Automated Foosball Table

Project Design Report

Design Team 01

Ahmed Alsalihi

Khalil Najjar

Bryan Van Scoy

Joe Zifer

Dr. Hartley

Dr. Tran

November 28, 2011

Table of Contents

List of Figures

List of Tables

1. Problem Statement

a. Need

The game of foosball has entered the homes of people all over the world since it was first invented in 1922 to replicate the game of soccer. The game was patented in the UK in 1923 and in the US in 1927 and remains as one of the oldest table games still played today, falling only to billiards and table tennis. Today, five annual World Championship Series events are hosted by the International Table Soccer Federation (ITSF) which culminates in an annual World Cup where the best players from over 40 countries compete for world titles. Even with the popularity of foosball, many tables remain highly unused, serving more as decorations than entertainment devices. The automated foosball table would integrate modern technology with a classic game to provide entertainment to foosball table owners everywhere.

b. Objective

The objective of this project is to design and build an automated foosball table which will serve as an opponent to a human player. Besides providing the thrill of challenging a robot to a game of foosball, this also allow players to practice their skills when a human opponent is unavailable. The system will also keep score of the game for ease of play. By doing so, the automated foosball table will provide an entertaining opponent for any game of foosball.

c. Research Survey

The concept of an automated foosball table has been explored by several groups of students, such as those from the University of Adelaide, the University of Sherbrooke, and the University of Waterloo. Each design consists of different ways of detecting the position of the ball, moving the foosmen, and determining where to move the foosmen for game play resembling a human's. The program to control foosmen movement can be implemented on either a microprocessor or laptop computer. This will need to communicate with the motors which need to be high-speed in order to accommodate real-time game play.

Perhaps the most difficult aspect of the project is detecting the position of the foosball. Several groups have used various types of cameras (high-speed, pin-hole) to track the ball. These designs are very complex and expensive. The group thought of a much simpler and cost-effective approach to achieving the same thing. The idea is to use infrared light emitting diodes (IR LEDs) and optical detectors (phototransistors) so that the ball breaks the light from the LED. This option allows for a very fast response, although the accuracy is limited since the LED's must be placed far enough apart to prevent illuminating multiple photo detectors. A possible alternative is to use optical reflective sensors in the bottom of the table which would reflect light back when the ball is over them indicating the position of the ball. This approach could potentially be more accurate depending on the number of sensors used, although increasing the number of sensors also increases the cost.

No patents for automated foosball tables were found. The only design found that resembles this tracking ability is a YouTube video that shows the reaction of the foosmen in response to the LED tracking.

d. Objective Tree

The objective tree is shown in Figure 1 which outlines the main objectives of the foosbot project. The specific blocks in the objective tree are derived from the marketing requirements of the project.

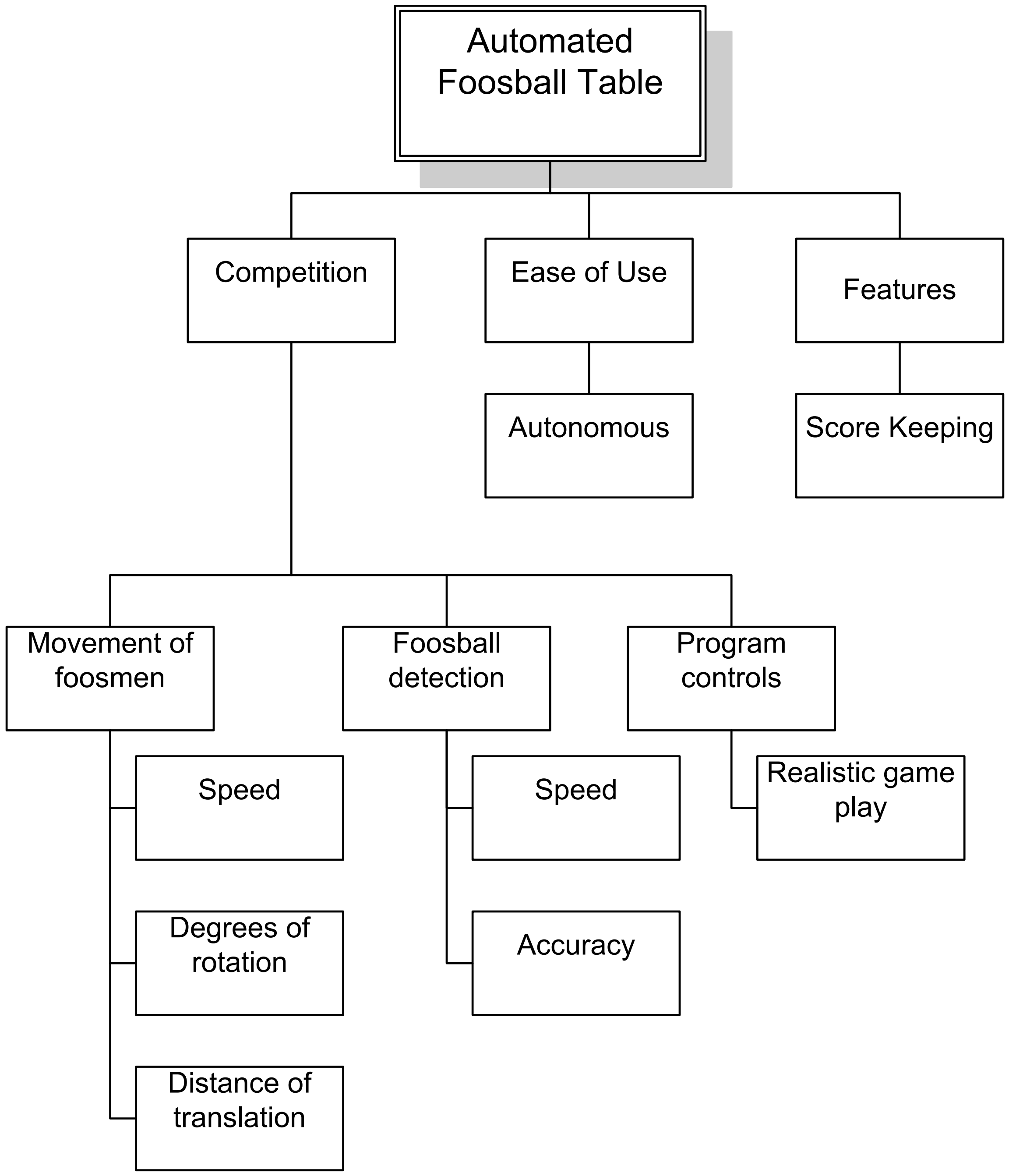


Figure 1: Objective tree.

2. Design Requirements Specification

Marketing Requirements	Engineering Requirements	Justification
Mechanical		
1,2	Each row of foosmen should be capable of moving independent of other rows.	To insure the ability to move and position the foosmen similar to a human.
	Each row of foosmen should be capable of independent rotational and linear movement.	
	The motors for linear movement should be able to provide at least 0.73 N-m of torque.	
	The motors for rotational movement should be able to provide at least 0.333 N-m of torque.	
Power		
4	The system should use a 120 VAC power supply.	The system will run anywhere in the USA (120VAC, 60Hz is the USA standard).
Communication		
3,5	The system should detect when a goal is scored, and differentiate between the two goals.	Assure tracking of the score.
Aesthetic Design		
5	The score should be displayed so that the players and audience can easily see the score.	The system should show and keep track of the score in an aesthetically pleasing way.
3	The system should automatically begin a new game when powered on.	The system should operate autonomously with little instruction from the user.
	The system should operate to play foosball whenever the foosball is detected on the table.	
Computation		
1	The position of the foosball should be detected at least every 2 ms.	The system should track the foosball quick enough for competitive play.
1	The position of the foosball should be detected with a resolution of at least 1.3 cm.	The system should accurately detect the

		position of the foosball.
1	The system should be able to track a ball traveling at 6.7 m/s.	Emulates human opponent reaction time.
	Control System	
1,2	The appropriate foosman should track the ball (take-over). For example, the center foosman should not attempt to track the ball when it is along the side of the table, instead the side foosman should track the ball.	To minimize time to reach the ball and perform a kick.
4	The system should be able to track and respond to any foosball location on the table.	To keep the robot playing continuously and eliminate "blind spots".

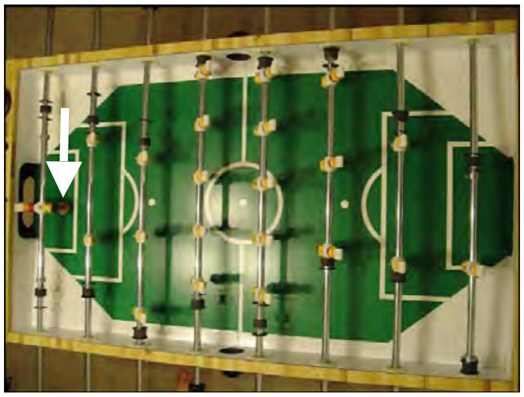
Marketing Requirements

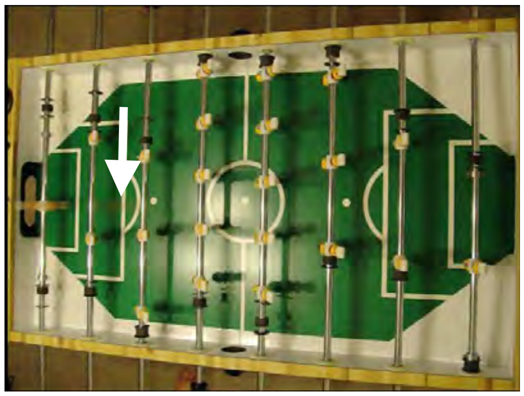
1. The system should be fast enough to provide a challenging opponent.
2. Rows of foosmen should move independent of other rows.
3. The system should operate autonomously.
4. The system should be easy to use.
5. The system should keep score of the game.

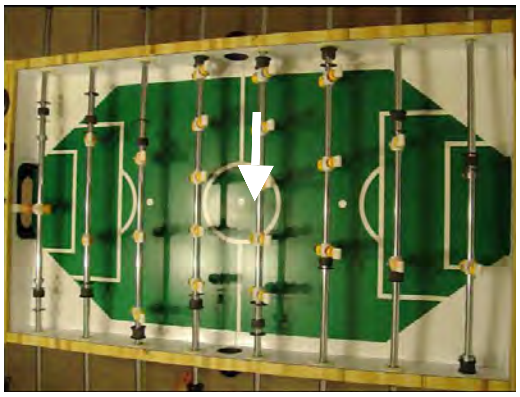
3. Accepted Technical Design

a. Speed of the Foosball - BRV

The maximum speed of the foosball is a critical parameter that must be determined accurately to ensure that the system response is fast enough to handle tracking of the foosball and movement of the foosmen at such high speeds. To determine the speed of the foosball, a digital camera operating at 15 frames-per-second (fps) is used to capture successive frames of a hard shot from the goalie as shown in Figure 2. The ball can be seen moving from left to right across the table (the images are blurry due to a relatively slow exposure time of the camera). By measuring the distance travelled by the foosball in each frame, and knowing that the time between successive frames is 1/15 of a second, the speed of the foosball can be calculated. For the recorded shot, the maximum speed of the foosball is calculated as $v = 5.85$ m/s. A 15% margin of error is used since only one shot is analyzed, so the system is designed for the maximum speed of the foosball to be $v_{max} = 6.7$ m/s.


Frame 1 – Initial condition


Frame 2 (t = 0 sec, x = 25 cm)


Frame 3 (t = 1/15 sec, x = 64 cm)

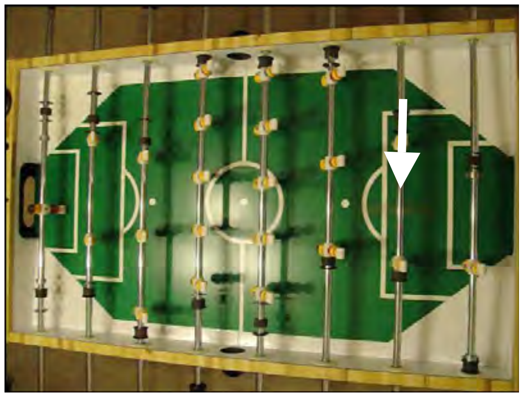

Frame 4 (t = 2/15 sec, x = 102 cm)

Figure 2: Frame-by-frame view of the foosball at a maximum speed of 5.85 m/s.

b. Motor Torque-Speed Requirements – KMN, BRV

The minimum torque-speed requirements of the stepper motors are split into two groups. First, the motors for rotary motion will be considered followed by the motors used for linear movement. Two test subjects were used to obtain the maximum angular acceleration required to emulate a human kick on the foosball table. The equipment used was the PASCO Science Workshop in conjunction with the PASCO Rotary Motion Sensor. The test setup included connecting the sensor to the workshop and sampling the sensor at 50 Hz. Each test subject held the end of the rod and the motion sensor was attached to the other side. The test subject then rotated the rod in an effort to emulate a kick on a foosball table rod. The rod handle has a radius of r_{rod} = 1.6 cm which is the distance at which the force is applied, and the rod has a mass of m = 2.041 kg. The moment of inertia of the rod can then be calculated as

$$I = \frac{1}{2}mr^2 = \frac{1}{2}(2.041\ kg)(0.016\ m)^2 = 0.00026\ kg\ m^2 \tag{1}$$

The results from the physics lab tests are displayed in Figure 3, with a maximum measured angular acceleration of $\alpha = 1276.27\ \frac{rad}{s^2}$. Using this value, the torque applied to the rod is

$$T = \alpha I = \left(1276.27\frac{rad}{s^2}\right)(0.00026\ kg\ m^2) = 0.333\ N\ m. \tag{2}$$

The force is then

$$F = \frac{T}{r_{rod}} = \frac{0.333\ N\ m}{0.016\ m} = 20.8\ N. \tag{3}$$

For the motors, this force is not applied directly to the handle of the rod, but is instead applied to an aluminum sleeve which is connected to both the rod and motor shaft. The sleeve is connected to the rod at a distance d = 1.6 cm from the center of the rod. Therefore, the required torque of the rotational motors is

$$T_{rotary} = Fd = (20.8\ N)(0.016\ m) = 0.333\ N\ m. \tag{4}$$

Figure 4 shows the test results for linear motion. The force sensor was placed against the rod and the rod pushed quickly across the table to obtain the plots shown, with a maximum measured force of F = 44.01 N. Note that the large spike in force near 1s on each plot corresponds to the rubber stopper on the rod striking the side of the foosball table, so this portion of the plots is ignored in calculations. For linear motion of the foosmen, rotational motors are connected through a pulley to linear rails. The circumference of the pulley is 105 mm, so the force is applied at a distance

$$r_{pulley} = \frac{C}{2\pi} = \frac{0.105\ m}{2\pi} = 0.0167\ m \tag{5}$$

from the center of the rod. Therefore, the required torque of the motors for linear motion is

$$T_{linear} = F\ r_{pully} = (44.01\ N)(0.0167\ m) = 0.73\ N\ m. \tag{6}$$

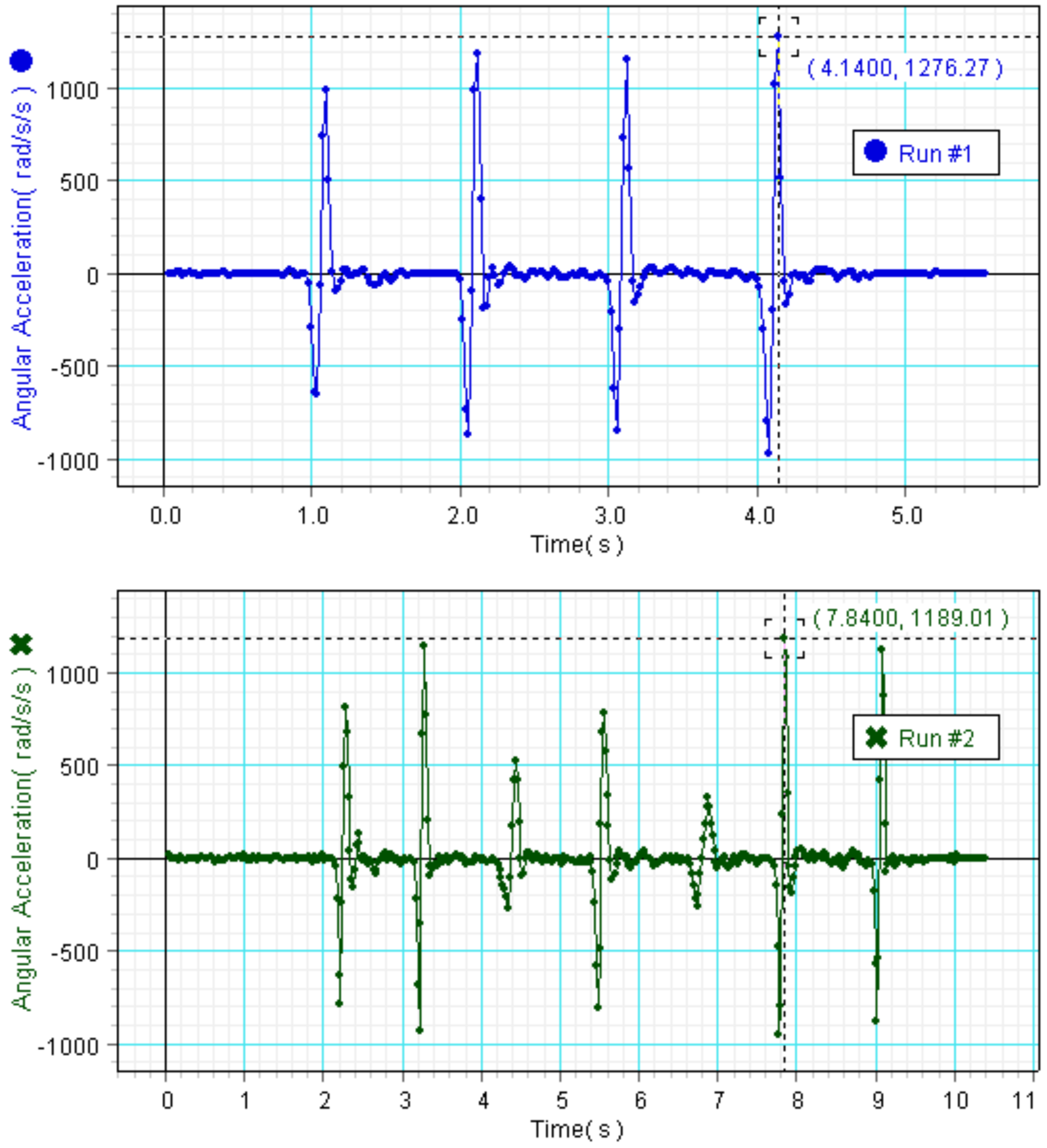
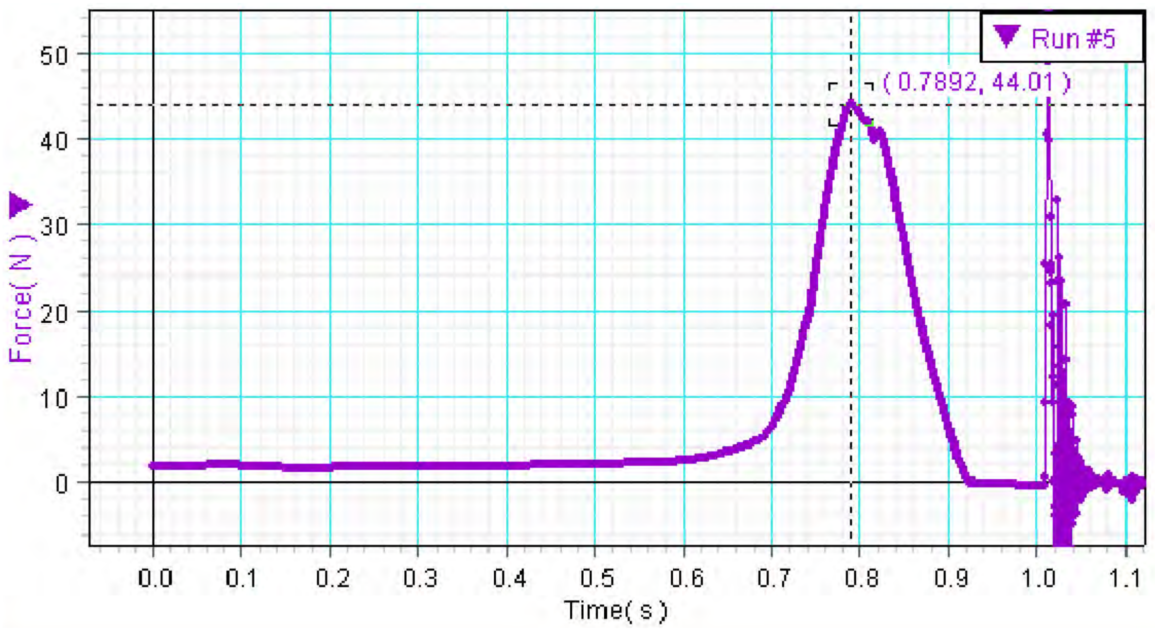


Figure 3: Angular Acceleration Test (Top – Test Subject 1, Bottom – Test Subject 2)



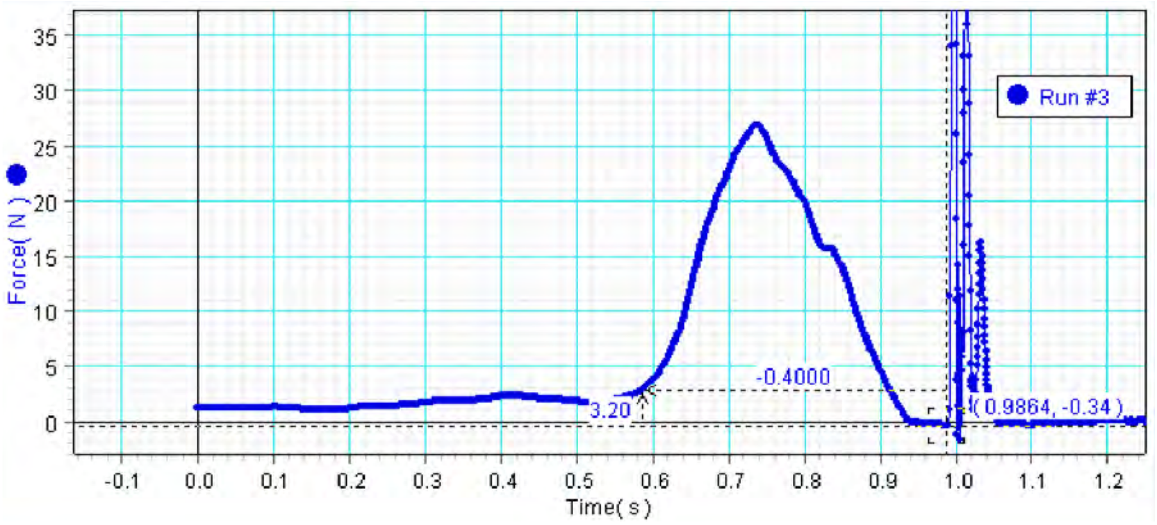


Figure 4: Linear Force Test (Top – Test Subject 1, Bottom – Test Subject 2)

c. IR LEDs and Phototransistors – BRV, KMN

Infrared light emitting diodes (IR LEDs) and phototransistors are optically coupled to detect the position of the foosball. An array of LEDs and phototransistors covers the entire playing field. The resolution *r* is defined as the distance between adjacent pairs of LEDs and phototransistors, which is chosen to be the same in both the *x* and *y* directions. In order to accurately locate the foosball of diameter 3.2 cm, the resolution is chosen to be $r = 1$ cm. Using the table coordinates shown in Figure 7 with dimensions 116.8 cm x 67.9 cm, the number of LEDs and phototransistors in the *x* and *y* directions needed for the desired resolution is

$$N_{LED,x} = N_{phototransistor,x} = \frac{x_{max}}{r} = \frac{116.8\ cm}{1\ cm} = 117 \tag{7}$$

and

$$N_{LED,y} = N_{phototransistor,y} = \frac{y_{max}}{r} = \frac{67.9\ cm}{1\ cm} = 68. \tag{8}$$

The total number of sensors is then

$$N_{LED} = N_{phototransistor} = 185. \tag{9}$$

With the LEDs and phototransistors separated by a distance *r*, the phototransistors will detect some interference from adjacent LEDs. In order to reduce this interference, several methods are used. First, the LED and phototransistor pairs are alternated so that an LED is directly adjacent to two phototransistors, and vice-versa. Another method of reducing interference is to recess the LEDs and phototransistors into the sides of the foosball table. This narrows the reception angle of the phototransistors so that only light from the corresponding LED is detected. Finally, the sensor pairs are grouped into N_{pulse} groups which are pulsed at different time intervals so that when one sensor pair is on, adjacent pairs are off. Combining each of these interference reduction techniques, the total space between sensor pairs that are on at any moment in time, r_{sensor}, is increased from *r* to

$$r_{sensor} = 2N_{pulse}r = 32\ cm \tag{10}$$

which provides a significant increase. Sixteen pulse groups are used, so the nearest adjacent sensor pairs that are on at the same time are 32 cm apart. This reduces interference while maintaining the required resolution. This concept is shown in Figure 5. The LEDs and transistors are alternated, and the pulse groups are shown by different color arrows indicating the direction of light travel. For simplicity, Figure 5 only shows the sensors for one sensing direction and only four pulse groups, but the same concept is used in both *x* and *y* directions for sixteen pulse groups.

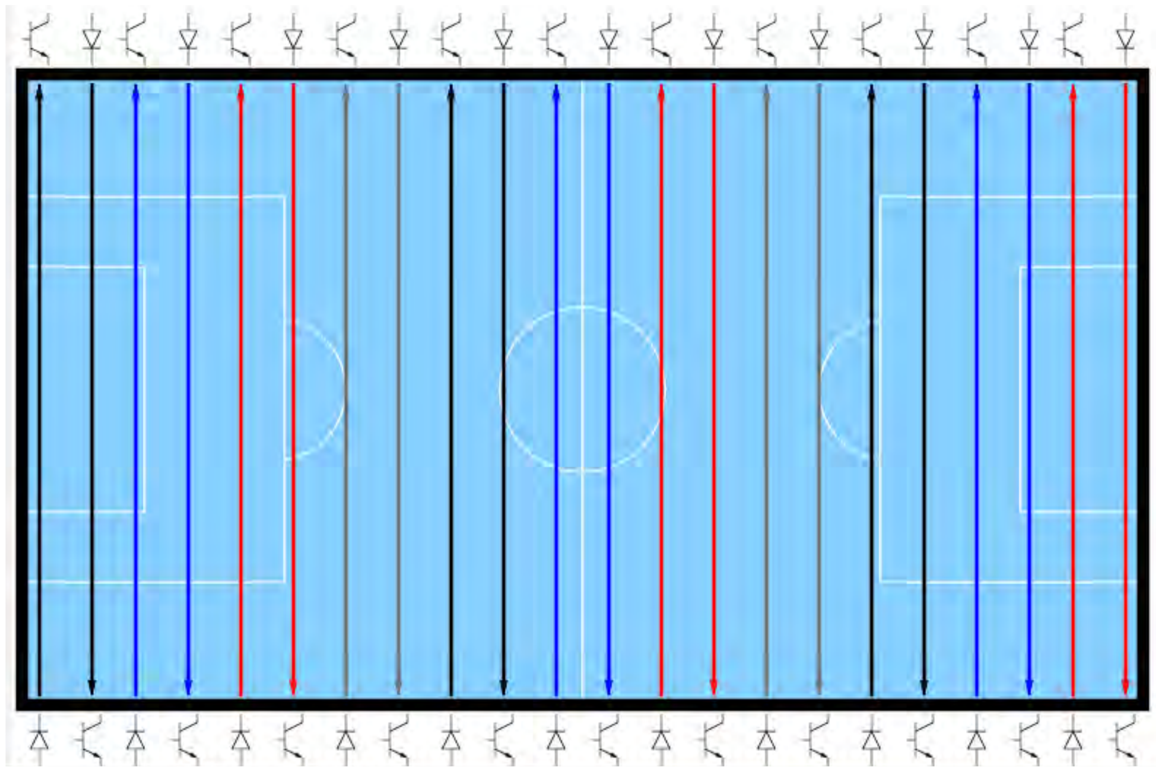


Figure 5: Sensor diagram using four pulse groups.

d. Interference Testing – KMN

In order to accurately track the ball, there must be no interference from adjacent pairs of IR LEDs during the time of sensing. To test for interference across the table during game play, an oscilloscope is used to monitor the voltage at the output of the phototransistor and a foosman is set to cross the path. The monitored voltage indicates that there is no change or fluctuation as the foosman crosses the path of the IR LED and phototransistor. Also, the sensing scheme is tested across the table to ensure that the ball crossing the path of the LED and phototransistor causes a measureable drop in voltage. For the test, two sensor pairs are placed 1cm apart. The captured image of the voltage drop from two phototransistors as the ball crosses its path can be seen in Figure 5. The voltage drops from about 4.8V to 0.6V when the ball is interrupting the sensor which is a significant drop that can easily be measured. Therefore, detection of the foosball can be accomplished using the sensing scheme described in Section 3.c.

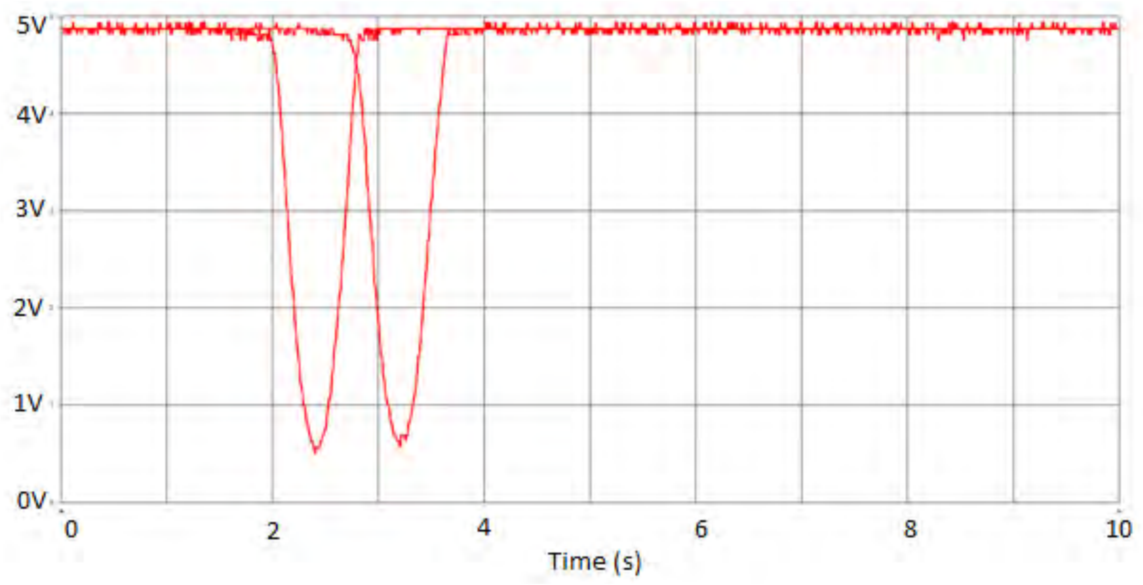


Figure 6: Voltage drop in detection of the ball in front of two phototransistors.

e. Game Strategy (Offense/Defense) - BRV

Foosball is in general a very simple game to play since the player has only limited control of the foosmen. In practice, however, many subtleties of the game must be considered in order to be effective against the best opponents.

Offense

When the ball is in front of a foosman, the row of players is rotated up quickly to produce a shot. Although this method seems trivial, it is the approach of most inexperienced players to simply hit the ball as hard as possible whenever possible. The offense can be improved upon by having the foosmen aim toward the opponent's goal. This is accomplished by slight linear movement simultaneously with the rotational shot. The amount of linear movement depends on the angle to the goal which can be calculated using the known coordinate of the foosman. An exact relationship between angle and amount of linear movement will require experimentation with the motors. This is a more advanced technique which will be implemented if time permits.

Defense

The simplest defensive strategy is to move the foosmen to always stay directly in front of the foosball. This strategy has many flaws, however, since it does not take into account which row is being controlled or the position of the goal. The complete defensive strategy is composed of several parts, depending on the row and the movement of the ball. Three defensive strategies are used, depending on the position and velocity of the ball.

1. Simple tracking – This involves having each row track the current position of the foosball. This is a very simple method and therefore is only used when the ball is behind a row. If the ball is behind the row, then simple tracking is always used for that row, no matter which tracking method is being used. This puts the foosmen in position to hit the ball as soon as the ball is in range.
2. Defensive tracking – This method places the foosmen in the optimal defensive position depending on the location of the ball. This is used when the other team is holding the foosball or moving it around slowly. The foosmen are positioned in the optimal defensive position prior to shot so that minimal movement is required once the ball is hit.
    a. Front two rows (offense) – The offense is positioned directly between the ball and the center of the goal. This method blocks the largest region of the goal possible. Both the first and second rows are positioned in the same spot since the foosmen are not always guaranteed to be in position due to takeover. When the front row is performing a takeover, the second row is still protecting the middle of the goal and vice-versa.

   b. Back two rows (defense) – The defender and goalie work together to always be in the optimal defensive position. Straight lines are drawn from the ball to each goal post, representing the possible shooting angles. The defender is positioned closest to the near post at a distance of one-third the distance between the drawn lines. Likewise, the goalie is positioned one-third of the way between the lines closest to the far post so that the defender and goalie work together to cover the entire goal.

3. Trajectory – This technique is used once a shot is made by the opponent. Assuming a linear path for the ball, the trajectory is calculated and each row of foosmen moved to where the ball intersects with the row. If the ball is struck such that it will bounce off the wall before reaching a row, the deflection is taken into account and the foosmen positioned accordingly. The equations used to calculate the trajectory are described in detail in Section 3.f.

f. Foosball Trajectory - BRV

In order to increase the response time of the system, the trajectory of the foosball is calculated using two consecutive coordinates in an attempt to predict the path of the foosball when the ball is moving toward our goal. This process assumes a straight line path for the foosball, which is a good approximation as long as the ball does not contact another player. When the ball does contact another player, the trajectory changes and the foosmen are commanded to move accordingly.

A general diagram of the process of calculating the foosball trajectory is shown in Figure 7. In order to calculate the trajectory, two consecutive coordinates, (x_1, y_1) and (x_2, y_2), must be known from the sensor inputs. The equation of the line through the points is then

$$y = \frac{y_2 - y_1}{x_2 - x_1}(x_R - x_1) + y_1 \tag{11}$$

where x_R is the x-coordinate of the row of foosmen. Since the location of the lines containing the foosmen are known, the intersection points can be calculated for each line. The foosmen are then commanded to move to the corresponding intersection point.

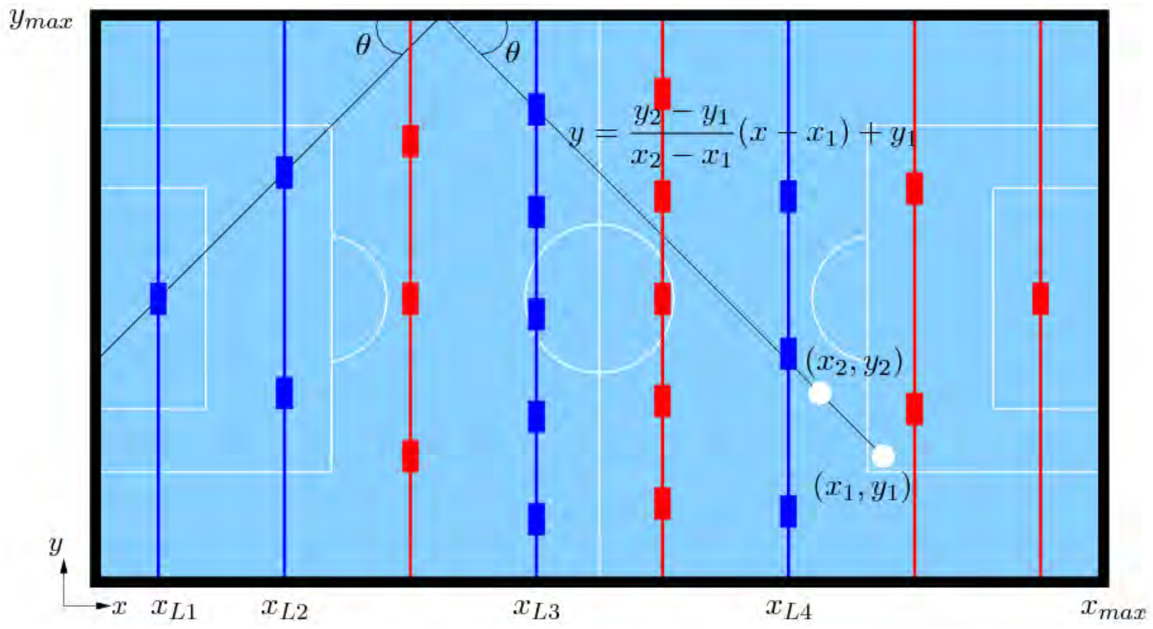


Figure 7: Foosball trajectory

In the case that the foosball deflects off a wall, the trajectory path must be modified to account for the deflection. Assuming the angle θ at which the ball strikes the wall is the same as the angle leaving the wall, the slope of the new trajectory path is known. The coordinate of where a top or bottom deflection would occur are first calculated as

$$x_{bottom} = x_1 - y_1 \frac{x_2 - x_1}{y_2 - y_1} \tag{12}$$

$$x_{top} = x_1 + (y_{max} - y_1) \frac{x_2 - x_1}{y_2 - y_1}. \tag{13}$$

If the ball is moving toward the top wall and a deflection occurs on the table, then the trajectories for each row after the deflection are calculated as

$$y = y_{max} - (x_R - x_{top}) \frac{y_2 - y_1}{x_2 - x_1} \tag{14}$$

Similarly, if the ball is moving toward the bottom wall and a deflection occurs on the table, then the trajectories for each row after the deflection are calculated as

$$y = -(x_R - x_{top}) \frac{y_2 - y_1}{x_2 - x_1}. \tag{15}$$

Using these equations, the intersection point for each row of foosmen can be calculated while taking into account a deflection off either wall.

g. Time-Division Multiplexing - BRV

The output from each phototransistor is converted to a digital signal using a comparator, and then the signal must be sent to the microprocessor. Due to the large number of sensors, this would require a large number of pins on the microprocessor if wired directly from the output of the comparator to an input pin on the microprocessor. To reduce the number of pins, time-division multiplexing is used to separate the pulse groups as shown in Figure 8. In time-division multiplexing, each signal takes turns in time using the channel. For example, the LEDs for group one are turned on, the address of the multiplexers are set to read in the data from group one, and the state of each input pin is checked. This process is then repeated for each group of sensors. By doing so, the number of required pins on the microprocessor is greatly reduced. The speed of the system is not compromised by the use of time-division multiplexing since the LEDs must already be grouped and each group pulsed separately to prevent interference. The only extra delay is due to the response time of the multiplexers which is on the order of 30 ns.

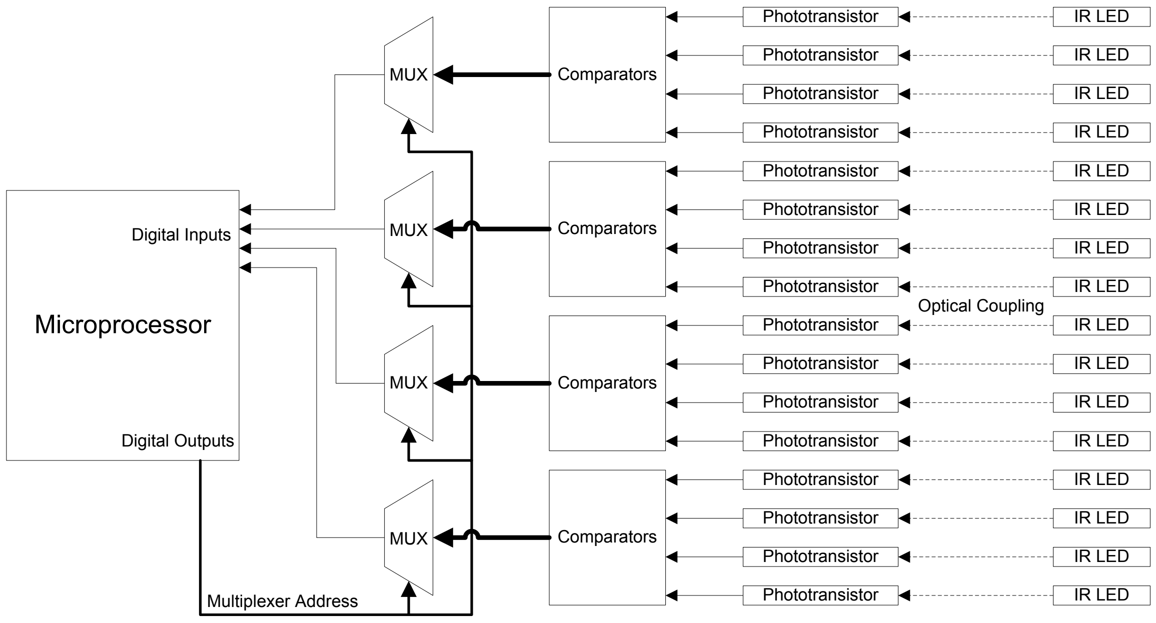


Figure 8: Multiplexing block diagram (number of blocks not to scale).

h. Scoring - BRV

For a complete design, the system should automatically detect the scoring of a goal by either team, and display the current score for players to see. Detecting a goal is done similar to sensing of the foosball on the table. An IR LED and optically coupled phototransistor pair are placed inside each goal, so that the optical connection is broken when a goal is scored. The foosball table used in the project has a plastic tube connected to the goal which is used as a ball return. The ball must travel through the tube when a goal is scored, so the sensors are placed on each of the plastic tubes. This also protects the sensors from other sources of interference since they are hidden under the table.

The score must be displayed for the players and audience to see. In order to provide an easy to read visual of the score, 10 LEDs are used to display the score for each team (20 LEDs total). This method is simple, yet provides the score in a similar manner to non-robotic foosball tables which have various types of sliders which are moved when a goal is scored. To reduce the pin count required by the microprocessor, the LEDs are multiplexed so that 5 pins are used instead of 20.

i. Microprocessor - BRV

Many design considerations must be accounted for in order to select the proper microprocessors for the project. The use of time-division multiplexing reduces the number of pins required for sensing so that a single microprocessor is capable of performing all of the required functions. Multiplexing is also utilized to display the score of the game using LEDs in order to reduce the required number of pins.

One of the main considerations for microprocessor selection is computational speed since the processor must perform all of the necessary operations in time for the system to respond to the foosball. For this reason, the AM3359 ARM Cortex A8 processor is chosen which has a clock speed of 550 – 720MHz. For ease of implementation, the BeagleBone development board is used for the project. The BeagleBone comes with a USB interface for easy programming, the AM3359 processor, 256MB DDR2 memory, and two rows of 46 pin headers. The pin headers allow for easy access to the pins on the microprocessor and are multiplexed so that different functions can be chosen. Functions available include SPI, I2C, five serial ports, seven A/D converters, and eight PWM outputs [1]. The AM3359 comes with the NEON SIMD (single instruction, multiple data) coprocessor which is capable of processing the following data types: signed/unsigned 8-bit, 16-bit, 32-bit, 64-bit, and single precision floating point. Single precision floating point will be used for most of the calculations (such as trajectory and foosmen position) which will be sufficient for providing the desired accuracy.

Table 1 shows the number of input and output pins required by the microprocessor along with the corresponding header pins and mode which will be used to connect to the microprocessor.

Table 1: I/O pins for microprocessor

Function	**Number of Pins**	**Type of Pins**	**BeagleBone**		
			Header	**Pins**	**Mode**
Power	0	-	DC Input	-	-
Phototransistor input	12	GPIO	P8	3 – 14	Mode7
MUX Address	4	GPIO	P8	15 – 18	Mode7
Tracking score (2 phototransistors and 2 IR LEDs)	4	GPIO	P8	19 – 22	Mode7
Display score (multiplexed)	5	GPIO	P8	23 – 27	Mode7
Motors for linear movement (RS-485, daisy-chain)	2	UART	P8	37, 38	Mode7
Motors for rotational movement (8-lead bipolar stepper motors, 4-wires each)	16	GPIO	P8	29 – 36, 39 – 46	Mode7
Encoders for rotational motors	4	GPIO	P9	27, 29 – 31	Mode7
Enable bits to H-Bridge Rotational Motor Drivers	4	GPIO	P9	23-26	Mode7

j. Takeover - BRV

Takeover is the determination of which foosman should be tracking the ball. This is a complicated process which depends on the available movement of each individual foosman in relation to the other foosmen in the row, along with the position and direction of travel of the ball. Takeover is a critical aspect of game play since a relatively large amount of time is needed to switch between different players tracking the ball, and the row applying takeover contributes no defense during the takeover process. To visualize the need for takeover, Figure 9 is shown which displays the range of each foosman in the offensive row. In the diagram, F1 to F3 refer to the foosmen in order from left to right, and the subscript R or L indicates the furthest right or left position that the foosman is capable of reaching. Note that no foosman is always capable of tracking the ball, and the regions overlap so that there is not a clearly defined transition point between foosmen.

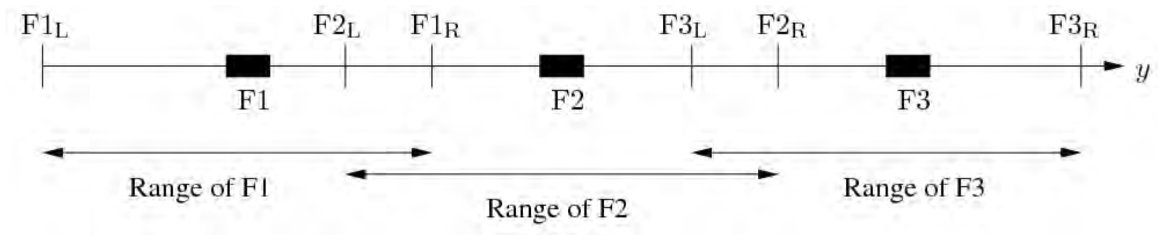


Figure 9: Example of takeover range notation (first row shown)

To control the takeover process, a separate state diagram is used for each row of foosmen as shown in Figure 10 through Figure 13. The defense, midfield, and offensive rows all use similar logic. A takeover occurs when either the foosball is out of the range of the currently tracking foosman, or an adjacent foosman is closer to the desired position and is also capable of moving to that position. For the goalie row, the logic is modified since the center foosman is capable of traversing the entire goal region. The center foosman always tracks the ball unless it is out of his range and the ball is behind the defensive row. Likewise, tracking switches to the center foosman whenever the ball is in front of the defensive row.

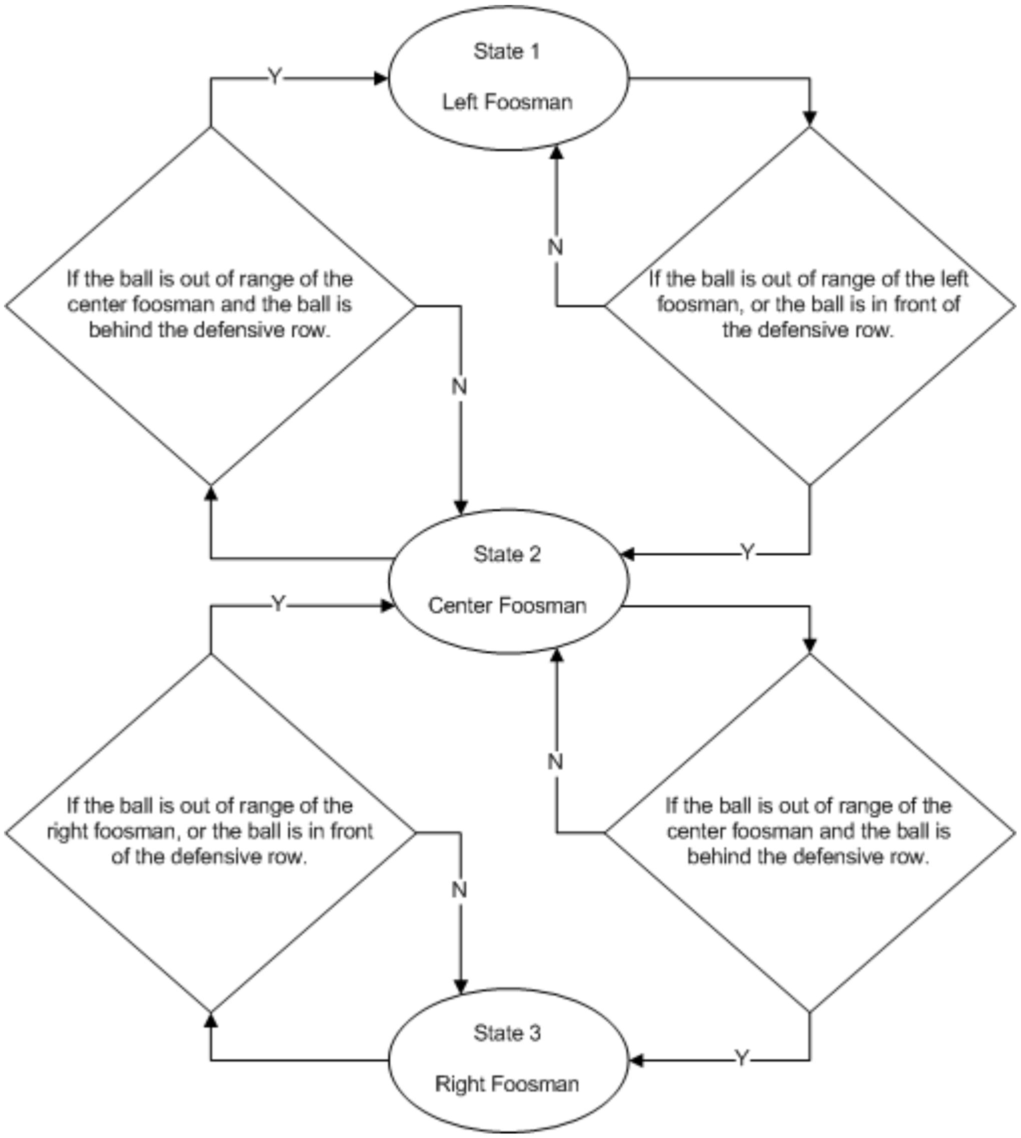

State 1
Left Foosman
If the ball is out of range of the center foosman and the ball is behind the defensive row.
If the ball is out of range of the left foosman, or the ball is in front of the defensive row.
State 2
Center Foosman
If the ball is out of range of the right foosman, or the ball is in front of the defensive row.
If the ball is out of range of the center foosman and the ball is behind the defensive row.
State 3
Right Foosman
Y
N

Figure 10: Takeover Flow Diagram – Goalie

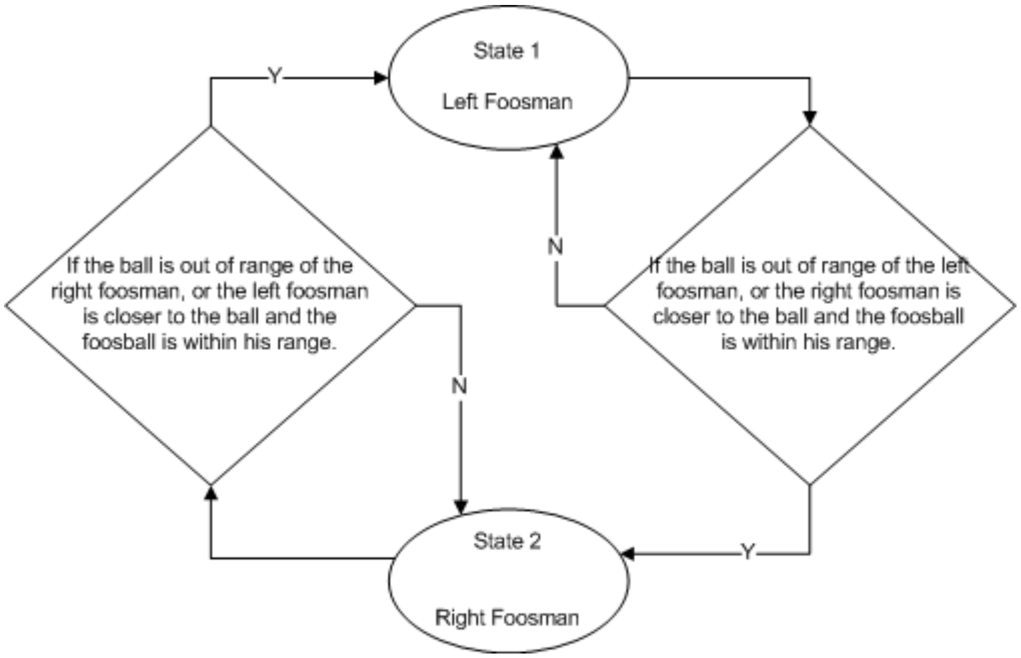

State 1
Left Foosman
Y
N
If the ball is out of range of the right foosman, or the left foosman is closer to the ball and the foosball is within his range.
N
If the ball is out of range of the left foosman, or the right foosman is closer to the ball and the foosball is within his range.
Y
State 2
Right Foosman

Figure 11: Takeover Flow Diagram – Defense

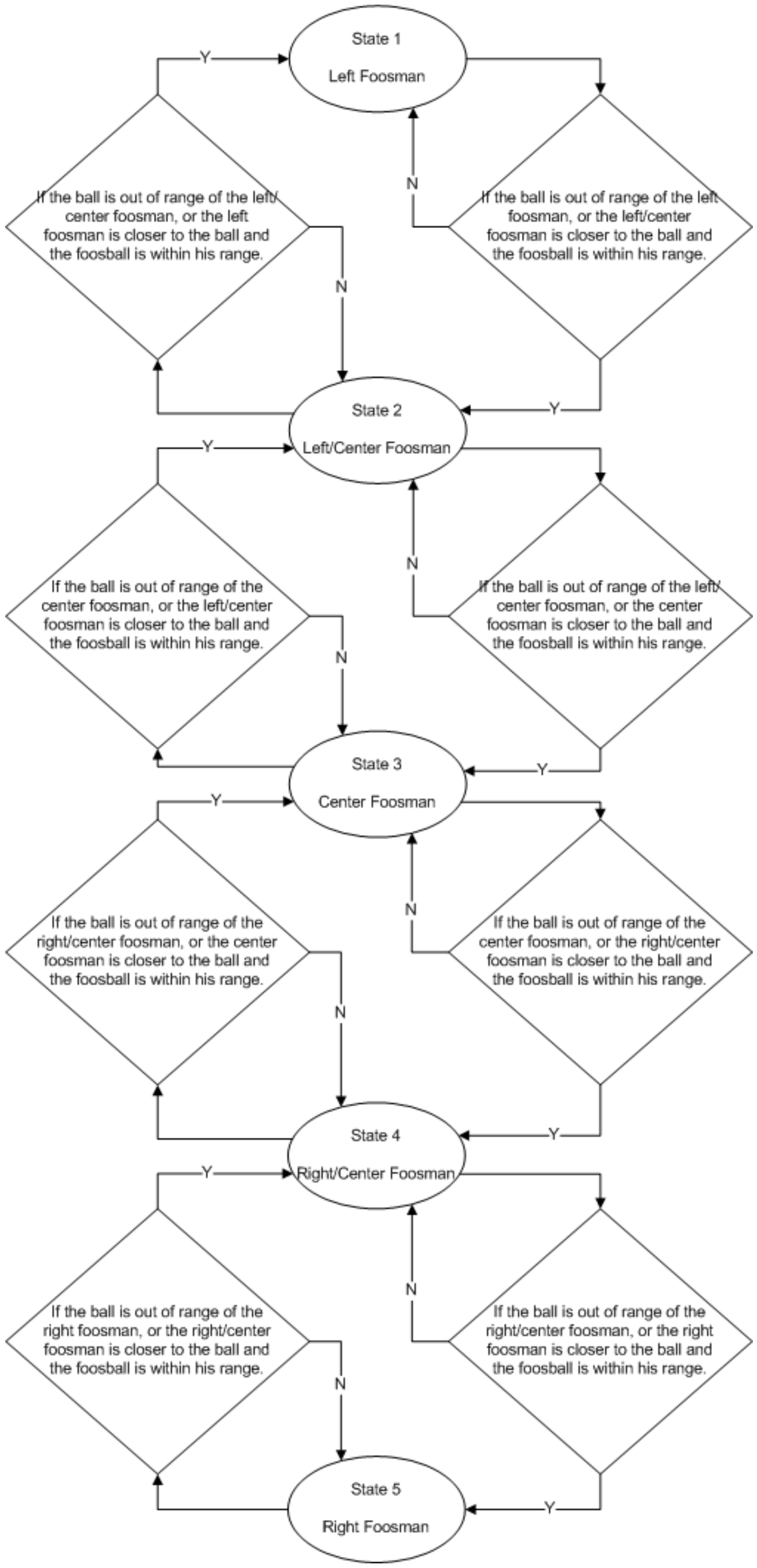

State 1
Left Foosman
If the ball is out of range of the left/center foosman, or the left foosman is closer to the ball and the foosball is within his range.
If the ball is out of range of the left foosman, or the left/center foosman is closer to the ball and the foosball is within his range.
State 2
Left/Center Foosman
If the ball is out of range of the center foosman, or the left/center foosman is closer to the ball and the foosball is within his range.
If the ball is out of range of the left/center foosman, or the center foosman is closer to the ball and the foosball is within his range.
State 3
Center Foosman
If the ball is out of range of the right/center foosman, or the center foosman is closer to the ball and the foosball is within his range.
If the ball is out of range of the center foosman, or the right/center foosman is closer to the ball and the foosball is within his range.
State 4
Right/Center Foosman
If the ball is out of range of the right foosman, or the right/center foosman is closer to the ball and the foosball is within his range.
If the ball is out of range of the right/center foosman, or the right foosman is closer to the ball and the foosball is within his range.
State 5
Right Foosman
Y
N

Figure 12: Takeover Flow Diagram – Midfield

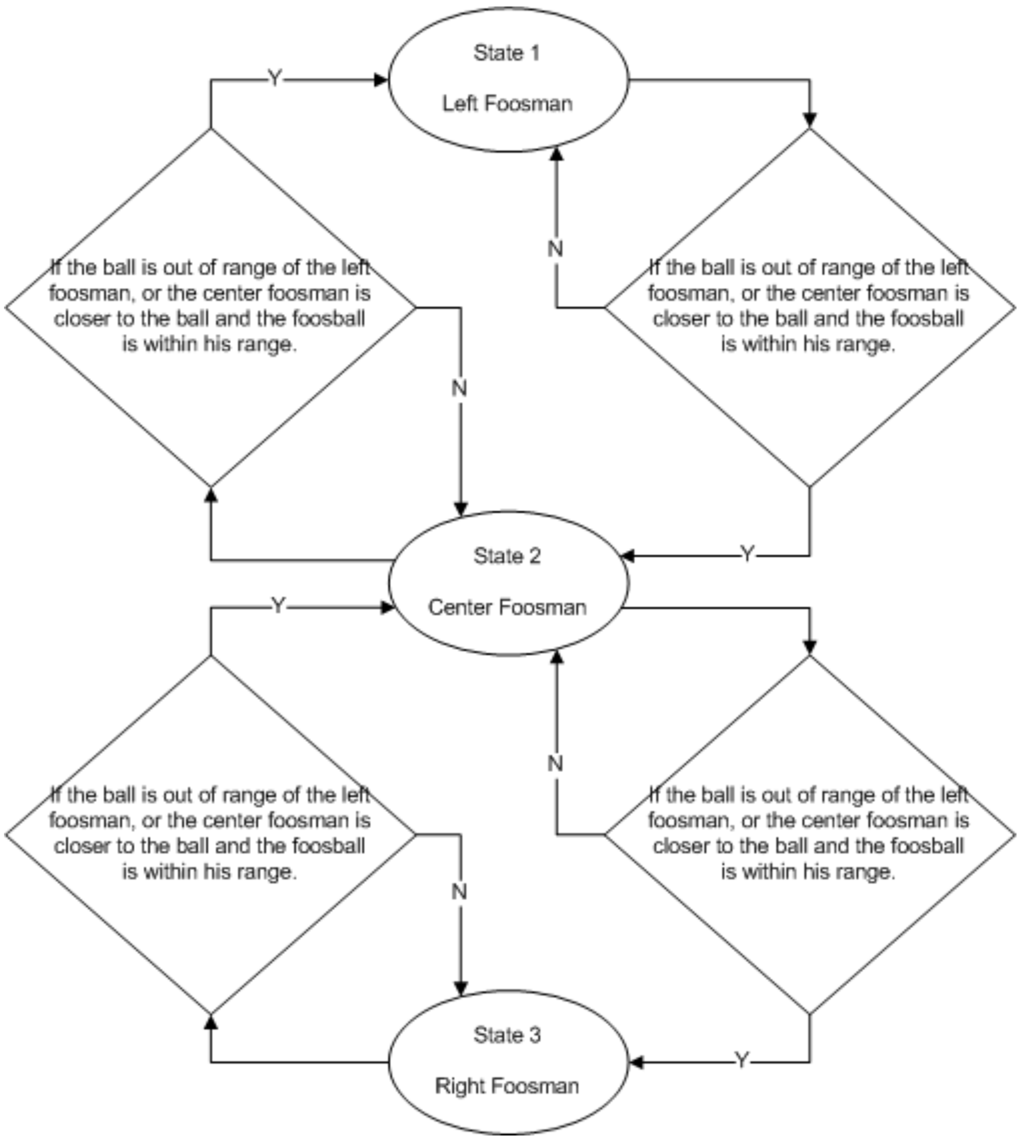

State 1
Left Foosman
If the ball is out of range of the left foosman, or the center foosman is closer to the ball and the foosball is within his range.
If the ball is out of range of the left foosman, or the center foosman is closer to the ball and the foosball is within his range.
Y
N
N
Y
State 2
Center Foosman
If the ball is out of range of the left foosman, or the center foosman is closer to the ball and the foosball is within his range.
If the ball is out of range of the left foosman, or the center foosman is closer to the ball and the foosball is within his range.
Y
N
N
Y
State 3
Right Foosman

Figure 13: Takeover Flow Diagram – Offense

k. Computational Speed - BRV

The speed of the pulsing LEDs must be fast enough so that all groups of sensors are read before the foosball is able to move the distance of the minimum resolution. Otherwise, the foosball could move while the sensors are being read and either no readings or multiple readings could occur, resulting in an incorrect location for the foosball. The time it takes for the ball to move the minimum resolution is

$$t_r = \frac{r}{v_{max}} = \frac{1\ cm}{6.7\ m/s} = 1.49\ ms, \tag{16}$$

so the foosball must be detected at least every 1.49 ms. The AM3359 microprocessor is capable of 2,000 Dhrystone million instructions per second (DMIPS), so it can perform two instructions every nanosecond. The timing diagram for the sensing process is shown in the right side of Figure 14, indicating the total time needed for sensing is 12.144 µs which is well below the required time. Note that the 1 µs delay between turning on the IR LEDs and polling the sensors is due to the combined 900 ns rise time of the LEDs and 5 ns rise time of the phototransistors.

In order to understand the timing of the entire system, the speed of the motors must be known. The worst case scenario shot is from the center of the offensive line of the opponent to the far goal post. Assuming the maximum speed of the ball calculated in Section 3.a and a shot from the offensive line of the opponent, the goalie has

$$t_{goalie\ (required)} = \frac{v_{max}}{d_{offensive\ line\ to\ goalie}} = \frac{6.7\ m/s}{0.292\ m} = 43.6\ ms \tag{17}$$

to move into position to block the shot.

Using the actual system parameters, the maximum time required by the goalie to block the optimal shot is calculated. The shaft of the motors used for linear motion is attached to a pulley which drives the linear rails. The radius of the pulley is calculated in Equation (4) to be 0.0167 m. Since the time required to block a shot is relatively short, the peak torque of the motors is used in calculations, which is $T_{peak} = 1.24\ N\ m$. The maximum force exerted by the motors on the pulley is then calculated as

$$F_{max} = \frac{T_{peak}}{r} = \frac{1.24\ Nm}{0.0167\ m} = 74.2\ N. \tag{18}$$

Assuming a constant acceleration by the motors while moving to block the shot, the velocity of the goalie is given by

$$v_{goalie} = \int_0^{t_{goalie}} a_{goalie}\, dt = \int_0^{t_{goalie}} \frac{F_{max}}{m_{rod}} dt = \frac{F_{max}\, t_{goalie}}{m_{rod}} \tag{19}$$

where m_{rod} is the mass of the rod. Integrating the velocity, the displacement of the goalie is calculated as

$$d_{goalie} = \int_0^{t_{goalie}} v_{goalie}\, dt = \int_0^{t_{goalie}} \frac{F_{max}\, t}{m_{rod}} dt = \frac{F_{max}\, t_{goalie}^2}{2\, m_{rod}}. \tag{20}$$

Using the defensive game strategy described in Section 3.b, the maximum distance required by the goalie to block a shot is $d_{goalie} = 1.69$ cm. Thus, the time it takes the goalie to move into position is

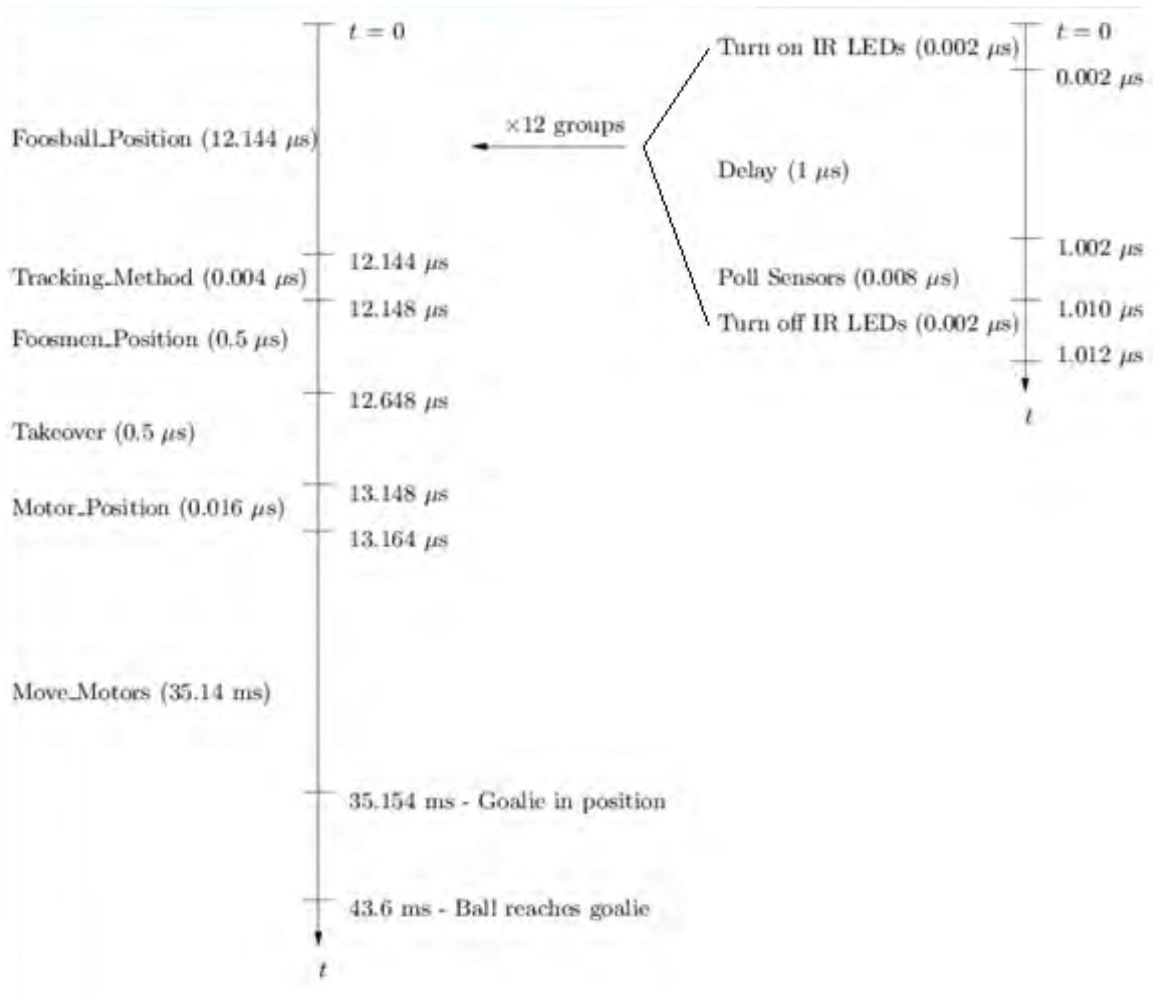
$$t_{goalie} = \sqrt{\frac{2\, d_{goalie}\, m_{rod}}{F_{max}}} = \sqrt{\frac{2(0.0169\, m)(2.268\, kg)}{74.2\, N}} = 32.14\, ms. \tag{21}$$

The motors do not move instantly when given a command. There is about a 3 ms delay between when the command is sent until motion begins. This timing is allocated as follows:

- Sending data (1 ms)
- Processing command (1 ms)
- Load command into motion generator (1 ms).

Figure 14 shows the complete timing diagram. The times required by each function are determined by multiplying the estimated number of instruction cycles in the simulation code in Section 3.p by the speed of the microprocessor. The timing diagram illustrates that the foosbot is capable of stopping the optimal shot, given the following assumptions:

- The players are in the optimal defensive position prior to the shot (no takeover can be taking place at the time of the shot).
- The speed of the ball is 6.7 m/s.
- The acceleration of the motors is constant with peak torque.
- The sensing process begins as soon as the ball triggers a sensor (the program is not in the middle of the loop when the shot is taken).



t = 0
Foosball_Position (12.144 μs)
12.144 μs
Tracking_Method (0.004 μs)
12.148 μs
Foosmen_Position (0.5 μs)
12.648 μs
Takeover (0.5 μs)
13.148 μs
Motor_Position (0.016 μs)
13.164 μs
Move_Motors (35.14 ms)
35.154 ms - Goalie in position
43.6 ms - Ball reaches goalie
t
×12 groups
Turn on IR LEDs (0.002 μs)
t = 0
0.002 μs
Delay (1 μs)
1.002 μs
Poll Sensors (0.008 μs)
1.010 μs
Turn off IR LEDs (0.002 μs)
1.012 μs
t

Figure 14: Timing diagram for shot with ball travelling 6.7 m/s.

I. Power Requirements - JAZ

The IR sensor power dissipation is now represented. Two IR Sensors were obtained and driven at full intensity while the current through the elements was monitored. A maximum of 9mA was measured on the ammeter (which is verified in the next equation). However, this only accounts for two sensors. Using $N_{LED,x}$ and $N_{LED,y}$, as shown on Page 7, it can be calculated that the total current for the IR Sensors will be given as

$$I_{LED,x} = \frac{V}{R_{LED,EQ}} = \frac{V}{R_{LED}/N_{LED}} \tag{22}$$

Using Ohm's Law, the equivalent resistance of the parallel configuration of sensors is

$$\frac{1}{R_{LED,EQ}} = \frac{1}{R_{LED,1}} + \frac{1}{R_{LED,2}} + \cdots + \frac{1}{R_{LED,N}} = \frac{N_{LED}}{R_{LED}}, \tag{23}$$

thus $R_{LED,EQ} = R_{LED}/N_{LED}$. Similarly, $I_{phototransistor,EQ} = R_{phototransistor}/N_{phototransistor}$. This shows the total current draw on all of the sensors; however, there will only be one group of 16 sensors powered at one point in time (due to the pulse groups described above in 3.g "Time Division Multiplexing". That is 16 LEDs and 16 Phototransistors. Total current due to LEDs is then calculated as

$$I_{LED,TOTAL} = \frac{V}{R_{LED,EQ}} = V\frac{16}{R_{LED}} = 3.3V * \frac{16}{150\ \Omega} = 352mA \tag{24}$$

Similarly for the phototransistors,

$$I_{phototransistor,TOTAL} = \frac{V}{R_{phototransistor,EQ}} = V\frac{16}{R_{phototransistor}}$$

$$= 5 * \frac{16}{470k} = .170mA$$

The total sensor power dissipation is then

$$\begin{aligned} P_{sensor,TOTAL} &= P_{LED} + P_{phototransistor} \\ &= V_{LED}I_{LED} + V_{phototransistor}I_{phototransistor} \\ &= 1.7V * .352A + 5V * .000170A = 599.25mW, \end{aligned} \tag{25}$$

Rotational motor power consumption will be limited to $P_{MR,Rated} = 7.2W$ maximum at rated voltage and current. Although there are four rotational motors, these NEMA17 standard steppers will only be utilized one at a time. Along with the rotational motors come the encoders and H-Bridge integrated circuit (IC). The encoder power and H-Bridge power consumption is $P_{ENCODERS} = 0.54W$ (includes all four) and

$P_{HBRIDGE} = 2.075W * 4 = 8.3W.$

Using 12 satellite "Pin Boards" adds some power consumption as well. There are two multiplexers/de-multiplexers, and four comparator ICs per Pin Board. That is

$$\begin{aligned} P_{Pinboard,TOTAL} &= (P_{MUX} + P_{DEMUX} + P_{COMP}) * 12 \\ &= (.75 + .75 + .0056) * 12 = 18.07W \end{aligned} \tag{26}$$

According to the Beagle Bone Processor Board datasheet the expected power consumption is $P_{BB} = 5W$ at maximum.

The main power dissipation will be in the linear motors, as expected. Each Myostat Motion NEMA23 Stepper Motor can use up to 30 watts, or $P_{ML} = 30W * 4 = 120W.$ These four motors will be supplies with their own dedicated 240 watt power supply.

Using the values listed above, the total power requirement of the system is

$$\begin{aligned} P_{FOOSBOT} = P_{sensor,TOTAL} + P_{MR,Rated} + P_{Pinboard,TOTAL} + P_{BB} \\ + P_{ML} \\ = .59925W + 7.2W + 0.54W + 8.3W \\ + 18.07W + 5W + 120W = 159.71W \end{aligned} \tag{27}$$

This power consumption expectation is based on using all four linear motors at their peak performance capabilities. If, after testing, we cannot supply enough power to the system, the control algorithms will be tailored such that only two linear motors will be driven simultaneously.

m. Level 0 Hardware - JAZ

The level 0 hardware block diagram is shown in Figure 15. This gives a general overview of system components and how the blocks interact with each other. Power is supplied through a standard 120 VAC supply for the entire system. The microprocessor reads inputs from the optical sensors to determine the position of the foosball. This information is then processed to determine the desired position of the foosmen. Using the motor drivers, the rotational and linear motors are then driven to the optimal position for foosball game play. Each function is described in detail in Table 2.

Due to changing of the hardware design, the level 1 and level 2 hardware block diagrams are omitted. More detail is supplied in the level 3 diagram in Section 3.n.

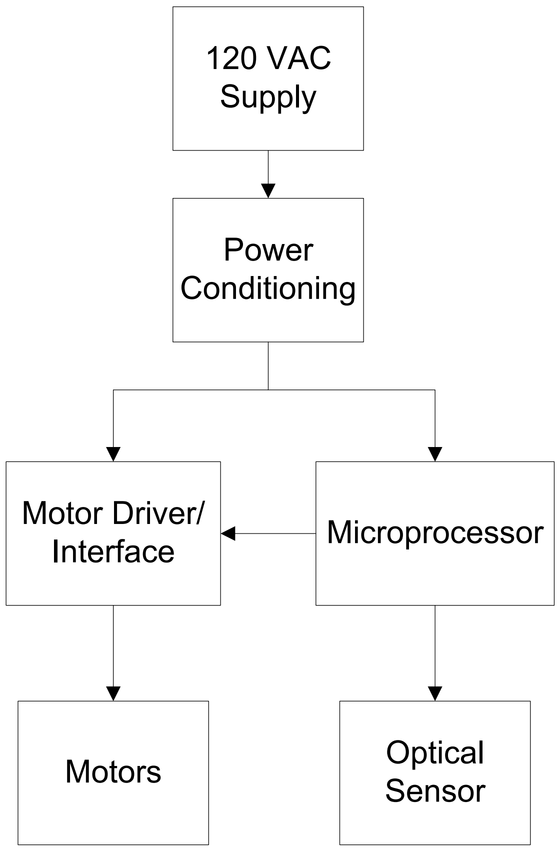


Figure 15: Level 0 Hardware Block Diagram

Table 2: Level 0 Hardware Functional Requirements Table

Module	Power Conditioning
Inputs	- 120 VAC Supply
Outputs	- Suitable power for each respective component (Motor Driver/Interface and Microprocessor)
Description	Provides power to each component.

Module	Motor Driver/Interface
Inputs	- Necessary Power to drive motors - Motor Commands from Microprocessor
Outputs	- Current Output to Motor
Description	Provides power to motors according to instructions from processor.

Module	Microprocessor
Inputs	- Inputs from Sensors
Outputs	- Motor Commands to Motor Driver/Interface
Description	Manages I/O from the optical sensor and sends commands to the motor driver/interface device.

Module	Motors
Inputs	- Current Managed by the Motor Driver/Interface
Outputs	- Resulting Mechanical Movement/Operation
Description	The Motors are the final output of the system which will react and control ball movement.

Module	Optical Sensor
Inputs	- Power Supply from power conditioner
Outputs	- Required data for foosball tracking and motor position
Description	The optical sensor provides the Microprocessor with the required inputs and information it needs to make decisions and control I/O.

n. Level 3 Hardware – JAZ

The final hardware layout strongly differs from the original design in multiple ways. The Foosbot team was fortunate enough to receive donations from Myostat Motion, Inc as well as Macron Dynamics. These sponsors have provided a power supply, linear motors, cabling, as well as linear belt travels.

Power Supplies
As shown in Figure 16 the system will still be receiving power from a 120VAC source; however, a standard surge protector strip has been chosen for its simple switch and low cost connection. The power will be supplied to the linear motors via the IDEC PS5R24 240 Watt power supply (PS1) supplied by Myostat Motion, Inc. This power supply is the dedicated power supply to the Myostat Cool Muscle Linear Motors.

The second power supply (PS2) is a HPC-300-101 300W computer power supply. It has a +/-12V, +/-5V, and +/-3.3V tap which will be used for powering the remainder of the system components.

Sensing Hardware
In Figure 17 and Figure 19 it can be seen that there will be 12 satellite printed circuit boards (PCBs) installed under the foosball table. These boards provide the interconnection between the Beagle Bone microprocessor board (BB) and the sensing phototransistors and IR LEDs. Each of the twelve “Pin Boards” is responsible for one section of the foosball table. Within that section of the table are 16 sensors (one from each group). Those 16 sensors will be turned on one-by-one with a 4 bit address given by the BeagleBone. This address is daisy chained and is native to every Pin Board via the J3 and J4 connectors seen in Figure 20. This address will be sent to the de-multiplexer (DMX) on each Pin Board which acts essentially as a switch. This same address is fed to the multiplexer (MX) to read the phototransistor corresponding to the IR LED which is currently active. Refer to Section 3.k for a detailed description of this process. It may also be noted that a comparator is used to monitor the phototransistor voltage with a 2.7V reference. This is used to obtain a digital signal at the Beagle Bone Processor. If this signal were kept as an analog signal, the ARM Cortex Processor would have to do an A/D conversion before continuing with more instructions which slow the foosball tracking process.

Figure 21 shows how each of the four rotational motors is controlled. By using the “H” bridge IC, the BeagleBone Processor board can control 4 pins to drive the coils of each NEMA17 Standard Stepper motor. Budget permitting, an encoder will be attached to each of the rotational motors to provide feedback to the Beagle Bone Processor. These encoders offer 200 pulses per revolution which matches the 1.8 degree step per pulse on the hybrid stepper motors.

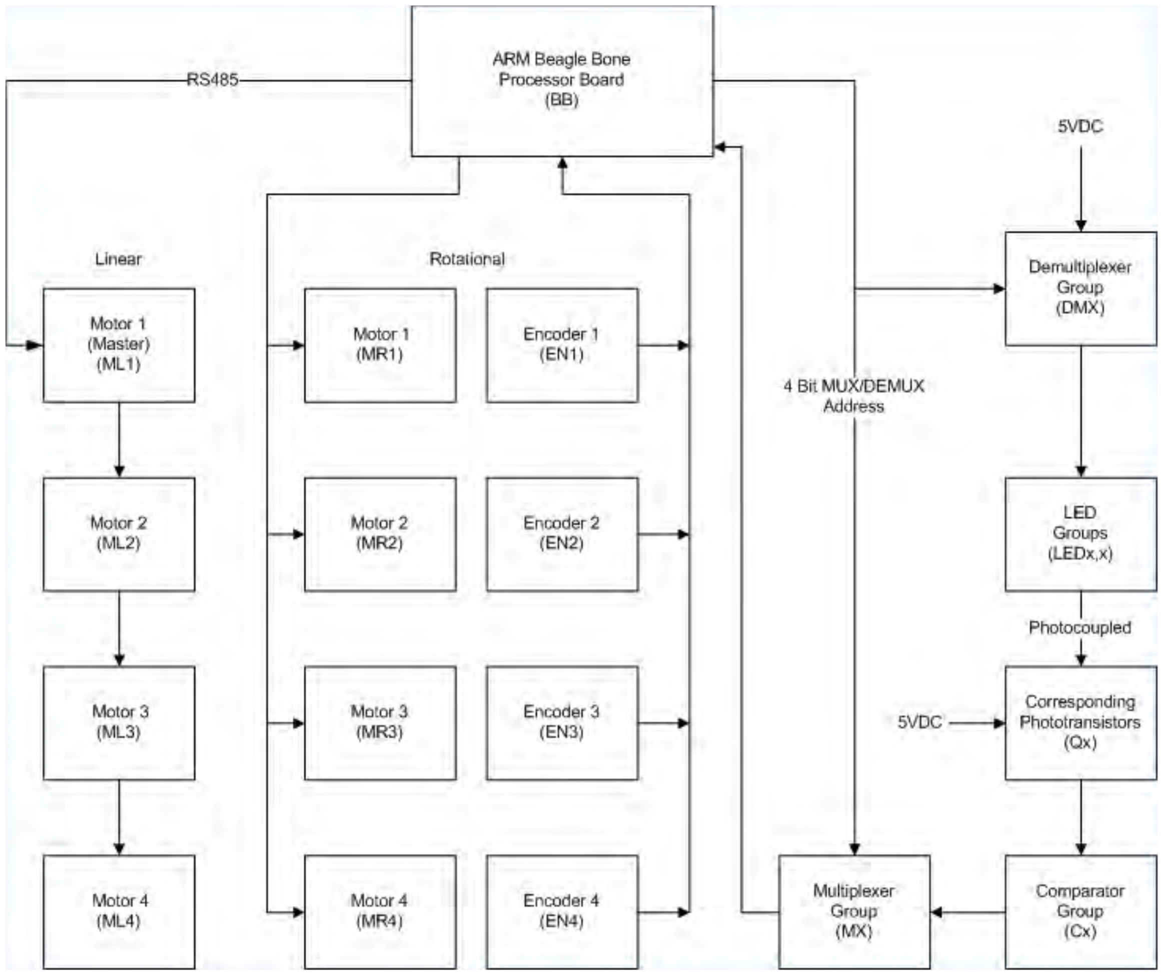


Figure 16: Level 3 Hardware Block Diagram (Motor Control)

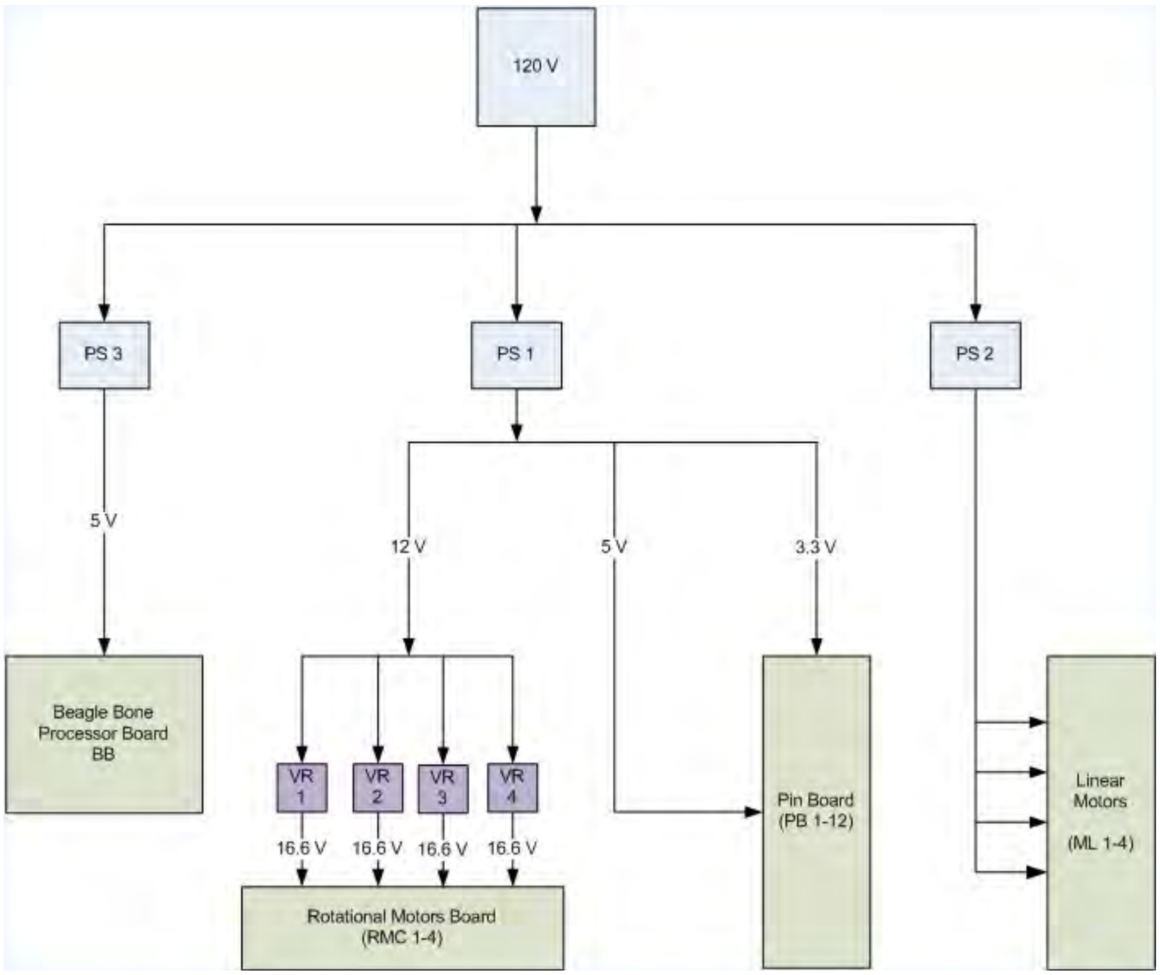


Figure 17: Hardware Block Diagram (Power)

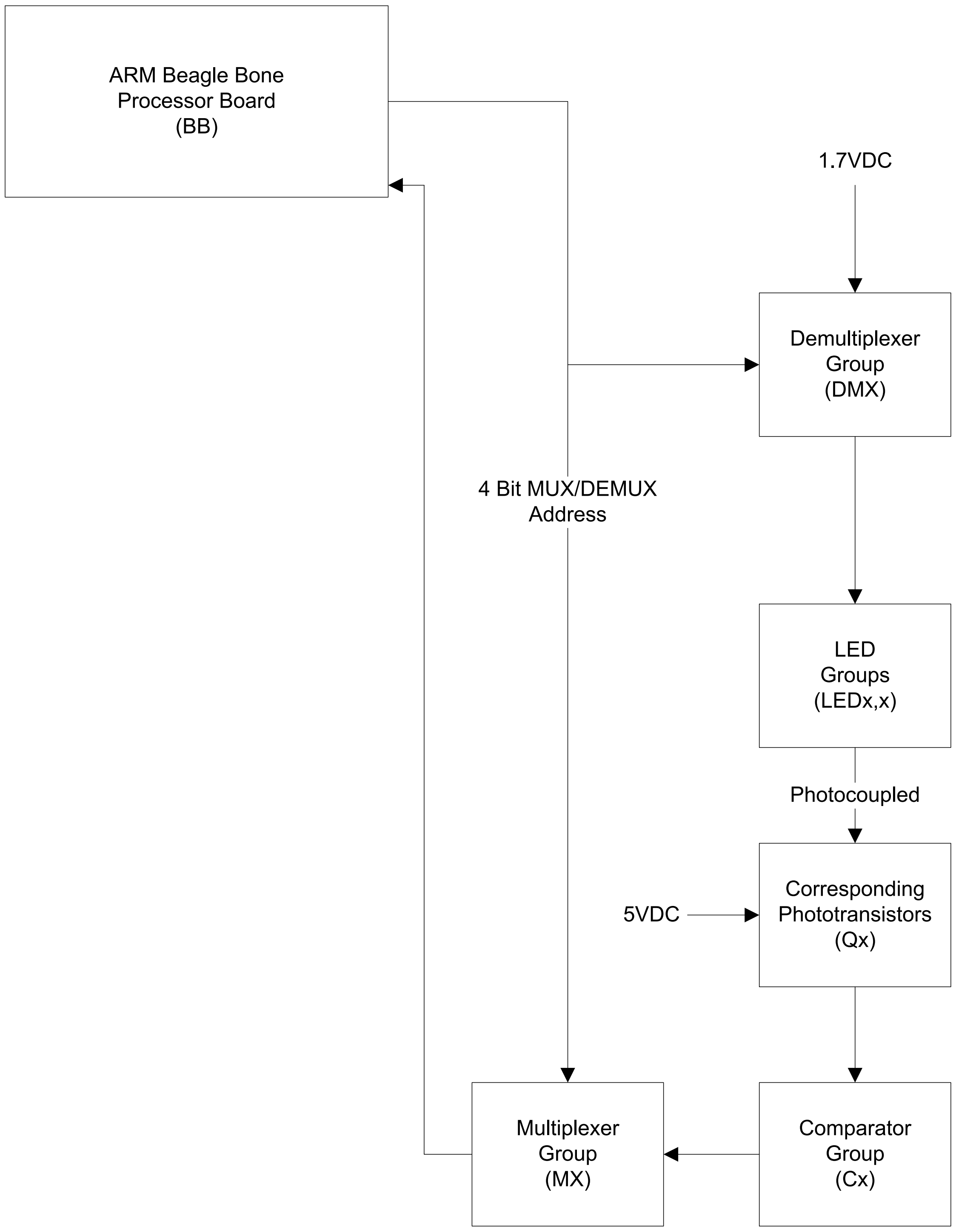


Figure 18: Level 3 Hardware Block Diagram (Sensing)

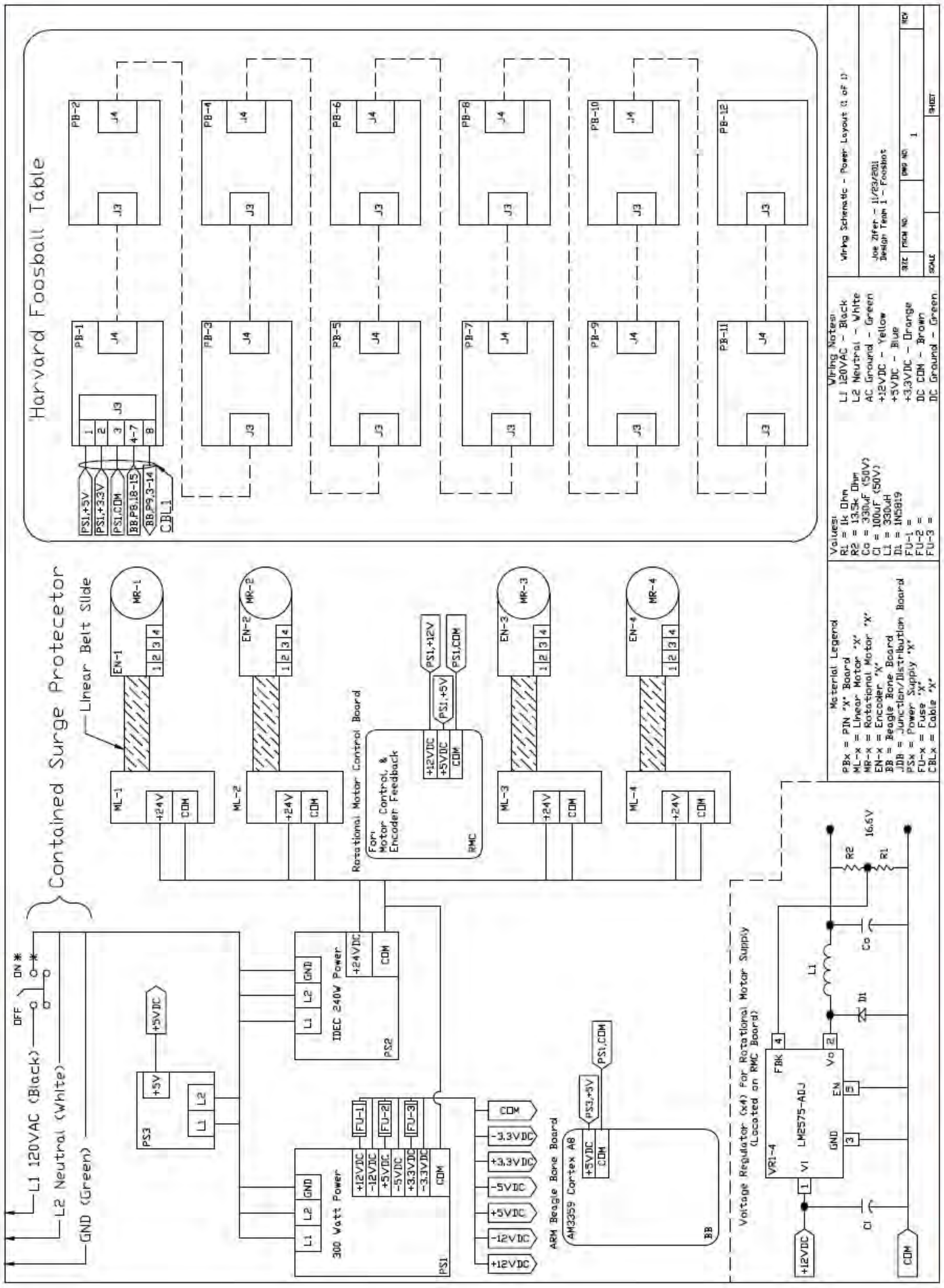


Figure 19: Power Layout

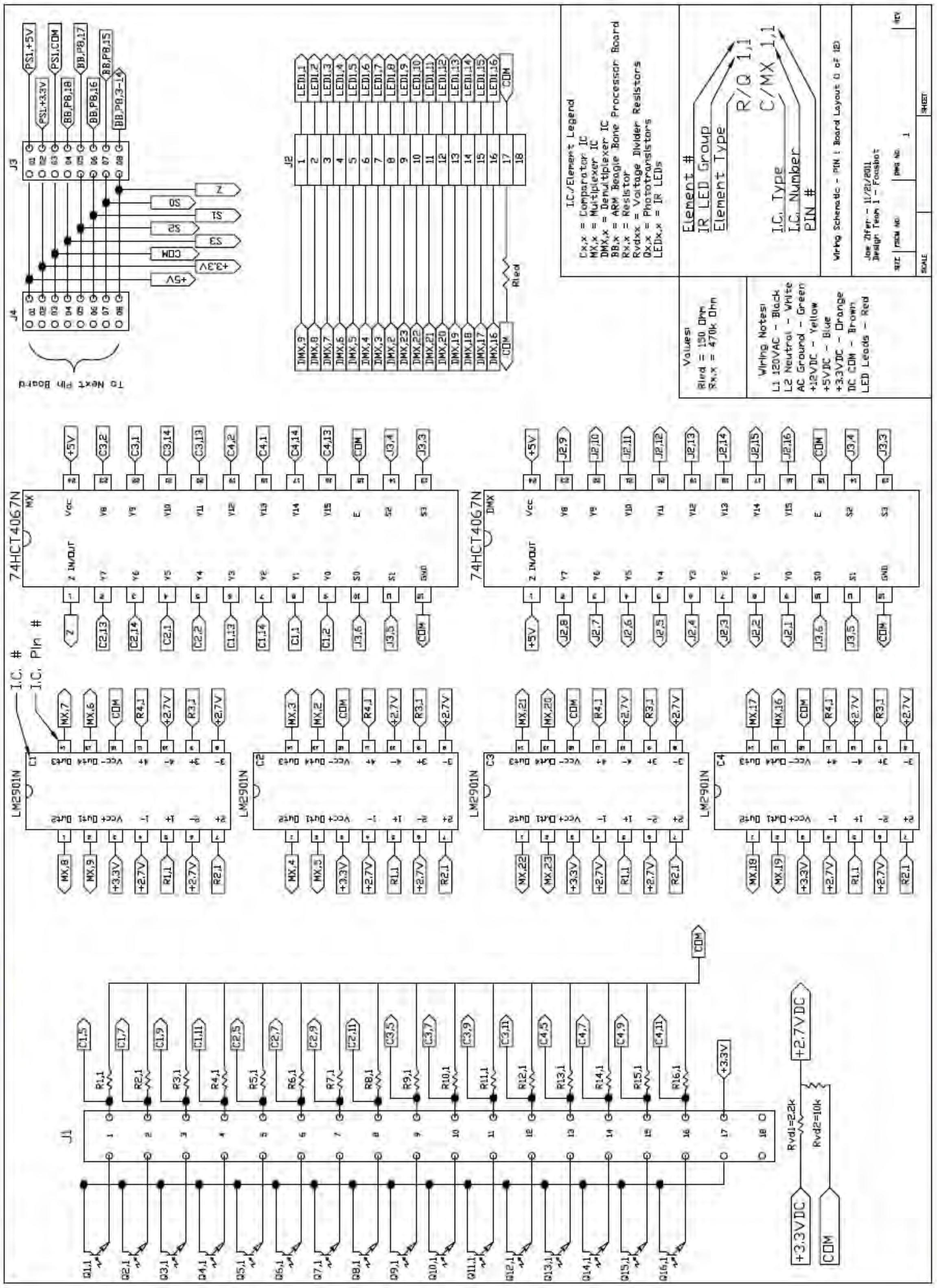


Figure 20: Sensing Pin Board Schematic

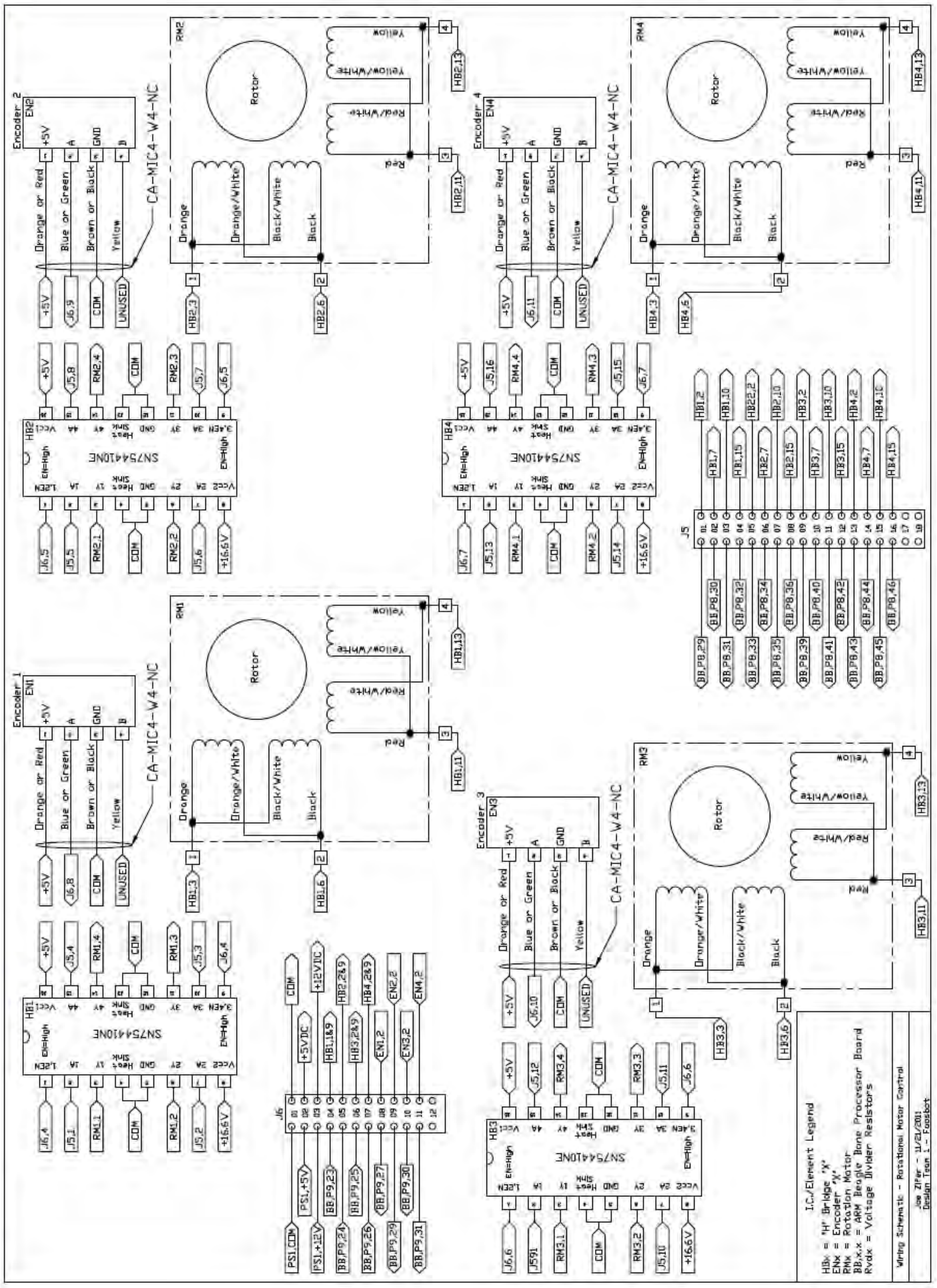


Figure 21: Rotational Motor Control Schematic

Table 3: Level 3 Hardware Functional Requirements Table

Power Supply 1

Module	PS1
Inputs	- 120 VAC Supply
Outputs	- 12VDC, 5VDC, 3.3VDC
Description	The Enlight HPC-300-101 300Watt power supply is responsible for delivering power to all components excluding the BeagleBone and the linear position motors. The power supply can support up to 13A at +12VDC.

Power Supply 2

Module	PS2
Inputs	- 120 VAC Supply
Outputs	- 24VDC for linear position motors (MLs)
Description	The IDEC PS5R24 is a 240W 10A power supply which will be dedicated to the 30W linear motors.

Power Supply 3

Module	PS3
Inputs	- 120 VAC Supply
Outputs	- 5VDC for BeagleBone
Description	The Volgen 5W KTPSO5 power supply is the dedicated power supply to the BeagleBone.

BeagleBone

Module	BB
Inputs	- 5VDC - 4 bit Encoder data - Phototransistor Digital Signals
Outputs	- 4 Bit Multiplexer and De-multiplexer address (S0-S3) - Motor Data via RS485 for linear position - 4 Bit Motor Data for rotational position
Description	The BeagleBone is the main processor board used for all communication and decision making. The BeagleBone contains all control algorithms and timing of the system.

Linear Position Motors

Module	ML(1-4)
Inputs	- 24VDC power from PS2 - Position command via RS485
Outputs	- Rotational movement for linear position of foosmen
Description	The four Myostat Cool Muscle CM1-X-23L20s are 30W stepper motors. They are used to move the linear belt slides to position the foosmen at the desired location. The BeagleBone sends this position data.

Voltage Regulator

Module	VR(1-4)
Inputs	- 12VDC from PS1
Outputs	- 16.6VDC to H-Bridge Driver
Description	The LM2575-ADJ regulator simply boosts the 12VDC to 16.6VDC to drive the H-Bridge IC.

H-Bridge Driver

Module	HB(1-4)
Inputs	- 16.6VDC from VR(1-4) - 4 bit pulse sequence from BeagleBone
Outputs	- 16.6VDC to Rotational Position Motors
Description	The SN754410 is a quad half H-Bridge which uses the high and low voltage pulses from the BeagleBone to excite the Rotational Position Motor in the correct direction. Note: This IC is only shown in the Rotational Motor Control Board Schematic.

Rotational Position Motors

Module	MR(1-4)
Inputs	- 16.6VDC Power via Voltage Regulator (1-4) - Step commands and direction via 4 bit address from BeagleBone
Outputs	- Rotational movement for angular position of foosmen
Description	The four HT17-274D stepper motors from Applied Motion serve as the angular position motor for the foosmen. They receive pulse sequences from the BeagleBone to actuate the rotor.

Rotational Motor Encoder

Module	EN(1-4)
Inputs	- 5VDC Power from PS1 - Optically coupled pulses from rotor
Outputs	- 2.4V pulses to BeagleBone
Description	The E8P OEM Optical Kit Encoder from US digital will directly mount to the double shaft of the rotational position motors. This will provide 200 pulses per revolution for angular position tracking.

De-multiplexer

Module	DMX
Inputs	- 5VDC - 4 bit address from BeagleBone (S0-S3)
Outputs	- 5VDC to corresponding IR LED
Description	The de-multiplexer on each Pin Board sends power to the desired LED based on the 4 bit address from the BeagleBone.

Phototransistors

Module	Qx,x
Inputs	- 3.3VDC - IR excitation from optically coupled LED

Outputs	- 3.3VDC to comparator
Description	The phototransistors detect the presence of the foosball. When an IR signal is seen from the corresponding LED, a 3.3VDC signal is sent to the comparator. When the ball interrupts the signal, the voltage drops to near zero. See Section 3.d for a plot of the phototransistor output.

Comparator

Module	Cx,x
Inputs	- 3.3VDC signal from phototransistors - 2.7VDC reference voltage from voltage divider
Outputs	- 3.3VDC to multiplexer
Description	The comparator looks at the incoming voltage from the phototransistors with respect to the reference voltage. If a voltage drop is seen below 2.7VDC, the comparator output will drop to 0V which signals the presence of the foosball.

Multiplexer

Module	MX
Inputs	- 4 Bit Address (S0-S3) - 3.3VDC
Outputs	- 3.3VDC - 0 VDC
Description	The comparator voltage being measured depends on the address from the BeagleBone.The multiplexer then relays the comparator voltages to the BeagleBone.

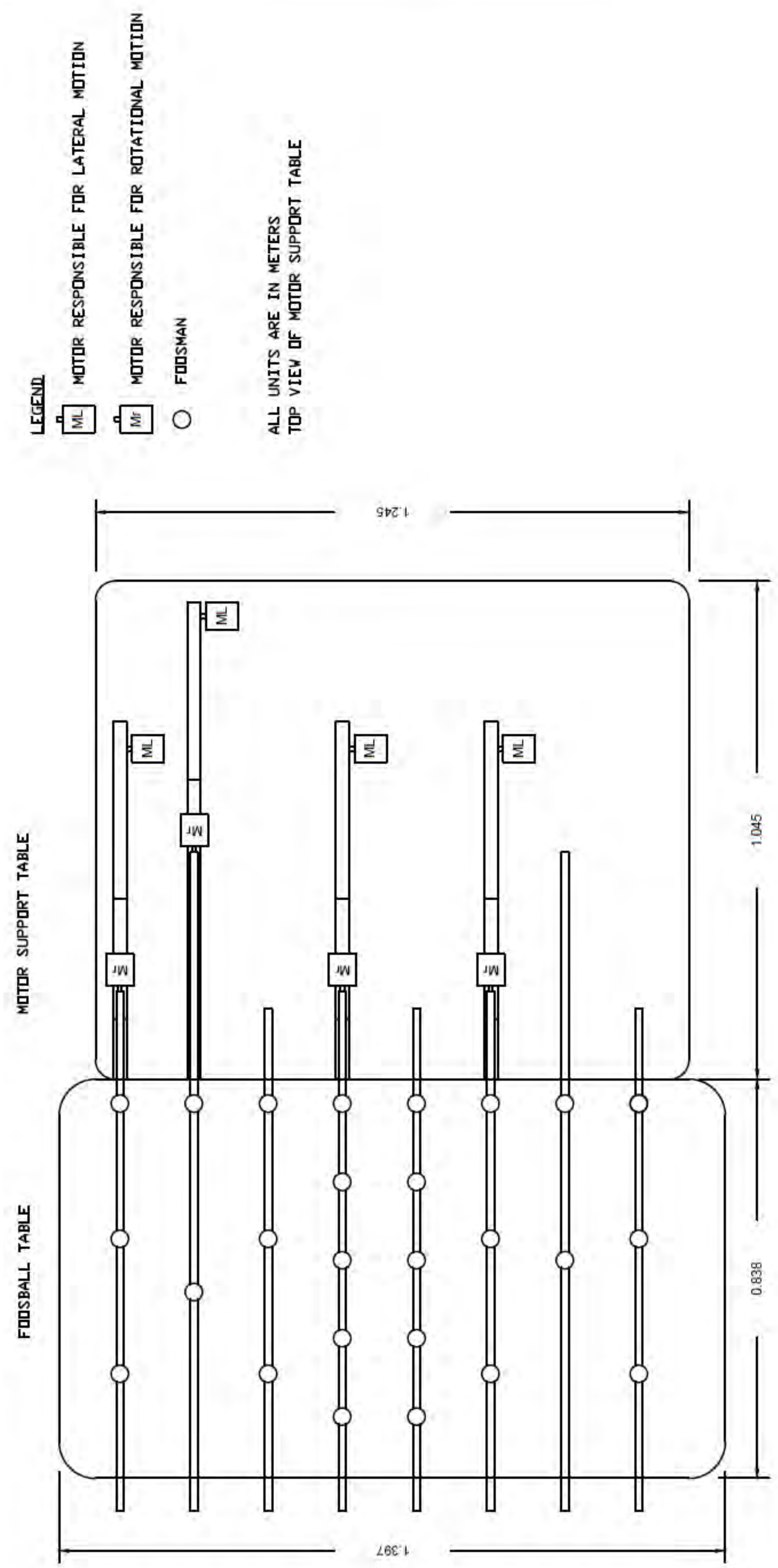

LEGEND
ML MOTOR RESPONSIBLE FOR LATERAL MOTION
Mr MOTOR RESPONSIBLE FOR ROTATIONAL MOTION
FOOSMAN
ALL UNITS ARE IN METERS
TOP VIEW OF MOTOR SUPPORT TABLE
MOTOR SUPPORT TABLE
FOOSBALL TABLE
ML
Mr
1.245
1.045
0.838
1.397

Figure 22: Physical arrangement of motors and slides with respect to foosball table.

o. Level 0 Software – BRV

The software for the foosbot is responsible for detecting the position of the foosball on the table and moving the motors accordingly to simulate the game of foosball. The level 0 software block diagram is shown in Figure 23 which illustrates this process, with the function of each block shown in Table 4. With the known position of the foosball, the foosmen may be positioned to block the goal for defense, and may also rotate to swing and hit the foosball for offense. Different strategies for positioning the foosmen may be implemented in the software, such as defensive or trajectory tracking of the foosball. The final outputs of the software are commands to the motors to move them into the desired positions.

Due to changing of the software design, the level 1 and level 2 software block diagrams are omitted. More detail is supplied in the level 3 diagram in Section 0.

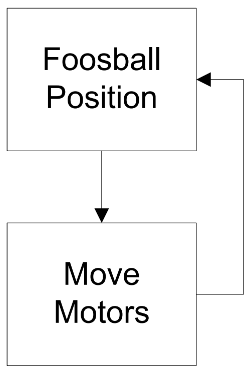


Figure 23: Level 0 Software Block Diagram

Table 4: Level 0 Software Functional Requirements Table

Module	Foosball Detection
Inputs	- Phototransistor voltage levels
Outputs	- Coordinate of foosball (x,y)
Description	Reads the output from the optical sensors and converts the information into the coordinate of the foosball.

Module	Move Motors
Inputs	- Coordinate of foosball (x,y)
Outputs	- Motor commands
Description	Utilizes various strategies of game play to determine where to position the foosmen based on the position of the foosball.

p. Level 3 Software - BRV

All of the software for the project is implemented on the AM3359 processor as described in Section 3.i. The microprocessor is responsible for polling the sensors to determine the location of the foosball, calculating the desired positions of the foosmen, and then sending motor commands to each of the motors to obtain the desired motion.

First, the table parameters are loaded. This includes all dimensions of the table, foosmen, and ball as well as the number of foosmen in each row. Since timing is an important factor in the foosbot, any calculations that need to be done and are always the same are also performed in this function and stored in memory. Examples of this include the maximum position of each motor and the range of positions of each foosman (needed for takeover). This reduces the amount of required calculations while playing foosball.

Initialization then occurs where all required registers and I/O ports are setup. This function also includes calibration of the motors for linear and rotational motion.

The goal interrupt service routine (ISR) is used to keep track of the score. When the sensors located in the goal indicate that a team has scored via the interruption of the path between the LED and the phototransistor, the score is updated. Also, the mode of play may be modified in this function depending on the score of the game. A simple way of modifying the mode is to change the acceleration of the motors to make the foosbot easier/harder to accommodate every level opponent. This will be implemented if time permits.

Once all initialization is done, the main loop is executed which consists of the following:

- Detecting the position of the foosball,
- Determining the tracking method to be used,
- Calculating the foosmen position based on the tracking method,
- Applying takeover to determine which foosman in each row should track the ball,
- Calculating the motor positions required to put the tracking foosmen in the desired locations,
- and moving the motors accordingly.

This process is then repeated until the game is over.

The design of the code that is used to determine the position of the foosmen can become quite complex, and the final result is difficult to visualize. For this reason, Matlab code is written to simulate the calculations required by the microprocessor. Although not every part of the system can be simulated, such as polling the sensors and sending motor commands to the motors, the rest of the system can accurately be simulated including determination of the tracking method, desired foosmen positions, takeover, and desired motor positions. The Matlab code used for the simulation is included below the functional requirements table for each corresponding module, shown in Table 5- Table 15. In several situations, the code used for simulation differs slightly from that of implementation. For these situations, the code which more accurately reflects that which will be used for implementation is shown.

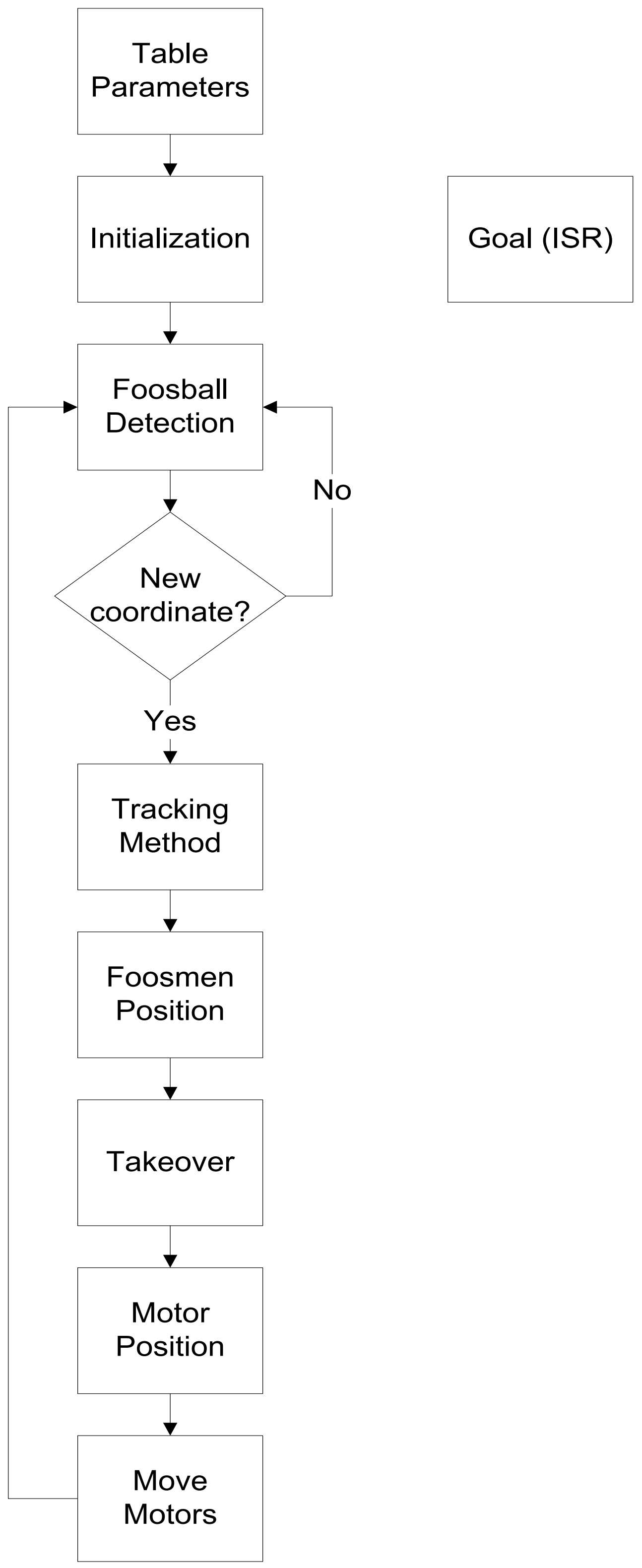


Figure 24: Level 3 Software Block Diagram

Table 5: Function - Main

Module	main
Inputs	None
Outputs	None
Description	This is the main function for the microprocessor which controls the flow of the program. Table parameters and initialization are performed, and then the main loop of sensing the position of the foosball, determining the tracking method, determining the optimal foosmen positions, calculating takeover, calculating the desired motor positions, and then commanding the motors to move to the desired positions. To optimize for speed, calculations are only performed if the position of the foosball has changed from the previous iteration. The following functions are called: - Table_Parameters - Initialization - Foosball_Position - Tracking_Method - Foosmen_Position - Takeover - Motor_Position - Move_Motors
How Invoked	Automatically called by microprocessor.
Max Time	The timing required to complete one iteration through the main loop depends on the position where the motors are being commanded to move. See Section 3.k for timing details.

```
function [] = main()

% Load table parameters and dimensions
Table = Table_Parameters();

% Initial conditions
score = [0,0];           % Score [foosbot, opponent]
x1    = .5*Table.x_max;  % Initial x-coordinate of ball
y1    = .5*Table.y_max;  % Initial y-coordinate of ball
motor = zeros(1,4);      % Initial motor positions

% Main loop
while (true)

    % Determine the current coordinate of the foosball
    [x_b,y_b] = Foosball_Position(x1,y1);

    % If the position of the ball has changed since the last iteration.
    if (x_b != x1 || y_b != y1)

        % Determine the tracking method
        Table = Tracking_Method(Table,x1,x_b);

        % Calculate optimal defensive positions
        y = Foosmen_Position(Table,x1,y1,x_b,y_b);

        % Determine takeover for each row
        Table = Takeover(Table,x_b,y,motor);

        % Calculate desired motor position
        motor = Motor_Position(Table,y,motor);

        % Command the motors to move to the desired positions
        Move_Motors(motor,y_b);

        % Store coordinates of the foosball.
        x1 = x_b;
        y1 = y_b;

    end
end

end
```

Table 6: Function - Initialization

Module	Initialization
Inputs	None
Outputs	None
Description	Initializes necessary registers on the microprocessor, communication ports for the motors, etc. Also resets the score to zero and initializes the current position of the foosball.
How Invoked	Called by main function before the main loop.
Max Time	The function is not called in the main loop, so the timing information for this function is not critical to the performance of the system.

Table 7: Function - Interpolation

Module	Interpolation
Inputs	xi – x-coordinate at which to evaluate the interpolating function. (x1,y1) – First point. (x2,y2) – Second point.
Outputs	yi – y-coordinate of the interpolating function at xi.
Description	Performs linear interpolation between the points (x1,y1) and (x2,y2), and evaluates the function at the point xi.
How Invoked	Called from the Foosmen_Position function.
Max Time	6 instructions = 0.003 μs

```
function [yi] = Interpolate(xi,x2,y2,x1,y1)

yi = y1 + (xi-x1)*(y2-y1)/(x2-x1);

end
```

Table 8: Function - Table_Parameters

Module	Table_Parameters
Inputs	None
Outputs	Table – Structure containing all table parameters.
Description	Initializes all table parameters such as dimensions, number of foosmen, tracking method, etc. into the Table structure. This simplifies parameter passing since all parameters relating to the table are contained within the one structure. Also, the dimensions of the table can easily be modified by changing the assigned value in this function. The function also calculates several constant values such as the x-coordinate of each row and the maximum position of each foosmen. These values do not change, and are therefore only calculated once to reduce computation times.
How Invoked	Called by main function before the main loop.
Max Time	The function is not called in the main loop, so the timing information for this function is not critical to the performance of the system.

```
function [Table] = Table_Parameters()

% Note:  All dimensions are in centimeters unless otherwise noted.

Table.x_max  = 116.8;     % Length of table
Table.y_max  = 68;        % Width of table
Table.y_goal = 18.5;      % Width of the goal
Table.f_w    = 2.5;       % Width of a foosman (in y-direction)
Table.b_w    = 3.5;       % Width of the foosball (in y-direction)
Table.y_wall = 1.75;      % Minimum distance of foosmen from the wall

% Foosmen rods (distance between the center of two consecutive foosmen)
Table.y_R = [18.3,...     % Goalie
             24.5,...     % Defense
             12,  ...     % Midfield
             18.5];       % Offense

% Motor limits (maximum position)
Table.m_lim(1) = Table.y_max-2*Table.y_R(1)-2*Table.y_wall;
Table.m_lim(2) = Table.y_max-1*Table.y_R(2)-2*Table.y_wall;
Table.m_lim(3) = Table.y_max-4*Table.y_R(3)-2*Table.y_wall;
Table.m_lim(4) = Table.y_max-2*Table.y_R(4)-2*Table.y_wall;

% Maximum position of each individual foosman
% Row 1
Table.R1_F1_low_lim =                Table.y_wall + 0*Table.y_R(1);
Table.R1_F2_low_lim =                Table.y_wall + 1*Table.y_R(1);
Table.R1_F3_low_lim =                Table.y_wall + 2*Table.y_R(1);
Table.R1_F1_up_lim  = Table.y_max - (Table.y_wall + 2*Table.y_R(1));
Table.R1_F2_up_lim  = Table.y_max - (Table.y_wall + 1*Table.y_R(1));
Table.R1_F3_up_lim  = Table.y_max - (Table.y_wall + 0*Table.y_R(1));

% Row 2
Table.R2_F1_low_lim =                Table.y_wall + 0*Table.y_R(2);
Table.R2_F2_low_lim =                Table.y_wall + 1*Table.y_R(2);
Table.R2_F1_up_lim  = Table.y_max - (Table.y_wall + 1*Table.y_R(2));
Table.R2_F2_up_lim  = Table.y_max - (Table.y_wall + 0*Table.y_R(2));

% Row 3
Table.R3_F1_low_lim =                (Table.y_wall + 0*Table.y_R(3);
Table.R3_F2_low_lim =                Table.y_wall + 1*Table.y_R(3);
Table.R3_F3_low_lim =                Table.y_wall + 2*Table.y_R(3);
Table.R3_F4_low_lim =                Table.y_wall + 3*Table.y_R(3);
Table.R3_F5_low_lim =                Table.y_wall + 4*Table.y_R(3);
Table.R3_F1_up_lim  = Table.y_max - (Table.y_wall + 4*Table.y_R(3));
Table.R3_F2_up_lim  = Table.y_max - (Table.y_wall + 3*Table.y_R(3));
Table.R3_F3_up_lim  = Table.y_max - (Table.y_wall + 2*Table.y_R(3));
Table.R3_F4_up_lim  = Table.y_max - (Table.y_wall + 1*Table.y_R(3));
Table.R3_F5_up_lim  = Table.y_max - (Table.y_wall + 0*Table.y_R(3));

% Row 4
Table.R4_F1_low_lim =                Table.y_wall + 0*Table.y_R(4);
Table.R4_F2_low_lim =                Table.y_wall + 1*Table.y_R(4);
Table.R4_F3_low_lim =                Table.y_wall + 2*Table.y_R(4);
Table.R4_F1_up_lim  = Table.y_max - (Table.y_wall + 2*Table.y_R(4));
Table.R4_F2_up_lim  = Table.y_max - (Table.y_wall + 1*Table.y_R(4));
Table.R4_F3_up_lim  = Table.y_max - (Table.y_wall + 0*Table.y_R(4));

% Goal posts - [Bottom, Top]
Table.y_g = [.5*(Table.y_max-Table.y_goal),...
             .5*(Table.y_max-Table.y_goal)+Table.y_goal];

% x-coordinate of each row
Table.x_R = [ 1*Table.x_max/16,...  % Row 1
              3*Table.x_max/16,...  % Row 2
              5*Table.x_max/16,...  % Row 3
              7*Table.x_max/16,...  % Row 4
              9*Table.x_max/16,...  % Row 5
             11*Table.x_max/16,...  % Row 6
             13*Table.x_max/16,...  % Row 7
             15*Table.x_max/16];    % Row 8

% State variables for takeover
Table.Row1 = struct('F1',1,'F2',2,'F3',3);                  % Goalie
Table.Row2 = struct('F1',1,'F2',2);                         % Defense
Table.Row3 = struct('F1',1,'F2',2,'F3',3,'F4',4,'F5',5);  % Midfield
Table.Row4 = struct('F1',1,'F2',2,'F3',3);                  % Offense
Table.R1   = Table.Row1.F1;
Table.R2   = Table.Row2.F1;
Table.R3   = Table.Row3.F1;
Table.R4   = Table.Row4.F1;

% Tracking method
Table.Tracking_Method = struct('Simple',1,'Defense',2,'Trajectory',3);
Table.Tracking        = Table.Tracking_Method.Defense;

end
```

Table 9: Function - Goal

Module	Goal (ISR)
Inputs	None
Outputs	None
Description	The score is updated along with the LEDs used to display the current score. If the scoring team has 10 points, then the game is reset. Time permitting, various levels of difficulty will be implemented in the ISR based on the current score. For example, the acceleration of the motors for linear motion may be reduced if the foosbot is a certain number of points ahead of the human opponent.
How Invoked	Interrupt service routine. This function is invoked when the sensors inside the goals are triggered, alerting that a goal has been scored. This allows for continuous monitoring of the goals so that a goal is never missed.
Max Time	The function is only invoked when a goal is scored, so game play has stopped. Therefore, the timing information for this function is not critical to the performance of the system.

Table 10: Function - Foosball_Position

Module	Foosball_Position
Inputs	(x1,y1) – Previous position of the foosball.
Outputs	(x2,y2) – Current position of the foosball.
Description	The IR LEDs are pulsed and the corresponding phototransistors polled to determine the current position of the foosball. The inputs to the microprocessor are active-low digital signals, meaning that 0V corresponds to the ball being sensed at the location. An adaptive sensing algorithm is used where the previous position of the ball is used as the starting point for the search. The points immediately surrounding the starting position are sensed until the ball is detected. This technique reduces sensing time since the entire table is not scanned each iteration. Also, the chance of having an interference affect the sensing process is reduced. If interference causes some of the sensors to be temporarily blocked while the ball is not near the point of interference, the correct position of the foosball will continue to be tracked accurately since only points near the foosball are used in detection.
How Invoked	Called by main function inside the main loop.
Max Time	13 μs (see Section 3.k for timing computation details)

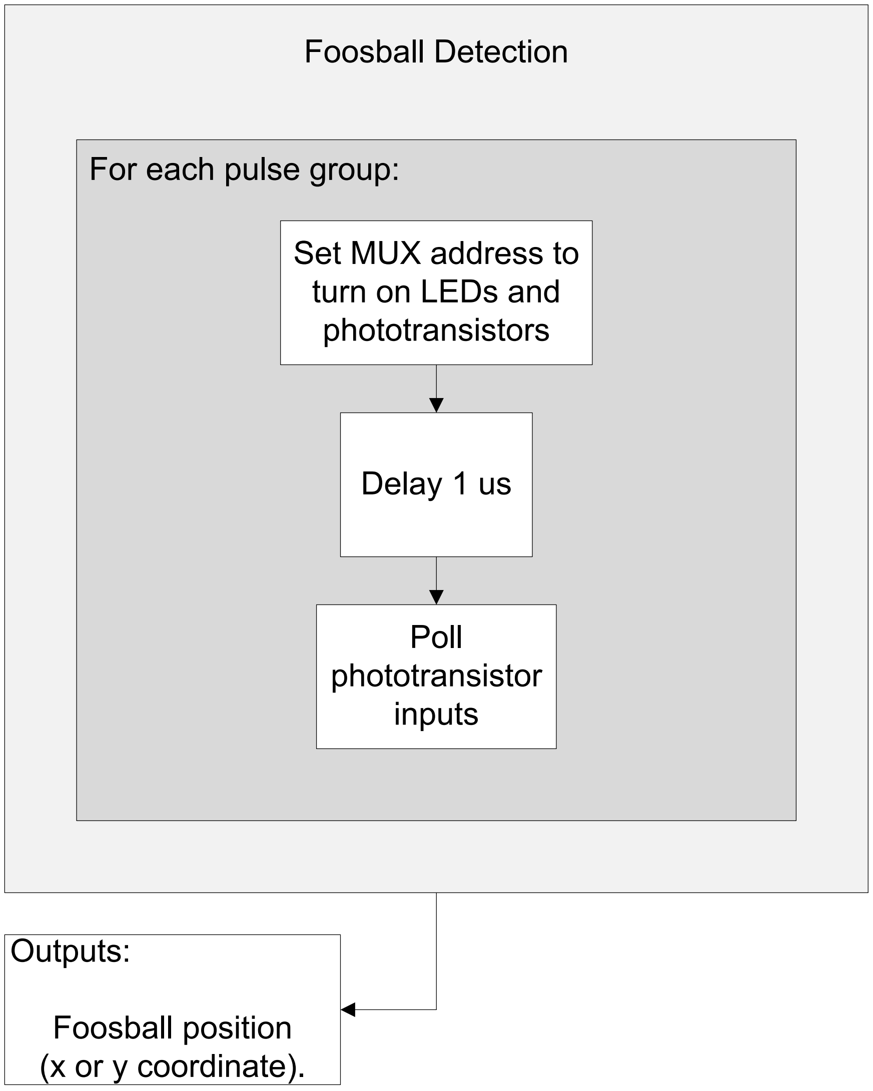

Foosball Detection
For each pulse group:
Set MUX address to turn on LEDs and phototransistors
Delay 1 us
Poll phototransistor inputs
Outputs:
Foosball position (x or y coordinate).

Figure 25: Foosball detection block diagram

Table 11: Function - Tracking_Method

Module	Tracking_Method
Inputs	Table – Structure of table parameters. x1 – Previous x-coordinate of the ball. x2 – Current x-coordinate of the ball.
Outputs	Table – Structure of table parameters (Table.Tracking is updated).
Description	Determines which tracking method is implemented depending on the current and previous x-coordinates of the foosball. If the ball is moving faster than 20% of the maximum foosball speed toward our goal, then trajectory tracking is used. Otherwise, if the ball is in front of our defense, then defensive tracking is used. Otherwise, simple tracking is used.
How Invoked	Called by main function inside the main loop.
Max Time	8 instructions = 0.004 μs

```
function [Table] = Tracking_Method(Table,x1,x2)

% If the ball is moving faster than 20% of maximum foosball speed
% toward our goal, then use trajectory.
if (x1-x2 > .2*Table.b_dx)
    Table.Tracking = Table.Tracking_Method.Trajectory;

% Else if the ball is to the right of the defensive line, then use
% defensive tracking.
elseif (x2 > Table.x_R(2))
    Table.Tracking = Table.Tracking_Method.Defense;

% Else use simple tracking.
else
    Table.Tracking = Table.Tracking_Method.Simple;
end

end
```

Table 12: Function - Foosmen_Position

Module	Foosmen_Position
Inputs	Table – Structure of table parameters. (x1,y1) – Previous position of foosball. (x2,y2) – Current position of foosball.

Outputs	y – Vector of length 4 containing the desired position of the foosmen.
	y(1) – Row 1 (goalie)
	y(2) – Row 2 (defense)
	y(3) – Row 3 (midfield)
	y(4) – Row 4 (offense)

Description	Calculates the desired position of each row of foosmen based on the tracking method used. See Section 3.b for a complete description of the different game strategies used, and how the foosmen are positioned for each of the strategies.
How Invoked	Called by main function inside the main loop.
Max Time	1000 instructions = 0.5 μs

```
function [y] = Foosmen_Position(Table,x1,y1,x2,y2)

% Tracking Methods:
%   - Simple
%   - Defensive
%   - Trajectory
switch Table.Tracking

    % Simple Tracking
    %   - Place foosmen at the current y-coordinate of the ball.
    case Table.Tracking_Method.Simple
        y(1:4) = y2;

    % Defensive Tracking
    %   - Place foosmen in optimal defensive positions depending on the
    %     current position of the foosball.
    case Table.Tracking_Method.Defense

        % Bottom region - Bottom half of the table.
        if (y2 < .5*Table.y_max)
            y_g1 = Table.y_g(1);  % Near post (y-coordinate)
            y_g2 = Table.y_g(2);  % Far  post (y-coordinate)
            s    = 1;             % Sign used for adding/subtracting
                                  % the width of the foosman, width of
                                  % the ball, etc.
        % Top region - Top half of the table.
        else
            y_g1 = Table.y_g(2);  % Near post (y-coordinate)
            y_g2 = Table.y_g(1);  % Far  post (y-coordinate)
            s    = -1;            % Sign
        end

        % Offense  - Half way between lines from ball to goal posts.
        % Midfield - Half way between lines from ball to goal posts.
        % Defense  - 1/3 of the way between lines from ball to goal
        %            posts, closest to near post.
        % Goalie   - 1/3 of the way between lines from ball to goal
        %            posts, closes to far post.
        y(4) =  ((Interpolate(Table.x_R(6),x2,y2,0,y_g1+s*.5*Table.b_w)
                  + s*.5*Table.f_w)...
               + (Interpolate(Table.x_R(6),x2,y2,0,y_g2-s*.5*Table.b_w)
                  - s*.5*Table.f_w))/2;
        y(3) =  ((Interpolate(Table.x_R(4),x2,y2,0,y_g1+s*.5*Table.b_w)
                  + s*.5*Table.f_w)...
               + (Interpolate(Table.x_R(4),x2,y2,0,y_g2-s*.5*Table.b_w)
                  - s*.5*Table.f_w))/2;

        d1 = Interpolate(Table.x_R(2),x2,y2,0,Table.y_g(i1)) +
                  s*.5*(Table.f_w+Table.b_w);
        d2 = Interpolate(Table.x_R(2),x2,y2,0,Table.y_g(i2)) -
                  s*.5*(Table.f_w+Table.b_w);
        y(2) = d1 + s*(abs(d2-d1)-Table.b_w-Table.f_w)/6;

        d1 = Interpolate(Table.x_R(1),x2,y2,0,Table.y_g(i1)) +
                  s*.5*(Table.f_w+Table.b_w);
        d2 = Interpolate(Table.x_R(1),x2,y2,0,Table.y_g(i2)) -
                  s*.5*(Table.f_w+Table.b_w);
        y(1) = d2 - s*(abs(d2-d1)-Table.b_w-Table.f_w)/3;

        % Use simple tracking for any row that the foosball is behind.
        if (x2 < Table.x_R(1))
            y(1:4) = y2;
        elseif (x2 < Table.x_R(2))
            y(2:4) = y2;
        elseif (x2 < Table.x_R(4))
            y(3:4) = y2;
        elseif (x2 < Table.x_R(6))
            y(4)   = y2;
        end

    % Trajectory Tracking
    %   - Place foosmen in the path of the foosball using linear
    %     interpolation. If the ball will bounce off the wall before
    %     reaching a row of foosmen, then the trajectory calculations
    %     take the deflection into account.
    case Table.Tracking_Method.Trajectory
        % Slope
        m = (y2-y1)/(x2-x1);

        % x-coordinate of a bottom deflection
        x_bottom = x1+(.5*Table.b_w-y1)/m;

        % x-coordinate of a top deflection
        x_top    = x1+(Table.y_max-.5*Table.b_w-y1)/m;

        % Calculate straight interpolation for each row
        y(4) = Interpolate(Table.x_R(6),x2,y2,x1,y1);
        y(3) = Interpolate(Table.x_R(4),x2,y2,x1,y1);
        y(2) = Interpolate(Table.x_R(2),x2,y2,x1,y1);
        y(1) = Interpolate(Table.x_R(1),x2,y2,x1,y1);

        % If the ball is moving toward the top wall and a deflection
```

```
        % occurs, then re-calculate the trajectory for any row after
        % the deflection.
        if (y2 > y1 && x_top > 0)
            if (Table.x_R(6) < x_top)
                y(4)=(Table.y_max-.5*Table.b_w)-m*(Table.x_R(6)-x_top);
                y(3)=(Table.y_max-.5*Table.b_w)-m*(Table.x_R(4)-x_top);
                y(2)=(Table.y_max-.5*Table.b_w)-m*(Table.x_R(2)-x_top);
                y(1)=(Table.y_max-.5*Table.b_w)-m*(Table.x_R(1)-x_top);
            elseif (Table.x_R(4) < x_top)
                y(3)=(Table.y_max-.5*Table.b_w)-m*(Table.x_R(4)-x_top);
                y(2)=(Table.y_max-.5*Table.b_w)-m*(Table.x_R(2)-x_top);
                y(1)=(Table.y_max-.5*Table.b_w)-m*(Table.x_R(1)-x_top);
            elseif (Table.x_R(2) < x_top)
                y(2)=(Table.y_max-.5*Table.b_w)-m*(Table.x_R(2)-x_top);
                y(1 =(Table.y_max-.5*Table.b_w)-m*(Table.x_R(1)-x_top);
            elseif (Table.x_R(1) < x_top)
                y(1)=(Table.y_max-.5*Table.b_w)-m*(Table.x_R(1)-x_top);
            end

        % Else if the ball is moving toward the bottom wall and a
        % deflection occurs, then re-calculate the trajectory for any
        % row after the deflection.
        elseif (y2 < y1 && x_bottom > 0)
            if (Table.x_R(6) < x_bottom)
                y(4) = .5*Table.b_w-m*(Table.x_R(6)-x_bottom);
                y(3) = .5*Table.b_w-m*(Table.x_R(4)-x_bottom);
                y(2) = .5*Table.b_w-m*(Table.x_R(2)-x_bottom);
                y(1) = .5*Table.b_w-m*(Table.x_R(1)-x_bottom);
            elseif (Table.x_R(4) < x_bottom)
                y(3) = .5*Table.b_w-m*(Table.x_R(4)-x_bottom);
                y(2) = .5*Table.b_w-m*(Table.x_R(2)-x_bottom);
                y(1) = .5*Table.b_w-m*(Table.x_R(1)-x_bottom);
            elseif (Table.x_R(2) < x_bottom)
                y(2) = .5*Table.b_w-m*(Table.x_R(2)-x_bottom);
                y(1) = .5*Table.b_w-m*(Table.x_R(1)-x_bottom);
            elseif (Table.x_R(1) < x_bottom)
                y(1) = .5*Table.b_w-m*(Table.x_R(1)-x_bottom);
            end
        end

        % Use simple tracking for any row that the foosball is behind.
        if (x2 < Table.x_R(1))
            y(1:4) = y2;
        elseif (x2 < Table.x_R(2))
            y(2:4) = y2;
        elseif (x2 < Table.x_R(4))
            y(3:4) = y2;
        elseif (x2 < Table.x_R(6))
            y(4)   = y2;
        end
end

end
```

Table 13: Function - Takeover

Module	Takeover
Inputs	Table – Structure of table parameters. x_b – Current x-coordinate of the ball. y – Vector of length 4 containing the desired y-coordinates of each row. motor – Vector of length 4 containing the current motor positions.
Outputs	Table – Structure of table parameters.
Description	Calculates the takeover for each row using finite state machines and the current motor positions and desired foosmen positions. The values of - Table.R1 - Table.R2 - Table.R3 - Table.R4 are updated if necessary to the desired foosman for tracking in each row. See Section 3.j for a complete description of the takeover process including flow diagrams for the finite state machine implementations for each row.
How Invoked	Called by main function inside the main loop.
Max Time	1000 instructions = 0.5 μs

```
function [Table] = Takeover(Table,x_b,y,motor)

% Calculate the current foosmen positions from the motor positions.
% Note: These are different than the desired positions (y).

% Row 2
y2(1) = motor(2)-.5*Table.f_w+Table.y_wall;                  % Foosman 1
y2(2) = motor(2)-.5*Table.f_w+Table.y_wall+1*Table.y_R(2);  % Foosman 2

% Row 3
y3(1) = motor(3)-.5*Table.f_w+Table.y_wall;                  % Foosman 1
y3(2) = motor(3)-.5*Table.f_w+Table.y_wall+1*Table.y_R(3);  % Foosman 2
y3(3) = motor(3)-.5*Table.f_w+Table.y_wall+2*Table.y_R(3);  % Foosman 3
y3(4) = motor(3)-.5*Table.f_w+Table.y_wall+3*Table.y_R(3);  % Foosman 4
y3(5) = motor(3)-.5*Table.f_w+Table.y_wall+4*Table.y_R(3);  % Foosman 5

% Row 4
y4(1) = motor(4)-.5*Table.f_w+Table.y_wall;                  % Foosman 1
y4(2) = motor(4)-.5*Table.f_w+Table.y_wall+1*Table.y_R(4);  % Foosman 2
y4(3) = motor(4)-.5*Table.f_w+Table.y_wall+2*Table.y_R(4);  % Foosman 3

% Row 1 - See block diagram in report for details.
switch Table.R1
    case Table.Row1.F1
        if (y(1) > Table.R1_F1_up_lim || x_b > Table.x_R(2))
            Table.R1 = Table.Row1.F2;
        end
    case Table.Row1.F2
        if (y(1) > Table.R1_F2_up_lim && x_b < Table.x_R(2))
            Table.R1 = Table.Row1.F3;
        elseif (y(1) < Table.R1_F2_low_lim && x_b < Table.x_R(2))
```

```
            Table.R1 = Table.Row1.F1;
        end
    case Table.Row1.F3
        if (y(1) < Table.R1_F3_low_lim || x_b > Table.x_R(2))
            Table.R1 = Table.Row1.F2;
        end
end

% Row 2 - See block diagram in report for details.
switch Table.R2
    case Table.Row2.F1
        if (y(2) > Table.R2_F1_up_lim || (abs(y(2)-y2(2)) < abs(y(2)-
                              y2(1)) && y(2) > Table.R2_F2_low_lim))
            Table.R2 = Table.Row2.F2;
        end
    case Table.Row2.F2
        if (y(2) < Table.R2_F2_low_lim || (abs(y(2)-y2(1)) < abs(y(2)-
                              y2(2)) && y(2) < Table.R2_F1_up_lim))
            Table.R2 = Table.Row2.F1;
        end
end

% Row 3 - See block diagram in report for details.
switch Table.R3
    case Table.Row3.F1
        if (y(3) > Table.R3_F1_up_lim || (abs(y(3)-y3(2)) < abs(y(3)-
                              y3(1)) && y(3) > Table.R3_F2_low_lim))
            Table.R3 = Table.Row3.F2;
        end
    case Table.Row3.F2
        if (y(3) > Table.R3_F2_up_lim || (abs(y(3)-y3(3)) < abs(y(3)-
                              y3(2)) && y(3) > Table.R3_F3_low_lim))
            Table.R3 = Table.Row3.F3;
        elseif (y(3) < Table.R3_F2_low_lim || (abs(y(3)-y3(1)) <
                        abs(y(3)-y3(2)) && y(3) < Table.R3_F1_up_lim))
            Table.R3 = Table.Row3.F1;
        end
    case Table.Row3.F3
        if (y(3) > Table.R3_F3_up_lim || (abs(y(3)-y3(4)) < abs(y(3)-
                              y3(3)) && y(3) > Table.R3_F4_low_lim))
            Table.R3 = Table.Row3.F4;
        elseif (y(3) < Table.R3_F3_low_lim || (abs(y(3)-y3(2)) <
                        abs(y(3)-y3(3)) && y(3) < Table.R3_F2_up_lim))
            Table.R3 = Table.Row3.F2;
        end
    case Table.Row3.F4
        if (y(3) > Table.R3_F4_up_lim || (abs(y(3)-y3(5)) < abs(y(3)-
                              y3(4)) && y(3) > Table.R3_F5_low_lim))
            Table.R3 = Table.Row3.F5;
        elseif (y(3) < Table.R3_F4_low_lim || (abs(y(3)-y3(3)) <
                        abs(y(3)-y3(4)) && y(3) < Table.R3_F3_up_lim))
            Table.R3 = Table.Row3.F3;
        end
    case Table.Row3.F5
        if (y(3) < Table.R3_F5_low_lim || (abs(y(3)-y3(4)) < abs(y(3)-
                              y3(5)) && y(3) < Table.R3_F4_up_lim))
            Table.R3 = Table.Row3.F4;
        end
end

% Row 4 - See block diagram in report for details.
switch Table.R4
    case Table.Row4.F1
        if (y(4) > Table.R4_F1_up_lim || (abs(y(4)-y4(2)) < abs(y(4)-
                              y4(1)) && y(4) > Table.R4_F2_low_lim))
            Table.R4 = Table.Row4.F2;
        end
    case Table.Row4.F2
        if (y(4) > Table.R4_F2_up_lim || (abs(y(4)-y4(3)) < abs(y(4)-
                              y4(2)) && y(4) > Table.R4_F3_low_lim))
            Table.R4 = Table.Row4.F3;
        elseif (y(4) < Table.R4_F2_low_lim || (abs(y(4)-y4(1)) <
                        abs(y(4)-y4(2)) && y(4) < Table.R4_F1_up_lim))
            Table.R4 = Table.Row4.F1;
        end
    case Table.Row4.F3
        if (y(4) < Table.R4_F3_low_lim || (abs(y(4)-y4(2)) < abs(y(4)-
                              y4(3)) && y(4) < Table.R4_F2_up_lim))
            Table.R4 = Table.Row4.F2;
        end
end

end
```

Table 14: Function - Motor_Position

Module	Motor_Position
Inputs	Table – Structure of table parameters. y – Vector of length 4 containing the desired y-coordinates of each row.
Outputs	motor – Vector of length 4 containing the motor positions for each row.
Description	Calculates the motor positions required to put the foosmen who are currently tracking the ball in the desired location for each row. The motor position is zero when the rod is pulled out all the way so that the foosmen are against the wall.
How Invoked	Called by main function inside the main loop.
Max Time	32 instructions = 0.016 μs

```
function [motor] = Motor_Position(Table,y,motor)

% Calculate motor positions based on desired foosmen location and which
% foosman is currently tracking the ball.
motor(1) = y(1) - (Table.R1-1)*Table.y_R(1) - Table.y_wall;
motor(2) = y(2) - (Table.R2-1)*Table.y_R(2) - Table.y_wall;
motor(3) = y(3) - (Table.R3-1)*Table.y_R(3) - Table.y_wall;
motor(4) = y(4) - (Table.R4-1)*Table.y_R(4) - Table.y_wall;

% Make sure the motor position is a valid length
for i = 1:4

    % Make sure motor position is non-negative.
    motor(i) = max([motor(i), 0]);

    % Make sure motor position is less than or equal to the maximum
    % position for that motor.
    motor(i) = min([motor(i), Table.m_lim(i)]);
end
end
```

Table 15: Function - Move_Motors

Module	Move_Motors
Inputs	motor – Vector of length 4 containing the motor positions for each row.
Outputs	None
Description	The desired motor positions are sent to the motors using RS-485 communication. The motors are connected in a daisy-chain so the commands to each motor go through the first (master) motor processor. Each motor has 24 programmable positions where it can be commanded to move. The motor is commanded to move to the available position closest to the desired position.
How Invoked	Called by main function inside the main loop.
Max Time	The timing required to move the motors depends on the position where the motor is being commanded to move. The minimum time required for motor movement is 3 ms. See Section 3.k for timing details.

4. Parts List

The complete parts list for the project is given in Table 16. The Refdes column gives the reference description for each component used in the technical drawings.

Table 16: Parts List

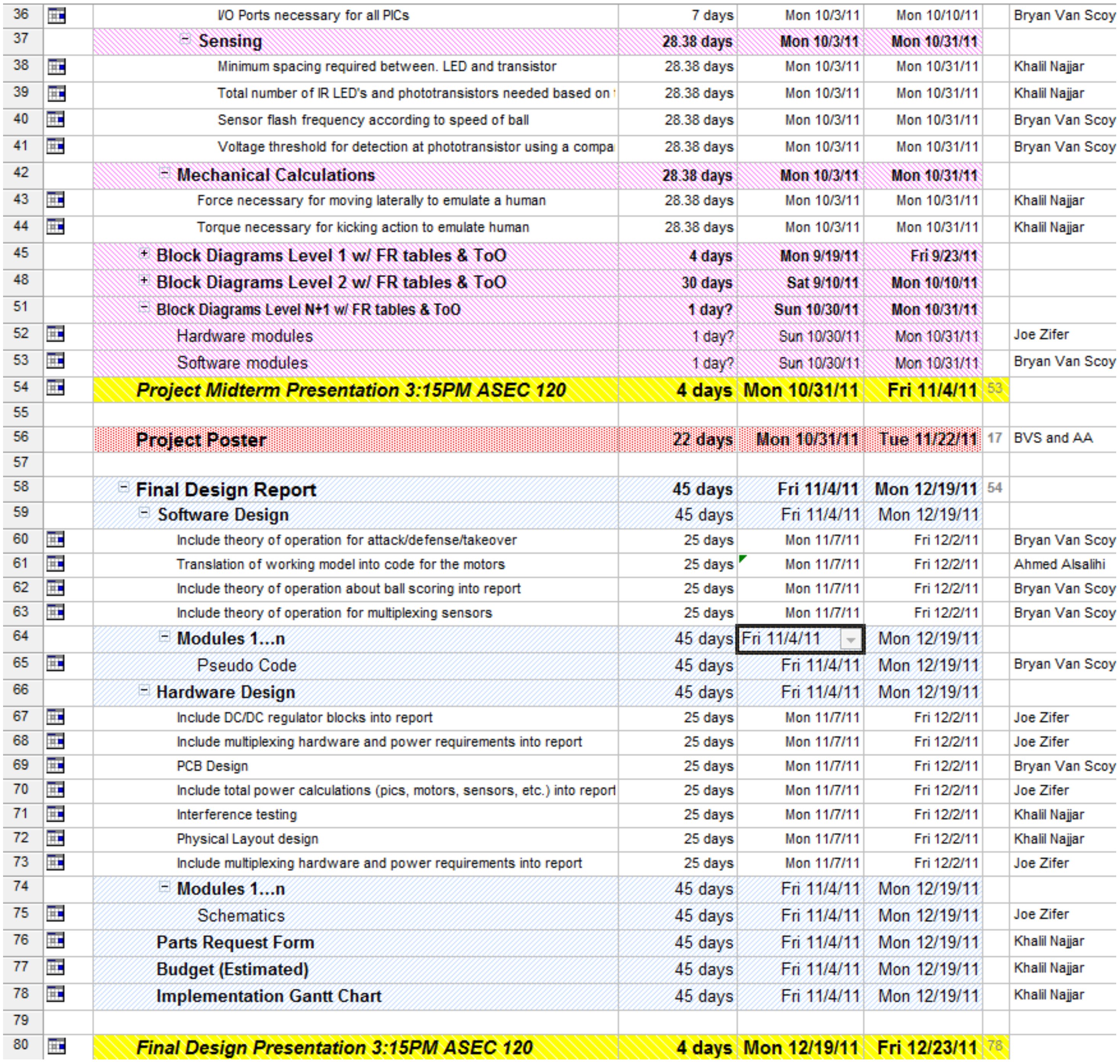
Qty.	Refdes	Part Num.	Description
1	Harvard	N/A	Foosball Table
1	BB	BB-BONE-000	BeagleBone with AM3359 ARM Processor
12	PB-1 - PB-12	N/A	PCB Sensor Board (w/ solder mask & silk scrn)
250	Qx,x	QSC112	IR Phototransistor
250	LEDx,x	QED123	IR LEDs
50	C1 - C4	LM2901N	4-Channel Comparator IC
25	MX, DMX	74HCT4067N	16-Channel Multiplexer IC
24	J3, J4	43045-0800	8-Pin PCB Header (Right angle)
24	CBL1	43025-0800	8-Pin Receptacle
24	J1, J2	43045-1827	18-Pin PCB Header (Vertical)
24		43025-1800	18-Pin Receptacle
200		43030-0007	Crimp Terminal 20-24 AWG
400		43030-0010	Crimp Terminal 26-30 AWG
4	CA-MIC4-W4-NC-2	CA-MIC4-W4-NC-2	2' cable
4	RMx	HT17-274D	NEMA 17 Stepper Motor, High Torque, Double Shaft
4	ENx	E8P-200-197-S-H-D-B	Encoders
200		CR-174L	.5mm LED Holder
200		CR-104	.3mm Phototransistor Holder
4	HBx	SN754410NE	Quad Half H-Bridge Driver IC
4	D1	1N5819	Schottky Barrier Rectifier
1	PS3	KTPS05-05015U	5V 1A Power Supply (for BeagleBone)
200	R1,1 - R12,16	31202	Resistor 470k Ohm
15	RLED	30162	Resistor 150 Ohm
15	Rvd2	29911	Resistor 10k Ohm
15	Rvd1	30314	Resistor 2.2k Ohm
4	R2	RN55D1352FB14	Resistor 13.5k Ohm
4	Ci	29962	Capacitor 100 uF
4	Co	UPS1J331MHD1TD	Capacitor 330 uF
4	L1	12LRS334C	Inductor 330 uH
4		IHWO18	Inline Fuse Holder 18 AWG
5	FU-3	ATO2	Blade Fuse 2A
5	FU-1, FU-2	ATO5	Blade Fuse 5A
5	FU-4	ATO10	Blade Fuse 10A
4	VR1 - VR4	LM2575T-5.0/NOPB	Voltage Regulator
4	ML-x	CM1-X-23L20	Motors for linear motion
1	PS2	PSU-240	Power Supply - 240W
1	PS1	PSU-300	Power Supply - 300W

The cost of producing the complete foosbot system can be quite expensive due to the required motors and number of sensors. Many of the parts for the project have been donated, greatly reducing the final cost. All costs are shown in Table 17 along with the total cost of the foosbot. Note that the cost for the donated items is included in the total.

Table 17: Revised Material Cost

Qty.	Part Num.	Description	Cost	Cost
1	N/A	Foosball Table	$60.00	$60.00
1	BB-BONE-000	BeagleBone with AM3359 ARM Processor	89.00	89.00
12	N/A	PCB Sensor Board (w/ silkscreen and solder mask)	23.38	280.57
250	QSC112	IR Phototransistor	0.16	40.25
250	QED123	IR LEDs	0.27	67.25
50	LM2901N	4-Channel Comparator IC	0.16	7.90
25	74HCT4067N	16-Channel Multiplexer IC	3.87	96.75
24	43045-0800	8-Pin PCB Header (Right angle)	1.59	38.16
24	43025-0800	8-Pin Receptacle	0.53	12.72
24	43045-1827	18-Pin PCB Header (Vertical)	2.75	66.00
24	43025-1800	18-Pin Receptacle	1.33	31.92
200	43030-0007	Crimp Terminal 20-24 AWG	0.07	14.00
400	43030-0010	Crimp Terminal 26-30 AWG	0.11	44.00
4	CA-MIC4-W4-NC-2	2' cable	7.30	29.20
4	HT17-274D	NEMA 17 Stepper Motor, High Torque, Double Shft	36.00	144.00
4	E8P-200-197-S-H-D-B	Encoders	37.34	149.36
200	CR-174L	.5mm LED Holder	0.13	26.60
200	CR-104	.3mm Phototransistor Holder	0.14	28.00
4	SN754410NE	Quad Half H-Bridge Driver IC	2.30	9.20
4	1N5819	Schottky Barrier Rectifier	0.20	0.80
1	KTPS05-05015U	5V 1A Power Supply (for BeagleBone)	7.56	7.56
200	31202	Resistor 470k Ohm		
15	30162	Resistor 150 Ohm		
15	29911	Resistor 10k Ohm		
15	30314	Resistor 2.2k Ohm		
4	RN55D1352FB14	Resistor 13.5k Ohm	0.10	0.40
4	29962	Capacitor 100 uF		
4	UPS1J331MHD1TD	Capacitor 330 uF	0.28	1.12
4	12LRS334C	Inductor 330 uH	0.71	2.84
4	IHWO18	Inline Fuse Holder 18 AWG	1.00	4.00
5	ATO2	Blade Fuse 2A	0.20	1.00
5	ATO5	Blade Fuse 5A	0.20	1.00
5	ATO10	Blade Fuse 10A	0.20	1.00
4	LM2575T-5.0/NOPB	Voltage Regulator (DONATED)	3.01	12.04
4	CM1-X-23L20	Motors for linear motion (DONATED)	400.00	1,600.00
1	PSU-240	Power Supply - 240W (DONATED)	40.00	40.00
1	PSU-300	Power Supply - 300W (DONATED)	50.00	50.00
4	MSA-PSC	Motor slides (DONATED)	288.00	1,152.00
			Total	$4,108.64

5. Project Schedules - KMN



36	▦	I/O Ports necessary for all PICs	7 days	Mon 10/3/11	Mon 10/10/11		Bryan Van Scoy
37		⊟ **Sensing**	**28.38 days**	**Mon 10/3/11**	**Mon 10/31/11**		
38	▦	Minimum spacing required between. LED and transistor	28.38 days	Mon 10/3/11	Mon 10/31/11		Khalil Najjar
39	▦	Total number of IR LED's and phototransistors needed based on	28.38 days	Mon 10/3/11	Mon 10/31/11		Khalil Najjar
40	▦	Sensor flash frequency according to speed of ball	28.38 days	Mon 10/3/11	Mon 10/31/11		Bryan Van Scoy
41	▦	Voltage threshold for detection at phototransistor using a compa	28.38 days	Mon 10/3/11	Mon 10/31/11		Bryan Van Scoy
42		⊟ **Mechanical Calculations**	**28.38 days**	**Mon 10/3/11**	**Mon 10/31/11**		
43	▦	Force necessary for moving laterally to emulate a human	28.38 days	Mon 10/3/11	Mon 10/31/11		Khalil Najjar
44	▦	Torque necessary for kicking action to emulate human	28.38 days	Mon 10/3/11	Mon 10/31/11		Khalil Najjar
45		⊞ **Block Diagrams Level 1 w/ FR tables & ToO**	**4 days**	**Mon 9/19/11**	**Fri 9/23/11**		
48		⊞ **Block Diagrams Level 2 w/ FR tables & ToO**	**30 days**	**Sat 9/10/11**	**Mon 10/10/11**		
51		⊟ **Block Diagrams Level N+1 w/ FR tables & ToO**	**1 day?**	**Sun 10/30/11**	**Mon 10/31/11**		
52	▦	Hardware modules	1 day?	Sun 10/30/11	Mon 10/31/11		Joe Zifer
53	▦	Software modules	1 day?	Sun 10/30/11	Mon 10/31/11		Bryan Van Scoy
54	▦	***Project Midterm Presentation 3:15PM ASEC 120***	**4 days**	**Mon 10/31/11**	**Fri 11/4/11**	53	
55							
56		**Project Poster**	**22 days**	**Mon 10/31/11**	**Tue 11/22/11**	17	BVS and AA
57							
58		⊟ **Final Design Report**	**45 days**	**Fri 11/4/11**	**Mon 12/19/11**	54	
59		⊟ **Software Design**	45 days	Fri 11/4/11	Mon 12/19/11		
60	▦	Include theory of operation for attack/defense/takeover	25 days	Mon 11/7/11	Fri 12/2/11		Bryan Van Scoy
61	▦	Translation of working model into code for the motors	25 days	Mon 11/7/11	Fri 12/2/11		Ahmed Alsalihi
62	▦	Include theory of operation about ball scoring into report	25 days	Mon 11/7/11	Fri 12/2/11		Bryan Van Scoy
63	▦	Include theory of operation for multiplexing sensors	25 days	Mon 11/7/11	Fri 12/2/11		Bryan Van Scoy
64		⊟ **Modules 1…n**	45 days	Fri 11/4/11	Mon 12/19/11		
65	▦	Pseudo Code	45 days	Fri 11/4/11	Mon 12/19/11		Bryan Van Scoy
66		⊟ **Hardware Design**	45 days	Fri 11/4/11	Mon 12/19/11		
67	▦	Include DC/DC regulator blocks into report	25 days	Mon 11/7/11	Fri 12/2/11		Joe Zifer
68	▦	Include multiplexing hardware and power requirements into report	25 days	Mon 11/7/11	Fri 12/2/11		Joe Zifer
69	▦	PCB Design	25 days	Mon 11/7/11	Fri 12/2/11		Bryan Van Scoy
70	▦	Include total power calculations (pics, motors, sensors, etc.) into report	25 days	Mon 11/7/11	Fri 12/2/11		Joe Zifer
71	▦	Interference testing	25 days	Mon 11/7/11	Fri 12/2/11		Khalil Najjar
72	▦	Physical Layout design	25 days	Mon 11/7/11	Fri 12/2/11		Khalil Najjar
73	▦	Include multiplexing hardware and power requirements into report	25 days	Mon 11/7/11	Fri 12/2/11		Joe Zifer
74		⊟ **Modules 1…n**	45 days	Fri 11/4/11	Mon 12/19/11		
75	▦	Schematics	45 days	Fri 11/4/11	Mon 12/19/11		Joe Zifer
76	▦	**Parts Request Form**	45 days	Fri 11/4/11	Mon 12/19/11		Khalil Najjar
77	▦	**Budget (Estimated)**	45 days	Fri 11/4/11	Mon 12/19/11		Khalil Najjar
78	▦	**Implementation Gantt Chart**	45 days	Fri 11/4/11	Mon 12/19/11		Khalil Najjar
79							
80	▦	***Final Design Presentation 3:15PM ASEC 120***	**4 days**	**Mon 12/19/11**	**Fri 12/23/11**	78	

6. Design Team Information

Ahmed Alsalihi, Electrical Engineering.

Khalil Najjar, Electrical Engineering.

Bryan Van Scoy, Electrical Engineering, Applied Mathematics.

Joe Zifer, Electrical Engineering.

7. Conclusions and Recommendations

Foosbot is going to be an exciting and challenging project. We look forward to taking on these challenges with the knowledge attained in studying at the University of Akron. Depending on how timing of the project goes and coordination with faculty, Foosbot may be entered at the RoboGames competition in California for “Best At Show”.

8. References

1. Coley, G. (2011, November 5). BeagleBone Rev A3 System Reference Manual.

2. Di Jasio, L. (2007). Programming 16-bit microcontrollers in C: Learning to fly the PIC24. Burlington, MA: Elsevier Inc.

3. International Table Soccer Federation. (2010). Retrieved March 30, 2010 from http://www.table-soccer.org/

4. Marks, MJ. (2010, May 11). Four Fun Facts about Foosball Tables. *Ezine Articles.* Retrieved March 30, 2011 from http://ezinearticles.com/?Four-Fun-Facts-About-Foosball-Tables

5. Neo. (2010, August 24). Top 10 Foosball Tables. *Crunchpost.* Retrieved March 30, 2010 from http://crunchpost.com/top-10-foosball-tables/

9. Appendix

Table 18: Datasheets

AM3359	http://www.ti.com/lit/ds/symlink/am3359.pdf
BeagleBone	http://beagleboard.org/static/BONESRM_latest.pdf
IR Phototransistor	http://www.fairchildsemi.com/ds/QS/QSC114.pdf
IR LED	http://www.fairchildsemi.com/ds/QE/QED123.pdf
Comparator	http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATASHEET/CD00000532.pdf
Multiplexer	http://www.nxp.com/documents/data_sheet/74HC_HCT4067.pdf
8-Pin PCB Header (Right Angle)	http://www.molex.com/webdocs/datasheets/pdf/en-us/0430450800_PCB_HEADERS.pdf
8-Pin Receptacle	http://www.molex.com/webdocs/datasheets/pdf/en-us/0430250800_CRIMP_HOUSINGS.pdf
18-Pin PCB Header (Vertical)	http://www.molex.com/webdocs/datasheets/pdf/en-us/0430451827_PCB_HEADERS.pdf
18-Pin Receptacle	http://www.molex.com/webdocs/datasheets/pdf/en-us/0430251800_CRIMP_HOUSINGS.pdf
Crimp Terminal 20-24 AWG	http://www.molex.com/webdocs/datasheets/pdf/en-us/0430300007_CRIMP_TERMINALS.pdf
Crimp Terminal 26-30 AWG	http://www.molex.com/webdocs/datasheets/pdf/en-us/0430300010_CRIMP_TERMINALS.pdf
Stepper Motor (rotational motion)	http://www.applied-motion.com/sites/default/files/HT17-274_RevC.pdf
Encoders	http://www.usdigital.com/assets/general/258_e8p_datasheet_0.pdf
.5mm LED Holder	http://www.bivar.com/images/cart/CR-174L.pdf
.3mm Phototransistor Holder	http://www.bivar.com/images/cart/CR-104.pdf
Regulator	http://www.national.com/ds/LM/LM2575.pdf
Fuses	
Quad Half H-Bridge Driver	http://www.ti.com/lit/ds/symlink/sn754410.pdf
Linear Rails	http://www.macrondynamics.com/catalog/pdf/MSA-PSC.pdf
Servo-Controlled Stepper Motors (linear motion)	http://www.myostat.ca/documents/CoolMuscleManual.pdf
Schottky Barrier Rectifier	http://www.fairchildsemi.com/ds/1N/1N5819.pdf
5V 1A Power Supply	http://media.digikey.com/pdf/Data%20Sheets/Volgen%20PDFs/KTPS05_Series.pdf