FOOSE: Football Operator and Optical Soccer Engine

Nathaniel Enos, Patrick Fenelon, Skyler Goodell, and Nicholas Phillips

Department of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — The objective of the FOOSE project is to automate one-half of a foosball table, in order to provide an entertaining human vs. robot variation on the classic game. Its major subsystems are image acquisition, table state interpretation, artificial intelligence, and move execution, which together form a complete foosball-playing system. This paper will detail the methodology behind and function of each subsystem, as well as addressing integration and testing of the complete system.**

***Index Terms* — Artificial intelligence, computer vision, image processing, optical sensors, parallel processing, real time systems.**

I. Introduction

Over the years, there have been several attempts to produce a semi-automated foosball table. Of these, some have been done in the context of senior design groups, to varying degrees of success. On the other hand, one of the group's inspirations is a high-performance professionally manufactured semi-automated foosball table, called the Star Kick board (seen in Fig. 1), which is sold to arcades in Europe for $27,000. The primary goal of the FOOSE project is to maximize performance while working within budget constraints. It should be here noted that the Project's budget constraints are being mitigated in part by a very generous scholarship from Soar Technologies, Inc.

This paper will be organized according to data flow; since the project has a linear data flow pattern, it is most reasonable to discuss the subsystems as they appear in order, starting with the current table state (ball position) and ending with an executed move, as will be discussed in Section II and ongoing.

II. Overview of Major Subsystems

Due to its nature, the FOOSE project is easily divided into a number of subsystems, each of which performs a major task on the data stream. In this section, each

Fig. 1 Star Kick board

subsystem will be discussed briefly, and each will be discussed in more detail in later sections.

A. Image Acquisition

To acquire an image of the table, the depth sensor of a Microsoft Kinect for Windows takes an image of the table. This image is passed to Table State Interpretation.

B. Table State Interpretation

The depth image is then processed (on an off-the-shelf computer) to extract the candidate ball coordinates. These coordinates are passed to the Physics Engine.

C. Physics Engine

The physics engine maintains a flexible internal model of what might be happening on the table. It takes in sets of potential coordinates and, using knowledge of past coordinates, outputs the coordinates most likely to be accurate to the Artificial Intelligence.

D. Artificial Intelligence

The Artificial Intelligence (AI) takes the ball's coordinates and uses its knowledge of previous board states and rod locations to calculate a move for all four computer-controlled rods. These moves are then outputted to the four Rod Control Boards.

E. Rod Control Board

The Rod Control Boards or RCBs are custom-printed PCBs which are each responsible for one computer-controlled rod, making four in total. Each takes in an absolute move position from the AI and powers the stepper motors to move to that location, and kick if necessary.

F. Mechanics

The mechanics subsystems consist of the stepper motors which actually move the rods, and the tracks and pulleys, cams, etc. which physically move and kick on command. On being powered by the RCB, this subsystem physically executes the move, the table state is changed, and the process begins again.

III. SPECIFICATIONS

Before discussing the major components of the project in more detail, it will be helpful to establish some figures to work towards when designing the subsystems. In testing, it was determined that a typical kick by a novice player travels at approximately 1 m/s. This value means different things for different subsystems, which can be broken down into two major categories:

A. Processing Subsystems

For the specifications of the processing subsystems, it is useful to discuss the combined total lag they impart to the system. Of major subsystems, those included under this heading are Image Acquisition, Table State Interpretation, the Physics Engine, and the AI. Although lag in this area is partially mitigated by the AI's ability to predict, the project will require that the lag be no more than 75ms. This lag would allow a ball at 1 m/s to move 75mm, or half the distance between any two rods of the foosball table, before reacting. This should be sufficient to ensure a good game.

B. Mechanical Subsystems

For the mechanical subsystems, the calculations become more involved. It is necessary to obtain motors with sufficient torque to move the rods linearly. For this, the following information is taken into account: A maximum hit travels at 1.08 m/s, the distance between rods is 0.15 m, the maximum distance any rod must move to intercept a ball is 0.20 m, the mass of a rod is 1.049 kg, and the radius of the pulley used in mechanical construction is 0.0129 m. From this information the following equations can be synthesized:

$$t = \frac{0.15\text{m}}{1.08\text{m/s}} = 0.138\text{s}$$

$$a = \frac{0.2\text{m}}{\text{t}^2} = 12.21\text{m/s}^2$$

$$F = (1.049) * 12.21 \ = 12.81N$$

$$T = 12.81 * .0129 = 0.165Nm$$

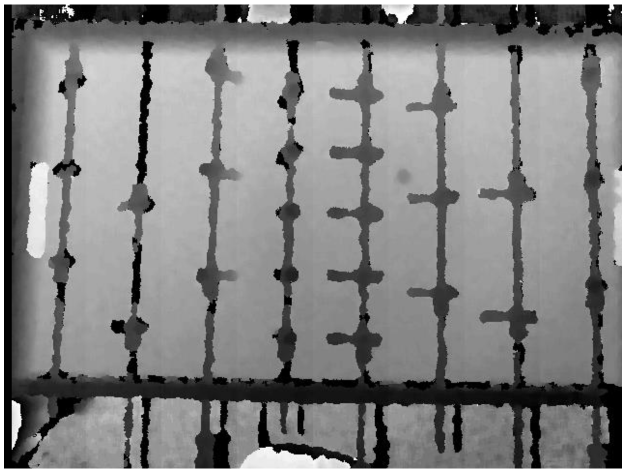


Fig. 2. Sample Output from Kinect

And so, in conclusion, the linear control motor must have 0.165Nm of torque at 1247rpm.

C. Defined Terms

For the purposes of the FOOSE project, the X direction is down the table (from goal to goal) and the Y direction is across the table; the direction along which the rods can move.

IV. IMAGE ACQUISITION

As aforementioned, the physical component of the image acquisition subsystem is the Kinect for Windows, using its depth sensor. This is capable of operating at 30 frames per second. Assuming a standard shot at 1 m/s, this means that the Kinect takes a picture of it every 33mm, which is a sufficient time resolution to allow for further image processing. The depth sensor operates at 640x480 pixels, which (as configured) gives it a resolution of approximately 508x282 pixels on the table itself, which translates to about one pixel per square millimeter. This offers more than enough resolution to accurately track the ball's position. A sample output can be seen above in Fig. 2. The Kinect is physically mounted above the table using a wooden box suspended from the ceiling by five strings. This design makes it capable of being manipulated in order to properly center and align it.

V. TABLE STATE INTERPRETATION

The Table State Interpretation (TSI) subsystem is tasked with taking raw depth sensor output from the Kinect hardware and extracting from it the coordinates most likely to be the ball. To that end, it uses a multi-stage approach, which processes the image in different ways using different computer vision (CV) algorithms. Before discussing the algorithms themselves, a word on what they run on:

A. Development Tools

The physical component on which the TSI, Physics Engine, and AI subsystems run is a regular computer composed of off-the-shelf parts. It runs Windows 8, which ensures compatibility with the Kinect SDK and the ability to connect to the Rod Control Board. The computer is one which was available to the team, and not custom-built for this project. It contains two 8-core AMD processors running at 2.3GHz, coupled with 32GB of RAM. This provides an unrestrictive environment in which to prototype and test code quickly. It also allows for parallelization, which improves total system performance. Except for the code running on the Rod Control Board, all code for this project was written in C#.

B. Initial Phase: Preparation

The initial phase of the Table State Interpretation subsystem is a preparation phase which is run once on each boot, which locates the table and calibrates the TSI system's table depth level. It does this by first taking 300 depth images of the table using the Kinect. These are then algorithmically processed using a Gaussian blur weighted in the X-direction, which helps eliminate puppets (as they are oriented that way). The depth level of these processed images is then averaged to obtain the table's normalized depth level.

C. Phase One: Table Surface Identification

This phase of the TSI subsystem begins the per-frame processing stages, hence the label of "Phase One". For the incoming depth frame, the table surface is identified by subtracting out the normalized value obtained in the initial phase, and then searching within a hardcoded range of that value to determine the location and shape of the table for this particular frame. This is necessary as neither the camera nor the table is rigidly fixed to any reference location, so they are capable of moving slightly relative to each other, necessitating table surface identification. A picture of the board surface with identified, highlighted board surface can be seen in Fig. 3. This algorithm also determines the four corners of the table, which are used as a reference for the coordinate system.

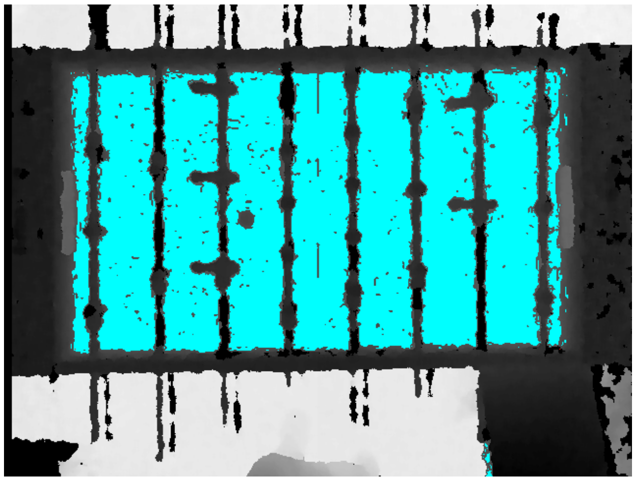


Fig. 3: Identified Table Surface

D. Phase Two: Ball Candidate Detection

Now that the table can be located in the frame, it is necessary to determine the coordinates of ball candidates. In order to do this, the Emgu CV implementation of the circular Hough transform is used. (Emgu CV is a C# wrapper for OpenCV, which allows OpenCV functions to be called from C#). The circular Hough transform locates all objects on the table surface that are roughly circular (above a threshold tolerance) and these coordinates move on to the next phase. In Fig. 4, the ball candidates can be seen as circles drawn over the table.

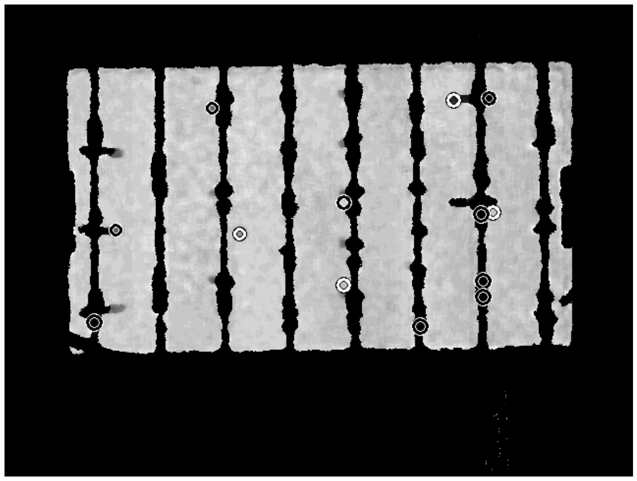


Fig. 4: Ball Candidates

E. Phase Three: Candidate Elimination

The primary circular object on the table besides the ball is typically the puppets, in normal operation. Since there are so many of them, and they are always on the table, they are a major source of noise that must be eliminated. In this phase, a breadth-first search is instantiated from each set of candidate ball coordinates. The actual ball will not be connected to anything, but a puppet will be connected to the rod, other puppets, etc. This helps reduce false ball detections by a large factor.

F. Phase Four: Candidate Ranking

In the final TSI phase, the remaining candidates are investigated further and ranked according to likelihood of being the ball. First, a Sobel gradient is determined around the remaining candidates' locations. Then, a Hough accumulator is run on these gradients. This generates an index of circularity, which can be compared among candidates. The most circular candidate is found, and then it and other candidates within a threshold of it are outputted to the physics engine as probable ball candidates.

G. Overall Algorithmic Performance

Overall, the TSI algorithms are very efficient compared to early prototypes. Effective use of parallelization throughout the subsystem has resulted in dramatic decreases in latency and increases in throughput. The system is currently capable of operating at 28.3fps, very near the maximum of 30fps allowed by the Kinect sensor. Additionally, the total lag is even lower than the target of 75ms, by a factor of 2/3. This can be seen in Figure 5:

	Actual	**Target**
Dropped frames	5%	0%
Effective framerate	28.3 fps	30fps
Average lag	50ms	75ms

Fig. 5. Table of Actual vs. Target TSI Performance

VI. Physics Engine & Ball Tracking

Real-time images from the Kinect sensor are subject to noise. Even a sophisticated and well-tuned circle detection algorithm will be susceptible to false positive detections and missed detections of the foosball. For these reasons, the Physics Engine and Ball Tracking subsystem is introduced between the TSI and the AI subsystems. When the TSI subsystem detects a circle on the field, it will pass the point to the ball tracking system. The ball tracking system will then integrate all of the potential ball positions and output to the AI the most likely actual position of the ball.

In the design phase of FOOSE, the intention was to take the potential ball detections and create a unified physical position through a Kalman filter. However, the Kalman filter was decided to not be appropriate for our domain. The problem lies in that erroneous readings from the Kinect potentially do not correlate to the ball position. Unlike in a domain like Satellite Global Positioning, where you may get incorrect readings that are still correlated to your actual position, the false positives from our algorithm do not give us any statistical insight into our actual position. Instead, we need a way to detect readings that are not the ball and filter them out completely.

The need to filter out false readings lead to the unique algorithm constructed for our ball tracking subsystem. The algorithm is based on a list of *watchers* which keep track of potential ball positions on the field. A watcher contains a physics model that it uses to try to predict where the ball should be in future ticks. Whenever a new circle position is read from the camera, each watcher in turn tries to integrate the new position into its current ball model. The model that most accurately predicts the new sensed position will then integrate it into its velocity and position predictions; otherwise, if no watcher can accurately integrate the position then a new watcher is spawned to model a ball located at that position. Then whenever the AI requests the position of the ball, the watcher with the most accepted ball updates (i.e. the most accurate ball model) will be selected to return the estimated position. Over time, watchers that are not getting updates from the camera are killed off.

This system allows a dynamic method of filtering out erroneous readings. As long as the ball is the most consistently detected circle on the field, then its watcher will always have the highest confidence readings, while the noisy circle detections will die off. The power of this system comes from its ability to continue to track the ball after a very rapid hit, when the camera will not be able to keep up with the foosball's movement. This sort of rapid change in ball position may initially look like an erroneous reading to the tracker, but as the confidence of the new position increases it will quickly usurp the old watcher and take its place as the new most accurate location of the ball.

In Figure 6, the output of the physics engine can be seen. Crosses represent previous locations at which the

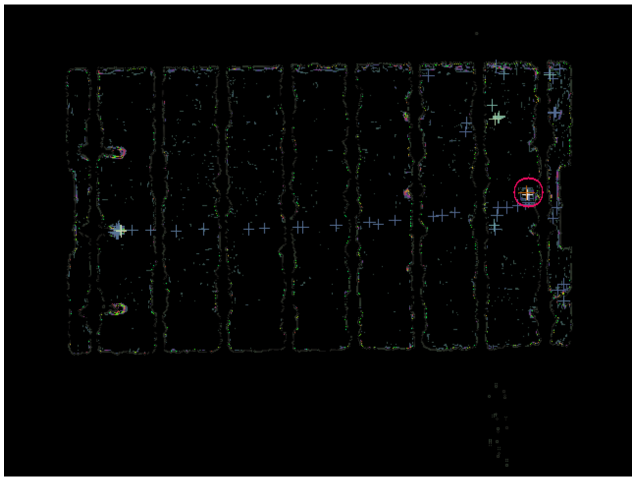


Fig. 6: Physics Engine Output

ball had been found, and the circle represents the current, predicted location of the ball.

VII. Artificial Intelligence

The artificial intelligence (AI) subsystem is responsible for calculating a move based on present and former table states, and outputting those moves to the appropriate Rod Control Boards.

A. Lateral Move Calculation

The AI subsystem begins when it is called by the physics engine with the most recent best coordinates for the ball. From these coordinates and the previous set of coordinates, it projects a line and finds the intersections with each rod, accounting for bounce. It then uses the calculated intersection points to choose a puppet to block at that intersection. Last, it uses a measured offset to calculate how many stepper motor "clicks" the rod must move in order to move the correct puppet to the correct position. It then saves this move.

B. Kick Calculation

Once all lateral moves have been calculated, kicks are determined. Because of the way kicking is implemented on a mechanical level, it is necessary to prepare for a kick before actually kicking. Fortunately, this is fairly easy to mitigate. Using the ball's velocity in the X-direction (determined by the physics engine) and the ball's X-position relative to the rod in question, it is easy to determine how long it will take the ball to intersect the rod. The AI contains the tested values regarding how long a kick takes to charge. So, the AI can determine whether to charge, kick, or not kick depending on how long it will take the ball to intersect the particular rod.

C. Move Output

After the determination of the lateral move and kick value for each Rod Control Board, the move calculation is complete. The moves are then output to their respective RCBs (determined during initialization, see below section) in the form of a three-byte value, where the first two bytes represent the lateral move (final absolute position, in clicks) and the final byte is either a 1 or a 0, where 0 is block, 1 is charge kick, and going from 1 to 0 actively kicks. The final "kick" byte is also used in initialization and calibration, as seen below. The RCBs immediately begin execution of the received move, and the AI subsystem awaits further input from the physics engine.

D. Initialization

As the AI is the intermediary between the computer and all four Rod Control Boards (RCB), it is responsible for certain functions outside of its normal move calculation. The first of these is initialization. When the system is booting up, the initialize function of the AI program is called, which automatically determines the correct serial port with which to address each RCB. It does this by enumerating each serial port in the system, and challenging each of them with a three-byte value (00 00 FF), which the code on the RCB will interpret as a request for board identification. An RCB will respond with one byte in the form of AX, where X is its number. So, RCB 3 will reply with 1010 0011. The AI then parses this and sorts the RCB's port into an array where it can be used later.

E. Calibration

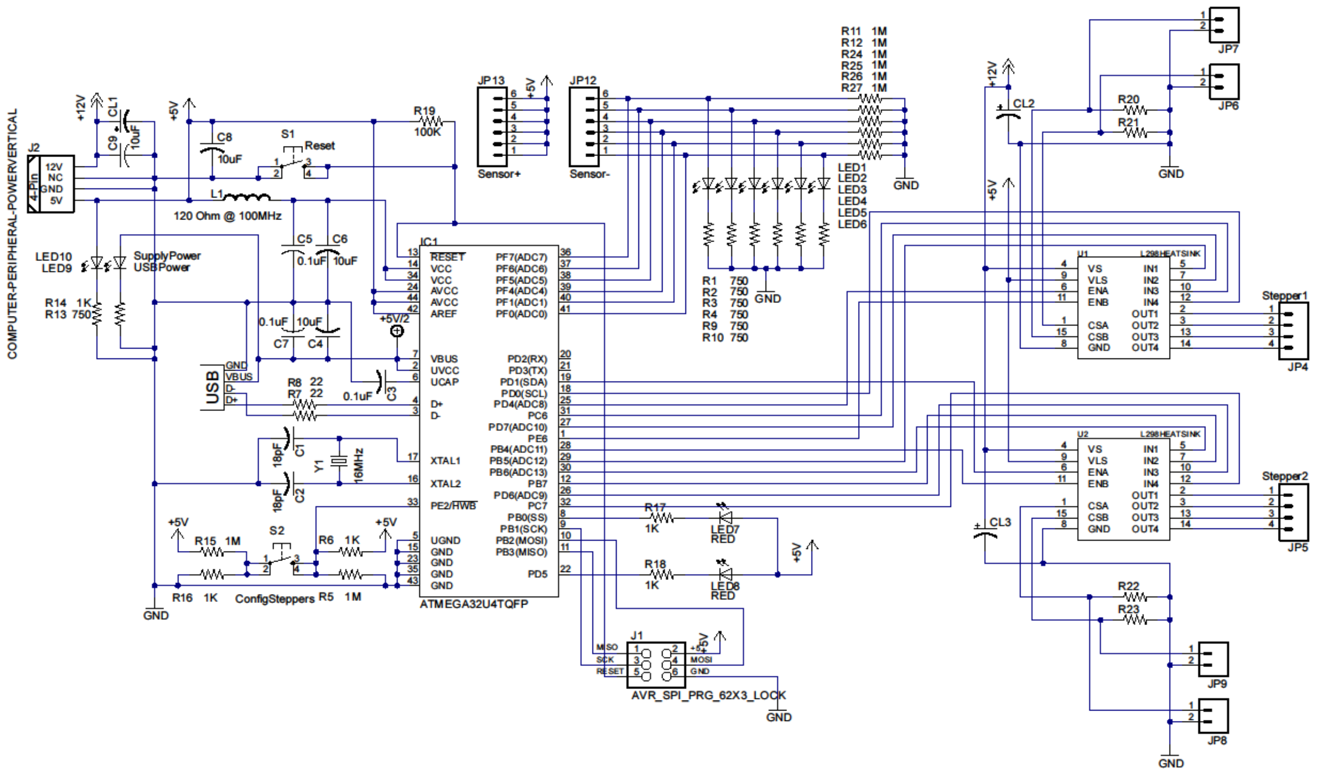
The AI is also responsible for requesting calibration updates from the RCBs themselves. The AI keeps an internal model of how many clicks each rod is capable of moving. While this number would be static under ideal conditions, due to changing conditions this number may change by a few clicks. The RCB automatically calibrates on power-up and stores this information. The AI requests this information during initialization in order to maintain an accurate internal representation of the board. Like the ID request, the calibration request is a 3-byte value; namely 00 00 F0. The RCB will respond with two bytes representing its maximum range in clicks. The AI then updates its internal information to match the measured values.

VIII. Rod Control Board

A. Overview

The Rod Control Board (RCB) is a custom-printed two-layer PCB which is used to control and power the stepper motors used in the mechanical subsystem. It connects to the computer via a Serial over USB connection, which the microcontroller, an Atmel ATMega32U4, is able to support natively. The microcontroller takes in moves from the AI via the Serial over USB line, which it uses to power the motors and execute the move. In this project, one RCB is used per rod controlled by the computer, for a total of four RCBs. The schematic for the RCB can be seen in Figure 7.

B. Microcontroller



2013-02-21 14:55:03 f=1.50 C:\Users\fenelonp\Dropbox\SeniorDesign\RCB3\RCB3.sch (Sheet: 1/1)

Fig. 7: RCB Schematic

As aforementioned, the RCB uses as its center the Atmel ATMega32U4 microcontroller. For this project, this model of ATMega proved to be ideal. It supports USB without the need for additional complex circuitry and has low cost. Also, the Arduino Leonardo uses the same model, which allowed the use of its bootloader. Using the Arduino bootloader greatly simplified coding for the ATMega, as it allowed the ATMega to communicate with the computer over USB directly using Arduino drivers, which saved a great deal of time in development, as this project requires two-way communication between the computer and the RCBs. The ATMega is being run at 16MHz, which is the speed supported by the Arduino bootloader, and a sufficiently high speed to be able to operate more or less instantaneously in normal operation. The ATMega's other specifications also turned out to be satisfactory for the FOOSE project, as seen in Figure 8. Its other useful features include many IO pins, which are mainly useful for controlling the H-bridges, but are also used for setting the RCB ID (two sets of two pins are sensed as binary bits and used for initialization, see Section VII, part D) and to sense the buttons used in the calibration process (section VII part E).

ATMega32U4 Specs	
Flash (Kbytes):	32 Kbytes
Pin Count:	44
Max Frequency:	16 MHz
Max I/O Pins:	26
USB Transceiver:	1
USB Speed:	Full Speed
SPI:	2

Fig. 8. ATMega32U4 Specifications [1]

C. Code

The RCB's microcontroller is capable of running C code. The code that runs on the RCB is responsible for all aspects of RCB control, including the aforementioned initialization and calibration routines (section VII parts D and E).

Primarily, the RCB code is responsible for executing the moves passed to the RCB by the AI subsystem. At all times, the RCB knows the rod's current position and velocity. It is constantly polling for a new move. When it receives a new move (which is formatted in terms of absolute clicks of the stepper motors), it calculates the direction and distance it needs to travel. Then, it either continues to the new destination or, needing to change directions, steps down the speed and then steps back up in the new direction. This slowing down is too quick to really perceive; the rod can change directions extremely quickly, a clear advantage in a foosball game. However quick, the slow down/speed up is still a necessary component of the code, as the stepper motors would exhibit degraded performance if the step down were not implemented.

The RCB code also governs kicking, of course. As aforementioned in section VII, part B, it is necessary to prepare for a kick before executing one. When the RCB receives a kick byte equal to 1, it charges the kick by rotating the kick stepper 90 degrees. When it receives a 0 after that, it rotates the rest of the way (270 degrees), thereby executing a kick and resetting the system. For more information about the mechanics of kicking, see section IX.

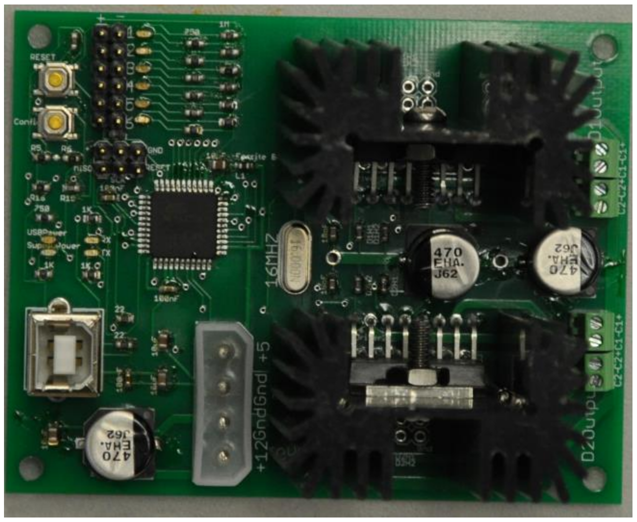


Fig. 9: Assembled RCB

D. H-Bridges

The H-bridges are perhaps the most important component of the RCB. They are responsible for controlling power output to the stepper motors. Essentially, they act as switches. For the RCB, the L298N full-bridge motor drivers are used. Each H-bridge is capable of controlling two power channels. As each stepper motor has two coils which need independent power, and each rod has two stepper motors, each RCB requires two H-bridges. The RCB's microcontroller controls the H-bridges, and the H-bridges route power to the stepper motors.

The L298N H-bridges have proven to be ideal for the FOOSE project. They are inexpensive, high-current (2 amps per channel), and reliable. [2] Early versions of the RCB utilized a more complex Texas Instruments chip specifically designed to be a stepper motor driver, but this component proved to be much less reliable than the H-bridges the group members had already been using for testing. They would frequently overheat and stop working, and it was discovered that the chip was not capable of sustaining the sort of power output the stepper motors required. The L298N has proven to be a reliable solution.

In testing with the H-bridges, it was found that, when run for long periods of time, they would experience thermal events much like the drivers. Unlike the drivers, the H-bridges can be easily mounted to heatsinks, and these can be seen as a prominent feature of the RCB's final design. The heatsinks (in combination with thermal paste) effectively absorb and dissipate excess heat from the H-bridges, even when run for long periods of time. This has greatly increased the reliability of the RCB's during extended gameplay. In fact, since being mounted to the heatsinks, the RCBs have not encountered any thermal faults at all.

E. Future Work

While the current RCB design seems solid, there are two major areas to which further work could be directed post-senior design.

The first of these is to increase the voltage on the stepper motors. Currently, they operate at 12V from standard computer power supplies. However, it may be possible to increase the performance of the system by doubling this to 24V, either from chained power supplies, or from dedicated 24V power supplies. This option was considered by the group, but ultimately not pursued due to time constraints.

The second option concerns the reliability of the RCBs. While currently very reliable in operation, it may be possible to prevent issues in the future by incorporating an array of diodes between the RCB output and the stepper motors. This would prevent transient current spikes caused by the inductive load of the stepper motors, and would protect the board from any anomalies in the stepper motors. Again, this option was favored, but ultimately not pursued due to time constraints.

IX. MECHANICS

A benefit of the foosball domain is that the game involves only two simple motions to play: linear motion to move the puppets back and forth, and rotational motion to kick the ball. This is a desirable quality, as none of the members of the FOOSE engineering group are mechanical engineers. However, combining the two motions into a single elegant component is non-trivial. This section discusses the engineering challenge of creating the final mechanical subsystem.

The starting point for the mechanical subsystem was to estimate the physical speed and forces required to maintain a game of foosball that is "entertaining to a novice user." After watching recordings of foosball games, it was determined that the fastest the ball speed that a novice user should be able to block is around 1 m/s. This is the maximum speed we expect the table to be able to block, and the value that is used to calculate a minimum torque requirement for the motors as shown in section III, part B. The stepper motors that were purchased for this task (Japan Servo KH56JM2-901) are well within the minimum torque to create the linear speed.

However, the Japan Servo motors could not meet the torque requirements for the rotational kick; in fact, none of the motors on our list of potential parts had quite enough torque to create a competitive kick. To get around this limitation, the senior design group designed a cam kicking mechanism that uses a spring force to load the kick, and

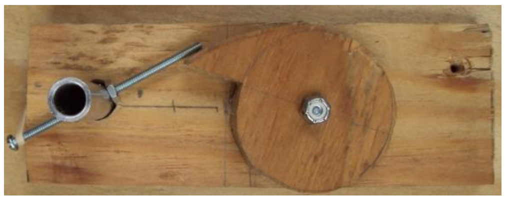


Fig. 10: Kick Cam

using the stepper motor instead only as the force to wind back the kick, as seen in Fig. 10. This solution is compact and can fit easily on a linear slider. A picture of the cam device can be seen in Fig. X.

X. Conclusion

In conclusion, the FOOSE project offers an innovative, cost-effective, and competent solution to a tough engineering problem. It successfully incorporates subsystems from computer vision, physics modeling, artificial intelligence, circuit design, and mechanical engineering into an entertaining game that fits in the den.

Acknowledgements

The members of FOOSE would like to give special thanks to Soar Technologies and its employees for their sponsorship and invaluable advice and expertise, without whom the FOOSE project would not have been possible.

Special thanks are also due to Dr. Samuel Richie, whose guidance and advice have been essential to getting the project to where it is today.

Skyler Goodell would like to thank Brandon Parmeter, for his robotics expertise and advice.

Biographies

Nathaniel Enos is a senior at the University of Central Florida, majoring in Electrical Engineering with a minor in business (2013 graduation). Nathaniel enjoys system controls and embedded processes. After graduation he will start a rotational program at Texas Instruments as a digital applications engineer.

Patrick Fenelon is a senior at the University of Central Florida. He will be graduating with a B.S. in Computer Engineering in May 2013. His interests include programming, math, computational geometry, and solving problems. After graduation, he will begin working on the Visual Studio debugger team at Microsoft.

Skyler Goodell is a senior at the University of Central Florida. He will be graduating with a B.S. in Computer Engineering in May 2013. His interests include machine learning, serious games and robotics. He will begin work at Microsoft on the Bing search engine shortly after graduation.

Nicholas Phillips is a senior at the University of Central Florida. He will be graduating with a Bachelor's of Computer Engineering in May 2013. His interests include reading, writing, and synthesizing information. After graduation, he will be working for Verizon Communications.

References

[1] Atmel. (2010). *ATmega32U4 Summary* [Online] Available: http://www.atmel.com/Images/7766S.pdf
[2] Sparkfun. *Full-Bridge Motor Driver Dual - L298N* [Online] Available: http://www.sparkfun.com/products/9479