The Sony AIBO: Using IR for Maze Navigation

Kyle Lawton and Elizabeth Shrecengost

Abstract

The goal of this project was to design a behavior that allows the Sony AIBO to navigate and explore a maze. This involved creating an algorithm that maps where the AIBO is located in the maze as well as developing ways for the AIBO to align itself with the walls of the maze. This was accomplished by taking and analyzing readings from the AIBO's IR sensor in order to update its representation of its surroundings. We found that the AIBO was able to successfully navigate a small maze and cope with most of the common anomalies it encountered.

I. Introduction

Recently, there have been many breakthroughs in artificial intelligence. Robots are able to do increasingly complex behaviors without human intervention. One of the classic examples of testing intelligence is putting a rat in a maze. Our objective was similar. The "rat" we used was an AIBO programmed to navigate and map an unknown maze.

To do this, we had to make the AIBO detect the walls of the maze and align itself to them. This allowed it to avoid drifting into the walls when moving through the maze. Another algorithm allowed the AIBO to map what it had seen, remember where it was in the maze and choose its next direction. With these two processes working together within a finite state automaton, the AIBO was able to explore, map, and complete a maze.

II. Background

A. History of the Sony AIBO

There is a clear pattern in human history of using technology to recreate the creatures dearest to us with as much realism as possible. Figurines resembling animals have been found that date back to Neolithic times. As early as the 16th century, clockmakers began using their skills to make mechanical animals and other similar novelties. Early in the 17th century Descartes proposed that animal bodies were really nothing more than complex machines.[1] By 1739 a Frenchman named Jacques de Vaucanson was able to create a mechanical duck with more than a thousand moving parts. It was able to flap its wings, eat, and digest grain.[2]

With the invention of computers in the first half of the twentieth century, a new field began to emerge: robotics. Soon robots were able to do things a human never could. Like any technology, robotics has been used for more than practical purposes. As the cost became cheaper, people began to use robots for entertainment. Soon people returned to the goal of simulating animals, this time in a more life-like fashion than ever before. Sony made one of these attempts, resulting in the AIBO.

Starting in 1993, research began on what was to become an "entertainment robot," which Sony envisioned would bring about new ways for humans and technology to interact. It was decided that the appearance that would best fit this purpose was that of one of man's favorite companions, the dog. In 1996 a rudimentary prototype was produced. It was small and had trouble balancing when it walked, but it was equipped with a camera and microphone. The first-generation AIBO, the ERS-110, was released in Japan in 1999. Though the AIBO was not originally intended for mass production, its huge success led Sony to release another model, the ERS-111, which was only a small improvement over the original. However, this paved the way for future models.[3]

B. AIBO Details

Since 1999, several more models have been released, the most recent being the ERS-7. Many more features have been added, including LEDs, improved cameras, voice recognition, wireless capabilities, and better sound. This project was developed using one ERS-7 AIBO and two ERS-210As. Because the 210A was used almost exclusively, a chart of its features is included in Table 1.

Factors such as the desired speed and the differences between surfaces can make seemingly simple motions, such as walking, a very complex procedure. We use a walk developed by the CMU RoboCup team, which is designed to be extremely stable.

The AIBO component utilized most in this project is the infrared sensor. By sending out IR light and timing how long it takes to return, it can measure its distance from objects in it surroundings. Unfortunately, the IR sensor has a limited range and is only able to measure distances between 100 and 900 millimeters. Also, measurements are taken at 32 millisecond intervals. This was crucial in mapping the walls of the maze and keeping the robot aligned between the walls.

Table 1: Features of the ERS-210A

Hardware	Details
384 MHz MIPS Processor	
32 MB RAM	
802.11b Wireless Ethernet	
Memory Stick Reader/Writer	
20 joints	18 PID joints with force sensing 2 Boolean joints
9 LEDs	
Video Camera	Field of view 57.6° wide and 47.8° high Resolutions: 208x160, 104x80, 52x40 Up to 25 frames per second
Stereo Microphones	
IR Distance Measure	Range of 100 – 900 millimeters
X, Y, and Z accelerometers	
8 Buttons	2 pressure sensitive, 6 Boolean
Sensor updates every 32 ms	4 sample per update

C. Tekkotsu

The popularity of the AIBO shows there is an interest in AI technology, one that is bound to increase as time goes on. As a result, Sony has dedicated funding to improving the AI capabilities of its product. This is what ultimately allowed our project to take place.

Tekkotsu is an application framework for robotic platforms that was developed at Carnegie Mellon University. It handles the low level and routine tasks for the user so they can concentrate on the high-level aspects of their program. We built upon the existing Tekkotsu framework to create a new behavior, maze navigation.

D. The Features of a Finite State Automaton

We used a finite state automaton to implement our maze navigation. A finite state automaton is basically a fully self-contained program that can be visualized with a graph containing multiple

nodes. Each of the nodes represents a specific behavior and each of the connections represents a transition between behaviors. A transition is activated when a certain condition is achieved by the active behavior. When working together, this is also known as a state machine.

III. Process

A. Initial Experimentation

In the beginning of the project, the team simply experimented with the Tekkotsu framework. First, sample movements were created in order to familiarize the team with making custom movements. Then a sample behavior for the dog was generated, which fully familiarized the team members with the C++ programming language that Tekkotsu is written in.

B. Alignment

1. The Importance of Alignment

The AIBO must be correctly aligned with its surroundings if it is to successfully complete the maze. One might intuitively think that the robot should never have to be realigned. After all, once it has started the maze on the correct path, the only thing that it should be doing is obeying a given set of directions. Unfortunately, this is a false assumption. When a computer program is designed for the virtual world, all of the parameters are known and set by the designer. When the programmer tells an object to move to a certain point in virtual space, the object can not really go anywhere but there. In the real world, on the other hand, the environment can not be predicted with absolute certainty. The dog's program does not work like a human's brain; the only thing it knows to do is move its limbs in a certain way for a certain period of time in order to complete its task under ideal conditions. If it does not make it to its target, it is unable to just realize this and move to the place that it was supposed to be under its basic program structure. In addition to not getting there in reality, the program would think that it had successfully made it to the place that it was supposed to be. However, if a human were told to walk to a point in space, they could just walk until they are at the desired vicinity of the target.

There are a couple of factors that can cause the robot to not go where it was supposed to go. First of all, surfaces can differ greatly from each other. If a surface is particularly slippery or soft, the robot is very liable not to go to its correct location. One more is the fact that the walk itself is not inherently perfect. When the robot first begins to walk, it must change to the starting state of the walking program so that it is able to walk. When it does this, it is very likely that the

momentum of changing into this state from its previous one could cause it to point in an undesired direction. All of these possible ways that the dog could become misaligned demonstrate the importance of continuously realigning the dog. Otherwise the dog could quickly become misaligned to the point where he begins to walk into walls. If the dog was off by only twenty degrees, this could cause it to do things that would be very detrimental to the goal of finishing the maze.

2. Centering the Dog

Even if the dog is aligned parallel to its adjacent walls, it still might not be in the center of the current corridor that it is in. Being closer to one side of the maze than another can cause a variety of problems. For one thing, the dog could walk into or scrape against a wall and cause damage to itself or the maze. Another thing is that it would be very difficult to make turns because it could knock into a wall as it is turning its body around. Centering the AIBO robot in the center of corridor is more complicated than it would seem. First the robot pans its head and takes infrared readings at two angles, once when the dog's head is panning to the left and another when the dog's head is panning to the right. The program then takes the sine of the angle that it found and multiplies by the distance that the infrared sensor detected for the distance of the closest point at that angle. This calculation will determine the distance from each wall. Then the program can use these values to compute how far and in what direction the robot should move in order to get in the center of the lane.

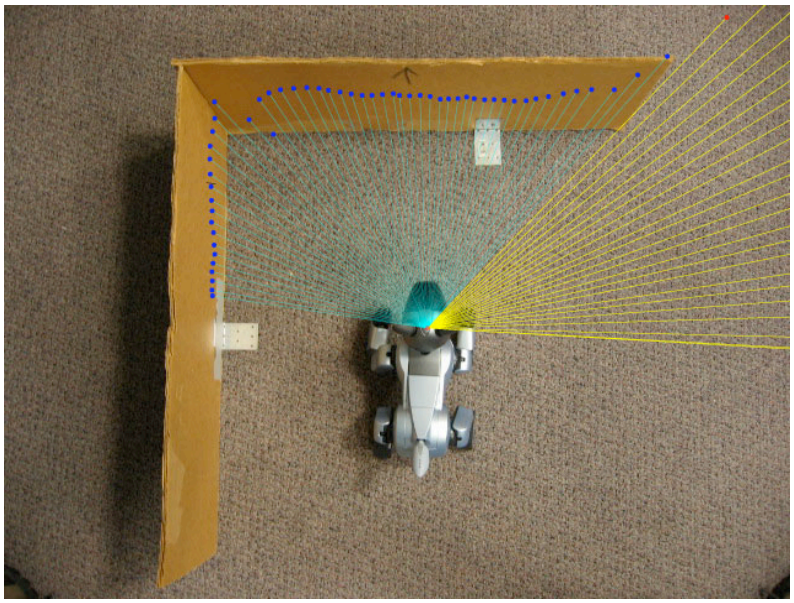

Figure 1: Example Data Spray

3. Aligning the Dog Parallel to its Adjacent Walls

One of the most complicated aspects of properly aligning the robot is making sure that it is parallel with its adjacent walls. There are many factors that one has to consider when trying to properly align the dog. Generally, the goal is to find a line that the dog can align itself to in order to be parallel to the closest wall. The process for this has several layers. The first layer is to determine what walls are close to the AIBO. If the AIBO is taking a significant number of measurements farther than approximately 700 mm, then the AIBO would say that it does not see a wall there as seen in Figure 2. The next step is to find all of the "breakpoints" from the Figure 3 graph, which describe the points in the angle vector that have a value of 0 for the derivative of the distance values in its respective vector such as in Figure 4. What this essentially means is that the program will record the angle at which the distance measurements changes from going down to going up, denoting a local minimum. The importance of these points is that it marks two critical places in the AIBO's current cell. Finding maximums will mark off both the corners, but more importantly, the angle of the minimum will give the angle the of AIBO's head that will make a line that is perpendicular line to that particular wall. If we take the shortest value of all of these breakpoints, we can accurately turn parallel to the correct wall because now the program knows at what angle it is able to form a perpendicular line with a parallel wall.

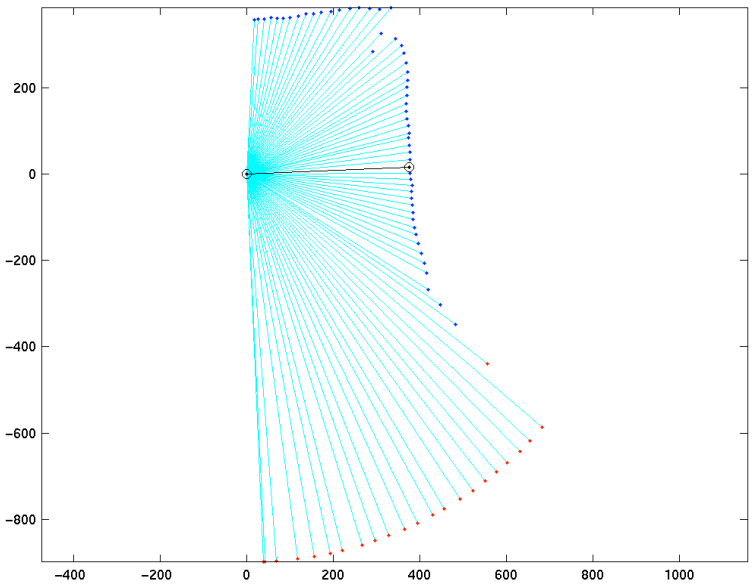

Figure 2: Example Graph of Distances and Angles

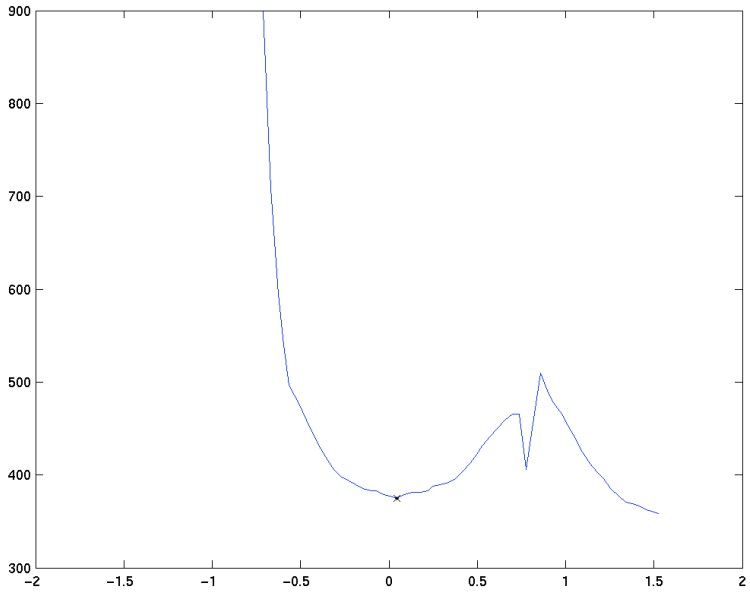

Figure 3: Graph of Angles and Their Distance

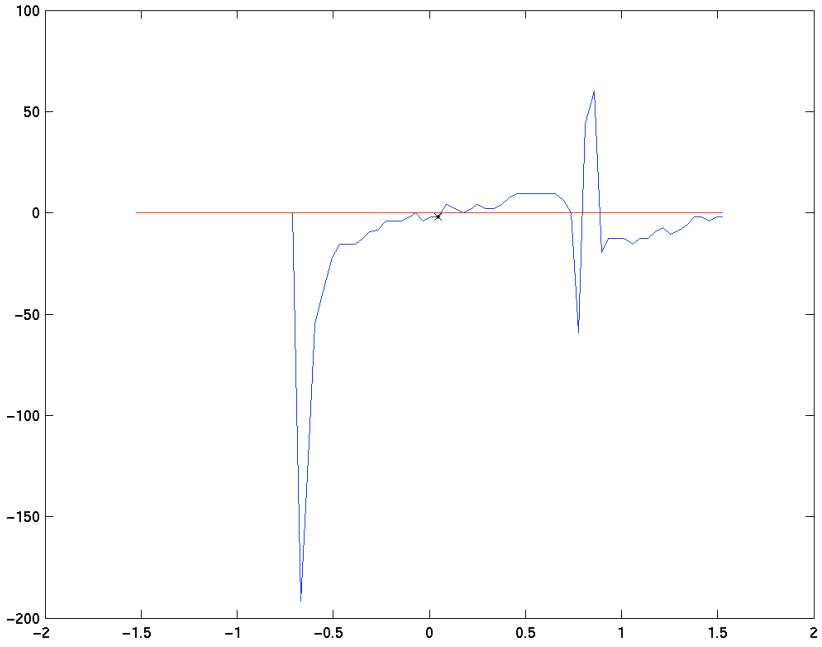

Figure 4: Derivative of Above Graph

C. Maze Generation and Navigation

Though the AIBO would not know beforehand what the maze it was navigating would look like, we needed a way to test how the navigation program was going to work. Repeatedly setting up physical mazes and sending the robot through them would have taken a large amount of time. This problem was solved by generating virtual mazes where navigation could be simulated.

The maze generator had to ensure that all mazes created would be suitable ones. For example, inaccessible areas in the middle of the maze would be undesirable, because that would leave less area for the AIBO to explore. Also, it is extremely important that the maze is solvable. In other words, the goal cannot be completely blocked off by walls. Examples of unsuitable mazes are depicted in Figure 5.

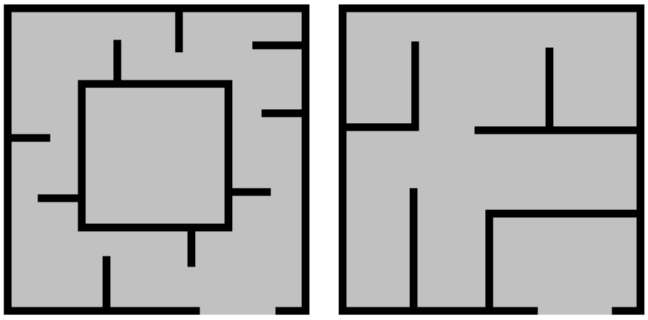

Figure 5: Examples of Mazes Not Suitable For This Project

We used the Union Find with Path Compression algorithm to generate a random maze. Once a maze is generated, it gives us a step-by-step image of what the robot has seen so far. Walls are represented by pound signs and unknown areas are represented by question marks. Figure 6 is an actual maze generated by our program, and Figure 7 shows a virtual AIBO navigating a similar maze. When comparing the partially explored maze to the master maze, it is possible to see if the AIBO is performing correctly. If it sees a wall that isn't in the master maze or a gap where there should be a wall, there has been an error.

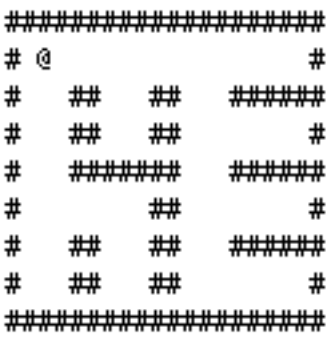

Figure 6: A Sample Maze

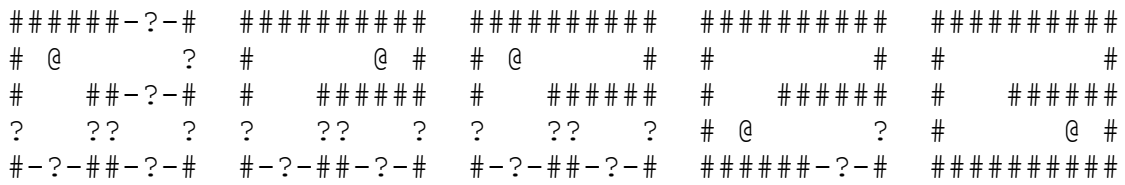

Figure 7: Navigating a Simulated Maze

What the AIBO has seen so far is stored in an array. Each cell of the array stores eight bits, with each pair of bits representing the walls surrounding a particular square of maze. The first two bits are the north wall, the second pair is the west wall, the third is the south wall, and the final pair is the east wall. 01 means there is a wall, 10 means there is no wall, and 11 means that the wall's status is unknown. If a cell stores 10111101, there is no wall to the north and a wall to the east, and the walls to the west and south haven't been mapped yet. Another array stores what cells have been explored. In order to consistently cover all the areas of a maze, the AIBO will always prefer unexplored cells to explored ones.

There is also a hierarchy of direction. For simplicity, the side to the right of the viewer east, the side closest to the viewer south, etc. If the robot cannot go east it will go south, if it cannot go south it will go west, and if it cannot go west it will go north. A stack stores the AIBO's moves, so it can go back to where there is an exit to an unexplored area if it reaches a dead end. This works much like a recursion, except for the fact that a robot can't instantly jump to the spot where another option exists. This method was chosen because it would allow the robot to explore the entire map. Another method, such as following the right wall, might have left an inner section of the map uncharted, as shown in Figure 8.

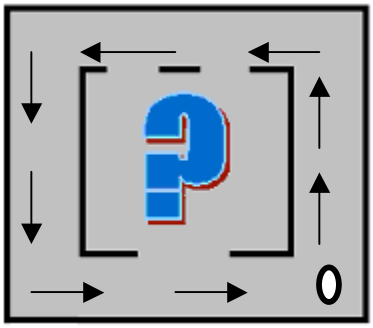

Figure 8: How Following the Right Wall Can Miss Areas

Once the AIBO senses that there are no walls within 900 millimeters, it knows that it is out of the maze and that it can stop exploring. Of course, the ability of the AIBO to map its surroundings correctly is highly dependent on correct alignment and positioning. False readings can cause the robot to go off in the wrong direction or prematurely believe that it has completed the maze.

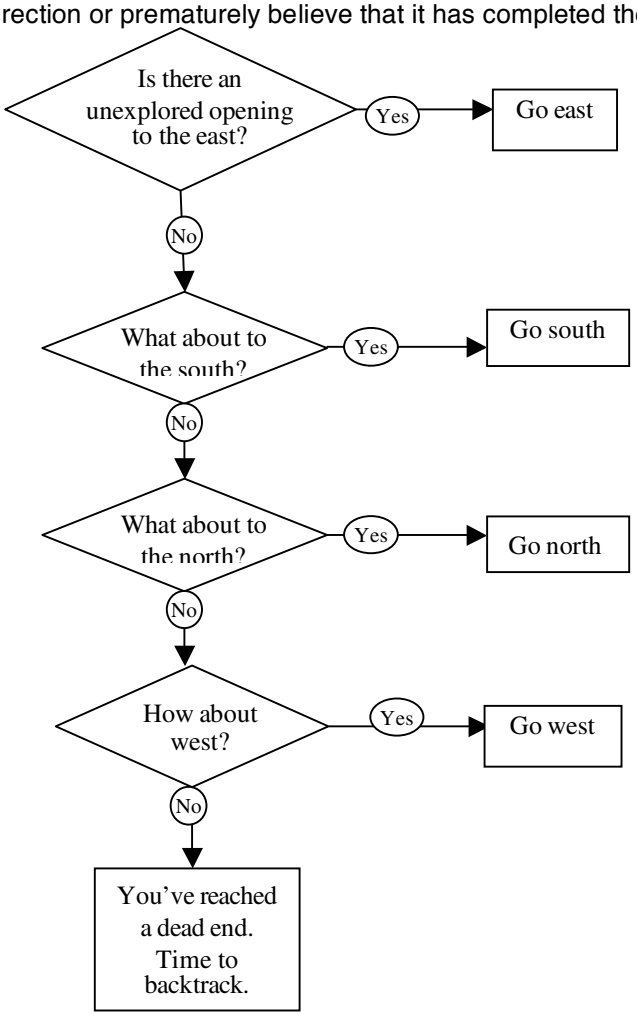

Figure 9: A Flow-chart of the AIBO's Navigation

D. Creating the Finite State Automaton

The outer shell of the automaton is the MazeRunner class. Its job is to begin the loop that will hopefully finish the maze. It starts by calling the firstAlign class whose job is to place the AIBO is the center of its current cell. Then it moves to the Alignment class. This tool aligns the robot parallel to the appropriate walls. Then the state is changed to the Decide class, which keeps track of the maze in order to tell the robot where to go. Finally once this step has completed, it will loop back to the firstAlign behavior. This cycle will continue until the robot detects no walls around it, at which point it will have completed the maze.

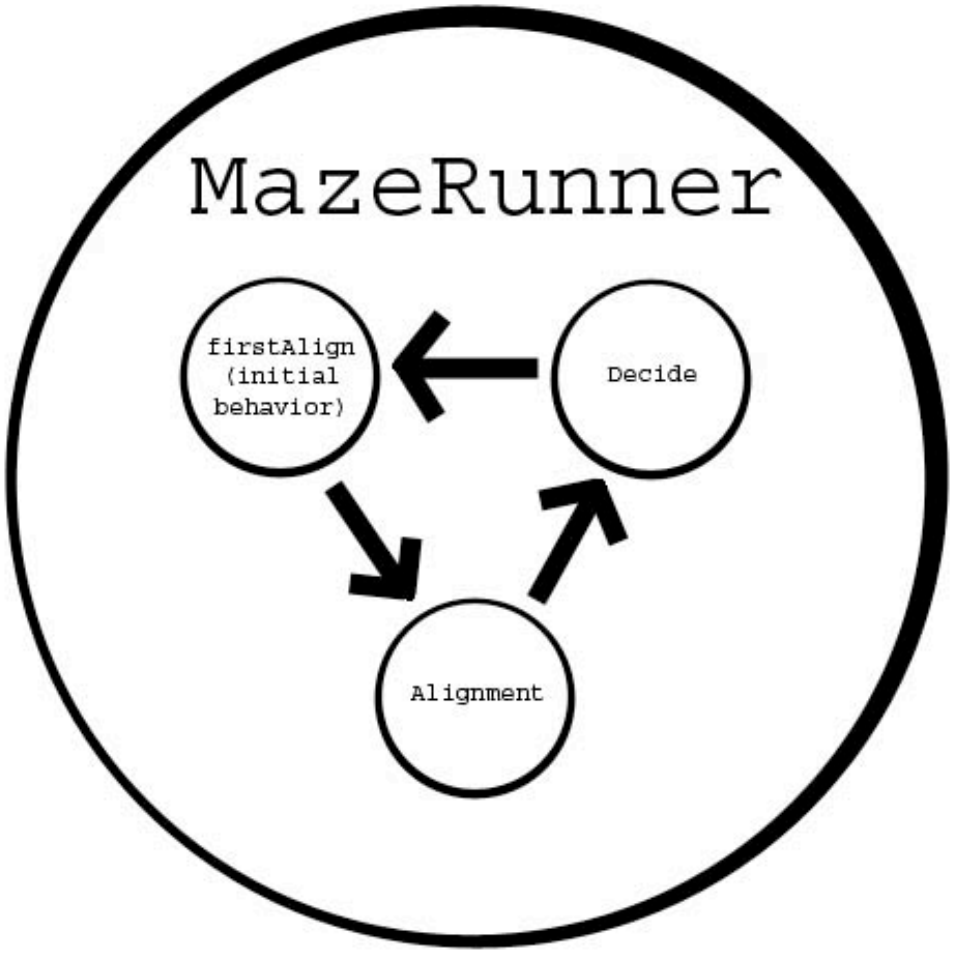

Figure 10: Representation of the State Machine

The robot transitions from one state to another by detecting when the robot has stopped walking. Once this event has occurred, the behavior will send a message that triggers the next behavior and terminates its current one.

IV. Results

The AIBO was able to properly go through a small section of maze. Unfortunately, unless calibrated very precisely, the walk is variable enough that eventually an error will occur that is too large for the error correction behaviors to handle.

V. Discussion

A. Prominent Difficulties

1. Transitioning Between States

One of the biggest problems that had to be solved was how the transition between states would be simple enough to easily operate, yet complex enough to make sure the robot is still properly navigating through the maze. The best way to do this is to listen for the robot to stop moving and then pass the current use of the waypoint walk However, when this was attempted, the program was not transitioning between classes. It would not crash; it would just stop after it had done something. This was then altered to use a text message to trigger the transition to the next node in the cycle.

2. Calibrating the Two Alignment Classes

There were a couple things that were difficult about trying to align the dog in the center of the cell. The first one was that the dog's body had to be aligned to the center of the cell and not the head despite the fact that the head is where the measurements are being taken. This problem was addressed by adding the distance between the head and the center of the body, about 100mm, to the appropriate functions. One more thing that was not working correctly was when the dog had to move into the center of a cell that was a left or right hand turn. Approximately half of the time that it would get to this situation, it would align completely incorrectly on its y-axis. Instead of aligning in the center of its current cell, it looked as though it was moving to the edge between the cell it is supposed to be moving to and its current cell. One of the best ways to help solve this and most other problems is to have the program constantly output the variables that are important

to its current function. Unfortunately, the reason for this incorrect movement could not be pinned down and completely solved, though it was likely caused by the program using the wrong values to determine how far over it should move.

Originally, the program was going to fit a line to the data that it found to the closest wall and then just align itself parallel to that. This was done by using matrices to get the slope and y-intercept of the line and then using that data to calculate the appropriate angle that it should turn. However, soon it was discovered that breakpoints were the best method for aligning parallel to walls. This switch was made for two reasons. The first reason is that breakpoints are more adaptable than fitting a line. The other reason is that breakpoints are far less complex than using matrices. However, we were still getting some anomalous behaviors from the robot. One was that the robot would often over adjust itself when it was turning. This was likely a result of not taking into account the momentum of the dog as it is turning.

Most of the misalignment problems could be solved given additional time. Pretty much all of the methods should be tweaked to account for such problems as momentum. Unfortunately, this is a long and cumbersome process that requires a lot of time for the robot to be tested and its values adjusted so the probability of failure for any given situation is low enough to reliably navigate.

B. Successes

The robot was able to successfully navigate through a small maze with a fairly high rate of success. However, as the maze becomes larger and more complicated, the chances of something going wrong greatly increase. Perhaps if the two alignment classes are improved, the effects of such errors can be reduced.

C. Future Work

There was not enough time to allow the program to be extremely adaptable. As a result, there are some things that there were not programmed because of time constraints. It would be desirable for the robot to be able to go through mazes with different types of walls, such as a maze with walls of differing thickness or a maze where the walls are curved. Another thing that could be improved is the time it takes to go through the maze. It could sense the walls around it while it was walking, so that it could realign itself as soon as it stopped. If the program could cut across corners, it would also reduce the time it would take to run through the maze. This means

that the robot would be able to recognize that it is approaching a corner and then make the appropriate movements to go right through the corner cell.

VI. Conclusion

A. Project Evaluation

There are several conclusions that can be drawn from this project. The first is the importance of accounting for errors in robotics. One must make sure that the program has an efficient way of testing where the errors are in a program in order to be able to efficiently fix them. Another would be that a good program should have an easy way to calibrate its actions for different environments. Finally, the finite state automaton turned out to be a very efficient way to allow the robot to accomplish the projects goal of moving through an unknown maze as well as the supplementary goal of generating a map for the maze.

B. Real-World Applications

While an AIBO finding its way through a maze may seem like a trivial thing, similar technology could be applied to simplify everyday tasks or even to save lives. Many places in the real world, such as a large building or a room with many pieces of furniture, could be viewed as a maze with obstacles to avoid. For example, a robotic mailman could make deliveries to each room in a building and remember where it has been. A robotic vacuum could clean a room or even a whole home in the most efficient way possible. Robotic rescue workers could go into hazardous situations, like a burning building, in place of humans and be able to find their way back out again.

Though it is highly doubtful that AIBOs will be used as search and rescue dogs any time soon, maze navigation will play a large role in the future.

VII. Acknowledgments

Team Project Advisor Ethan Tira-Thompson
Teaching Assistant Jack Shi
Dr. David S. Touretzky, for the use of the equipment
The Robotics Education Laboratory, for the use of the maze walls
Dr. Peter Lee, for facilitating a productive working environment

[1] Cook, Aaron, Khanna, Abhijit, and McDermott, Maureen; “A Brief History: The Development of AI,” *Artificial Intelligence,* 28 July, 2004 <http://bsc.edu/~spitts/cognitive/projects/index.htm>.
[2] Strandh, Sigvar; *The History of the Machine,* (Dorset Press, New York, 1979).
[3] Unknown; “AIBO: From Conception to Reality,” *AIBO-Friends.com*, (9 December, 2002), 28 July, 2004 <http://www.aibo-friends.com/contentid-36.html>.