An Attempt at 4-Legged Robocup

Michael Elliot

School of Arts and Sciences

Boston College, Chestnut Hill, MA

Abstract. An attempt to design and implement an entire AIBO to be Robocup ready proved a challenging, and rewarding experience. Due to the large scale time requirements and time consuming technical issues that had to be dealt with, my experimentation was cut short.

Successes, however, include the creation of a vision system and of an omni-directional walking system that could be used to achieve the ultimate goal of playing soccer. These systems are in many ways a final product in themselves—typically teams are separated into groups which complete each sub-model in the dog, giving them freedom to explore many options. In addition, I had no starting point and had to learn the OPEN-R programming environment on my own, and thus, development required a significantly greater amount of time as compared to a situation where there was already a working system with knowledgeable people there to oversee.

1 Introduction

The idea of competing in a Robocup competition excited was me very much. However, as I began developing my own code for Sony's AIBO ERS-220, I realized the great challenge that I faced as a single worker in a world where typical teams can have over ten members diligently working on various parts of the design, implementation, and verification. Although my eyes were big, as one person I could not be expected to get through the technical issues surrounding a

new programming target (i.e. Sony's AIBOs) and develop what often ten graduate students would tackle together.

The systems implemented involve both walking, object recognition and localization. These designs followed those of the previous big-leaguers in the Robocup tournaments**.** Typically my goal was to capture the ground breaking designs, and implement the simplest versions of those designs. Even with big eyes, my goal was never to truly be competitive in the Robocup, but instead to learn the intricacies of autonomous agents and distributed learning. Unfortunately, distributed learning was never achieved due to the lack of time and help from additional team members.

2 The Robot

The ERS-220 can be seen in Figure 1. As show in the exterior shots, the AIBO robot is designed with joints similar to that of actual living dogs. Each leg consists of 3 joints which can be accessed via primitive commands in the code running by the robot. The primitive addresses are available in the Model Information guide provided by Sony at their OPEN-R Website. The robots come with a single in-face camera which is accessible in a similar fashion. All details of other needs, such as buttons or foot sensors, are found in the Model Information guide and can be accessed using the same primitive commands.

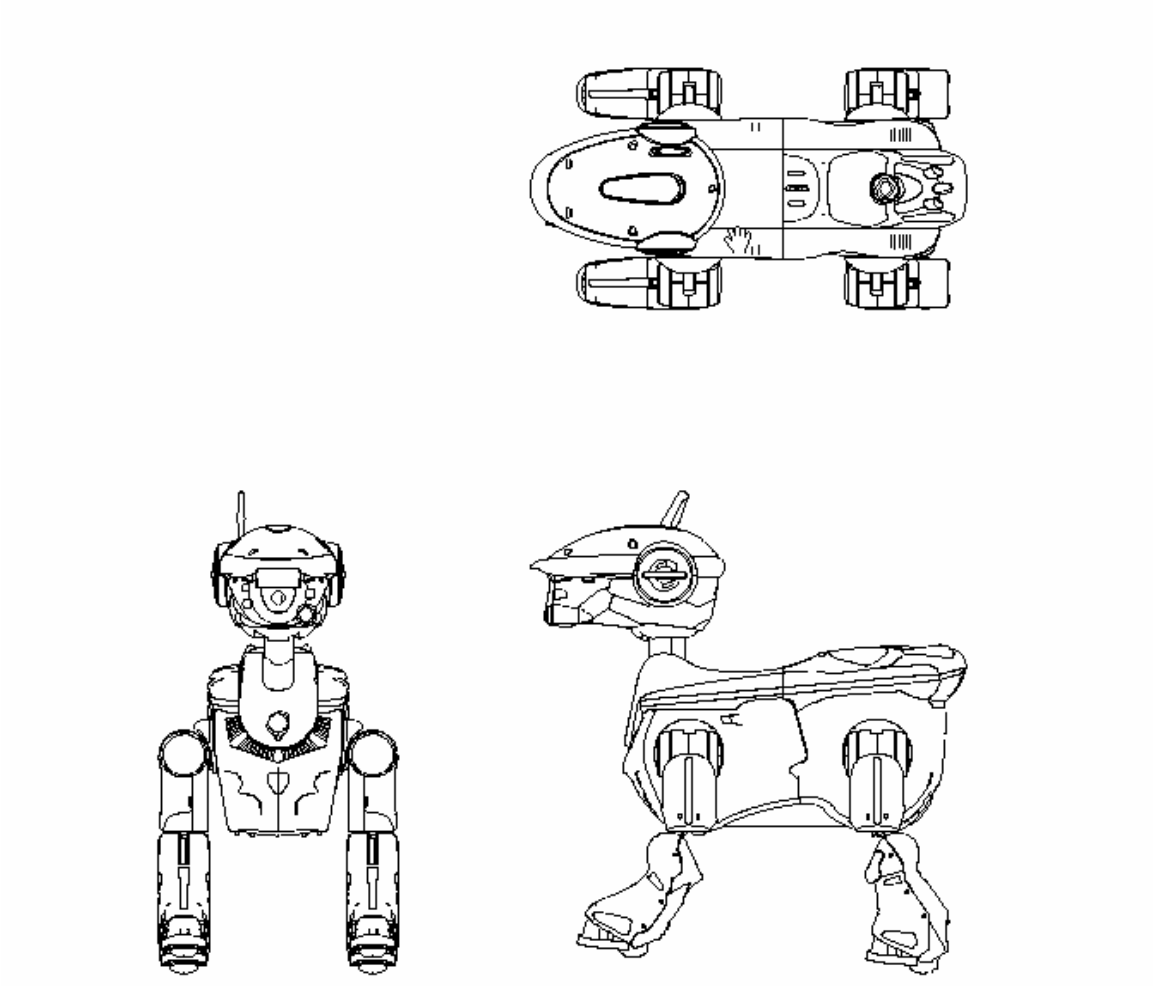

Figure 1 – ERS-220 Model Appearance

2-1 Programming Environment

In order to compile OPEN-R programs, a Linux based compiler must be used. These are available through the OPEN-R website provided by Sony. The trick, however, lies in the fact that the floppy-to-memory-stick converter is Windows based (note that it will not work with a dual processor system!). Therefore we must utilize Cygwin for general implementation situations. There are Cygwin binaries available at the website. Cygwin can be obtained at www.cygwin.org.

A Unix or Linux system can be used to implement the code as well, and instructions to do so are available on the OPEN-R Website. In this case an FTP server can be utilized to transfer code over a wireless network and then the robot can be remotely rebooted. This method is possible because on boot the robot copies the code into memory, leaving the memory stick alone. In order to do this, the FTP server, which fortunately is provided by Sony, must be running at all times. See appendix C for more information.

2-2 Programming Model

By nature of being an autonomous agent, the ERS-220 cannot rely on an operating system to perform the necessary tasks required to keep track of the state of the robot. To accomplish this and maintain a multi-threaded approach, each defined object runs concurrently. A round-robin approach is taken by the robot's processor, giving equal processor time to each running object. To date there is no publicized way to avoid this threading protocol.

In order to communicate to another object or share data, inter-object communication must be utilized. This is done through a subject and observer model, where the observer waits for information to be passed after declaring a ready state. This ready state declares to the subject that the observer is ready to receive data. This can be seen in Figure 2. This information, however must be known at compile time as well as at runtime. All inter-object communications

have to be written out in the connect.cfg file located in the OPEN-R/MW/CONF/ directory on the memory stick.

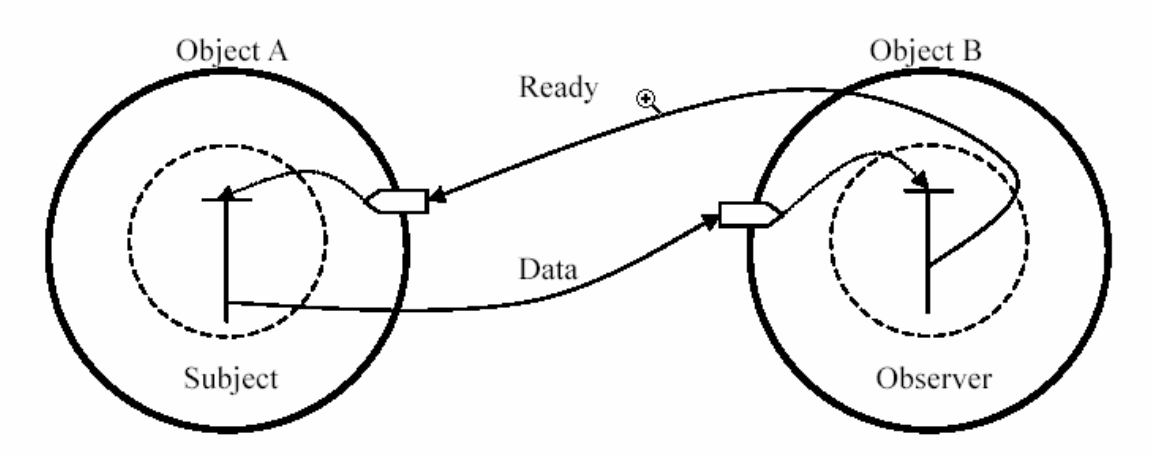

Figure 2 – Inter-object communication visualization

Also due to the autonomous nature of the robots, there must exist a Core Class—which inherits from the Object Class—in each object running on the AIBO. There are four important routines that must be implemented by this class. They are Init, Start, Stop, and Destroy methods. The doStart method which is called on startup by the robot, makes a call to Init and Start. There are also methods used by the inter-object communication services, but those are defined in a stub file. This stub file is used by a routine provided by the OPEN-R SDK and must be called in the Makefile. Fortunately due to the complexity of this programming model, there are examples of working code available. However, these examples tend to be far simpler than a realistic piece of code. It is possible to follow directions provided in the programming guide available from Sony. Most importantly, having a sound design and image of what needs to be

implemented and what types of inter-object communication are needed, is necessary in order to better maintain and configure the necessary stub.cfg and connect.cfg files.

2-3 Makefiles and compiling

The compiling of the code for the robots is somewhat detailed. This is mostly because the target processor isn't the typical one. The compiler is located at /usr/loca/OPEN_R_SDK/bin/. Also a binary maker must be run from /usr/local/OPEN_R_SDK/OPEN_R/bin/mkbin. In general, these Makefiles are of the plug and play nature. They can also set up a directory structure that may be of benefit. Appendix A contains a very fine example that automates much of the work.

2-4 Running Code with Wireless Network Communication

OPEN-R provides many important objects that can be utilized and must be used when programming the AIBOs. This also provides support for memory protection and wireless communication. These files are available from the OPEN-R SDK and are found at /usr/local/OPEN_R_SDK/OPEN-R/MS/ in the Unix tree structure. These files should be mounted onto the memory stick's root. In general the WCONSOLE with memory protection should be chosen, thus mounting /usr/local/OPEN-R/MS/WCONSOLE/MS/OPEN-R to the root is

required. Memory protection prevents objects from touching memory used by other objects through allocation of memory in only blocks of 4096 byte chunks. Although this can lead to a decrease in memory utilization, it obviously prevents nasty memory related bugs.

The wireless communication settings on the AIBO do not utilize DCHP. DCHP allows a host to obtain an IP address and all other needed information such as default gateway and default Mask, dynamically at startup. For reasons that I am unaware of, the AIBOs must have a static IP address and have all other network configuration settings known at boot. I was fortunate to obtain not only static wireless information, but also an IP address for each robot. See Appendix B for details of what the wlanconf.txt file looks like. If this is not perfect, the dogs will not work on the wireless network.

The last thing to do before running is to edit the OBJECT.CFG file located in /OPEN-R/MW/CONF/ directory. This file contains all of the objects to run. These objects will be run regardless if they do not require inter-object communication. A nice thing to do here is include an FTP server so that it can be utilized to run reboot commands remotely and other such things. The objects must be placed into /OPEN-R/MW/OBJS/. Further details are available in the Open-R SDK Programmer's Guide provided by Sony.

3 Previous Works

Robocup was started to provide a common interest to merge what seemed to be a variety of drifting, inter-dependant Computer Science Topics. These topics include Robotics, Distributed Computing, Artificial Intelligence, and Computer Vision. Thus a common ground and common playing field, robotic soccer, was selected as the method to further artificial intelligence, robotics, distributed computing, and computer vision. At first there was just a simulation league, where artificial intelligence and distributed computing was the focus. Here eleven separate threads of execution, typically coming from eleven separate computers were connected to the 'soccer server' in a client server fashion. The realism of this simulation league continues to grow to allow for coach interaction using predefined commands. In addition there are three robot (small, medium, and large) leagues. However, the nature of this league is inherently unfair in that robotic design is left up to the teams, resulting often in largely better teams in some cases.

Like the simulation league in many ways, there is one other league that keeps the playing field perfectly equal in regards to hardware and that is the 4-legged league. The hardware used here is provided by Sony in their AIBO robotic dogs and this is clearly where my area of interest lies. Also, this illuminates the need for the design and creation of a new set of robots. Although costly, the AIBOs provide a nice structure to work with, providing many important and helpful features that aid in robotic soccer.

The leader of the pack in the 4-legged Robocup league belongs to Carnegie Mellon University (CMU) and Manuela Veloso, the clear visionary of the

entire league. Since 1998 and the release of OPEN-R by Sony, he has had groups of students doing brilliant work and improving elements of the team each year. It is their Vision System's design (from 2002) that I attempted to implement. There are many features of their design which make later, higher level work much easier. More or less, the bulk of the work takes place at a lower level, which feeds the decision processes with nice structures that make decision making faster, and more precise.

Although CMU is always a front-runner in Robocup competition, University of New South Wales of Sydney Australia has been a force in the development of faster walking mechanism for the dogs—they are inherently slow, easily to push over, and generally difficult to program. Their improvements and design for the 2000 Robocup are my basis for implementation of walking code. Their use of omni directional methods provides a remote like interface for the artificial intelligence portion of the system and tries to minimize its own effect on the vision system, which is attached to an often bobbing head.

Typically teams implement a variety of kicking methods including a header and side kicks. However, the most effective kicking motion seems to be by using the wedge located on the chest of the AIBOs. By falling on the ball, there is a large mass behind its acceleration, sending it much faster than otherwise able. The legs, in this case, are used as guides.

Other ideas, such as placing the goalie dog inside the goal when the ball is in the corner, allow the dog to maintain a few of the ball, thus giving it the best chance of preventing a goal.

As Robocup has grown, the 4-legged league has placed humans at the controls of one team of AIBOs, to determine just how far the artificial intelligence teams were coming. Inter-squad competition seemed to indicate an improvement in play, but to use humans gives a somewhat consistent point of comparison. In 2001 the humans were defeated for the first time by the artificial intelligence provided by the winning team.

The goal of Robocup is to not only push new research and extend the field of Computer Science through competition, but its ultimate goal is to compete with actual humans in a real game of soccer with large humanoid robots. Although the field of robotics is the lagging section of the many fielded research of Robocup, it is hoped that these ventures in artificial intelligence, distributed decision making, and vision, will lead to faster understanding of what is needed from the robots in order to be successful and also to determine how to teach robots to work as in groups, that do not have a 'master' dog dictating what needs to be next. These ideas create human like abilities in each robot, and keep dependencies as low as possible. These abilities can then be used in other areas, as needed, to aid in better lifestyles for all.

4 My implementation and experiences

Following is the general principles behind these systems, as well as my experiences during their creation.

4-1 Omni-Directional, remote-like, locomotion

A fast, reliable, and remote-like implementation of a Motion system was the reason University of New South Wales was able to win all competitions in 2000 at the 4-legged championships. The great improvement in speed comes from looking at the AIBO's legs as a circular motion, creating a rectangular locus that can be modeled using a few formulas. With such a design it is possible to maintain steady, fast speeds simply by maintaining the size of the locus, and traversing at the same steady state. Furthermore, this design gives the ability to raise the legs off of the ground as little as possible leading to a minimized amount of up and down motion which can affect the vision systems.

Omni-directional control is obtained by changing the inclination in the plane we created for forward motion. Thus if the locus plane is parallel to the robot you achieve forward and backwards motion (likewise for sideways motion when the locus is perpendicular to the robot). To turn however, the circular motion is the best way to understand the method used. Each leg's motion is similar to a wheel, as the limbs rotate in circular pattern. Thus, by moving some of the legs farther, and therefore faster than the others, a turn is achieved. The ability to control each leg's speed is therefore necessary, and the speed of a leg is determined by the width of the locus and the speed at which it is traversed.

The speed at which the locus is traversed is dependant on its size (moving the legs higher off of the ground takes more time) and the rate of the gears.

The method of locomotion is described by the notion of gaits. A gait is the protocol or policy that is followed when a four legged animal or robot moves. There are three widely accepted gaits: crawl, trot, and pace. The crawl has all four legs moving at different times, where the trot has opposite legs working in tandem, and the pace has legs on the same side of the body working in tandem. All of these were implemented by University of New South Wales team. I borrowed some of their parameters and rates in my own implementation. I found similar results that indicated the trot gait was the best choice for fast, steady motion.

The design allows for the manipulation of a variety of parameters including gait, distance to move legs (negative distance taking the dog backwards), distance left each step should take, turn in angle, starting position of legs, and heights to raise front and rear legs to. This implementation allows for optimization of these settings. Fortunately University of New South Wales did this optimization, and only a small amount of tweaking was required for our carpets at Boston College.

The walking mechanisms were tested at a slower than could be used gear speed. I did this because of the risk of damage when causing the dogs to move at such high speeds. Regardless, the dogs will be highly effective in receiving commands. I implemented a joystick like interaction with this code which means that when I indicate a walk command such as (walk 85 0 0) the dog will move

forward at eighty five millimeters per step, with zero motion to the left, and zero motion in a circular fashion (i.e. the rotation of the entire robot).

In order to verify that these parameters and system worked, I had to create a way to access the module in the robot, and thus created an extension to the telnet server that was written by Sony and freely distributed. I could then simply play and verify all of the things that University of new South Wales claimed were true about this design. The difference in walking speed is considerable, as I raced the Sony walking code with that of this team at the same gear speeds, and found it to be nearly twice as fast.

4-2Vision and World Model

As previously mentioned, I used the method described in CMU's team paper from 2002. Vision is a multiple step process that inherently requires much of the robot's processing time. It is this time that I was worried about utilizing too much of.

As inputs we get eighteen frames per second from the robot. The first step in the vision process is segmentation. On each pass of a color, the highest level bits are used to look up a related color in a lookup table that allows for variance in color and in particular for lighting differences. The lookup table is 65 kilobytes big, maps each color to a more generic one and was directly borrowed from CMU's work. The resulting image is stored for possible later use. As lookups are done, tracks of runs of similar colors are kept track of and recorded

into an array with an x and y, and an ending x. The objects also contain a parent node. This makes for easier and faster connection of components. Starting with a disjoint forest of nodes (these are the individual runs previously calculated) with the parent of each node being itself, adjacent rows are joined. The actual parents must be determined if there is a case where a row is connected from multiple locations. Figure 3 shows this in better detail. The parent nodes are then sorted based on size and color for later use. In addition, once this is performed, the regions that weren't of exact matches needs to be merged, in the interest of perhaps locating an actual object like the ball, goal, or landmark. I merged on the assumption that the resulting object had a density of .89.

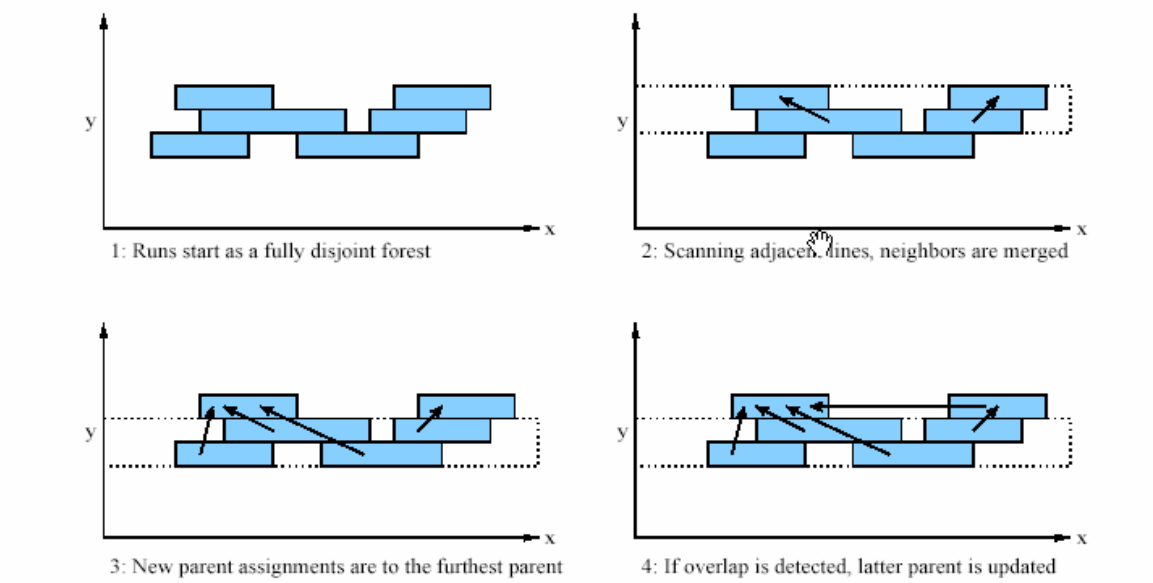

Figure 3 – Visualization of Region Creation

Using these regions we can then begin to locate objects and calculate their distances. The ball is detected by scanning through the pink (The ball in actual Robocup play is yellow). regions and determining which ones have the greatest chance of being the ball. First and foremost the ball must be on the

ground and no more than five degrees above the robot's head. It also must be surrounded by the field. Anything that is not sufficiently wide, tall, or large enough is disgarded. With these elements in mind the best option in the region structure can be chosen. The known size of the ball can then be used to calculate how far from it the robot is. The kinematics of the robot can also be used to decide what angle from the robot the ball is. To test up to this point I created a ball tracking piece of code that allows the head to look around for the ball until it is found and then maintains focus on the ball, assuming it continues to match the criteria. Once again the gears were turned down in this testing stage to maintain the robot's condition. The robots were able to identify the ball versus a post-it note of similar color. If the colors are too close in nature, then the robots begin to accept, but then start to look away. In the Robocup setting this recognition would suffice as there as there would be no one flashing a post-it note in front of the robots during the matches!

The remaining part of the localization and object identification remain untested. This is due to the difficulty in the actual construction of a field. I am currently working towards putting together a field. The verification of goal recognition seems to be the most reasonable to verify (as it is straight forward) and, due to the goal's large and non-moving nature, seems to be an easy target. In addition, the land marks which would be needed to construct the world model of the field are also easy to detect, as they are the only objects of particular colors on the field. Thus finding them in an image and calculating their distance

is not challenging, given that I have the formulas that have been defined in a paper by CMU.

The position of the ball relative to the robot is calculated by taking the current position of the dog's joints, the position of its head, the ball's size in height (i.e. pixels) and the ball's actual height. Geometry is then used to calculate the position of the ball. This method, however, may be subject to some inaccuracy when the ball is not fully visible, resulting in the difficulty in detecting the height the ball, and thus preventing an accurate reading of the location.

The markers on the field are found by considering the pairs of colors that are associated with them—pink combined with yellow, green, or blue regions). Each marker has a unique sequence of 2 colors on it. This way it is reasonable to attain where on the field a robot is located. The largest ten regions of these colors from our object structure are analyzed to determine if they are vertically adjacent. If any two of them are, they are considered to possibly be landmarks. If the size of both regions are the same, the regions represent a landmark. Once the location of the marker and its size, etc. are found, the position of the robot on the field can then be triangulated. Because we know that the landmarks are ten centimeters tall, and information about the robot's camera height is known, the distance to the landmark, and therefore the position of the landmark relative to the robot, can be calculated. See Figure 4.

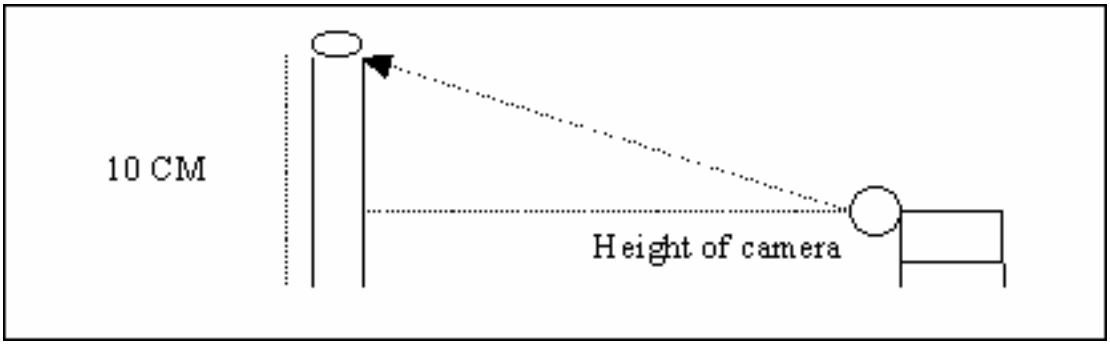

Figure 4 – Calculation of positions relative to landmarks

Although finding the large yellow area on the field (i.e the goal) is relatively simple, goal detection is a highly challenging aspect to robotic soccer because particular locations need to be aimed at on the goal to maximize the chance of a goal. The method used by CMU uses the finding of the corners of the goal to use as targets. For simpler calculations other than those for shooting, I simply looked at the location of the back wall of the goal. By locating the goal, a better plan of attack can be calculated and it is then possible to determine a rough field position. To find the goal, as stated earlier, simply search through the list of objects that has the largest yellow region. If one of these regions is of a reasonable size, has white walls next to it and has green beneath it, the yellow region is in fact the goal. The white area is the white walls of the field, the green being the actual rug color (which is dependant on location of the field).

Robot detection in Robocup utilizes uniforms that more or less cover a large portion of the dog. These uniforms are pieces of blue or red cloth, depending on the team's color. Because we attempt to merge large regions with a particular density level, hopefully even larger regions that correspond to other robots can be generated. In this process we need to look only at regions of red

and blue nature. We want to look at each line of the video and correlate all similar red or blue regions. Although we want to be insensitive to some noise, thin lines of green must be noticed and marked, because this would imply more than one robot. At this point we can look to determine the size of the proposed robot and determine if it is too large to be just one robot. If it is, then we throw out the region.

A visual sonar system was not implemented by me, but CMU does do this. This system would greatly aid in the decision making processes. This system can be implemented by determining which objects exist at five degree intervals. Routes and paths for robots, passes and shots can then easily be chosen from the outputs of this type of system.

For localization, I implemented CM-Pack's Probabilistic Constraint Based Landmark Localization (PCBL). The process is fairly simple. I implemented a somewhat watered down version for speed of calculation and speed of programming. I initialized the location of an individual robot to be the center of the field (all dimensions are in centimeters—see field dimensions for Robocup league). On sight of a landmark, the location of the landmark is returned by the vision. Its size can then be used to triangulate the distance to the marker, and because we have enough information to calculate the distance to the object, we can associate a relative distance to that landmark. As soon as a separate landmark becomes visible as the dog's head looks around, the distance from that object is calculated and the resulting new position can be found. Although not implemented, the robots could easily announce to one another their current

position, and then with that information, could add each other to the world model. However, opponents must be located purely through a robot's own vision. Should a robot lose track of the ball for more than 5 seconds it can request the ball's location from the other robots. From them the average of the other's vision can be assumed, until the ball is located yet again. Each time an opponent is viewed; its position is translated into the world model and updated. There is no attempt at guessing the next position.

Higher level decision making has not been implemented at the current time, as this world model system has not been tested to date. However, motivation can easily be placed on the robot to see the ball and kick it into the goal. This is the target work to be accomplished before presentation time.

5 Future Work

There is obviously much work left to do to actually have a Robocup ready team. My vision includes addressing many of the soccer issues. It would also rely merely on a shared vision of the world. With the wireless capabilities of the robots, their own view of the world can be shared and therefore an agreed upon world can be obtained. The dogs could then easily make their own decisions, assuming they all follow the same protocol for decision making.

In particular I intended to utilize my soccer knowledge and the notion of creating triangles on the field as passing options. Thus I intended to have the roles of each player dynamic as is true in human soccer, minus the role of the

goalie. This way you can never be sure where the attack will come from and 'guarding' one dog or path will not suffice to prevent scoring opportunities.

Soccer, fortunately, is a simple game. Should I have had time to implement this area, I feel that I would have had tremendous success as my understanding of the fundamentals of soccer is high. Perhaps if I were working with more people implementing these robots, a higher level of excitement may have occurred.

On the topic of the world model, it is possible to guess at the future locations of opposing robots. This could be guessed at by looking at the robot's current position and velocity. What remains to be seen is if there is a reliable way of calculating those parameters from an already moving robot's camera. This could aid in the decision making process. In addition, before making an action, the dogs could verify the location of an opposing dog, and have a back-up plan be ready if the first one does not look like a good option. However, the amount of time required to propagate the necessary information may be too great to accomplish. As processors continue to speed up, perhaps last second decisions can be made, like those of an actual human soccer player.

One of the largest slow-downs occurred as a result of not having debugging tools that many of the other teams utilize. University of New South Wales has a beautiful Java graphical user interface (GUI) that shows the state of the robot, what it is seeing, what its model of the world looks like, what it is trying to do with the ball next, and even more. Of course I would have loved to have been able to use this GUI, but clearly this would have forced me to use their

exact system of representation. This limitation would have put more work on me than I already had, so instead I chose an original route. For future work, I think a large debugging tool, preferable written in Java (to allow for cross platform use) would be highly useful**.** This also means standardizing many of the models and outputs for each internal module of the robots.

A simple telnet approach where various pieces of information can be printed would have been another useful debugging tool. This is clearly a much simpler approach than the GUI approach. It could only show string representations, but from a realistic point of view for a single worker, this may be a more realistic.

6 Problems and Technical Issues

The fact is that without years of development not only with the robots and their code, but also of tools, I was at a large disadvantage from the beginning. I faced a new programming environment that is unknown by everyone at Boston College. My voyage was therefore somewhat lonely, but when technical issues came up, it became difficult to remain focused on implementation of other things, when nothing seemed to work correctly. Issues faced ranged from IP issues, DCHP issues, wireless issues, compiler corruption issues, and general issues with Cygwin (Linux embedded into Windows).

The hardest part was getting use to and using the inter-object communication model used in the robots. Without an operating system, the robots were designed on a threaded model. The major drawback is that each

object is only capable of communicating with other objects through shared memory and a system of event driven activities. Without good documentation on this (it was not available until December), trying to do multiple objects was impossible.

Some of the technical issues that had to be addressed involved the transfer of code to the robot itself. The time consuming fifteen minute process of compiling and transferring the code was very lengthy. In the end I was able to use an FTP script that sent all needed files over the wireless LAN and then rebooted the robot. This proved tricky, as it is necessary to remember to include so many things to ensure successful reboot.

When one builds an object, there is still a remaining step required to allow the code to run on the dogs. This is the compression of the *.bin files and the movement of them into the proper directory with the proper configuration files. Typically, however, the FTP server is not in this group of files. Therefore I compiled and did all necessary steps on the FTP server binaries, and then included a copy in the Makefile that simply moves the file into the new location for use. I also played with the idea of editing the configuration files, but once those are set once, they do not need to be set again, as they are manually created. Thus there would be little reason to edit a config file via a script, such as OBJECT.CFG, when it only needs to be edited once.

The time lost due to not having the right examples and direction may have been enough to complete more of what I had wished to. However it is clear that someone had to do this work in order for further development on these dogs at

Boston College. The issues addressed through this journey are documented and available so that others may enjoy the fruits of my labors through a challenging and sometimes depressing journey.

7 Conclusions

I feel that my work, given the stresses I faced, were remarkable. All completed modules show signs of proper implementation, and thus I feel that with sufficient time I could have developed a system that would have been respected and could have been entered into the Robocup competition. Furthermore, my understanding of a wide range of robotic and real time issues has grown, and I too as a person have grown through facing what often seems as an impossible mission.

References

[1] J. Bruce, T. Balch, and M. Veloso. CMVision. (http://www.coral.cs.cmu.edu/~jbruce/cmvision/).

[2] J. Bruce, T. Balch, and M Veloso. Fast and inexpensive color image segmentation for interactive robots. (http://www.ri.cmu.edu/cgi-bin/tech_reports.cgi)

[3] Pedro Lima, Tucker Balch, Masahiro Fujita, Raul Rojas, Manuela Veloso, and Holly A. Yanco. Robocup 2001: A report on research issues that surfaced during the Competitions and Conference. IEEE Robotics and Automation Magazine, June 2002

[4] Bernhard Hengst, Darren Ibbotson, Son Bao Pham, Claude Sammut. Omnidirectional Locomotion for Quadruped Robots. School of Computer Science and Engineering, University of New South Wales.

[5] Dalgliesh, J. and Lawther, M. (1999). Playing soccer with Quadruped Robots. Computer Engineering Thesis, University of New South Wales.

[6] Hornby, G. S., Fujita, M., Takamoto, T., Hanagata, O. Evolving Robust Gaits with Aibo. IEEE International Conference on Robotics and Automation. Pp. 3040-3045.

[7] William Uther, Scott Lenser, James Bruce, Martin Hock, and Mauela veloso. CM-Pack'02. http://www.openr.org/robocup/index.html.

[8] William Uther, Scott Lenser, James Bruce, Martin Hock, and Mauela veloso. CM-Pack'01. http://www.openr.org/robocup/index.html.

[9] Maayan Roth, Douglas Vail, and Manuela Veloso. A World Model for Multi-Robot Teams with Communication. School of Computer Science, Carnegie Mellon University.

[10] OPEN-R SDK: Programmer's Guide. 2002, Sony Corporation.

[11] OPEN-R SDK: Model Information: ERS-220. 2002, Sony Corporation.

[12] OPEN-R SDK: Installation Guide. 2002, Sony Corporation.

Appendix A – Example Makefile with highly automated process

```
#
# Copyright 2002 Sony Corporation
#
# Permission to use, copy, modify, and redistribute this software for
# non-commercial use is hereby granted.
#
# This software is provided "as is" without warranty of any kind,
# either expressed or implied, including but not limited to the
# implied warranties of fitness for a particular purpose.
#

PREFIX=/usr/local/OPEN_R_SDK
INSTALLDIR=../MS
CXX=$(PREFIX)/bin/mipsel-linux-g++
STRIP=$(PREFIX)/bin/mipsel-linux-strip
MKBIN=$(PREFIX)/OPEN_R/bin/mkbin
STUBGEN=$(PREFIX)/OPEN_R/bin/stubgen2
MKBINFLAGS=-p $(PREFIX)
LIBS=-lObjectComm -lOPENR
CXXFLAGS= \
        -pipe \
        -O2 \
        -I. \
        -I$(PREFIX)/OPEN_R/include/R4000 \
        -I$(PREFIX)/OPEN_R/include
TARGET=robot.bin
MSDIR=/flash
INSTALLOBJS=$(TARGET) \
        ../actuators/actuator.bin \
        ../sensors/sensor.bin \
        ../SoundPlay/SoundPlay/sndplay.bin \
        # ../ImageViewer/JPEGEncoder/jpegen.bin
INSTALLMYCFS=../conf/camera.cf \
        ../conf/mycolor.cf \
        ../conf/*.cdt
INSTALLCFGS=connect.cfg object.cfg designdb.cfg

#
# When OPENR_DEBUG is defined, OSYSDEBUG() is available.
#
#CXXFLAGS+= -DOPENR_DEBUG

.PHONY: all install clean
```

```
all: robot.bin

installms: all
	(cd ../actuators;	${MAKE})
	(cd ../sensors;		${MAKE})
	(cd ../PowerMonitor/PowerMonitor;	${MAKE})
	#	(cd ../ImageViewer/JPEGEncoder;		${MAKE})
	cp -r $(PREFIX)/OPEN_R/MS/WCONSOLE/nomemprot/OPEN-R
$(MSDIR)
	cp $(INSTALLOBJS) $(MSDIR)/OPEN-R/MW/OBJS/
	cp ../PowerMonitor/PowerMonitor/powerMonitor.bin $(MSDIR)/OPEN-
R/MW/OBJS/POWERMON.BIN
	cp $(INSTALLCFGS) $(MSDIR)/OPEN-R/MW/CONF/
	cp wlanconf.txt $(MSDIR)/OPEN-R/SYSTEM/CONF/
	mkdir -p $(MSDIR)/OPEN-R/APP/CONF/
	cp -r ../SoundPlay/SoundPlay/wav $(MSDIR)/
	cp $(INSTALLMYCFS) $(MSDIR)/

%.o: %.cc
	$(CXX) $(CXXFLAGS) -o $@ -c $^

robot.bin: ShooterStub.o Shooter.o blob.o LoadTbl.o LoadCDT.o
CameraParam.o  Robot.o robot.ocf
	$(MKBIN) $(MKBINFLAGS) -o $@ $^ $(LIBS)
	$(STRIP) $@
	gzip $@ && mv $@.gz $@

install: robot.bin
	cp robot.bin $(INSTALLDIR)/OPEN-R/MW/OBJS/CAMERA.BIN

clean:
	rm -f *.o *.bin *.elf *.snap.cc
	rm -f ShooterStub.cc ShooterStub.h def.h entry.h
	rm -f $(INSTALLDIR)/OPEN-R/MW/OBJS/CAMERA.BIN

ShooterStub.cc: stub.cfg
	$(STUBGEN) stub.cfg

Shooter.o: Shooter.cc Shooter.h ../lib/Robot.h
	$(CXX) $(CXXFLAGS) -c Shooter.cc

Robot.o: ../lib/Robot.cc ../lib/Robot.h ../include/DRX900.h ../include/myfatfs.h
../lib/LoadTbl.h ../lib/LoadCDT.h
	$(CXX) $(CXXFLAGS) -c ../lib/Robot.cc

LoadTbl.o: ../lib/LoadTbl.cc ../lib/LoadTbl.h
```

```
	$(CXX) $(CXXFLAGS) -c ../lib/LoadTbl.cc

LoadCDT.o: ../lib/LoadCDT.cc ../lib/LoadCDT.h
	$(CXX) $(CXXFLAGS) -c ../lib/LoadCDT.cc

CameraParam.o: ../lib/CameraParam.cc ../lib/CameraParam.h
	$(CXX) $(CXXFLAGS) -c ../lib/CameraParam.cc

blob.o: ../lib/blob.cc ../lib/blob.h
	$(CXX) $(CXXFLAGS) -c ../lib/blob.cc
```

Appendix B – WLAN configuration on AIBO at BC (Fulton Hall)

HOSTNAME=AIBO1
ETHER_IP=136.167.127.252
ETHER_NETMASK=255.255.254.0
IP_GATEWAY=136.167.126.1
ESSID=RoamAbout Default Network Name
WEPENABLE=0
APMODE=1
CHANNEL=6

HOSTNAME=AIBO2
ETHER_IP=136.167.127.251
ETHER_NETMASK=255.255.254.0
IP_GATEWAY=136.167.126.1
ESSID=RoamAbout Default Network Name
WEPENABLE=0
APMODE=1
CHANNEL=6

HOSTNAME=AIBO3
ETHER_IP=136.167.127.250
ETHER_NETMASK=255.255.254.0
IP_GATEWAY=136.167.126.1
ESSID=RoamAbout Default Network Name
WEPENABLE=0
APMODE=1
CHANNEL=6

HOSTNAME=AIBO4
ETHER_IP=136.167.127.249
ETHER_NETMASK=255.255.254.0
IP_GATEWAY=136.167.126.1
ESSID=RoamAbout Default Network Name
WEPENABLE=0
APMODE=1
CHANNEL=6