

AIBO Motion and Vision Algorithms

Haoqian Chen, Elena Glassman, Chengjou Liao, Yantian Martin, Lisa Shank, Jon Stahlman

Abstract

The two chief goals of the AIBO Motion and Vision Algorithms project were to design an upright walk and to process the AIBO's real-time images so it can follow a line. The CMU RoboCup team uses a walk that positions AIBO on its elbows for speed, stability, and surface area for pushing a soccer ball. Our desire was to implement a more realistic walk, with improved stability to reduce the challenge of image processing. The test of following a line is an introduction to image processing, and has application to the popular CMU Mobot competition. Using both client- and server-side application frameworks, the parameters of the walk were varied and set through intuitive optimization and objective evaluation. The image processing was based around a basic algorithm chosen for its accuracy and efficiency. We found that the vision algorithm successfully directs AIBO to follow a line with various obstacles. The more realistic, upright walk designed was more stable than the RoboCup walk at low speeds, and allows a better field of view due to a higher camera position. This walk is now being tested in the AIBO Lab at CMU, and the line following algorithm can be implemented to give a person greater control over AIBO's trajectories.

I. Introduction

The goal of our motion algorithms is to create an upright walk as well as a walk designed for improved camera stability. The previous walk implemented by the CMU RoboCup team was based more on functionality than style. With the previous walk, the AIBO robot walked on its knees, for the purpose of lowering its body in order to hit the soccer ball. However, we are interested in finding a more life-like walk, one in which the dog walks on its paws. Also, the previous walk was optimized for speed at the expense of the camera's stability. By making the camera steadier, the video analysis and object tracking can be more accurate.

At Carnegie Mellon University's Robotics Institute, the Skinnerbots research project is developing a system for the Sony AIBO platform that aims to allow training the AIBO much like a biological dog. One of the project's most significant accomplishments thus far was the creation of the open-source Tekkotsu framework, which handles low-level details so programmers are free to concentrate on high-level behaviors.

The Tekkotsu developers didn't start writing code from scratch; they built on the successes of a few other teams. One prominent example of past progress inherited by Tekkotsu is the CMVision library developed by the CORAL Group's Color Machine Vision project. By providing a simple yet robust low-level color recognition system for real-time robotics, CMVision allowed the Tekkotsu researchers to save the time and effort it would take to develop their own vision subsystem. In turn, Tekkotsu was able to create behaviors for the AIBO that could take advantage of the AIBO's

vision.

Due to the specific interests of the team members, and with this kind of foundation to build upon, the subject of vision was chosen for this aspect of the project. Specifically, the team decided to design and implement a line-following algorithm. Line recognition, a field more closely researched in projects such as the Mobot race competition at CMU, was an area of low priority to the Tekkotsu developers. Therefore, we decided to create algorithms that allow the AIBO not only to recognize a line, but also to follow it autonomously. To do so, we utilized server-side development tools, object-oriented programming in Java, and a great deal of ingenuity.

II. Background

Sony released the first AIBOs in June 1999, the beginning of a technological phenomenon. For the first time, consumers could have an intelligent and trainable robotic companion. Three thousand robots were sold in 20 minutes in Japan while the U.S. took four days to exhaust its supply of two thousand. Again in November, Sony released another 10,000 robots for which Sony received over 135,000 orders. As consumer interest in Artificial Intelligence increases, products such as these are bound to appear more frequently in the market and the competition to make the most realistic robotic companion will increase. Therefore, Sony has committed major funding to research to increase the AI capabilities of the robotic companion.

Part of this funding is directed to college graduate students in the form of grants for the continuing research of improved AI. Our team project leader, Ethan Tira-Thompson, is one such recipient of this funding. He is in the Masters program at CMU for Robotics. Ethan is actively engaged in improving the learning capabilities of the AIBO companion. He is also concerned with the promotion of the AIBO through his website, www.Tekkotsu.org, which documents all of his team's work. This website showcases his current topics of research as well as documents all the commands used in programming AIBO. This was our major resource for the project, as it outlined basic AIBO programming and event hierarchy within the dog's internal hardware. Sony has programmed AIBO with its own extensive code, which includes voice and pattern recognition and behavioral responses. The goal of the CMU team is to add even more complex behaviors to this code. Their final objective is to advance AI to where robots learn exactly as humans do.

III. Motion Algorithms

A. Carnegie Mellon Legged RoboCup Team

The CMU RoboCup Team works on multi-robot cooperation and teamwork among robots. It has a soccer team of four AIBOs with which it competes at the annual RoboCup tournament. The team has placed in the top three the last five years. The CMU RoboCup team developed the walking algorithm which we used to create our walk. Basically, this algorithm allowed us to extract the walking parameters from the code and change them as we wished. Therefore, our

section of the project was not about learning how to code and implementing algorithms. It actually was concerned with the motion of legged robots and creating a new walk. Although this may sound simple at first, it proved much more difficult than first thought.

B. Approach

In any jointed robot, there is an inherent "give" in the joints. In other words, even when the joint is supposedly stationary, it still can move in tiny increments. The instability of the camera caused by the give in the joints is particularly noticeable in AIBO because its camera is mounted in the head, which is affected both by the joints from head to body and from body to legs.

In animals, compensation for the instability of walking is done subconsciously in the neck and in how the eye moves. For example, imagine a runner. His head is constantly moving up and down because his body is, yet he does not become disoriented. Also, he is constantly moving his eyes and focusing on certain objects. However, in robots, we must program reactions such as these. For our image tracking to work successfully, we need a stable image unaffected by either the give in the joints or the lack of head compensation.

Therefore, we analyzed several techniques of image stabilization before deciding upon a final approach. Our first option was a real-time software image stabilizer. We actually found research and an implementation of this technique online that was developed at CMU. However, we soon realized the programming was too complex for our levels of programming experience. Second, we contemplated the idea of using head compensation. With this technique, we would have analyzed the current walk and moved the head to compensate for the movement of the camera to achieve a smoother picture. Third, we could choose to create an entirely new walk that would walk upright on its paws and would be smoother than the original. We decided on this last idea because not only would the new walk hopefully solve our problem of camera stabilization, it would also make the dog act more realistically.

C. Implementation

As mentioned before, little coding was needed on this section of the project. However, we still needed to write some C++ code to tweak the fifty parameters controlling the motion of the legs. Ethan already had a Java-based controller built into Tekkotsu to remotely control the dog from a computer. Not wanting to implement a new interface from scratch, we created a menu to add to his interface so that we could change each variable individually and see how it affected the motion of the dog. You can see the completed interface in Figure 1.

Next, we needed a method to determine whether we had actually improved the camera stability. Therefore, we used the accelerometer installed in the robot to gauge our improvement. The accelerometer consists of as a mass/spring system located in the core of the body, and it measures force exerted on the body in all three dimensions. We wrote code to activate the accelerometer on the AIBO and output the acceleration data for each new walk.

After writing these two programs, we were ready to find our new behavior. A behavior is any application that runs on the dog. In this case, it is the walk algorithm. At first, we used trial and error, as we did not always know what each variable did. After many hours of working with the robot and many crashes, we discovered how each variable affected the walk and what parameters were best to change.

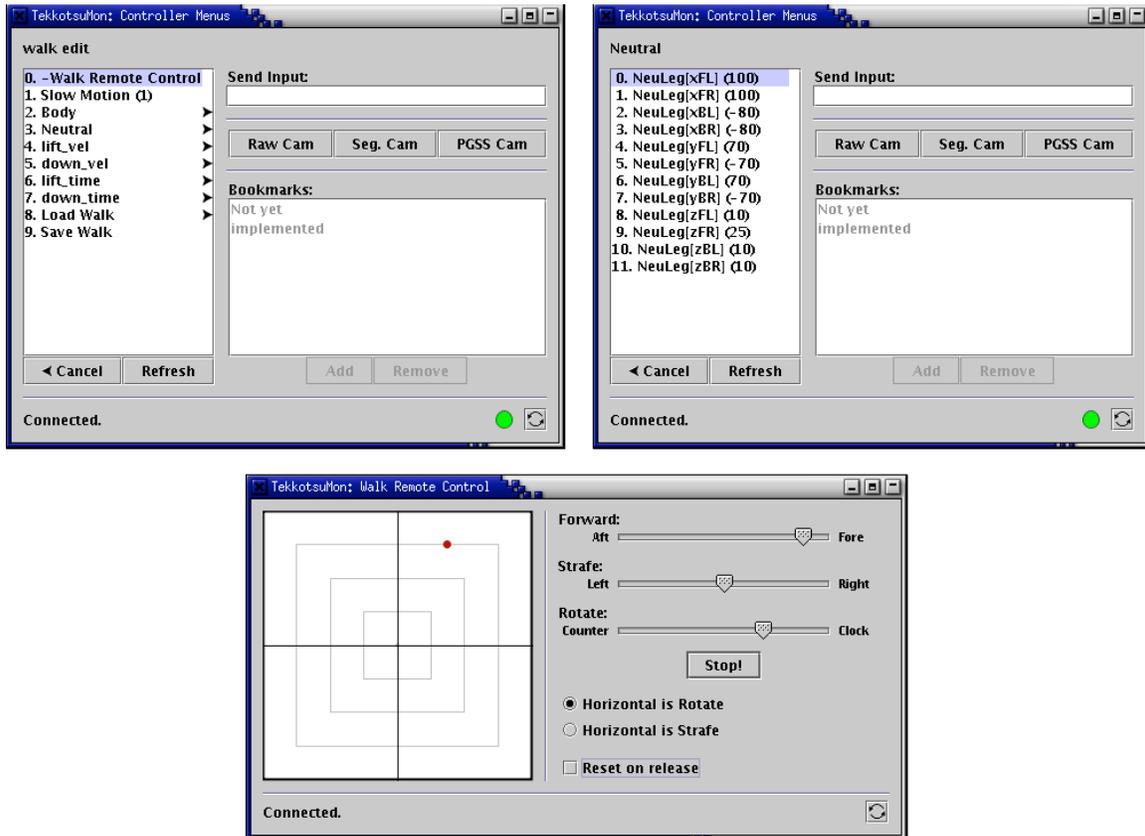


Figure 1: Graphical User Interface

Lift velocity and down velocity parameters measure the speed of each leg at the beginning and end of each step. Lift time and down time specify the amount of time it takes for the AIBO to lift a leg up and put it back down. Neutral position controls the steps' position relative to the body. The body angle parameter controls the angle of the body relative to the ground. Likewise, the body height parameter is used to assign the height above the ground. The period is the amount of time it takes for one cycle of motion. There are also two parameters called hop and sway, which we used mainly for experimental purposes. The hop parameter causes the AIBO to push down its legs during each walk motion. Sway causes the AIBO to sway towards the left and right side.

Gaits are different styles of walk for the AIBO. There are four gaits available for four-legged locomotion: trot, crawl, pace, and gallop. However, only the first three are mechanically feasible

for the AIBO.

The trot uses synchronized motion of diagonal legs, beginning with the front right and back left legs. The most unique aspect of this gait is it keeps the force of gravity balanced between the legs. Crawl directs motion of one leg at a time, starting with front right, then left back, front left, and back right leg. The pace gait causes the AIBO to move both legs on one side at once, starting with the right pair. The best illustration for the gallop gait is the motion of a race horse. Unfortunately, the AIBO's motors are not powerful enough for the gallop.

After many hours and much experimentation, we found a trot that worked well. It met the conditions of standing on its paws and holding the camera more stable than the previous walk. To show that our walk improved the stability of the camera, we created graphs using Microsoft Excel with the data collected from the accelerometer and compared the graphs of the original walk to those of our new walk.

C. Obtaining the Data

1. Constants

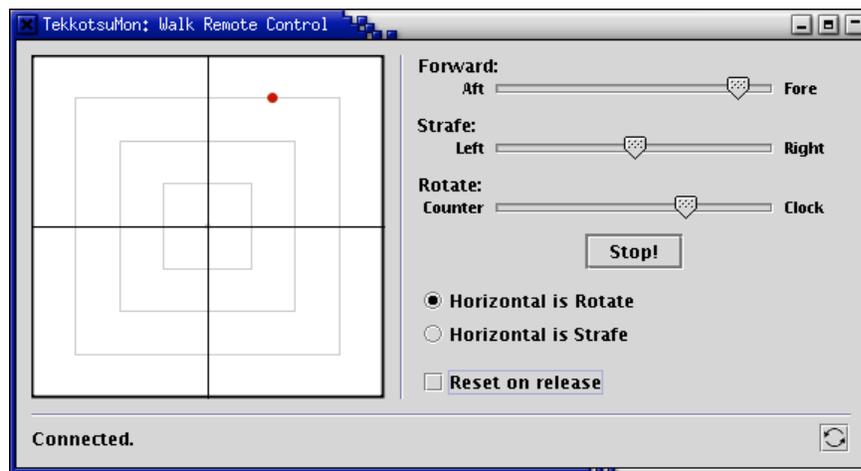


Figure 2: TekkotsuMon: Walk Remote Controller

To make sure the data from the walks are comparable, certain variables must be kept constant. For this experiment, the speed for each trial is the same. In Trial 1, PGGS 5, which is our best walk, and Default, or the RoboCup walk, were both tested at quarter, half, three-quarters, and full speed. The speed was set by placing the red target on one of the three concentric squares or the very edge for full speed, as shown in Figure 2. The other constants are the time interval of 10 seconds, which the accelerometer reads in milliseconds, and the path, which is a straight line.

2. Procedures

- Set AIBO in an area with plenty of room to move in a straight line.
- Load the preferred walk.
- Click to turn on the PGSSWalkGUI to obtain a primary target for the line following algorithm.
- Click on the Accelerometer and open a terminal to get the accelerometer reading.
- Place the controller on the desired speed and click on the line following button.
- Release the emergency stop button.
- Press the emergency stop button after 10,000 milliseconds.
- Stop, save, and graph the square sum of acceleration data.

D. Data and Conclusion

1. Motion Experiments at Quarter Speed

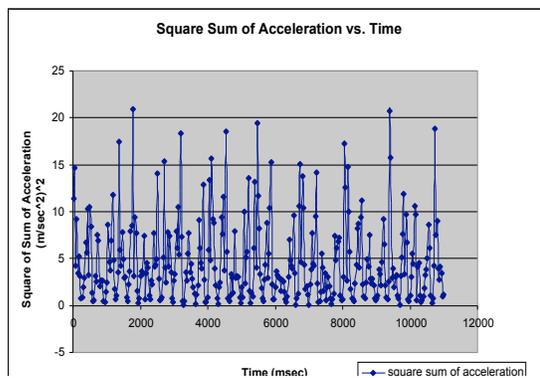


Figure 3: Reading for PGSS 5

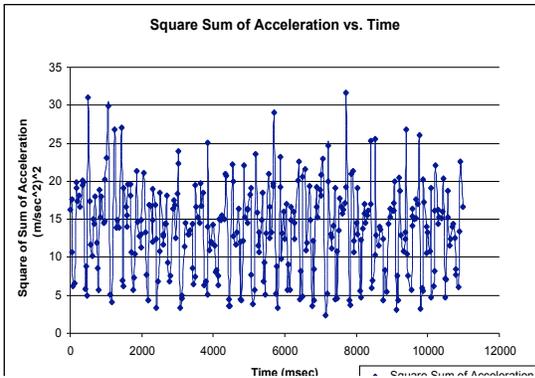


Figure 4: Reading for Default

At quarter speed, PGSS 5 is better. The shape and peak of the graphs are very chaotic and misleading; however if one were to pay attention to the y-axis on the graphs on Figure 3 and Figure 4, one notices that the range of the Default is larger than the PGSS 5, indicating a larger square sum of acceleration, thus a rockier walk.

2. Motion Experiments at Half Speed

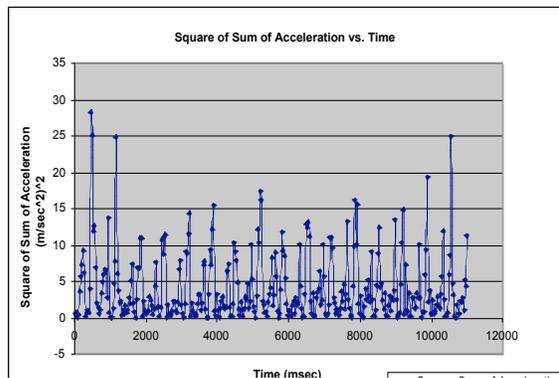


Figure 5: Reading for PGSS 5

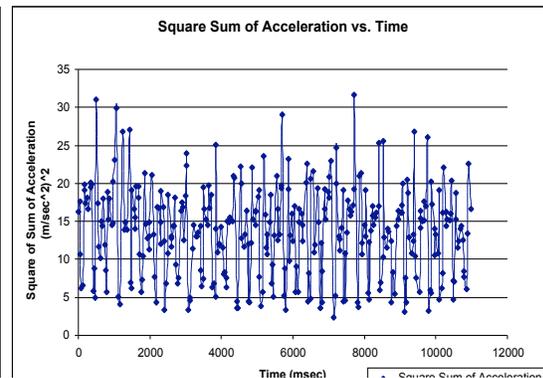


Figure 6: Reading for Default

At quarter speed, PGSS 5 is still better as shown in Figure 5 and Figure 6. Since the range of the scale of the y-axis is the same, one must examine the general shape and peaks of the graphs. Figure 5 shows less peaks than Figure 6, and its highest peak of 28 $(\text{m/sec}^2)^2$ is lower than Figure 5's peak of 32 $(\text{m/sec}^2)^2$.

3. Motion Experiments at Three Quarter Speed

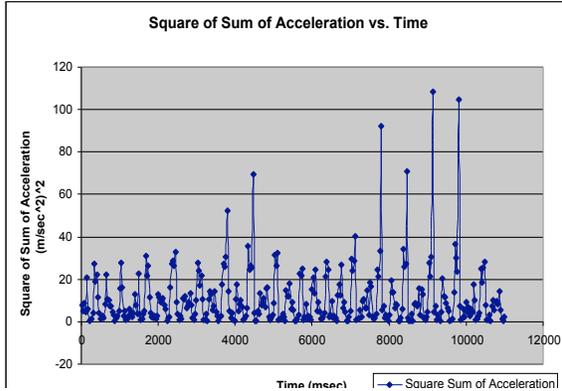


Figure 7: Reading for PGSS 5

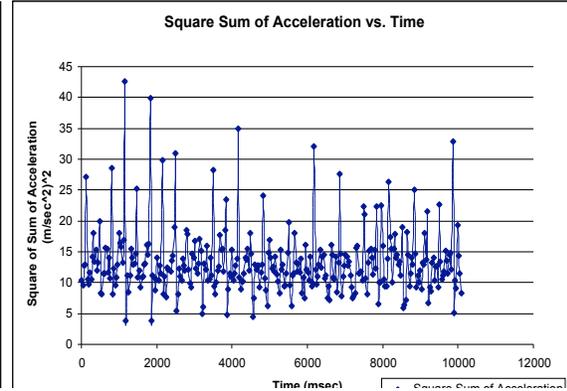


Figure 8: Reading for Default

As the speed increases, the performance of the PGSS 5 decreases. As shown in Figure 7, there are times when the peaks reach beyond 100 $(\text{m/sec}^2)^2$, while Figure 7's peaks range from 40+ $(\text{m/sec}^2)^2$ to 30+ $(\text{m/sec}^2)^2$. But the general data of Figure 7 are in the 30 $(\text{m/sec}^2)^2$ area, which can also be found in Figure 8. Overall, even though the Default Walk performs better than the PGSS 5, our walk doesn't fall that far behind in stability.

4. Motion Experiments at Full Speed

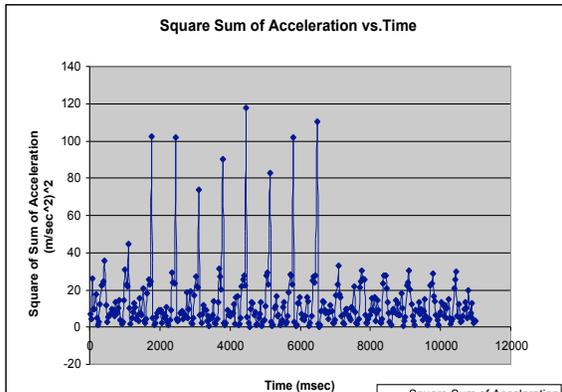


Figure 9: Reading for PGSS 5

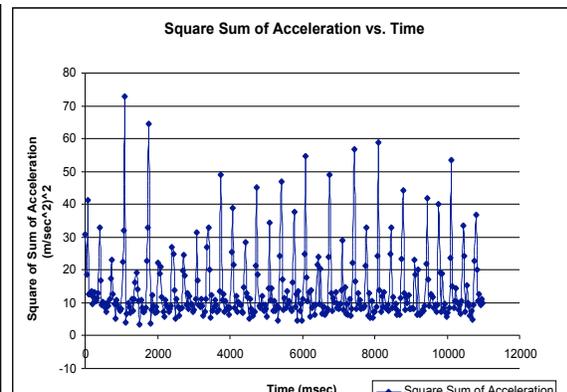


Figure 10: Reading for Default

At full speed, they behave similarly to their performance at three quarter speed. The peaks of Figure 9 reach all the way up to the 120 $(\text{m/sec}^2)^2$ range, while Figure 10 barely passes the 70 $(\text{m/sec}^2)^2$. But ignoring the peaks, the general data points of Figure 9 oscillate below 40 $(\text{m/sec}^2)^2$; however, Figure 10's data points fluctuate below 50 $(\text{m/sec}^2)^2$. But in the overall performance, Default at this speed is the better walk.

IV. Vision Algorithms

A. Background

1. Tekkotsu Raw and Segmented Vision

In order for the AIBO to follow a pink line in its image frame, it first has to view and process what it sees. The AIBO uses a CCD camera located inside its head to see each image frame. This CCD camera is capable of a raw output of 16.8 million colors in a 176x144 pixel image with a field of view 57.6° wide and 47.8° tall for up to twenty-five frames per second of real time video. The raw output from the CCD camera is easy for humans to assess and interpret, but the AIBO is unfortunately not able to process 16.8 million colors.

Therefore, the Color Machine Vision Project (CMVision for short) was started so that a simple vision system could be developed for robots. The CCD camera captures light from the environment and stores this information in the YUV color space, which specifies the hue and intensity of each color. Each pixel in the frame has its own color information, causing a large amount of memory to be taken up. This information is too large to send over the network efficiently, so the AIBO compresses it through a process known as segmentation (Figure 12). The AIBO is given a color map - a small list of basic colors - which it then uses to map every color in the image onto the index of the best match in the color map. The AIBO then uses RLE (Run-Length Encoding), which converts each color run into a series of triplets. Each triplet contains the start coordinates of the color run, the color and the width of the run. This dramatically compresses the information, allowing it to be passed over the network to a computer quickly. The triplets are then decompressed into an array on the computer, where each element in the array represents a pixel and has a specific color assigned to it in RGB format. The code needed for segmentation and RLE is already present in the AIBO; we just needed to process the data being transmitted.

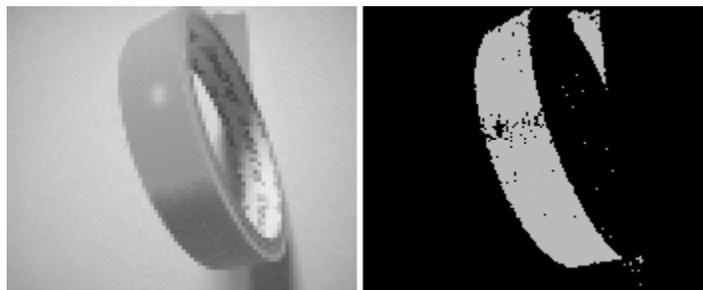


Figure 12: Raw to Segmented

2. TekkotsuMon - External AIBO Development Tools

TekkotsuMon is basically the server-side subset of application development tools of Tekkotsu. In other words, TekkotsuMon allows behaviors to be run on a computer separate from the AIBO

through the use of the wireless network. The code is in Java, unlike the client-side (AIBO-side) native C++ language. Because of this fact, multiple classes can be written to interact with each other in a user-friendly graphical user interface (GUI). For example, one of the classes in TekkotsuMon is called ControllerGUI; from ControllerGUI other classes (i.e. behaviors) like WalkGUI or HeadGUI can be called and will open up in new windows.

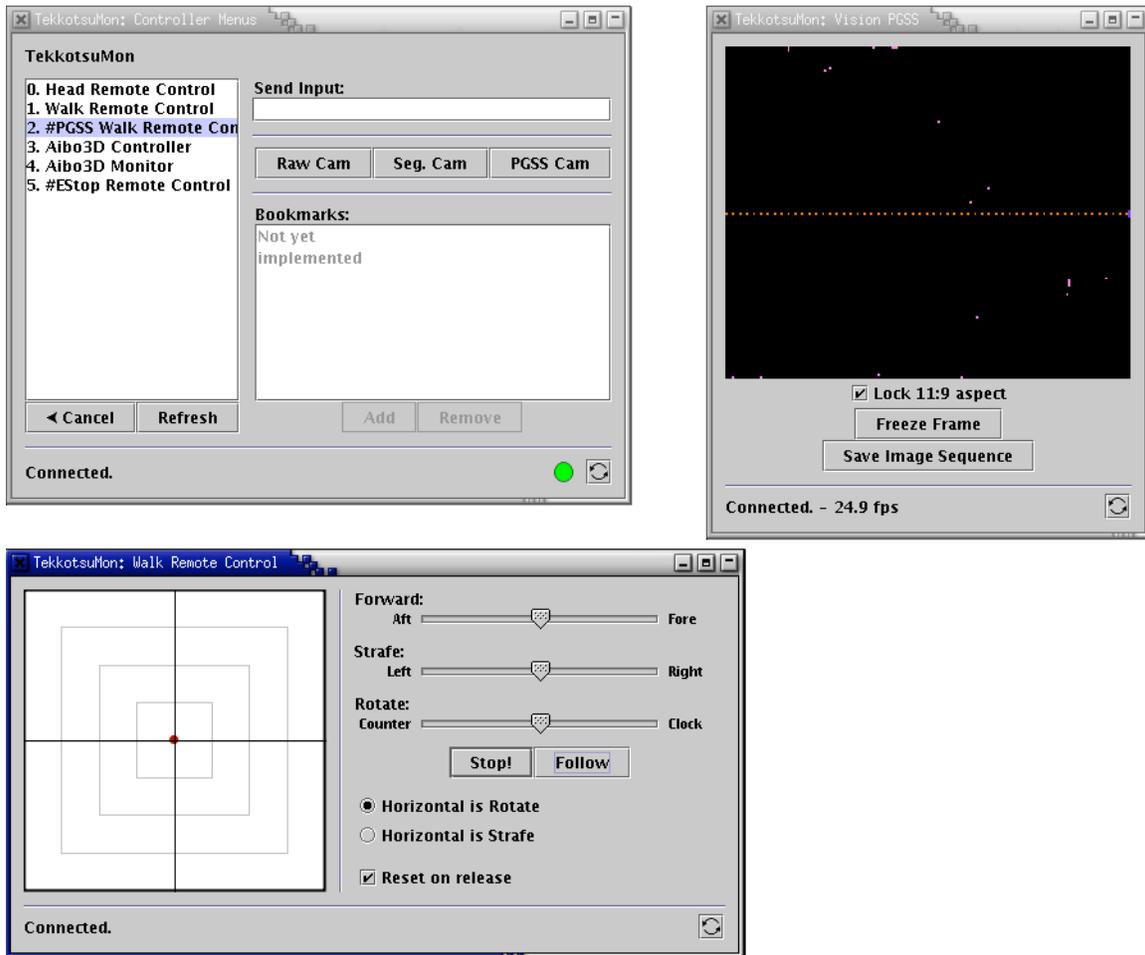


Figure 13: TekkotsuMon Vision GUI

In the implementation of our vision algorithms, we modified the TekkotsuMon code to suit our needs. The resulting classes were altered versions of ControllerGUI, WalkGUI, and VisionGUI called PGSSControllerGUI, PGSSWalkGUI, and PGSSVisionGUI, respectively, as shown in Figure 13. So in order to actually implement our algorithms for the AIBO to use through TekkotsuMon, we had to process the segmented image coming through PGSSVisionGUI. A “listener” was defined to receive the image and then process it using the algorithms. Then from here the PGSSVisionGUI communicated with the PGSSWalkGUI to control the direction and speed of the AIBO.

3. Hough Transformation

Though simple in its superficial appearance, a robot's ability to detect and follow a line while walking is a rather difficult problem and can provide for many practical applications in robot vision. In fact, the only well-known research accomplished in the field of line recognition while in motion is the Mobot racing competition at CMU and the internationally known RoboCup. Even so, the Mobot competition involves non-legged robots and the RoboCup legged competition AIBOs only deal with recognizing lines as boundaries.

For purposes such as these, the Hough Transformation is one of the best known methods of line recognition in the field of computer image processing. When an image is received through a robot's camera, the image is quite easily displayed for the human user to interpret. However, to the robot, a raw image is overwhelmingly complex, and it would barely know where to start. The application of segmented vision to create a simpler, compressed image helps to temper the problem somewhat, but still results in an image that can contain a great deal of static and other complications.

Therefore, in 1962, P.V.C. Hough invented a method called the Hough Transformation that could essentially detect arbitrary objects in an image. The most common application of the Hough Transformation, however, deals with the detection of lines. In simplest terms, the Hough Transformation inverts the parametric characterization of a sought boundary, and then finds the parameters as a function of the feature point values. One way to do this is through normal parameterization:

$$x \cos \theta + y \sin \theta = r$$

where r is the normal from the origin to the line and θ is the orientation for r with respect to the x axis; for any point (x,y) on the line where r and θ are constant (Figure 14).

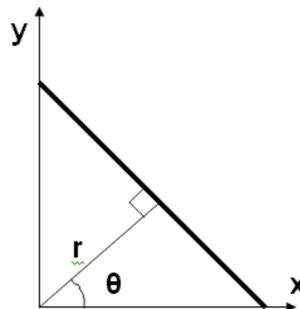


Figure 14: Normal Parameterization

Essentially, all the points in the original image are transformed into curves into a discrete parameter space or the theta-r plane (Figure 15). Once all the points are transformed, the

parameter space image is quantized using a 2D array of accumulators. An accumulator is defined as a cell in the parameter space 2D array that counts the number of transformed points at the location of the cell. Then, the cell (θ, r) which has the greatest value is chosen as the most prominent line of the original image.

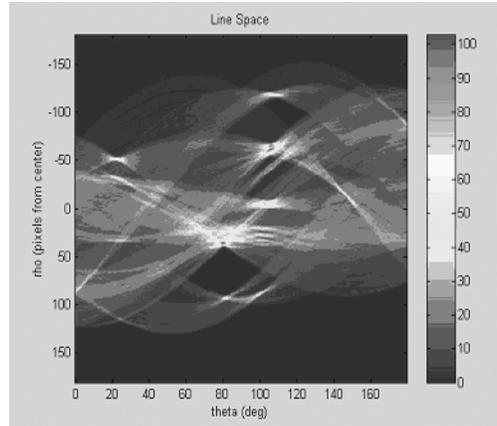


Figure 15: Parameter Space/Theta-r Plane

The Hough Transformation method of line recognition is particularly effective when trying to recognize single or multiple lines and/or edges in static images. In fact, the method can even recognize dashed lines, which are a very difficult problem yet to be solved for real-time motion images. However, in the implementation of the line recognition methods for real-time motion images, efficiency is a major factor that is perhaps even more significant than accuracy. Indeed, the Hough Transformation is highly accurate in identifying lines, but the time and power needed to do so is also high. Therefore, running the Hough Transformation on every static frame in real-time motion would take massive computational power and would easily overwhelm a system like AIBO.

Hence, for this project, a different approach was taken. Instead of using an intricate algorithm for line recognition like the Hough Transformation, a "hack" algorithm was put together. With the help of graduate student Alok Ladsariya, a simple method was devised that could recognize a line and also be efficient enough to be run on every frame of output from the camera.

4. The Basic Line-Following Algorithm

The original "hack" was the algorithm on which all improvements were made. The compressed, segmented images are streamed in from the AIBO and decoded. Each time an image is received, a two-dimensional array with a one-to-one correspondence to pixels in the image is updated. This Region Map reflects the image, but with each element as either zero, corresponding to a pink image pixel, or negative one, corresponding to all other colors. Using a version of the PaintBucket algorithm from the PGSS Computer Science Core homework, the algorithm identifies and labels each distinct, connected region. The PaintBucket recursively "colors" each connected region of

zeros with a different label.

With distinct regions differentiated, the largest region that has pixels belonging to the center row of the screen is identified. The average position of the pixels in that region in the center row is calculated, and shown on the screen in the TekkotsuMon graphical user interface as a blue target. If the blue target is to the left of the center of the screen, AIBO rotates left; the same process is followed to rotate right.

This algorithm makes two simple but fundamental assumptions. It is assumed that the line will be the largest region, and that it will cross the center row once. During trial runs, it became apparent that these assumptions caused some problems, and several improvements were made to the basic framework. For example, AIBO often encountered situations that violate the assumptions, which tended to cause the robot to get lost. Lost is defined as having no regions exist in the center row and therefore no blue target to follow. If lost, AIBO was programmed to stop forward movement while rotating to relocate the line. The direction of rotation was determined by which side of the screen the blue target was on more often during the previous fifteen frames. Finally, the speed of direction adjustment was made proportional to the blue target's distance from the center column of the screen. The result was that AIBO only made minor adjustments when it was slightly off the line and major adjustments for tight bends.

C. Implementation

After developing the basis of our line following algorithm for the AIBO we were able to begin implementation. Because we wanted to take advantage of the server-side GUI interfaces that had already been created in the TekkotsuMon external monitoring tools, we decided to modify previously created code for our needs. By using the server-side monitoring tools, we were able to see what the AIBO was actually "thinking" as it applied the line following algorithm.

Once the initial algorithm was implemented, many complications were found that could break it. Some of these issues were solved automatically by the dull simplicity of the algorithm, while others required enhancements to the algorithm.

1. Solved Algorithm Issues

Even before the implementation of the algorithm, there were some issues considered that would have theoretically broken the algorithm. Contrary to theory however, the straightforward nature of the algorithm allowed the AIBO to bypass some of these complications. These issues include dashed and branched lines. Other complications that were solved came up after the implementation of the initial algorithm when the AIBO failed to follow the line due to unpredictable circumstances. These included dealing with lines having slope opposite their location with respect to the center column and velocity compensation for sharp turns.

The first theoretical issue that was considered was the case involving dashed lines as shown in

Figure 16. The "lost" condition that was supposed to be an improvement for the initial algorithm became a theoretical problem when the algorithm was navigating a dashed line. Fundamentally, what should have happened was the algorithm would become lost during the brief break in the line, and then the AIBO would begin to rotate and then find the line in the opposite direction. Therefore, as per theory, this result does occur partly because the algorithm does become lost for a short time. Nevertheless, the time for which the algorithm was lost was too short for a reaction. As a result, the AIBO continues forward in the direction it was last going and quickly finds the next line segment in the dashed line.

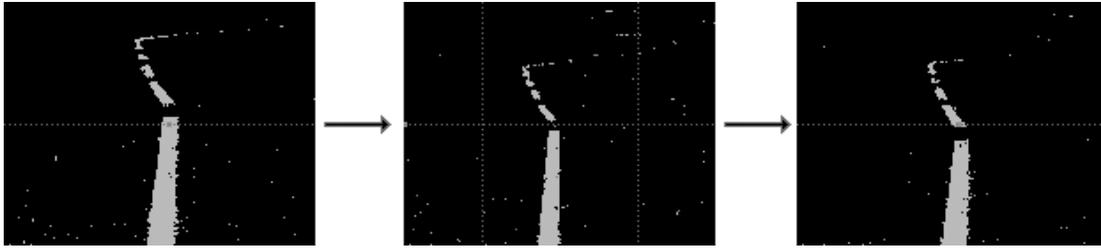


Figure 16: Dashed Line

The next issue considered was a line that branches into two paths, with one ending abruptly and the other continuing on as in Figure 17. According to the algorithm, the region containing both paths is only one region, so as both paths are detected on the center row, an average is taken for the blue dot. This results in the blue dot going between the two paths. However, this complication ended up taking care of itself. In actuality, the algorithm would indeed go between the two paths, but as soon as the shorter path ended, the blue dot jumped over to the correct path. Therefore, the algorithm inherently solved another theoretical issue.

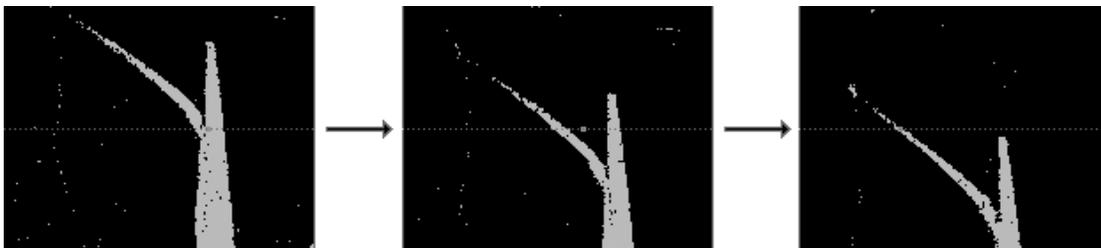


Figure 17: Branched Line

After much observation, it was noted that AIBO could lose the line when the line sloped toward the center of the image. For example, the line may be sloping to the right, but it intersects with the center row on the left side of the screen. The basic algorithm will turn left, but this leads to a decrease in the slope of the line. AIBO is not programmed to handle horizontal lines, so decreases in the slope of the line are to be avoided. Additionally, one can infer from the slope the next direction in which AIBO will have to turn in the near future, which in this special case is the opposite of what the basic algorithm will do. To address this, we wrote a method that allows AIBO to compensate for the error.

The "Slope-Opposite-Direction" method tracks what the basic algorithm identified as the line (the largest region which crosses the center row) with two more blue targets. The upper target's position is at the intersection of the line and the row one-fourth of the way down the screen, and similarly the lower target's position is at the intersection of the line and the row three-fourths of the way down the screen. The slope of the line is determined by the relative x positions of these three targets. If the slope is from one of the sides of the image into the center, AIBO compromises and goes straight.

2. Unsolved Algorithm Issues

An unsolved algorithm issue deals with bends at acute angles. The region is recognized as one entire region, but similar to the branching problem, the region crosses the center line twice. The current algorithm averages the positions of the pixels from the region in the center row, which turns out to be between the two intersections of the region and center row. AIBO therefore steers towards the middle of the acute angle, which resembles an upside-down V, and becomes lost when the last of the line (the tip of the upside-down V) passes out of the center row. At this point, whichever side of the screen the primary blue target happened to be on for most the previous fifteen frames is the direction in which AIBO begins to rotate to search.

The assumption that the line will be the largest region is most noticeable and troublesome when other large pink objects fall into AIBO's field of view. In the future, simple features can be extracted from the regions in the image, such as perimeter and a "granularity measurement." Area is already measured and applied. Secondly, these measurements must be combined to make a decision as to which region is most likely the line. Our hypothesis is that AIBO should identify the largest, most elongated and least grainy region in the image as the line.

Elongation is being considered as a measurement for shape differentiation in the future. We defined it as the ratio of a region's perimeter to its area. Elongated regions should have larger ratios of perimeter to area than less elongated regions, such as squares or circles. A similar measurement found in the field of computer vision is compactness, defined as a ratio of the perimeter squared to the area of a region. This will hopefully help in differentiating between lines and other shapes.

A measurement of granularity may be handy in differentiating between pink tape lines, which come up as very solid in the image, and less pink objects such as arms, since a much smaller percentage of the pixels pass the threshold and are labeled pink. The latter creates a grainy, Swiss-cheese like region, and to differentiate between it and a solid, fully pink object, rays could be extended from each point of the region's boundary. The rays would stop at the next pink pixel encountered. By summing the lengths of all the rays, a measurement can be obtained describing a region's graininess. The higher the number, the fewer pink pixels are by the region's boundaries and the fewer internal holes exist within the region, since rays extending from those boundaries will eventually run into the pink pixels at the opposite side of the region's hole.

Each of these measures has a different scale, so they cannot be combined as they are. From statistics, we can borrow standardization. To pick the best region, one wants to pick the region will has the best combined score from the three measurements mentioned above. Yet the scores themselves are not relevant; it is the region's score relative to all the other regions present. The standardized scores, or 'z-scores,' can be calculated and each region's standardized scores for each measurement can be summed. This sum can be compared across all regions, taking into account all three measurements equally.

Finally, we found that during our experimentation with line following algorithms, AIBO's performance, evaluated subjectively, decreased as the complexity of the methods increased. It is hypothesized that much more comprehensive methods must be employed before AIBO's line following performance increases and exceeds its current level.

D. Conclusion

The image processing algorithm was designed as a basic template, with many improvements added later to handle the special cases that were encountered. As a result, the algorithm successfully directs AIBO to follow a line with small branches, breaks, and bends at angles greater than ninety degrees. However, there are still cases in which the assumptions that were made for constructing the basic algorithm break down. Therefore, further research in shape recognition will greatly benefit the algorithm.

V. Motion and Vision Combined

A. Obtaining the Data

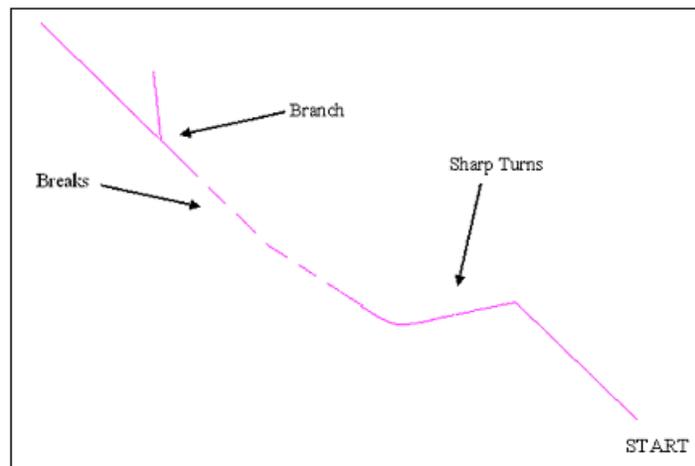


Figure 18: The Course

B. Constants

To be able to interpret the different data, some conditions must be identical. The most obvious one is the course marked by neon pink tape as drawn in Figure 18. The experiments were all run

on the course, which tested all the improvements made by the line following algorithm, such as sharp turns, breaks, and branches. The trials were also performed under full speed and constant head position.

C. Procedures

- Click to turn on the PGSSWalkGUI to obtain a primary target for the line following algorithm.
- Open the Head Remote Control and place the red target on the second concentric square vertically below the origin.
- Set AIBO so that its blue primary target is on the very beginning of the course/tip of the tape.
- Load the preferred walk.
- Click on the Accelerometer and open a terminal to get the accelerometer reading.
- Place the controller on full speed and click on the line following button.
- Release the emergency stop button.
- Observe the way different walks accomplish the task of following the pink line.
- Press the emergency stop button after AIBO finishes the course.
- Stop, save, and graph the square sum of acceleration data.

D. Data and Conclusion

1. Combination of Motion and Sight Experiments at Full Speed

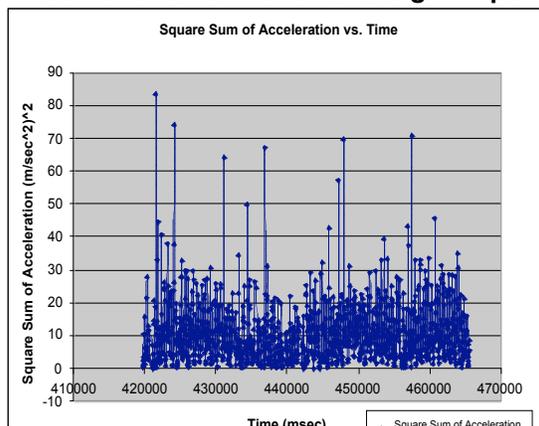


Figure 19: Trial 1 of PGSS 5

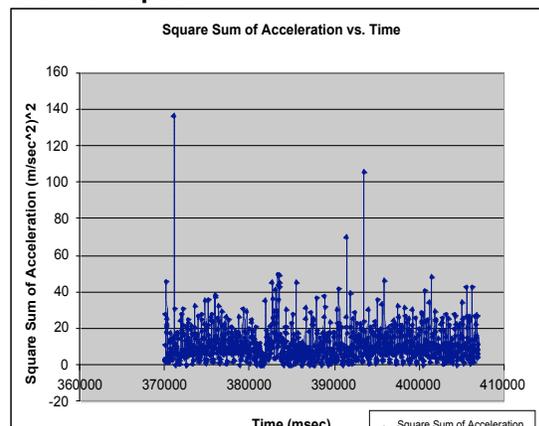


Figure 20: Trial 2 of PGSS 5

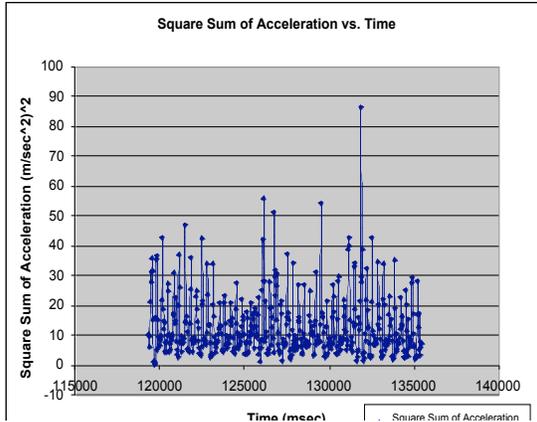


Figure 21: Trial 1 of Default

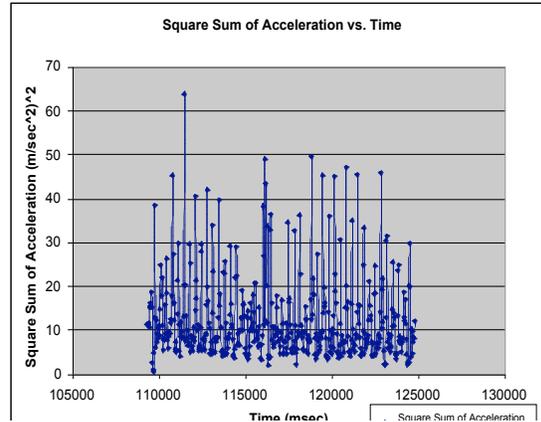


Figure 22: Trial 2 of Default

AIBO completed the task in all of the trials by not veering off the course. However, the walk was very shaky using the PGSS 5 (shown by Figure 19 and Figure 20) by peaking well above the crest in of the Default (Figure 21 and Figure 22). Also there is a clear difference in the time it takes AIBO to complete the course with the different walks. The PGSS takes about an average of 45 to 80 seconds, while the Default uses about 15 seconds.

2. Conclusion

For all the walks, line-following accuracy is inversely related to velocity. At low speed, AIBO has a chance to acknowledge the small changes in its course and adjust to them. At high speed, before it can read a change, such as a break, the blue primary target is already reading past the break. We have attempted to mitigate this by raising the camera so it can see farther ahead without suffering from a horizon-flattening effect, and also by providing a smoother walk so the vision system can obtain higher quality video. Although we have been unable to match the speed of the already heavily optimized RoboCup walk, we have succeeded in smoothing the walk for low speeds and raising the camera for better vision while moving.

VI. Acknowledgements

The members of the team would like to whole-heartedly thank the following people for making our project possible:

Ethan J. Tira-Thompson

Alok Ladsariya

Dr. David Touretzky

Greg Kesden

Sony

CMU School of Computer Science and Robotics Institute

P. Matt Jennings

Dr. Barry Luokkala

Pennsylvania Governor's School for the Sciences

VII. References

Ladsariya, Alok. Personal Interview.
Jennings, Paul Matthew. Personal Interview.
Touretzky, David. Personal Interview.
Kesden, Greg. Personal Interview.
Glassman, Martin. Personal Interview.
Dr. Carl Burch. PaintBucket method.
<http://www.dai.ed.ac.uk/CVonline/>
http://www.ri.cmu.edu/pubs/pub_3505.html
<http://ccrma-www.stanford.edu/CCRMA/Courses/252/sensors/node9.html>
<http://www.tekkotsu.org/>
<http://www-2.cs.cmu.edu/~robosoccer/legged/legged-team.html>
www.dai.ed.ac.uk/HIPR2/hough.htm
www.dai.ed.ac.uk/CVonline/LOCAL_COPIES/MARBLE/medium/contours/feature.htm
www.physik.uni-osnabrueck.de/nonlinop/Hough/LineHough.html
www.ri.cmu.edu/labs/lab_60.html
www.nynatur.dk/artikler/artikel_tekster/robo_pets.html
www003.upp.so_net.ne.jp/studio_mm/kameo/kameo.html
www.opus.co.jp/products/aibo/
www.generation5.org/aibo.shtml
www.ohta-kyoko.com/aibo/aibo.html
www.vdelnevo.co.uk/
www.d2.dion.ne.jp/~narumifu/diary.html
www.21stcentury.co.uk/robotics/aibo.asp
digitalcamera.gr.jp/html/HotNews/backno/HotNews001001-31.ht
www.seaple.icc.ne.jp/~somari/aibo.htm
www-2.cs.cmu.edu/afs/cs/project/robosoccer/www/legged/legged-team.html#publications
<http://www.icaen.uiowa.edu/~dip/LECTURE/Shape3.html#scalar>
http://www.cis.temple.edu/~latecki/CIS581-02/Project2/Shahram_Hough.doc
<http://www.cs.iastate.edu/~basjie/acad/past/imageprocessing/hough.htm>
<http://www.dai.ed.ac.uk/cgi-bin/rbf/CVONLINE/entries.pl?TAG572>

Appendix A:

For pragmatic reasons, the team's code has not been included with this document. Instead, it will be posted at the following address: http://www.tekkotsu.org/code/pgss_project_2003.tar.gz.