Single Player Foosball Table with an Autonomous Opponent

Design and Final Report By:

- Michael Aeberhard
- Shane Connelly
- Evan Tarr
- Nardis Walker

Submitted: December 10th, 2007 **Instructor:** Dr. James Hamblen
Fall 2007 ECE 4007/L01

Table of Contents

1. Executive Summary

An automated foosball table offers a challenging player versus computer match-up in a game of table-football (foosball). In addition to being challenging for the expert foosball player, it allows one to play without the need of finding a formidable human opponent. Furthermore, the idea of "man versus machine" makes an automated foosball table an interesting and fun challenge to play against. The autonomous foosball table (AFT) falls within the same category as typical arcade games found in entertainment centers, such as pinball, air hockey, and arcade videogames. Currently there is no such automated foosball arcade machine in mass production; therefore, there is an excellent opportunity to market such a product. By offering an AFT at a competitive arcade machine price, the potential for profitability is great. This document will examine the underlying technology used in building an AFT and prove that a challenging computer-controlled opponent can be developed. The completed prototype plays fairly well against an average foosball player and functions reliably, showing the potential for a finalized table. The issues encountered which hinder the AFT were budgetary. With faster motors and larger gears, the table would function at an advanced level with ease. After the successful demonstration of the prototype, further development can be made to improve presentability and gameplay mechanics for a marketable and manufacturable arcade machine.

2. Introduction

A prototype foosball table with a robotic opponent was designed and constructed by project engineers. An autonomous foosball table will effectively bring a fast-paced multi-player game to lone players or teams of players.

2.1 Objective

The ultimate goal was to create an automated foosball opponent that can compete with a human player. It is challenging, but not overly so, thereby encouraging players of all skill levels. With this level of playability attained, it could be sold to a number of commercial locations as well as occasional individuals. Bars, arcades, theme parks, and locations with small fun centers like some airports and movie theaters would be interested in the product at a competitive price.

2.2 Motivation

There currently is only one AFT on the market, however, it is very expensive and has low functionality. The goal was to develop an AFT that can be offered at a competitive arcade machine price, and has more functionality than what is currently on the market. With a competitive price, the AFT will be easily marketed to the customer and offer the manufacturer a big profit.

2.3 Background

Robotic foosball tables have been created in recent years by students at a few universities, but as of yet there have been no commercial applications. The Foosbot, a project of students at Rice University, used a series of infrared LEDs and phototransistors to track the ball as it traversed the table and potentiometers on the gears to track player position [1]. It is said to be undefeated, but has yet to play against experienced foosball players. To achieve better tracking, the system will utilize a camera above the table for tracking the ball and lateral player movement, and servos for

accurate player rotation. KiRo, a project out of the University of Freiburg in Germany, uses a top mounted camera as in this design and has defeated 85% of its opponents, including expert players [2]. It, however, uses much more expensive equipment than what the proposed prototype will use, and is thus not commercially viable.

The underlying technology of the table lies largely within three fields, each of which has vast amounts of research available. Accurate tracking through computer vision is integral to the success of the device, and several resources are available at Georgia Tech. The Computational Perception Laboratory, for example, currently has a project underway to track eyes as people approach a camera [3]. This project is a rather complex task compared to following a colored ball, but the underlying research is vast and readily available. PIC control of the servos used to control the players is another important aspect of the device, and another area with a great amount of available resources. The technology has been around since the 70's and has been used in a wide variety of projects due to its low cost and extensive collection of application notes [4]. The final field in use will be simple artificial intelligence used to decide the best strategies for both offense and defense in a given scenario. Artificial intelligence has been applied in varying degrees to machines for decades to accomplish goals as simple as case based reasoning in a recommendation system, or as complex as defeating a world champion chess player [5]. All of these combined can form a challenging and playable autonomous foosball robot.

3. Project Description and Goals

The main project goal was to complete a working prototype for an AFT, where a human player faces a robotic opponent. From the human perspective of the game, the foosball table is very similar to a regular table. The player(s) on the human side are controlled via a series of four handles that can be moved in and out and rotated to move the players linearly across the playing field and to kick the ball towards the opponent's goal. The autonomous side consists of:

- Eight servo motors used to manipulate the handles of the foosball table
- A microcontroller to activate the servo motors and communicate with the computer
- An over-head mounted webcam to track the ball and players
- A computer to process the webcam images, implement artificial intelligence, and communicate with the microcontroller

The initial prototype goals were to make a simple functioning automated player to play against, one that is at least able to defend the goal and make an effort to kick the ball toward the other end of the table. Once a simple level of gameplay was established, the next goal was to improve the artificial intelligence (AI) of the automated player in order to increase the challenge of the game for the human player. The project prototype proves that an automated foosball table is feasible and cost-effective, and that a simple level of AI can be achieved for a somewhat challenging game to the novice foosball player. At a target price of $5,000 the table will also be affordable to various establishments such as bars and arcades.

Budget constraints for the prototype slowed the project some and kept its functionality to a minimum. Proper motors to move the players at a competitive speed were found to be very expensive, so lower-end servos had to be used. The prototype was still able to exhibit the desired gameplay, but at an undesirable speed. Gearing the motors up to achieve faster lateral movement was chosen, but even the gears proved to be much more expensive than allowed by the limited budget available.

Budget constraints for the prototype slowed the project some and kept its functionality to a minimum. Proper motors to move the players at a competitive speed were found to be very expensive, so lower-end servos had to be used. The prototype was still able to exhibit the desired gameplay, but at an undesirable speed. Gearing the motors up to achieve faster lateral movement was chosen, but even the gears proved to be much more expensive than allowed by the limited budget available.

4. Technical Specifications

Table 1 shows the projected technical specifications versus the actual tested specifications of the implemented automated foosball table.

Table 1. Automated foosball table technical specifications.

Item	Specification Goals	Demonstrated Specifications
Camera frame rate	min. 60 frames per second	30 frames per second
Camera resolution	min. 30 pixels per sq. in.	39.5 pixels per sq. in.
Localized ball tracking success rate	80% of frames minimum	93% of frames
Kick rate (ball velocity)	up to 10 feet per second	1.5 feet per second
Lateral player speed	up to 2.5 feet per second	0.77 feet per second
Lateral player position resolution	at least 1 cm	0.228 mm
Move and kick success rate	75% of attempts minimum	33% of attempts
Unopposed goal rate	50% of attempts minimum	10% of attempts
Goalie blocking success rate	90% of attempts minimum	72% of attempts
Reaction time from webcam	200 ms maximum	100ms
Power requirements	TBD	4.3A, 9.5V
Weight	TBD	69 lbs
Table dimensions	TBD	5’ x 4’ x 8’ (W x L x H)

The first issue that we ran into involved the webcam. The Phillips SPC900NC was chosen due to its listed framerate of 90 fps and resolution of 1280x1024 pixels. Upon using it, it was discovered that the camera used only USB 1.1 which supplies a relatively low data rate. This only allowed the camera to capture 30 fps at 320x240 pixels. The drivers supplied by Phillips read in this data and triple each frame and quadruple each pixel in order to achieve the listed values. Since this internally modified data is of no use to the vision software written, the framerate and fps used had to be lowered. The image size was still sufficient to supersede our required resolution, and the software was able to track the ball well over our goal of 80% of frames.

The kicking and movement speeds achieved were well under our goals due to budget constraints. The kicking speed can be easily increased with the purchase of faster servos. In order to increase the lateral movement speed, we chose to gear up the AX-12 servos. What was not anticipated was that gears of the size we needed must be custom made, and therefore have very high setup costs. Both of these issues could be remedied in a final product with a higher budget. Without the appropriate speed, the players could only get to their desired location if the ball was moving very slowly, so many of the other specifications failed as well.

5. Design Approach and Details

5.1 Design Overview

The AFT consists of six main components:

1. A standard foosball table with a playing field of 24 by 48 inches
2. Four high torque AX-12 UART controlled servos for lateral movement [6]
3. Four high speed, HS-81 PWM controlled servos for the kicking motion [7]
4. A servo control board, utilizing a PIC microcontroller for instruction processing
5. A webcam to visually acquire images of the foosball table as it is played
6. A PC to process image data, track the ball and opponent players, make intelligent decisions of how to play, and send servo control commands to the servo control board

Figure 1 shows a basic system overview of the AFT and all of its components and interconnections.

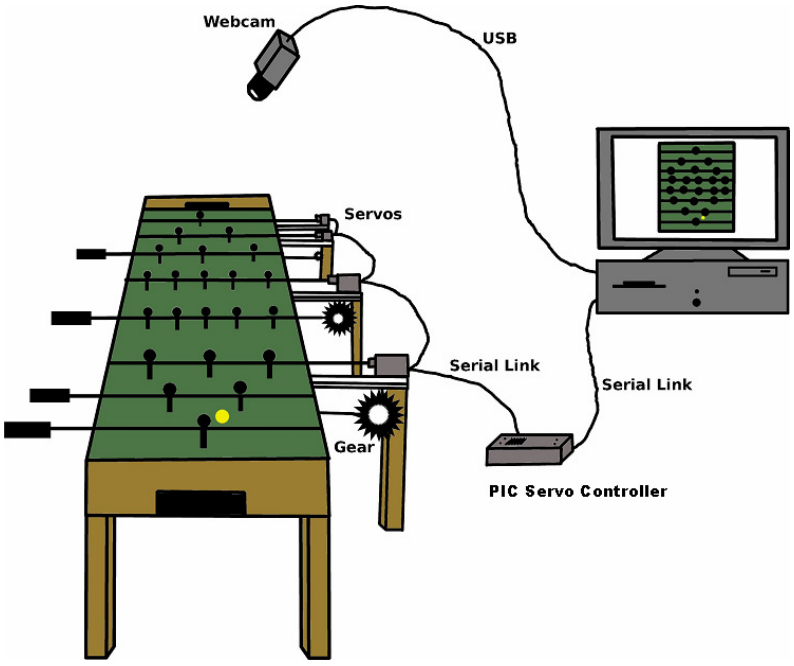


Figure 1. Automated foosball table system overview.

5.2 Image Processing Design

The image detection is done via a Philips SPC-900NC, which is a standard USB 1.1 webcam. The webcam supports a maximum resolution of 640x480 pixels and a maximum frame rate of 30 uncompressed frames per second. Due to bandwidth problems imposed by USB1.1, the maximum resolution and frame rate may not be achieved at the same time, as seen below in *Table 2*.

Table 2. Supported frame rates and resolutions of the SPC-900NC Source: Linux PWC Wiki FAQ

	5 FPS	10 FPS	15 FPS	20 FPS	25 FPS	30 FPS
128x96	Yes	Yes	Yes	Yes	Yes	Yes
160x120	Yes	Yes	Yes	Yes	Yes	Yes
176x144	Yes	Yes	Yes	Yes	Yes	Compressed
320x240	Yes	Compressed	Compressed	Compressed	Compressed	Compressed
352x288	Yes	Compressed	Compressed	Compressed	Compressed	Compressed
640x480	Compressed	Compressed	Compressed	No	No	No

The implemented system runs at a compromise of 320x240 pixels at 30 compressed frames per second. In order to accommodate the simplest algorithm and largest realized visual area, the camera is aligned such that the y-axis (the short direction) of the table uses the 240 pixel range while the x-axis (from goal to goal) of the table uses the 320 pixels.

To process the image data, the Java programming language was chosen. This is because the Java Media Framework (JMF) API allows for developer-friendly commands to acquire images while the rest of the Java platform handles high-level classes to handle image and color data [8]. By utilizing these features, fast, effective image processing is attained. The software runs in three stages:

1. The user selects the foosball table outline color.

   a. The playing surface is found by finding a rectangular object consisting of similar colors to the selected color.
   b. The number of pixels per square inch is calculated a posteriori by the knowledge that a foosball table is 24x48 inches.
2. The user selects the color of the human-controlled foosball players.
   a. The locations of each of the rows of players are calculated by finding the first instance of a similar color and calculating the center of mass near that pixel. After this is done, the distance between players on that row is calculated.
   b. The table is assumed to be symmetrical about the midfield, and thus, the computer controlled player locations are calculated by subtracting the x-axis value of each row, in pixels from the maximum table x-axis value in pixels.
3. The user selects the color of the foosball.
   a. The center of mass of the foosball is found by nearby color comparisons.
   b. All future searches for the ball are localized if possible given the previous location.

At this point, if the user reselects a color, it is assumed the user is choosing a new foosball color to track.

The localization of the foosball is done by taking a predictive location and then searching in a 20 pixel distance in every direction. If the ball is not found, the entire table is rescanned so as to find the ball. If, in 10 consecutive frames, the ball is not found, a goal is registered in the direction of the side the ball was last predicted to be going into.

Figure 2. Java tracking graphical user interface.

In addition to tracking the ball, on every frame, the location of each of the human-controlled players is recalculated. The graphical user interface used to visually show the tracking of the ball and the players is shown in *Figure 2* above. This is done by scanning all pixels at the x-axis value for each row within a 5-pixel range in any direction until the top player of the row is found. At this point, the distance between players on any given row is already known as a fixed value, so the locations of each of the other players on the row can be quickly computed.

The overall software flow for the image processing is summarized by the software flow chart in *Figure 3* below.

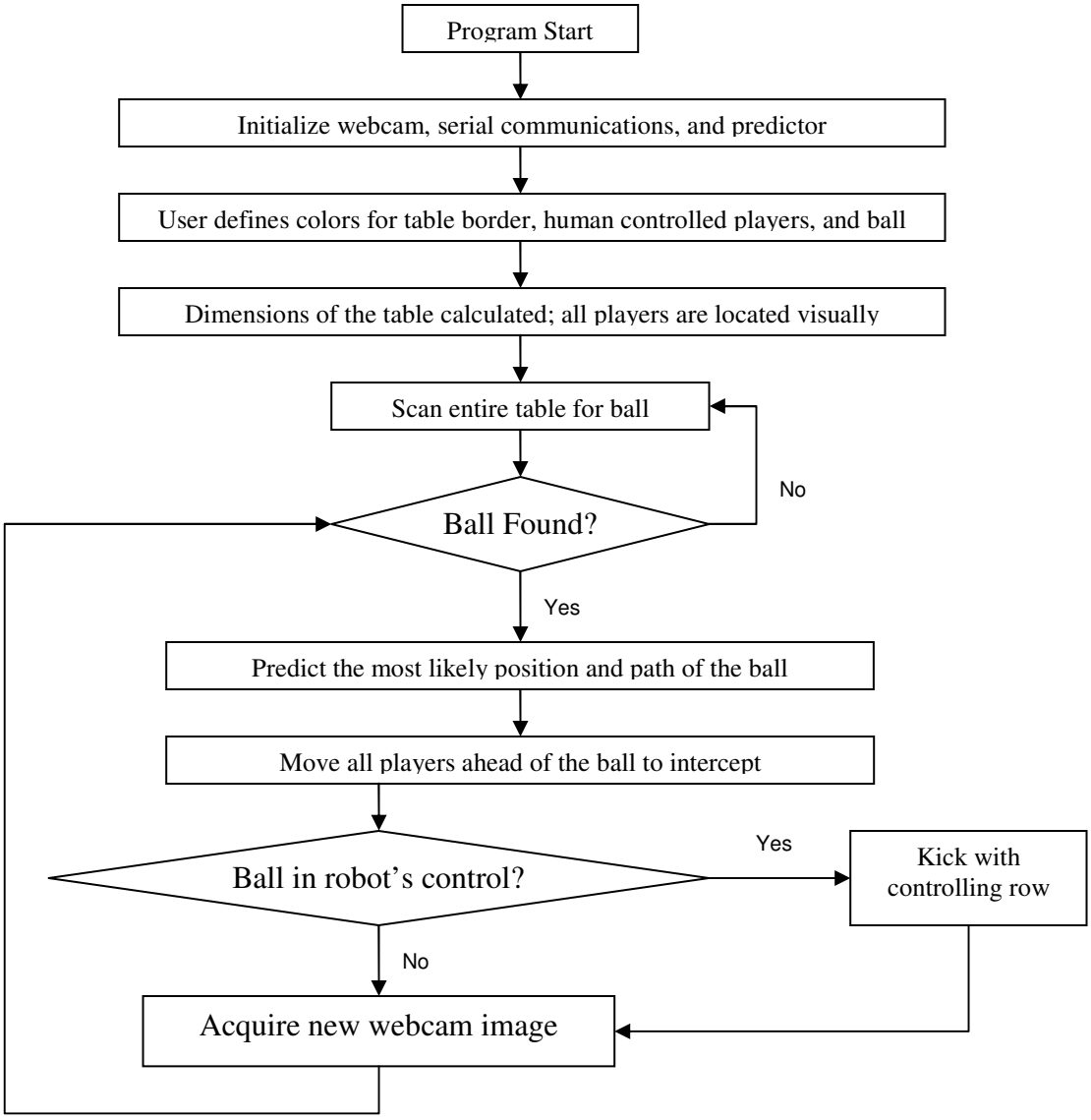


Figure 3. Image processing software flow chart.

5.3 PC-Controller Communications

The computer is linked to the microcontroller via an RS-232 serial link. This provides an easy to use interface with sufficient bandwidth for sending commands. The commands sent from the PC are in the form of a 2 byte packet created specifically for the AFT.

Each packet is generated to follow the form shown in *Table 3*. The packet will always start with the two most significant bits as 10. This serves to prevent the misreading of ambiguous all high

or all low packets. The third bit establishes the function of the packet. If the packet will be running simple servo control, the bit is set to 0. If the bit is set as 1, the packet serves some special function as described in *Table 4*, where the second byte of the packet determines the function of the command. If the packet is used for servo control, bits 3-5 are used for selecting which of the eight servos is to be used and bits 6-15 tell the selected servo where to rotate to. For the AX-12 servos, these ten bits indicate a physical position. For the HS-81 PWMs, the ten bits are either all ones to raise the player 90 degrees, or all zeros to lower the player to a downward position.

Table 3. Packet formation for servo control.

Bits	0-1	2	3-5	6-15
Usage	Reliability	Packet Function	Addressing	Positional Data

Table 4. Packet formation for special commands.

Second Byte	**Function**
0x01	Center all handles
0x02	Move all handles to start position
0x03	Move all handles into kick position
0x04	Move all handles into idle position
0x05	Move offense and midfield into defend position
0x06	Move offense and midfield into idle position

On the PC end, serial communication is handled by WriteS.java which houses various functions for generating the proper packet and then writing it to the serial port of the PC. The microcontroller then receives the packets and converts them to servo control messages, as described in the next section.

5.4 Servo Controller Board

The servo controller board interprets the instruction packets across the serial connection coming from the image processing computer. The controller then parses these instructions packets and then carries out the proper action required to control the correct servo. There are two major parts to the controller board design: the hardware design and the software design. The picture of the prototype servo controller board as implemented is shown in *Figure 4*.

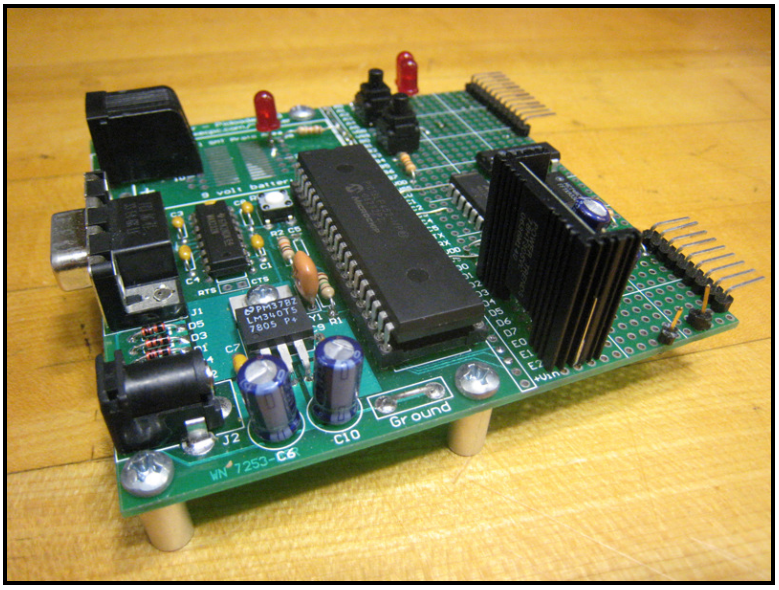


Figure 4. Prototype servo controller board.

5.4.1 Hardware Design

The hardware design connects all of the components of the servo controller board together onto a single printed circuit board (PCB) required to carry out the functions of communicating with the eight servos and the image processing PC. A block diagram overview of the servo controller board is shown in *Figure 5*.

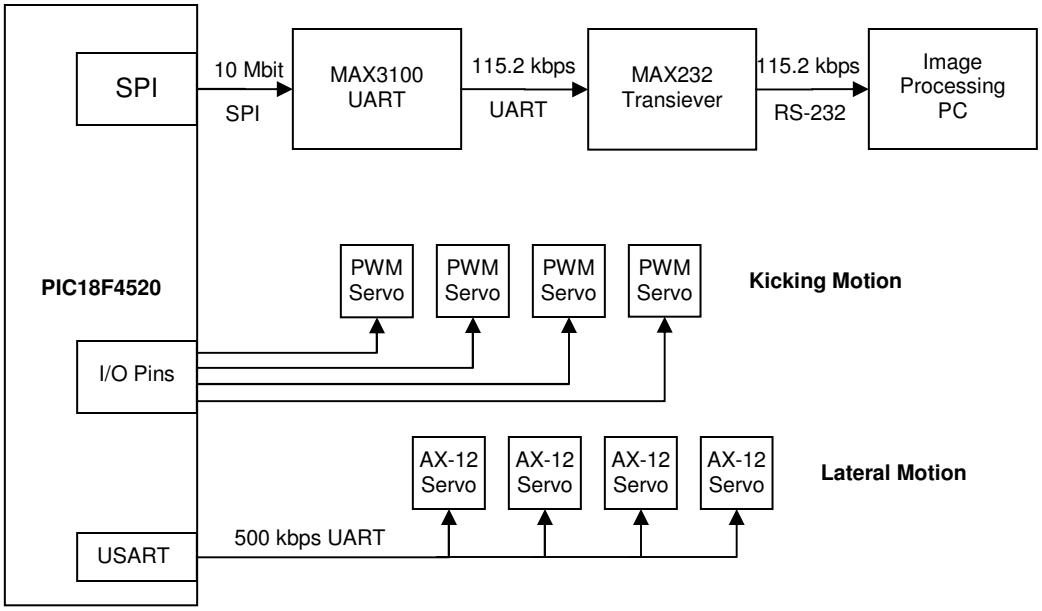


Figure 5. Servo controller board block diagram.

The main processor on the servo controller board is a Microchip PIC18F4520 microcontroller [9]. This microcontroller has all of the features necessary to implement the required functions of the servo controller.

The internal UART (Universal Asynchronous Receiver Transmitter) is used to directly communicate with the AX-12 lateral motion servos at a speed of 500 kbps. The AX-12 servos communicate on a single data bus, where receiver and transmitter function are implemented on a single line. Therefore, external circuitry is required for the transmit and the receive pins on the microcontroller, in order to control which function is currently connected to the common data bus. *Figure 6* shows this circuitry, as specified in the AX-12 User Manual [6].

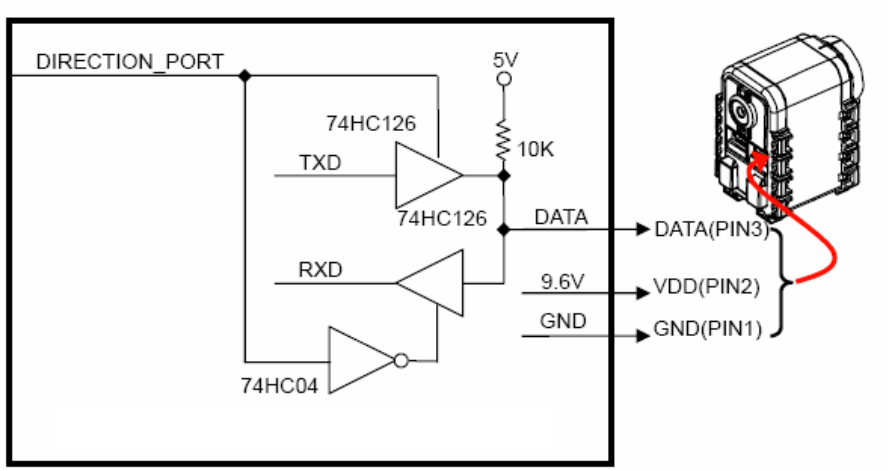


Figure 6. AX-12 communication circuit. Source: AX-12 User Manual

The *TXD* and *RXD* pins are connected to the *RX* and *TX* pins on the PIC18F, and the *DIRECTION_PORT* signal is connected to an output pin on the PIC18F (pin *RC0* in this design).

In order to produce the pulse-width modulated signal for the PWM kicking motion servos, a separate 8-pin PIC12F615 microcontroller is used for each PWM servo [10]. The main PIC18F communicates with each of these PIC12F controllers through two output signals: a *control* signal and a *direction* signal. The *control* signal indicates that a kicking motion is required, and the *direction* signal indicates in which direction that the kicking motion should go: forwards (for kicking) or backwards (for defending). The PIC12F controllers then interpret these signals and produce the required PWM signal for the kicking servo.

The PIC12F615 controllers produce a 50 Hz PWM signal with a duty cycle varying between 1.0 ms and 2.0 ms. This is the standard type of signal accepted by most PWM servos on the market, including the HS-81 servos used in this design. *Table 5* shows the produced PWM signals from the PIC12F controllers based on the input *control* and *direction* signals from the PIC18F.

Table 5. PIC12F615 PWM servo input/output signals.

Input Signals	PWM Duty cycle	PWM Period	Servo Position
control = 0 direction = 0	1.5 ms	50 Hz	Idle (down)
control = 1 direction = 0	2.0 ms	50 Hz	Kick (forwards)
control = 1 direction = 1	1.0 ms	50 Hz	Defend (backwards)
control = 0 direction = 1	Not Implemented	Not Implemented	Not Implemented

To communicate with the image processing PC, the PIC18F uses its internal SPI module to communicate with an external MAX3100 SPI to UART chip. The MAX3100 simply converts a SPI (synchronous peripheral interface) communications interface to a UART interface. It was necessary to implement a second UART for the PC communications externally because the PIC18F only has one internal UART module. The MAX3100 UART can be configured for several different baud rates, but in this design a baud rate of 115.2 kbaud was used in order to maximize the bandwidth. Refer to the MAX3100 datasheet for more information on its features and configurations [11]. The *IRQ* pin from the MAX3100 is connected to the *RB0/INT0* pin on the PIC18F controller, so that a signal can be produced for the PIC18F when a new byte has arrived from the image processing PC. The transmit and receive pins on the MAX3100 are further connected to a MAX232 transceiver [12]. From the MAX232 transceiver, the transmit and receive pins are wired into a standard DB9 connector to which a standard RS-232 cable can be attached to.

The servo controller board required two different power sources: one for the microcontrollers, and one for the servos. A separate power source is required for the servos because of their high rate of power consumption. Separating these two sources of power protects the microcontrollers

from being over-powered, and therefore increases reliability. Both sources of power can range from 9.0-12V, from which a voltage regulator then produces the required 5V for the microcontrollers and PWM servos. The AX-12s are directly connected to the high-power 9-12V source.

As a final hardware design, a printed circuit board was design using DipTrace in order to keep the size and complexity of the servo controller board down [13]. The resulting PCB design is shown in *Figure 7*.

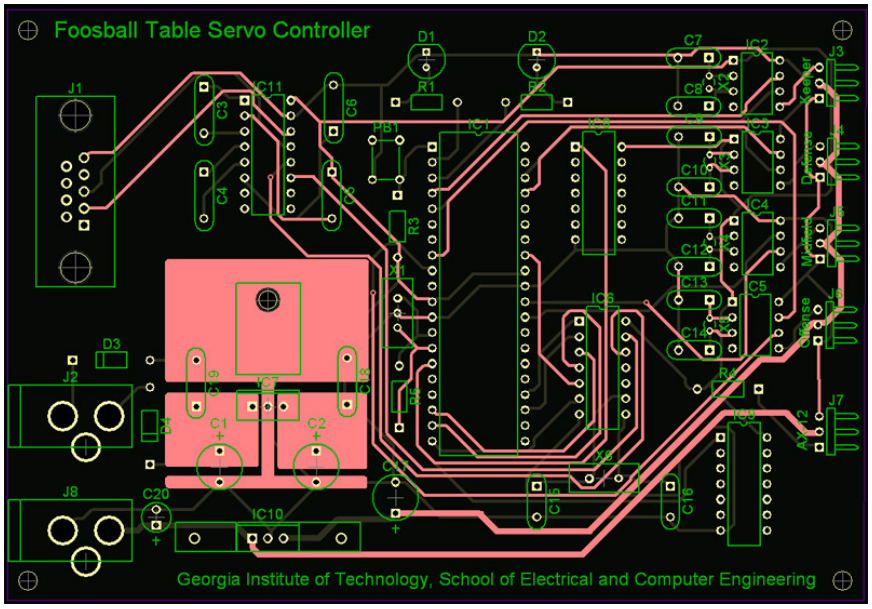


Figure 7. Servo controller board PCB design.

The full schematic and a parts list required to manufacture the servo controller board can be found in *Appendix D*.

5.4.2 Software Design

The PIC18F4520 source code is written in C and compiled with the Microchip C18 compiler [14]. In addition to the main program code, two libraries were written that implemented the

functions necessary for setting up and communicating with the AX-12 servos and the image processing PC.

The AX-12 lateral motion servos are controlled via a bi-directional UART data bus, which is connected to the PIC18F4520's internal UART module. The AX-12 servos have their own communications protocol that must be implemented in software in order to successfully communicate with them. The most common instruction is a WRITE instruction, which writes a value into memory to a specific AX-12 servo. The AX-12s are identified on the common data bus by a unique ID, which is specified in the instruction. For more information on how to implement the various instructions that are supported by the AX-12 communications protocol, please refer to the AX-12 User Manual [6]. A special library was written to easily implement all of the necessary functions to communicate with the AX-12 servos. *Table 6* shows an overview of these functions from the AX-12 library.

Table 6. AX-12 library function overview.

Function Name	Description
AX_SetId()	Assigns a new ID to a connected AX-12 servo.
AX_Ping()	Pings the connected AX-12 servos.
AX_RxByte()	Receives a byte from an AX-12 servo.
AX_ByteRdy()	Determines whether a byte is ready to be ready on the AX-12 data bus.
AX_TxPacket()	Sends a valid instruction packet to a connected AX-12 servo as per the AX-12 communications protocol.
AX_SendByte()	Sends a single byte onto the AX-12 data bus.
AX_SetupUSART()	Initializes the USART module for AX-12 communication.

The source code for the AX-12 library for the PIC18F4520 is shown in its entirety in *Appendix B*.

A second library was written for the PIC18F4520 to communicate with the MAX3100 SPI to UART chip in order to simplify the communication code from the main program code. This library includes functions to initialize, configure, read configuration and read/write data to the MAX3100 chip. The source code and descriptions for the functions can be found in *Appendix B*.

The main program code for the PIC18F4520 listens to the image processing PC's instruction packets through the MAX3100 SPI to UART chip, and once an instruction is received, parses it, and then carries out the appropriate servo function, as specified by the PC-to-Servo Communications Protocol in *Section 5.3*. The main program flow for the PIC184520 is shown in *Figure 8*.

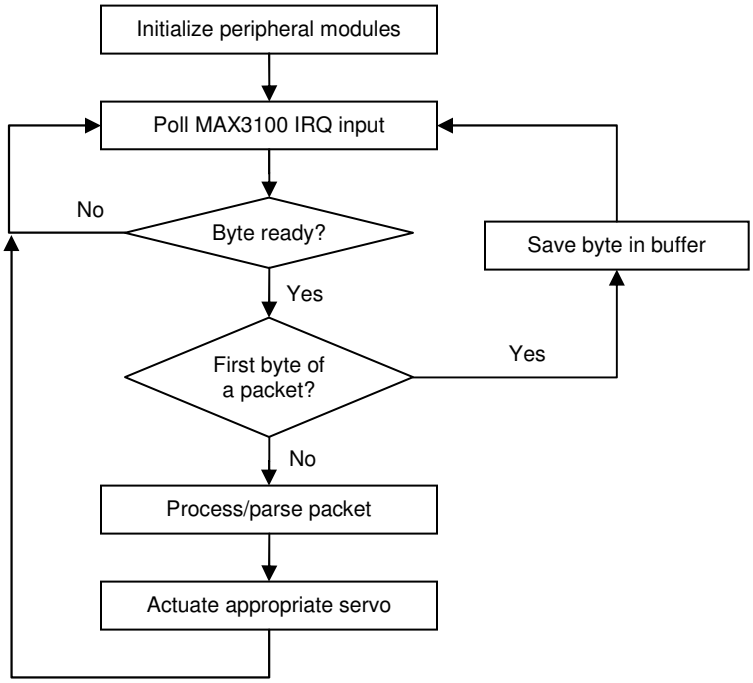


Figure 8. PIC18F4520 main program flow chart.

The initialization sets up all of the necessary required peripheral components and configures all of the input/output ports of the PIC18F to their proper values. Once initialized, the main loop continuously waits for a byte from the MAX3100 through the *RB0/INT0* pin. Once a byte is received, it is verified and determined if it is the first or second byte of an instruction packet. If it's the first byte, it is placed in a buffer and then waits for the next byte. If it is the second byte, the instruction in the buffer is parsed, and the proper function of the instruction is carried out, either moving a AX-12 or a PWM servo to a certain position. For more information on the specific function of the instruction packets, refer to *Section 5.3*. The main program source code is shown in its entirety in *Appendix B*.

The code for all of the PIC12F615 PWM controllers was written in assembly. Their programs simply continuously monitor the *control* and *direction* pins and change the duty cycle of the PWM module as required. The program flow for the PIC12F PWM controllers is shown in *Figure 9*.

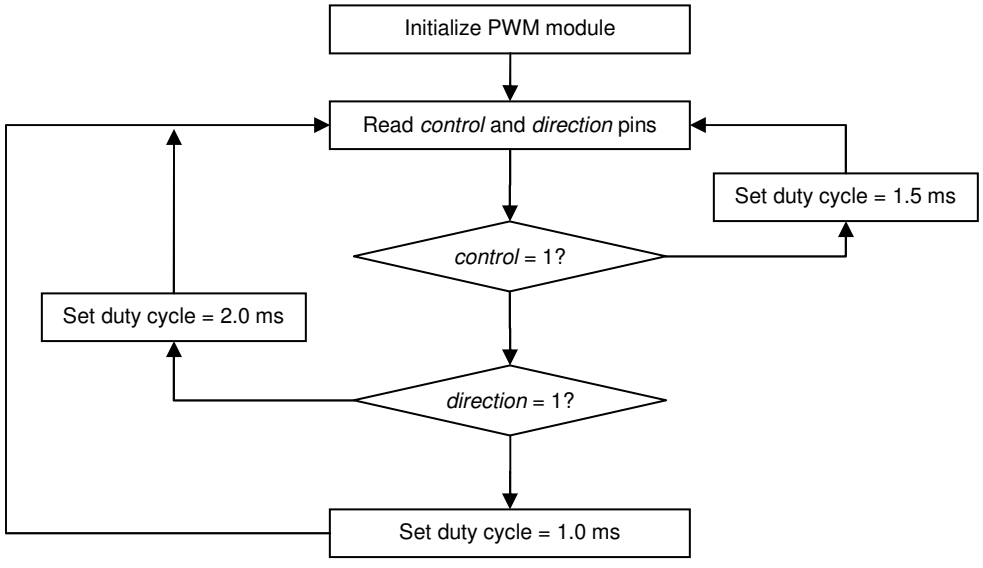


Figure 9. PIC12F615 program flow chart.

The full assembly source code for the PIC12F615 PWM controllers is shown in *Appendix C*.

5.5 Mechanical Design

The table construction centers on a simple DMI Sports foosball table. Attached directly to it are an overhead webcam mount and a side table to support the servos and circuitry. SolidWorks was used to create a preliminary mechanical design. A photograph of the actual prototype and the SolidWorks design are shown in *Figure 10*.

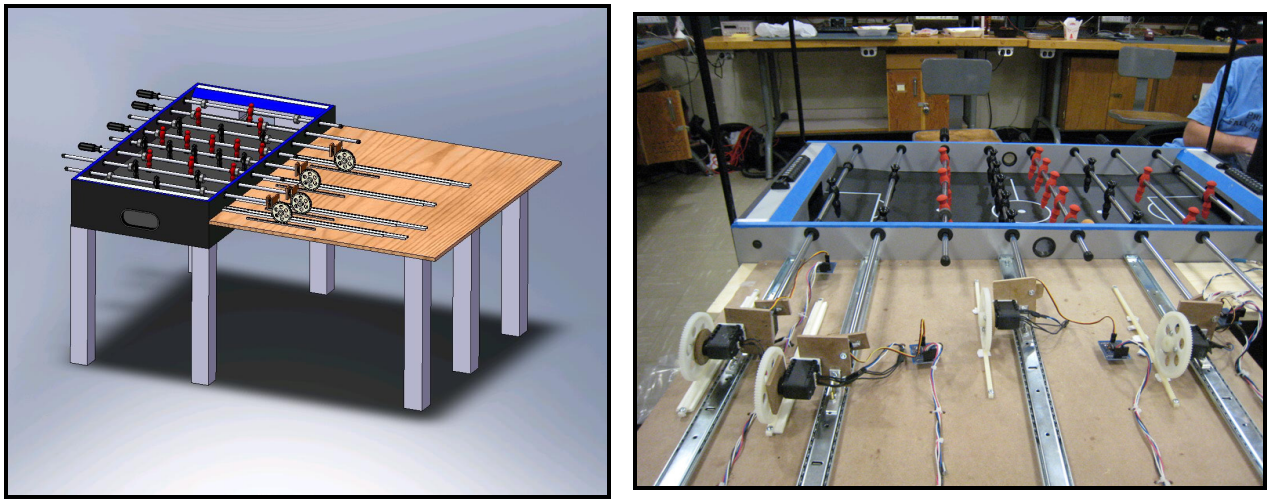


Figure 10. SolidWorks and actual mechanical design.

In order to fit the entire playing surface into the field of view of the webcam, it had to be placed at least 53" above the table. A metal frame reaching 54" above the table is attached to each corner of the table with a one foot wide medium density fiberboard (MDF) crosspiece attached, centered over the table. This crosspiece has a 3" hole bored into its center for the webcam to look down through. The webcam itself is attached with a #12 machine screw into its tripod mounting hole. A small piece of wood, cut at 45 degrees, must be mounted between the MDF and the webcam in order to yield a downward orientation.

In order to support the servos, a table must be built extending 3 feet from the original foosball table, two inches below the handles. The table is made of MDF, with 2"x2" legs, to maintain a light weight while providing ample support and rigidity. On top of the side-table, underneath each robotically controlled handle, drawer slides are mounted in order to keep the servo assembly rigid and parallel to the extended table. The assemblies units are attached to the drawer slides by an aluminum L-bracket. This serves to keep the servos at a constant height, while still allowing smooth lateral motion. Alongside each slide, a nylon rack is attached, while an accompanying 3.5" gear is mounted directly to the AX-12. The AX-12 and HS-81 servos are mounted on opposite sides of a small piece of MDF, with the HS-81 being attached directly to a rod holding players on the foosball table. As the AX-12 spins, it can now traverse the length of the rack, pushing and pulling the players with it and as the HS-81 spins, the players will spin as well. Double sided foam tape is placed between the racks and the table in order to keep the rack in place and at the proper height. Should the racks be too low, the gear and rack will not meet properly and slippage can occur. The SolidWorks design and the implemented design of the full servo assemblies can be found *Figure 11*.

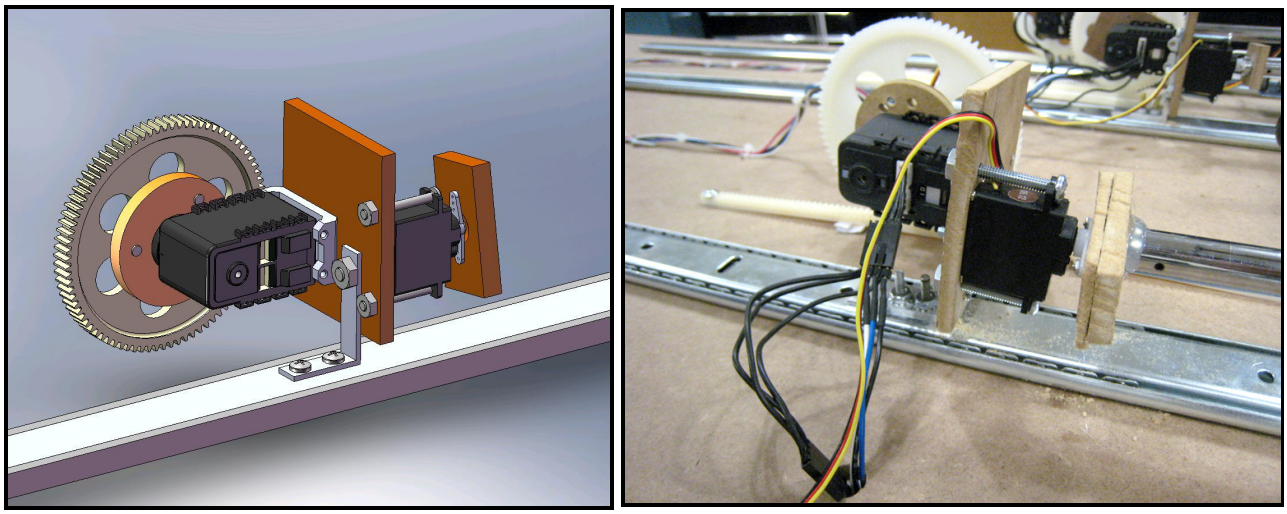


Figure 11. SolidWorks design and actual mechanical design of the servo assemblies.

The servo controller board and PWM controller boards are also mounted directly to the top of the extended side-table, with the necessary wiring for the assemblies run underneath the MDF.

For the complete mechanical design drawings, refer to *Appendix E*.

5.6 Codes and Standards

The United States Table Soccer Association outlines the rules to which a game of foosball should be played [15]. The automated foosball table is to conform to these rules, including the rule about not spinning the rods beyond 360 degrees. This prevents the servo motors from turning continuously and prevents the motors from over-torque and unfair game play.

In order for the machine to function properly and safely, a final product would need to abide by UL-22, the Underwriter Laboratories Standard for Amusement and Gaming Machines [16]. This standard defines various aspects of the machine's design, including the wiring and circuitry. Due to the interactive nature of the game, it must be able to withstand the wear and tear specified by the standard, guaranteeing a reliable product.

An example specification from the UL-22 standard that would need to be followed would be that electrical loads can not exceed 60A at 250V. The AFT uses two different power supplies so that the motors do not interfere with the power to the electronics. A 9V/300mA was used to power the microcontroller and the electronics, and a 9V/4A power supply was used to power the motors. The prototype is well below the UL-22 standard.

The RS-232 connector standards must be followed to allow communication from the microcontroller to the PC [17]. The receiving and transmission of data to and from the microcontroller is controlled by pins 2 and 3, the transmit and receive pins of a standard RS-232 pinout. The RS-232 connector standards need to be followed. The AFT does not use parity or stop bits features of the RS-232 standard.

As the baud rate is increased, the length of the RS-232 cable must be decreased in order to maintain reliable data transfer. The maximum cable length is 50ft or a capacitance of 2500 pF. However, at lower speeds the length of the cable can be increase. It has been know for cables lengths to surpass 150 feet with higher quality cables. *Table 7* below lists some common cable lengths and their maximum transmission rate.

Table 7. Typical RS-232 baud rates and maximum cable lengths.

Baud Rate	Maximum Cable Length (ft)
19200	20
9600	50
2400	100

The baud rate used in the AFT is 115200 and the cable length was 5ft. This cable length allows for reliable transmission at this baud rate.

USB 2.0 was originally intended to be used in the AFT design to meet the bandwidth requirements of high-speed web cam data transform, but due to hardware constraints of the purchased web cam, USB 1.1 was used instead. In order to get the maximum efficiency out of the available bandwidth, it was necessary to careful interface with the web cam. The standards set in the Universal Serial Bus Specifications were used [18].

The data rate for the USB 1.1 ranges from 1.5Mbps – 12Mbps depending of the device being used. Typically 1.5Mbps is the data rate for devices that do not require much bandwidth and 12Mbps for larger devices utilize high speed transfer.

The Java Media Framwork (JMF) was used to develop the image processing aspect of the prototype [19]. Java Media Framework applications allow real-time images to be easily incorporated with Java. This API (application program interface) within the program allows users to write applications consistent with the operating environment, therefore giving the application cross-platform compatibility.

5.7 Constraints, Alternatives, and Tradeoffs

The most significant constraint in the design and build of the autonomous foosball table is that of funding. With a budget under $500, the supply of parts that can be purchased becomes the primary limiting factor for building a prototype. For example, the lateral velocity could be increased by custom machining gears with a larger radius. However, in doing so, there would be a tooling setup cost of $1500, a cost well beyond the budget. Another constraint is to abide by standard foosball rules, which forbids spinning of the handles. Because of this, there could not be a continuous motor on the handles for kicking.

Other design alternatives were also considered in regards to the motion system instead of servos, including the use of linear actuators and stepper motors. However, linear actuators provide much less control than servos and stepper motors have an added complexity while not providing anything other than a slightly decreased cost in addition to not having the resolution available in

most servos. An alternative to using a computer and USB webcam would be to design all of the software on an FPGA and then interface it directly to a CMOS or CCD camera. This would have added increased processing and acquisition speed. However, at the onset of the design, the specifications of the SPC-900NC webcam suggested a higher frame rate than what was actually possible in the hardware, providing little reason to approach the problem with the FPGA and CMOS/CCD combination.

6. Schedule, Tasks, and Milestones

In the development phase of the prototype, it was important to maintain a schedule and continuously readjust the schedule to meet actual development time. It was also necessary to identify the various tasks required to complete the project, so that the work can be split up amongst the development team, and a proper plan for component integration can be made. As a whole, the development team was behind compared to the project schedule, but was able to make up time through efficient communication and development towards the end of the prototype development phase.

6.1 Schedule

The projected and actual prototype development schedules can be found in *Appendix G*. E-Mail proved to be a more effective way of communicating amongst the team member in terms of staying up-to-date with actual develop, as opposed to using the Google Calendar application.

The original schedule was designed to give the development team a month of test and revision time, thus ensuring enough time to come up with solutions to all problems and to finalize the design. Since the team ended up falling behind an several important aspects of the prototype design, this month was fully utilized to complete a working prototype of the project.

6.2 Tasks

There were several components of the prototype that were isolated into tasks that could be completed by one or more persons. The responsibility for completing each task was assigned to specific members of the development team. *Table 8* describes these tasks that were required to complete the automated foosball table, and how many people worked on each task.

Table 8. Overview of the tasks that were required to complete the prototype.

Task	Description	Persons
Mechanical assembly	Assembly of the mechanical parts required to move the handles of the foosball table, which includes the enclosures for the electrical components and a mounting mechanism for the ball-sensing camera.	2
Microcontroller development	Design and programming of the microcontroller and its circuitry that will manage the communication between the control motors of the foosball table and the image processing computer.	1
Image processor development	Development of the computer software that will interpret the images from the overhead-mounted webcam and run the necessary processing for ball detection and path prediction.	1
Computer-controlled communication	Design of the communications protocol between the processing computer and the motor controller.	1
Presentation materials	Development of the materials required for properly presenting the prototype development, including the final design documents.	4

6.3 Milestones

During the development phase, there are certain milestones that needed to be achieved before development could continue to the next phase. It was important to identify these milestones and make sure that they are met in a timely fashion in order to keep the development of the prototype on schedule. The important milestones and the date on which they were met are shown in *Table 9*.

Table 9. Development milestones and the date that they were met.

Milestone	Date Met
Preliminary parts search and selection of proper motors	8/28
Initial parts order made	9/5
Complete Proposal	9/12
Foosball table built	9/19
Basic AX-12 servo to microcontroller communication	10/3
Basic vision processing completed	10/3
Basic mechanical design completed, begin implementing	10/10
Initial mechanical assembly of 1 handle completed	10/17

PWM to microcontroller communication	10/24
Completed PC to microcontroller communications	11/7
Webcam mounted onto foosball table	11/21
Completed assembly of all 4 handles	11/28
Final component integration and testing completed	12/6

Refer to the Gantt chart in *Appendix G* to get a better overview of when these milestones were met compared to their projected date.

7. Project Demonstration

For the technical demonstration of the project, a game of foosball was played. This exhibited how all of the subsystems of the prototype worked together to form the completed design. Instructions for operation of the AFT can be found in *Figure 12*, below. The prototype performed well, although slower than desired. The primary failure of the AFT was that the players moved and kicked at less than one third of what was determined necessary in the initial technical specifications. This, in turn, caused goals to be missed in the form of scoring and blocking success rates as well.

Testing the movement specifications was simply a matter of watching the game and using a stopwatch to measure timing. From the distance moved and the time required to do so, a speed was determined. Movement resolution was calculated based on the circumference of the gear used and the fact that the servo has 1024 positions over 300 degrees of rotation. Camera frame rate and resolution was determined by viewing the output of the webcam on the laptop. Success rates were calculated by watching the game and dividing successes by attempts.

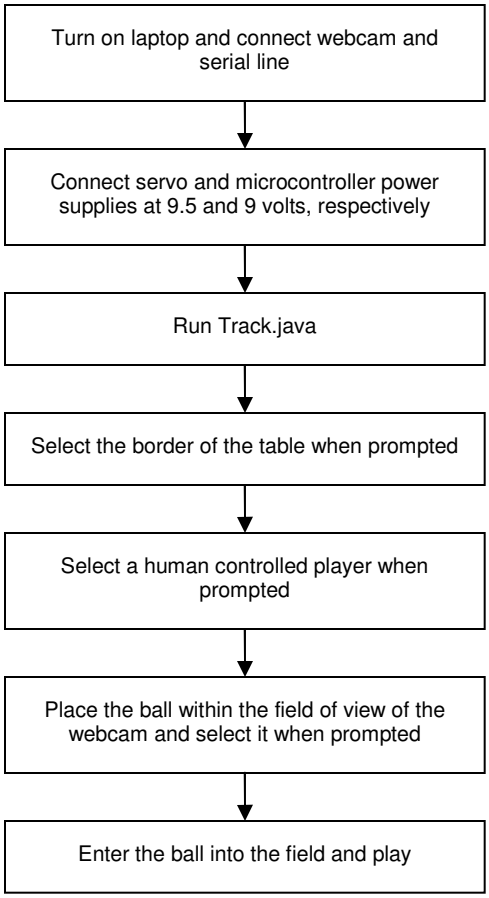


Figure 12. Instructions for starting a game with the automated foosball table.

8. Marketing and Cost Analysis

8.1 Marking Analysis

The targeted market for an AFT is the arcade entertainment industry. While individual customers are not likely to purchase such a foosball machine a few such sales to very active, wealthy foosball enthusiasts, looking for a unique in-home game may be expected. However, arcades or "fun" centers have the resources to purchase a more sophisticated form of a foosball table to offer a variety of challenges to their customers. The following is an example list of potential customers to which the AFT may be marketed:

- Chuck E. Cheese (http://www.chuckecheese.com/)
- Celebration Station (http://www.celebrationstation.com/)
- Malibu Grand Prix (http://www.malibugrandprix.com/)
- Dave and Busters (http://www.daveandbusters.com/)

Local bars and pubs may also be a potential market for such a foosball table, as they are also popular entertainment locations where customers would consider playing such a game and where similar games already exist.

In order to be successful, the foosball table must be a challenging and fun experience to the player. Such a machine can be tailored towards the intermediate to expert foosball player who has trouble finding a challenging human competitor. A successfully designed computer-controlled foosball player has the potential to approach perfection and challenge any expert-level player. Given a high expectation of victory, a computer-controlled opponent motivates an expert player to invest in playing the game, hence the marketable appeal of an AFT. Various difficulty

levels would also open up the marketability to the beginner-level foosball player. The marketing appeal is strong for a game that does not require a human opponent.

Figure 13. Star Kick foosball arcade game.

Currently there is only one AFT on the market, the Star-Kick, shown in *Figure 13*. The Star-Kick has its roots in a project by the School of Computer Science at the University of Freiburg in Germany. However, the marketed arcade version of the Star-Kick is listed at a price of $27,000, which is out of the range for smaller arcade centers and local bars [20]. The goal of the proposed prototype is to compete with this product by improving the gameplay, and significantly lower the cost of this unique type of arcade machine.

8.2 Cost Analysis

The final cost of an AFT should be competitive with other similar entertainment devices that the customer would place in their facility. Such devices include pool tables, pinball machines, air hockey, driving simulators, and arcade video games. The short list in *Table 8* shows the price for some of these typical entertainment machines.

Table 10. Typical entertainment machine prices. Source: ArcadeGameSuperstore

Name	Description	Price
NASCAR Pinball	Coin-operated NASCAR-themed pinball machine (new).	$4,395
Dracula Pinball	Bram Stroker's Dracula themed pinball machine (1993).	$3,249
Great American Air Hockey Table	8ft. air hockey table with electronic overhead scoring.	$3,470
Bubble Hockey Machine	Signature Stick Hockey bubble hockey machine.	$1,245
Ford Racing Full Blown	Driving arcade game based on recognizable Ford cars by Sega.	$8,795
Dance Dance Revolution Supernova	Arcade version of the DDR video game. 2 Players.	$13,895
Golden Tee Live 2007	The latest installment of the famous Golden Tee arcade machine.	$5,395
Great American Eagle Billiard Table	Standard bar-quality coin-operated billiard table.	$1,825
Tornado Cyclone II Foosball Table	Typical quality human vs. human foosball table.	$1,195
Average Price:		$4,829

From the list in *Table 10* of typical arcade machines, a targeted price range can be determined for an AFT. The higher priced arcade machines are typically larger, more complex, and usually have a computer opponent component; therefore it is viable to put the AFT into a similar range, depending on the extra features that the final product may have. A realistic selling price for such an arcade machine would fall in the price range of $5,000 - $8,000.

The cost of each unit comes out to $710 based upon the parts list in *Table 11*. Engineering costs, based upon four engineers working 16 hours a week for 14 weeks at $25 per hour come to a total of $22,400. Assuming that the AFT would sell at $5,000, a large profit can be made from such a product. *Appendix F* shows the profitability of an AFT at this price, given costs for parts, cost of labor, and the selling of 500 units. For a selling price of $5,000, a profitability of 43.2% can be achieved, which amounts to $1,081,214 in profit.

Table 11. Prototype parts cost.

Item	Quantity	Price
DMI Foosball Table	1	$100
Phillips SPC-900NC Webcam	1	$90
AX-12 Servo	4	$40 x 4 = $160
HS-81 Servo	4	$15 x 4 = $60
Cheap Laptop	1	$100
Serial Cable, 5ft	1	$2
Power Supply	1	$50
9V DC Adapter	1	$5
PIC 18F4520	1	$20
PIC 12F615	4	$1 x 4 = $4
MAX3100	1	$6
MAX232	1	$3
Voltage Regulators	1	$6
Wiring	1	$5
Oscillators	4	$1 x 4 = $4
Assorted ICs	1	$3
MDF/lumber	1	$20
Assorted Hardware	1	$10
Drawer Slide	4	$10 x 4 = $40
Shelf Rack	6	$4.5 x 6 = $27
Total	**1**	**$710.00**

In conclusion, the market for such an arcade machine is well established, and the void for such a unique machine gives it a great amount of selling power. Given costs of parts and labor, selling such an arcade machine at a competitive price compared to other machines in the market allows the manufacturer to make a substantial profit.

9. Summary and Conclusions

The prototype has been completed and is in full operation, yet some things would have to be changed before the design can be finalized. The biggest change would be to switch from a computer based system to an FPGA board for central processing. This is necessitated by the image acquisition portion of the project. As currently implemented, there is a large latency between a visual event and the computer receiving the image of it. By using a CCD or CMOS image sensor directly coupled to an FPGA board, the system would be capable of receiving and processing hundreds of frames per second. These frames could be processed directly by the hardware, with much lower limitations from software and bus latency. With a webcam linked to a computer, latency is introduced by the webcam itself, the USB chip at both sides of the line, the device's drivers, and the operating system. Implementing an FPGA board would add a good deal of complexity to the design, but it would eliminate many of those latency sources. A suitable FPGA board would likely cost more than a cheap computer, but it would be able to handle the duties of all of the PIC boards and allows the use of an image sensor much cheaper than an equivalent webcam. The price of the system as a whole would remain similar to the current projection, but due to the large change, a new prototype cycle would have to begin to handle the alteration in system architecture and rewriting of the code.

Another major change that would be introduced if another cycle were to begin would lie with the chosen motors. In order to achieve the speeds desired for lateral movement, gears of at least 10 inches in diameter would be needed. This would be large and expensive to implement, so movement would likely switch to a pulley system with a high speed servo. A faster servo would cost more, but gears would no longer be needed so the overall price should not change much.

The speed of the kicking servos is also far too slow, so DC motors with an H-bridge and an electric stop mechanism would be used. While the price of this system would not change much, it does increase the complexity of the system. The electrical stop would stop the motor by discharging the electromechanical energy stored in the motor through a series of transistors to ground, which would only be turned on when a stop command is issued. Without it, the motor would slow to a stop and the system would be unsure of the precise locations of the motors.

One addition to the table would be to augment the basic scoring system. While a scoring system has been partially implemented in the prototype, it was not refined due to time constraints. Adding a simple seven segment LED display would be inexpensive and easy to implement and could display the current score accurately. If more time and money were devoted to this system, a small LCD screen could be used for aesthetic appeal. A high end model could go as far as to store user profiles and statistics.

Another modification would be to modify the predictive algorithm. In the prototype that was implemented, a linear average of previous locations is used to predict the most likely path of the ball. This is due to the limitations in computing power as well as the general inaccuracies introduced in the webcam. Without a faster processing device and more real-time data, there is little advantage to implementing a more complex prediction algorithm. However, with a FPGA design, the acquisition and processing would not be as costly, and therefore, a Kalman filter could be used, which would provide more accurate predictions to the future state of the ball and players.

While the AFT does not meet the initial design goals we had, it does operate at enough of a capacity to serve as a proof of concept prototype. The tracking software works well, as well as control of the servos. With a better frame rate from the camera and faster physical operation, the game could easily be very difficult to beat. By scaling the speed of operation and prediction algorithms, virtually any level of difficulty could be attained.

10. References

[1] Wikipedia, :"Foosbot", 2007 [Online]. Available: http://en.wikipedia.org/wiki/Foosbot. [Accessed: Sept. 11, 2007]

[2] University of Freiburg, "KiRo - The Table Soccer Robot", 2007 [Online]. Available: http://www.informatik.uni-freiburg.de/~kiro/english/. [Accessed: Sept. 12, 2007]

[3] Georgia Tech's Computational Perception Laboratory, "Eye Detection and Tracking", 2007 [Online]. Available: http://www.cc.gatech.edu/cpl/projects/pupil/index.html. [Accessed: Sept. 12, 2007]

[4] Wikipedia, "PIC Microcontroller", 2007 [Online]. Available: http://en.wikipedia.org/wiki/PIC_microcontroller. [Accessed: Sept. 11, 2007]

[5] Association for the Advancement of Artificial Intelligence, "AI Overview", 2007 [Online]. Available: http://www.aaai.org/AITopics/html/welcome.html. [Accessed: Sept. 12, 2007]

[6] Tribotix Robotis Technical Staff, *Dynamixel AX-12 User's Manual*, Tribotix, Jun. 16 2006 [Online]. Available: http://www.tribotix.info/Downloads/Robotis/Dynamixels/AX-12(english).pdf [Accessed Oct 3, 2007]

[7] Hi-Tec Technical Staff, *HS-81 Micro Servo Specification*, Hi-Tec [Online]. Available: http://www.robotshop.ca/PDF/hs81.pdf [Accessed Nov. 3 2007]

[8] Java Media Framework (JMF) API, 2007 [Online] Available: http://java.sun.com/products/java-media/jmf/ [Accessed: Oct. 1, 2007]

[9] Microchip Technical Staff, *PIC18F4520 Datasheet*, Microchip, 2007 [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/39631D.pdf [Accessed Sept 20, 2007]

[10] Microchip Technical Staff, *PIC12F615 Datasheet*, Microchip, 2006 [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/41302A.pdf [Accessed Oct 28, 2007]

[11] Maxim IC Technical Staff, *MAX3100 Datasheet*, Maxim IC, Dec. 2001 [Online]. Available: http://datasheets.maxim-ic.com/en/ds/MAX3100.pdf [Accessed Oct. 15 2007]

[12] Maxim IC Technical Staff, *MAX232 Datasheet*, Maxim IC, Jan. 2006 [Online]. Available: http://datasheets.maxim-ic.com/en/ds/MAX220-MAX249.pdf [Accessed Oct. 10 2007]

[13] DipTrace Homepage [Online]. Available: http://www.diptrace.com/ [Accessed Nov. 15 2007]

[14] Microchip Technical Staff, *Microchip C18 Compiler User's Guide*, Micrchip, 2005 [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/C18_User_Guide_51288j.pdf [Accessed Oct 3, 2007]

[15] United States Table Soccer Association, "USTSA Rules of Play", 2007 [Online]. Available: http://www.foosball.com/learn/rules/ustsa/ [Accessed Sept. 14, 2007]

[16] Underwriter Laboratories, Inc., *UL-22 Amusement and Gaming Machines Standard*, [Online] Available: http://ulstandardsinfonet.ul.com/scopes/scopes.asp?fn=0022.html [Accessed Dec 2, 2007]

[17] Electronics Industries Association, "EIA Standard RS-232-C Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Data Interchange", August 1969, reprinted in Telebyte Technology *Data Communication Library*, Greenlawn NY, 1985.

[18] Compaq Computer Corporation et. al., *Universal Serial Bus Specification*, USB Implementers Forum, Inc., April 27, 2000 (Rev. 2.0).

[19] Sun Develpoment Network, *Java Media Framework API*, Sun Microsystems 2007 [Online] Available: http://java.sun.com/products/java-media/jmf/ [Accessed Oct. 10 2007]

[20] Merkur Star Kick, 2007 [Online]. Available: http://www.merkur-starkick.de/default.htm. [Accessed: Sept. 13, 2007].

Appendix A

Image Processing Source Code

```
/**
 * AI.java
 * This class initiates a movement of a row of players to a specified location.
 *
 * @author Team FIFA, ECE4884L01, Georgia Institute of Technology
 * @version 1.0, December 2007
*/

public class AI {
	private double pixelsPerInch = 0;

	/**
    * Class constructor specifying the pixels per inch of the table.
	 *
	 * @param ppi The number of pixels per inch defined on the table.
    */
	public AI(double ppi) {
		pixelsPerInch = ppi;
	}

	/**
	 * Uses the goalie to intercept a ball going toward the goal
	 *
	 * @param c The serial controller object used to send data to the servo control board
	 * @param yInterceptPosition The predicted future location of the ball, given in pixels
	 *                           from the bottom of the table.
	*/
	public void goalIntercept(WriteS c, double yInterceptPosition) {
		if (yInterceptPosition != -1) {
			//we use the "- 8.75" because the hard stoppers imposed on the goalie do
not permit
			//the goalie's movement to the absolute edge of the table.
			c.move(3,(yInterceptPosition / pixelsPerInch) - 8.75);
		}
	}

	/**
	 * Uses the defensemen to intercept a ball going toward the goal
	 *
	 * @param c The serial controller object used to send data to the servo control board
	 * @param yInterceptPosition The predicted future location of the ball, given in pixels
	 *                           from the bottom of the table.
	*/
	public void defenseIntercept(WriteS c, double yInterceptPosition) {
		if (yInterceptPosition != -1) {
			c.move(2,yInterceptPosition / pixelsPerInch);
		}
	}

	/**
	 * Uses the midfielders to intercept a ball going toward the goal
	 *
	 * @param c The serial controller object used to send data to the servo control board
	 * @param yInterceptPosition The predicted future location of the ball, given in pixels
	 *                           from the bottom of the table.
	*/
	public void midfieldIntercept(WriteS c, double yInterceptPosition) {
		if (yInterceptPosition != -1) {
			c.move(1,yInterceptPosition / pixelsPerInch);
		}
	}

	/**
	 * Uses the strikers to intercept a ball going toward the goal
	 *
	 * @param c The serial controller object used to send data to the servo control board
	 * @param yInterceptPosition The predicted future location of the ball, given in pixels
	 *                           from the bottom of the table.
	*/
	public void strikerIntercept(WriteS c, double yInterceptPosition) {
		if (yInterceptPosition != -1) {
```

```
                        c.move(0,yInterceptPosition / pixelsPerInch);
                }
        }
}
```

```
/**
 * FrameGrabber.java
 * This class is used to grab an image from the webcam.     It uses an internal timeout to
prepare an image
 *
 * @author Team FIFA, ECE4884L01, Georgia Institute of Technology
 * @version 1.0, December 2007
*/

package jmfYUV;

import javax.swing.*;
import javax.swing.event.*;
import java.io.*;
import javax.media.*;
import javax.media.format.*;
import javax.media.util.*;
import javax.media.control.*;
import javax.media.protocol.*;
import java.util.*;
import java.awt.*;
import java.awt.image.*;
import java.awt.event.*;
import java.util.Properties;
import javax.swing.Timer;

public class FrameGrabber
{
        private DataSource dataSource = null;
        private Player player = null;
        private CaptureDeviceInfo di = null;
        private MediaLocator ml = null;
        private FormatControl formatControls[];
        private Buffer buf = null;
        private Image img = null;
        private VideoFormat vf = null;
        private BufferToImage btoi = null;
        private YUVFormat userFormat = null;
        private FrameGrabbingControl fgc = null;
        private boolean newFrame = false;
        private final static String DEFAULT_DEV_NAME = "v4l:Philips SPC 900NC webcam:0";
        private final static int FRAMERATE = 15;

        /**
         * Class constructor.
                */
        public FrameGrabber()
        {
                di = CaptureDeviceManager.getDevice(DEFAULT_DEV_NAME); //locate the predefined
image source
                Dimension viewSize = new Dimension(320, 240); //preset the size to 320x240 px
                Format[] cfmt = di.getFormats(); //get available formats
                for (int i = 0; i < cfmt.length; i++) {
                        if (cfmt[i] instanceof YUVFormat) {
                                userFormat = (YUVFormat)cfmt[i];
                                Dimension d = userFormat.getSize();
                                if (viewSize.equals(d)) //find the dimensional format we want
                                        break;
                                userFormat = null;
                        }
                }
                ml = di.getLocator(); //create the CaptureDeviceInfo object we want
        }

        /**
         * Grab a new frame and store it internally.
         */
        private void grabFrame() {
                        fgc = (FrameGrabbingControl)
                                player.getControl("javax.media.control.FrameGrabbingControl");
                        buf = fgc.grabFrame(); //grab a buffer of the frame
```

```
                        // Convert it to an image
                        btoi = new BufferToImage((VideoFormat)buf.getFormat());
                        img = btoi.createImage(buf);
        }

        /**
         * Initialize frame grabs, calculating the internal refresh rate to grab at.
         */
        public void start() {
                try {
                        dataSource = Manager.createDataSource(ml);
                        dataSource.start();

                        formatControls = ((CaptureDevice) dataSource).getFormatControls();
                        for (int i=0; i < formatControls.length; i++) {
                                System.out.println(i+":"+formatControls[i].toString());
                        }
                        formatControls[0].setFormat(userFormat);
                        player = Manager.createRealizedPlayer(dataSource);
                        player.start();
                        fgc = (FrameGrabbingControl)
                        player.getControl("javax.media.control.FrameGrabbingControl");
                } catch (Exception e) {}
                ActionListener grabNewFrame = new ActionListener() { //this interrupts every
second
                        public void actionPerformed(ActionEvent evt) {
                                grabFrame();
                                newFrame = true;
                        }
                };
                int framerateTimeout = (int)(1.0/(FRAMERATE * 1.0) * 1000) + 1;
                new Timer(framerateTimeout, grabNewFrame).start(); //interrupt timeout
        }

        /**
         * Public accessor for the internal image from the last grab
         *
         * @return The last grabbed image
         * @see Image
         * @see #getBufferedImage
         */
        public Image getImage() {
                newFrame = false;
                return img;
        }

        /**
         * Public accessor for the internal image from the last grab, in BufferedImage form
         *
         * @return The last grabbed image
         * @see #getImage
         */
        public BufferedImage getBufferedImage() {
                return (BufferedImage)(getImage());
        }

        /**
         * Public accessor to tell a calling class when a new image from the webcam is available
         *
         * @return A boolean indicating if a new image is available
         */
        public boolean imageAvailable() {
                return newFrame;
        }

        /**
         * Public accessor for the frame rate of the frame grabber
         */
        public int getFrameCount() {
                return FRAMERATE;
```

```
        }
}
```

```
/**
 * Predictor.java
 * This class is used to predict locations, determine when goals are scored, and predict y-
intercept locations of a row of players.
 * It uses a simple prediction based upon average slopes of x- and y-locations.  When a goal is
scored, this class plays a
 * random audio clip depending on the most likely scorer.
 *
 * @author Team FIFA, ECE4884L01, Georgia Institute of Technology
 * @version 1.0, December 2007
*/
import java.awt.image.*;
import java.awt.*;
import sun.audio.*;
import java.io.*;
import java.util.Random;

public class Predictor {
	private int xLocation, yLocation, yFutureLoc, xFutureLoc, count;
	private int[][] lastPositions = new int[2][3]; //3 frames @ 30fps = 0.1s
	private double ballRadius = 0; //we need this to calculate bounces
	private int colorSearchSize; // "distance" to search in color space for colored objects
	private double deltaY, deltaX; //x- and y- slopes
	private boolean bottomDefenseControl = false; //this determines which defenseman should
grab control of the ball
	private int myscore = 0, oppscore = 0, lostballframes = 0; //used in scoring algorithm

	/**
	* Adds a new ball location to the stack.
	*
	* @param xloc The new x-location of the ball, in pixels
	* @param yloc the new y-location of the ball, in pixels
	* @see #clear
	*/
	public void addPosition(int xloc,int yloc) {
		lostballframes = 0;
		int i;
		for (i = 0; i < 2; i++) {
			lastPositions[0][i] = lastPositions[0][i + 1];
			lastPositions[1][i] = lastPositions[1][i + 1];
		}
		lastPositions[0][2] = xloc;
		lastPositions[1][2] = yloc;
		count++;
		if (count > 2) {
			count = 3; //if we have more than 2 frames of interest, we do not care of
previous values
		}
	}

	/**
	* Predicts the goalie intercept path with the predicted y-location of the ball
	*
	* @param xpos The new x-location of the ball, in pixels
	* @return A double predicting the pixel location the goalie should go to
	* @see #findDefenseIntercept
	* @see #findMidfieldPosition
	* @see #findStrikerPosition
	*/
	public double findGoalIntercept(int xpos) {
		deltaY = (lastPositions[1][0] + lastPositions[1][2]) / 2.0; //average of the last
2 y slopes ((y3 - y2) + (y2 - y1));
		return yLocation + deltaY;
	}

	/**
	* Aligns the defense to block shots by computing the angle between the ball and the goal
	*
	* @param goalXPos The x-location of the robot's goal, in pixels
	* @param rowXPos The x-location of the robot's defense line, in pixels
	* @param tableYMin The minimum y-location of the foosball table, in pixels
```

```
	 * @param tableYMax The maximum y-location of the foosball table, in pixels
	 * @param playerSpacing The spacing between defensive players, in pixels
	 * @return A double predicting the pixel location the defense should go to
	 * @see #findGoalIntercept
	 * @see #findMidfieldPosition
	 * @see #findStrikerPosition
	 */
	public double findDefenseIntercept(int goalXPos, int rowXPos, int tableYMin, int
tableYMax, int playerSpacing) {
		double controlZone = 0.45; //this is used as a sort of schmitt trigger, where in
the middle 10% of the table,
                                                                                //the
defensive players do not sporadically attempt to gain control of the ball
		double pixelControlZone = tableYMin + (tableYMax - tableYMin) * controlZone;
		double returnLocation = 0; //the y-location to go to
		double goalLocation = tableYMax - ((tableYMax - tableYMin) / 2);
		if (yLocation < pixelControlZone) {
			bottomDefenseControl = false;
			returnLocation = goalLocation - (goalLocation - yLocation + deltaX) /
(xLocation + deltaY - goalXPos) * (rowXPos - goalXPos) + playerSpacing;
		} else if (yLocation > (tableYMax - pixelControlZone)) {
			bottomDefenseControl = true;
			returnLocation = goalLocation + (yLocation + deltaX - goalLocation) /
(xLocation + deltaY - goalXPos) * (rowXPos - goalXPos);
		} else if (bottomDefenseControl) {
			returnLocation = 136; //go to the maximum position
		} else {
			returnLocation = 0; //go to a zero position
		}
		return returnLocation;
	}

	/**
	* Aligns the midfield to block kicks from the opponent
	*
	* @param tableYMax The maximum y-location of the foosball table, in pixels
	* @param playerSpacing The spacing between defensive players, in pixels
	* @return A double predicting the pixel location the midfield should go to
	* @see #findGoalIntercept
	* @see #findDefenseIntercept
	* @see #findStrikerPosition
	*/
	public double findMidfieldPosition(int playerSpacing, int tableYMax) {
		double returnLocation = yLocation + deltaY;
		while (returnLocation < (tableYMax - playerSpacing)) {
			returnLocation += playerSpacing;
		}
		return returnLocation;
	}

	/**
	* Aligns the strikers to block kicks from the opponent
	*
	* @param tableYMax The maximum y-location of the foosball table, in pixels
	* @param playerSpacing The spacing between defensive players, in pixels
	* @return A double predicting the pixel location the strikers should go to
	* @see #findGoalIntercept
	* @see #findDefenseIntercept
	* @see #findMidfieldPosition
	*/
	public double findStrikerPosition(int playerSpacing, int tableYMax) {
		double returnLocation = yLocation + deltaY;
		while (returnLocation < (tableYMax - playerSpacing)) {
			returnLocation += playerSpacing;
		}
		return returnLocation;
	}

	/**
	* Public accessor to return the last calculated y-velocity of the ball
	*
```

```
        * @return A double representing the y-velocity of the ball, in pixels per frame
        * @see #xVelocity
        */
        public double yVelocity() {
                return deltaY;
        }

        /**
        * Public accessor to return the last calculated x-velocity of the ball
        *
        * @return A double representing the x-velocity of the ball, in pixels per frame
        * @see #xVelocity
        */
        public double xVelocity() {
                return deltaX;
        }

        /**
        * Public accessor to return the predicted x-location of the ball
        *
        * @return A double representing the predicted x-location of the ball, in pixels
        * @see #getFutureY
        * @see #getxPosition
        * @see #getyPosition
        */
        public int getFutureX() {
                return xFutureLoc;
        }

        /**
        * Public accessor to return the predicted x-location of the ball
        *
        * @return A double representing the predicted x-location of the ball, in pixels
        * @see #getFutureX
        * @see #getxPosition
        * @see #getyPosition
        */
        public int getFutureY() {
                return yFutureLoc;
        }

        /**
        * Public accessor to return the predicted x-location of the ball
        *
        * @return A double representing the last known x-location of the ball, in pixels
        * @see #getyPosition
        * @see #getFutureX
        * @see #getFutureY
        */
        public int getxPosition() {
                return xLocation;
        }

        /**
        * Public accessor to return the last known y-location of the ball
        *
        * @return A double representing the last known y-location of the ball, in pixels
        * @see #getxPosition
        * @see #getFutureX
        * @see #getFutureY
        */
        public int getyPosition() {
                return yLocation;
        }

   /**
        * Upon a lost frame, this function clears the last known positions of the ball, to avoid
making false assumptions of the velocity
        *
        * @see #addPosition
        */
```

```
	public void clear() {
		int i;
		count = 0;
		for (i = 0; i < 3; i++) {
			lastPositions[0][i] = 0;
			lastPositions[1][i] = 0;
		}
	}

	/**
	* Upon a lost frame, this function determines whether a goal was likely scored or not.
10 missing frames of the ball indicate
	* a likely goal.  If a goal is determined to be scored, play an audio clip depending on
the most likely scorer.
	*/
	public void missedFrame(int tableMinX, int tableWidth) {
		String [] goodGoals = {"goal.wav", "chant.wav", "cheer.wav", "crowd.wav",
"goal2.wav", "freesoundGoal.wav", "computer_error.wav", "destroyed.wav",
"no_ordinary_machine.wav"};
		String [] badGoals = {"boo.wav", "about_time.wav", "doh.wav", "doing_my_best.wav",
"error.wav", "dah_duh_duh.wav"};
		String playfile = "";
		lostballframes++;
		if (lostballframes == 10) {
			try {
				Random randNum = new Random();;
				if (xLocation + deltaX < (tableWidth + tableMinX) / 2) { //opponent
scored
					oppscore++;
					playfile = badGoals[randNum.nextInt(badGoals.length)];
				} else { //computer scored
					playfile = goodGoals[randNum.nextInt(badGoals.length)];
					myscore++;
				}
				InputStream in = new FileInputStream("" + playfile);
				AudioStream as = new AudioStream(in);
				AudioPlayer.player.start(as); //play the audio file
			} catch (Exception e) {}

		}
	}

	/**
	* Public accessor to return the computer's score
	*
	* @return The computer's score
	*/
	public int getMyScore() {
		return myscore;
	}

	/**
	* Public accessor to return the human's score
	*
	* @return The human's score
	*/
	public int getOppScore() {
		return oppscore;
	}

	/**
	* Public mutator to set the ball radius
	*
	* @param br The radius of the ball, in pixels
	*/
	public void setBallRadius(int br) {
		ballRadius = br;
	}

	/**
    * Sets the search size in RGB for locating objects on the table
```

```
         *
         * @param value An integer representing the maximum value any R/G/B data can be away from
a determined color value
         */
        public void setColorSearchSize(int value) {
                colorSearchSize = value;
        }

        /**
    * This calculates the diameter of the ball being used by searching within the localized space
for similar colors
         *
         * @param x A webcam An image from the webcam
         * @param r The red pixel value of the ball
         * @param g The green pixel value of the ball
         * @param b The blue pixel value of the ball
         * @param xpos The x-position of the ball
         * @param ypos The y-position of the ball
         * @param searchDist The pixel distance to search in each direction for similar colors
    */
        public void calcBallSize(BufferedImage x, int r, int g, int b, int xpos, int ypos, int
searchDist) {
                int i, j, numpos = 0;
                for (i = xpos - searchDist; i < xpos + searchDist; i++) {
                        for (j = ypos - searchDist; j < ypos + searchDist; j++) {
                                Color pixel = new Color(x.getRGB(i,j));
                                if (pixel.getRed() > r-colorSearchSize && pixel.getRed() <
r+colorSearchSize
                                && pixel.getGreen() > g-colorSearchSize && pixel.getGreen() <
g+colorSearchSize
                                && pixel.getBlue() > b-colorSearchSize && pixel.getBlue() <
b+colorSearchSize){
                                        numpos++;
                                }
                        }
                }

                //assume a round (circular) ball.
                // area = pi * r * r
                // r = sqrt(area / pi)
                ballRadius = Math.sqrt(numpos / Math.PI);
        }

        /**
    * Recalculate the most most likely position of the ball in the next frame.
    */
        public void recalculate() {
                //use y = mx + b
                if (count > 2) {
                   //average of the last 2 y slopes ((y3 - y2) + (y2 - y1));
                        deltaY = (lastPositions[1][1] - lastPositions[1][0] + lastPositions[1][2]
- lastPositions[1][1]) / 2.0;
                        //average of the last 2 x slopes ((x3 - x2) + (x2 - x1));
                        deltaX = (lastPositions[0][1] - lastPositions[0][0] + lastPositions[0][2]
- lastPositions[0][1]) / 2.0;
                        yLocation = lastPositions[1][2] + (int) deltaY;
                        xLocation = lastPositions[0][2] + (int) deltaX;
                        yFutureLoc = yLocation + (int)(deltaY * 1.75); //There is a lag of
approximately 1.75 frames from the webcam
                        xFutureLoc = xLocation + (int)(deltaX * 1.75);
                } else {
                        xLocation = lastPositions[0][2];
                        yLocation = lastPositions[1][2];
                        yFutureLoc = yLocation;
                        xFutureLoc = xLocation;
                }
        }

        /**
    * Class constructor.
    */
```

```
	public Predictor() {
		int i;
		deltaY = deltaX = 0.0;
		for (i = 0; i < 3; i++) { //initialize all previous locations to (0,0)
			lastPositions[0][i] = 0;
			lastPositions[1][i] = 0;
			yFutureLoc = 0;
			xFutureLoc = 0;
		}
	}
}
```

```
/**
 * Table.java
 * This class creates a virtual foosball table.  It includes player information as well as
 * information on the table color and size.
 *
 * @author Team FIFA, ECE4884L01, Georgia Institute of Technology
 * @version 1.0, December 2007
*/

import java.awt.*;
import java.awt.image.*;

public class Table {
	private int colorSearchSize = 0; // "distance" to search in color space for colored
objects
	private int playerHeight = 0; // average player height, in pixels
	private int playerCount = 0; // total number of players on the board
	private int rOppColor, gOppColor, bOppColor; //opponent color
	private int rOutlineColor, gOutlineColor, bOutlineColor; //table outline color
	private int minX = 0, minY = 0, maxX = 0, maxY = 0;
	public double pixelsPerInch = 1.0; //calculated later
	boolean foundPlayers = false;

	// 4 rows, 5 items of interest per row
	// 1. Players on that row
	// 2. Distance between players on that row
	// 3. X-position of the row
	// 4. Minimum Y position
	// 5. Current Y position
	private int[][] rows = new int[4][5];
	private int[][] myrows = new int[4][5];

	/**
    * Class constructor.
    */
	public Table() {
		char i, j;
		rows[0][0] = 3; //3 players in attack row
		rows[1][0] = 5; //5 players in midfield row
		rows[2][0] = 2; //2 players in defense row
		rows[3][0] = 1; //1 player as the goalie
		for (i = 1; i < 4; i++) {
			for (j = 0; j < 4; j++) {
				rows[j][i] = 0; //initialize all locational data to 0
				myrows[j][i] = 0;
			}
		}

		for (i = 0; i < 4; i++) {
			playerCount += rows[i][0]; //count the total number of players
			myrows[i][0] = rows[i][0]; //assume a "fair" game where the computer has
the same number of players
		}
	}

	/**
    * Calculates the pixels per inch in the foosball table.  A standard foosball table has a
playing surface of
	 * 24 by 48 inches.  Once we have the boundary as found by the picture, we can verify the
physical dimensions.
	 * This data is stored internally in the "pixelsPerInch" variable
	 *
	 * @see #getInches
	 */
	public void calcScale() {
		double wcalc = getTableWidth() / 48.0; //48 inches wide
		double hcalc = getTableHeight() / 24.0; //24 inches tall
		if (wcalc > (1.2 * hcalc) || wcalc < (0.8 * hcalc)) {
			System.out.println("Warning!  Table dimensions do not agree with standard
foosball table!");
		}
```

```
                pixelsPerInch = (wcalc + hcalc) / 2; //approximate the pixels per inch between the
height and width
                System.out.println("Found " + pixelsPerInch * pixelsPerInch + " pixels per square
inch");
        }

        /**
    * Calculates the "inch" value from a pixel value, using the getInches
         *
         * @return A double representing the physical number of inches
         * @see #calcScale
         */
        public double getInches(double p) {
                return (p / pixelsPerInch);
        }

        /**
    * Public accessor to return the average player diameter, in pixels
         *
         * @return The player cross-section "height" or diameter, in number of pixels
         */
        public int getPlayerHeight() {
                return playerHeight;
        }

        /**
    * Public accessor to return the distance between players on a given row
         *
         * @param row The row number (0-3) to return data about
         * @return The distance between the centers of each player on a row
         */
        public int getRowSpacing(int row) {
                return myrows[row][1];
        }

        /**
    * Public accessor to return the minimum y-location of a given row
         *
         * @param rownum The row number (0-3) to return data about
         * @param myPlayers A boolean representing if the data to be returned is about the
computer's players,
         *                  as opposed to the human's players
         * @return The minimum y-location, in pixels, of a particular row
         */
        public int getRowMinY(int rownum, boolean myPlayers) {
                int y = 0;
                if (myPlayers) {
                        y = myrows[rownum-1][3];
                } else {
                        y = rows[rownum-1][3];
                }
                return y;
        }

        /**
    * Public accessor to get the x-position of a given row
         *
         * @param row The row number (0-3) to set
         * @param myPlayers  A boolean representing if the data to be returned is about the
computer's players,
         *                  as opposed to the human's players
         * @return The x-position of the row, in pixels
         */
        public int getRowXPosition(int row, boolean myPlayers) {
                int x = 0;
                if (myPlayers) {
                        x = myrows[row][2];
                } else {
                        x = rows[row][2];
                }
                return x;
```

```
        }

        /**
    * Relocates all of the opponent players to track them as they move.
         *
         * @param x The image to locate the players on
         */
        public void relocate(BufferedImage x) {
                int i, j;
                if (foundPlayers) { //basic error check to ensure we CAN locate the players
                        for (i = 0; i < 4; i++) { //loop through all rows
                                for (j = minY; j < maxY; j++) { //start looping through all y
pixels in a specified x-position
                                        Color pixel = new Color(x.getRGB(rows[i][2],j));
                                        if (pixel.getRed() > rOppColor-colorSearchSize &&
pixel.getRed() < rOppColor+colorSearchSize
                                        && pixel.getGreen() > gOppColor-colorSearchSize &&
pixel.getGreen() < gOppColor+colorSearchSize
                                        && pixel.getBlue() > bOppColor-colorSearchSize &&
pixel.getBlue() < bOppColor+colorSearchSize){
                                                rows[i][4] = j; //update the new position
                                                break; //we only need to know the top player of a
row, because the others are fixed positions away
                                        }
                                }
                        }
                }
        }

        /**
    * Public accessor to return the minimum x-position, in pixels, of the table
         *
         * @return The minimum x-position of the table
         * @see #getMinY
         * @see #getMaxX
         * @see #getMaxY
         * @see #getTableWidth
         * @see #getTableHeight
         */
        public int getMinX() {
                return minX;
        }

        /**
    * Public accessor to return the minimum y-position, in pixels, of the table
         *
         * @return The minimum y-position of the table
         * @see #getMinX
         * @see #getMaxY
         * @see #getMaxX
         * @see #getTableWidth
         * @see #getTableHeight
         */
        public int getMinY() {
                return minY;
        }

        /**
    * Public accessor to return the maximum y-position, in pixels, of the table
         *
         * @return The maximum y-position of the table
         * @see #getMinY
         * @see #getMinX
         * @see #getMaxX
         * @see #getTableWidth
         * @see #getTableHeight
         */
        public int getMaxY() {
                return maxY;
        }
```

```
        /**
    * Public accessor to return the maximum x-position, in pixels, of the table
         *
         * @return The maximum x-position of the table
         * @see #getMinX
         * @see #getMaxY
         * @see #getMinY
         * @see #getTableWidth
         * @see #getTableHeight
         */
        public int getMaxX() {
                return maxX;
        }

        /**
    * Public accessor to compute the width of the table, in pixels
         *
         * @return The minimum x-position subtracted from the maximum x-position of the table, in
pixels
         * @see #getMinX
         * @see #getMinY
         * @see #getMaxX
         * @see #getMaxY
         * @see #getTableHeight
         */
        public int getTableWidth() {
                return (maxX - minX);
        }

        /**
    * Public accessor to compute the height of the table, in pixels
         *
         * @return The minimum y-position subtracted from the maximum y-position of the table, in
pixels
         * @see #getMinX
         * @see #getMinY
         * @see #getMaxX
         * @see #getMaxY
         * @see #getTableWidth
         */
        public int getTableHeight() {
                return (maxY - minY);
        }

        /**
    * Public mutator to set the x-position of a given row
         *
         * @param row The row number (0-3) to set
         * @param pos The x-position, in pixels
         */
        public void setXPos(int row, int pos) {
                rows[row][2] = pos;
        }

        /**
    * Public mutator to set the minimum y-position of a given row
         *
         * @param row The row number (0-3) to set
         * @param pos The minimum y-position, in pixels
         */
        public void setTop(int row, int pos) {
                rows[row][3] = pos;
        }

        /**
    * Public accessor to get the minimum y-position of a given row
         *
         * @param row The row number (0-3) to set
         * @return The minimum y-position of the row, in pixels
         */
        public int getTop(int row) {
```

```
                return rows[row][3];
        }

        /**
    * Sets the search size in RGB for locating objects on the table
         *
         * @param value An integer representing the maximum value any R/G/B data can be away from
a determined color value
         */
        public void setColorSearchSize(int value) {
                colorSearchSize = value;
        }

        /**
    * Draws the players onto a Graphics object
         *
         * @return The modified graphics object
         * @see Graphics
         */
        public Graphics drawPlayers(Graphics g) {
                int i,j,xpos,ypos;
                g.setColor(Color.RED); //Draw in Red
                for (i = 0; i < 4; i++) {
                        xpos = rows[i][2];
                        for (j = 0; j < rows[i][0]; j++) { //loop through the number of players on
a row
                                ypos = rows[i][4] + j * rows[i][1] + playerHeight / 2; //compute
the y-location of each player
                                g.fillOval(xpos+5,ypos+25,10,10); //draw circles over the players
                        }
                }
                return g;
        }

        /**
    * Determines the row that is in control of a ball, given the x-location of that ball.
         *
         * @param x The x-location of the ball
         * @return A control row.  (0-3) indicates the human's row is in control while (4-7)
indicates the robot's row is in control
         */
        public int findController(int x) {
                int i, r = 0, min = 9999;
                for (i = 0; i < 4; i++) { //loop through all rows
                        if (Math.abs(rows[i][2] - x) < min) {
                                min = Math.abs(rows[i][2] - x);
                                r = i;
                        }
                        if (Math.abs(myrows[i][2] - x) < min) {
                                min = Math.abs(myrows[i][2] - x);
                                r = i + 4;
                        }
                }

                return r;
        }

        /**
    * Locates the computer's players from data about the human's players, assuming a symmetrical
table
         */
        public void findMyPlayers() {
                int i;
                for (i = 0; i < 4; i++) {
                  myrows[i][1] = rows[i][1]; //assume the same number of players on either side
                  myrows[i][2] = (maxX - rows[i][2] + minX); //set the x-location of each row
                  myrows[i][3] = (maxY - rows[i][3]) + minY; //set the minimum y-location of each
row
                  myrows[i][4] = myrows[i][3]; //set the "current" y-location from the minimum y-
location
                }
```

```
	}

	/**
	 * Determines the location of each player row and then computes the distances between
players on that row.
	 * After this is done, the robot's positions are determined, assuming a symmetrical table
	 *
	 * @param x An image from the webcam
	 * @param mousex The x-location to analyze for color of the human players
	 * @param mousey The y-location to analyze for color of the human players
	 * @see #findMyPlayers
	*/
	public void findRows(BufferedImage x, int mousex, int mousey) {
		x = blur(x); //blurring smoothes out color fluctuatoins
		Color pixel = new Color(x.getRGB(mousex,mousey));
		int r = pixel.getRed();
		int g = pixel.getGreen();
		int b = pixel.getBlue();
		rOppColor = r;
		gOppColor = g;
		bOppColor = b;
		/*
		// The following code works to visually locate the players.  However, due to
inconsistent lighting concerns,
		// it was taken out of the final code, to be replaced with predefined values.

		int i, j, loop1, loop2, loop3, loop4, startpos, rowdist = 0, lastPlayerPos;
		int ypos = 0, xpos = 0, countpx = 0, rowcount = 0, startPlayerPos = maxY,
playerMaxPos = 0;
		for(loop1 = (int)((maxX - minX) * .2); loop1 < maxX; loop1++) {
			for(loop2 = minY; loop2 < maxY; loop2++) {
				pixel = new Color(x.getRGB(loop1,loop2));
				if (pixel.getRed() > r-colorSearchSize && pixel.getRed() <
r+colorSearchSize
				&& pixel.getGreen() > g-colorSearchSize && pixel.getGreen() <
g+colorSearchSize
				&& pixel.getBlue() > b-colorSearchSize && pixel.getBlue() <
b+colorSearchSize) {
					for (loop3 = loop1 - 5; loop3 < loop1 + 15; loop3++) {
						for (loop4 = loop2 - 5; loop4 < loop2 + 15; loop4++)
{
							pixel = new Color(x.getRGB(loop3,loop4));
							if (pixel.getRed() > r-colorSearchSize
								&& pixel.getRed() < r+colorSearchSize
								&& pixel.getGreen() > g-
colorSearchSize
								&& pixel.getGreen() <
g+colorSearchSize
								&& pixel.getBlue() > b-
colorSearchSize
								&& pixel.getBlue() <
b+colorSearchSize) {
									if (loop4 < startPlayerPos) {
										startPlayerPos =
loop4;
									}
									if (loop4 > playerMaxPos) {
										playerMaxPos = loop4;
									}
									ypos += loop4;
									xpos += loop3;
									countpx++;
							}
						}
					}
					if (countpx > 2) {
						rows[rowcount][2] = (int) Math.floor(xpos /
countpx); //set x
						rows[rowcount][3] = startPlayerPos; //set top
System.out.println("Found player row " + rowcount + " at " + rows[rowcount][2] + ":" +
startPlayerPos);
```

```
                                        playerHeight += (playerMaxPos - startPlayerPos);
                                        ypos = xpos = countpx = playerMaxPos = 0;
                                        rowcount++;
                                        startPlayerPos = maxY;
                                        loop1 += 20; //skip forward to the next row
                                        break; //break the vertical search
                                } else {
                                        //false positive
                                        ypos = xpos = countpx = playerMaxPos = 0;
                                        startPlayerPos = maxY;
                                }
                        }
                }
                if (rowcount == 4) {
                        System.out.println("Found all player rows.  Moving on...");
                        break; //break after we find 4 rows
                }
        }

        //find the average player height
        playerHeight = playerHeight / rowcount;
        System.out.println("Average player height: " + playerHeight + " px");

        //now, find the distance between players on the row
        for (i = 0; i < 4; i++) {
                if (rows[i][0] > 1) {
                        ypos = 0;
                        System.out.println("Searching for more players on row " + i + " at
xpos " + rows[i][2]);
                        countpx = rowdist = 0;
                        startPlayerPos = maxY;
                        lastPlayerPos = rows[i][3];
                        for (j = rows[i][3] + 2 * playerHeight; j < maxY; j++) {
                                if (rows[i][2] >= 10) {
                                        startpos = -10;
                                } else {
                                        startpos = -1 * rows[i][2];
                                }
                                for (int n = startpos; n <= -1 * startpos; n++) {
                                        pixel = new Color(x.getRGB(rows[i][2] + n,j));
                                        if (pixel.getRed() > r-colorSearchSize &&
pixel.getRed() < r+colorSearchSize
                                        && pixel.getGreen() > g-colorSearchSize &&
pixel.getGreen() < g+colorSearchSize
                                        && pixel.getBlue() > b-colorSearchSize &&
pixel.getBlue() < b+colorSearchSize) {
                                                for (loop3 = j - 5; loop3 < j + 15; loop3++)
{
                                                        pixel = new Color(x.getRGB(rows[i][2]
+ n,loop3));
                                                        if (pixel.getRed() > r-
colorSearchSize
                                                                && pixel.getRed() <
r+colorSearchSize
                                                                && pixel.getGreen() > g-
colorSearchSize
                                                                && pixel.getGreen() <
g+colorSearchSize
                                                                && pixel.getBlue() > b-
colorSearchSize
                                                                && pixel.getBlue() <
b+colorSearchSize) {
                                                                        if (loop3 <
startPlayerPos) {
                                                                                startPlayerPos
= loop3;
                                                                        }
                                                                        countpx++;
                                                        }
                                                }
                                        }
```

```
                                        }
                                        if (countpx > 2) {
                                                rowdist += (startPlayerPos - lastPlayerPos);
                                                lastPlayerPos = startPlayerPos;
                                                j += playerHeight * 2;
                                                startPlayerPos = maxY;
                                                countpx = 0;
                                                rows[i][1] = rowdist;
                                                rowdist = 0;
                                                break;

                                        } else {
                                                countpx = 0;
                                                startPlayerPos = maxY;
                                        }
                                }
                        } else {
                                rows[i][1] = 0;
                        }
                }
                */

                //The following represents the predefined values that replaced the above code.
These values were found through
                // simple image analysis
                playerHeight = 6;

                //distance between players on the row
                rows[0][1] = 52;
                rows[1][1] = 31;
                rows[2][1] = 78;
                rows[3][1] = 0;

                //x-location data
                rows[0][2] = 98;
                rows[1][2] = 169;
                rows[2][2] = 241;
                rows[3][2] = 277;

                //y-location data
                rows[0][3] = rows[0][4] = 52;
                rows[1][3] = rows[1][4] = 52;
                rows[2][3] = rows[2][4] = 55;
                rows[3][3] = rows[3][4] = 104;

                findMyPlayers();
                foundPlayers = true;
        }

        /**
    * Finds the maximum distance any player on a row can move
         *
         * @param x An image from the webcam
         */
        public void findRange(BufferedImage x) {
                int i, j;
                for (i = 0; i < 4; i++) { //loop through all rows
                        for (j = minY; j < maxY; j++) {
                                Color pixel = new Color(x.getRGB(i,j));
                                if (pixel.getRed() > rOppColor-colorSearchSize && pixel.getRed() <
rOppColor+colorSearchSize
                                && pixel.getGreen() > gOppColor-colorSearchSize && pixel.getGreen()
< gOppColor+colorSearchSize
                                && pixel.getBlue() > bOppColor-colorSearchSize && pixel.getBlue() <
bOppColor+colorSearchSize){
                                        rows[i][4] = rows[i][3] - j;
                                        break; //we only need to find the first 2 players on a row
to calculate the distance between players
                                }
                        }
                }
```

```
	}

	/**
    * Finds the minimum and maximum x and y locations of the table through color analysis of the
outline
	 *
	 * @param x An image from the webcam
	 * @param mousex The x-location to analyze for color of the table outline
	 * @param mousey The y-location to analyze for color of the table outline
	 */
	public void findOutline(BufferedImage x, int mousex, int mousey) {
		x = blur(x);
		Color pixel = new Color(x.getRGB(mousex, mousey));
		rOutlineColor = pixel.getRed();
		gOutlineColor = pixel.getGreen();
		bOutlineColor = pixel.getBlue();
		int i, j;
		int imHeight = x.getHeight(), imWidth = x.getWidth();

		//first, find the leftmost location of the table (minX)
		boolean found = false;
		j = imHeight / 2; //assume the table falls somewhere in the middle of the image
		for(i = 0; i < imWidth; i++) {
			pixel = new Color(x.getRGB(i,j));
			if (pixel.getRed() > rOutlineColor-colorSearchSize && pixel.getRed() <
rOutlineColor+colorSearchSize
				&& pixel.getGreen() > gOutlineColor-colorSearchSize &&
pixel.getGreen() < gOutlineColor+colorSearchSize
				&& pixel.getBlue() > bOutlineColor-colorSearchSize &&
pixel.getBlue() < bOutlineColor+colorSearchSize) {
					found = true;
			} else if (found) { //wait until we fall off the outline to assign the
location
				minX = i;
				break;
			}
		}

		//then, find the rightmost location of the table (maxX)
		found = false;
		for(i = imWidth - 1; i > 0; i--) {
			pixel = new Color(x.getRGB(i,j));
			if (pixel.getRed() > rOutlineColor-colorSearchSize && pixel.getRed() <
rOutlineColor+colorSearchSize
				&& pixel.getGreen() > gOutlineColor-colorSearchSize &&
pixel.getGreen() < gOutlineColor+colorSearchSize
				&& pixel.getBlue() > bOutlineColor-colorSearchSize &&
pixel.getBlue() < bOutlineColor+colorSearchSize) {
					found = true;
			} else if (found) { //wait until we fall off the outline to assign the
location
				maxX = i;
				break;
			}
		}

		//then, find the topmost location of the table (minY)
		found = false;
		i = imWidth / 2; //assume the table falls somewhere in the middle of the image
		for(j = 0; j < imHeight; j++) {
			pixel = new Color(x.getRGB(i,j));
			if (pixel.getRed() > rOutlineColor-colorSearchSize && pixel.getRed() <
rOutlineColor+colorSearchSize
				&& pixel.getGreen() > gOutlineColor-colorSearchSize &&
pixel.getGreen() < gOutlineColor+colorSearchSize
				&& pixel.getBlue() > bOutlineColor-colorSearchSize &&
pixel.getBlue() < bOutlineColor+colorSearchSize) {
					found = true;
			} else if (found) { //wait until we fall off the outline to assign the
location
				minY = j;
```

```
                                break;
                        }
                }

                //finally, find the bottommost location of the table (maxY)
                found = false;
                for(j = imHeight-1; j > 0; j--) {
                        pixel = new Color(x.getRGB(i,j));
                        if (pixel.getRed() > rOutlineColor-colorSearchSize && pixel.getRed() <
rOutlineColor+colorSearchSize
                                && pixel.getGreen() > gOutlineColor-colorSearchSize &&
pixel.getGreen() < gOutlineColor+colorSearchSize
                                && pixel.getBlue() > bOutlineColor-colorSearchSize &&
pixel.getBlue() < bOutlineColor+colorSearchSize) {
                                        found = true;
                        } else if (found) { //wait until we fall off the outline to assign the
location
                                maxY = j;
                                break;
                        }
                }

                System.out.println("Found outline: x(" + minX + "-" + maxX + "), y(" + minY + "-"
+ maxY + ")");
        }


        /**
     * Computes a blurred image, blurring 1 pixel in each direction
         *
         * @param x An image to blur
         * @return The blurred image
         * @see BufferedImage
         */
        public BufferedImage blur(BufferedImage x) {
                int blurWidth = 1;
                int w = x.getWidth();
                int h = x.getHeight();
                int i, j, r=0, g=0, b=0, p, q;
                int blursize = (2 * blurWidth + 1) * (2 * blurWidth + 1);
                BufferedImage res = new BufferedImage(w,h,BufferedImage.TYPE_INT_RGB);
                for (i = blurWidth; i < w-blurWidth; i++) {
                        for (j = blurWidth; j < h-blurWidth; j++)
                                r = g = b = 0;
                                for (p = -1 * blurWidth; p <= blurWidth; p++) { //blur 1 px left
and right
                                        for (q = -1 * blurWidth; q <= blurWidth; q++) {{ //blur 1
px up and down
                                                Color c = new Color(x.getRGB(i + p, j + q));
                                                r += c.getRed();
                                                g += c.getGreen();
                                                b += c.getBlue();
                                        }
                                }
                                Color n = new Color(r/blursize, g/blursize, b/blursize); //average
all the pixels in the region
                                res.setRGB(i,j,n.getRGB()); //assign the new value to the created
image
                        }
                }
                return res;
        }
}
```

```
/**
 * Track.java
 * This class is the main control class to all the foosball files.  It initiates all the required
objects from assistant classes
 * (AI, FrameGrabber, Predictor, Table, and WriteS).  It tracks the foosball via a limited search
space on every frame, checking
 * the FrameGrabber object for whenever a new image is available.  It also initiates a GUI to
draw the location of the ball as
 * well as crosshairs over the ball.
 *
 * @author Team FIFA, ECE4884L01, Georgia Institute of Technology
 * @version 1.0, December 2007
*/
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.Timer;
import jmfYUV.*;

public class Track extends Frame {
        private BufferedImage image1; //the last image from the webcam
        private FrameGrabber vision1; //the image acquiring object
        private int xloc, yloc, imHeight, imWidth, framecount = 0, FPS = 0;
        private float camFrameRate; //the frame rate we're acquiring at
        private int rBallColor, gBallColor, bBallColor; //ball colors
        private int rOppColor, gOppColor, bOppColor; //opponent colors
        private int colorSearchSize = 20; //colors within this RGB distance will be matched
        private int searchSize = 20; //number of pixels to look for the ball in each direction
        private boolean found; //boolean indicating if the ball is found
        private int stage = 0;
        private Predictor p = new Predictor();
        private Table t = new Table();
        private WriteS c = new WriteS();
        private AI a;

        /**
         * Delay the main process for 2 milliseconds
         */
        public void sleep() {
                try {
                        Thread.sleep(2);
                } catch(Exception e) {
                        System.out.println("Error");
                }
        }

        /**
         * Redraw everything in the GUI
         *
         * @param g The graphics object to draw on
         */
        public void paint(Graphics g) {
                int xline, yline;
                g.drawImage(image1, 10, 30, this); //first, add the last webcam image

                if (stage > 0) {
                        g.setColor(Color.BLUE); //draw the table's bounding box
                        g.drawRect(t.getMinX() + 10, t.getMinY() + 30, t.getTableWidth(),
t.getTableHeight());
                }

                if (stage > 2) {
                        g = t.drawPlayers(g); //draw the human's players
                }

                if (found) {
                        //draw the crosshairs
                        xline = 10 + xloc;
                        yline = 30 + yloc;
                        g.setColor(Color.GREEN);
                        g.drawLine(xline,30,xline,imHeight + 30);
```

```
                        g.drawLine(10,yline,imWidth + 10,yline);
                        g.drawRect(xline - searchSize, yline - searchSize, searchSize * 2,
searchSize * 2);

                        //write the text under the image
                        g.setColor(Color.BLACK);
                        g.clearRect(0,imHeight + 30,imWidth + 10,60); //clear out the old text
                        g.drawString("Current position: " + xloc + ":" + yloc,10,imHeight + 45);
                } else {
                        //write the text under the image
                        g.setColor(Color.BLACK);
                        g.clearRect(0,imHeight + 30,imWidth + 10,60); //clear out the old text
                        g.drawString("Could not locate ball",10,imHeight + 45);
                }
                g.drawString("Processing: " + FPS + " FPS",10,imHeight + 60);
                g.drawString("Camera providing: " + camFrameRate + " FPS",10,imHeight + 75);
                g.drawString("My Score: " + p.getMyScore() + ".  Your score: " +
p.getOppScore(),10,imHeight+90);

        }

        /**
         * Repaint the graphics on a GUI update
         *
         * @param g The graphics object to draw on
         */
        public void update(Graphics g) {
                paint(g);
        }

        /*
         * Listen for window closes.  If one occurs, return all players to their center position
and set them down, then
         * disconnect the controller and exit the program
         */
        class WindowListener extends WindowAdapter {
                public void windowClosing(WindowEvent e) {
                        c.centerAll();
                        c.setAllDown();
                        c.disconnect();
                        System.exit(0);
                }
        }

        /**
         * Constructor for the Track class.
        */
        public Track() {
                c.setAllUp(); //lift all players temporarily
                try {
                        Thread.sleep(250);
                } catch(Exception e) {
                        System.out.println("Thread timing error");
                }
                c.setAllDown(); //set all players back down
                try {
                        Thread.sleep(250);
                } catch(Exception e) {
                        System.out.println("Thread timing error");
                }
                c.startPosition(); //move all rows against the wall
                found = false; //start up without the ball found

                //predefined colors for the ball.  we overwrite these
                rBallColor = 200;
                gBallColor = 200;
                bBallColor = 20;
                xloc = yloc = framecount = 0;

                //set up the webcam controller and grab a new image
                vision1 = new FrameGrabber();
```

```
		vision1.start();

		image1 = vision1.getBufferedImage();
		imHeight = 240;
		imWidth = 320;

		//define the GUI controls and setup
		addWindowListener(new WindowListener());
		setTitle("Foosball Tracking");
		setSize(imWidth + 10,imHeight + 105); //set window size
		setVisible(true); //make the window visible
		addMouseListener (new CalcPixel());

		findBall(image1);

		ActionListener secondElapse = new ActionListener() { //this interrupts every
second
			public void actionPerformed(ActionEvent evt) {
				repaint(); //show the data we have every second
				FPS = framecount;
				camFrameRate = vision1.getFrameCount();
				framecount = 0;
			}
		};
		new Timer(1000, secondElapse).start(); //interrupt every second

		while(true) {
			framecount++;
			if (vision1.imageAvailable()) {
				image1 = vision1.getBufferedImage(); //grab an image
				t.relocate(image1); //find the new human players' locations
				findBall(image1); //find the ball
				if (stage > 2) { //we've located the players
					int controlrow = t.findController(p.getFutureX()); //find
which row is most likely to control the ball next time

					if (p.getFutureX() <= t.getRowXPosition(1,true)) { //ball
behind midfield
						if (c.getRaised(1) == false || c.getRaised(0) ==
false) {

							c.setOffenseUp();
							sleep(); //artificially delay the next
command
						}
					} else if (p.getFutureX() > t.getRowXPosition(0,true)) {
//ball ahead of strikers
						a.strikerIntercept(c,t.getMaxY()-
p.findStrikerPosition(t.getRowSpacing(0), t.getMaxY()) - t.getPlayerHeight());
						if (c.getRaised(0) == true || c.getRaised(1) ==
true) {

							c.setOffenseDown();
							sleep(); //artificially delay the next
command
						}
					} else if (p.getFutureX() <= t.getRowXPosition(0,true)) {
//ball behind strikers
						a.midfieldIntercept(c,t.getMaxY()-
p.findMidfieldPosition(t.getRowSpacing(1), t.getMaxY()) - t.getPlayerHeight());
						if (c.getRaised(0) == false) {
							c.setUp(0);
							sleep(); //artificially delay the next
command
						}
						if (c.getRaised(1) == true) {
							c.setDown(1);
							sleep(); //artificially delay the next
command
						}
					}

					//kick players if they have control of the ball
```

```
                                        if (controlrow == 7 && p.getFutureX() >=
t.getRowXPosition(3,true)) {
                                                c.kick(3);
                                        } else if (controlrow == 6 && p.getFutureX() >=
t.getRowXPosition(2,true)) {
                                                c.kick(2);
                                        } else if (controlrow == 5 && p.getFutureX() >=
t.getRowXPosition(1,true)) {
                                                c.kick(1);
                                        } else if (controlrow == 4 && p.getxPosition() >=
t.getRowXPosition(0,true)) {
                                                c.kick(0);
                                        }

                                        //always move the defense and goalie to intercept
                                        double gi = t.getMaxY()-
p.findGoalIntercept(t.getRowXPosition(3,true))- t.getPlayerHeight();
                                        a.goalIntercept(c,gi);
                                        double di = t.getMaxY()-p.findDefenseIntercept(t.getMinX(),
t.getRowXPosition(2,true), t.getMinY(), t.getMaxY(), t.getRowSpacing(2)) - t.getPlayerHeight();
                                        try {
                                                Thread.sleep(2); //artificially delay the next
command
                                        } catch(Exception e) {
                                                System.out.println("Error");
                                        }
                                        a.defenseIntercept(c,di);
                                }
                        }

                        //sleep for 1 ms to check the next frame
                        try {
                                Thread.sleep(1);
                        } catch(Exception e) {
                                System.out.println("Error");
                        }
                }
        }

        /**
         * Finds the current location of the ball.  Limits the search to a window about the
predicted location
         *
         * @param x The most recent webcam capture
         */
        private void findBall(BufferedImage x) {
                int xpos, ypos, numpos, xlower, xupper, ylower, yupper;

                xpos = ypos = numpos = 0;
                if (found) { //ball was found on last frame, localize the search
                        p.recalculate(); //try to find the new positions
                        xlower = p.getxPosition() - searchSize;
                        xupper = p.getxPosition() + searchSize;
                        ylower = p.getyPosition() - searchSize;
                        yupper = p.getyPosition() + searchSize;
                        if (xlower < t.getMinX()) { //make sure we're only looking on the table
                                xlower = t.getMinX();
                        } else if (xupper >= (t.getMinX() + t.getTableWidth())) {
                                xupper = t.getMinX() + t.getTableWidth();
                        }
                        if (ylower < t.getMinY()) { //make sure we're only looking on the table
                                ylower = t.getMinY();
                        } else if (yupper >= (t.getMinY() + t.getTableHeight())) {
                                yupper = t.getMinY() + t.getTableHeight();
                        }
                } else { //default to searching the entire image space.  computationally expensive
                        xlower = t.getMinX();
                        ylower = t.getMinY();
                        xupper = t.getMinX() + t.getTableWidth();
                        yupper = t.getMinY() + t.getTableHeight();
                }
```

```
                for(int loop1 = xlower; loop1 < xupper; loop1++) {
                        for(int loop2 = ylower; loop2 < yupper; loop2++) {
                                Color pixel = new Color(x.getRGB(loop1,loop2));
                                if (pixel.getRed() > rBallColor-colorSearchSize && pixel.getRed() <
rBallColor+colorSearchSize
                                && pixel.getGreen() > gBallColor-colorSearchSize &&
pixel.getGreen() < gBallColor+colorSearchSize
                                && pixel.getBlue() > bBallColor - colorSearchSize &&
pixel.getBlue() < bBallColor+colorSearchSize){
                                        xpos += loop1;
                                        ypos += loop2;
                                        numpos++; //count the number of pixels in the area near in
the color search space
                                }
                        }
                }
                if (numpos > 0) { //ensure we found the image
                        //calculate the center of mass
                        xloc = xpos / numpos;
                        yloc = ypos / numpos;
                        found = true;
                        p.addPosition(xloc,yloc); //add the new location to the predictor
                } else {
                        if (found) {
                                p.clear(); //clear out old locations in the predictor
                        }
                        if (stage > 3) {
                                p.missedFrame(t.getMinX(), t.getTableWidth()); //see if we need to
register a goal
                        }
                        found = false;
                }
        }

        /**
         * This starts a new instance of our class
         */
        public static void main(String[] args) {
                System.out.println("Click the outline of the table to begin");
                new Track();
        }

        /*
         * This processes mouse events within the window
         */
        private class CalcPixel extends MouseAdapter {
                public void mouseClicked(MouseEvent event) {
                        //grab the x and y location of the click
                        int mousex = event.getPoint().x;
                        int mousey = event.getPoint().y;
                        if (mousex > 10 && mousey > 30 && mousex < (imWidth + 10) && mousey <
(imHeight + 30)) {
                                Color pixel = new Color(image1.getRGB(mousex-10,mousey-30));
                                switch (stage) {
                                case 0: //the user selected the table outline color
                                        t.setColorSearchSize(colorSearchSize * 2);
                                        t.findOutline(image1, mousex-10, mousey-30);
                                        if (t.getTableWidth() > 20 && t.getTableHeight() > 20) {
                                                stage++;
                                                t.calcScale();
                                                a = new AI(t.pixelsPerInch);
                                                System.out.println("Table found.  Please pull all
opponent players as far out as possible and click one to begin.");
                                        } else {
                                                System.out.println("Not a large enough table area
("+pixel.getRed()+","+pixel.getGreen()+","+pixel.getBlue()+")  Click the outline again.");
                                        }
                                        break;
                                case 1: //the user selected the player color
                                        rOppColor = pixel.getRed();
```

```
                                    gOppColor = pixel.getGreen();
                                    bOppColor = pixel.getBlue();
                                    t.setColorSearchSize(colorSearchSize * 2);
                                    t.findRows(image1, mousex-10, mousey-30); //find the
players
                                    System.out.println("Tracking new color.  RGB=" + rOppColor
+ ":" + gOppColor + ":" + bOppColor);
                                    stage+=2;
                                    System.out.println("Players found.  Please click the ball
to begin.");
                                    break;
                                case 2: //null case (antiquated)
                                case 3: ///the user selected the ball color
                                    c.centerAll(); //center all the players after the ball is
found
                                    rBallColor = pixel.getRed();
                                    gBallColor = pixel.getGreen();
                                    bBallColor = pixel.getBlue();
                                    System.out.println("Tracking new color.  RGB=" + rBallColor
+ ":" + gBallColor + ":" + bBallColor);
                                    p.setColorSearchSize(colorSearchSize);
                                    p.calcBallSize(image1, rBallColor, gBallColor, bBallColor,
mousex-10, mousey-30, searchSize);
                                    stage++;
                                    break;
                                default:
                                    //the default behavior is just to track a new ball color
                                    rBallColor = pixel.getRed();
                                    gBallColor = pixel.getGreen();
                                    bBallColor = pixel.getBlue();
                                    System.out.println("Tracking new color.  RGB=" + rBallColor
+ ":" + gBallColor + ":" + bBallColor);
                                }
                            }
                        }
                    }
                }
```

```
/**
 * WriteS.java
 * This class is used to communicate with the motor control board across the serial connection on
the computer.  It communicates at 115,200 bps,
 * 1 stop bit, 8 data bits, no parity bits, and no flow control.
 *
 * @author Team FIFA, ECE4884L01, Georgia Institute of Technology
 * @version 1.0, December 2007
*/

import java.io.*;
import java.util.*;
import javax.comm.*;
import java.awt.event.*;
import java.lang.String;
import java.lang.Character;
import java.lang.Math;
import javax.swing.Timer;

public class WriteS {
	static Enumeration portList;
	static CommPortIdentifier portId;
	static SerialPort serialPort;
	static OutputStream outputStream;
	static boolean outputBufferEmptyFlag = false;
	private boolean[] raised = new boolean[4];
	private boolean[] nextval = new boolean[4];
	private int[] timeleft = new int[4];
	private int[] packet = {1,0,2,2,2,2,2,2,2,2,2,2,2,2,2,2}; //initialize packet, 2=not sets

	/**
	 * Class constructor
	 */
	public WriteS() {
		connect(); //connect to the motor control board
		for (int i = 0; i < 4; i++) { //loop through all rows, assuming all players are
down and have no future agenda
			timeleft[i] = 0;
			raised[i] = false;
			nextval[i] = false;
		}

		//this actionlistener is used in a state-variable configuration to tell the PWM
servos what to do "next" after
		//they complete their current action
		ActionListener checkservos = new ActionListener() {
			public void actionPerformed(ActionEvent evt) {
				for (int i = 0; i < 4; i++) {
					if (timeleft[i] > 1) { //there is still time left on the
last action
						timeleft[i] = timeleft[i] - 1; //decrement the timer
					} else if (timeleft[i] == 1) { //time has expired
						timeleft[i] = 0;
						raised[i] = ! raised[i];
						if (nextval[i] != raised[i]) { //perform the next
action, if applicable
							setLiftID(i,nextval[i]);
						}
					}
				}
			}
		};
		new Timer(10, checkservos).start(); //interrupt every 10ms
	}

	/**
	 * Returns the kick status of a PWM servo
	 *
	 * @param ID The ID of the PWM servo (0-3)
	 * @return A boolean indicating if the PWM is in the "up" position
	 */
```

```
	public boolean getRaised(int ID) {
		return raised[ID];
	}

	/**
	 * Special command to the servo to set all rows in their initial position, against the
wall
	 *
	 * @see #centerAll
	 */
	public void startPosition() {
		//Command packet: [10100000 00000010]
		packet[2] = 1;
		for (int i = 3; i <= 13; i++) {
			packet[i]=0;
		}
		packet[14] = 1;
		packet[15] = 0;
		sendPacket();
	}

	/**
	 * Special command to the servo to set all PWM servos to the "up" position
	 *
	 * @see #setAllDown
	 * @see #setOffenseDown
	 * @see #setOffenseUp
	 */
	public void setAllUp() {
		//Command packet: [10100000 00000011]
		nextval[0] = nextval[1] = nextval[2] = nextval[3] = raised[0] = raised[1] =
raised[2] = raised[3] = true;
		timeleft[0] = 5;
		timeleft[1] = 5;
		timeleft[2] = 5;
		timeleft[3] = 10;
		packet[2] = 1;
		for (int i = 3; i <= 13; i++) {
			packet[i]=0;
		}
		packet[14] = 1;
		packet[15] = 1;
		sendPacket();
	}

	/**
	 * Special command to the servo to set just the midfield and striker PWM servos to the
"up" position
	 *
	 * @see #setAllUp
	 * @see #setAllDown
	 * @see #setOffenseDown
	 */
	public void setOffenseUp() {
		//Command packet: [10100000 00000101]
		nextval[0] = nextval[1] = raised[0] = raised[1] = true;
		timeleft[0] = 5;
		timeleft[1] = 5;
		packet[2] = 1;
		for (int i = 3; i <= 12; i++) {
			packet[i]=0;
		}
		packet[13] = 1;
		packet[14] = 0;
		packet[15] = 1;
		sendPacket();
	}

	/**
	 * Special command to the servo to set the midfield and striker PWM servos to the "down"
position
```

```
         *
         * @see #setAllUp
         * @see #setAllDown
         * @see #setOffenseUp
         */
        public void setOffenseDown() {
                //Command packet: [10100000 00000110]
                nextval[0] = nextval[1] = raised[0] = raised[1] = false;
                timeleft[0] = 5;
                timeleft[1] = 5;
                packet[2] = 1;
                for (int i = 3; i <= 12; i++) {
                        packet[i]=0;
                }
                packet[13] = 1;
                packet[14] = 1;
                packet[15] = 0;
                sendPacket();
        }

        /**
         * Special command to the servo to set all PWM servos to the "down" position
         *
         * @see #setAllUp
         * @see #setOffenseDown
         * @see #setOffenseUp
         */
        public void setAllDown() {
                //Command packet: [10100000 00000100]
                nextval[0] = nextval[1] = nextval[2] = nextval[3] = raised[0] = raised[1] =
raised[2] = raised[3] = false;
                timeleft[0] = timeleft[1] = timeleft[2] = 5;
                timeleft[3] = 10;
                packet[2] = 1;
                for (int i = 3; i <= 12; i++) {
                        packet[i]=0;
                }
                packet[13] = 1;
                packet[14] = 0;
                packet[15] = 0;
                sendPacket();
        }

        /**
         * Special command to the servo to set all rows to a centered position
         *
         * @see #startPosition
         */
        public void centerAll() {
                //Command packet: [10100000 00000001]
                packet[2] = 1;
                for (int i = 3; i <= 14; i++) {
                        packet[i]=0;
                }
                packet[15] = 1;
                sendPacket();
        }

        /**
         * Moves a row to a specified hexidecimal position
         *
         * @param ID The row number of the servo to move
         * @param pos The hexidecimal position the servo. Should be to between 0x000 and 0x3FF,
where the position is the same as that used
         *                                          internally by the servo
         */
        public void moveHex(int ID, int pos) {
                packet[2]=0;
                packet[3]=0;
                packet[4]=ID/2;
                packet[5]=ID%2;
```

```
		for (int i = 0; i <= 9; i++) {
			packet[i+6]=pos/((int)Math.pow(2,9-i));
			pos=pos%((int)Math.pow(2,9-i));
		}
		sendPacket();
	}

	/**
	 * Public accessor indicating whether a PWM servo currently is in limbo, i.e. it is
executing a previous command
	 *
	 * @param ID The row number of the servo
	 * @return A boolean indicating if the determined servo is currently moving
	 */
	public boolean inLimbo(int ID) {
		return (timeleft[ID] > 0);
	}

	/**
	 * Moves a row to a specified physical position.	This function internally converts the
inch position to a servo position
	 *
	 * @param ID The row number of the servo to move
	 * @param loc The number of inches away from the wall the row's stopper should end at
	 */
	public void move(int ID, double loc) {
		packet[2]=0;
		packet[3]=0;
		packet[4]=ID/2;
		packet[5]=ID%2;

		//convert pos from physical position to servo position
		double ipp = 11.0*5.0/6.0/1024.0;	//inches per postional bit
		int maxPos = 0;
		int start_pos = 0;	//starting position of servo, varies by handle

		//we have to avoid over-pulling or over-pushing the servos past the physical
bounds of the table.  if we do that,
		//the servos will go into a torque-overload setting, and shut down.
		switch (ID) {
			case 3:
				start_pos = 0xAF;
				maxPos = 0x350;
				break;
			case 2:
				start_pos = 0;
				maxPos = 0x3FF;
				break;
			case 1:
				start_pos = 0x130;
				maxPos = 0x2B0;
				break;
			case 0:
				start_pos = 0x8F;
				maxPos = 0x350;
				break;
		}
		int pos = (int)(loc/ipp) + start_pos;
		if (pos > maxPos) {
			pos = maxPos;
		} else if (pos < start_pos) {
			pos = start_pos;
		}

		//while the goalie does have specific physical bounds, we want to software limit
it as well, so as to avoid
		//going beyond the goal range.
		if (ID == 3 && pos == start_pos) {
			pos = 0x11E; //the physical servo location of the minimum goal value
		} else if (ID == 3 && pos == maxPos) {
			pos = 0x2E1; //the physical servo location of the maximum goal value
```

```
                }

                //convert the positional data into a packet
                if (pos <= maxPos && pos >= start_pos) {
                        for (int i = 0; i <= 9; i++) {
                                packet[i+6]=pos/((int)Math.pow(2,9-i));
                                pos=pos%((int)Math.pow(2,9-i));
                        }
                        sendPacket();
                }
        }

        /**
         * Sets a PWM servo in either the "up" or "down" position
         *
         * @param ID The row number of the servo to move
         * @param value A boolean indicating if the servo should be "up"
         */
        public void setLiftID(int ID, boolean value) {
                if (value) {
                        setUp(ID);
                } else {
                        setDown(ID);
                }
        }

        /**
         * Initiates a temporal function where the PWM servo first is lifted, and after a
timeout, is set back down
         *
         * @param ID The row number of the servo to kick
         */
        public void kick(int ID) {
                if (timeleft[ID] == 0) {
                        setUp(ID);
                        nextval[ID] = false;
                }
        }

        /**
         * Initiates a "kick and move" function.     The PWM servo will be kicked at the current
position and as the lateral movement begins
         *
         * @param ID The row number to move and kick
         * @param pos The number of inches away from the wall the row's stopper should end at
         */
        public void kickhere(int ID, double pos) {
                move(ID, pos);
                kick(ID);
        }

        /**
         * Sets a PWM servo in to the "up" position
         *
         * @param ID The row number of the servo to move
         * @see #setDown
         */
        public void setUp(int ID) {
                nextval[ID] = true;
                if (! raised[ID]) {
                        if (ID == 3) {
                                timeleft[ID] = 10; //the goalie is a different (slower) servo
                        } else {
                                timeleft[ID] = 5; //default servo behavior
                        }
                        packet[2]=0;
                        packet[3]=1;
                        packet[4]=ID/2;
                        packet[5]=ID%2;
                        for (int i = 0; i <= 8; i++) {
                                packet[i+6]=1;
```

```
                        }
                        packet[15] = 1;
                        sendPacket();
                }
        }

        /**
         * Sets a PWM servo in to the "down" position
         *
         * @param ID The row number of the servo to move
         * @see #setUp
         */
        public void setDown(int ID) {
                nextval[ID] = false;
                if (raised[ID]) {
                        if (ID == 3) {
                                timeleft[ID] = 10; //the goalie is a different (slower) servo
                        } else {
                                timeleft[ID] = 5; //default servo behavior
                        }
                        packet[2]=0;
                        packet[3]=1;
                        packet[4]=ID/2;
                        packet[5]=ID%2;
                        for (int i = 0; i <= 9; i++) {
                                packet[i+6]=0;
                        }
                        sendPacket();
                }
        }

        /**
         * Closes the serial port
         */
        public void disconnect() {
                serialPort.close();
        }

        /**
         * Connects the serial port to the motor control board at 115,200 bps, 8 data
bits/packet, 1 stop bit, no parity, no flow control.
         */
        public void connect() {
                int defaultBaudRate = 115200;
                int defaultDatabits = SerialPort.DATABITS_8;
                int defaultStopbits = SerialPort.STOPBITS_1;
                int defaultParity = SerialPort.PARITY_NONE;
                boolean portFound = false;
                String defaultPort = "/dev/ttyS0"; //open serial port 0 in linux
                portList = CommPortIdentifier.getPortIdentifiers();
                while (portList.hasMoreElements()) {
                        portId = (CommPortIdentifier) portList.nextElement();
                        if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
                                if (portId.getName().equals(defaultPort)) {
                                        System.out.println("Found port " + defaultPort);
                                        portFound = true;

                                        //try to open the serial port
                                        try {
                                                serialPort = (SerialPort) portId.open("Foosball",
2000);
                                        } catch (PortInUseException e) {
                                                System.out.println("Port in use.");
                                                continue;
                                        }

                                        try {
                                                outputStream = serialPort.getOutputStream();
                                        } catch (IOException e) {}
```

```
                                        //there is a weird bug in the version of java that was
used, limiting the setSerialPortParams command
                                        //the workaround is to use up a little time internally
while we wait for the stream to settle.
                                        //if the stream does not settle in time, we attempt to re-
initiate the connection a second time below.
                                        //the workaround is done via the "System.out.print()"
commands, sending null characters to the command line
                                        try {
                                                System.out.print("");
                                                serialPort.setSerialPortParams(defaultBaudRate,
defaultDatabits, defaultStopbits, defaultParity);
                                                System.out.print("");
                                        } catch (Exception e) {}
                                        try {
                                                System.out.print("");
                                                serialPort.setSerialPortParams(defaultBaudRate,
defaultDatabits, defaultStopbits, defaultParity);
                                                System.out.print("");
                                        } catch (Exception e) {}

                                        try {

        serialPort.setFlowControlMode(SerialPort.FLOWCONTROL_NONE);
                                        } catch (UnsupportedCommOperationException e) {}

                                        try {
                                                serialPort.notifyOnOutputEmpty(true);
                                        } catch (Exception e) {
                                        System.out.println("Error setting event notification");
                                        System.out.println(e.toString());
                                        System.exit(-1);
                                        }
                                }
                        }
                }
                if (!portFound) {
                                System.out.println("port " + defaultPort + " not found.");
                }
        }

        /**
         * Sends a packet across the serial port.    The packet is a set of bits in the packet[]
array, consisting of 2-bytes.
         */
        public void sendPacket() {
                char packet_char = 0;
                byte[] packet_bytes =new byte[2];
                char i, shift_packet;

                //construct the first byte
                packet_char = 0;
                for (i=0; i < 8; i++) {
                        shift_packet = (char)packet[i];
                        shift_packet <<= (7-i);
                        packet_char |= shift_packet;
                }
                packet_bytes[0] = (byte)(packet_char & 0xFF); //convert from unicode

                //construct the second byte
                packet_char = 0;
                for (i=8; i < 16; i++) {
                        shift_packet = (char)packet[i];
                        shift_packet <<= (15-i);
                        packet_char |= shift_packet;
                }
                packet_bytes[1] = (byte)(packet_char & 0xFF); //convert from unicode
                try {
                        outputStream.write(packet_bytes);
                } catch (IOException e) {
                        System.out.println("SERIAL ERROR");
```

```
                    }
                }
}
```

Appendix B

PIC18F4520 Servo Controller
Main Program and Library Source Code

```
/*******************************************************************************
 *
 * Team FIFA Servo Controller Program
 * File: servo.c
 *
 *******************************************************************************
 *
 * Developed By: Michael Aeberhard (michael.aeberhard@gatech.edu)
 * Date: October 26th, 2007
 * Purpose: This program reads in data packets from the PC and carries out their
 * instructions to the servo motors that ultimatley manipulate the handles on
 * the foosball table.
 *
 *******************************************************************************/

#pragma config OSC = HSPLL, WDT = OFF, LVP = OFF

#include <p18f452.h>          //Use PIC184520 naming conventions
#include <timers.h>                   //Timer functions for the PIC18F
#include <ax12.h>                     //Functions for AX12 servos
#include <pc_uart.h>          //Functions for PC UART communication

//PIC macros
#define EnableInterrupts      INTCONbits.GIEH = 1;
#define DisableInterrupts     INTCONbits.GIEH = 0;

//Servo motor IDs for Foosball table
#define SERVO_ALL                     0xFE
#define SERVO_KEEPER          0x04
#define SERVO_DEFENSE         0x03
#define SERVO_MIDFIELD        0x02
#define SERVO_OFFENSE         0x01
#define SERVO_TEST                    0x01

//PWM Servo Macros
#define PWM_OFFENSE_CTRL      PORTDbits.RD4
#define PWM_OFFENSE_DIR               PORTDbits.RD5
#define PWM_MIDFIELD_CTRL     PORTBbits.RB5
#define PWM_MIDFIELD_DIR      PORTBbits.RB4
#define PWM_DEFENSE_CTRL      PORTDbits.RD6
#define PWM_DEFENSE_DIR               PORTDbits.RD7
#define PWM_KEEPER_CTRL               PORTEbits.RE0
#define PWM_KEEPER_DIR        PORTEbits.RE1

//Port macros
#define AX_DATA_DIRECTION     PORTCbits.RC0
#define LED0                          PORTAbits.RA0
#define LED1                          PORTAbits.RA1
#define PButton1                      PORTAbits.RA4
#define PButton2                      PORTBbits.RB2

//Assignment macros
#define AX_DATA_RX                    0
#define AX_DATA_TX                    1

//function prototypes
void delay_1s(void);
void delay_5ms(void);
void servoCommand(void);
void specialCommand(void);

#pragma code

//Global Variables
unsigned char pc_buffer[2];
unsigned char pc_index;
unsigned char packetReady;
unsigned char packet[2];
unsigned char numberPackets;

char center[3] = { 0x1E, 0x00, 0x02 };
```

```
//AX12 Start Position
char offenseStart[3] = { 0x1E, 0x8F, 0x00};
char midfieldStart[3] = { 0x1E, 0x30, 0x01 };
char defenseStart[3] = { 0x1E, 0x00, 0x00 };
char keeperStart[3] = { 0x1E, 0xAF, 0x00 };

char offenseEnd[3] = { 0x1E, 0xBF, 0x00};
char midfieldEnd[3] = { 0x1E, 0x30, 0x01 };
char defenseEnd[3] = { 0x1E, 0xFF, 0x03 };
char keeperEnd[3] = { 0x1E, 0xAF, 0x00 };


/*****************************************************************************
 * Begin main program code
 *****************************************************************************/

void main() {
        char movingSpeed[3] = { 0x20, 0xFF, 0x03 };
        char maxTorque[3] = { 0x22, 0xFF, 0x03 };
        unsigned char PCread[2] = { 0, 0 };
        int i;

        pc_index = 0;
        pc_buffer[0] = 0;
        pc_buffer[1] = 0;
        packetReady = 0;
        packet[0] = 0;
        packet[1] = 0;
        numberPackets = 0;

        //Setup Control Registers
        ADCON1 = 0x07;                //Set all A/D ports as digital I/O
        LATA = 0;
        LATB = 0;
        LATE = 0;
        PORTA = 0;
        PORTB = 0;
        PORTE = 0;
        TRISA = 0b00010000;           //Set RA4 as input, all others output
        TRISB = 0b00000101;           //Input for RB0 and RB2
        TRISC = 0b11000110;           //Set bit 7 and 6 of PortC as inputs, others as outputs
        TRISD = 0b00000000;           //Set all bits of PortD as outputs
        TRISE = 0b00000000;

        //Timer2 Registers Prescaler = 16
        //TMR2 PostScaler = 16 - PR2 = 255
        //Freq = 200 Hz - Period = 4992 microseconds
        OpenTimer2(T2_POST_1_16 & T2_PS_1_16);
        PR2 = 195;
        TMR2 = 0;

        //SPI Setup for RS-232/UART communication
        PC_SetupSPI();
        PC_UART_SELECT = 1;
        PC_UART_SHUTDN = 1;

        //Setup internal USART for servo communication
        AX_SetupUSART();

        LED0 = 0;
        LED1 = 0;

        //Initialize the PWMs
        PWM_OFFENSE_DIR = 0;
        PWM_OFFENSE_CTRL = 0;
        PWM_MIDFIELD_DIR = 0;
        PWM_MIDFIELD_CTRL = 0;
        PWM_DEFENSE_DIR = 0;
        PWM_DEFENSE_CTRL = 0;
        PWM_KEEPER_DIR = 0;
```

```
	PWM_KEEPER_CTRL = 0;

	AX_DATA_DIRECTION = AX_DATA_TX;
	//Set fast moving speed
	AX_TxPacket(SERVO_OFFENSE, I_WRITE_DATA, movingSpeed, 3);
	delay_5ms();
	delay_5ms();
	//Set high torque
	AX_TxPacket(SERVO_OFFENSE, I_WRITE_DATA, maxTorque, 3);
	delay_5ms();

	//Set fast moving speed
	AX_TxPacket(SERVO_MIDFIELD, I_WRITE_DATA, movingSpeed, 3);
	delay_5ms();
	delay_5ms();
	//Set high torque
	AX_TxPacket(SERVO_MIDFIELD, I_WRITE_DATA, maxTorque, 3);
	delay_5ms();

	//Set fast moving speed
	AX_TxPacket(SERVO_DEFENSE, I_WRITE_DATA, movingSpeed, 3);
	delay_5ms();
	delay_5ms();
	//Set high torque
	AX_TxPacket(SERVO_DEFENSE, I_WRITE_DATA, maxTorque, 3);
	delay_5ms();

	//Set fast moving speed
	AX_TxPacket(SERVO_KEEPER, I_WRITE_DATA, movingSpeed, 3);
	delay_5ms();
	delay_5ms();
	//Set high torque
	AX_TxPacket(SERVO_KEEPER, I_WRITE_DATA, maxTorque, 3);
	delay_5ms();

	while (1) {
		LED1 = !LED1;
		LED0 = 0;
		PB1 = PButton1;
		PB2 = PButton2;

		if (PORTBbits.RB0 == 0) {
				rxData = PC_ReadData();
				if (pc_index == 0) {
					if (rxData & 0x80) {
						pc_buffer[0] = rxData;
						pc_index = 1;
					}
				} else {
					pc_buffer[1] = rxData;
					pc_index = 0;
					packet[0] = pc_buffer[0];
					packet[1] = pc_buffer[1];
					packetReady = 1;
				}
			LED0 = 1;
		}

		if (packetReady == 1) {
			if ((packet[0] & 0x20) == 0x20) {		//Check if its a special
command
				specialCommand();
			} else {
				servoCommand();
			}
			packetReady = 0;
		}
	}
}
```

```
/*****************************************************************************
 * specialCommand()
 *
 * Parses a special command packet and carries out the necessary servo action.
 *
 * Parameters: none
 * Returns: void
 *****************************************************************************/
void specialCommand() {
	if (packet[1] == 0x01) {		//center all AX12s
		AX_TxPacket(SERVO_ALL, I_WRITE_DATA, center, 3);
	} else if (packet[1] == 0x02) {		//start position all AX12s
		AX_TxPacket(SERVO_OFFENSE, I_WRITE_DATA, offenseStart, 3);
		AX_TxPacket(SERVO_MIDFIELD, I_WRITE_DATA, midfieldStart, 3);
		AX_TxPacket(SERVO_DEFENSE, I_WRITE_DATA, defenseStart, 3);
		AX_TxPacket(SERVO_KEEPER, I_WRITE_DATA, keeperStart, 3);
	} else if (packet[1] == 0x03) {		//kick all
		PWM_KEEPER_CTRL = 1;
		PWM_DEFENSE_CTRL = 1;
		PWM_MIDFIELD_CTRL = 1;
		PWM_OFFENSE_CTRL = 1;
	} else if (packet[1] == 0x04) {		//idle all
		PWM_KEEPER_CTRL = 0;
		PWM_DEFENSE_CTRL = 0;
		PWM_MIDFIELD_CTRL = 0;
		PWM_OFFENSE_CTRL = 0;
	} else if (packet[1] == 0x05) {	//kick offense/midfield
		PWM_MIDFIELD_CTRL = 1;
		PWM_OFFENSE_CTRL = 1;
	} else if (packet[1] == 0x06) {	//idle offense/midfield
		PWM_MIDFIELD_CTRL = 0;
		PWM_OFFENSE_CTRL = 0;
	}
}

/*****************************************************************************
 * servoCommand()
 *
 * Parses a servo command packet and carries out the necessary servo action.
 *
 * Parameters: none
 * Returns: void
 *****************************************************************************/
void servoCommand() {
	unsigned char servoAddress = (char) (packet[0] & 0x0C) >> 2;
	unsigned char servoType = (packet[0] & 0x10) >> 4;
	unsigned char servoValue[2] = { (packet[0] & 0x03) , (packet[1] & 0xFF) };
	unsigned char control, direction;
	unsigned char data[3];

	if (servoType == 1) {		//PWM servo
		control = servoValue[0] & 0x01;
		direction = (servoValue[0] & 0x02) >> 1;

		if (servoAddress == 3) {
			if (control == 1) {
				PWM_KEEPER_CTRL = 1;
			} else {
				PWM_KEEPER_CTRL = 0;
			}
			if (direction == 1) {
				PWM_KEEPER_DIR = 1;
			} else {
				PWM_KEEPER_DIR = 0;
			}
		} else if (servoAddress == 2) {
			if (control == 1) {
				PWM_DEFENSE_CTRL = 1;
			} else {
				PWM_DEFENSE_CTRL = 0;
			}
```

```
                        if (direction == 1) {
                                PWM_DEFENSE_DIR = 1;
                        } else {
                                PWM_DEFENSE_DIR = 0;
                        }
                } else if (servoAddress == 1) {
                        if (control == 1) {
                                PWM_MIDFIELD_CTRL = 1;
                        } else {
                                PWM_MIDFIELD_CTRL = 0;
                        }
                        if (direction == 1) {
                                PWM_MIDFIELD_DIR = 1;
                        } else {
                                PWM_MIDFIELD_DIR = 0;
                        }
                } else {
                        if (control == 1) {
                                PWM_OFFENSE_CTRL = 1;
                        } else {
                                PWM_OFFENSE_CTRL = 0;
                        }
                        if (direction == 1) {
                                PWM_OFFENSE_DIR = 1;
                        } else {
                                PWM_OFFENSE_DIR = 0;
                        }
                }
        } else {                                        //AX-12 Servo
                data[0] = 0x1E;
                data[1] = servoValue[1];
                data[2] = servoValue[0];
                if (servoAddress == 3) {
                        AX_TxPacket(SERVO_KEEPER, I_WRITE_DATA, data, 3);
                } else if (servoAddress == 2) {
                        AX_TxPacket(SERVO_DEFENSE, I_WRITE_DATA, data, 3);
                } else if (servoAddress == 1) {
                        AX_TxPacket(SERVO_MIDFIELD, I_WRITE_DATA, data, 3);
                } else {
                        AX_TxPacket(SERVO_OFFENSE, I_WRITE_DATA, data, 3);
                }
        }
}

/*******************************************************************************
 * delay_1s()
 *
 * Delays for 1 second.
 *
 * Parameters: none
 * Returns: void
 *******************************************************************************/
void delay_1s() {
        unsigned char delayCount;
        delayCount = 200;
        while (delayCount > 0) {
                delay_5ms();
                delayCount--;
        }
}


/*******************************************************************************
 * delay_5ms()
 *
 * Delays for 5 milliseconds.
 *
 * Parameters: none
 * Returns: void
 *******************************************************************************/
void delay_5ms() {
```

```
        while (PIR1bits.TMR2IF == 0) {
                ;
        }
        PIR1bits.TMR2IF = 0;
        TMR2 = 0;
}
```

```
#ifndef __AX12_H
#define __AX12_H

/*********************************************************************************
 *
 * Dynamixel AX-12 Servo Control Library
 * Filename: ax12.h
 *
 * THIS IS THE HEADER FILE
 *
 * Instructions: Include this file in your main program source code, and define
 * the following header directives, depending on which serial line is used to
 * interface with the AX12 data line.
 *
 *
 *********************************************************************************
 *
 * Developed By: Michael Aeberhard (michael.aeberhard@gatech.edu)
 * Date: October 5th, 2007
 * Purpose: Easy-to-use function for controlling the AX-12 servos.
 *
 * Restrictions: This library assumes the use of the PIC18F4520 microcontroller
 * from Microchip along with the C18 Compiler.
 *
 *********************************************************************************
 *
 * This file contains numerous functions that can be used to control a AX-12
 * servo motor.  Some of the functions are more generic, while others are much
 * more specific.  The original purpose of this library was to write a set of
 * useful functions to be used in the automated foosball table senior design
 * project by Michael Aeberhard, Shane Connelly, Evan Tarr, and Nardis Walker
 * at the Georiga Institute of Technology.
 *
 *********************************************************************************/

#include <p18f452.h>          //Use PIC184520 naming conventions

//Servo motor control table addresses
//EEPROM
#define CT_ID                       0x03
#define CT_BAUD_RATE          0x04
#define CT_RETURN_DELAY             0x05
#define CT_CW_ANGLELIMIT_L    0x06
#define CT_CW_ANGLELIMIT_H    0x07
#define CT_CCW_ANGLELIMIT_L   0x08
#define CT_CCW_ANGLELIMIT_H   0x09
#define CT_MAX_TORQUE_L             0x0E
#define CT_MAX_TORQUE_H             0x0F
#define CT_STATUS_RETURN      0x10
#define CT_ALARM_LED          0x11
//RAM
#define CT_TORQUE_ENABLE      0x18
#define CT_GOAL_POSITION_L    0x1E
#define CT_GOAL_POSITION_H    0x1F
#define CT_MOVING_SPEED_L     0x20
#define CT_MOVING_SPEED_H     0x21
#define CT_PRESENT_POS_L      0x24
#define CT_PRESENT_POS_H      0x25
#define CT_PRESENT_SPEED_L    0x26
#define CT_PRESENT_SPEED_H    0x27
#define CT_PRESENT_LOAD_L     0x28
#define CT_PRESENT_LOAD_H     0x29
#define CT_PRESENT_VOLTAGE    0x2A
#define CT_REGISTERED_INST    0x2C
#define CT_MOVING                   0x2E

//Servo Motor Instruction Set
#define I_PING                      0x01
#define I_READ_DATA                 0x02
#define I_WRITE_DATA          0x03
#define I_REG_WRITE                 0x04
```

```
#define I_ACTION                    0x05
#define I_RESET                             0x06
#define I_SYNC_WRITE          0x07

//Function prototypes for all of the function in the library
//Please refer to the ax12.c file for instruction on how to use these functions
void AX_SetId(char, char);
void AX_Ping(char);
char* AX_RxPacket(void);
char AX_RxByte(void);
char AX_ByteRdy();
void AX_TxPacket(char, char, char*, char);
void AX_SendByte(char);
void AX_SetupUSART(void);

#endif
```

```
/********************************************************************************
 *
 * Dynamixel AX-12 Servo Control Library
 * Filename: ax12.c
 *
 * THIS IS THE CODE FILE
 *
 ********************************************************************************
 *
 * Developed By: Michael Aeberhard (michael.aeberhard@gatech.edu)
 * Date: October 5th, 2007
 * Purpose: Easy-to-use function for controlling the AX-12 servos.
 *
 * Restrictions: This library assumes the use of the PIC18F4520 microcontroller
 * from Microchip along with the C18 Compiler.
 *
 * Instructions for changing from USART to SPI.
 *
 ********************************************************************************
 *
 * This file contains numerous functions that can be used to control a AX-12
 * servo motor.  Some of the functions are more generic, while others are much
 * more specific.  The original purpose of this library was to write a set of
 * useful functions to be used in the automated foosball table senior design
 * project by Michael Aeberhard, Shane Connelly, Evan Tarr, and Nardis Walker
 * at the Georiga Institute of Technology.
 *
 ********************************************************************************/

#include <p18f452.h>          //Use PIC184520 naming conventions
#include <usart.h>                    //Functions for the on-chip EUSART
#include <ax12.h>                     //Prototype declarations for AX12 library

/********************************************************************************
 * AX_SetId()
 *
 * Changes the ID of an AX12 servo to a new value.
 *
 * Parameters:
 *             oldid (char) - the old id # of the AX12 servo
 *             newid (char) - the new id # to which to change to
 * Returns: void
 ********************************************************************************/
void AX_SetId(char oldid, char newid) {
       char parameters[2], paramLength;
       parameters[0] = CT_ID;
       parameters[1] = newid;
       paramLength = 2;
       AX_TxPacket(oldid, I_WRITE_DATA, parameters, paramLength);
} //end AX_SetId()


/********************************************************************************
 * AX_Ping()
 *
 * Pings an AX12 servo for a status packet.
 *
 * Parameters:
 *             id (char) - the id of the AX12 to ping
 * Returns: void
 ********************************************************************************/
void AX_Ping(char id) {
       AX_TxPacket(id, I_PING, 0, 0);
} //end AX_Ping()


/********************************************************************************
 * AX_RxPacket()
 *
 * Receives a packet from an AX12 servo.
 *
```

```
 * Parameters: none
 * Returns:
 *             char* - pointer to the retrieved string
 ******************************************************************************/
char* AX_RxPacket() {
        //char length, id, errorByte, checksum, i, curByte;
        char i, length;
        char packet[32];
        packet[0] = AX_RxByte();      //0xFF
        packet[1] = AX_RxByte();      //0xFF
        packet[2] = AX_RxByte();      //length
        packet[3] = AX_RxByte();      //error
        length = packet[3] + 3;
        for (i = 3; i < length; i++) {
                packet[i] = AX_RxByte();
        }
        return packet;
} //end AX_RxPacket()


/******************************************************************************
 * AX_RxByte()
 *
 * Receives a byte from an AX12 servo data line.
 *
 * Parameters: none
 * Returns: (char) the received byte
 ******************************************************************************/
char AX_RxByte() {
        while (DataRdyUSART() == 0) {
                ;
        }
        return ReadUSART();
} //end AX_RxByte()


/******************************************************************************
 * AX_RxByteRdy()
 *
 * Determines if a byte from an AX12 servo is ready to be read.
 *
 * Parameters: none
 * Returns: (char) 1 if byte is ready, 0 if not
 ******************************************************************************/
char AX_ByteRdy() {
        return DataRdyUSART();
} //end AX_ByteRdy()


/******************************************************************************
 * AX_TxPacket()
 *
 * Transmits a packet on the AX12 servo data line.
 *
 * Parameters:
 *             id (char) - the ID of the AX12 which should receive the packet
 *             instruction (char) - the instruction type of the packet
 *             parameters (char*) - data array of the packet
 *             paramLength (char) - length of the data array
 * Returns: void
 ******************************************************************************/
void AX_TxPacket(char id, char instruction, char *parameters, char paramLength) {
        char length, i, checksum;
        length = 2 + paramLength;
        checksum = 0;
        AX_SendByte(0xFF);            //required by AX12 protocol to initiate a packet
        AX_SendByte(0xFF);            //required by AX12 protocol to initiate a packet
        AX_SendByte(id);
        checksum = checksum + id;
        AX_SendByte(length);
        checksum = checksum + length;
```

```
        AX_SendByte(instruction);
        checksum = checksum + instruction;
        for (i = 0; i < paramLength; i++) {
                AX_SendByte(parameters[i]);
                checksum = checksum + parameters[i];
        }
        checksum = ~checksum;
        AX_SendByte(checksum);
        while(BusyUSART()) {          //wait for last byte to be sent
        }
} //end AX_TxPacket()


/*******************************************************************************
 * AX_SendByte()
 *
 * Sends a single byte onto the AX12 data line.
 *
 * Parameters:
 *             toSend (char) - byte to send onto the data line
 * Returns: void
 ******************************************************************************/
void AX_SendByte(char toSend) {
        while(BusyUSART()) {          //wait for transmit register to be ready
        }
        WriteUSART(toSend);
} //AX_SendByte()


/*******************************************************************************
 * AX_SetupUSART()
 *
 * Setup the PIC USART to transmit at 1 MBit/s.
 *
 * Parameters:
 *             id (char) - the ID of the AX12 which should receive the packet
 *             instruction (char) - the instruction type of the packet
 *             parameters (char*) - data array of the packet
 *             paramLength (char) - length of the data array
 * Returns: void
 ******************************************************************************/
void AX_SetupUSART() {
        OpenUSART(USART_TX_INT_OFF & USART_RX_INT_OFF & USART_BRGH_HIGH & USART_EIGHT_BIT &
USART_ASYNCH_MODE, 9);
        TXSTAbits.BRGH = 1;    //500 kbps
        SPBRG = 0x04;
} //end AX_SetupUSART()
```

```
#ifndef __PC_UART_H
#define __PC_UART_H

/*****************************************************************************
 *
 * PC SPI UART with MAXIM 3100 Library
 * Filename: pc_uart.h
 *
 * THIS IS THE HEADER FILE
 *
 *****************************************************************************
 *
 * Developed By: Michael Aeberhard (michael.aeberhard@gatech.edu)
 * Date: October 11th, 2007
 * Purpose: Easy-to-use functions sending and receiving data via the MAXIM 3100
 * UART chip using the PIC's onboard SPI module.
 *
 * Restrictions: This library assumes the use of the PIC18F4520 microcontroller
 * from Microchip along with the C18 Compiler.
 *
 *****************************************************************************
 *
 * This file contains numerous functions that can be used to communicate with
 * a PC.  Some of the functions are more generic, while others are much
 * more specific.  The original purpose of this library was to write a set of
 * useful functions to be used in the automated foosball table senior design
 * project by Michael Aeberhard, Shane Connelly, Evan Tarr, and Nardis Walker
 * at the Georiga Institute of Technology.
 *
 *****************************************************************************/

#include <p18f452.h>          //Use PIC184520 naming conventions

//PIC macros
#define EnableInterrupts      INTCONbits.GIEH = 1;
#define DisableInterrupts     INTCONbits.GIEH = 0;

//MAX3100 Pin assignments to the PIC
#define PC_UART_SELECT        PORTDbits.RD0
#define PC_UART_IRQ                   PORTBbits.RB0
#define PC_UART_SHUTDN        PORTDbits.RD1

//Function prototypes for all of the function in the library
//Please refer to the pc_uart.c file for instruction on how to use these functions
void PC_SetupSPI(void);
void PC_WriteConfiguration(void);
void PC_WriteData(unsigned char);
char PC_ReadData(void);
void PC_ReadConfiguration(unsigned char*);
void PC_WriteString(unsigned char*, unsigned char);

#endif
```

```
/************************************************************************
 *
 * PC SPI UART with MAXIM 3100 Library
 * Filename: pc_uart.c
 *
 * THIS IS THE CODE FILE
 *
 ************************************************************************
 *
 * Developed By: Michael Aeberhard (michael.aeberhard@gatech.edu)
 * Date: October 11th, 2007
 * Purpose: Easy-to-use functions sending and receiving data via the MAXIM 3100
 * UART chip using the PIC's onboard SPI module.
 *
 * Restrictions: This library assumes the use of the PIC18F4520 microcontroller
 * from Microchip along with the C18 Compiler.
 *
 ************************************************************************
 *
 * This file contains numerous functions that can be used to communicate with
 * a PC.  Some of the functions are more generic, while others are much
 * more specific.  The original purpose of this library was to write a set of
 * useful functions to be used in the automated foosball table senior design
 * project by Michael Aeberhard, Shane Connelly, Evan Tarr, and Nardis Walker
 * at the Georiga Institute of Technology.
 *
 ************************************************************************/

#include <p18f452.h>          //Use PIC184520 naming conventions
#include <spi.h>                      //Functions for the on-chip SPI
#include <delays.h>                   //Functions for delays
#include <pc_uart.h>          //Prototype declarations for PC UART library


/************************************************************************
 * PC_SetupSPI()
 *
 * Sets up the SPI module to communicate at 1 MBit/s.
 *
 * Parameters: none
 * Returns: void
 ************************************************************************/
void PC_SetupSPI() {
        //SPI Setup for RS-232/UART communication
        OpenSPI(SPI_FOSC_4, MODE_00, SMPMID);
        Delay10TCYx(1);
        PC_WriteConfiguration();
} //end PC_SetupSPI()


/************************************************************************
 * PC_WriteConfiguration()
 *
 * Send a write configuration to the MAX3100 UART chip.
 *
 * Parameters: none
 * Returns: void
 ************************************************************************/
void PC_WriteConfiguration() {
        unsigned char config[2] = { 0xCC, 0x00 };
        unsigned char read[2] = { 0, 0 };
        unsigned char i;
        PC_UART_SELECT = 0;
        for (i = 0; i < sizeof(config); i++) {
                while (WriteSPI(config[i]) == 0xFF) {
                        ;
                }
                read[i] = SSPBUF;
        }
        PC_UART_SELECT = 1;
} //end PC_WriteConfiguration()
```

```
/*****************************************************************************
 * PC_ReadConfiguration()
 *
 * Receives a byte from an AX12 servo data line.
 *
 * Parameters:
 *             *readResult - buffer to store the result of the configuration read.
 * Returns: void
 *****************************************************************************/
void PC_ReadConfiguration(unsigned char *readResult) {
        unsigned char config[2] = { 0x40, 0x00 };
        unsigned char i;
        PC_UART_SELECT = 0;
        for (i = 0; i < sizeof(config); i++) {
                while (WriteSPI(config[i]) == 0xFF) {
                        ;
                }
                readResult[i] = SSPBUF;
        }
        PC_UART_SELECT = 1;
} //end PC_ReadConfiguration()


/*****************************************************************************
 * PC_WriteData()
 *
 * Outputs a byte for transmission to the UART.
 *
 * Parameters:
 *             toWrite - byte to write
 * Returns: (char) 1 if byte is ready, 0 if not
 *****************************************************************************/
void PC_WriteData(unsigned char toWrite) {
        unsigned char data[2] = { 0x80, toWrite };
        unsigned char read[2] = { 0, 0 };
        unsigned char i;
        PC_UART_SELECT = 0;
        for (i = 0; i < sizeof(data); i++) {
                while (WriteSPI(data[i]) == 0xFF) {
                        ;
                }
                read[i] = SSPBUF;
        }
        PC_UART_SELECT = 1;
} //end PC_WriteData()


/*****************************************************************************
 * PC_ReadData()
 *
 * Reads a byte from the UART.
 *
 * Parameters:
 *             toWrite - byte to write
 * Returns: (char) 1 if byte is ready, 0 if not
 *****************************************************************************/
char PC_ReadData() {
        unsigned char data[2] = { 0x00, 0x00 };
        unsigned char read[2] = { 0, 0 };
        unsigned char i;
        PC_UART_SELECT = 0;
        for (i = 0; i < sizeof(data); i++) {
                while (WriteSPI(data[i]) == 0xFF) {
                        ;
                }
                read[i] = SSPBUF;
        }
        PC_UART_SELECT = 1;
        return read[1];
```

```
} //end PC_ReadData()


/*****************************************************************************
 * PC_WriteString()
 *
 * Outputs a string for transmission to the UART.
 *
 * Parameters:
 *             *toWrite - pointer to the string to write
 *             length - length of the string
 * Returns: void
 *****************************************************************************/
void PC_WriteString(unsigned char *toWrite, unsigned char length) {
        unsigned char i;
        unsigned char readConfig[2] = { 0, 0 };
        int j;
        DisableInterrupts;
        for (i = 0; i < length; i++) {
                PC_WriteData(toWrite[i]);
                Delay10TCYx(100);
                while (PC_UART_IRQ == 1) {
                        ;
                }
                Delay10TCYx(100);
        }
        EnableInterrupts;
} //end PC_WriteString()
```

Appendix C

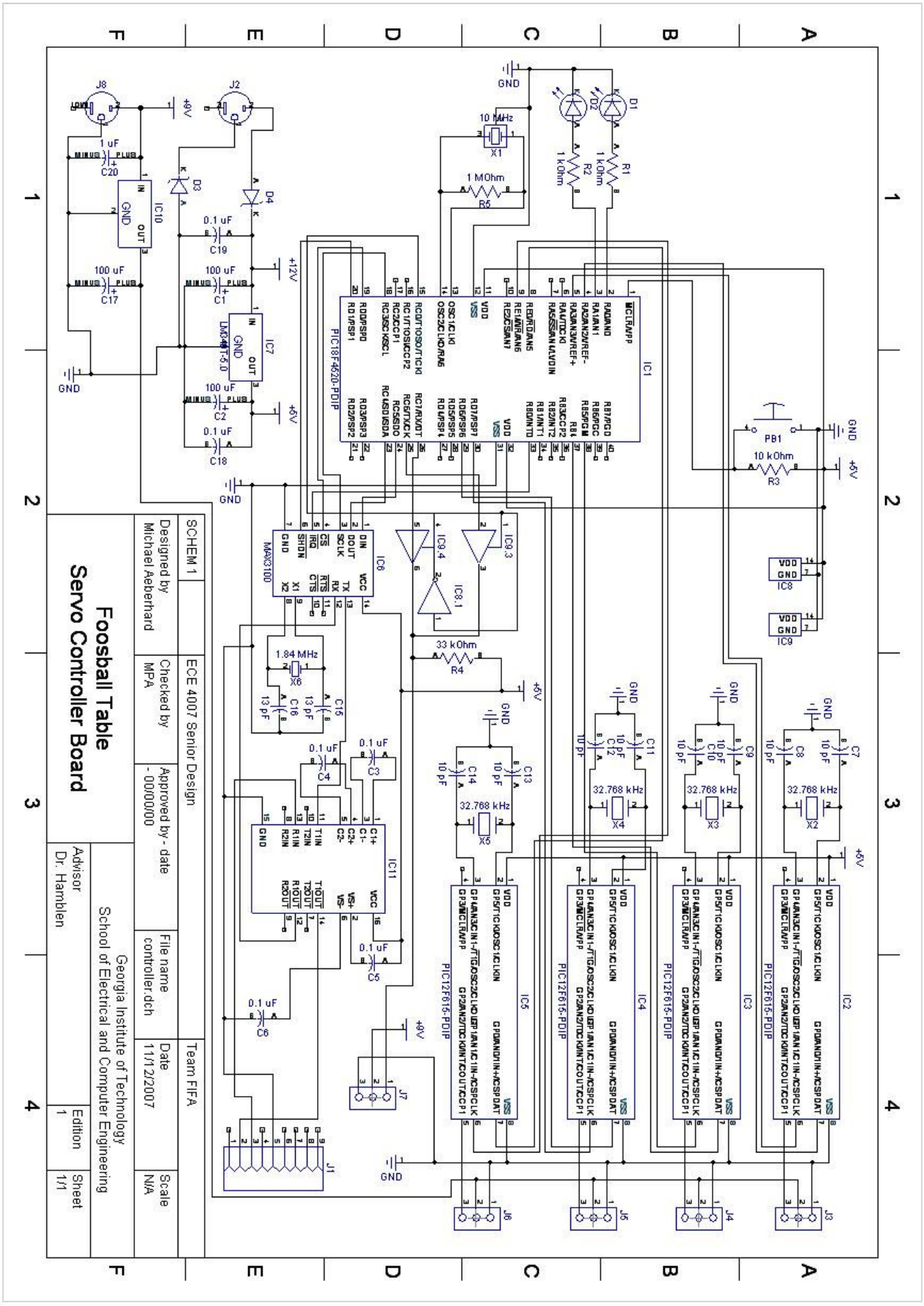
PIC12F615 PWM Controller Source Code

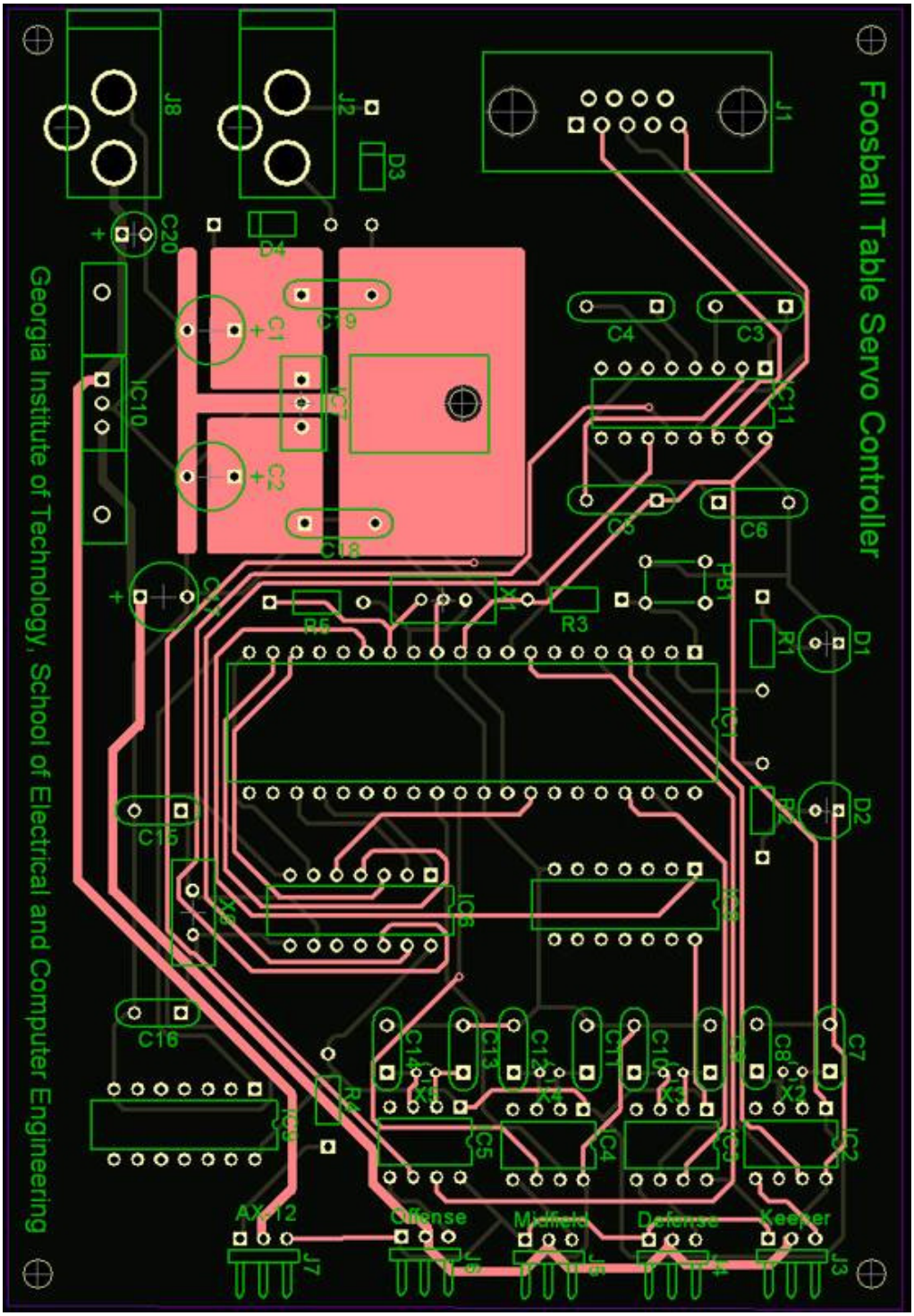
```
;
; Team FIFA PWM Servo Controller
;
; Developed By: Michael Aeberhard
; Date: October 31st, 2007
; File: main.asm
; Purpose: Generate a PWM signal for a PWM module based on the input signals.
;

        list            p=12F615
        #include        P12F615.INC
        __config        _LP_OSC & _PWRTE_OFF & _WDT_OFF & _CP_OFF

MOVLF   macro  literal,dest
        movlw  literal
        movwf  dest
        endm

;Constants
backCCP                 equ             0x1C            ;9
backCCPL                equ             2
idleCCP                 equ             0x0C            ;12
idleCCPL                equ             2
kickCCP                 equ             0x2C            ;16
kickCCPL                equ             3
PR2value                equ             28


;;;;;;; Vectors ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        org  0x0000             ;Reset vector
        goto  Main

        org  0x0004             ;Interrupt vector
        goto  $

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

Initial
                bcf             STATUS,IRP
                bcf             STATUS,RP1
                bsf             STATUS,RP0
                bsf             TRISIO,TRISIO0          ;GP0 as input
                bsf             TRISIO,TRISIO1          ;GP1 as input
                clrf            ANSEL

                MOVLF           PR2value,PR2            ;Set PR2 for 50 Hz
                bsf             TRISIO,TRISIO2          ;Set GP2 for PWM use
                bcf             STATUS,RP0
                bcf             PIR1,TMR2IF
                bcf             T2CON,T2CKPS1           ;Set Timer 2 prescalar of 4
                bsf             T2CON,T2CKPS0
                bsf             T2CON,TMR2ON            ;Turn Timer 2 on
                MOVLF           idleCCP,CCP1CON         ;Set for PWM mode and set duty cycle
                MOVLF           idleCCPL,CCPR1L         ;Set duty cycle of PWM
Test0           btfss           PIR1,TMR2IF
                goto            Test0
                bsf             STATUS,RP0
                bcf             TRISIO,TRISIO2
                bcf             STATUS,RP0
                return

Main
                call    Initial

MainLoop
                btfsc   GPIO,0                          ;Skip if GP0 is 0 (idle servo)
                call    ServoKick                       ;GP0 is 1 - call to change duty cycle
                btfss   GPIO,0                          ;Skip if GP0 is 1 (kick action)
                call    ServoIdle
```

```
                goto    MainLoop

ServoActive
                call    ServoKick
                btfsc   GPIO,1                  ;Skip if GP1 is 0 (kick)
                call    ServoBack               ;Reverse servo (defend)
                btfss   GPIO,1                  ;Skip if GP1 is 1 (defend)
                call    ServoKick
                goto    MainLoop

ServoBack
                MOVLF   backCCP,CCP1CON         ;Set for PWM mode and set duty cycle
                MOVLF   backCCPL,CCPR1L         ;Set duty cycle of PWM
                return

ServoKick
                MOVLF   kickCCP,CCP1CON         ;Set for PWM mode and set duty cycle
                MOVLF   kickCCPL,CCPR1L         ;Set duty cycle of PWM
                return

ServoIdle
                MOVLF   idleCCP,CCP1CON         ;Set for PWM mode and set f duty cycle
                MOVLF   idleCCPL,CCPR1L         ;Set duty cycle of PWM
                return

END
```

Appendix D

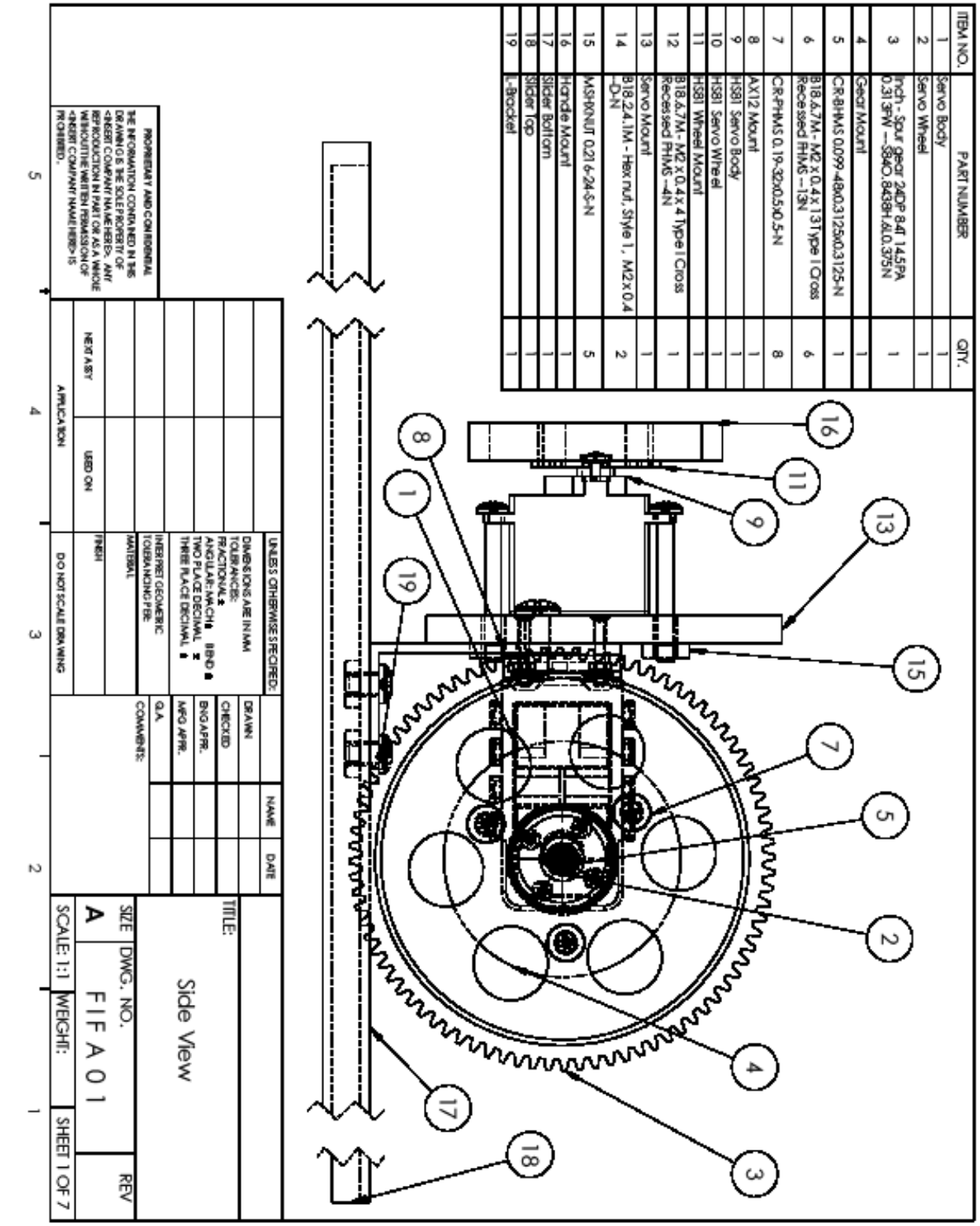
Servo Controller Board
Schematic and PCB Design


Foosball Table
Servo Controller Board
SCHEM 1
ECE 4007 Senior Design
Team FIFA
Designed by
Michael Aeberhard
Checked by
MPA
Approved by - date
- 00/00/00
File name
controller.dch
Date
11/12/2007
Scale
N/A
Edition
1
Sheet
1/1
Advisor
Dr. Hamblen
Georgia Institute of Technology
School of Electrical and Computer Engineering
PIC18F4620-PDIP
PIC12F615-PDIP
MAX3100
LM340T-5.0
10 MHz
1 MOhm
1 kOhm
10 kOhm
33 kOhm
1.84 MHz
32.768 kHz
10 pF
13 pF
0.1 uF
1 uF
100 uF
+5V
+6V
+9V
+12V
GND


Foosball Table Servo Controller
Georgia Institute of Technology, School of Electrical and Computer Engineering
AX-12
Offense
Midfield
Defense
Keeper

Appendix E

Mechanical Design Drawings



ITEM NO.	PART NUMBER	QTY.
1	Servo Body	1
2	Servo Wheel	1
3	Inch - Spur gear 24DP 84T 14.5PA 0.313FW --S64O.8438H.4L0.375N	1
4	Gear Mount	1
5	CR-8HMS.0.099-48x0.3125x0.3125-N	1
6	B18.6.7M - M2 x 0.4 x 13 Type I Cross Recessed PHMS --13N	6
7	CR-PHMS.0.19-32x0.5x0.5-N	8
8	AX12 Mount	1
9	HS81 Servo Body	1
10	HS81 Servo Wheel	1
11	HS81 Wheel Mount	1
12	B18.6.7M - M2 x 0.4 x 4 Type I Cross Recessed PHMS --4N	1
13	Servo Mount	1
14	B18.2.4.1M - Hex nut, Style 1, M2 x 0.4 --D-N	2
15	MSHBNUT.0.21.6-24-S-N	5
16	Handle Mount	1
17	Slider Bottom	1
18	Slider Top	1
19	L-Bracket	1

PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

NEXT ASSY | USED ON | APPLICATION

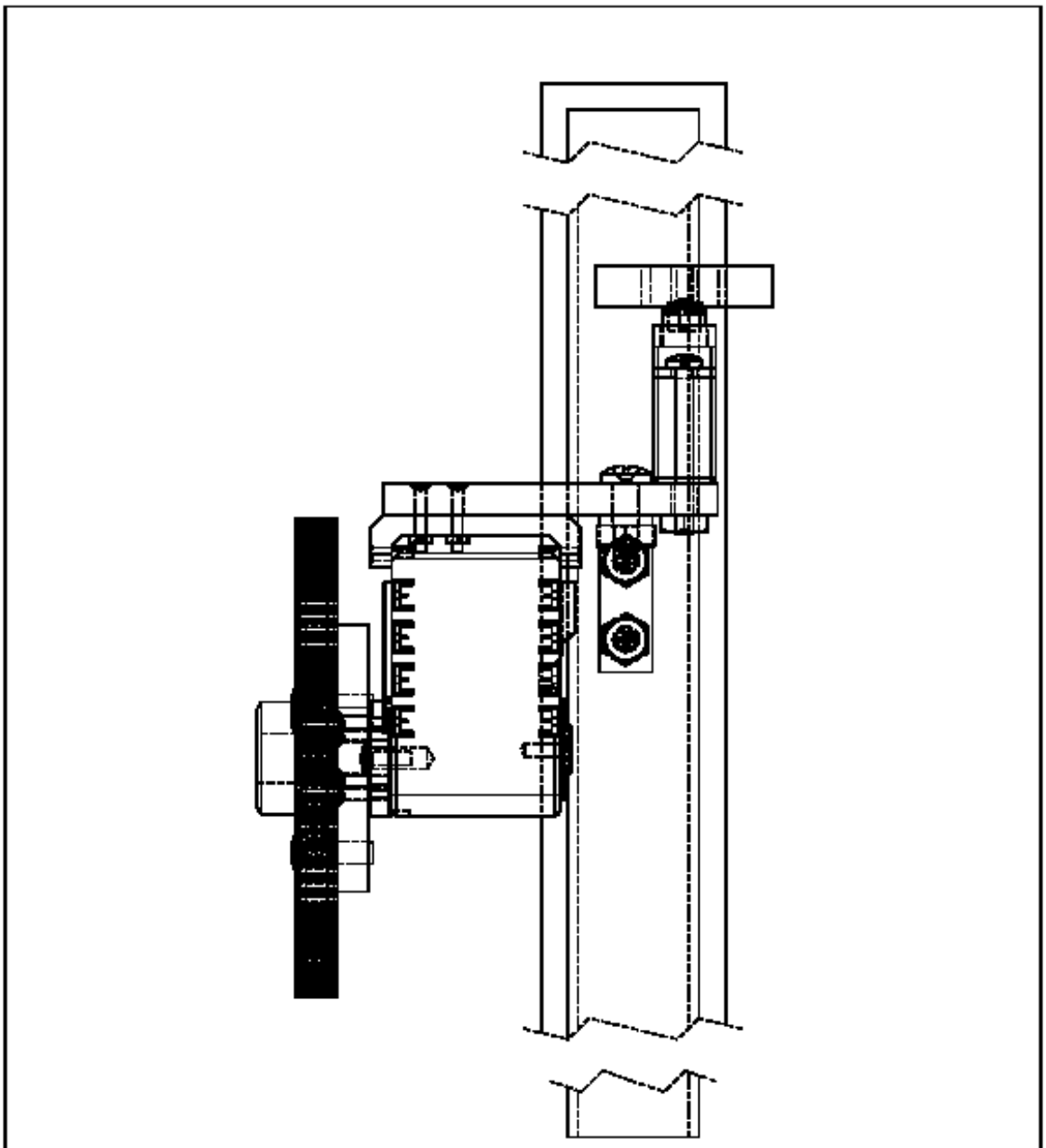
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MM
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ± .x
THREE PLACE DECIMAL ± .x

INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
DO NOT SCALE DRAWING

	NAME	DATE
DRAWN		
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		
COMMENTS:		

TITLE: Side View

SIZE: A | DWG. NO.: FIFA01 | REV:
SCALE: 1:1 | WEIGHT: | SHEET 1 OF 7

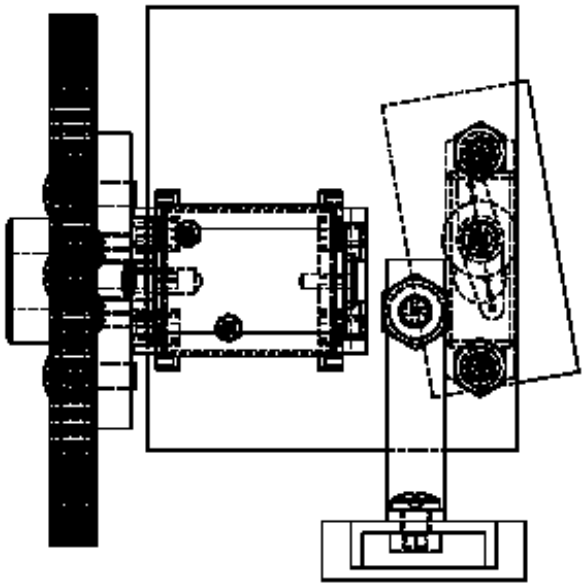


		DIMENSIONS ARE IN MM		NAME	DATE
		TOLERANCES:	DRAWN		
		FRACTIONAL ±	CHECKED		
		ANGULAR: MACH ± BEND ±	ENG APPR.		
		TWO PLACE DECIMAL ±	MFG APPR.		
		THREE PLACE DECIMAL ±	Q.A.		
		MATERIAL	COMMENTS:		
NEXT ASSY	USED ON	FINISH			
APPLICATION		DO NOT SCALE DRAWING			

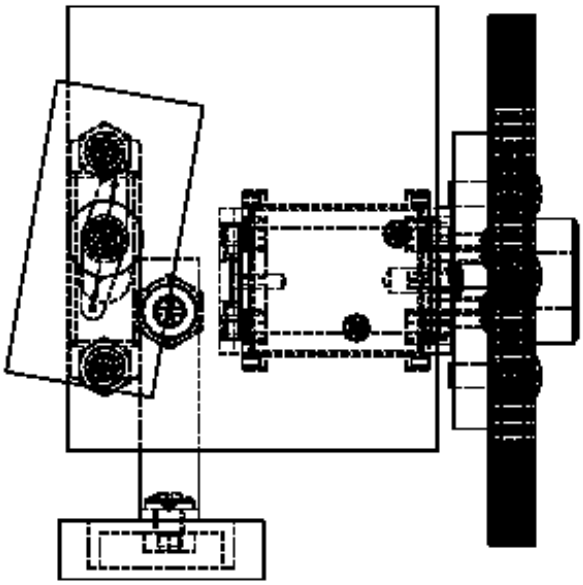
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.

Top View

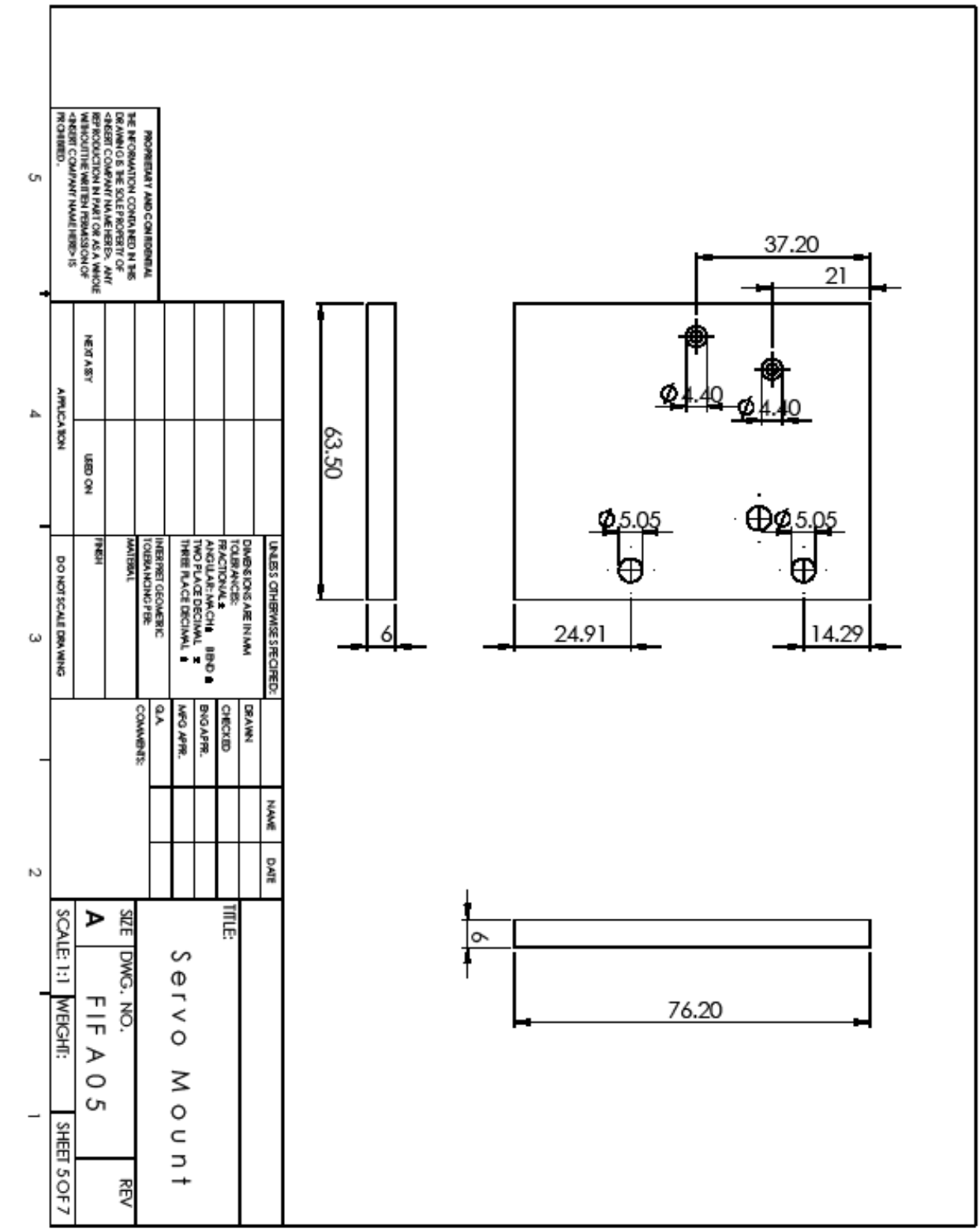
SIZE	DWG. NO.	REV.
A	FIFA02	
SCALE:1:1	WEIGHT:	SHEET 2 OF 7

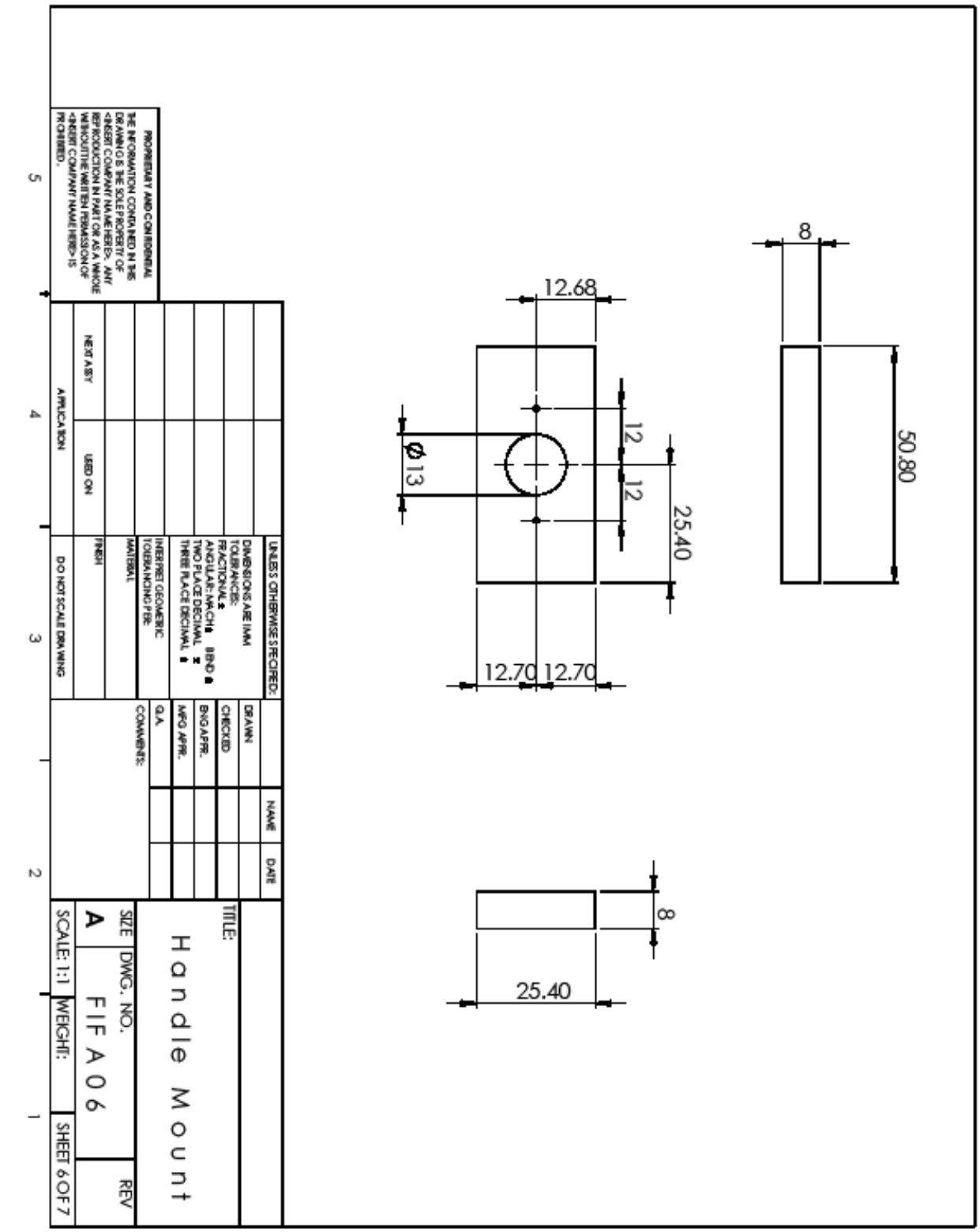


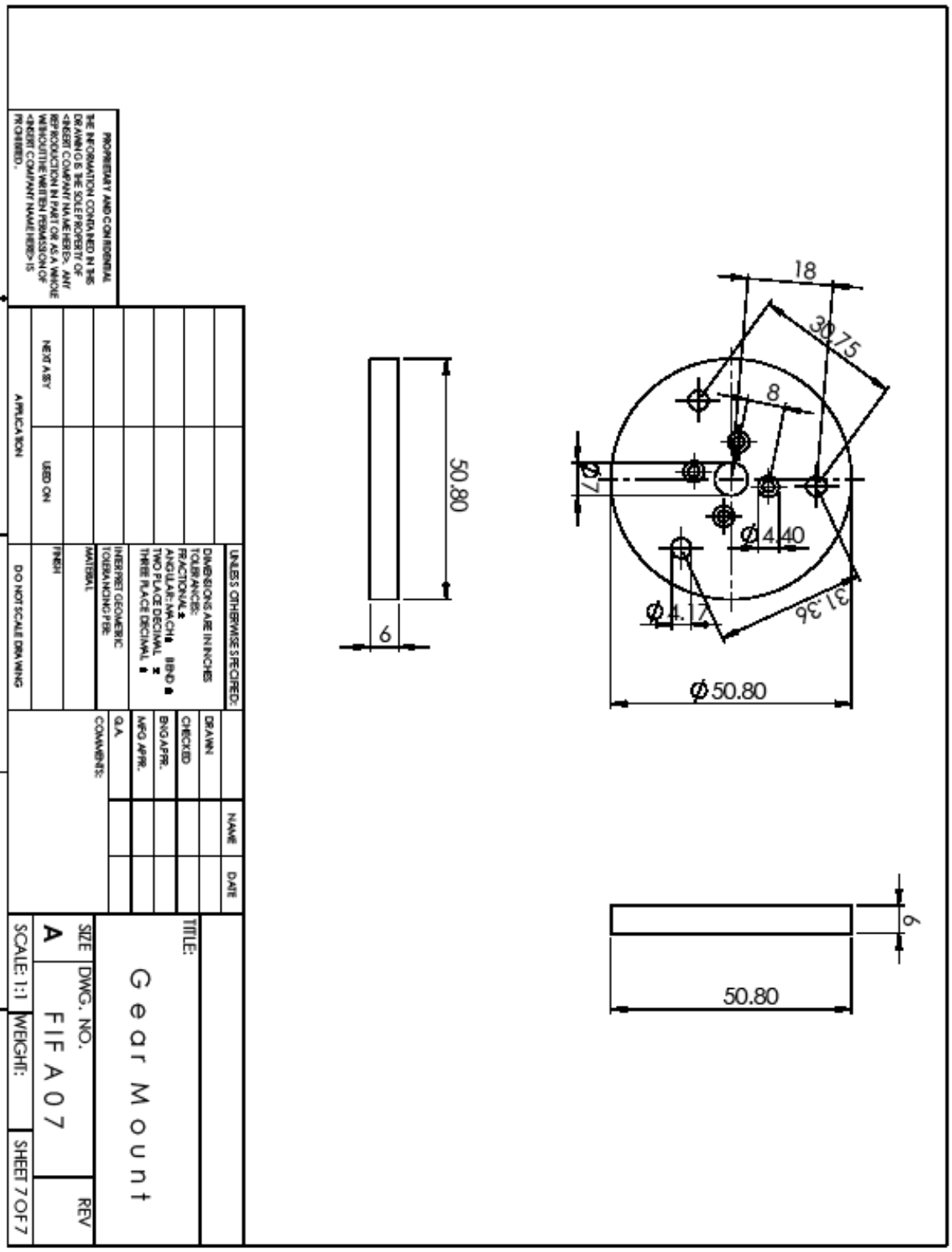
PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.	NEXT ASSY / USED ON APPLICATION	DIMENSIONS ARE IN MM TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ± MATERIAL FINISH DO NOT SCALE DRAWING	NAME / DATE DRAWN CHECKED ENG APPR. MFG APPR. Q.A. COMMENTS:	Back View SIZE A DWG. NO. FIFA03 REV. SCALE:1:1 WEIGHT: SHEET 3 OF 7	



PROPRIETARY AND CONFIDENTIAL THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.	NEXT ASSY / USED ON APPLICATION	DIMENSIONS ARE IN MM TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ± MATERIAL FINISH DO NOT SCALE DRAWING	NAME / DATE DRAWN CHECKED ENG APPR. MFG APPR. Q.A. COMMENTS:	Front View SIZE A DWG. NO. FIFA04 REV. SCALE:1:1 WEIGHT: SHEET 4 OF 7	


37.20
21
Ø4.40
Ø4.40
Ø5.05
Ø5.05
63.50
6
24.91
14.29
6
76.20
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.
NEXT ASSY
USED ON
APPLICATION
DO NOT SCALE DRAWING
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MM
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
NAME
DATE
DRAWN
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
TITLE:
Servo Mount
SIZE
A
DWG. NO.
FIFA05
REV
SCALE: 1:1
WEIGHT:
SHEET 5 OF 7


8
50.80
12.68
12
12
25.40
Ø13
12.70
12.70
8
25.40
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.
NEXT ASSY
USED ON
APPLICATION
DO NOT SCALE DRAWING
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MM
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ±
THREE PLACE DECIMAL ±
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
FINISH
NAME
DATE
DRAWN
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
TITLE:
Handle Mount
SIZE
A
DWG. NO.
FIFA06
REV
SCALE: 1:1
WEIGHT:
SHEET 6 OF 7


18
30.75
8
Ø7
Ø4.40
Ø4.17
31.36
Ø50.80
50.80
6
6
50.80
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF <INSERT COMPANY NAME HERE>. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF <INSERT COMPANY NAME HERE> IS PROHIBITED.
NEXT ASSY
USED ON
APPLICATION
DO NOT SCALE DRAWING
FINISH
MATERIAL
INTERPRET GEOMETRIC TOLERANCING PER:
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES:
FRACTIONAL ±
ANGULAR: MACH ± BEND ±
TWO PLACE DECIMAL ± .xx
THREE PLACE DECIMAL ± .xxx
DRAWN
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
NAME
DATE
TITLE:
Gear Mount
SIZE
A
DWG. NO.
FIFA07
REV
SCALE: 1:1
WEIGHT:
SHEET 7 OF 7

Appendix F

Cost Analysis

Cost and price calculations for a an automated foosball table with a selling price of $5,000.

Example of Cost and Price Calculations

Fringe Benefits	***25%***	of labor
Overhead	***55%***	of materials, labor & fringe
Sales & Marketing Expense	***25%***	of selling price
Warranty & Support Expense	***5%***	of selling price

Development Cost (Non-recurring Cost)

What it costs the company to develop the product

Parts	***1,600***
Labor	***22,400***
Fringe Benefits, % of Labor	5,600
Subtotal	29,600
Overhead, % of Matl, Labor & Fringe	16,280
Total	$45,880

Determination of Selling Price

What the customer pays the company for the finished product

Based on: ***500*** units

Parts Cost	***710***	
Assembly Labor	***25***	
Testing Labor	***50***	
Total Labor	75	
Fringe Benefits, % of Labor	19	
Subtotal	804	
Overhead, % of Matl, Labor & Fringe	442	
Subtotal, Input Costs	1,246	
Sales & Marketing Expense	1,250	
Warranty & Support Expense	250	
Amortized Development Costs	92	
Subtotal, All Costs	2,838	
Profit	2,162	43.2%
Selling Price	***$5,000***	

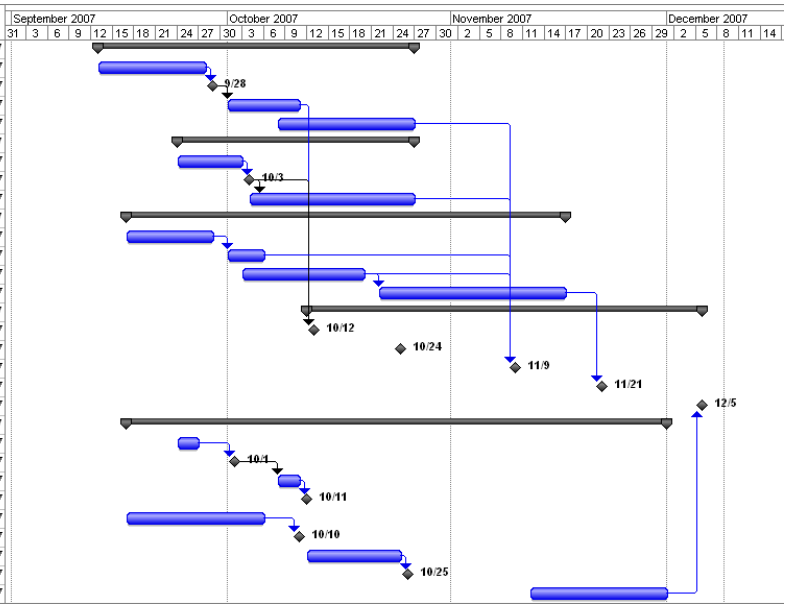
Total Revenue	$2,500,000
Total Profit	$1,081,214

Appendix G

Prototype Development Gantt Chart

Projected Prototype Development

ID	Task Name	Duration	Start	Finish
1	**PIC Microcontroller Board**	**32 days?**	**Thu 9/13/07**	**Fri 10/26/07**
2	Assemble PIC board	11 days	Thu 9/13/07	Thu 9/27/07
3	Demonstrate PIC Board	1 day?	Fri 9/28/07	Fri 9/28/07
4	Servo motor communication	8 days?	Mon 10/1/07	Wed 10/10/07
5	Communications with compute	15 days?	Mon 10/8/07	Fri 10/26/07
6	**Mechanical Assembly**	**25 days?**	**Mon 9/24/07**	**Fri 10/26/07**
7	Work on basic assembly	7 days?	Mon 9/24/07	Tue 10/2/07
8	Basic assembly finished	1 day	Wed 10/3/07	Wed 10/3/07
9	Continue assembly improveme	17 days?	Thu 10/4/07	Fri 10/26/07
10	**Image Processing Computer**	**45 days?**	**Mon 9/17/07**	**Fri 11/16/07**
11	Ball detection	10 days?	Mon 9/17/07	Fri 9/28/07
12	Ball path prediciting	5 days?	Mon 10/1/07	Fri 10/5/07
13	Communication with microcont	13 days?	Wed 10/3/07	Fri 10/19/07
14	Artificial Intelligence	20 days?	Mon 10/22/07	Fri 11/16/07
15	**Prototype Integration Mileston**	**39 days?**	**Fri 10/12/07**	**Wed 12/5/07**
16	Lateral movement of one hanc	1 day?	Fri 10/12/07	Fri 10/12/07
17	Lateral and kick movement of	1 day?	Wed 10/24/07	Wed 10/24/07
18	Automated movement towards	1 day?	Fri 11/9/07	Fri 11/9/07
19	Prototype completed	1 day?	Wed 11/21/07	Wed 11/21/07
20	Prototype Presentation	1 day?	Wed 12/5/07	Wed 12/5/07
21	**Presentation Materials**	**55 days?**	**Mon 9/17/07**	**Fri 11/30/07**
22	Work on project summary	3 days?	Mon 9/24/07	Wed 9/26/07
23	Draft finished	1 day?	Mon 10/1/07	Mon 10/1/07
24	Make revisions to project sum	3 days?	Mon 10/8/07	Wed 10/10/07
25	Final Project Summary due	1 day?	Thu 10/11/07	Thu 10/11/07
26	Work on website	15 days?	Mon 9/17/07	Fri 10/5/07
27	Website live	1 day?	Wed 10/10/07	Wed 10/10/07
28	Work on PDR	9 days?	Fri 10/12/07	Wed 10/24/07
29	Preliminary Design Review	1 day?	Thu 10/25/07	Thu 10/25/07
30	Design Report	15 days?	Mon 11/12/07	Fri 11/30/07



Actual Prototype Development

ID	Task Name	Duration	Start	Finish	Predecessors
1	**PIC Microcontroller Board**	**54 days?**	**Thu 9/13/07**	**Tue 11/27/07**	
2	Assemble PIC board	11 days	Thu 9/13/07	Thu 9/27/07	
3	Demonstrate PIC Board	1 day?	Fri 10/5/07	Fri 10/5/07	2
4	Servo motor communication	8 days?	Wed 10/17/07	Fri 10/26/07	3
5	Communications with compute	37 days?	Mon 10/8/07	Tue 11/27/07	
6	**Mechanical Assembly**	**54 days?**	**Mon 9/24/07**	**Thu 12/6/07**	
7	Work on basic assembly	15 days?	Mon 9/24/07	Fri 10/12/07	
8	Basic assembly finished	1 day	Mon 10/15/07	Mon 10/15/07	7
9	Continue assembly improveme	38 days?	Tue 10/16/07	Thu 12/6/07	8
10	**Image Processing Computer**	**59 days?**	**Mon 9/17/07**	**Thu 12/6/07**	
11	Ball detection	20 days?	Mon 9/17/07	Fri 10/12/07	
12	Ball path prediciting	35 days?	Mon 10/15/07	Fri 11/30/07	11
13	Communication with microcont	40 days?	Wed 10/3/07	Tue 11/27/07	
14	Artificial Intelligence	7 days?	Wed 11/28/07	Thu 12/6/07	13
15	**Prototype Integration Mileston**	**33 days?**	**Wed 10/24/07**	**Fri 12/7/07**	
16	Lateral movement of one hanc	1 day?	Fri 11/23/07	Fri 11/23/07	8,4
17	Lateral and kick movement of	26 days?	Wed 10/24/07	Wed 11/28/07	
18	Automated movement towards	3 days?	Mon 12/3/07	Wed 12/5/07	12,13,5
19	Prototype completed	1 day?	Fri 12/7/07	Fri 12/7/07	14
20	Prototype Presentation	1 day?	Fri 12/7/07	Fri 12/7/07	
21	**Presentation Materials**	**63 days?**	**Mon 9/17/07**	**Wed 12/12/07**	
22	Work on project summary	3 days?	Mon 9/24/07	Wed 9/26/07	
23	Draft finished	1 day?	Mon 10/1/07	Mon 10/1/07	22
24	Make revisions to project sum	3 days?	Mon 10/8/07	Wed 10/10/07	23
25	Final Project Summary due	1 day?	Thu 10/11/07	Thu 10/11/07	24
26	Work on website	15 days?	Mon 9/17/07	Fri 10/5/07	
27	Website live	1 day?	Wed 10/10/07	Wed 10/10/07	26
28	Work on PDR	9 days?	Fri 10/12/07	Wed 10/24/07	
29	Preliminary Design Review	1 day?	Thu 10/25/07	Thu 10/25/07	28
30	Design Report	5 days	Thu 12/6/07	Wed 12/12/07