Sony AIBO

Senior Project Document

Jeremiah Anderson
Lindsey Schurig
Alex Diricco

May 20th, 2006

Table of Contents

1. Project Background

1.1 Institute for Research in Intelligent Systems

IRIS oversees the Intelligent Systems Laboratory, and in doing so, handles policies and management issues pertaining to ISL. IRIS serves as an advocate for curriculum, research, outreach, and projects related to the ISL and its mission. As a result, IRIS provides high-quality regional, national, and international research and instructional services in intelligent systems design, analysis, and implementation.

The Institute for Research in Intelligent Systems' mission is to lead the California State University system in the discovery, development, analysis, and integration of accessible intelligent systems research and technologies (e.g. autonomous robotics applications) for use in our community and the industry. IRIS has a number of objectives, as referenced from the website:

> Manage the use of the ISL and all equipment under the jurisdiction of ISL (i.e. all instrumentation and equipment secured through ISL-related contracts and grants);
>
> Foster collaborative work within the College of Engineering, Computer Science, and Construction Management (ECC);
>
> Assist in forming partnerships for funded projects with regional industries, federal, state, and local agencies;
>
> Serve as a liaison to constituents when opportunities arise to use the ISL as a resource for instruction or research;
>
> Outreach and recruitment for the ISL and related curriculum/majors;
>
> Oversight of camps, seminars, workshops, and curriculum decisions related to the ISL and its mission;
>
> Provide a venue for mentoring and guiding student projects and competition entries;
>
> Provide a venue for assisting in funding and travel opportunities for competitions, conferences, and workshops; and dissemination of ISL- and IRIS-related news and activities.

1.2 Intelligent Systems Laboratory

The Intelligent Systems Laboratory (ISL) of the Institute for Research in Intelligent Systems (IRIS) is designed to facilitate the development of cross-disciplinary courses and provide exciting collaborative research possibilities. The ISL enables students and faculty to work together in the investigation, design, and implementation of control algorithms using non-traditional techniques from sub-disciplines of Artificial Intelligence, such as fuzzy logic, neural networks, genetic algorithms, hybrid approaches, etc. The ISL also provides students with the opportunity to work with people from other disciplines. This collaborative work gives students the experience of working with non-majors on a joint project, an experience they will all need to be successful in their careers. The robotics equipment acquired for the ISL is to be used to support instruction of both undergraduate and graduate students through new and recently modified courses in the areas of Machine Intelligence, Intelligent Systems Design and Applications, Intelligent Control, Autonomous Robots, and others. Robotic kits at the basic, intermediate, and advanced levels are used to facilitate research, research training, and integrated research/education activities at various academic levels.

The ISL is partially funded by National Science Foundation (NSF) Major Research Instrumentation (MRI)/Research in Undergraduate Institutions (RUI) grant EIA-0321385 for 2003-2006. Additional funding is from the constituency of the College of Engineering, Computer Science, and Technology.

1.3 Robocup

The Robocup is a yearly international competition where autonomous robots compete in real-life soccer games. Different leagues exist within the Robocup, including a four-legged league, which uses as its platform the AIBO robot from Sony. Sony developed the AIBO originally as a pseudo-household pet, but upon demand from hobbyists around the world, created a programming language that allowed individuals to modify the AIBOs behaviors. In the case of the Robocup, university teams from around the world modify AIBOs to play in the competition.

1.3.1 Why Robocup?

The ISL's major goal is to perform research that will lead to the creation of autonomous search-and-rescue robots. Dr. Renner and Dr. Juliano decided that creating an AIBO-based soccer team would help the ISL achieve its ultimate goal by helping the ISL team gain an understanding of visual processing and communication algorithms. With this in mind, a set of ten AIBOs were purchased and investigation into AIBO behaviors began.

1.3.2 Choosing a Codebase

Before beginning work on the project, the team was first faced with a decision on how the AIBOs would be programmed. Various programming languages exist that allow developers to modify AIBO behaviors. These include:

Open-R: A proprietary Sony language based on C++.
R-Code: A proprietary Sony scripting language.
Tekkotsu: A third-party scripting language.
Other Teams: Teams that had already created soccer code.

Considerations for choosing a codebase to use were focused on ease of modification and the ability to use a simulator. The reason ease of modification was desirable is obvious. Being that creating a robot team that plays soccer is a complex task, it was best to choose a codebase that would allow the team to quickly get up and running. The need for a simulator is less obvious, however. Running a game of AIBO soccer utilizes a soccer field nearly 19 feet long and 12 feet wide. Because Chico has a distinct lack of space, the ISL team does not have full time access to a large room where a field can be permanently setup. By having a simulator, behaviors can be modified and studied immediately after compilation of code. Additionally, Sony recently discontinued production of the AIBO platform, and having a simulator reduces wear and tear on the real-life robots.

With these factors in mind, the team decided to use the German Team code as a basis to begin the project. The German code is split into two areas, Open-R code, and YABSL. The YABSL code is used to modify robot behaviors and is fairly easy to understand. The team targeted this as the area that would be modified initially. Also, the German Team included tools with the code that would make the software development process easier to handle. A simulator was a part of this package. An additional plus was that the German Team had already competed in and won multiple Robocup contests.

2. AIBO Design

2.1 Specifications

The AIBO is an autonomous robot, designed to be an alternative household pet. It can also be modified to perform other tasks, such as playing soccer. The following are the hardware specifications:

CPU 64 bit RISC processor
Input/Output PC Card Slot Type 2 In/Out
AC in Power Supply Connector Input
Image Input CMOS Image Sensor (300K pixel)
Audio Input Miniature Stereo Microphones
Audio Output Miniature Speaker
Built-in Sensors Temperature Sensor
IR Distance Sensor, Acceleration Sensor
Pressure Sensors (head, face, back, legs and tail)
Vibration Sensor
Power Consumption approx. 9W (autonomous mode)
Battery Charging Time approx. 2 hours
LCD Display Time, Date, Volume, Battery Condition

2.2 Sensors

Sony designed AIBO with four main categories of sensors, including tactile, auditory, visual, and balance. Below are some examples of the uses of each of these four types of senses in a soccer game environment.

2.2.1 Tactile

The head and back sensors are utilized to transition the AIBO into different game modes. The AIBO has six states: Initial, Ready, Set, Playing, Penalized, and Finished. The AIBO also has tactile sensors on its feet, allowing it to realize when it has bumped into an object or another player.

2.2.2 Auditory

The AIBO has microphones in what would be considered the ear area, giving the AIBO the capability to listen to its surrounding environment. It also has a speaker, from which it can "bark" when it's happy or upset. These sensors are not being currently used in the game environment.

2.2.4 Visual

The AIBO has a color camera, allowing it to the detect color, shape and movements of nearby objects. The AIBO's ability to see certain objects and colors can be modified by using the German Team

utility Robot Control. For example, to play soccer, the AIBO needs to see the orange ball, color-coded positioning flags, and colored-coded goals. Other AIBOs on the field must also be detected. Robot Control is utilized for color calibration of the AIBO camera. By modifying what colors the AIBO considers “orange” for example, game ball detection is improved by allowing the AIBO to see more distinctly the area of orange when looking at the ball.

2.2.5 Balance

The AIBO has a number of sensors, which allow it to detect distance from itself to other objects. The head distance sensor measures distance between the AIBO and other objects, as well as the distance between the AIBO’s head and the ground. The chest distance sensor measures the distance between the AIBO and other objects on the field. This is useful to determine how far the AIBO is from the ball, from other dogs, and from any objects that may be on the field.

2.3 Vision

While detecting objects within the confines of the field is difficult, it is an attainable goal since important objects are identified by specific colors. Additionally, the layout of the field objects remain static. The main objects that the AIBO needs to detect on the field are the ball, flags, goals, field lines, and other dogs. Obviously, detecting these objects can pose a problem because of varying factors. The main factor that can affect the way the AIBOs detect objects is light. There are certain specifications for lighting during competitions, but in some cases fields will not have the correct lighting, necessitating adaptable code. Even so, it can be difficult for the dogs to detect objects under less-than-optimal lighting conditions. Below are descriptions of the ways in which the AIBO detects objects:

2.3.1 Detecting the Ball

The ball for soccer is bright orange, making it easier for the AIBOs to detect. To find the ball, each player is constantly scanning the field. A large mass of orange pixels is considered to be a possible ball. Possible balls are scanned in four directions: horizontally, vertically, and both diagonal directions for the edge. The scan stops, and the possible ball is considered to be a ball if the last 8 pixels belong to the green, yellow, or sky-blue color classes. From here, the dog calculates the center and radius of the ball, and sends this information to the team. The dogs communicate with each other the placement of the ball with the 'Team Ball Locator'. Each dog sends the position of the ball relative to itself, as well as the calculated position and speed to all team members. From here, each player can add a time update, compensating for the fact that

the ball has most likely decreased in speed since the information was sent. The dogs have three states they can be in with regards to the ball: 'Ball was Seen', 'Ball was not Perceived for a Short Period of Time', and 'Ball was not Seen for Some Time'.

2.3.2 Detecting Field Flags

The field flags are necessary for the AIBOs to realize where they are in position on the field, in other words, for self-localization. The camera scans the horizon looking for pink pixels. If a large area of pink is found, this is clustered and merged, in order to determine the center of the flag. The dog then scans above and below the pink area, looking for the other color of the flag, either blue or yellow. The distance between each section of the flag is calculated, and if the values are the correct distance apart, the object is determined to be a flag. Depending on which color is on top - blue, yellow, or pink, the dogs can determine which part of the field they are on.

2.3.3 Detecting Goals

To play soccer, the AIBOs need to be able to detect where the goals are. The goals and the flags are the most important objects to detect for self-localization. To detect the goal, the dog knows pre-determined items about the goal, including the fact that the left and right edges are straight and parallel, and the height of the goal. The dog also knows the goal area is likely to be blocked by the goalie and possibly other dogs or objects. The dog scans the field for any goal colored areas, blue or yellow. Once a possible goal is found, to determine the goal's properties, the dog scans the lines on the outside of the goal to find the edges of the area. When finding the edge, the line is followed until it crosses an edge, or the color deviates too much from the ideal color (meaning there is either an obstruction, or shadow). If the area is determined to be a goal, the dog takes the required action.

2.3.4 Detecting Field Lines

In order to know the edges of the field, the dogs must be able to determine where the lines are marking the outside of the field. The dogs find the lines by first looking for areas of white. Depending on the size of the white object, if it is too large or too small, the AIBO can rule it out. The dog will calculate the distance between the white and itself, knowing the height and rotation of the camera, and adding its perspective. If there is green around the white object, and it is the right size, it is determined to be a field line.

2.3.5 Detecting Other Dogs

A team can work more effectively when they know the locations of opposing team members on the field. In order to detect the other teams' players, the AIBO first scans the field for three points of color together in the opponent's colors. If the other team is blue, the dog will scan for points of blue and white together. From there, the AIBO tries to find the foot point of the robot, and calculates the distance relative to itself.

2.4 Open-R Code

Open-R is Sony's proprietary C++ based programming language for AIBO robots. Functions can be programmed using standard C++ code, and standard C++ libraries can be used. However, there are also some major differences. The three primary differences are in the way AIBO code is executed, the way data is passed, and the special code used to interface with the robot's sensors and servos.

2.4.1 Open-R Objects

Most standard software today uses code that is executed sequentially. What this means is that when a line of code is being executed, it can be known exactly which will be the next line of code to be executed. In Open-R, however, this is not the case. Instead, Open-R uses the concept of Objects to allow for code to be executed in a non-sequential order. Here, the concept of 'Objects' does not refer to standard C++ class objects. Instead, Open-R Objects are individual sets of code that are executed by the AIBO processor in a near-simultaneous fashion. Processor time is devoted to Open-R Objects in what could be described as a multi-tasking manner; similar to the way Windows divides processing time among multiple applications.

An example: There are two OPEN-R Objects. One controls decisions while the other controls movement. If the decision object decides to move the robot, it will pass data to the movement object telling the robot what movement to perform. The moving object will then begin to execute moving code. During this time, the decision object is still active. If it decides that the robot needs to move in a different direction, it will send a message to the moving object notifying the moving object that it has new data for it to process. When the moving object is ready to receive new data, it will go ahead and grab the data from memory and begin to execute the new instructions. This process repeats until the AIBO is shut down.

The main advantage of such a system is that it allows the AIBO to perform multiple functions simultaneously. If AIBO code had to be

executed in a sequential manner, it would not be able to process any other information as it was executing walking code. Clearly, a non-object oriented system is not ideal for autonomous systems.

2.4.2 Data Passing

Unlike C++, where arguments can be passed between functions simply by putting them in-between parenthesis, the Object structure of Open-R forces data to be passed in a different manner. Since Open-R Object code executes simultaneously, if an object sent data to a currently busy object, the busy object would not have the ability to receive the data, thus losing the data. While the sending object could constantly send the data over and over again until the receiving object was ready, this would not be the most efficient use of processor time and the data pipeline.

Instead, Open-R passes data between objects through a three-step process. First, a sending object moves the desired data to be sent onto a special memory area on the AIBO. This area consists of a special C++ struct that must be initialized ahead of time. Then, the sending object sends a much smaller message directly to the receiving object that simply tells the receiving object data is waiting for it. When the receiving object is no longer busy, it knows it has a message waiting for it and retrieves the message from memory.

2.4.3 Sensor Interface

The last major difference between Open-R and C++ involves the way code interfaces with the on-board sensors. Sony decided to allow developers to directly interface with the hardware. While this allows developers to have more flexibility with the AIBO, it presents a couple of problems. The first problem is that there are no checks in place to make sure that code does not try to cause the AIBO to do something that would harm the servos. A coding mistake can easily cause a servo motor to burn out. The other main problem is that this ability adds a large amount of complexity to the coding of an AIBO program. Hardware must be initialized directly using hardware IDs and multiple lines of obtuse set-up code. The lack of code commenting in the Sony provided sample code makes this task even more difficult.

Of course, the advantages are that AIBO soccer-playing performance can be pushed at the possible expense of shorter servo lifetime. Additionally, once the team understands the code structure better, modifications to sensor code will become easier.

3. Environment Setup

To this point, the major tools from the German Team that are being utilized are the .Net project, Makestick, Simulator, and Robot Control.

3.1 .Net (WinXP)

The code is set up so that .NET is used to compile the Open-R code, the YABSL behavior code, and all the tools that are provided by the German Team.

3.2 Makestick

Makestick is used to write the executable code to a Sony memory stick. The stick is then inserted into the dog. When one is writing the code to the stick they can specify the player role (goalie, player2, player3, player4), pick the team color (red or blue), and set the network related protocols that the dogs need if they are to be controlled by Robot Control or communicate with each other.

3.3 Simulator

The simulator is used to test code changes out before they are written to the dogs. The simulator simulates how the dogs theoretically will act when they are playing in real life. Due to current era processor limitations this has not a viable full-game simulation option as running all eight dogs brings the simulator to a crawl. However, it is useful for observing dog-on-ball interactions and interactions between two to three players (goalie vs. opponent, for example).

3.4 Robot Control

Robot Control is used when the dogs are playing an actual game. It allows the dogs to communicate over a wireless network. It can also be used as a game control console. Dogs can be penalized or placed into ready, set, or play modes from the control console. There are other options that have yet to be explored in-depth. Currently, the lack of testing space and dog hardware connection problems have hampered efforts to obtain a better understanding of the tool.

4. XABSL / YABSL

XABSL (eXtensible Agent Behavior Specification Language), or YABSL as it is now in its second phase, is responsible for decision-making based on the state of the dogs. This language provides the team with the capability to modify behaviors without having the change the German Team's Open-R code. Behaviors can be modified through the YABSL files. YABSL files are compiled and converted to XABSL, which are then converted to XML. The German Team Open-R code uses the resulting XML files as a basis for creating behaviors. YABSL uses hierarchies of behavior modules called options that contain state machines to make decisions. These state machines manage the transitions to new behaviors depending on the last state and the recent situation the dog is encountering. There are four modules in the YABSL architecture, Agents, Options, States, Basic Behaviors, and Symbols as detailed below.

4.1 Agents

There is one agents file, which contains the default soccer agent behavior. This file is the root document of YABSL behavior, includes all options, and defines all agents. An agents file has the following syntax:

<agents definition file> ::=

```
/***
Title:                  <title>
Platform:               <platform>
Software-Environment:   <software-environment>
*/

{include "<include file>";}
{
        <doc comment> agent <id> ("<title>", <root-option>);
        <doc comment> agent <id> ("<agent-title>", <root-option>);
}
```

There has to be at least one agent element inside the agents definition file. Each agent element must include the id, agent-title, and root-option. The id is the id of the agent. This must be used to select that agent. The agent-title is need for documentation purposes, and the root-option is the root option of each agent.

4.2 Options

Each option has to be defined in a separate file. Syntax for the option files is as follows:

```
<option file> ::=
{include "<include file>"; }
<doc comment> option <name>
        {
                { <parameter> }
                [ <common decision tree> ]
                <state>
                { <state>  }
        }

<parameter> ::=
<doc comment>
        float @<name> [[<range>]] ["<measure>"] | bool @<name> |
enum <enumeration> @<name>;

<common decision tree> ::=
 common decision
 {
        <doc comment> if ( <boolean expression> ) <decision tree>
        {
                <doc comment> else   <doc comment> if ( <boolean
                expression> )   <decision tree> |
                <doc comment> else { <doc comment> if ( <boolean
                expression> ) } <decision tree>
        }
 }
```

Name is the name of the option, and must be the same as the option file name. For the parameter, Name must include the @ symbol. Range and measure are the values of the decimal range and measure parameter values. If the parameter is enumerated, this must be specified.

4.3 States

Each state represents a single state option that a dog can be in at any moment in time. The syntax for a state:

```
<state> ::=
[initial] [target] state <name>
{
        decision
        {
                [else] <decision tree>
        }
        action
        {
                { <output symbol> = <decimal expression> | <boolean
                expression> | enumerated expression> ;}

                [ <option> [ <parameter list> ] ; | <basic behavior>
                        [ <parameter list> ] ; ]
        }
}
```

Initial or target are the first state of the option. Name is the state name. The decision tree defines the state transitions. If the else branch after the decision tree is taken, this means no transition was returned from the branch to the decision tree. Output symbol controls the output symbol to be set during this state. The option and basic behavior detail the basic behavior to be executed during this state.

4.4 Basic Behaviors

Basic behaviors are written in C++. YABSL basic behavior files are used to control when these behaviors are used. Example syntax:

```
<basic behavior definition file> ::=
{ include "<include file>"; }
<doc comment> namespace <id>("<title>")
{
        { <behavior> }
}

<behavior> ::=
<doc comment> behavior <name>
[
        {
                { <parameter> }
        }
];

<parameter> ::=
<doc comment>
float <name> [[<range>]] ["<measure>"] | bool <name> | enum
<enumeration> <name>;
```

Id must be the same as the file name, and is the name of the basic behavior. Name, range, measure and enumeration are required in decimal if these values apply. There are a few basic behaviors, including go-to, get-behind-ball, and simple-action. The details of each are below:

Basic Behavior "go-to": Lets the agent move to a point

Parameter	Measure	Range	Description
go-to.x	px	1..78	X of destination position
go-to.y	px	1..22	Y of destination position

Basic Behavior "get-behind-ball" : Approaches the ball and gets behind it

Basic Behavior "simple-action": Performs a basic action as moving or kicking

Parameter	Measure	Range	Description
simple-action.action	int	0..10	The id of the action to be performed

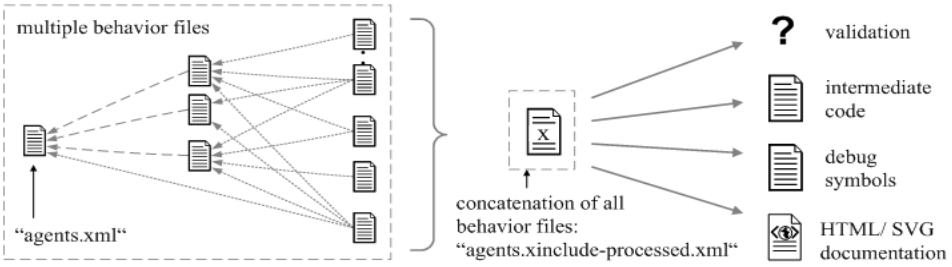
4.5 Symbols

```
<symbol definition file> ::=
{ include "<include file>"; }
<doc comment> namespace <id>("<title>")
{
        { <enumeration> | <input symbol> | <output symbol> |
        <constant> }
}

<enumeration> ::=
<doc comment> enum <name>
{
        <enum-element>
        { <enum-element> }
};

<input symbol> ::=
<doc comment>
float input <name> ["<measure>"] | bool input <name> | enum
<enumeration> input <name>
[ (
        { <parameter> }
) ];

<parameter> ::=
<doc comment> float <name> [[<range>]] ["<measure>"] | bool
<name> | enum <enumeration> <name>;

<output symbol> ::=
<doc comment>
float output <name> ["<measure>"] | bool output <name> | enum
<enumeration> output <name>;

<constant> ::=
<doc comment> float const <name> = <value> ["<measure>"];
```

Name defines the name of the symbol, range and measure are the range and measure of the symbol if applicable. Value is the decimal value of the symbol, and enumberation, has to be specified if the symbol is enumerated. The common input symbols are listed below:

Name	Type	Measure	Description
player-role	enumerated	defender midfielder striker	The dynamically assigned role of the player
x	decimal	px	The x position of the player
y	decimal	px	The y position of the player
ball.x	decimal	px	The x position of the ball
ball.y	decimal	px	The y position of the ball
ball.distance	decimal	px	The distance to the ball
ball.local.direction	decimal	int	The direction of the ball if in the local area of the player. If the ball is not in the local area, -1 is returned
most-westerly-teammate.x	decimal	px	The x position of the most westerly teammate

4.6 Modifying Behaviors



Basically, the option exists to play soccer, which is contained in the agent file. From there the robot can choose from three different states: midfielder, striker, or defender. The midfielder handles the ball, and can choose from two states, pass or kick. The defender waits behind the ball, and can choose from the two states, own-team-has-ball or opponent-team-has-ball, and from there choose from other states depending on which is determined to be true. The striker waits near the opponent goal in front of team players for a pass in an attempt to create a goal scoring opportunity. In these various options and states, each robot has basic behaviors that can be performed. Some example basic behaviors include go-to-ball and kick. Additional examples of state transitions are listed below. There are a number of different options an AIBO can be in at a given moment, and under each option, states to choose from. Options are bold, states are plain font, and basic behaviors are italics.

play soccer -> **striker, defender or midfielder**
striker -> go to
defender -> go to
midfielder -> **pass, dribble**, or go to
pass -> simple action or get behind ball
dribble -> simple action or get behind ball

play-soccer - the soccer root behavior
- state machine
- state striker
- state mid-fielder
- state defender

midfielder - handles the ball
- state machine
- state get-to-ball
- state pass
- state dribble

pass - passes the ball to a teammate
- state machine
- state get-behind-ball
- state kick

dribble - dribble the ball without kicking
- state machine
- state behind-ball
- state behind-ball-near-opponent-goal
- state not-behind-ball

striker - waits in front of other players for a pass
- state machine
- state initial

defender - waits behind ball
- state machine
- state own-team-has-ball
- state opponent-team-has-ball

5. Competitions

There are two major competitions for AIBO soccer. One is the United States Open and the other is the Robocup international.

5.1 United States Open

This years US Open was held in Atlanta Georgia where Brown University, Bowdoin, Dortmund University (German Team), Carnegie Mellon University, UT Austin, and the University of Pennsylvania competed. Dortmund University ended up coming in first place and Carnegie Mellon came in second.

5.2 Robocup International

This year's international competition will be held in Bremen, Germany. This is where the best teams from around the world will compete. Next year the international competition is supposed to be held in Atlanta, Georgia.

6. Project Issues

Most of the problems facing the project so far have been in the area of actual code development. Major issues include time management, setup of the development environment, lack of code commenting, and code compilation errors.

6.1 Time Management

A major issue the team ran across dealt with time management. Due to class scheduling, the team was only able to meet once a week for two hours. This made communication and coordination between team members difficult. Fortunately, the summer break will allow the team to work together much more often.

6.2 Environment Setup

Because the AIBO development is complex, a major portion of time was spent getting the development environment and AIBO tools properly set up. Documentation from both Sony and the German team was somewhat lacking in that department. Many hours were spent getting .Net to properly compile tools and robotics code. Part of the difficulty was based upon the fact that only one individual on the team had had previous .Net experience. Additional time was spent getting proprietary German team tools to work properly. The German team has been helping out over email, however they are often slow to provide responses. Most of the issues have been resolved, however, and once the environment is set up properly, development can begin in earnest (see section 6.4 Code Modification for details on the last environment problem).

6.3 Code/Documentation Issues

Sony provided developers with sample OPEN-R code. This code was used to study how to interface with the AIBO robots, however none of the code from Sony was commented. In OPEN-R, even simple operations need a large amount of code before they will execute. A simple “Hello World” program takes over 200 lines of code. Code for simple servo movements are in the 500-line range. A large portion of project time was spent going through the code line by line, analyzing it, and commenting it. The team has experienced similar problems with the German team code. The code is indeed commented, but comments are sparse and the accompanying documentation only gives information regarding the overall program structure. This will continue to be an issue for the team as it begins to modify new areas of code. Currently, code is being analyzed in small parts as the project progresses.

6.4 Code Modification

After managing to modify and compile behavior code, it was found that the behavior of the AIBO dogs had not actually changed. The team has tried to troubleshoot the problem, but has so far been unable to find a solution. Unfortunately, this as has led to some team downtime. Correspondence with the German Team is underway to resolve the issue. Once this issue is resolved, modification of AIBO code can begin. In the meantime, documentation of the team's findings is underway.

7. What's Next?

7.1 Targeted Behaviors

The three most essential behaviors that the dogs use when they are playing in a game is they need to know exactly where they are on the field at all times, they need to know how to get the ball down the field in an optimal manner and the goalie needs properly defend the goal.

7.1.1 Field Position

Field positioning is probably the most essential part of the game. When the dogs do not know where they are on the field they have a tendency to run off of the field and get penalized for 30 seconds. A dog just lying on the field dead is better then one that is not on the field at all. The dead dog at least stands the chance of blocking the ball at some point.

7.1.2 Goalie Behavior

The goalie behavior is the second most important aspect of the game because without a goalie in the box at all times to defend the goal the other team has a real easy time of scoring goals. The only time the goalie should have even one paw out of the goalie box is when the ball is near the goal and it is making it harder for the opponent to score, but it should always have at least one paw in the box at all times incase the opponent passes to a teammate so the goalie has time to retreat to a better defensive position.

7.1.3 Ball Handling

Ball handling is key so that the team can score a goal. An ideal striker or forward will know how to dribble the ball so that it does not have the ball for too long, know where its teammates are on the field in a passing situation, and know how to make a shot on goal that a) goes toward the goal and b) does not go directly to the goalie for an easy block.

7.2 Alternative Code Bases

Researching alternative code bases is essential to the project to get ideas on how to make the existing German code better. Additionally, it is possible that other code bases may be easier to manipulate. If this is the case, an evaluation will take place as to whether the project should continue to use the German code. Major considerations will include the team's understanding of the German code at the time.

7.3 Future Competitions

The ultimate goal of the AIBO project is to compete in future U.S. and international Robocop competitions. This is a large task to undertake, but can be accomplished with proper planning and time. Behavior modification is currently underway and major progress to fulfilling this goal should be made over the summer.

8. Appendices

8.1 Works Cited

Juliano, Benjoe. Renner, Renee.
Institute for Research in Intelligent Systems.
http://iris.ecst.csuchico.edu/welcome/index.htm.
California State University, Chico.

Juliano, Benjoe. Renner, Renee.
Intelligent Systems Laboratory.
http://isl.ecst.csuchico.edu/welcome/index.htm.
California State University, Chico

The 2005 German Team.
German Team Robocup 2005 Manual
Center for Computing Technology, Universitat Bremen.
Institut for Informatik, Humboldt-Universitat zu Berlin.
Institute for Robot Research, Dortmund University.

Robocup Technical Committee.
RoboCup Four-Legged League Rule Book 2006

Loetzsch. Bach. Burkhard. Jungel.
Designing Agent Behavior with the Extensible Agent Behavior Specifications Language XABSL.
Institut fur Informatik, LFG Kunsliche Intellignez, Humboldt-Universitat zu Berlin.

Loetzsch, Martin. Jungel, Matthia. Risler, Max.
The Extensible Agent Behaivor Specification Language.
http://www2.informatik.hu-berlin.de/ki/XABSL/language.html.
Institut fur Informatik, LFG Kunsliche Intellignez, Humboldt-Universitat zu Berlin.

AIBO.
http://en.wikipedia.org/wiki/Aibo.
Wikipedia

Sony Bots.
http://www.iirobotics.com/webpages/index_result.php?id=291&cat=sony&rafa=1. iRobotics

8.2 Senior Project Proposal

The Problem:

To write XABSL (Extensible Agent Behavior Specifications Language) files, in order to control the AIBO team's behavior to play soccer.

The Approach:

In order to finish this extensive project in one semester, we have decided to take an approach similar to what the Bremen team did last year. We are going to use the Vision, Walking, and other pre-existing, software that the German team has already coded in C++, and only modify the existing XABSL files. As opposed to taking many years to research methods for robot mobility and vision, etc., and writing the thousands of lines of C++ code, we can just focus on controlling the behavior by using XABSL. This makes the project goal attainable in one semester's time.

Through modifying the XABSL files to control behavior, we can map out the desired actions for each player on our team, including the goalie, and defense and offense. We can code how to react to certain situations, which way to kick to make a goal, when to kick, and other related behaviors. Because Sony has discontinued production of AIBOs, we will be using Webots software to simulate games until we have what we feel is code we can use for competition.

The Goal:

Our goal (not only for this semester, but for next year as well) is to make it to the International Robocup in 2007 with much of the XABSL files modified by our own development team. We are going to focus on fine-tuning the dog behavior to play soccer, and integrate the new rules for 2007. The competition will be held in Atlanta, Georgia, and we plan to enter the competition to show off the behaviors that we have implemented in our AIBOs.

8.3 Theme J Capstone Proposal

Alternate Capstone Course - CSCI 498:
Research through the Intelligent Systems Laboratory will present a practically application of artificial intelligence. Students will address issues of knowledge representation and natural language processing from a computational perspective. Students will write and alter existing programs to perform analyses of natural language and logical reasoning on the Sony AIBO robots. Students will explore the limits of computation, using practical and theoretical approaches. Approximately 10.0 hours activity per week.

Course Objectives:
The primary objective of working with ISL is to introduce students to programming techniques employed when designing and implementing applications of artificial intelligence. Emphasis will be placed on the learning of specific software tools, languages, and programming environments on the AIBO Robots. This project will promote a 'hands-on' approach for understanding, as well as a challenging avenue for exploration and creativity.

Specifically:

Gain a perspective of AI and its foundations. (I)
Receive an introduction to the intermediate principles of AI problem solving, perception, knowledge representation, language processing, and learning. (I)
Experience applications of AI within expert systems, artificial neural networks and other machine learning models. (I)
Experience basic programming in C++ and XML as AI logic tools. (I)
Discuss the potential, limitations, and implications of intelligent systems.

B=Basic I=Intermediate A=Advanced

Course Outcomes:
Upon successful completion of this course, students should possess the following:

An appreciation for the implementation of AI
Problem-solving strategies for approaching AI solutions
Intermediate-level skill set for programming in C++ and XML
Beginner-level skill set for using a subset of AI applications and modeling tools

Accreditation Category Content:

Topic	Percentage	Hours
Algorithms	40%	60
Data Structures	00%	00
Software Design	40%	60
Concepts of Programming	20%	30
Computer Organization and Architecture	00%	00

Relationship of Course to Program Objectives:
This course supports the achievement of the following program objectives:

Students will be able to analyze and solve computing problems, or problems in related areas, and to continually upgrade their knowledge and skills. (P)

Students will be effective oral and written communicators and be able to function effectively as members of multi-disciplinary teams. (P)

Students will have an appreciation for the individual, society, and human heritage and they will be aware of the impact of their work on society and the environment. (I)

Those graduates who pursue careers as computing professionals will have the skills to use and design new and innovative systems that meet society's needs. (P)

Those graduates who pursue advanced degrees will have the skills to succeed in graduate programs in computing and related fields. (P)

I=Introduced P=Practiced R=Reinforced