August 2009

Project Pele: Humanoid Robotic Programming A Study in Artificial Intelligence

Erik Emmanuel Santiago Fajardo
Worcester Polytechnic Institute

Joseph Michael Schlesinger
Worcester Polytechnic Institute

Neil Nanisetty
Worcester Polytechnic Institute

Neil Satya Nanisetty
Worcester Polytechnic Institute

Wade Omare Mitchell-Evans
Worcester Polytechnic Institute

Follow this and additional works at: https://digitalcommons.wpi.edu/mqp-all

Project Pele:

Humanoid Robotic Programming

-A Study in Artificial Intelligence

A Major Qualifying Project Report:
Submitted to the faculty of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science By:

Erik Fajardo(RBE/ECE)

Wade Mitchell-Evans(ME)

Neil Nanisetty(MA)

Joe Schlesinger(ECE)

In Partnership with
Huazhong University of Science and Technology
Partners: Cai Youfei, Yu Dongdong, Ruan Jiabiao, Liu Jianwei, Zhou Junqiang, Wang Peng

Date: August 25, 2009

Approved:

Professor Yiming (Kevin) Rong, Major Advisor, ME
Professor Jon Abraham, Co Advisor, MA
Professor Liang Gao, Co Advisor, HUST
Professor Wenjing Lou, Co Advisor, ECE

Acknowdglements:

We would like to say thanks for all the support we received from HUST University over the entirety of the project. The project could not have been accomplished without the resourses provided and the help issued. We would also like to thank our project parthers for the time they spent with us in order to successfully complete our project goal. It was hard work but we stuck together through the cultural barrier and learned a great deal. Finally, we would like to thank our advisor and co-advisors for there steady leadership and help during and after the project.

Abstract:

In the ever changing world of technology, the humanoid robot has been a constant member of science fiction culture. However, there has been an ongoing project on the campus of Huazhong University of Science and Technology (HUST) in Wuhan, China to build and program humanoid robots capable of playing soccer in the Federation of International Robot-soccer Association's (FIRA's) Human Robot World Cup Soccer Tournament (HuroCup). Our project goal was to develop a humanoid robot capable of independently displaying effective soccer skills. We divided the tasks into two teams; one designed a ball kicking robot program while the other designed a path tracking robot program. After each group completed their four major objectives, we had created a superior program than its predecessors. Using our optimized code as a foundation, another group can further develop these robot programs to demonstrate even more humanlike soccer skills.

Table of Contents

Table of Figures

Table of Equations

List of Tables

Chapter 1: Introduction

From the dawn of creation the world has been on a constant course of evolution. Darwinism has left the weaker species by the way side promoting a world of superior specimens. The same is partially true for the human race based on the theory we have evolved from apes to Neanderthals to Homo sapiens. However, from this point we have relatively stopped evolving on a biological level, instead vesting our survival on the evolution of technology. From the wheel to electricity to computers, technology has been constantly changing in order to make our lives more convenient and comfortable. There are some tasks however that humans are still forced to do such as working in dangerous factory environments or working in spaces where humans are subjected to extreme measures of risk. Tasks such as these are the main area where the use of a humanoid robot would be most beneficial.

In the ever changing world of technology, one desire has remained constant from the invention of the first robot; the desire to create a robot that resembles a man. The humanoid robot has been a constant member of science fiction culture and robotics dream but with the march of technology it has slowly been shifting from a novel concept to a reality. Over the past few years there has been an ongoing project on the campus of Huazhong University of Science and Technology (HUST) in Wuhan, China.

The test of the school's design is to place the robot in an annual soccer robotics completion that pits designs of the past year to see which design features are superior. The end goal of this project was to create a robot capable of playing soccer in the Federation of International Robot-soccer Association's (FIRA's) Human Robot World Cup Soccer Tournament (HuroCup). This is a competition very similar to RoboCup which was conceived by Manuel Veloso, Hiroke Kitano and Peter Stone, "to speed the evolution of robotic science". [1] Here the latest robotics advancements of the year are put on display to the public.

In China the competition is known as the FIRA HuroCup and breaks the robot programming challenge of playing soccer into separate categories. *Our project goal was to*

---

[1] Page 69 Almost Human

develop a humanoid robot capable of independently displaying effective soccer skills. This work was done in partnership with colleagues from the Wuhan Institute of Science and Technology. This robot was designed to complete tasks in a more efficient or faster manner than the robot which had previously been entered into the FIRA HuroCup. We broke this project into two teams based on 2 FIRA categories, one designed a ball kicking robot program while the other designed a path tracking robot program.

The first event whose performance we were required to improve was the ball kicking or penalty shooting event of the FIRA HuroCup. In this challenge, the robot had to locate and kick a ball that was positioned randomly within a predefined space on the field. Whereas in the actual challenge an opposing robot goalie would be positioned in front of the goal, our requirements were limited to successfully scoring a goal without an active goal keeper. HUST's Robot Soccer Club was able to perform four out of five successful shots, and our technical sponsor required us to improve on this shooting aspect.

The competition rules were such that the ball kicking robot needed to be able to identify a ball placed in a semicircular area in front of the goal. After locating the ball and the goal it needed to score five times out of five chances with one kick. The robot had 90 seconds to kick the ball into the goal during each attempt. When we completed our four project objectives, our robot was able to score every time. We also set up a barrier in front of the goal and the robot was able to avoid this obstacle to score.

To complete our goal, we finished four objectives. First, we created an image processing algorithm strategy to allow the robot to locate the object in the field and discern it from the background. Second, we developed an optimal ball kicking strategy to implement. This strategy was tested and researched to be the most effective in kicking mechanics. Next, we converted the strategy into DSP code. This had to be done so the robot could understand the programming and know how to react. Lastly, we tested the movement of the servos and the entire system while debugging any problems.

Another event that the HUST Robotic Soccer Club participates in FIRA HuroCup competition is the Marathon. Similar to an actual human marathon run, the HuroCup marathon challenge requires the robot to follow a line for 42.195 m (1/1000 of a human marathon

distance) in the shortest time possible. For the competition requirements, the robot needed to autonomously locate and track a 42 meter path in 15 minutes while maintaining a line deviation less than 40cm. This tests the endurance of the humanoid robots, as well as the robustness of the control algorithms.

For our path tracking robot, it needed to round the track for a total distance of 42 meters in 10 minutes. The robot was required to stay within 40 cm from the closest point of the center line, and any more deviation from this path constituted as a departure from the field. Human handlers were not allowed to interfere with the function of the robot once it began down the path.

To effectively complete our goal, we created four objectives. First, we created an image processing algorithm strategy to allow the robot to locate the white line and discern it from the background. Second, we developed an optimal path tracking strategy to implement. This strategy was tested and researched to be the most effective in path designation and determination. Next, we converted the strategy into DSP code. This had to be done so the robot could understand the programming and know how to react. Lastly, we tested the movement of the servos and the entire system while debugging any problems.

After both groups completed their four objectives, we had two working robot programs. Each group met all the requirements and even went beyond what was needed. We recommend the use of our strategies for future humanoid soccer robots. Using our code as a foundation, another group can further develop these robots to demonstrate more humanlike soccer skills.

Chapter 2: Background

2.1 Introduction

Our goal as stated in chapter 1 is to develop a humanoid robot capable of displaying effective soccer skills and techniques. To help us accomplish our goal, we explored general humanoid robotic history. Then we researched the four main areas of humanoid robotic development including: Image Processing, Motion Control, Artificial Intelligence, and Control Systems. Finally, the project plan explains the steps we took to complete our goal.

2.2 Humanoid Robotic History

'The word "Robot" was coined by the Czech playwright Karel Capek in his play R.U.R., produced in 1923.[2] In this play the robots are used to spell humans from the drudgery of everyday work; a scenario that could easily be a future reality. This however is a distant reality as the robots of today are nowhere as advanced as the general population thinks.[3] Most robots currently employed are confined to packaging facilities and the assembly lines of factories. The efficiency and accuracy of these robots however serve as a catalyst for the spread of robot use to other fields such as space exploration, medical and personal use. In the performance of these tasks however there is an underlying constant, the fact that these are all human tasks. As such the universal solution is a robot in the form of a human or simply a humanoid robot.

The definition of humanoid as stated by Webster's Dictionary is "having human form or characteristics". While it is true that robots such as quadriplegics or other purpose specific variations might be superior in performance, having a robot in human form is initially required based on UYGafdi's Theory. This can be summarized from practical experience as from a psychological point of view the casual observer is less intimidated and consequently more affectionate to a similarly structured form.[4] This is simple human nature as we fear the

---

[2] Almost Human Page 70
[3] Almost Human Page 71
[4] Robocup 2000 Page 271

unknown and the field of robotics is an area whose future has been poisoned by science fiction movies where robots are usually portrayed in a villainous light.

As such there is an ongoing drive to create an autonomous humanoid to coexist with mankind.

This integrated design work is led by corporate teams such as Honda, Toyota, and Sony, government/corporate teams such as National Institute of Advanced Industrial Science and Technology in Tsukuba (AIST), Korea Advanced Institute on Science and Technology (KAIST), National Aeronautics and Space Administration (NASA), and the German space agency Deutschen Zentrum fur Luft- und Raumfahrt (DLR), and university led teams with long traditions in mechatronics such as Waseda, Massachusetts Institute of Technology (MIT) and Technical University Munich (TUM)[5]

Such research and development is an expensive and time consuming struggle. As a result most teams of research are mainly focused on a subsystem of the humanoid robot such as vision systems, sensory reception, or bipedal locomotion.

2.3.1 Image Processing:

The first major step for humanoid robotic design is Image Processing. We initially researched general biology vision to understand the process. Next we looked into machine vision and compared its three components to biology vision with: cameras, programming software and electronic processing board technology. For camera technology, we looked into the two different types of Camera sensors CCD and CMOS. Then we researched a few different programming languages. Finally we compared different electronic processing boards.

After researching the three main components for machine vision we used programming software to optimize the mathematical image processing algorithms. These techniques can be split into five major categories: thresholding, image filtration, edge detection, digital morphology, and blob detection. Using this preliminary research we were able to design the best possible image processing algorithm and chose the most effective camera, programming language and processing board technologies to use.

---

[5] Robotics State of the Art and Future Challenges Page 72

2.3.2.1 Biology Vision

To understand how robots process images, we first need to understand how biological species process image data. The study of Biology to aid in the design of new technology is called Biomimetics.[6] Biomimetics focuses on making robots adapt more human functions like vision. For us to program a humanoid robot to see and react to images, we need to understand the important steps for image processing. To process images, humans use three main tools: the eye, the optic nerve and the brain. This process is shown In figure 1.
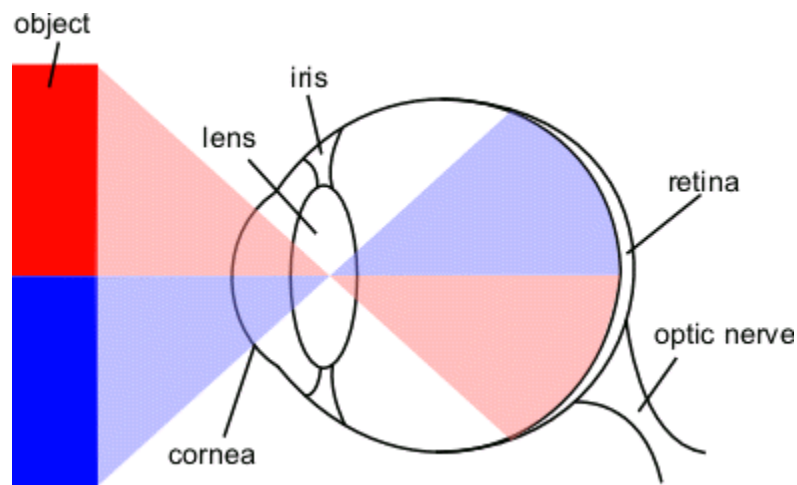
Figure 1: Retinal Inversion

2.3.2.2 Eye

The tool used to collect image data in a biological system is called the eye. The eye works in a couple steps. First the eye must locate the object. Then the iris filters all the light entering so the eye only gathers a set amount. This acts just like a brightness adjuster would in a mechanical system. Next the lens focuses the light that passes through the iris. After the light is filtered in the lens the neurons in the eye convert the light to chemical energy. Cones then convert the color within the light while rods convert the brightness. All this new information is then sent to the second human image processing tool called the optic nerve.

2.3.2.3 Optic Nerve

The second tool humans use to process images is the optic nerve which reorganizes the image data into useful information. The optic nerve is connected to the eye by the optic disk. It organizes all the light gathered from the eye in this area. This task is called stereo image

---

[6] Computer Vision Tutorial Page 1

processing. [7] Stereo Processing allows humans to judge the distance from the object in question along with its 3-D shape. This is only possible if both eyes function completely. All of this data is then transferred to the brain or the final human image processing tool.

2.3.2.4 Brain

The final and most important step of image processing is done by the brain. After receiving the information from the optic nerve, the brain assembles the data into an image you are able to recognize. Each part of the brain deals with different parts of the processing step. The brain will split the processing steps into: color, motion, shape, and size. Putting all this information together it will be able to classify objects you have seen before and register new objects as well.

2.3.3.1 Machine Vision

Machine vision is a key application of computer vision which is the science and technology of allowing machines and robots to see. [8] Machine vision systems must be able to: identify objects, detect position, and recognize shape. Typical machine vision systems will mirror how biological creatures process images.

Most systems will use a camera to collect image information like an eye, organize the information using different programming languages like the optic nerve and translate the information into something the robot can understand using an electronic processing board like the brain. For a robot to process images it needs: a camera, a programming language, and an electronic processing board.

2.3.3.2.1 Cameras

Cameras are the most important and most effective tool used to gather image data for mechanical systems such as robots. The camera will act exactly like the human eye by gathering light within the area of vision through a lens. Cameras that are used in robotics are categorized as either digital or analog. A robot will use a digital camera to collect image data. However, the

---

[7] Computer Vision Tutorial Page 3
[8] Machine Vision Algorithms and Applications Page 1

most important feature of a camera is the digital sensor.[9] There are two types of digital sensors: charged-coupled device (CCD) and complementary metal-oxide-semiconductor (CMOS).

2.3.3.2.2 Charged-Coupled Device (CCD) Sensor

The CCD sensor is one type of camera used to collect image pixels. First the CCD camera collects the pixel data and sends it to the chip. Since this data is analog, it uses an analog-to-digital converter (ADC) to turn the pixel value into a digital value.[10] This type of sensor creates a high quality and low noise image. Unfortunately, light sensitivity is much higher and the power consumed is much greater than CMOS.

2.3.3.2.3 Complementary Metal-Oxide-Semiconductor (CMOS) Sensor

The second sensor is the CMOS as shown in figure 2. The CMOS device uses several transistors to amplify the charge of each pixel in the image.[11] This data is digital already, but the sensor is more sensitive to background disruption. CMOS takes about 100 times less power than its counterpart CCD. However, the CMOS pixels might not be as well defined as the CCD. Overall, this sensor is very helpful in collecting image data.
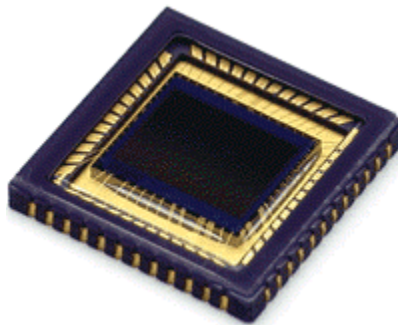
Figure 2: CMOS Sensor

2.3.3.3.1 Programming Languages

Programming languages are great at organizing the data collected by the camera. To compile image processing algorithms, the use of high level programming languages are essential.[12] Visual C++, Visual C, Visual FORTRAN 90, Matlab, Visual Basic and Visual Java are

---

[9] Machine Vision Algorithms and Applications Page 46
[10] Computer Vision Tutorial Page 9
[11] Computer Vision Tutorial Page 9
[12] Image Processing Techniques for Machine Vision Page 1

just a couple of programming languages used in organizing information. Using such developmental tools we can analyze and implement the different image processing algorithm techniques.

2.3.3.4.1 Electronic Processing Board

For the robot to understand the mathematical algorithms, electronic processing boards are needed to bridge the gap. The electronic processing board collects the data from the image processing algorithms and converts it into a language the robot understands. Acting like the brain in a biological system, the board translates the data and has the robot act based on the information. There are many different types of processing board technologies like Digital Signal Processer (DSP), microprocessors, and embedded systems. These different Electronic Processing board technologies will be discussed in more detail in section 2.6.

2.3.4.1 Image Processing Algorithms

Image processing algorithms are the basis behind machine vision and image computer analysis. Using a programming language discussed in section 2.4.3.3, we can use different algorithms to help the robot distinguish the objects it detects in the field. Sometimes it is applicable to use all possible algorithms simultaneously to allow optimal image recognition. This is not always the case because the more algorithms used takes longer to compute within the program complier.

When using such techniques, the programmer needs to use algorithms that are most effective together to complete the task. Although using all techniques together is not practical, it depends what images the robot will see and what tasks it needs to accomplish. Image processing algorithms can be broken into five main categories: thresholding, filtration, edge detection, digital morphology, and blob detection.

2.3.4.2.1 Thresholding

The first image processing algorithm technique is called thresholding. Thresholding is the technique of converting a grey-level image into a bi-level image.[13] A grey-level image is the brightness values of each pixel in an image. Bi-level images only contain black and white pixels.

---

[13] Image Processing Techniques for Machine Vision Page 1

Thresholding is a good way of recognizing pixels with similar brightness values and organizing them accordingly.

First the pixels collected are organized into grey level brightness bins. These bins are then used to create a histogram. This technique is called histogram equalization which is used in every thresholding method. Using this histogram, we next set a threshold value. The value of the threshold can be found using four types of thresholding techniques. Applying this number to the histogram, it defines the pixels collected from the image as a binary number either zero or one. The darker the pixel is the lower a number it receives. Pixels higher than the set threshold number will be assigned a value of one and lower pixel values will be assigned a zero. This technique separates the pixel colors between white and black.

The robot can define the different objects just by pixel brightness. The background and objects in the image are easier to identify with this method. Although some are more effective than others, there are many different ways to go about using a threshold. Using more than one threshold per image will yield better results since most pictures have more than two shades of pixels. There are four main types of thresholding techniques: P-tile, Edge Pixel, Iterative or Optimal, and Adaptive methods.

2.3.4.2.2 Histogram Equalization

Histogram Equalization is the main tool used in thresholding techniques to enhance the contrast of images. It organizes the pixel brightness or the grey level brightness from 0 to 255. [14] Each pixel is sorted into bins which are inputted into a histogram graph. By applying a threshold value to the graph, the pixels can be classified as either white or black. This procedure can also be used to enhance the contrast of red, green and blue color panes. Every thresholding method uses histogram equalization effectively as shown in figure 3.
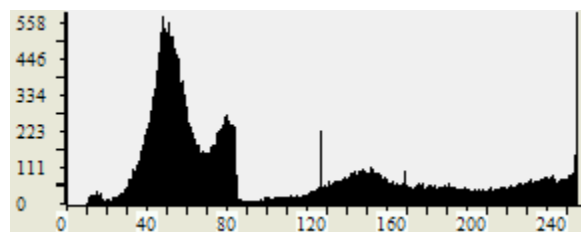
Figure 3: Histogram Equalization

---

[14] Histogram Equalization Page 1

2.3.4.2.3 P-tile

The first thresholding technique is called p-tile or percentile. It takes the cumulative sum of pixel intensities in the image. This percentile number becomes the set threshold.[15] This method collects the input data into bins of black pixels which are organized in histograms as mentioned in section 2.3.4.2.2. Since each bin contains a percentage of black pixels, the program finds which bins have a value greater or equal to the threshold value. P-tile is advantageous if the object the robot will view is known beforehand since it is computationally cheaper than the other methods.

2.3.4.2.4 Edge Pixel

The second main thresholding technique is called the edge pixel method. The value of the threshold is based on the concept of a digital laplacian operator. A digital laplacian operator is used to define the pixels surrounding the edge of the object.[16] The histogram then takes into consideration the probability density of the pixels that lie on or near the object. The points with large laplacian values are given the value of one while the pixels below this threshold are valued as zero.

If there is a large contrast between the object and the background, the histogram will have two hills dividing a long valley. If the object in the picture is only a small portion of the image, the hills and valleys become more alike. In this case, using a digital laplacian will only include the high values of the histogram. The edge pixel method is best used when the object is less significant in the picture and there is not much difference in the contrast of background.

2.3.4.2.5 Iterative or Optimal

The iterative method is the third main thresholding technique. Also known as the optimal thresholding method, it takes into consideration the mean grey level of the pixels in the image. First, it makes several passes of the object redefining the threshold level.[17] Using the values from each threshold, a mean grey level for the pixels is found by taking the average of the pixels below the initial threshold and above the last one. This value becomes the new

---

[15] A Comparison of Thresholding Methods Page 2
[16] A Threshold Selection Technique Page 2
[17] Image Processing Techniques for Machine Vision Page 1

threshold value and this process continues until these values are equal. This method is very effective because it sets a more accurate threshold level for the image in question.

2.3.4.2.6 Adaptive

The last thresholding technique is the adaptive method. In this case, the image is divided into patches. A global threshold value is found by taking the weighted sum of the thresholds for each patch.[18] This technique is much better at defining the edges of the image but a great deal of fuzziness is collected as well. This method is very effective since it is able to avoid the different lighting conditions but it is also the most computational heavy of the four techniques.

2.3.4.3.1 Filtration

Filtration is the second main image processing technique. Filtering deals with the problem of extra noise in the image. These algorithms makes it easier to distinguish an object in an image in there is too much extra pixel interference. Filter algorithms are used to either smooth or enhance the image. If an image has high frequency pixels, a smoothing filter must be applied while frequency filters are needed for low frequency pixels. To smooth images spatial filter techniques are used while frequency filters are used to enhance the image.[19] Mean, Median and Gaussian are the three main types of spatial filters. For low frequency pixels, a frequency filter can be applied.

2.3.4.3.2 Image Noise

The most important reason for filtration algorithms is the concept of image noise. Image noise includes all the extra pixels that interfere with the image. Every picture collects noise, but it depends on factors like detector sensitivity, light radiation, and other ambient parameters. Noise can be classified as either independent noise for extraneous sources or noise dependent on the image itself. Different filters are implemented based on how much irrelevant noise is in the picture.

---

[18] A Comparison of Thresholding Methods Page 2
[19] Digital Filters Page 1

2.3.4.3.3 Mean Filter

The mean filter is the easiest method to smooth images. This filter works by replacing each pixel with the average value of its neighbors.[20] This neighborhood of pixels is known as a kernel. To understand how this filter works, it is important to know about convulsion filters and kernels. A kernel is 3 X 3 image array that is used in convulsion filtering. Mean filtering works very similar to convulsion. This method involves multiplying different image pixel arrays together. The larger the array the more complex the image is. For example, grey scale pixels are entered into a 3 X 3 array. The bigger the array the more smoothing is necessary.

2.3.4.3.4 Median Filter

The second method to smooth image noise is known as the median filter. The median filter does a much better job of eliminating noise than the mean filter. This works similar to the mean filter by looking at a pixels neighbor to determine the median value. Instead of averaging these pixel values, the median is taken from the set. This grants two main bonuses over the mean filter. First, the median filter will not be adversely affected by an outlier pixel value. Second, the median value will be a pixel in the image thus edge detection is much more effective.[21] The only downside is if the image in question has too much Gaussian noise. The median filter becomes less effective if than the mean filter.

2.3.4.3.5 Gaussian Filter

The Gaussian Filter is a type of convulsion filter that removes detail and noise. Using the Gaussian distribution in 2-D, smoothing the image becomes possible with convolution methods. Since in 2-D, the Gaussian distribution is circular symmetric, convulsion can be used in both x and y directions.[22] These kernels being composed in both directions can only be done this way because they are Gaussian. The effect of the filter is that it blurs the image. The Gaussian uses a weighted average of the pixel neighborhood instead of the uniform average the mean filter uses. This allows for preserving the edges in the image better than the mean filter. This method however, is more computationally heavy then either the mean or median filters.

---

[20] Mean Filters Page 1
[21] Median Filters Page 1
[22] Gaussian Filter Page 1

2.3.4.3.6 Frequency Filter

For the frequency domain, the frequency filter is used. Since pixels in the spatial domain are computationally cheaper, the frequency filter re-transforms the frequency domain pixels into the spatial domain by multiplying the filter with a Fourier transform.[23] Since pixels in the frequency domain are in sine or cosine form, the Fourier transform is the tool used to change these values into the spatial domain. This method is only used when the other three options are not possible.

2.3.4.4.1 Edge Detection

The third image processing technique is edge detection. Edge detection is used to identify the outline of an image within its background. This is done by converting a 2-D image into a set of curves.[24] This allows the algorithm to identify the area, shape, and perimeter of objects in the image. Edge detection has a variety of methods capable of locating edge pixels including measuring edge strength and defining edge enhancement. An important concept in edge detection is known as a gradient. The gradient equation is as follows:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right] \qquad \text{2.3 (1)}$$

Equation 1: Gradient

An edge is found where there is the biggest change among pixels. This is found by finding the maximum value of the derivate at any given point. A similar indicator is the second derivative will be zero at the given point. The gradient finds the most rapid change of intensity in the image and using this value can calculate the edge strength. The edge strength equation is shown below:

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \qquad \text{2.3 (2)}$$

Equation 2: Edge Strenght

---

[23] Frequency Filter Page 1
[24] Edge Detection PPT Page 2

Every edge detection technique uses the gradient in some capacity to detect the edges in the image. The other method is known as edge enhancement. Usually, the algorithm will increase the contrast between the edge pixels and its background. Different methods produce different results depending on the situation. There are two distinct edge detection techniques called Sobel and Canny.

2.3.4.4.2 Sobel

The Sobel edge detection method is the most commonly used. This technique works by creating templates of the image to form convulsion masks.[25] These templates are then used to find the gradient at the center of the image. This allows for the better approximation of the derivates at the center point of the object. An example template can be seen here:

Figure 4: Sobel Gradient

2.3.4.4.3 Canny

The second method is known as the Canny edge detection technique. The Canny edge detector is considered an optimal technique since it needs to follow three main conditions:

1. All edges in an image need to be identified
2. Distance between edge pixels and actual edge must be very small
3. No more than one edge should be detected if only one exists

Since these steps are essential in edge detection it is imperative they work so the Canny method will work as well. The method works by first applying a Gaussian convulsion filter to the image as mentioned in section 2.3.4.3.5. After convolving in both x and y directions, the

---

[25] Image Processing Techniques for Machine Vision Page 2

magnitude is found for each pixel. This method is so effective because it removes noise while finding the edge.

2.3.4.5.1 Digital Morphology

Digital morphology is the fourth image processing technique. Digital morphology deals with the shapes of objects in an image. Shapes are a collection of pixels that form a 2-D object. These objects are referred to as structuring elements. Different mathematical methods are used to either enhance or highlight the shape in the image. Usually, geometric morphology deals with the binarization of the image pixels. Algorithms are computationally cheaper since binary data is much easier to process. This is the main way to reconstruct the pixels to their bi-level forms. Binary erosion and dilation as well as the concept of opening and closing are ways to go about geometric restructuring.

2.3.4.5.2 Binarization

Binarization is an essential concept of digital morphology. Binarization deals with simplifying the 256 grey-scale and color pixels into a bi-level image. This reduces computational speed while improving the effect of algorithm analysis.[26] Picking the right threshold value, as shown in section 2.3.4.2.1, is important for the effectiveness of the binarization algorithm. The downside is if there is too much interference in the image. The more interference, the more thresholds are needed and therefore the effect of binarization diminishes. The algorithms become more computationally heavy to compensate.

2.3.4.5.3 Erosion and Dilation

Erosion and dilation are digital morphology techniques used to effectively reconstruct geometric objects. Eliminating the set of pixels that form a pattern is essentially what erosion does. Dilation is the addition of pixels to a set area. Erosion is the complement of dilation. Erosion and dilation theory is based largely on the Euclidean space theory of unions and intersections.[27] Dilation is the union of all pixels within the structuring element. By using

---

[26] Image Binarization Page 1
[27] Recursive Binary Dilation and Erosion Using Digital Line Structuring Elements in Arbitrary Orientations Page 1

erosion and dilation together, an effective image processing tool known as opening and closing can be implemented.

2.3.4.5.4 Opening and Closing

When erosion and dilation are combined, this concept is called opening and closing. Opening is when erosion is applied to a structuring element followed immediately by dilation to the same element. Closing is the exact opposite.[28] Opening is most effective for removing the noise from an image after thresholding as occurred. Closing is very effective at smoothing the outlines of an image. Since mathematical morphology has a nonlinear nature, it is essential for image analysis.[29] To get the effect the user wants, using both erosion and dilation side by side numerous times can be very beneficial. The more steps used however raises the computational heaviness of the algorithm.

2.3.4.6 Blob Detection

The fifth image processing technique is blob detection. Blobs refer to any small, compact, bright or dark objects. This of course covers a large area of objects so there are different levels of algorithms that can be applied. There are four main factors for the effectiveness of blob detection algorithms: [30]

1. Reliability
2. Accuracy
3. Scalability
4. Speed

The algorithm must be reliable against all noise. A filter should be applied beforehand to increase the results. The sub-pixel resolution must be extremely high and be very accurate. Scalability is important because pixels of different sizes must be extracted. Since blob detection

---

[28] Image Processing Techniques for Machine Vision Page 5
[29] Image Binarization Page 1
[30] FAST AND SUBPIXEL PRECISE BLOB DETECTION AND ATTRIBUTION Page 1

algorithms can be computationally heavy based on the image and what other algorithms have been implemented, the speed of run time processing is extremely important. These four steps are crucial to Blob detection algorithm effectiveness as shown in figure 5 .

Figure 5: Blob Detection

2.3.5 Conclusion

For a humanoid robot to function, it needs the ability to identify objects and process the data collected. To accomplish this, researching how biological creatures gather image data was completed. After this step, an analysis between biology vision and machine vision was needed. Then different image processing algorithms were researched to understand what tools could be implemented. The combination of this research benefited the creation of our humanoid robot's vision system and helped to complete our goal mentioned in chapter one.

2.4.1 Motion Control

Being able to actuate the joints is one of the major componenets of our project. In order to implement this module we first needed to do some general research into the history of robotics locomotion. This gave us a solid foundation of knowledge upon which we could make future plans for our own design. The results of this reaserch is summarized in the following sections and constitute of Degree of Freedom requirements, various joint actuation methods and special considerations for the maintenance of balance during robot operation.

2.4.2 DOF Requirements

In order for the robot to perform human like action it is necessary that the robots design is one that contains enough joints and the rights amount of degrees of freedom. For the optimal design the robot would have the same amount of DOFs as a human however in terms of project feasibility, the desire is to keep the DOFs to a minimum. As such our design will

probably have approximately the standard 24 DOFs as seen in the PINO Robot. "Each leg has 6 DOFs, each arm has 5 DOFs, the neck has 2 DOFs and the trunk has 2 DOFs."[31]

2.4.3 Joint Actuation Method

These joints would all have to be actuated in some way and for this task there are 3 main options. These options are the use of hydraulics, air cylinders or electrical motors. The first two are clever solutions that are currently being developed and further researched. As such they are a bit out of our capabilities to implement, which leaves the highly practical method of employing electric motors as our only option. The past years have seen an advancement in these motors making them more light weight and powerful. DC motors are the motors of choice as they are the more reliable and energy efficient when compared to AC motors. Furthermore Brushless DC motors are compact in nature while providing even greater efficiency due to the fact there is no possibility of speed losses due to resistance between the sliding connections between the brushes and the commutator as seen in conventional DC motors. Similarly there is no bearing and brush friction loss. [32]

2.4.4 Bipedal Motion

This project requires that the robot walk around on two legs. In terms of bipedal movement there are a variety of options, starting with the major choice of how the robot will actually walk. In current practice there are two main methods of accomplishing the task of bi-pedal motion. The first is Passive Bipedal and the second is Full Actuation.

2.4.5.1 Passive and Full Actuation Methods

The Passive Bipedal method promotes efficiency as there is less actuation employed. The motion follows the dynamics of human walking by emulating the controlled fall that humans naturally exhibit. Therefore the main parts that have to be directly controlled are the upper regions such as the "hips" and "thigh" of the robot. The lower joints such as the knee and feet are made in such a way that they automatically fall in a manner to mimic a human step. On the other hand Full Actuation provides complete control over all limbs. This means that each

---

[31] RoboCup Page272

[32] Robotics Page 96

joint of the leg is actuated and can be moved independent of outside stimuli. Therefore in order for the robot to walk; all the actuation of the joints have to be programmed to work cohesively to mimic the gait of a human.

In reality both methods have their advantages and disadvantages depending on the situation in which they are employed. For the purpose of our project of creating soccer playing robot the method of Full Actuation has the edge over Passive Bipedal. While Passive Bi-Pedal has the advantage of being less cumbersome, cheaper, lighter and more energy efficient; the fact that Full Actuation allows to robot to perform more complex tasks is crucial in playing soccer. For example kicking the ball with power is more easily accomplished with Full Actuation. As such it is the method that the team will use in the development of the robot.

2.4.5.2 Balance Consideration

In order for bipedal walking to be successfully implement the issue of maintain balance is one that merits major consideration. There are two types of balancing to consider, static and dynamic balancing. Static balancing has been employed from the beginning of humanoid robot development. Dynamic balancing however is the future of robotics as it enables the robot to deal with more challenging terrain and situations.

2.4.5.3 Static Balancing

"A statically balanced system avoids tipping and the ensuing horizontal accelerations by keeping the center of mass of the body over the polygon of support formed by the feet."[33] One of the first designs using static balance was the mechanical horse patented by Lewis A. Rygg in 1893. Since then there has been ongoing research in this area. From this research the Zero Moment Point Theory was developed and first postulated by Miomir Vukobratovic in 1963. The theory simply states that by keeping the moments of the robot at a value of zero balance can be maintained during walking. This can be achieved by counteracting the moments cause by the movement of the legs by moving the upper body of the robot so that the sum of the moments about the COG is zero.

---

[33] Legged Robots Page 4
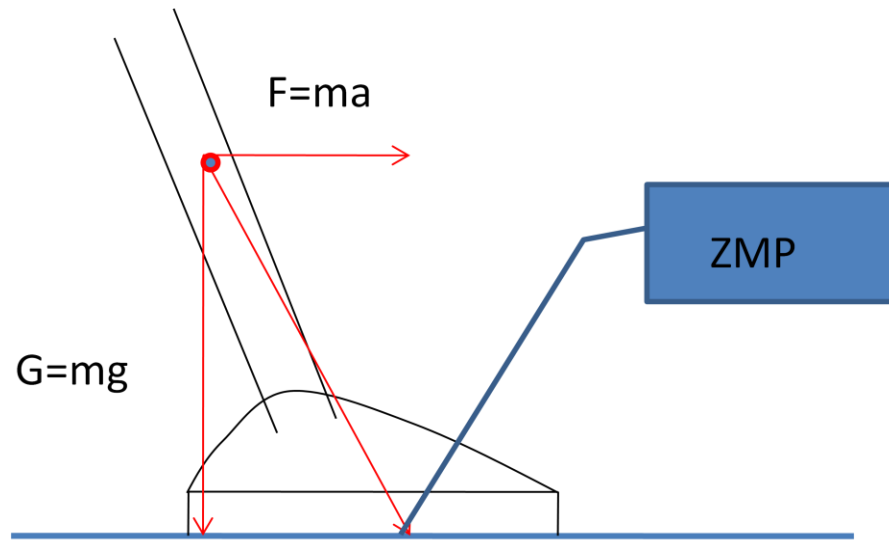
Figure 6: ZMP

2.4.5.5 Dynamic Balancing

While the method of static balancing is effective it is limited by the fact that its steps are all preplanned in nature. As such it cannot respond to an unexpected outside stimulus such as a sharp push or the degradation of terrain. This is where dynamic balancing comes to the forefront as the optimal solution. "Unlike a statically balanced system that must always operate in or near a state of equilibrium, an actively balanced system is permitted to tip and accelerate for short periods of time."[34] This means that self recovery of balance is an accomplishable task making the robot more human-like in nature. This system is based on the concept of balancing an inverted pendulum and probably first developed by Claude Shannon in 1951. Here he used an electric cart to move forward and backwards to balance an inverted pendulum. This was the start that led to the development of the first legged robot to employ this method of dynamic balancing. The robot was built Miura and Shimoyama in 1984 and was the first walking machine that actively balanced.

2.4.5.6 Chosen Method

For our project we will be using the principle of static balancing and hence the zero moment point theory to achieve stability while the robot is operating. This is chosen as it is the easiest to implement, the most cost effective and the least time consuming method. Also

---

[34] Legged Robots Page5

dynamic balancing is a luxury as the terrain in question is topographically flat and the robots, as far as we can discern are not allowed to run into each other. Therefore there are no outside forces and as such dynamic balancing cannot be considered as a requirement. With static balancing we will be able to investigate the first method of balancing. We will also be able to keep the robot upright during activities such as walking, turning and kicking which is required to play soccer.

2.5.1 Artificial Intelligence

Critical to any system is an effective means of control thus any parameters for the robot are key. To be competitive on a world standard, a soccer robot's artificial intelligence (AI) must be efficient in guiding the robot through obstacles and other challenges, as well as detecting balls, goal posts, friends and foes, and being able to maneuver around them. Any mechanical advantage would amount to naught if the robot's AI cannot leverage it towards making a successful goal in a real competition environment. This section will explain the different AI programming techniques and structures that can and have been used on robot soccer platforms, demonstrating methods to improve the effectiveness and performance of the robot.

2.5.2 Human Emulation

In most cases, robot soccer agents are programmed dauntingly by hand. One alternative would be to use machine learning (ML) techniques to either program the robots with human playing styles and behaviors, or even to adapt and predict the movements of the opposing team on the field. The robot minds, known as agents, can be programmed using models derived from pairing the sensor data inputs on the agents to human defined outputs or actions. The main advantage of this method is that it makes use of the human players' ability to play many games well and quickly, thus making sense if it is transposed into a robot domain. By emulating human playing behavior, the agents' AI can be tuned to enact more fluid and natural styles of play.

ML techniques are no strangers to the programming field. In video games, such as those in the first person shooter (FPS) genre, Thurau, Bauckhage, and Sagerer[35] (2004) provide a good

---

35   Programming Robosoccer Agents, page 4

summary of how imitation can be used at reactive, tactical, and strategic levels. Using the FPS game Quake II, they built a MATLAB interface to allow a human to play the game while simultaneously recording pairs of in-game state-vectors and actions. A self-organizing map was used to reduce the dimensionality of state-vectors, and then used multi-layer neural networks to map state-vectors to actions. Neural networks are also used to learn trajectories, aiming behavior, and their various combinations,[36] making better models that can take context into account before performing an action. Primitive movements are extracted as blocks for more complicated strings of movement, and conditional probabilities on the state-vector and the last action are learned.[37] Other research by Priesterjahn, Kramer, Weimer, & Goebels (2005) use genetic algorithms to improve imitation models.[38] Alternative research involves agents modeling opponent behaviors and using said model to better defeat them.

More specific to AI challenges in a soccer domain, research has already been done into improving the effectiveness of agents in a virtual soccer environment, devoid of actual physical robot platforms. Riley, Veloso, and Kaminka (2002) use ML techniques to create a coach agent in a Robosoccer[39] domain. The coach can learn from previous games, using models to predict team formations and passing behavior in order to beat the opposing team. For instance, models can be used to best select the ideal plan to defeat the opponent in set plays.[40] Models are not learned in some cases, and instead are predefined and used to classify the opposing teams through a similarity metric.

Aler, Valls, Camacho, and Lopez look into transforming Robosoccer into an interactive game, permitting human control of a Robosoccer agent and then use ML techniques to clone his/her in-game behavior. The figure below shows the GUI soccerclient that allows a human user to play Robosoccer as a video game, designed in such a way that the only information displayed to the user is the same as the one available o the actual agent in the simulation field.

---

36   *Ibid*, page 4
37   Programming Robosoccer Agents, page 4
38   *Ibid*, page 4
39   The Robosoccer simulator is a challenging environment for artificial intelligence, where a human has to program a team of agents and introduce it into a soccer virtual environment. (*Ibid*, page 1)
40   *Ibid*, page 3
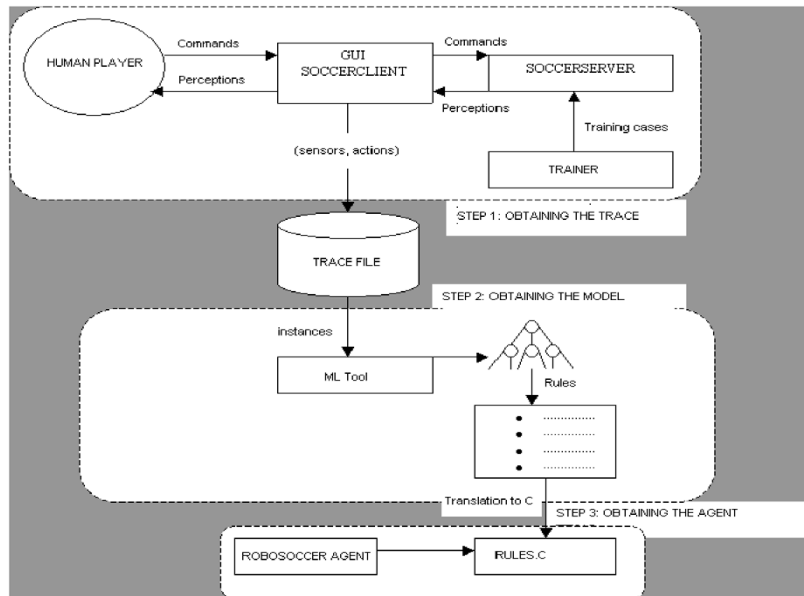
Aler et. al conclude that using ML algorithms are indeed effective, at least in the low-level behaviors of their scope, and the modeled agent was able to provide a challenging experience for a human. Key to providing the user feedback is a user-friendly and responsive interface for human play,[41] and a responsive interface is the crux for learning low-level behaviors.[42]   The Soccerserver interface used by Robosoccer was not originally conceived as a video game, thus limiting the level of interactive play compared against commercially available games. Their method works best on reactive behaviors (mapped input-outputs), but degrades when the action of the human depends on hidden variables, like making use of human memories and predictions about the opponents.[43]

In terms of further study, more attention could be applied toward cognitive functions applied by a human playing Robosoccer, such as planning, opponent prediction, and trajectory computation.[44]   Using new attributes, they plan to add estimations of some of these hidden variables to the agent. Special algorithms can also be used, for instance, to track objects even as they go out of view, in the same way that human memory works. By also pre-programming all

---

41   Programming Robosoccer agents, page 8
42   *Ibid*, page 9
43   *Ibid*, page 9
44   *Ibid*, page 9

lower-level behavior, the human player can focus more on higher-level decision-making and strategy, leaving the execution of more rudimentary details to the computer.[45]

2.5.3 Team Coordination Architecture

With the goal of having a full robot team best the reigning human soccer champions by 2050, robot players must not only have the individual coordination that the best human players posses, but also teamwork and strategic movement the best teams employ. The benefits for robot coordination affect the FIRA challenge as well as various commercial and industrial applications for robots, such as cleaning work and assembly technologies. Developing suitable controllers and architectures is important for advancing collective robot performance.

Camacho, Fernandez, and Rodelgo (2005) explore technologies for intelligent systems and multi-agent systems (MASs),[46] coming to describe an instantiation of SkeletonAgent, a multi-agent framework capable of integrating classical AI techniques to build intelligent systems, into the robot soccer domain. Likewise, Harmati and Skrzypczyk (2008) describe collision free target tracking problem of a multi-agent robot system.[47] Both papers approach the problem of optimizing robot agent coordination, though not necessarily limited to robot soccer in their application. SkeletonAgent is a software agent framework that can be used to help concert movement between multiple robot agents, and is described by the figure below[48].

---

45   Programming Robosoccer agents, page 9
46   Roboskeleton
47   Robot team coordination
48   Roboskeleton, page 2
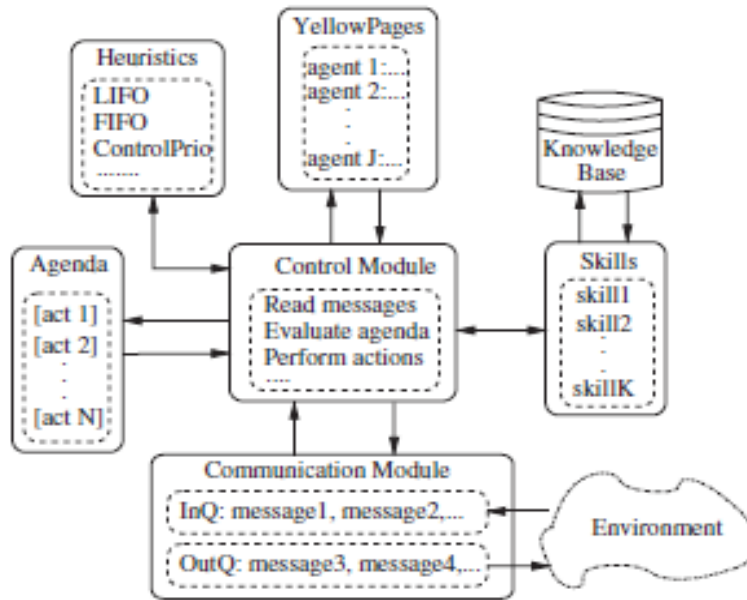
Figure 8: Skeleton Agent Architecture

The framework is divided into the following modules, interconnected as in the previous figure:

- *Agenda/skills*: This dynamic structure stores items called *acts*, which are correlate to the skills of the agent. Each act contains a list of lower-level skills/routines that are executed in the proper succession to perform the desired act, and when the particular act cannot be further broken down, the agent executes the base-level act.[49] Camacho et al. (2005) show that if the skill can be decomposed into several sub-acts, these can be independently managed in the agenda.[50] This builds upon compounding small, simple tasks into progressively larger and more complex outputs.

- *Heuristics/control module*: This decides what act to select from the agenda at any time, according to a preselected control policy, such as last in/first out (LIFO) or first in/first out (FIFO).

- *Knowledge base*: archives the knowledge that can be used by agent skills, such as files, databases, tables, etc.

- *Yellow pages*: stores the information about all agents belonging to the team, consisting of a list of partners and a list of their corresponding skills.

---

49 Roboskeleteon, page 2
50 *Ibid*, page 2

- *Communication module*: responsible for serializing and de-serializing information to and from other agents on the network, as decided by the control module.

Harmati and Skrzypczyk tackle the problem of generic robot formations, where team members have to follow the target and achieve a favorable position relative to other teammates, targets, and obstacles/opponents. Their method is to convert formations, essentially a control problem, into a game theoretic problem and provide solutions in this framework. To improve team performance, they build upon the semi-cooperative Stackelberg equilibrium[51] proposed by Harmati (2006).[52] System robustness is improved with the control of cost function component with a PD-like fuzzy controller. Santos (2002) proves soft computing techniques to be efficient in poorly defined system organization,[53] and Hwang, Tan, and Chen (2004) prove that it is also effective in multi-agent coordination[54].

The sample of methods described exemplify the types of options and possibilities available in tuning the AI of robotic systems, all of which can be applied toward improving the performance of individual soccer robots and teams.

2.6.1 Control Systems

At the heart of every robot are its behaviors, the sum of its abilities to receive inputs from its surroundings, perform due processing and calculations from this information, and then promptly affect said surroundings through some means. This is in essence a closed-loop system, where a controller manipulates the system inputs to obtain desirable effects on the output of the system. What defines "desirable" conditions is part of the behavior proponent of a robotic system. Without an effective control system, a robot will not be able perform its desired tasks successfully, whether that would be cleaning floors, disabling landmines, or kicking balls in between goal posts.

---

51   Stackleberg equilibrium: A game theoretic equilibrium in which one player acts as a leader and another as a follower, the leader setting strategy taking account of the follower's optimal response. (http://www-personal.umich.edu/~alandear/glossary/s.html)

52   Robot team coordination, page 2

53   *Ibid*, page 2

54   *Ibid*, page 2

This section will delve into possible control models for humanoid robots. One particular consideration is the shape; since a human-shaped robot has a relatively high center of mass while upright, the stance can be unstable and prone to tipping. The robot would have to propel itself forward on its bipedal drivers, and still be in control of its center of gravity for maximum stability and speed.

To be able to play soccer against human opponents, much less a top-tier team, the robot has to identify teammates, opponents, boundaries, and the ball itself. It also needs to balance itself through many dynamic situations, and effectively maneuver around changing and static obstacles. On collective higher-level actions, robot teams must be able to quickly coordinate and move towards the best positions to materialize a goal. The processing in control units must be done in real-time, so variables like CPU size, current consumption, memory type and capacity, on-chip peripherals, as well as the level of development tools are important factors that influence the selection of the control processor and algorithms used in a particular mobile robot.

2.6.2 Gait Generation

Effective gait generation is the means by which a bipedal humanoid robot can move in a quick yet balanced manner, having the robot's legs move through calculated steps and positions. Gait planning for humanoids is essentially different from path planning for serial robotic arms, since the center of mass is in constant motion while the feet periodically interact with the ground in a unilateral way. Since this means that there are only repulsive and no attractive forces between the feet and ground, ground interaction must thus be planned out to avoid postural instability.[55]

To further understand dynamic stability and help monitor and control a walking robot, it is useful to understand the concept of zero moment point (ZMP).[56] The ZMP is the point on the ground where the sum of tipping moments on the robot, due to gravity and inertia forces, equals zero.[57]

55Observer-based Dynamic Walking Control for Biped Robots, page 3
56Observer-based Dynamic Walking Control for Biped Robots, page 3
57*Ibid*, page 3
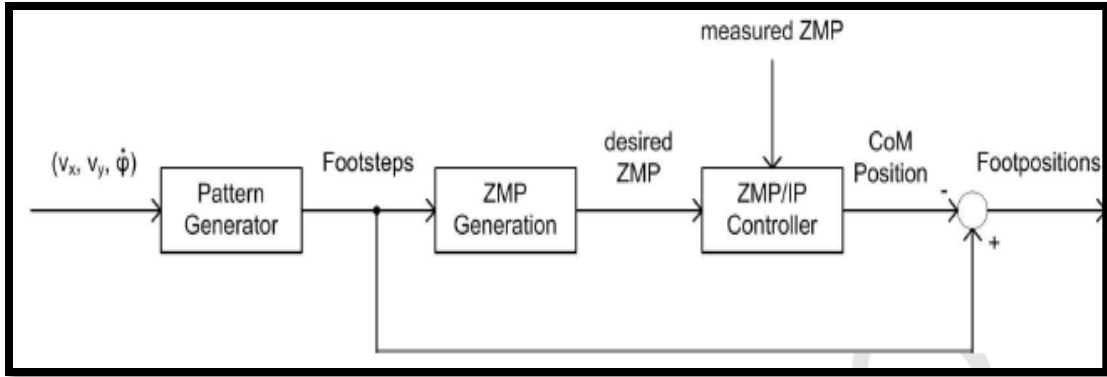
In the figure above, one can see the how the close-loop system of the gait is run, determining the best placement of foot and body movement all to keep the resulting aggregate motion stable. The gait generation takes in the desired translation and rotational speed vector (V_X, V_Y, φ) as the input, which is not synonymous to the center of mass's (CoM) speed, but rather its desired average. The playing field is designed to be fully flat, but more robust gait generation algorithm might take into account uneven terrain, resulting in irregular gait patterns at different velocities.

2.6.3 Trajectory Following

Control systems also guide the robot through a determined path, and are responsible for the fidelity of the trajectory interconnecting the start and target positions. On wheeled mobile robots or serial robot manipulators, a control system would involve the use of an inverse kinematic model, which calculates the necessary movement needed by a system to reach a desired end location in space.

The robot's desired movement can be described into a given ZMP trajectory, which can be calculated using a simplified model using the center of mass to represent the entire body. As only one foot is connected to the ground during the walk, and considering only that contact point and the center of mass, the system can be likened to an inverted pendulum. Contracting or extending the leg can alter the robot's height altered, allowing greater control over CoM trajectory. By restricting the inverted pendulum such that the CoM only moves along an

---

[58] Observer-based Dynamic Walking Control for Biped Robots, page 4

arbitrary plan, it results in simple linear dynamics called the 3D Linear Inverted Pedulum Mode (3D-LIPM).[59]

Walking on overall flat terrain defines the constraint plane as horizontal (k_x = k_y = 0), despite slight deviations in level consistency. Using a global coordinate frame, we use *m* as mass of the pendulum, *g* for the gravity acceleration and τ_x and τ_y as the torques around the x- and y-axis, the following equations[60] display the pendulum's dynamics:

$$\ddot{y} = \frac{g}{z_h}y - \frac{1}{mz_h}\tau_x \qquad \text{2.6 (1)}$$

$$\ddot{x} = \frac{g}{z_h}x + \frac{1}{mz_h}\tau_y. \qquad \text{2.6 (2)}$$

Equation 3: Plane Acceleration

According to this model, the position (p_x, p_y) of the ZMP on the floor can be calculate using the following equations[61]:

$$p_x = -\frac{\tau_y}{mg} \qquad \text{2.6 (3)}$$

$$p_y = \frac{\tau_x}{mg}. \qquad \text{2.6 (4)}$$

Equation 4: Position

From there, a simply substitution of equations 3 and 4 into equations 1 and 2 yields the following ZMP equations[62]:

$$p_x = x - \frac{z_h}{g}\ddot{x} \qquad \text{2.6 (5)}$$

$$p_y = y - \frac{z_h}{g}\ddot{y} \qquad \text{2.6 (6)}$$

Equation 5: ZMP Position

---

[59] Observer-based Dynamic Walking Control for Biped Robots, page 5
[60] *Ibid,* page 6
[61] *Ibid,* page 6
[62] *Ibid,* page 6

It can be seen that for a constant height z_h of the constraint plane, the ZMP position depends on the position and acceleration of the CoM on the plane and the x- and y-components can be addressed separately.

From the equations given, we can ascertain the required ZMP positions needed to as the controller outputs, thus propelling the robot onward with stability.

2.6.4.1 Hardware Background and Operation

This section describes the different critical hardware involved in the processing and operation of the HUST soccer robot. Each unit will be described in its general functionality, and how it works together with the robot as a whole. This will serve to explain the importance of each unit, and the considerations that need to be made to ensure the best performance from given resources.

2.6.4.2.1 Robot Board Components

2.6.4.2.2 Microprocessor

The microprocessor is essentially an entire central processing unit (CPU) onto an integrated circuit (IC). They are for the most part general purpose, although there are different types dedicated for specific uses. At the core of all robots is a microprocessor of some type, usually a microcontroller, used for the robot's central operations. They contain within them areas called registers that store numbers to be used for data manipulation, and areas called arithmetic logic units (ALUs) that perform arithmetic or logic operations on that data.

2.6.4.2.3 Digital Signal Processor

Digital Signal Processors (DSP) are specialized microprocessors with architectures designed specifically for the types of operations required in digital signal processing; it takes in a signal and manipulates it mathematically through digital means. They are programmable devices, running their own native instruction code, and can perform millions of operations per second, and gain further performance advantages by their dedicated architectures.

Firstly a signal must be defined. A signal is stream of information representative of a certain condition, and in this particular application it is an electrical signal. Since most data,

especially as representations of the "real world," come in analog or continuous representation, they need to go through an analog-to-digital converter (ADC) so that digital processing can be applied. In most cases, a signal is a voltage or a current that represents the condition, such as temperature, pressure, and so on.

Signals often need to be processed to remove unwanted information, such as noise and other extraneous data, for further simplification down the line and to make the information more immediately useful. For the robot's own purposes, the signal is to processed is a video stream from the CCD camera mounted at the robot's "head." The basic idea of digital signal processing is that the image seen by the robot needs to be reduced to a simpler form and thus give the robot a better ability to recognize key objects, such as white lines, yellow tennis balls, or blue goal posts. By reducing the image to what is truly needed, it makes it much easier for an embedded processor to take these in as inputs and produce movements that would be in line with our strategies.

To get the best performance for a particular use, factors like operations per second, power consumption, cost, and unique features of a particular processor all add up toward judging which is the best processor to use. As mentioned before, these devices run their own native instruction code, which can be optimized for ease of use, better functionality for specific tasks (like audio and/or video processing), or compatibility with other hardware. Given the synergy of all these factors, a manufacturer's provided sheet might not immediately be the best source for comparison between different DSPs, and further investigation must be made into their more particular features.

2.6.4.3.1 Servos

The primary actuators in the robot are servos, and this section describes the basics of servo operation and control. A servo is essentially an electromechanical device that controls the angle of a load of rotating around a pivot. The servos at each joint are responsible for the direct actuation of knees, ankles, hips and feet. Different servos are shown the following figure:

Figure 10:Different Servo Sizes

2.6.4.3.2 Basic Design

A servo system consists of:

- Power supply
- Interface
- Position Controller
    - Servo Controller
- Servo Motor (typically limited by a mechanical stop)
- Feedback sensor

The following figure illustrates how the different parts interact with each other to enable effective and accurate servo control.
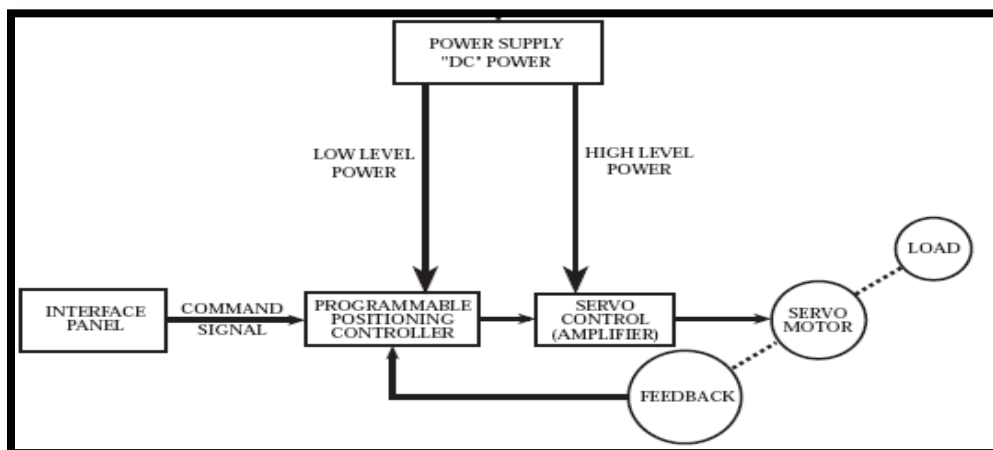
Figure 11: Servo Control Loop

2.6.4.3.3 Servo Control

To communicate to the servo which angle to at, it uses a method called Pulse Width Modulation (PWM). By PWM, the servos assigned angle is determined by the length of an electric pulse sent to it. Pulses are sent in at a regular frequency, but by modulating the width of these pulses, one modulates the position of the servo. The figure below illustrates the example of PWM control.
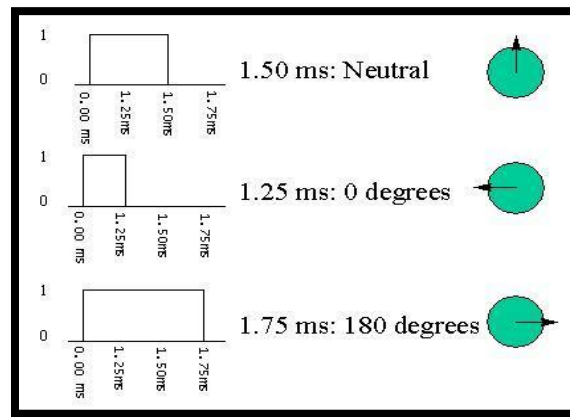
Figure 12: Example of PWM Control

Servos usually work using proportional control to reach the desired position. The further the actual servo position is from the value assigned by the pulse width, the faster the motor would have to move, and the closer the position is the slower the motor would be.

2.6.4.3.4 Servo Connections

Since servos receive electric pulses, they can be connected to the outputs of a control board. Counters and timers on embedded systems can be used so that users can vary the pulse through the program they download onto the system chip. A downside of this method of control is that some boards may have limited outputs, counters, or timers, and having to control a large swath of servos might be impossible without using more resources.

Alternatively, some servos may have their own dedicated microcontrollers on each servo unit, and instead of directly receiving pulses they can receive byte commands over a serial connection. The microcontrollers would then be responsible for sending pulses to its own servo. These servo units can be daisy-chained together with no problem, since each byte command would be designated to a particular servo through its ID number. Not only can data

be sent to the servos, the servos can also return data packets back with information about themselves such as feedback measurements and such, eliminating the need for extraneous sensors. With this type of servo, multiple units can be connected to the control board via a single serial port connection.

2.6.4.3.5 Joint Connections

Actuators like servos and motors have different means to be connected to other effectors. Apart from the mechanical advantages gained by placing these actuators in front of gear trains, they can also be connected via chains, belts, and linkages, each offering their own particular advantages and tradeoffs in terms of performance.

2.7.1 Project Plan: Ball Kicking

To effectively complete our goal, we needed to create a set of objectives to follow. We developed four main objectives to complete our project. First, we created an image processing algorithm strategy to allow the robot to locate the object in the field and discern it from the background. Second, we developed an optimal ball kicking strategy to implement. This strategy was tested and researched to be the most effective in path designation and kicking mechanics. Next, we converted both strategies into DSP code. This had to be done so the robot could understand the programming and know how to react. Lastly, we tested the movement of the servos and the entire system while debugging any problems.

We had to make sure the robot moved effectively and accomplished all the tasks needed. This included following all the rules of the game and kicking the ball into the goal. These tasks were divided among our group members over the course of our project. Chapter 3 is a more detailed explanation of our objectives and what we designed.

2.7.2 Project Plan: Path Tracking

To effectively complete our goal, we needed to create a set of objectives to follow. We developed four main objectives to complete our project. First, we created an image processing algorithm strategy to allow the robot to locate the white line and discern it from the background. Second, we developed an optimal path strategy to implement. This strategy was tested and researched to be the most effective in path designation and determination. Next,

we converted both strategies into DSP code. This had to be done so the robot could understand the programming and know how to react. Lastly, we tested the movement of the servos and the entire system while debugging any problems.

We had to make sure the robot moved effectively and accomplished all the tasks needed. This included following all the rules of walking 42 meters in 15 minutes. These tasks were divided among our group members over the course of our project. Chapter 3 is a more detailed explanation of our objectives and what we designed.

2.8 Conclusion

The goal, as stated in chapter 1, was to develop a humanoid robot capable of displaying effective soccer skills and techniques. By completing preliminary research in general robotic history and the four main areas of robotic design we were able to build a solid base. By understanding image processing, motion control, artificial intelligence and robotic control systems we gained the knowledge to program our respective robots. This foundation helped us create a project plan for each robot. These outlined steps made it possible to complete our goal.

Chapter 3: Methodology

3.1 Introduction

To complete our goal as stated in chapter 1, we needed to complete several steps. We first separated into either the ball kicking or path finding robot group. Although similar, each group's methods were slightly different as each robot had its own unique task to accomplish. We divided the completion of the project into four major objectives for each group. These objectives were similar but they were catered to the task each robot needed to complete.

For the Ball Kicking group, we first developed an image processing algorithm strategy. Once this was developed, an optimal kicking strategy was designed. After the designs were tested and in place, we needed to convert these methods into DSP so the robot could function. Next, we tested our code with the movement of the servos and examined how well they worked while debugging any problems. Finally, after debugging, we completed our ball kicking robot.

For the Path Finding group, we first developed an image processing algorithm strategy. Once this was developed, the optimal path finding strategy was designed. After the designs were tested and in place, we needed to convert these methods into DSP so the robot could function. Next, we tested our code with the movement of the servos and examined how well they worked while debugging any problems. Finally, after debugging, we completed our path finding robot.

3.2.1 Ball Kicking: Objective 1: Create Image Processing Algorithm Strategy

Our first objective for the ball kicking group was to create an image processing algorithm strategy for our robot. We created our algorithm from our background research done in section 2.3.1. By researching the different algorithms, we were able to develop a strategy for our robot. Using this foundation, we picked the best algorithms for our image processing method. For the ball kicking robot, the first and most important step was for the robot to be able to identify the ball with a CCD Digital Camera. For this we used the following mathematical algorithm techniques together in chronological order:

1. RGB Color Segmentation

2. Color Binarization

3. Auto Thresholding

4. Median Filtering

5. Blob Detection

Each individual algorithm was chosen based on effectiveness and high computational speed. Although a more direct algorithm would create a better result, we went for a method that finds the ball every time while not being computationally heavy. Since we have a controlled environment with only one yellow object on a green background, we designed our method taking this into consideration.

3.2.2 Key Data

Most of our key data for this objective was collected qualitatively. We utilized our prior research into the different types of image processing seen in section 2.3.1 to develop our strategy. We chose the algorithms to implement for the tasks the robot needed to accomplish. After looking at the robotics' club design, we found where the process could be made more efficient and less computationally heavy. The algorithm process we created was then implemented in Matlab. Then using the programming language described in 2.3.3.3.1, we tested our strategy.

3.2.3 Deliverable

At the end of this objective, we created an image processing algorithm for which our robot could use. This method was researched and tested in Matlab. After researching different methods to use, we designed the best strategy for our robot. Using this program, our robot was able to process the images and locate the ball effectively.

3.3.1 Path Tracking: Objective 1: Create Image Processing Algorithm Strategy

The first objective for the path tracking group was to create an image processing algorithm strategy for the robot. We created our algorithm from our background research done in section 2.3.1. The primary goal for image processing was to extract the track from the video images. In order to do this in the most efficient and effective manner a few different

options were initially tested and the best was chosen and optimized. Listed below are the algorithms that were used in the order in which they were applied.

1. Binarization

2. Erosion

3. Linear Angle Calculation

Each individual algorithm was chosen based on effectiveness and high computational speed. Although a more direct algorithm would create a better result, we went for a method that finds the path every time while not being computationally heavy. Making sure the robot would not lose the path; we made sure it could discern angles and curves in the direction. An image processing method had to be adaptable and work for any line the robot would detect. Since we have a controlled environment with only one white path on a green background, we designed our method taking this into consideration.

3.3.2 Key Data

Most of our key data for this objective was collected qualitatively. We utilized our prior research into the different types of image processing seen in section 2.3.1 to develop our strategy. We chose the algorithms to implement for the tasks the robot needed to accomplish. After looking at the robotics' club design, we found where the process could be more efficient and less computationally heavy. The algorithm process we created was then implemented in VC++. There using the programming language described in 2.3.3.3.1, we tested our strategy.

3.3.3 Deliverable

At the end of this objective, we created an image processing algorithm which our robot could use. We developed an interface to implement our algorithm strategy. (Appendix A) The code was written in VC++ and tested using this language. (Appendix B) This interface was able to load camera images and apply the algorithms to locate the path in the picture for the robot. After researching different methods to use, we designed the best strategy for our robot. Using this program, our robot will be able to process images while locating the line effectively and continuously.

3.4.1 Ball kicking: Objective 2: Create Ball Kicking Strategy

The second objective for the ball kicking group was to create a goal scoring strategy. To accomplish this task, we developed two strategies for two different scenarios. This way, we had more choices and were able to compare the strengths and weaknesses of each. Our first challenge was to score three goals out of five attempts. The second scenario was to score three goals while a barrier was placed in front of the goal. Each scenario required necessary steps to complete the task successfully. For each strategy the robot needed to turn at an acute angle and kick the ball in a diagonal direction.

3.4.2 Strategy 1 w/o Barrier

For this strategy, there are a few key characteristics. First, the camera can move in four directions: up, down, right and left. The field is divided into six different camera positions so the robot can see any corner of the field. Each section is called a quadrant. Quadrants are an easier way to spot the ball in the field of view. When the image of the ball is in view, the camera automatically turned until the image of the ball was in the center quadrant. The distance of the robot to the ball was calculated by using the height of the robot, the angle of camera movement, and the distance between the robot and the ball. The distance was divided into the X direction and Y direction. By setting the distance of each step, the robot knows how many steps it needs to move towards the ball. Also, during the walking process, it stopped to readjust for error if it lost sight of the ball.

3.4.3 Strategy 2 w/o Barrier

This method used a similar way to collect image data as mentioned in section 3.4.2 but used this information a bit differently. Instead of walking to the quadrant with the ball, the robot gathered an image after each step. It did not calculate the distance between the object while moving forward blindly. It needed to keep receiving feedback in order to keep moving and continually halted until more information was processed.

4.4.4 Strategies with Barrier

To complete our second scenario, we needed to implement a different strategy when faced with an obstacle in the goal. For strategy one, instead of dividing the camera view into six quadrants mentioned in section 3.4.2, the area was divided into ten different quadrants. Each quadrant represented a different angle with which the robot could maneuver. This way the robot had a larger view of the playing field and can plan accordingly. The robot made decisions based on the strategy implemented in section 3.4.2.

Strategy two involved changing the angle with which the robot turned. This was based on the data gathered from the image. When the robot looked at each quadrant, it scanned more of an area. This way it was able to see all obstacles as well as the goal and barrier.

3.4.5 Key Data

This objective was completed by researching and then designing our own kicking strategies. Taking the approach and comparing with previous approaches we were able to pick the optimal one. Both strategies had their advantages and disadvantages. After examining what method was best suited for our situation, we were able to create and use such a strategy for our robot. By comparing this strategy to the robotics club design, we could find what was more effective and therefore enhanced its movement.

3.4.6 Deliverable

A final ball kicking strategy was implemented at the end of this objective. We were able to discern and design which method would work best for us. When we combined this strategy with our image processing strategy from section 3.2.1 we could program our robot to walk a line.

3.5.1 Path Tracking: Objective 2: Create Path Tracking Strategy

For the second objective of the path tracking group, we needed to implement a path finding strategy for our robot. For our purposes there were two strategies that we found that were most suitable to be employed. The first was the area calculation method and the second

was the angle calculation method. Both methods can be used but have their respective positive and negative aspects.

3.5.2 Area Method

The area method is the cruder of the two methods. With this method the processed image was first divided into 3 equal horizontal areas. These areas are the immediate position, future position and long term position from bottom to top. The bottom two areas are then further divided into 3 equal blocks per row. This gives the robot's vision a sense of direction; namely left, center and right. In order for the robot to process the image and determine which direction to travel; all 3 blocks in the immediate position and future position are analyzed. Information from the long term position was considered surplus and ignored to increase processing speed. Each block was scanned for the one containing the largest area of white pixels. This block was chosen as the direction the robot needed to travel. Higher priority was given to future blocks as they allowed the robot to almost anticipate the turn and act more intuitively.
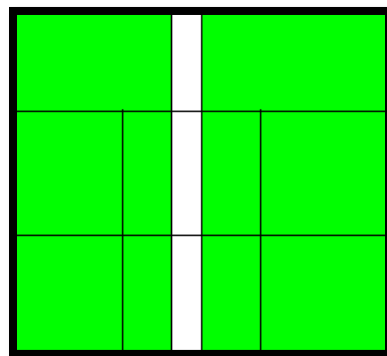
Figure 13: Image Division for Area Method

3.5.3 Angle Method

The next strategy was the more sophisticated angle method. First, we scanned all the pixels in every fifth line. If the line had more than 20 white pixels, we calculated the coordinates for the center of the line. We set a threshold of 20 to avoid collecting pixels that would corrupt the image. Next, we set the first center point as the start point. The calculation was then computed again and the previous center point became the final point. Finally, we formed a line with the starting and final point. The angle was calculated between that line and the vertical line. The robot then moves in the direction of the calculated angle.

42
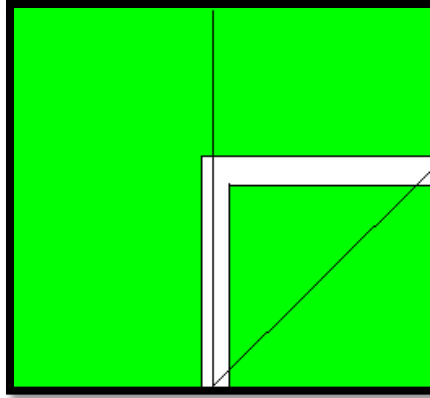
3.5.4 Key Data

To implement this objective, we had to use qualitative research to identify the best methods available. Both angle and area method had their advantages and disadvantages. After examining what method was best suited for our situation, we were able to create and use such a strategy for our robot. By comparing this strategy to the robotics club design, we were able to find what was more effective and therefore enhanced the robots movement.

3.5.5 Deliverable

A final path tracking strategy was implemented at the end of this objective. We were able to discern and design which method would work best for us. When we combined this strategy with our image processing strategy from section 3.3.1 we can program our robot to walk a line.

3.6.1 Objective 3: Convert Processes into DSP Language

For our third objective, we needed to translate our image processing and walking strategies into a language the robot could understand. We were regulated into using a DSP module for both the ball kicking and path finding robots. As explained in section 2.6, DSP is programmed directly into the processing board so no human interference is necessary. This of course made it difficult to convert our image processing algorithms programmed in Matlab and VC++.

3.6.2 Key Data

For this objective, both groups first learned how to program in DSP. This was done by extensive research and help from the HUST robotics club. Once we were able to get the servos to move properly, we could begin implementing the rest of our code.

3.6.3 Deliverable

A fully coded electronic processing board was operational for both robots when we completed this objective. The DSP was written and burnt into the board for both the ball kicking robot (Appendix C) and path tracking robot. (Appendix D)

3.7.1 Objective 4: Debugging and Servo Testing

Testing and debugging the servos was the fourth and last objective for both ball kicking and path finding robots. After completing objectives one to three, we had both a ball kicking and path finding robot that would work in theory. Unfortunately, we needed to debug any problems that arose to get the robots to function at maximum capacity. We developed a testing interface to see what input and output values each servo had. (Appendix E) The interface was designed in VC++. (Appendix F) Testing the robots took quite a bit of time and effort but to complete our goal it was extremely necessary.

3.7.2 Key Data

To completely debug our robots, we needed to research and learn about the different problems that would arise. By understanding how the robots worked before, we could then compare the debugging process and implement our own strategy.

3.8 Final Deliverable: Ball Kicking Robot

After completing objectives one through four, we had a working ball kicking robot. The robot successfully located the ball on a level single color playing field using the image processing algorithm we developed. It moved towards the ball and was able to position itself accordingly using the ball kicking strategy we developed. The servos moved in unison because of the translation of our strategies into DSP code. The robot did not fall over because of our servos testing. We were able to program a fully operational ball kicking robot able to kick a ball five out of five times into a net.

3.9 Final Deliverable: Path Tracking Robot

After completing objectives one through four, we had a fully functional path finding robot. The robot successfully located the path using the image processing algorithm we developed. It did not deviate and was able to successfully navigate turns and angles. The path finding strategy we developed worked extremely well since it did move faster than the previous model. The servos moved in unison because of the translation of our strategies into DSP code. The robot did not fall over because of our servos testing and they moved well in unison. We were able to program a fully operational path tracking robot able to follow a 42m path.

3.10 Conclusion

We completed our goal by following the four objectives laid out in this chapter. First both groups designed an image processing strategy. Then, the gait strategy was tested and designed. After both strategies were chosen, they were converted into DSP code. Finally, the code was debugged with the real robot and corrected accordingly. After these tasks were finished, each robotic group had a functioning robot capable of displaying soccer techniques and skills. Referring to sections 3.8 and 3.9, we explain the total finished product and process for each group.

Chapter 4: Findings and Analysis

4.1 Introduction

Our methods and goals for this project were centered on the completion of a fully functional ball kicking robot and path tracking robot. We organized our finding section based on our methods and research of robotics. Our findings were based on research, prior knowledge, previous development of humanoid robots and technical skill in programming and mathematics. Upon arriving at HUST, we received tangible specifics on the robot platform to be further developed. For our project, the robotic system had already been completed by the HUST Robotics club. Therefore, no new mechanical design was necessary. However, an analysis was necessary to gain insight into the movement and functionality of the platform.

First, we conducted a mechanical design analysis of our robot, and then calculated the impact of force on the upper and lower parts of the robot. Proving the force was negligible, we calculated the ZMP effect on gait planning. Next we created an image processing strategy for the ball kicking and path finding robots. Each algorithm strategy was researched and effectiveness was proven using mathematical reasoning.

After finishing these steps, we had to integrate gait strategies for both the ball kicking and path finding robots. Using the tools available, we choose an effective ball kicking strategy for a field with and without a barrier. For the path finding gait strategy, we picked a method that was more successful than its counterpart. Finally, by putting everything together, we programmed both the ball kicking and path finding robots. We described how each robot completes its tasks and functions. Based on our findings we formed our recommendations and conclusions.
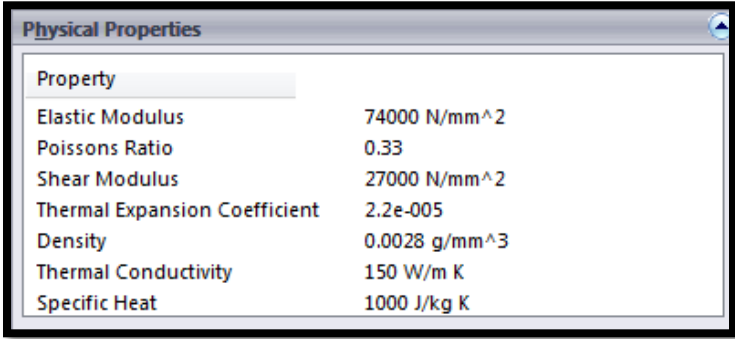
4.2.1 Mechanical Design Analysis

4.2.2 Weight Distribution

In order to gain a better understanding of how changes in the walking gait would affect the robot overall stability it was necessary to do a weight distribution analysis of the robot system. The physical model was not available to do testing and experiments on, such taking the actual

weight of the body components or finding the maximum angles of lean before breaking the limits of stable equilibrium. As such we were left with the option of doing rough analysis of robot by using the Solid Work's Model previously created by the Wuhan Robotics Club. (Appendix G)

Before we could begin the analysis however, several modifications had to be made to the model. The first and most time consuming was changing the material properties of all of the metal components that had been machined by the HUST Robotics Club. All the part had to be changed individual to be Aluminum 2018 Alloy. The material properties of this alloy can be found in Figureihsdiuhc . The next step was to fix the mates so that new parts could be added without moving other components of the robot. After this was completed Battery Packs were created at an estimated weight of 100g each and added to the legs of the robot. The final modification was to remove the arm assemblies as the robot we received from the HUST Robotics Club did not have these appendages.

Physical Properties	
Property	
Elastic Modulus	74000 N/mm^2
Poissons Ratio	0.33
Shear Modulus	27000 N/mm^2
Thermal Expansion Coefficient	2.2e-005
Density	0.0028 g/mm^3
Thermal Conductivity	150 W/m K
Specific Heat	1000 J/kg K

Figure 15: Aluminum 2018 Alloy Properties

Using SolidWork's Mass Properties it is possible to calculate the overall mass of the robot. The mass of all the individual appendages and core were first calculated separately. These were then added to find the upper and lower body mass respectively. Adding these totals gave the overall robot mass and using Solid Works we were also to locate the Center of Gravity of the robot. (Appendix H)

Weight Distribution					
Part	Weight (g)	Appendages	Weight (g)	Bodies	Weight (g)
Head	94.03				
Core	274.62				
Right Arm	0				
Left Arm	0	Arms	0		
				Upper Body	368.65
Left Leg	520.2				
Right Leg	520.2	Legs	1040		
				Lower Body	1040.39
				Total	1409.04

Table 1: Robotic Platform Weight Distribution

Weight Ratio	
Upper	Lower
26%	74%

Table 2: Weight Ratio

What we learnt from the mass analysis was the COG was surprisingly positioned. The Center of Gravity (COG) is the point on a body through which all of the weight of the object appears to act. As such it is an important factor when it comes to the balance of the robot. In keeping the robot balanced the COG has to fall within the support polygon formed at the base as detailed in section 4.4.3. According to the Solid Works Model that center of gravity actually falls between the legs of the robot which is initially surprising. However after review the data from the weights of the upper and lower body it can be seen that the robot is dramatically bottom heavy with a majority of its overall mass attributed to the leg appendages. This can be seen in the weight ratio of the robot which is at an unbalance 26% to 74%. This however is advantageous as the lower the COG the greater the area of stable equilibrium.

4.2.3 Mechanical Structure

The HUST Robotics Soccer Club provided the mechanical platform of the humanoid robot. Their design was similar to what we expected and previous discussed in section 2.4.2. The design is shown below is glancing.

1. The distribution of degrees of freedom (DOF).

(1) DOFs of leg: The mechanical construction of leg is the most important part of humanoid; especially our robot's main task is walking. It must have the capability to move left-right, move forward and turn left-right. We design each leg has 6 DOFs, the ankle has 2 DOFs (roll and pitch), the knee has 1 DOF (pitch), the hip has 3 DOFs (roll, pitch and yaw).

(2) DOFs of arm: each arm of robot has 3 DOFs, the shoulder has roll and pitch DOF, the elbow has roll DOF.

(3) DOFs of head: Head has 2 DOFs, roll and pitch, to look up-down, left-right.

Totally, the robot has 20 DOFs.
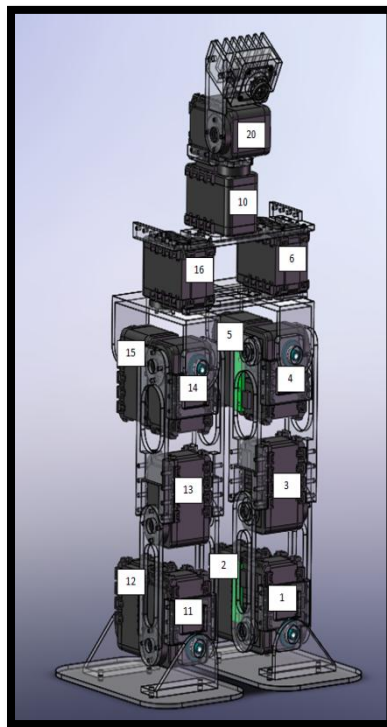
Figure 16: Servo Layout

2. The size of each part.

Because we want to make a robot more like human, the proportion that each part of robot count for its height need to be similar to human's. According to the data, each part of human body count for the height show as below.

Part	Head	Thigh	Calf	Foot	Arm	Breast	Torso	Shoulder	Hip
Proportion	0.157	0.281	0.220	0.146	0.312	0.161	0.382	0.229	0.160

Table 3: Human Body Part Proportionality

The height of robot is about 40cm, the size of each part show as below: (unit/cm)

Part	Head	Thigh	Calf	Foot	Arm	Breast	Torso	Shoulder	Hip
Size	6.3	11	8.8	5.8	12.5	6.5	15.3	9	6.4

Table 4: Robot Body Part Proportionality

4.2.4.1 Benchmark Testing

The benchmark for the robot's performance was initially set by the HUST Robotics Team during the FIRA Competition. At the competition the team achieved great results with 5 championships and 4 second place finishes. In Free Kicking they won the competition with a 4 out of 5 success rate and in path tracking they came in second, walking 42m in 15 minutes.
In order to quantify their results we decided to perform controlled tests of the robot on our own test field. This would give us a means to directly compare our competing programs as the testing environment would be a constant.

4.2.4.2 Ball Kicking

For Ball Kicking we created a test field as can be seen below in Figure 10. This field was different from the FIRA field as the kicking area was a triangle instead of a semi-circle. The ball was placed randomly in the square and the robot was placed in its start position. As soon as the robot was turned on the timer began.
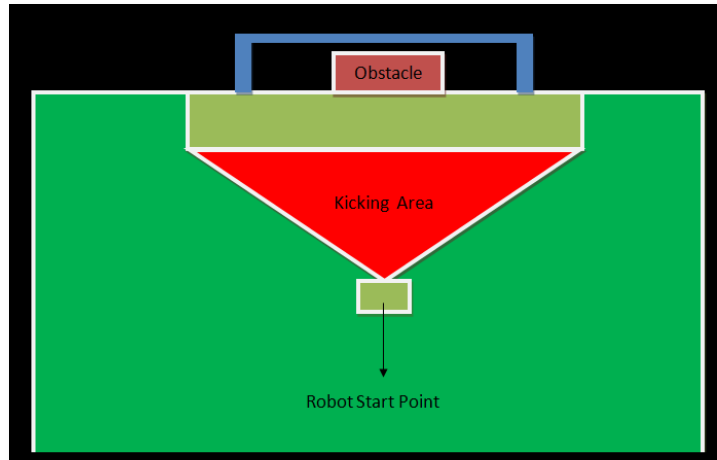
Figure 17: Ball Kicking Test Field

This test was conducted 5 times and robot was able to score four out of 5 attempts on our test field. All attempts were completed under the 90sec limit with an average time of 70sec.

4.2.4.3 Path Tracking

Similarly for Path Tracking we built our own test field which is depicted in Figure iwhidw. This field had a 45 degrees corner, a 90 degrees corner, and a constant radius curve. The variety of curves was deemed necessary to test the robustness of the programs. We placed the robot on the line, switched it on and recorded the amount of time it took to walk a lap. This was repeated 3 times and an average was taken. We also recorded 3 times for the amount of time it took for the Robot to walk a standard distance of 1m.
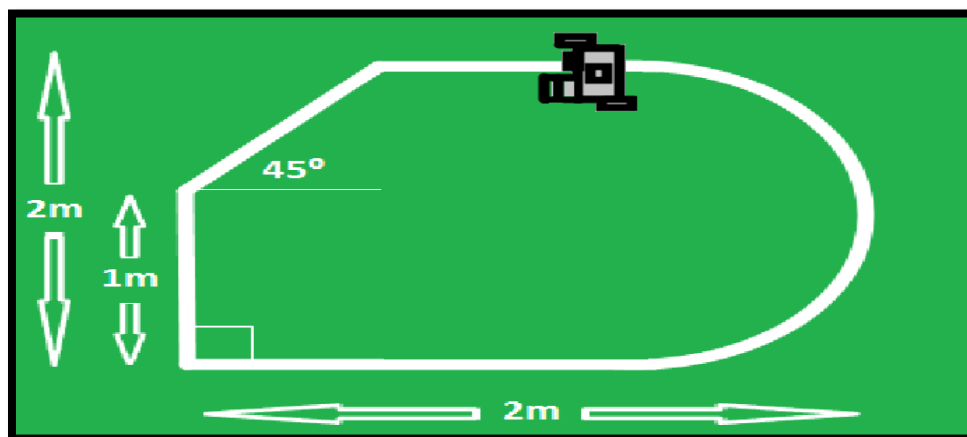
Figure 18: Test Track

For Path Tracking the robot proved capable of walking a lap in 86 seconds and able to walk a meter in 23 seconds. It however could not walk continuously around the track as the robot was unable to navigate the right angled turn. In order to account for this during the recording of lap times the corner of the right angled turn was designated as the start/finish line.

4.3 Force Influence on Mechanical Design

In order to develop a walking strategy, we needed to calculate the outside forces acting on the robot. In this case, we defined a series of tests to prove the force acting on the robot as a whole is negligible. When this was proven, it made our ZMP force calculations more effective and helpful. This analysis will help us design a kinematic model for both ball kicking and path finding robots programs.

To measure the forces on the robot, we needed to utilize a set of equations related to the forces acting on general objects. According to Newton's 2nd law of motion, F = ma or the force is equal to the mass times its acceleration. In order for us to show F ≈ 0, we will prove a ≈ 0 for all real numbers. We know a = v' or the acceleration equals the derivate of velocity with respect to time. By measuring how fast the robot moves over different distances, we can measure the velocity of the robot. This is possible since v = d/t or the velocity is distance covered over time.

First we evaluated how fast the robot covered the distances of 25cm, 50cm, 75cm, and 100cm. By recording 3 different time trials for each distance, we calculated the average for each removing human error. The following chart shows our results:

# Trials	25cm	50cm	75cm	100cm	Distance
1	5	11	14	20	Times(s)
2	7	10	14	20	
3	7	11	15	20	
Averages	6.333333	10.66667	14.33333	20	s
Velocity	3.947368	4.6875	5.232558	5	cm/s

Table 5: Measured Velocities

Since we now have the velocity numbers, we can use the mass of the robot to calculate force. Next we calculated the derivate of velocity. The incremental speed changes are as follows: .740131579, 0.54505814, and -0.23255814. When we use the equations of motion we

get a = v'. We calculated the velocity equations as follows: $v_1 = .740131579x_1$, $v_2 = 0.54505814x_2$ and $v_3 = -0.23255814x_3$. The derivates of v_1, v_2, v_3 are as follows: .740131579, 0.54505814, and -0.23255814. Next we convert the cm/s into m/s to match the units size with the mass of the robot so we divide the values of acceleration by 100 thus: $a_1 = .00740131579$, $a_2 = .0054505814$ and $a_3 = .0023255814$. Since these values are incrementally close to 0, we consider them unsignificant. Using the mass we calculated in solidworks, we plugged the values of a_1,a_2,a_3 into the force equation we see that the total force can be considered as 0 on the complete robotic design, and as such it does not need to be taken into consideration.

4.4.1 ZMP Effect on Gait Planning

Gait planning is the harmonious unification of a robot's servos rotating in time and space to produce a stable bipedal gate. The optimal strategy employs the technique of performing predefined walking poses, which guarantee's an effective and steady walking track for the robot. First, we planned the walking pose and center of gravity according to the structure of the robot. Then we established a viable kinematics model. Finally, we worked out the kinematic equations to match the programmed walking tracks. The movement of the joints had to equal its respective cosine and sine data. Complete gait planning includes two main ideas: walking pose planning and ZMP track planning.

4.4.2 ZMP/Walking Track Planning

Walking pose is each servo position in the relative coordinate system. Each servo helps with the walking process of the robot in space at a specific time. Zero Moment Point is a concept related with dynamics of a humanoid robot and the implementation of legged locomotion. It specifies the point with respect to which dynamic reaction force at the contact of the foot with the ground does not produce any moment. This point is where the total inertia force equals 0. The concept assumes the contact area is planar and has sufficiently high friction to keep the feet from sliding. It can be worked out as follows:

$$X_{zmp} = \frac{\sum_{i=1}^{n} m_i(\ddot{Z}_i + g_z)X_i - \sum_{i=1}^{n} m_i(\ddot{X}_i + g_z)Z_i}{\sum_{i=1}^{n} m_i(\ddot{Z}_i + g_z)} \qquad 4.4\ (1)$$

Equation 6: X Coordinate ZMP

$$Y_{zmp} = \frac{\sum_{i=1}^{n} m_i(\ddot{Z}_i + g_z)Y_i - \sum_{i=1}^{n} m_i(\ddot{Y}_i + g_z)Z_i}{\sum_{i=1}^{n} m_i(\ddot{Z}_i + g_z)}$$

4.4 (2)

Equation 7: Z Coordinate ZMP

The x and y axis are the directions with which the forces are acting on the robot. When the robot's walking speed is low, the inertia forces can be eliminated in the former equations. This fact was proven in section 4.3 for our robot using mathematical analysis. Since Zi represents the value of inertia, this value is 0 in the equation. After simplification of the ZMP track equation, it is used to obtain a center of gravity like so:

$$X = \frac{\sum_{i=1}^{n} m_i g X_i}{\sum_{i=1}^{n} m_i g}$$

4.4 (3)

Equation 8: Simplified X Coordinate ZMP Track

$$Y = \frac{\sum_{i=1}^{n} m_i g Y_i}{\sum_{i=1}^{n} m_i g}$$

4.4 (4)

Equation 9: Simplified Y Coordinate ZMP Track

Using prior research, the observation of human walking and the structure of our robot, we planned the robot's walking pose and motion track as follows:

1. The robots two feet are parallel at all times.
2. The projection of robot's center of gravity is sin wave.
3. The height from ground to the lower back does not change during walking.
4. Robot is always vertical.
5. The track of walking foot has a sin wave in the vertical plane.
6. The Robots soles are parallel with ground.

54

Finally, in order to reduce the impact collision and change the servos' angle smoothly, we set the track of robot's center of gravity and walking foot as a sine wave because the foot's velocity is 0 when it touches the ground.

4.4.3 Motion Control Model

From our research of human walking examples, we divided the walking process into three steps. Three walking steps are the starting gait, walking gait and the stopping gait. The walking flow chart is as follows:
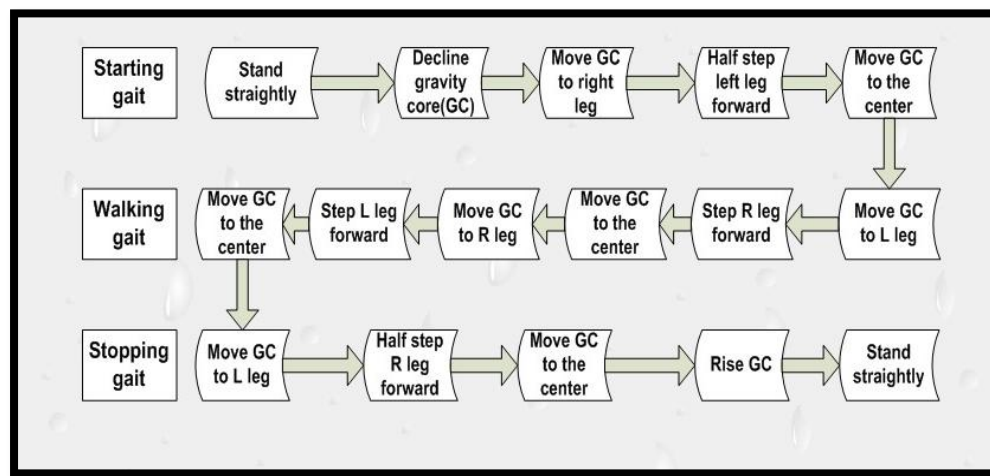
Figure 19:Walking Flow Chart

Assuming the robot will walk upright, we researched the 10 rotational degrees of freedom for its legs. First, we established a homogeneous coordinate system for each servo on the legs as showing in figure 20.
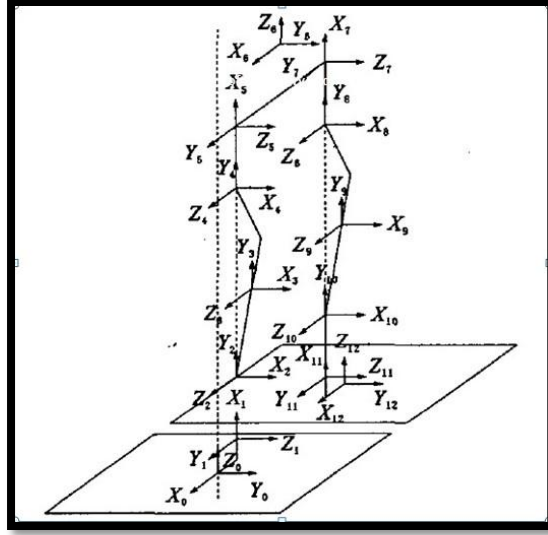
Figure 20: Homogeneous Coordinate System

From the planned gait, we knew the upper body would be vertical at all times. Using this we get the matrix as follows for the servo angles where $^0T_6$ equation is the right side of the robot and $^6T_7$ is the left side of the robot:

$$^0T_6 = {}^0T_1 \, {}^1T_2 \, {}^2T_3 \, {}^3T_4 \, {}^4T_5 \, {}^5T_6$$

$$= \begin{bmatrix} n_{x1} & o_{x1} & a_{x1} & p_{x1} \\ n_{y1} & o_{y1} & a_{y1} & p_{y1} \\ n_{z1} & o_{z1} & a_{z1} & p_{z1} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & p_{x1} \\ 0 & 1 & 0 & p_{y1} \\ 0 & 0 & 1 & p_{z1} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$
{}^{0}T_{1} = \begin{bmatrix} \sin\theta_1 & \cos\theta_1 & 0 & -10.2 \\ 0 & 0 & 1 & 0 \\ \cos\theta_1 & -\sin\theta_1 & 0 & 23 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\qquad
{}^{1}T_{2} = \begin{bmatrix} \sin\theta_2 & \cos\theta_2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \cos\theta_2 & -\sin\theta_2 & 0 & 17.8 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{2}T_{3} = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & 0 \\ \sin\theta_3 & \cos\theta_3 & 0 & 65 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\qquad
{}^{3}T_{4} = \begin{bmatrix} \cos\theta_4 & -\sin\theta_4 & 0 & 0 \\ \sin\theta_4 & \cos\theta_4 & 0 & 86.5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{4}T_{5} = \begin{bmatrix} 0 & 0 & 1 & -17.8 \\ \cos\theta_5 & -\sin\theta_5 & 0 & 0 \\ \sin\theta_5 & \cos\theta_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\qquad
{}^{5}T_{6} = \begin{bmatrix} 0 & 0 & 1 & -17.1 \\ 1 & 0 & 0 & -38.4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{6}T_{7} = {}^{6}T_{7}\,{}^{7}T_{8}\,{}^{8}T_{9}\,{}^{9}T_{10}\,{}^{10}T_{11}\,{}^{11}T_{12}
$$

$$
= \begin{bmatrix} n_{x2} & o_{x2} & a_{x2} & p_{x2} \\ n_{y2} & o_{y2} & a_{y2} & p_{y2} \\ n_{z2} & o_{z2} & a_{z2} & p_{z2} \\ 0 & 0 & 0 & 1 \end{bmatrix}
= \begin{bmatrix} 1 & 0 & 0 & p_{x2} \\ 0 & 1 & 0 & p_{y2} \\ 0 & 0 & 1 & p_{z2} \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{6}T_{7} = \begin{bmatrix} \sin\theta_6 & \cos\theta_6 & 0 & -38.4 \\ 0 & 0 & 1 & 0 \\ \cos\theta_6 & -\sin\theta_6 & 0 & 17.1 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\qquad
{}^{7}T_{8} = \begin{bmatrix} \sin\theta_7 & \cos\theta_7 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \cos\theta_7 & -\sin\theta_7 & 0 & 17.8 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{8}T_{9} = \begin{bmatrix} \cos\theta_8 & -\sin\theta_8 & 0 & 0 \\ \sin\theta_8 & \cos\theta_8 & 0 & -86.5 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{9}T_{10} = \begin{bmatrix} \cos\theta_9 & -\sin\theta_9 & 0 & 0 \\ \sin\theta_9 & \cos\theta_9 & 0 & -65 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
{}^{10}T_{11} = \begin{bmatrix} 0 & 0 & 1 & -17.8 \\ \cos\theta_{10} & -\sin\theta_{10} & 0 & 0 \\ \sin\theta_{10} & \cos\theta_{10} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad {}^{11}T_{12} = \begin{bmatrix} 0 & 0 & 1 & -23 \\ 1 & 0 & 0 & -10.2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Figure 21:Servo Angle Matrices

By working out the equations and simplifying these matrixes with MATLAB, we attained the relationships between the angles as follows:

$$
\theta_2 + \theta_3 + \theta_4 = 0; \theta_1 + \theta_5 = 0
$$

$$
\theta_7 + \theta_8 + \theta_9 = 0; \theta_6 + \theta_{10} = 0
$$

Equation 10: Angle Relationships

4.4.4 Joint Motion

We analyzed the robot's front and lateral movement separately due to little coupling. The robot's center of gravity has to move to the supporting leg during walking. The movement of the center of gravity is performed by the joints lateral motion as follows:
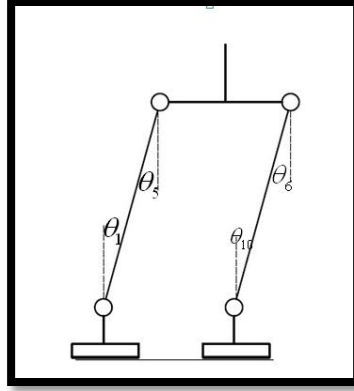
Figure 22: Lateral Joint Movement

When working out the lateral joint movement, it is assumed that the front joint is unable to move at all. The center of gravity for the robot swings around a central line like the sine wave in the figure 16.
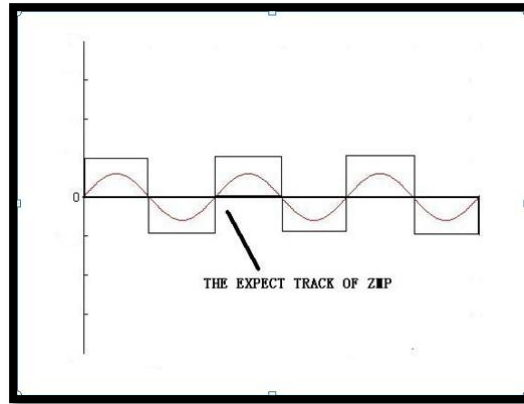
Figure 23: ZMP Expected Track Z Axis

Since the body of the robot stays vertical and its sole stays parallel with the ground, the following equations are true: $\theta_1 + \theta_5 = 0, \quad \theta_6 + \theta_{10} = 0 \quad |\theta_1| = |\theta_5| = |\theta_6| = |\theta_{10}| = \theta$

Based on the former ZMP equations, we can get the robot's center of gravity expression as follows:

$$X = \frac{\sum_{i=1}^{n} m_i g X_i}{\sum_{i=1}^{n} m_i g} = \frac{\sum_{i=1}^{n} m_i l_i \sin \theta}{\sum_{i=1}^{n} m_i} \qquad \text{4.4 (5)}$$

Equation 11: Center of Gravity

The starting gait applies when the time is between 0 and 2. The walking gait applies when the time is between 2 and 4 while the stopping gait is the time between 4 and 6. These are the deviations between the lateral joint movements. The kinematic equations are as follows:

$$\begin{cases} X = 45.5 \sin(\pi * t / 2) & t \in (0, 2s) \\ X = 45.5 \sin(\pi * t) & t \in (2, 4s) \\ X = 45.5 \sin(\pi * t / 2) & t \in (4, 6s) \end{cases} \qquad \text{4.4 (6)}$$

Equation 12: Kinematic

We worked out the angle curves from the equations with MATLAB as follows:
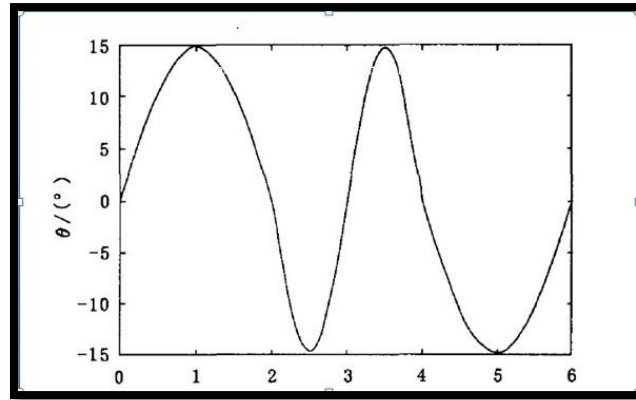
Figure 24:Angle Curves

The front movement is performed by the joints movement as showed in the figure 18.

60

Figure 25: Front Joint Movement

When calculating the front joint movement it is assumed the lateral joints stay the same thus: $\theta_1 = \theta_{10} = \theta_5 = \theta_6 = 0$ By simplifying the matrix of $^0T_6$ and $^6T_{12}$, we get the following expression:

$$
\begin{cases}
p_{x1} = -243/5 \\
p_{y1} = -8.43 - 86.5 * \sin(\theta_2 + \theta_3) - 65 * \sin\theta_2 \\
p_{z1} = 5.9 + 86.5 * \cos(\theta_2 + \theta_2) + 65 * \cos\theta_2
\end{cases}
$$

4.4 (7)

$$
\begin{cases}
p_{x2} = -243/5 \\
p_{y2} = 8.43 + 65 * \sin(\theta_7 + \theta_8) + 86.5 * \sin\theta_7 \\
p_{z2} = -5.9 - 65 * \cos(\theta_7 + \theta_8) - 86.5 * \cos\theta_7
\end{cases}
$$

Equation 13 : Joint Movement

In the starting gait, the robot first lowers the center of gravity by squatting. Next it takes a half step forward in order to perform the transition from standing to walking. In the walking gait, robot walks while its sole is parallel with ground. Considering the motion of two feet is symmetrical, the track on the left foot is the same as the right foot during the next cycle. In the

stopping gait, the movement is just opposite as the starting gait. The joint movements can be

described with the following equations:

$$
\begin{cases}
P_{y1} = -8.43 & P_{y2} = 8.43 & t \in [0,1s) \\
P_{z1} = 157 - 20t & P_{z2} = -157 + 20t & t \in [1,2s)
\end{cases}
$$

$$
\begin{cases}
P_{y1} = -8.43 + 15t & P_{y2} = 8.43 + 15t & t \in [1,2s) \\
P_{z1} = 137 & P_{z2} = -137 & t \in [1,2s)
\end{cases}
$$

$$
\begin{cases}
P_{y1} = -30t + 6.57 & P_{y2} = -30t + 23.43 & t \in [2,3s) \\
P_{z1} = 137 - 10\sin(2\pi t) & P_{z2} = -137 & t \in [2,3s)
\end{cases}
$$

$$
\begin{cases}
P_{y1} = 30t - 23.43 & P_{y2} = 30t - 6.57 & t \in [3,4s) \\
P_{z1} = 137 & P_{z2} = -137 + 10\sin(2\pi t) & t \in [3,4s)
\end{cases}
$$

$$
\begin{cases}
P_{y1} = -15t + 6.57 & P_{y2} = -15t + 23.43 & t \in [4,5s) \\
P_{z1} = 137 & P_{z2} = -137 & t \in [4,5s)
\end{cases}
$$

$$
\begin{cases}
P_{y1} = -8.43 & P_{y2} = 137 + 20t & t \in [5,6s] \\
P_{z1} = 8.43 & P_{z2} = -137 - 20t & t \in [5,6s]
\end{cases}
$$

4.4 (7)

Equation 14: Simplified Joint Movement

With the equations above, we can work out the values of each angle within the whole cycle using MATLAB. (Appendix I)

4.5.1 Ball Kicking Image Processing Algorithm Strategy

The image processing section of the free-kicking robot allows the robot to execute its first function; find the ball that it needs to kick. A process was outlined that would lead to this task being completed:

1. Consider Constraints and Considerations
2. Define Algorithmic Requirements
3. Choose Platform for Development
4. Develop an Algorithm
5. Refine Algorithm until Requirements Satisfied

4.5.2.1 Constraints and Considerations:

4.5.2.2 The Hardware:

In designing the image processing algorithm, a series of considerations had to be made. The hardware platform chosen by our sponsor, the TMS320C6000 Digital Signal Processor (DSP) from Texas instruments, is from the popular TMS320 series, which currently dominates a large portion of the DSP market. While by itself a very capable platform signal processing, image processing is by its very nature a very computationally complex field. Through the entire development process, its limited processing nature was kept in mind.

With enough time, processing the image would prove no problem. However, given the inherit nature of video input and the task of working in real-time with a ball, the allotment for computational time is limited. Given an arrival of video into the DSP at ten frames a second, each image is allowed 0.1sec or 100 milliseconds for processing before the next image arrives from the camera to the processor. Therefore, care must be taken to limit the computational time required within this envelope. Failure to do so would in the least develop into a slowly-reacting robot and at its worst result in a time-overrun that would prevent *any* image from being fully-processed, freezing the robot in its tracks.

Another requirement for the algorithm is for it to reliably detect a ball in an image given a range of disrupting factors, including traditionally notorious fluctuations in lighting. Without reliable detection of the ball, the entire point of the algorithm is voided.

In addition to the requirements of the abstract nature of the algorithm, it also needs to be able to fit within the more structured constraints of the DSP. The algorithm needed to have few enough steps such that it could fit within the limited amount of memory onboard the DSP. The DSP also requires the code be formatted in a specific manner, either in chip-specific assembly or from a member of the C family, (either ANSI C or C++). Therefore, when choosing a development platform, it is important for the code to be able to port to one of these platforms.

4.5.3 Algorithm Requirements:

Given the above set of considerations and constraints, a set of requirements was listed for the completion of a successful image-processing algorithm for free kicking:

- **Input**: Image containing a tennis ball, RGB formatted
- **Output**: X,Y coordinate of a ball
- **Quick Processing**: less than 100 milliseconds to process an image
- **Reliable Processing**: properly identify a ball in an image a large portion of the time
- **Portable**: able to run using one of the DSP coding methods
- **Small**: able to fit on the DSP's limited code storage

4.5.4.1 Choosing the Platform for Development:

With the design requirements in mind, it was time to choose a platform on which the image processing algorithm would be designed. In looking at the current state of image processing research, two main choices became self-evident: MATLAB and C-language based derivatives. These two choices were analyzed for their properties based on the constraints of both the project itself and that of the algorithm.

4.5.4.2 C/C++/VC++

The key disadvantage to using a C based language is the complexity of the programming process. Created at a time when it was more important to write fast code then to write code fast, the C development process is quite long, requiring extensive debugging.

C also possesses a very low-level interface. While this dramatically increases its speed once it is compiled, it also affects how easily it is interfaced with various forms of information. For the case of image processing, it is rather difficult to import image files into the program to be processed. This makes trying large varieties of images in order to test the algorithm's robustness difficult. Interfacing with even more complex forms of media such as video represents monumental hurdles.

This same low-level interface also makes it very difficult to try new types of processes. Each process must be hunted for, imported and integrated until it is compatible with the rest of the code. It is then outputted either to an interface (which must also be coded in the same manner) or to a file (which then must be viewed by a separate program, yet another task).

The chief advantage of coding in a C based language, and why it is still largely used, is that this is a platform natively supported by the DSP. Testing the code on the computer and testing the code on the DSP becomes almost a transparent process. There is no need for conversion between the two. This gives it a large edge over MATLAB, which requires the conversion to native DSP code before it can be run on the DSP.

4.5.4.3 MATLAB

MATLAB, in stark contrast to the C family, was created from the ground up for rapid development and implementation of new algorithms, especially those requiring the processing of large amounts of data, such as image processing. There exists within the language a large quantity of pre-defined image processing functions, and creating compound or new algorithms is a process that is relatively painless compared to the C family.

Inputting data into MATLAB is a fairly straightforward process. Quick, well-documented examples exist for inputting nearly any type of data. Interfacing directly with hardware devices to test real-time processing is made as simple as dragging and dropping functions blocks into place with the use of MATLAB's Simulink package. Outputting and viewing data is just as easy as inputting it. Images and data can be used on-the-fly with single line functions.

In previous times, MATLAB was only suitable for refining the algorithms themselves. In the end, they still had to be hand-coded for use on a DSP. This changed, however, with its Real-Time Embedded Workshop package. With this package, MATLAB now allows integrated support for automatic conversion from MATLAB code to DSP C-code.

Simply put: MATLAB allows a researcher to focus less on the particulars of coding a new algorithm and more on the abstract nature of the process itself. This refocusing from a hard-

line coding process to a more researcher-friendly prototyping and development process has lead in recent years to rapid wide-spread adoption of MATLAB in both research labs and industry. Given this, MATLAB was chosen as the development platform for our Image Processing.

4.5.5.1 Algorithm Developement

In choosing how to isolate and identify a ball in a picture, it seemed wise to go about designing an algorithm that was to define and capitalize on the unique properties of a tennis ball. Three properties were identified; the ball's color was unique and bright, it was large and uniform and the ball was circular from any angle of view. The process is outlined below in order of functions performed on the image.

4.5.5.2 Color Separation:

Each color image is comprised of red, green and blue components. The first function in the algorithm separates an image into its individual colors so that later sections can operate on each separate color in either an independent or combined fashion. The result of the process is shown below:
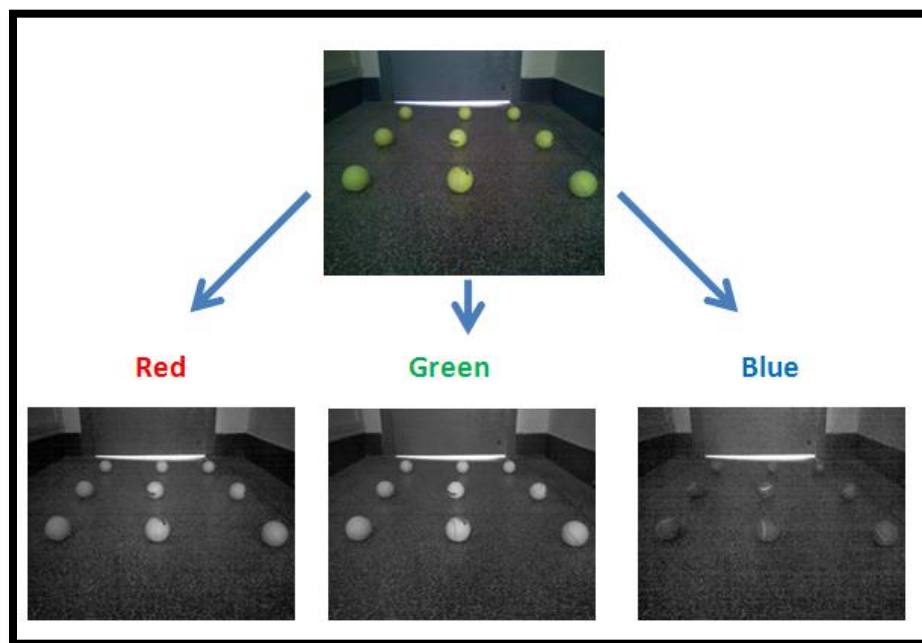
Figure 26:Color Seperation

4.5.5.3 Color Math

These red, green and blue components were then added and subtracted in such a way that the resultant image showed the balls more clearly. This works because in terms of red, green and blue, a tennis ball's color is largely green with a red component. By subtracting out the blue component, what's left is an image in which the ball is highlighted and the background is marginalized. The result of the process is shown below:
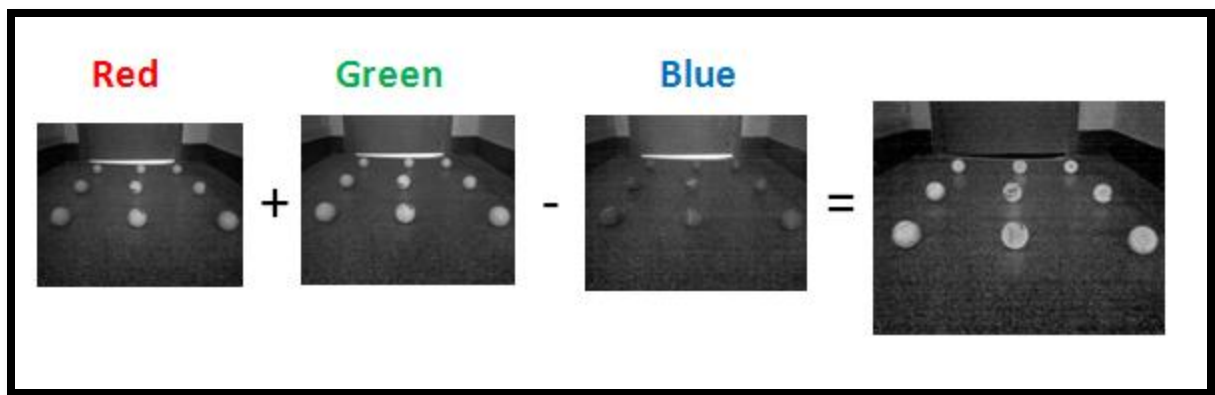
Figure 27: Optimal Colorization

4.5.5.4 Median Filtering

The image resultant of the color math is still quite grainy, as it lacks any definition of larger objects. A Median Filter is passed over the image. This smoothens out the image so that grainy noise is filtered out and the image is more clearly defined as ball/background. The result of the process is shown below:
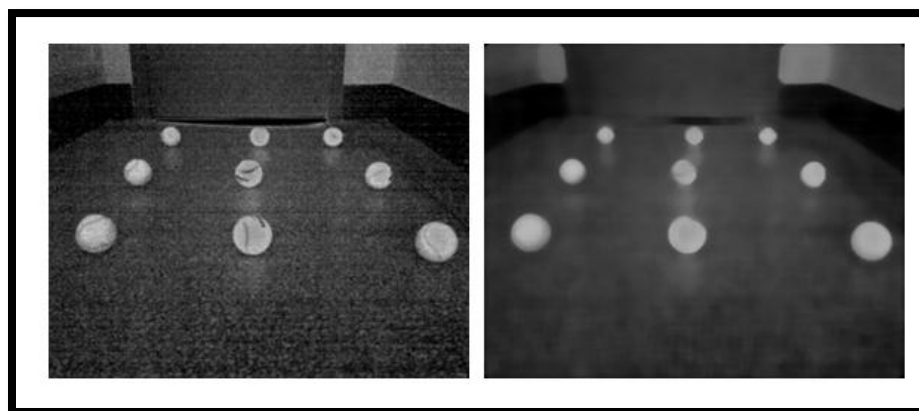
Figure 28: Median Filtering

67

4.5.5.5 Black-white Autothresholding

With more clearly defined objects, it's time to decide whether a pixel belongs to an object or not. This task is performed by black-white auto-thresholding. This process takes a composite average of the brightness of an image and using this decides whether each pixel has an above average brightness, in which case it becomes white, or whether it has below average brightness, in which case it becomes black. The result of the process is shown below:
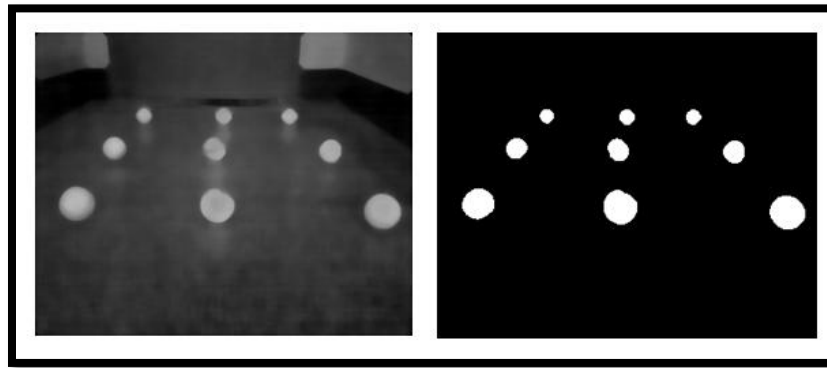
Figure 29: Thresholding

4.5.5.6 Sobel Edge Detection

After going through a binarization process, Sobel Edge Detection defines the edges of objects. This is the algorithmic equivalent of drawing lines around objects. The result of the process is shown below:
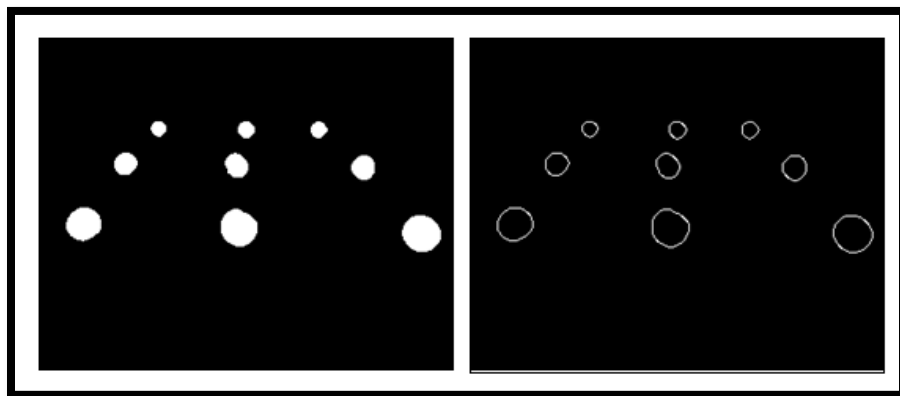
Figure 30: Edge Detection

4.5.5.7 Circular Hough Transform

The task of identifying circles lies with the Circular Hough Transform. This complex algorithm identifies circles in a binary image based on the edges found with the Sobel Edge Detection method. The result of the process is shown below:
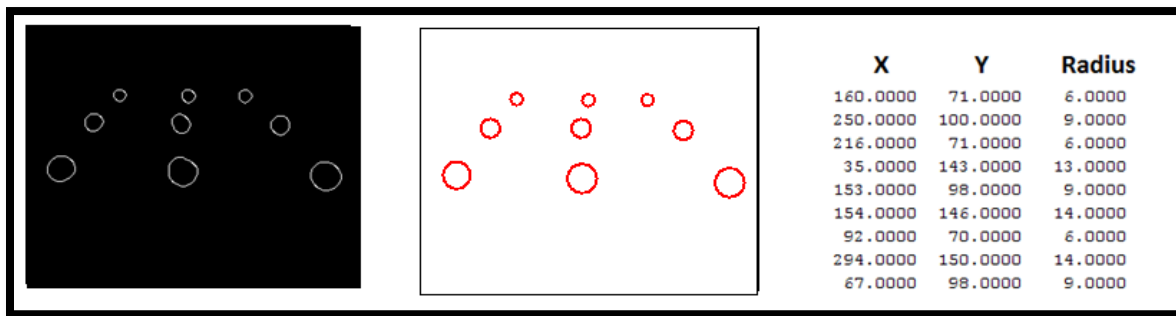
Figure 31: Hough Transform

4.5.6.1 First Algorithm Reviewed

While the algorithm completed its task, in that it was typically able to detect a tennis ball, there was still large room for improvement to be made before it could be considered fully successful. The biggest deficiency in this first algorithm was the amount of time it took to process each frame. Based on computational time in the computer simulation, it could be estimated that it would take up to 1.2 seconds to fully process a frame. This result of 1200 milliseconds lies far from the originally stated requirement of 100 milliseconds. Clearly there was a significant need for improvement in computational speed.

Another factor requiring improvement was the robustness of the algorithm, that is to say, how repeatedly it would successfully find the ball in an image. It was found that the original algorithm was easily confused by factors such as a varying lighting conditions or noisy backgrounds.

Both of these problems could be attributed to the last steps in the algorithm: those that capitalized on the circular aspect of the ball. When the algorithm was analyzed for computational time, it was found that the Sobel Edge Detection on Hough Circular Transform accounted for roughly 70% of the image processing time. Also, by looking at results of the

69

algorithm function-by-function, it was found that most of the failure to isolate the ball occurred because of roughly-circular ball outlines not being detected by the selective Hough Circular Transform. A breakdown of the computational times used by each function is shown below:
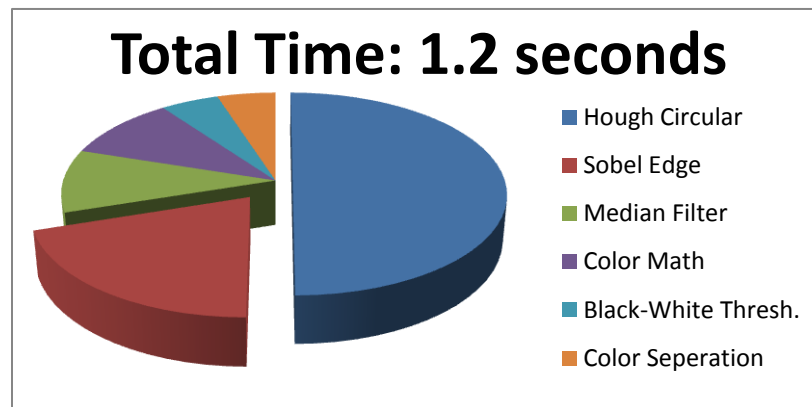
Figure 32:Compuation Time Breakdown

In the design of a new, improved algorithm, these two deficiencies were heavily examined, and new ideas were compiled so as to determine what could be done about them. The following choices were then made:

1.) Resize the Image Before Processing

2.) Improve the Color Math, Incorporate Boolean Math Between the Colors

3.) Median filter after Binarization

4.) Replace Circular Detection Components with Blob Detection and Centroiding

4.5.6.2 Resize the Image before Processing

Image resizing was implemented in order to improve the speed at which the image was processed. Nearly all functions are dependent on the size of the image on which they are operating. While the resizing process itself does take some time, it is one of the faster ones available, and while its contribution to speed improvement varies on the factor of which the image is resized, it easily compensates for its cost. The result of the process is shown below:

70
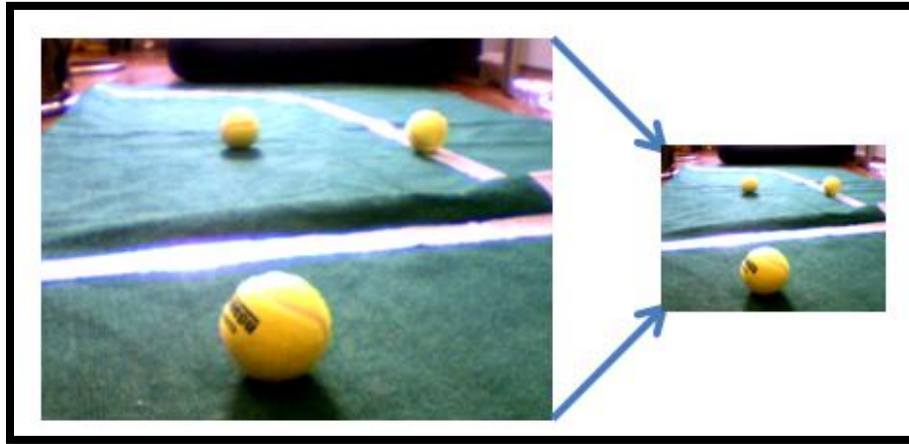
4.5.6.3 Improve the Color Math

The old color math algorithm was not as refined as it could be. While simple and thus fast, it did not do as much to isolate the ball as was possible. With no emphasis now on the ball's circular nature, more focus was now needed on the other two properties, unique, bright color and large, uniform shape, in order to compensate. The exact proportions of the colors of a ball were more closely recorded and examined, leading to new "levels " against which they were multiplied. These levels were   divided into two sections: the image's green base and its red base. A copy of the blue portion of the image was then subtracted from each of these. The result of the process is shown below:
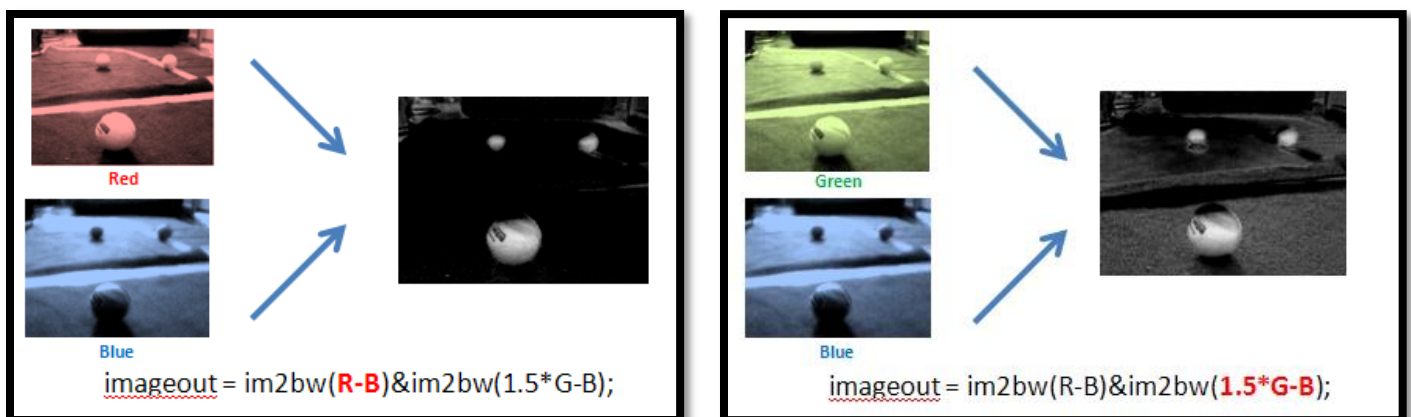
Figure 34: Color Math

71

A large improvement came from choosing to incorporate a Boolean logical component to the color sections. The results of the red and green portions were passed through a binarization function, turning each into a series of black/white 1's and 0's. The results of each binarization were combined using a logical AND component. This led to a binary image in which only those pixels where were white in *both* the green *and* red components remained white. The result of the process is shown below:
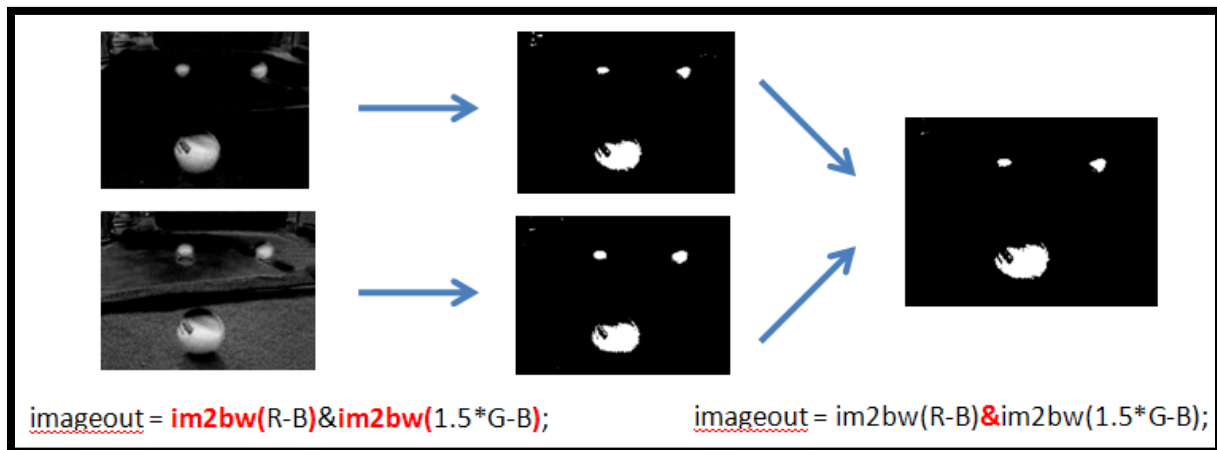
Figure 35: Color Binarization

4.5.6.4 Median filter after Binarization

A marked improvement in speed was derived from applying the median filter after the binarization process induced in the logical portion of the color math. Previously, the median filter was required to work with larger, computational intensive 8-bit integer sets; it was now only required to work with 1-bit black/white sets, leading to a large increase in the function's speed while performing an identical process. The result of the process is shown below:
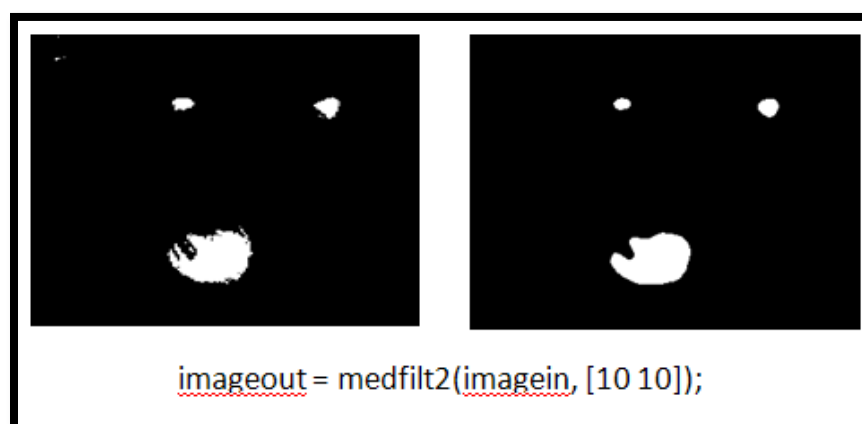
Figure 36: Median Filtration

4.5.6.5 Replace Circular Detection Components with Blob Detection and Centroiding

It was chosen to drop the idea of the ball's circular nature as a property to be exploited. Instead, more emphasis was placed on the first two properties, those of the ball's color uniqueness, and its large, uniform distribution. The hope was that by dropping the Sobel Edge Detection and Hough Circular transforms that caused both of these deficits and replacing them with new, faster, more robust functions, the problems and pitfalls of the first algorithm would be solved.

With the improvement in color math and post-color math median filtering, specific detection of a circle was not required. Upon viewing the post median-filtered image, the tennis balls are clearly evident as massed collections of white pixels. The task at hand became taking these masses, and deriving coordinates from them. This coordinate derivation had previously been covered by the Sobel Edge Detection/Hough Circular Transform function set.   Following the decision to drop this set, a new function was needed to replace this capability while providing improvements in speed and reliability.

With further research, it became evident that Blob Detection was the traditional candidate for this task when specific shape was not required. Blob Detection works by identifying large masses of adjacent binary pixels, analogous to the filtered image representation of a ball. The result of the process is shown below:

ballindex = bwconncomp(imageout,8);
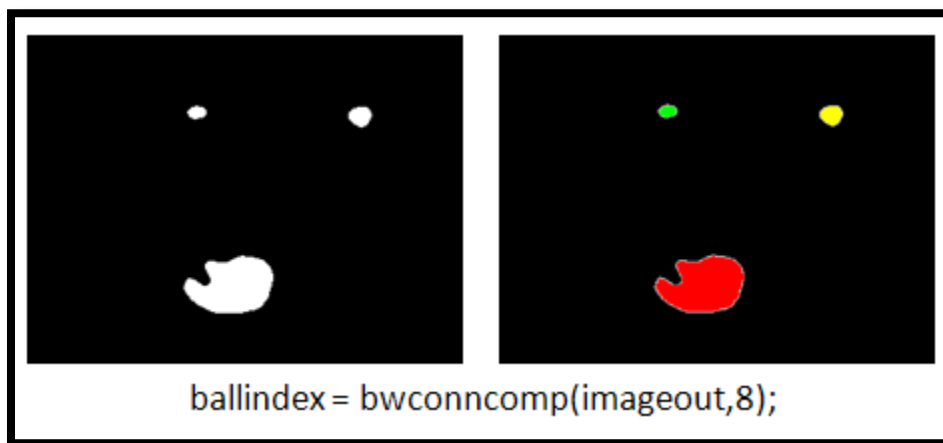
Figure 37: Blob Detection

73

As balls closer to the camera produced a larger blob, the area of the blob became an important means of identifying the distance of the ball from the camera, and was also a value included in the information returned by the algorithm. As a means of making the algorithm even more robust, it was chosen that only the blob with the largest area would have its coordinate passed. The result of the process is shown below:


```
if length(area) > 0,
   areas = [];
   for k = 1:length(area)
      areas = [areas getfield(area, {k,1}, 'Area')];
   end
   [largest index] = max(areas);
```
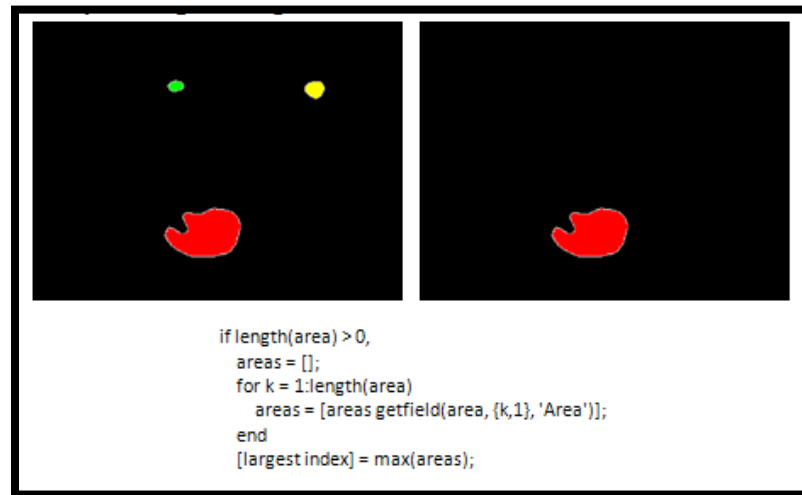
Figure 38: Area Calculation

After the chief blob was identified, a Blob Centroiding was taken of it, identifying the center of the pixel mass found in Bob Detection. The result of this process is a set of coordinates prescribed by the requirements of the algorithm. The result of the process is shown below:

X	Y	Area
0.54	0.23	5.45%

```
centroid = regionprops(ballindex,'Centroid');

XCords = 1-centroid(index).Centroid(1)/320;
YCords = 1-centroid(index).Centroid(1)/240;
Area = largest;
```
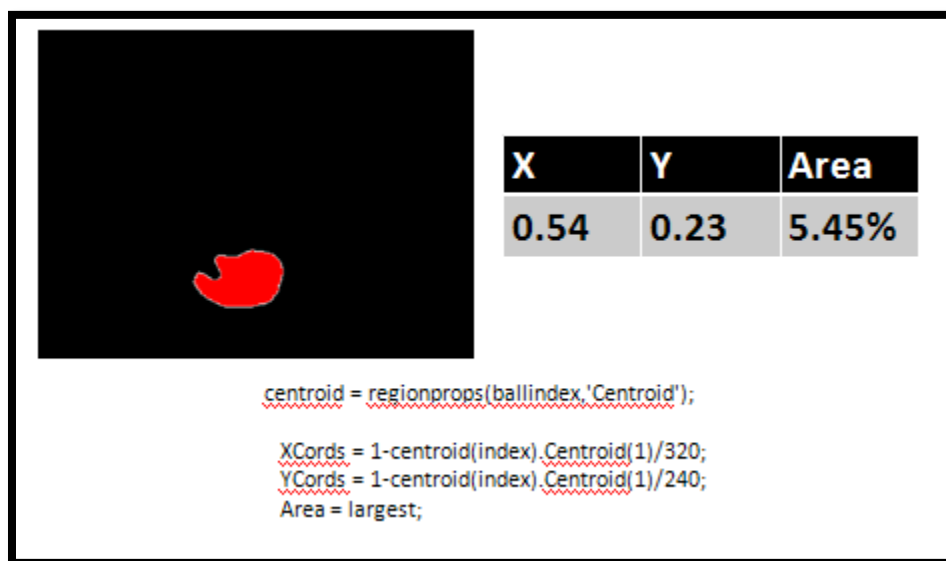
Figure 39: Centroid Location

74

As the image was resized, the pixel position of this center given by the Blob Centroiding was divided by the height and width of the image size, giving a relative and thus universal set of coordinates. An example of this process is shown below:

$$image\ resizing: 740x480 \rightarrow 320x240$$
$$image\ coordinates\ (pixel): (268, 153)$$
$$coordinate\ division: \frac{268, 153}{320, 240}$$
$$image\ size\ independent\ coordinates: (0.806, 0.638)$$

4.5.7 Refinement Reviewed:

With a newly designed algorithm in place, a comparison was made between it and its previous counterpart. Where as in the original algorithm both speed and robustness were deficient, the new algorithm excelled.

A large improvement was made on the speed of the process. Average calculated processing time of an image saw a marked improvement from 1200 milliseconds to 80 milliseconds, a 15x speed improvement. This new speed also falls within the time envelope allowed by the 10 frame per second processing requirement. In this area, the new algorithm met the target specifications. The breakdown of the function times are shown below:
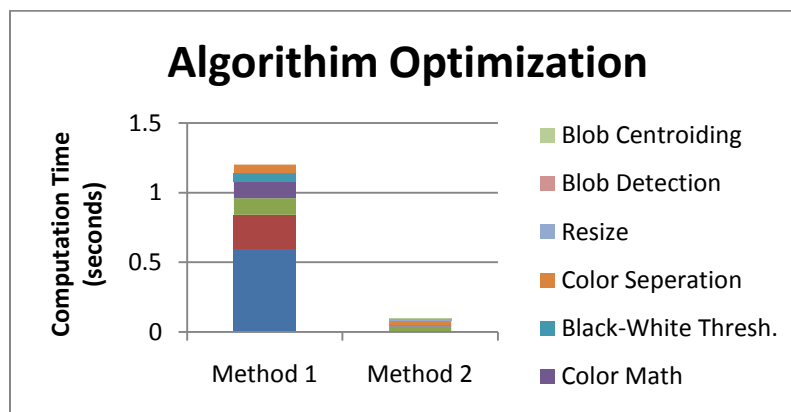
Figure 40: Method Comparison

The new algorithm was also notably more robust than its predecessor. The improvement in color math led to a process which was far more independent of lighting conditions. This, combined with shelving the use of the circular property as a means of identification, led to a marked improvement in the ability of the process to properly identify balls.

4.6.1 Path Tracking Image Processing Algorithm Strategy

For the path tracking robot, an effective image processing strategy was needed for the robot to walk faster than its predecessor. To complete our task and make the robot walk faster, a computationally cheap image processing algorithm strategy was implemented. Our final algorithm process was constructed as follows:

1. Binarization

2. Erosion

3. Linear slope/angle calculation

This process was developed to be the most efficient and computationally cheaper compared to other researched methods. This is proven in our following analysis. To test our image processing algorithms, we used VC++. This platform was more efficient to use than its counterparts: Microsoft supply the Microsoft Foundation Classes (MFC) and Application Programming Interface (API). Interfaces can be built easier in MFC and API but VC++ provided more options for interface creation and allowed serial communication. We decided to use VC++ to implement our image processing strategy. This helped us test our algorithms for the robot functionality.

4.6.2 Interface Construction

For interface construction, we decided to implement VC++ but we had very little knowledge of the programming language. After research and comparing with other interface programs, we decided VC++ was the best platform for our needs. We used books to learn the foundation of VC++ which helped us build the interface quicker. Unfortunately, VC++ includes a large amount of library functions which made are job harder. However, with more libraries to access, we had more options for designing our interface. This did add to the research time

76

necessary to complete our goal. The other platforms MFC and API contained a large variety of classes and functions to choose from so we utilized functions that were similar.   When we succeeded in coding a simple interface, we used this as a stepping stone for our final program. Using our collected knowledge, we coded the image processing interface mentioned in section 3.3.3.

4.6.3 Binarization Technique

After designing the interface to test our image processing strategy, the first method we used was binarization. Binarization is used to separate the image into two parts: white and black. This is accomplished using the values of the pixels with which the image is composed of.

Each image has different bytes to express the value of each pixel. The two kinds we need to worry about are grey-scale and true-color images. As explained in section 2.3.4.2.1, a grey-scale image uses a byte to describe the value of each pixel. The image has two colors including white and black. True color images on the other hand use RGB to form the value of each pixel. This implies each pixel has three bytes. Each pixel can have a value of a number between 1 and 256. Three bytes can generate up to 16777216 kinds of colors, so it can simulate any colors in reality. When we grab a new image, it is a true color image containing three bytes each.

When we coded the algorithm for binarization we set a threshold. This helped average the three bytes so that it makes it easier to use. These bytes are converted into a gray scale image, based on their value and the value of the threshold. When the value was greater than the threshold, it was given a value of 255 while the other pixels became 0. Now the image has only two values. This helps us distinguish the road from the background.

4.6.4 Erosion Technique

After binarization, we used the erosion technique to make the edge of the object smother while eliminating noise. Erosion worked by evaluating each pixel. If the given pixel had a gray scale value of 0, all the pixels surrounding it would be examined. If one of those pixels had a different value than zero, the middle pixel was changed to the gray scale value of 255. If all the values surrounding the pixel are 0, the middle number is converted to 0 as well. This continued until all the pixels in the image were evaluated. This way the black pixels surrounding the line would change to white while all the other pixels outside the image become black.

4.6.5 Slope/Angle Calculation technique

For our last task, we needed to calculate where the line was in the image. We decided to implement a slope/angle calculation method we developed. Our camera would first collect the starting point and final point of the line by scanning the boundary the image. It would then calculate the number of pixels which were valued as the road while retrieving the coordinate of the centre point for each boundary.

After obtaining the number of pixels in each boundary, the starting point is calculated. For example, if the left most side point in the boundary is smaller than the point below, the bottom point is set as the start point. The final point is found the same way except in the opposite direction. When the number of the point higher than the point on the right this point becomes the final one.

To calculate the angle, we need to use the coordinates of the start and final points. Using the points we can calculate the slope of the line. The angle is calculated using the equation angle = arc tan (m) where m is the value of the slope of the line. This can be applied to every possible line the image produces.

4.6.6.1 Algorithm Analysis:

As mentioned earlier this section, the three techniques used to find the line were binarization, erosion and linear angle calculation. This was the least computationally and most effective way to process the image. The first time we developed a strategy, median filtering, edge detection and linear Hough transforms were used. Although this made the image clearer, it was not necessary for our process since it added computational heaviness. Using mathematical reasoning and analysis, we were able to deduce which process was faster and more effective.

For analysis, we first looked into the process of binarization. Then thresholding techniques were compared and the most efficient method was used based on computational cheapness and ease of implementation. For the second part, we mathematically compared the differences between a median and mean filter. We then looked into the effect of erosion on the image. This was compared to our previous filtering techniques and we chose our ideal method for eliminating noise. Finally, we developed a linear angle calculation method. We implemented

our own slope/angle calculation method after testing other algorithms like sample edge detection techniques and linear Hough transform techniques. We chose the slope/angle calculation method since it was computationally cheaper and easier to implement.

4.6.6.2.1 Binarization

We first needed to deal with all the information that was gathered by the camera. We wanted to develop an algorithm that was fast and efficient in finding the line in the image. As mentioned in section 4.6.3, Binarization was the best way to convert the true colors in the image into data we can manage. Since the amounts of colors in an image are so vast, we needed a way to simplify this data if possible. If the pixel data is left unsorted, we greatly increase the computational speed of the algorithm. We knew beforehand that our environment called for only needing two pixel colors in the image. The color of the path in contrast to the background was the algorithm required. Taking this into consideration, we chose to apply the method of binarization first. By applying binarization first, we make the rest of the process much simpler since it only deals with the values of zero and one.

Once we decided to use binarization, we needed to develop a threshold value for which the pixels could be sorted. This way, each pixel would be assigned a value of either zero or one. If the pixel in question had a value above the set threshold, it was assigned the value of 255 which corresponds to the color white. All values under the threshold were assigned a 0 which was black. We had to develop the best way to find our threshold value as to make the image processing algorithm work optimally.

4.6.6.2.2 Thresholding Technique

Choosing the right thresholding technique was essential for our image processing algorithm to function properly. First, we used our prior knowledge and our background research with thresholding techniques written in section 2.3.4.2.1. Then we took our sample pixel values and simulated the results using our image interface we developed. By comparing results, we decided how to obtain our thresholding value and what it would be.

After researching all the methods of thresholding techniques, we decided to use an iterative method mentioned in section 2.3.4.2.5. We first scanned the image and developed an original threshold. By completing this a few more times, we were able to come up with the best

threshold values for the grey-level image. We applied the three different threshold values to our sample image data. Picking the middle values proved to be the best choice, since it did not exclude too many white pixels or includes too many black ones.

The first threshold we used had R = 70, G = 100 and B = 80 as shown below:

Threshold   R = 70   G = 100   B = 80

	R	G	B	AVG
1	100	120	132	117.3333
2	200	240	225	221.6667
3	40	56	89	61.66667
4	74	66	109	83
5	79	82	200	120.3333
6	240	100	50	130
7	140	111	152	134.3333
8	171	162	116	149.6667
9	154	179	101	144.6667

Table 6: Threshold 1

This method takes the RGB values and sorts them according to their values. The yellow highlighted areas are the pixel values that will be assigned the value of 255 while the purple areas will become 0. This kernel then looks as follows:

117.3333	221.6667	61.66667
83	120.3333	130
134.3333	149.6667	144.6667

Table 7: Average 1

With Binarization applied to the kernel we get the values like so:

Kernel		
255	255	0
0	0	0
255	255	255

Table 8: Kernel 1

The second threshold we used had R = 110, G = 130 and B = 110 as shown below:

Threshold   R = 110   G = 130   B = 110

	R	G	B	AVG
1	100	120	132	117.3333
2	200	240	225	221.6667
3	40	56	89	61.66667
4	74	66	109	83
5	79	82	200	120.3333
6	240	100	50	130
7	140	111	152	134.3333
8	171	162	116	149.6667
9	154	179	101	144.6667

Table 9:Threshold 2

This method takes the RGB values and sorts them according to their values. The yellow highlighted areas are the pixel values that will be assigned the value of 255 while the purple areas will become 0. This kernel then looks as follows:

117.3333	221.6667	61.66667
83	120.3333	130
134.3333	149.6667	144.6667

Table 10: Average 2

With Binarization applied to the kernel we get the values like so:

Kernel		
0	255	0
0	0	0
0	255	0

Table 11: Kernel 2

The third threshold we used had R = 200, G = 250 and B = 150 as shown below:

Threshold	R = 200	G = 250	B = 150	
	R	G	B	AVG
1	100	120	132	117.3333
2	200	240	225	221.6667
3	40	56	89	61.66667
4	74	66	109	83
5	79	82	200	120.3333
6	240	100	50	130
7	140	111	152	134.3333
8	171	162	116	149.6667
9	154	179	101	144.6667

Table 12: Threshold 3

This method takes the RGB values and sorts them according to their values. The yellow highlighted areas are the pixel values that will be assigned the value of 255 while the purple areas will become 0. This kernel then looks as follows:

117.3333	221.6667	61.66667
83	120.3333	130
134.3333	149.6667	144.6667

Table 13: Average 3

With Binarization applied to the kernel we get the values like so:

Kernel		
0	0	0
0	0	0
0	0	0

Table 14: Kernel 3

When we compared the results, a few characteristics jumped out at us. With the different thresholding values, they each allowed different values to pass. For the first results, too many pixels were valued as white. This posed a problem because too much data looked like the line. The second threshold gives the right amount of pixels because this doesn't include unnecessary pixels. The last threshold doesn't pick up any values. This number is too high and will ignore the line in the picture altogether. By picking the second procedure, we were able to

make a precise indication of the true value for each pixel. The effectiveness of this technique helped complete our algorithm.

4.6.6.3 Median vs. Mean Filtering

The second algorithm comparison completed was between the mean and median filters. This process is used to help get rid of the excess image noise. We looked into a set of data collected from the image processing interface we developed. This data set contained nine pixel values collected from the image of the path. Each pixel has a red, green and blue value (RGB). For both median and mean filtering, the averages of the pixels RGB values are taken. We conducted two different scenarios where the averages were truncated in one and left alone in the other. The values of the pixels are as shown in figure:

	R	G	B	AVG
1	100	120	132	117.3333
2	200	240	225	221.6667
3	40	56	89	61.66667
4	74	66	109	83
5	79	82	200	120.3333
6	240	100	50	130
7	140	111	152	134.3333
8	171	162	116	149.6667
9	154	179	101	144.6667

Table 15: Pixel Table

The averages and truncated averages were then placed into 3x3 kernels. These kernels as mentioned in section 2.3.4.3.3 are what the mean and median algorithms use to filter their results through convolution methods. Both kernels were calculated like so:

Kernel

117.3333333	221.6666667	61.66666667
83	120.3333333	130
134.3333333	149.6666667	144.6666667

Table 17:Kernel 4

Truncated Kernel

177	222	62
83	120	130
134	150	145

Table 16: Truncated Kernel 1

Using these calculated kernels, we computed the mean and median for both. The values are shown below:

Mean	Median
129.1852	130

Table 19: Mean/Median

Truncated

Mean	Median
135.8889	130
136	

Table 18: Truncated Mean/Median

After calculating the kernels, we replace the middle value with the mean and median values respectively:

Mean Filter

117.3333	221.6667	61.66667
83	129.1852	130
134.3333	149.6667	144.6667

Table 20:Mean Filters

Truncated Mean

117	222	62
83	129	130
134	150	145

Median Filter

117.3333	221.6667	61.66667
83	130	130
134.3333	149.6667	144.6667

Table 21: Median Filters

Truncated Median

117	222	62
83	130	130
134	150	145

This process is repeated over and over again until all pixel values are evaluated this way. The bigger the neighborhood of values used the more skewed the results become. The best results occur when small kernels are analyzed through the filtering process. This is accomplished for every single pixel value. More filtering utilized yields better results, but more computational time is needed.

When we analyzed the data, we found two distinct characteristics. First, the mean filter replaced the value with a pixel value not represented in the set in both cases. This causes problems if this pixel in question is an edge. The filter will compute a different edge and artificially create one that is non-existent. This poses a big problem for our design since we need to be exact as possible when locating the line in the image. If this is done incorrectly, the robot will process the wrong information. The second distinct characteristic is that the median filter is a much more robust average in terms of its calculated value. Other pixels in adjacent neighborhoods will have similar median values. This makes the median filter the much better choice.

4.6.6.4 Erosion vs. Filtering Techniques

After mathematical analysis showed that using a median filter would be most appropriate, we had to compare this technique with morphological erosion. As explained in section 2.3.4.5.3, erosion is another technique used to eliminate excess noise from an image. Erosion is best used when there is a very similar discrepancy between edge pixels and the background. Since erosion algorithms are computationally cheaper than any filter technique we could implement, we wanted to see if applying this technique was applicable.

To apply erosion, we first took the 3x3 kernels we created after using binarization. The kernel has already been converted with its values having either 0 or 255. An example kernel from our image data used is shown in table 11. First the middle pixel value is evaluated as either a 0 or 255. If the pixel value is 255, we check all the surrounding pixels. If any of these pixels are 0, they are all converted to 0 as well. When the middle pixel has a value of 0, if any values of the surrounding pixels are not equal to 0, it is then converted to 255. In our case, since the middle value is 0 with a pixel valuing 255 in quadrant 2 and 8, the middle value is thus converted to 255. This method then evaluates the other surrounding pixels in the same way.

Since we applied binarization to the image pixels first, the erosion algorithm becomes more effective than the filtering algorithms. Erosion is much more computationally cheaper than its rivals because it only has to sort the binary numbers. For the filtering techniques to apply, they convolute kernels with the regular pixel values intact. Even though they take into consideration and combine the RGB values, this is still computationally heavier since this need to be applied to the whole image. Erosion is much more efficient combined with binarization. Although median filtering does a more thorough job since the binarization thresholding might miss a few values, for our purpose, the complexity of the image is so small that it would be unnecessary to use such an algorithm in our process.

4.6.6.5.1 Slope/Angle Calculation Method

The third mathematical algorithm analysis dealt with using the slope/angle calculation method explained in section 4.6.5. Before we determined which method to use, we researched and tested different models and equations. First we looked into using edge detection algorithms. Next, we tried using Linear Hough transformations. Finally we developed a much

simpler slope angle calculation method that would be much easier and much less computationally heavy to implement.

4.6.6.5.2 Edge Detection

The first algorithm we thought would be most effective was the edge detection technique. Since we had already successfully eliminated all the color and noise from the image using previous techniques, we were left with a recognizable path in our image. Using this information, the edge detection technique would be effective in locating the edges of the path.

Edge detection works using the concept of gradients explained in section 2.3.4.4.1. The gradient is used to calculate the biggest change in intensity. Partial derivates in both the y-axis and x-axis directions are used to calculate the value of change. Using the starting point (152, 0) and final point (288, 239) coordinates from the straight line with curve path, we can calculate the gradient in every direction and find where the line turns. Since the coordinates form the line y = m*x + b, we can find the partial derivates as so:

1. The partial derivate with respect to x: First we solve the equation as so => x = (y- b)/m => y/m – b/m => 1/m is the value of the derivate or 136/239 = .5690376569

2. The partial derivate with respect to y: becomes the value of m = 239/136 = 1.757353

From the values of the partial derivates we can get three values: horizontal change, vertical change and 45 degree angle change. The horizontal change is calculated by the partial derivative in the y direction, the vertical by the x direction and the 45 degree by both x and y. In this case we get (0, 1.757353), for vertical (.5690376569, 0) and (.5690376569, 1.757353) for the 45 degree angle. Using these numbers, we know the line changes at the point (.5690376569, 1.757353). We can also calculate the gradient direction angle by using the equation:

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right) \qquad 4.6(1)$$

Equation 15: Gradient Direction Angle

.

Thus the gradient direction angle is found to be 72.05784745. This number can be used to calculate the direction of the line. We also applied this method to a line including a right angle.

When we first used this method we found it to be very effective but very computationally heavy. Since the algorithm needed to take partial derivates at each step for each direction it ate up a great deal of computational cycles. As stated in section 2.3.4.4.1, using the edge detection technique is good if there are a lot of lines in an image. Since we already know what our robot will be looking at and have made it as clear as possible, the excess computational cycles are unnecessary. We turned to the linear Hough transformations to try and get the slope/angle calculation more efficiently.

4.6.6.5.3 Linear Hough transform

The second technique we looked into was the linear Hough transform. This algorithm transforms a line in the image space into a point in Hough space. This makes it easier to trace a line in an image. By converting the points into the Hough space, it makes tracing lines in an image more efficient since it finds all the lines in the given image. If we take the line $y = m*x + b$ it transforms into (m, b) in the Hough space. Alternatively, if in the Hough space the line $b = -x*m + y$ transforms into (x, y).

When we apply this technique, we use two different starting and final point coordinates, we can use system of equations to calculate the values of m and b. When we solve the equations for the coordinates (152, 0) and 288, 239) we get (m, b) = (1.757353, -267.118). For the path that includes a right angle the coordinates are (148, 0) and (319, 176). When we apply the system of equations we get (m, b) = (1.029239766, -152.3274854). With these coordinates, we can calculate the direction angle using the equation angle = arc tan m. Thus we get the direction angle for the first coordinate 60.35849393 and the second path direction angle is 45.82553047.

4.6.6.5.4 Slope/Angle calculation Comparison

After rigorous mathematical analysis, we decided to use our own slope/angle calculation method. Our version does the same exact process as the linear Hough transform accept gets rid of the computational heaviness. Since we already modified the image using binarization and erosion, we knew the image would contain a clear path. The use of either an edge detection or Hough transform algorithm is necessary only if the image in question contains a lot of edges and lines. These particular algorithms are very effective in identifying the

edges and lines in an image. Since this is unnecessary, they both add a great deal of computational heaviness to the strategy. By applying our sound technique, the robot can find and follow the line every time in an extremely fast and efficient way.

For our technique, the slope/angle calculation, we simply used the slope equation for a line. Then we applied the angle equation to the value we calculated for the slope. To compute the slope, we took the change in y over the change in x. We received the same results when we used the linear Hough transform algorithm without unnecessary computational heaviness. This method works extremely well for our case because we already have a line clearly defined in the image. There is no need to check if it is able to transform into the Hough space or has any edges using the edge detection algorithm. The results we exactly the same and our robot was able to follow the path efficiently and effectively.

4.7 Ball kicking gait strategy implemented

For the ball kicking robot to effectively move and kick the soccer ball into the goal every time, we needed to test and compare our different strategies. As mentioned in section 3.4.1, we had a few options to consider. After careful testing and debugging each gait strategy, we decided to move forward with strategy one as mentioned in section 3.4.2. We chose this approach based on a few factors.

When we compared both plans, a few characteristics jumped out. For strategy one, the robot is able to calculate the exact distance between itself and the ball. Using this information, it will move according to how far away it actually is. If the camera is accurate and the movement fast, the robot can move effectively towards the ball. This also decreases the time the robot needs to readjust and test between each image scan of the field.

Regarding strategy two as described in section 3.4.3, the robot scans different areas of the field. It then moves towards the area with the best view. It continues this routine until it is right above the ball. After a certain amount of times, the robot will scan the area it saw the ball in more often. The precision of the camera is not always accurate and the robot can miss the ball quite easily.

We chose strategy one for a number of reasons. First and for most, the robot will know exactly where the ball is at all times so it reduces the error. Although it is less flexible, using

strategy one leaves nothing to chance. We were able to get the robot to score every time because it calculated where the ball. This was not left to chance which made our gait program more effective. Another positive is the fact that the time it took to readjust the steps the robot would take toward the ball was significantly smaller. Too much time was needed to readjust the robot while using strategy two.
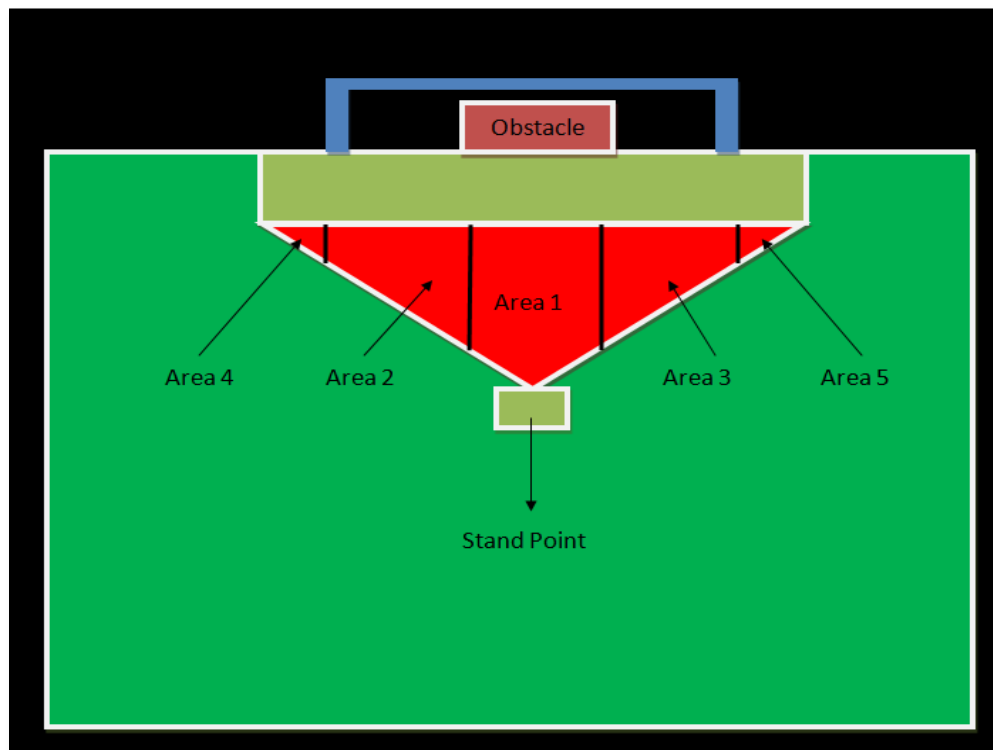

Figure 41: Kicking Area Division

4.8.1 Path Tracking Gait Strategy Implemented

After binarization and erosion have been finished, the robot needs to extract information efficiently from the processed image in order to make a decision on where to walk. Two strategies have been researched; the area method and the angle method as discussed in Section 3.5.1.

Area Method	Angle Method
Pros	Pros
Walks Generally on the Path	Simple to Calculate the Angle
Easy to Find Way Back to Line	Fast Computation Speed
Easier to Implement Directional Commands	More Accurate Information for Walking Direction
Cons	Cons
Scans All the Pixels in the 6 Areas	Possible to Deviate More from the Line
Takes a Long Time Complete Calculations	Easier to Lose Path Information
Used by HUST Robotics Club	Possible to Get Wrong Point Due to Noise

Table 22: Area/ Angle Method Comparison

Upon weighing up the positives and negatives of the two methods as shown in Table 22 we decided to implement the angle calculation method over the area calculation method. Both methods could have been implemented but there was one factor that made the angle method more appealing. This was the fact that the Wuhan Robot Team was currently using the area method with the robot. While we could have worked with their strategy and optimized it to be more precise and able to process right angled turns the thought of developing our own code from scratch appeared to be more challenging.
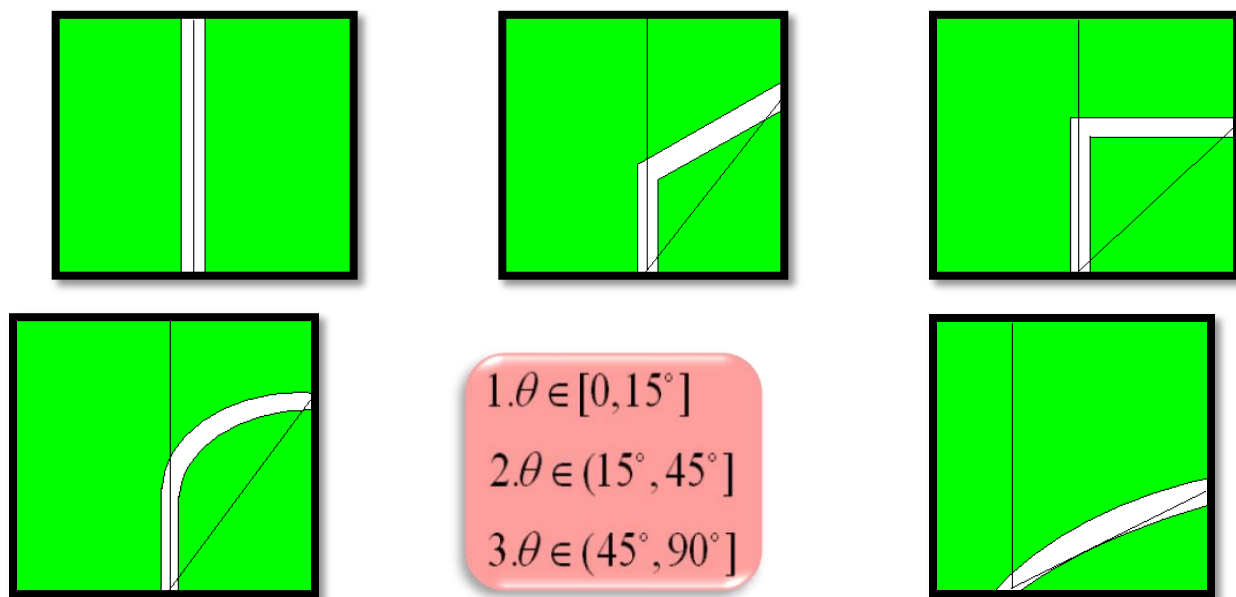

$$1. \theta \in [0, 15°]$$
$$2. \theta \in (15°, 45°]$$
$$3. \theta \in (45°, 90°]$$

Figure 42: Angle Detection

After debugging the robot, we found that angle method is superior to the area method. The robot can walk exactly on the road with area method, but it wastes a lot of time and makes it difficult for the robot to navigate the right angle. While the robot will deviate a little from the road with the angle method, the robot has better forecasting ability and as such is able to make quick reactions towards road information Secondly, by dividing the angle into several intervals, robot can rotate to different angles in order to adapt to a variety of road information while the area method will not implement this function.

4.8.2 Gait Strategy

In order to have a stable gait it is imperative that the COG of the robot be kept as low as possible. As seen in section 4.2.2, the COG is already relatively low but a lower position is always advantageous. As such of main modification to the gait will be to modify the walking stance so that the knees of the robot are more bent. This means that instead of relying more on shuffling forward the robot will be able to walk with a gait that resembles an actual step. Turning will be accomplished by angling of the hips. With greater a greater angle resulting in faster turning rate.

4.8.3 Implementation

The walking of the robot was broken into 4 different gait categories; begin, straight ahead, turn left and turn right. These different categories are then broken down into 3 individual phases; weight transfer, lift off, forward swing. These phases would all be implemented by the servos of the robot working in cohesion to attain the desired position in a synchronized manner. An interface for gait planning was created in VC++ in order to directly control all of the servos in a time saving manner. This interface was used to input and fine tune the servo position for each phase of a step.

4.9 Ball Kicking Complete Functionality

The robot can recognize the ball placed in the ball kicking area of the field, strafe, move forward, and, if necessary, turn to an acute angle in order to align with the ball. It then can kick

the ball into the gate with a 20cm barrier placed in the center of goalkeeper area. The image processing algorithm strategy for ball recognition is fast and accurate. The motion of the robot is smooth and accurate while the ball kicking action is precise.

4.10 Path Tracking Complete Functionality

The robot can recognize and distinguish the line from the field. It can then accurately analyze the line finding the start point, the end point and the angle between these points. The robot can then plan a path and smoothly follow the line for a distance of up to 42m. It can navigate a variety of turns; obtuse, acute, right-angled and constant radius while maintaining a line deviation of less than 40cm.

4.11 Conclusion

By completing and analyzing our findings, we completed our project goal. Using our background research and following our methods helped us to organize and finish two robotic programs that succeeded in surpassing its older counterparts.

First, we conducted a mechanical design analysis of the previous robot. Then we calculated the impact of force on the robot which helped us calculate the ZMP effect on gait planning. Next we created an image processing strategy for the ball kicking and path finding robots. Each algorithm was researched and effectiveness was proven using mathematical reasoning.

After finishing these steps, we had to integrate gait strategies for both ball kicking and path finding robots. Using the tools available, we choose an effective ball kicking strategy for a field with and without a barrier. For the path finding gait strategy, we picked a method that was very successful. Finally, by putting everything together, we programmed both the ball kicking and path finding robots. We described how each robot completes its tasks and functions. Based on our findings we formed our recommendations and conclusions.

Chapter 5: Recommendations and Conclusions

5.1 Conclusions

In collaboration with students from Huazhong University of Science and Technology, we were tasked with improving the performance of their school's entry into the Federation of International Robot-soccer Association's (FIRA's) Human Robot World Cup Soccer Tournament (HuroCup). *Our goal was to develop a humanoid robot that could display effective soccer skills and techniques.* The intention was to gain further knowledge and experience in robotics programming. Given that the robot platform was already built and designed by the HUST Robot Soccer Club, the technical sponsor challenged our MQP team to improve the performance of the robot in two events: marathon running and penalty kicking.

After two months of preparation and two months of collaboration in Wuhan, China, the HUST/WPI team was able to successfully surpass the performance markers previously established by the Robot Soccer Club in both events. While the penalty kicking performance showed tangible improvement from a four of out five to a five out of five success rate, the marathon running improvement did not officially reach the 42 m/10 min. goal required by the technical sponsor. The MQP team was able to get the robot to run 42 m in 12 minutes on the smaller test course that had tighter turns, which increased the difficulty of maneuvering the robot for the distance specified. We estimate however that if the test had been conducted on the official FIRA track that the team could have beaten the mark of 42m in 10 minutes set by the advisor.

All along the project the MQP was faced with many challenges. Communication of ideas and strategies was an issue due to the language barrier. Debugging was an intensive process, requiring repeated testing and verification of the functionality of the program code being uploaded onto the robot. There were also repeated setbacks due to the unavailability of the hardware in the earlier weeks, as well as malfunctioning or broken hardware that had to be identified and replaced. Despite the challenges, the robot now displays a marked improvement over its previous performance, and thus validates the combined efforts of the MQP team.

5.2 Recommendations

Although the team was able to improve the robot's performance in the required events, there are still unexplored avenues left in terms of further boosting the effectiveness of the robot. Because of a limited time frame, the team focused solely on improving the software side of the robot, retooling the control and vision input algorithms. For a more thorough and holistic approach to improvement, other performance critical aspects of the robot cannot be ignored.

The first major step would be identifying areas of improvement through a mechanical analysis of the robot, and then proceeding forward with ideas for further iterations of the robot's physical design. The team had no choice in the matter of using another team's robot platform, but perhaps future projects may arise where students have the opportunity to develop an entire new platform based on previous projects.

Another step would be to make each challenge more realistic in terms of difficulty. In terms of the actual events, the penalty kicking actual involves a head-to-head showdown between the robot striker and the opposing robot goalie. One step further would be to develop the ball kicking algorithm to, in addition, recognize the position of an obstacle/goalie and shoot around it. The first challenge would be to be able to consistently best a randomly placed static goalie, but it would naturally move up to an intelligent and dynamic goalie.

Another improvement would be to have both legs have the ability to kick the ball. This would cut down the time for position the robot significantly. Further improvement could then be made by having the robot calculate and adjust its stride on the fly. This would mean that the robot would not have to pause repeatedly to reassess its position. But instead could approach the ball in a seamless and more realistic manner.

Finally, one of the end programs could be fusing the marathon and penalty kicking programs into a singular robot behavior. With a successful combination of the two programs, the robot would be that much closer towards more realistic play, and would no longer be limited by having to switch programs from isolated events and challenges. The robot would then be able to fluidly follow a line to within proximity of a ball, then kick the ball into a goal, all in one motion.

Authorship:

Bibliography

Aler, R., Valls, J. M., Camacho, D., & Lopez, A. (2009). Programming Robosoccer agents by modeling human behavior. *Science Direct* .

Bekey, G., Ambrose, R., Kumar, V., Laverly, D., Sanderson, A., Brian, W., et al. (2008). *Robotics State of the Art and Future Challenges.* London: Imperical Press.

Camacho, D., Fernandez, F., & Rodelgo, M. A. (2005). Roboskeleton: An architecture for coordinating robot soccer agents. *Science Direct* .

Carsten Steger, M. U. (2008). *Machine Vision Algorithms and Applications.* PRC: Wiley-VCH.

Castiotta, J., & Heslop, M. (n.d.). Humanoid Robot Developement 2007 MQP Report.

Chellappa, R. (n.d.). Edge Detection. Maryland, USA.

Cromwell, B. (2009). *Histogram Equalization.* Retrieved July 5, 2009, from Contrast Enhancement through Localized Histogram Equalization: http://www.cromwell-intl.com/3d/histogram/

Czarnetzki, S., Kerner, S., & Urbann, O. (2009). Observer-based Dynamic Walking Control for Biped Robots. *Science Direct* .

Gutkind, L. (2006). *Almost Human, Making Robots Think.* New York: W.W. Norton & Company Inc.

Haralick, D. N. (2000). Recursive Binary Dilation and Erosion Using Digital. *IEEE TRANSACTIONS ON IMAGE PROCESSING , 9* (5), 749-759.

Harmati, I., & Skyrzypczyk, K. (2008). Robot team coordination for target tracking using fuzzy logic controller in game theoretic framework. *Science Direct* .

Henden, P. C. (2004, November 20). *Exercise in Computer Vision.* Retrieved June 30, 2009, from http://www.pvv.org/~perchrh/papers/datasyn/paper2/report.pdf

Hinz, S. (2005). FAST AND SUBPIXEL PRECISE BLOB DETECTION AND ATTRIBUTION. *Remote Sensing Technology Technical University Munich* , 1-4.

J. S. WESZKA, R. N. (1974, December). *A Threshold Selection Technique.* Retrieved June 20, 2009, from http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1672451&isnumber=35080?tag=1

J. Sauvola, M. P. (1999, January 21). Adaptive document image binarization. *Pattern Recognition* , 1-12.

Jong H. Park, Y. K. (n.d.). ZMP of Biped Robot. *ZMP Trajectory Generation for Reduced Trunk Motions of Biped Robots* , 1-6.

McCloy, D. &. (1986). *Robotics an Introduction.* New York: Halsted Press.

Raibert, M. (1986). *Legged Robots that Balance.* Massachusetts: The MIT Press.

Robert Fisher, S. P. (2000). Retrieved July 15, 2009, from Digital Filtering: http://homepages.inf.ed.ac.uk/rbf/HIPR2/filtops.htm

Robots, Society of. (2009). *Programming- Computer Vison Tutorial.* Retrieved July 1st, 2009, from

Society of Robots: http://www.societyofrobots.com

Rong, Y. (n.d.). MQP in China: Extension of WPI Practice on Project Based Engineering Education.

Stone, P., & Balch, T. &. (2001). *RoboCup 2000: Robot Soccer World Cup IV. .* Germany: Springer Verlag.

Tosunoglu, A. M. (n.d.). IMAGE PROCESSING TECHNIQUES FOR MACHINE VISION. Miami, Florida, USA.

Retrieved July 10, 2009, from http://www.eng.fiu.edu/mme/robotics/elib/am_st_fiu_ppr_2000.pdf