Team Top Dog’s Aibo Robotics Research Report

David Elrod
Jack Liu
Bryan Knight

May, 2005
Table of Contents

Abstract ................................................................................................................................. 3
Introduction .......................................................................................................................... 4
Materials and Methodology ............................................................................................... 5
  Finite State Machine ........................................................................................................ 5
  Tracking Behavior ............................................................................................................ 5
  Encircle Behavior ............................................................................................................ 6
  Kicking Behavior ............................................................................................................. 6
  Send Behavior and Wait Behavior .................................................................................. 7
Results ................................................................................................................................ 8
  Finite State Machine ....................................................................................................... 8
  Tracking Behavior ............................................................................................................ 8
  Encircle Behavior ............................................................................................................ 8
  Kicking Behavior ............................................................................................................. 8
  Send Behavior and Wait Behavior .................................................................................. 8
Discussion ............................................................................................................................ 9
  Finite State Machine ....................................................................................................... 9
  Tracking Behavior ............................................................................................................ 9
  Encircle Behavior ............................................................................................................ 9
  Kicking Behavior ............................................................................................................. 9
  Send Behavior and Wait Behavior .................................................................................. 10
Appendices ......................................................................................................................... 11
  A. Glossary of Terms: ..................................................................................................... 11
  B. Tekkotsu Setup for Cygwin ....................................................................................... 12
References ............................................................................................................................ 17
Abstract

This report covers the implementation of a finite state machine for passing a ball between two Sony Aibos. The methodology of each state is covered, including the tracking of the ball, positioning the Aibo for a kick, the actual kick, and the communication between the two Aibos. Overall results of the system are covered as well as discussion of potential improvements.
Introduction

The purpose of this project was to create a starting point or platform off of which RPI students could build a RoboCup team.

In Robocup soccer, it was observed that there was minimal teamwork among Aibos if not non-existent. From the start of the competition, the Aibo holding the ball would try to run the other defenders including the goalie over in order to score the goal while the other Aibo on the same team would sit back for defense. But what need is there for them to sit back for defense when one of their teammate is on offense? This illustrates the point that Aibos in Robocup do not pass to each other. There was no any kind of passing in order to obtain better field position in real soccer. With that, the purpose of this project is to create a way for one Aibo to pass the ball a few feet in front of an open teammate who tells the other Aibo that it has better field position or has a good look at a shot at the goal. The team calls this lead passing.

The goal of this project is to demonstrate this lead passing ability between two Aibos. In short, the first Aibo holding the ball will pass the ball to another Aibo who is open. However, it won’t pass the ball to where the Aibo is but kick it to a spot where it’s a little bit ahead of the other Aibo. Once the ball is kicked, it will tell the other Aibo that it has kicked the ball and that it should get it. The other Aibo now will look for the ball which will be a few feet in front. The lead passing mechanism is an efficient and quick way to pass the ball. With no lead passing, the first Aibo would kick the ball to where the Aibo is standing. The standing Aibo will have to wait for the ball to reach it before it can do anything with it.

The things that need to be accomplished in order to demonstrate this is to have a kicking mechanism, a way for the Aibo to recognize objects, a way for the Aibo to transition between different behaviors (going from walking up to ball and kicking it) and a way for the two Aibos to communicate.
Materials and Methodology

Finite State Machine

When approaching this undertaking, we saw that the implementation of a finite state machine (also referred to as FSM) could prove useful in the solution. We chose to implement an FSM, because it lends itself to top-down design well. With a generalized idea of what actions we wanted the Aibo to perform and the order we wanted it to perform them in, an FSM seemed the natural conclusion.

Additionally, Tekkotsu has built in support for finite state machines, as well as predefined state-nodes, transitions, and events. Our generalized plan of actions for the Aibo to perform were to have it track the ball, position itself to kick, do so, and then send a message to another Aibo telling it that it had passed it the ball. With this in mind it became easy to split up what behaviors we would need to implement, and what events we would need to have the Aibo watch or “listen” for to transition from one behavior to another.

In Tekkotsu, the FSM itself is contained by a state-node, thus allowing for layering to become a simple task. From there, the FSM contains various state-nodes, defines transitions from specific state-nodes to other state-nodes, and a starting state-node whose behavior gets performed first. For clarity’s sake, the behaviors we defined were called: Track, Encircle/Position, Kick, Send, and Wait. These correspond directly with the generalized plan we had defined early, except that now, one Aibo will wait and do nothing until it receives the message from the other Aibo that its ok to go after the ball.

Although there are many pre-defined events in Tekkotsu, we decided to define our own. This allowed for easier tracking of our code and a quick safe-guard against our behaviors accidentally performing unwanted actions. With our own events defined, the next order of business was to create our FSM’s controlling node. Our state-nodes there contained our behaviors, which looped as such:

Track \rightarrow \text{Encircle/Position}
\text{Encircle/Position} \rightarrow \text{Kick}
\text{Kick} \rightarrow \text{Send}
\text{Send} \rightarrow \text{Wait}
\text{Wait} \rightarrow \text{Track}

Seeing as how we had two Aibos, we actually had two FSM’s that were identical, other than that one’s starting state-node executes the Track behavior, and the other’s starting state-node contains the Wait behavior.

Each behavior itself defines a condition, which if met, declares that an event has happened. If the event is one that the FSM is watching or “listening” for, then it will make the appropriate transition to a different behavior, different behaviors, or termination.

Tracking Behavior

To approach the ball, the Aibo’s camera would be utilized to locate an object to position the head and the body such that it will approach the object straight on. Once the
ball became a large portion of the screen, the Aibo would stop and transition into the next state. Vision segmentation was a way for the Aibo to recognize raw objects and react accordingly. In order to do this, the object must be mapped to a color and the Aibo must be trained to recognize that color.

The vision setup involved several steps. First, pictures of the objects that was to be used in segmentation were taken using the camera on the Aibo. The format of the pictures had to be in YUV. This provides better results then RGB because the color could change if lighting conditions changed or if taken at a different angle. The camera can be accessed by going to Tekkotsu’s Controller GUI-> Raw Camera. To take a picture, freeze the image and then save it as a .PNG file. The setup for taking the pictures included wrapping construction paper around a soda bottle which will act as a marker and spreading black construction paper around the objects so the distinction between the colors are clear. Several pictures were taken of the objects at different angles as well as the background.

Once the pictures are taken, they need to be loaded into a program called TileTrain which is included in the Tekkotsu package. In the program, the images are loaded and each color for the object needs to be selected and saved as particular color segmentation. For example, select all the green that appears in the pictures and save it. Then do the same for the other colors. Finally, save the entire thing which will create a .TM and .COL file. These two files will be used in Tekkotsu for segmentation.

After the two files are created, they must be copied to the “Project_Root”/ms/config directory. Then, changes to tekkotsu.cfg need to be made. Inside the file, the default color segmentation files need to be commented out and be replaced with the names of the .TM and .COL files.

Now a segmentation ID needs to be created for the colors used. These changes are to be made in ProjectInterface.h in Project_Root/Shared folder. To use the user identified color segmentation, it needs to be added to the StartupBehavior_SetupVision.cc in the project folder. Now it’s created, it can be used in a behavior.

To stop the Aibo, the size of the object was needed. This was done by utilizing a function in the segmentation software called ‘getArea’. The function returned the area the object took up in the camera’s field of vision. Once the area was beyond a threshold value, determined experimentally, the velocity of the Aibo would be set to zero.

**Encircle Behavior**

This behavior made the Aibo encircle the ball while having its head turned 90 degrees. The kicking mechanism of the Aibo was in such a way that the head turned from right to left. When the Aibo encircles the ball, the head is turned 90 degrees to the left. It must be left because the once the Aibo saw the target, it will transition into the kick. If the head was turned to the right, the Aibo would see the target on its right side when it turned and will kick the ball to the left.

The way the Aibo was made to encircle the ball was by altering the setTargetVelocity function, with three parameters: x, y and theta. Parameter x affected how fast the Aibo would walk forward and backward where as y changed how fast the Aibo would strafe to the left and right. Changing the angle affected how much the Aibo
would curve in its walking either in the x or y direction. To encircle the ball, the y and theta values were altered. The Y value was set to a value of ~80 and the angle was set to a value of ~1.

**Kicking Behavior**

The kicking behavior utilized Tekkotsu’s MotionSequence scripting framework. The framework establishes frames for the Aibo to transition through to create the motion. Each frame sets the values for each joint on the Aibo and a transition time, which determines how fast or slow the transition to the new position. Furthermore, files can be loaded into the script via the “load” command.

The script is loaded by creating a new shared object within the Tekkotsu framework of type MotionSequenceMC. The framework parses the file to create the motion as a “prunable” motion, which is a motion that has a definite time span (in comparison to a walk). Once the motion begins, the behavior will listen for the event when the kick completes.

The actual mechanics of the kick was the using the head as a means of driving the ball forward. The Aibo would move its weight to the right-hand side, and then swing to its left, moving its head in the process to achieve maximum momentum into the ball. This would result in the Aibo kicking the ball to its left.

**Send Behavior and Wait Behavior**

The send and wait behaviors use the Aibo’s wireless capabilities to communicate between two separate Aibos. Communication was created via socket programming; each Aibo served as both a server and a client and communicates on port 9092. In the send behavior, the sending Aibo will create a connection to the other Aibo; from there the sending Aibo will write to the socket the word “go.” The waiting Aibo will listen on port 9092 for a socket connection, then will read off the socket. If the data from the socket equates to the word “go,” the behavior will finish.
Results

Finite State Machine

The finite state machine was successfully implemented and served our needs as anticipated.

Tracking Behavior

The Track Behavior was also successful as color segmentation achieved the goal we had hoped it would. The Aibo was successfully able to identify the colors it had been trained to, which made transitioning on an event dealing with that color possible.

Encircle Behavior

The Encircle behavior was a success to a degree. The finished version only had one small nuance in that it was a static behavior that did not account for where the ball was when moving. A strictly better solution would have been to make the behavior dynamic by keeping track of where the ball was using the Aibo’s Chest IR sensor. The behavior would then adjust itself to keep the ball at a certain distance, so as to make sure it is correctly positioned for the kick behavior that will follow it.

Kicking Behavior

The final version of the kick was able to kick relatively well, given the ball was sufficiently close enough to the robot. The ball would go errant if the ball was too far or too close to the robot. This is due to the ball rolling off the head, resulting in the ball going to the left if it was too far and going right if it was too close.

Send Behavior and Wait Behavior

A proof of concept was developed that showed communication between the two Aibos was possible; however, when applied to the overall behavior, the Aibo could not create a socket connection with the other Aibo (a “Connection Refused” error). In summary, communication between the two Aibos was not achieved.
Discussion

Finite State Machine

Although the concept was straightforward and Tekkotsu has built in support for FSM’s, progress was slow at times, and very fast other times. Most of this stemmed from having to learn how to use transitions meaningfully as well as some less then first-class documentation on having a state-node post an event as having occurred. It should also be mentioned that although we found it both highly useful and important, a finite state machine does little on its own with out meaningful behaviors to populate its State-nodes.

Tracking Behavior

This was a very straightforward process. However, since a large number of shared files were altered, it’s highly recommended that a backup be made of those files. By following the “SegmentedVision.doc”, the process was very easy. In order to get the best results, there are a few things that should be done when taking the pictures of the objects that is being segmented. First, construction paper was used in the demonstration and it gave good results. It should be used again in segmentation. Second, when taking the pictures, use a black background so the Aibo can easily distinguish the two colors. Third, always set the lighting conditions to what will be used later on. For example, if the Aibo will be tested in relatively dark lighting, the pictures should also be taken in a relatively dark setting. Lastly, take pictures of the objects from many different angles.

Encircle Behavior

This was a relatively easy behavior to write because it was just altering values from a given function. The hard part was the trial and error nature of testing the behavior after modifying the values of the speed of the walk as well as the radius at which to encircle at.

Kicking Behavior

Initially, the Aibo would go into an unstable state when the script was executed. After debugging, it was realized that the robot was attempting to execute numerous kicks at the same time. This was remedied by allowing the kick to execute if it wasn’t doing the kick already.

A second bug was the Aibo would be left in an unstable position at the end of the kick, which resulted in it falling over if it attempted to walk. This was addressed by having the Aibo reposition itself into a “standing” stance, allowing for a graceful transition into any other motion.

An improvement to the kicking motion would be to incorporate the infrared sensor to determine the distance from the ball. With that information, the Aibo could calculate how far to move the head out to make full contact with the ball. Furthermore, allowing the speed of the kick to change in real-time would allow for a more accurate and dynamic kick.
**Send Behavior and Wait Behavior**

Two problems were discovered when implementing the Send Behavior and the Wait Behavior. A consistent problem with the Send Behavior was it was not able to create a socket connection with the other Aibo, resulting in the Aibo perpetually attempting to connect. From debugging logs, this was due to the other Aibo refusing the connection. Unfortunately, no solution could be devised for this problem.

The second issue was the Wait Behavior and processing the string that was received from the socket. Once the string was processed, it was supposed to be compared with the string “go.” However, the mechanism for comparing two ‘char’ strings did not give the correct answer. This could be related to additional white space characters within the string received from the socket. Due to time constraints, this problem was not resolved.
Appendices

A. Glossary of Terms:

Aibo – A commercial robot, developed by Sony Entertainment, which has numerous capabilities as a platform for robotic research. Currently, the ERS-7 boasts a 64-bit processor, wireless communication, three infrared sensors, a video camera, and other sensors.

Behavior – A module that defines a particular action the Aibo will do (walking, staring at a ball, etc). A behavior has four functions, a constructor, DoStart, DoStop, and processEvent. The constructor, like all C++ classes, define the initial values for the Behavior. The DoStart defines what motions will be utilized, what events will be listened for, and what other shared resources will be used. The DoStop is the opposite of DoStart, where it releases the resources it obtained in the DoStart function, while the processEvent function carries out the actions themselves.

Event – An event is an occurrence within the Tekkotsu framework; they are broken down into various categories (sensor events, button events, vision events, etc). Each event also has a source and a value. Events are typically processed in Behaviors in their “processEvent” member functions.

State-Node – A Tekkotsu class that inherits from BehaviorBase. A recursive data structure, as it can serve either as the control flow of a Finite State Machine, or a node containing a behavior within the Finite State Machine. A State-Node usually has at least one transition (and can have more), but could have none for a termination state.

Finite State Machine (FSM) – A method of controlling the flow of actions and the resulting events of those actions within a system. Specifically in Tekkotsu: a model of over-all behavior composed of State-Nodes, Transitions and Behaviors.

Tekkotsu – A programming language, developed my Carnegie Mellon University, that allows for C++ programming of the Aibo robot.

Transition – A transition connects two state nodes together and is one-directional. It takes the destination node, and what will trigger the transition (set time, an event, etc) to fire. The transition is then added to the source state node.
B. Tekkotsu Setup for Cygwin

This is a comprehensive setup guide for installing Tekkotsu on Cygwin. Do NOT follow other setup guides such as the SUNY Albany’s or CMU’s. This guide is intended for those who want to use Cygwin instead of putting LINUX on their computers although LINUX would be easier. There is a guide for that too if you want to follow that route.

Now, the first thing you need to do is to install Cygwin. To do that, go to Cygwin.com. Near the center of the page is a “Install or update now!” link. Click on that and it will download Setup.exe to your computer. This is your main file to download AND install Cygwin. Put the file anywhere on your computer, I suggest the desktop for easy access.

Once you have that on your computer, run it. Go to next and say “Download without Installing”. I recommend this because you can see which ones are actually being installed because Cygwin does this thing where if the packages that you downloaded are not default, it won’t install it. Then click next. Now it asks you to select a “Local Package Directory”. This is where Cygwin downloads the packaged files and this is where it will look to do the actual installation. Note when they ask you for your Cygwin root directory, which should be c:\cygwin or something like that. That’s where they put the extracted files. When it asks for internet connection, say direct and click next. Now it asks you to pick a site to download Cygwin files from.
I go with the one selected on the picture above because it has a fast connection. Click next.

Here comes the most important part of the process. In order to do this right, you must download the right packages so that when you’re running Tekkotsu for example, you will be able to copy files onto the disk which didn’t already exist yet. In this manner, you won’t be doing a full install onto the memory stick which takes up a lot of time. Also, you will also have something which tells you where the error is if you download the right packages. I made a list of all the packages that you will NEED. I suggest you go through the Cygwin selector and explicitly pick out these packages to install. To pick out the packages that you want to install, click on the view button on the upper right portion of the Cygwin installer. It should say Category. Click on view until “Full” appears next to it. Now you should see all the names of packages on the right. To install a package, you’ll need to click under the “Name” column for that package until a version comes up, ie: 4.13-1. The first one should be the newest. If there are already packages selected to be downloaded, leave those alone. Compare their list to your list and make sure all the packages in your list are going to get downloaded.

**Packages:**
- ash
- base-files
- base-passwd
- bash
- binutils
- bzip2
- coreutils
- crypt
- cygipc
- cygutils
- Cygwin
- Cygwin-doc
- Db
- Diffutils
- Editrights
- Expat
- Findutils
- Gawk
- Gcc
- Gcc-mingw
- Gdb
- Gdbm
- Gettext
- Grep
- Gruff
- Gzip
- Less
- Libiconv
- Login
- Make
- Man
- Mingw-runtime
- Mktemp
- More
- Ncurses
- Patch
- Pcre
- Perl
- Popt
- Readline
- Rsync
- Sed
- Tar
- Tcltk
- Termcap
- Terminfo
- Texinfo
- W32api
- Which
- Zlib
Once you have it installed, click on Setup.exe again and this time, say install from local directory, this being the directory where you downloaded the files before. Click next and now select c:\cygwin as your root directory. Then it asks for the local directory. Find the one for you and click next. Then it will go to a screen where it asks you to select the packages to install. This is similar to the selection screen when you downloaded the files except now the only files you see are the ones that you downloaded. Go to the “Full” view and make sure everyone package is clicked under “New” to be installed. Do that and once it finish, you should have a working Cygwin program.

Now that you have Cygwin installed, the next thing to do is to install the files needed for Open-R. Go to openr.aibo.com/ sign up for an account. This is free and is necessary to get the files that you need. Once you have done that, log in to the site and go to download. It should be number 2. Then click on OPEN-R SDK. You first want to get the OPEN_R_SDK-1.1.5-r3.tar.gz file. Download the file as a .gz. If Windows changes the extension to a .tar.tar, change it to a .tar.gz. I have problems with this but resolved it by changing the file extension. Now you also need the mipsel-devtools-3.3.2-bin-r1.tar.gz. It should be half way down the page and is about 30 MB. Put these files into a directory of your choice. Now, download Tekkotsu:  http://www-2.cs.cmu.edu/~tekkotsu/TekkotsuInstall.html

Once, all three files are downloaded, copy them to the /usr/local directory in Cygwin. Now, open up Cygwin and go to the /usr/local directory. When Cygwin starts, you probably will be at the /home directory. You’ll have to do a few “cd ..” to get out and find where the /usr/local directory is. Once in the /usr/local directory and all three of the files are there, you’ll untar the OPEN-R file using the following command “tar –zxvf <filename>”. Now untar the mipsel-devtools-3.3.2-bin-r1.tar.gz file also using “tar –zxvf”. Finally, untar the Tekkotsu file. It’s very important that the extraction of the files be done in that order. Now rename the Tekkotsu_<version> number directory to just Tekkotsu. This can be done either in Cygwin or in windows.

Now that you have Tekkotsu installed, the next thing you have to do is change some settings and then do a “make”. Go into the Tekkotsu/project folder. Open a file called Environment.conf. Change the “MEMSTICK_ROOT ?=” value to “/cygdrive/e”. Now go into “project\ums\open-r\system\conf” and open up the wlandflt.txt file, also known as the wireless file. This is the file that sets up the ip address of the Aibo. Change the ESSID to “nyscat”. Also make sure that APMODE is set to 1 and that DHCP is also set to 1. Note that even if the Aibo has a static ip, use DHCP. Now, go back into your “project” folder. Type “make”. This will create a working environment for you to do. It will take about 15 minutes so go grab some snacks at the vending machine. If anything goes, review the guide and check with the first lecture from SUNY Albany which should be included.

When you’re writing the files to the memory stick, you’ll need to change the write permissions in two directories. But this can only be changed when the memory stick is in the computer.
Make sure /open-r/system/conf/* have write permissions enabled
Make sure /open-r/mw/conf/* have write permissions enabled

To do this, you’ll have to first go to /cygdrive/e by exiting out of the project folder and then keep doing “cd ..” until you can’t anymore. When you’re at the directory before /conf, do a “chmod –R 777 conf” to set that directory to have write permissions. Do the same for the other directory as well. This should only need to be done once. However, I have had problems when I try to write onto the memory stick where it says that I did not set permissions on those directories even though I did it before. Therefore, keep an eye out for this when you do a make and change it accordingly.
References
