the development cycle, we will initially be investigating the TCP Gateway, which we did not use this year, and is described in the Sony Aibo Programmer's Guide. [13] This year truly was a learning experience, and we are ready to start again with the knowledge that we have gained.

Acknowledgements

We would like to thank Brown University for granting us the Salomon Grant, and our collaborator Meinolf Sellman. Also, thanks to the Robocup Federation and US Open organizers, namely Tucker Balch and Manuela Veloso. Finally thanks to all the other teams that helped us improve our system and compete in the US Open.

References

[1] B. Browning, J. Bruce, M. Bowling, and M. Veloso. Stp: Skills, tactics and plays for multi-robot control in adversarial environments, 2004.

[2] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, 1972.

[3] Cohen et. al. The university of pennsylvania robocup 2004 legged soccer team. Technical report, 2004.

[4] I. Dahm et. al. Robust color classification for robot soccer.

[5] Matrinez et. al. The 2004 metrobots four-legget league team. Technical report, Columbia University, City University of New York, Brooklyn College, 2004.

[6] Thomas Rőfer et. al. German team robocup 2004. Technical report.

[7] Velosa et. al. Cmpack'04: Team report. Technical report, Carnegie Mellon University, 2004.

[8] Z. Khan, T. Balch, and F. Dellaert. A rao-blackwellized particle filter for eigentracking. 2004.

[9] N. Kohl and P. Stone. Machine learning for fast quadrupedal locomotion. *The Nineteenth National Conference on Artificial Intelligence*, pages 611–616, 2004.

[10] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004.

[11] Sridharan M. Kuhlmann G. Stone P. Practical vision-based monte carlo localization on a legged robot. *Robotics and Automation*, 2005.

[12] Sony Corporation. *OPEN-R SDK Model Information for ERS-7*, 115-01 edition, 2004.

[13] Sony Corporation. *OPEN-R SDK Programmer's Guide*, 115-01 edition, 2004.

[14] Sebastian Thrun. Probabilistic robotics. *Commun. ACM*, 45(3):52–57, 2002.