

# Striker: Autonomous Robotic Air Hockey Experience

Efrain Cruz, Loubens Decamp, Luis Narvaez,  
Brian Thomas

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

**Abstract** — The objective of this project is to create an autonomous robotic arm that has the capabilities of playing air hockey against a human opponent. Using computer vision algorithms, a camera will detect the puck. Centroid coordinates and trajectory coordinates are sent through Bluetooth communication protocols. The robotic arm receives the coordinates and moves to the position that corresponds to the trajectory coordinates. Additional features include a screen for replays and displaying scores, puck return system, and LED strips for lighting effects.

**Index Terms** — Bluetooth, computer vision, end effector, microcontrollers, motion analysis, power control.

## I. INTRODUCTION

Striker represents air hockey re-imagined for the avid player. The premise is that a human player plays its counterpart in the form of an autonomous robot that will have multiple playing difficulties to challenge all skill levels that play Striker. The gamer will have the option of selecting particular playing difficulties by virtue of a smart phone application.

The Air Hockey experience mimics an arcade game experience. After scoring against the robotic opponent sounds play, lights flash, and an instant replay will play on a screen. Afterwards, a puck return mechanism, composed of motors and belts, will direct the puck back to the player for convenience. The screen used to play instant replays displays time, score, and various statistics.

## II. Subsystems

Striker is comprised of several subsystems. The subsystems are user interface, tracking, robotic arm, audio and visual, puck return, and power supply. The subsystems are all interconnected through the main system.

### A. User Interface

The initial interaction the player has with Striker is through an Android Application. The application was created through Eclipse SDK with java.

The application begins by displaying home screen and allows the player to press the play button. After selecting play, it will give the user the option to play against the robotic arm, or to control the robotic arm. Upon making the decision, the application will display the Bluetooth address that corresponds with Striker's Bluetooth module. This Bluetooth module is connected to the serial communication pins on the Raspberry Pi. By pressing the displayed address, connection to Striker will initiate. A message displaying "Connected" confirms connection to Striker.

If the player has chosen to play as the robotic arm, a seek bar will be displayed on the screen. The position of the seek bar corresponds to the location of the robotic arm. If the player chooses to play against the robotic arm, the tracking system calculations determine the location of the robotic arm. At conclusion of game, player can press restart to go back to the home screen.

### B. Tracking

The tracking subsystem uses the Raspberry Pi as the controller. The Raspberry Pi operates on the Raspbian operating system. The inputs for the computer vision come from the Raspberry Pi camera board, which is mounted over the air hockey playing surface. The camera board is a 5 MP Omnivision 5647 sensor and connects to the Raspberry Pi using short ribbon cable. This cable connects to the BCM2835 processor of the Raspberry Pi through the CSI bus [1].

The lens on Raspberry Pi camera does not have a wide enough angle to capture sufficient area on the air hockey playing table. The solution was to buy a wide screen lens. The lens we are using is a 0.4x super wide angle lens with a viewing angle of 140°. This lens allows the camera to track the puck on  $\frac{3}{4}$  of the table length.

The computer vision algorithm is written in Python with OpenCV libraries included. Figure 1 displays the flow of the programming code. The algorithm starts by capturing a frame from the camera; it then reduces the quality, converts the image from RGB values to HSV values and filters for the range of HSV values. The puck's color is orange, which contrasts well against the white playing area on the air hockey table. Therefore, the range of colors that the filter is set to be looking for corresponds with light orange to dark oranges. The range for HSV values in the algorithm is (150, 50, 0) to (180, 256, 256).

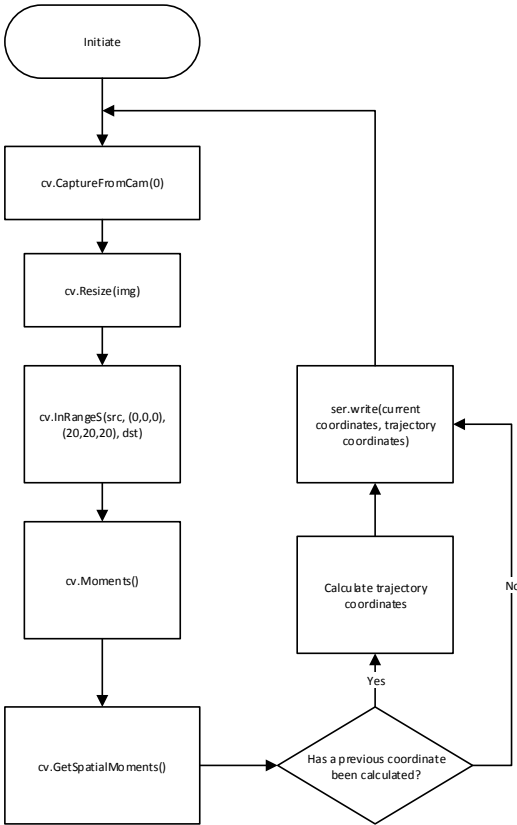


Fig.1. Program flow for puck tracking algorithm

From the filter, the algorithm creates a new image. The new image displays white and black pixels. The white pixels represent orange objects and black pixels represents any color outside of the filter. Figure 2 displays the new image that is created through the HSV filter, with respect to the original image. The white pixels are used to calculate the centroid of the puck according to pixel coordinates. With every two calculated centroid coordinates, the algorithm is able to calculate trajectory coordinates using trigonometric identities. The trajectory coordinates specify at what location the puck will cross the translational axis of the robotic arm.

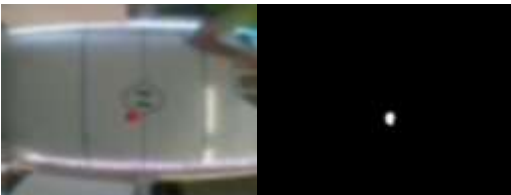


Fig.2. Original image with puck on the table (left). Filtered image (right).

The data for the coordinates is then outputted through Raspberry Pi ttyAMA0 serial port. Pins 8 and 10 correspond with TX and RX, respectively. The TX pin is connected direction to the robotic arm's processor and to the main systems processor. The RX pin is connected to the RXN 42 Bluetooth module. Pin 2 and 3 on the module correspond with TX and RX, respectively. This pin receives data from the android application if the player has chosen to control the robotic arm instead of playing against in.

After transmitting, the code captures another frame from the Raspberry Pi camera board and loops again through the code.

### C. Robotic Arm

The robotic arm is mechanically designed to move laterally on a track. The driving system of the end-effector is a belt-driven pulley that is connected to a bipolar stepper motor. There are additional sub-components that contribute to the final design that include a servo for yaw, solenoid for striking the puck and the stepper motor for driving the arm. Coordinates of the puck's position will be determined by the tracking system's algorithm as explained in section B. Since we want to reduce latency and increase reaction time, we decided to have a dedicated microcontroller specifically to handle robot arm operations. This will be controlled by Atmel's Atmega328.

Our servomotor is used to provide a yaw motion to the end-effector's mallet. This will guide the strike toward the human opponent's goal, adding a dynamic experience with a challenging opponent. For our servomotor, we will have an open loop design in which preset angles will be programmed into the robot arm microcontroller. We will have 24 preset values that are in increments of 6 degrees. These preset values were determined while testing the system and measuring distances to the goal. They are also influenced by the x-axis limit programmed into the controller, which is set at 24. Yaw movement is limited to +/- 15 degrees off center. This constraint was established by having the end-effector placed at an extreme x-coordinate or lateral position and assuming the puck's trajectory is coming from the midpoint of the table.

The solenoid that will be used to perform the *striking* motion will be attached to the end-effector housing. Based on our algorithm, when the puck has reached the end-effector's position, a signal will be sent to activate the solenoid. The solenoid used in our testing was a 'pull' solenoid, thus is used in our final design. We are using the pull solenoid due to its lighter weight compared to the push solenoid obtained as well as the striking force and

stroke range of 0–4 in. The circuit configuration of the solenoid is further explained in the PCB subsection.

In order for the solenoid to accurately activate when the puck reaches its zero position on the y-axis, we must send the digital pulse signal prior to the puck reaching the end-effector. This will compensate for the latency in data transmission, processing and puck speed. Through various tests, the optimum y-coordinate to send the signal is when the puck reaches ‘05’.

In order for the linear motion to occur, our stepper motor has to have a correct control signal. The stepper motor in use is a NEMA 14 sized motor. Because stepper motors move based on a series of pulses, it is imperative that the timing sequence of the control signals for each phase be correctly programmed. For our particular project, we chose to go with full step timing sequence. Although it is the least efficient control method, we are able to obtain better torque-speed ratios doing so. The full-step timing sequence used in our project can be seen in Figure 3 (a) and (b). You can see that the continuous rotation is achieved by keeping two opposite poles on opposite phases energized at all times.

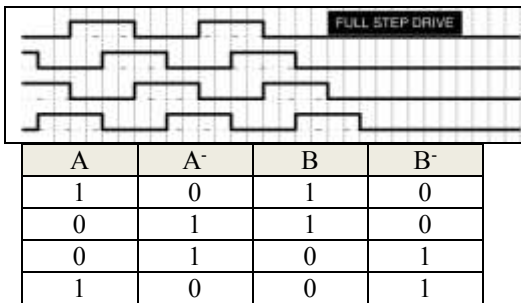


Fig. 3. Timing Sequence Used for Stepper Motor Control

The program flow of our robotic arm is programmed primarily around Arduino’s built-in libraries and classes. The main class we are using for the stepper control is the ‘myStepper’ class. Our algorithm is laid out in Figure #. You can see that during game play, the Raspberry pi will transmit a single x-coordinate representing the *projected* position at Y=0. This X-coordinate will be used to move the end-effector into position while the Y-coordinate will continuously feed into the buffer. After the arm moves to the predicted X position, a comparison of the Y-coordinate will be executed to determine when to actuate the mallet.

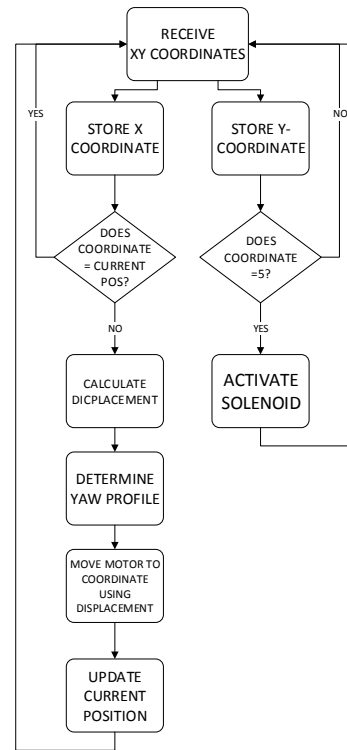


Fig. 4. Software Flow Diagram of Robot Algorithm

Due to the design constraints of our power supply and torque-speed characteristics of our motor, we determined the optimum speed of the motor is 300 rpm. This is significantly less than the required specifications, however in order to obtain higher speeds, the torque would be sacrificed, preventing movement of the end-effector. As seen in Figure 5, the excerpt shows how we vary the no-load rotation speed in our algorithm through one of Arduino’s built-in library functions for stepper motors.

```

//Setup Pin Descriptions and Deal Data for Stepper
void setup() {
  pinMode(A,OUTPUT);
  pinMode(Abar,OUTPUT);
  pinMode(B,OUTPUT);
  pinMode(Bbar,OUTPUT);
  myStepper.setSpeed(300); //set rpm
  Serial.begin(115200); //set baudrate
}

```

Fig. 5. Striker setup excerpt

#### D. Audio and Visual

LED lighting is incorporated to add visual appeal to the overall gaming experience. Addressable RGB LED’s were chosen based on their multitude of color options. In particular, the LPD8806 addressable LED has 3 channels of output (RGB) consisting of 7 bits per channel resulting

in 2,097,152 color variations. PWM is used to control the brightness of the corresponding RGB output [2].

When using LED's, careful consideration as it relates to current draw needs to be monitored. 5 volts with a current supply of 2 amps is suitable for a strip that is 5 feet long under full illumination, 100 percent duty cycle. However, this scenario is seldom seen especially when LED brightness is constantly changing. Furthermore, 50 percent duty cycle during constant illumination is more than adequate brightness. The Arduino language allows the programmer to calibrate the duty cycle by setting values from 0 to 255 as illustrated in figure 6. This numeric range translates to 0% duty-cycle to 100% duty-cycle.

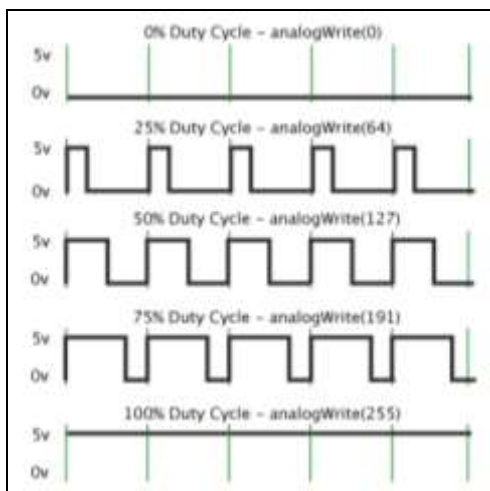


Fig.6. Arduino PWM Protocol

To safe guard against excessive current consumption, LED brightness was adjusted to 127 in the Arduino code which equates to 50% duty-cycle. Testing the LED design schemes that are incorporated in striker draw a max current of 212 mA which is well within our design specifications for power supply.

Originally, one functional use of the LED's was to have them duplicate the puck movement along the length of the table. Background LED's will be white while a single red LED will illuminate to indicate the puck's position. This is facilitated by having the LED's represent integers. The corresponding y-coordinates are converted to integer values (1-54) representing the number of LED's. However, this feature was not feasible due in part that the tracking system could only register two y-values length wise on the table. Consequently, an LED game mode program is employed to enhance the gaming experience and aid in the tracking of the puck. A white background with a blue and green strobe pattern is used. This particular

arrangement was determined to be optimal during testing procedures for the tracking system.

```

void setup() {
  pinMode(4, OUTPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);
  pinMode(7, OUTPUT);
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);
  pinMode(14, OUTPUT);
  pinMode(15, OUTPUT);
  pinMode(16, OUTPUT);
  pinMode(17, OUTPUT);
  pinMode(18, OUTPUT);
  pinMode(19, OUTPUT);
  pinMode(20, OUTPUT);
  pinMode(21, OUTPUT);
  pinMode(22, OUTPUT);
  pinMode(23, OUTPUT);
  pinMode(24, OUTPUT);
  pinMode(25, OUTPUT);
  pinMode(26, OUTPUT);
  pinMode(27, OUTPUT);
  pinMode(28, OUTPUT);
  pinMode(29, OUTPUT);
  pinMode(30, OUTPUT);
  pinMode(31, OUTPUT);
  pinMode(32, OUTPUT);
  pinMode(33, OUTPUT);
  pinMode(34, OUTPUT);
  pinMode(35, OUTPUT);
  pinMode(36, OUTPUT);
  pinMode(37, OUTPUT);
  pinMode(38, OUTPUT);
  pinMode(39, OUTPUT);
  pinMode(40, OUTPUT);
  pinMode(41, OUTPUT);
  pinMode(42, OUTPUT);
  pinMode(43, OUTPUT);
  pinMode(44, OUTPUT);
  pinMode(45, OUTPUT);
  pinMode(46, OUTPUT);
  pinMode(47, OUTPUT);
  pinMode(48, OUTPUT);
  pinMode(49, OUTPUT);
  pinMode(50, OUTPUT);
  pinMode(51, OUTPUT);
  pinMode(52, OUTPUT);
  pinMode(53, OUTPUT);
  pinMode(54, OUTPUT);
}

void loop() {
  digitalWrite(4, HIGH);
  digitalWrite(5, HIGH);
  digitalWrite(6, HIGH);
  digitalWrite(7, HIGH);
  digitalWrite(8, HIGH);
  digitalWrite(9, HIGH);
  digitalWrite(10, HIGH);
  digitalWrite(11, HIGH);
  digitalWrite(12, HIGH);
  digitalWrite(13, HIGH);
  digitalWrite(14, HIGH);
  digitalWrite(15, HIGH);
  digitalWrite(16, HIGH);
  digitalWrite(17, HIGH);
  digitalWrite(18, HIGH);
  digitalWrite(19, HIGH);
  digitalWrite(20, HIGH);
  digitalWrite(21, HIGH);
  digitalWrite(22, HIGH);
  digitalWrite(23, HIGH);
  digitalWrite(24, HIGH);
  digitalWrite(25, HIGH);
  digitalWrite(26, HIGH);
  digitalWrite(27, HIGH);
  digitalWrite(28, HIGH);
  digitalWrite(29, HIGH);
  digitalWrite(30, HIGH);
  digitalWrite(31, HIGH);
  digitalWrite(32, HIGH);
  digitalWrite(33, HIGH);
  digitalWrite(34, HIGH);
  digitalWrite(35, HIGH);
  digitalWrite(36, HIGH);
  digitalWrite(37, HIGH);
  digitalWrite(38, HIGH);
  digitalWrite(39, HIGH);
  digitalWrite(40, HIGH);
  digitalWrite(41, HIGH);
  digitalWrite(42, HIGH);
  digitalWrite(43, HIGH);
  digitalWrite(44, HIGH);
  digitalWrite(45, HIGH);
  digitalWrite(46, HIGH);
  digitalWrite(47, HIGH);
  digitalWrite(48, HIGH);
  digitalWrite(49, HIGH);
  digitalWrite(50, HIGH);
  digitalWrite(51, HIGH);
  digitalWrite(52, HIGH);
  digitalWrite(53, HIGH);
  digitalWrite(54, HIGH);
}

```

Fig. 7. LED In-Game Play Lighting Effect

An alternative use for the LED's is to implement a celebration display when a goal is scored by either robotic arm or human opponent. An algorithm was created based upon sample codes available from the LPD8806 library. The goal was to create a dynamic display for visual appeal.

Another visual as well as an audio component is incorporated in Striker. A 15.6" monitor is displayed on the support structure above the playing surface along with speakers. The goal is to incorporate video instant replay with background music accompaniment. To facilitate the audio/video goals, a Texas Instruments DM365 (Davinci processor) is developed via a Leopard board 365 with a 5 mega-pixel cmos camera from Leopard imaging. Ideally a video resolution of 720p at 30 frames per second is suitable for the monitor being used. Moreover, it is convenient to have a device that can manipulate mp3 audio files as well [3].

The DM365 requires a Linux environment to manipulate the SDK that is used to program the DM365. RidgeRun embedded solutions provides a free SDK that is used on the Leopard board. For that reason it is necessary install the current version of virtual box on the host PC with Ubuntu 12.04 LTS OS installed within the virtual box.

Creating the SDK is initiated by installing code soucery tool chain IDE which is used to create and debug environment for embedded C and C++. Next, Texas Instrument's SDK for the DM365 processor is installed and mounted to the code soucery tool chain. Lastly, the RidgeRun SDK is installed. The RidgeRun application is where the configuration of the SDK is done and subsequently sent to the target device for evaluation. Designing of the SDK is accomplished by opening a terminal window in Ubuntu running the 'make config' on the same header path where the Ridgerun SDK is located. For our project purposes, the SDK is set to boot from a SD

card. Once the settings are configured, the file images are sent to the location of the SD card on the host machine.

Now the SD card is removed from the host machine and attached to the Leopard Board which is the target machine. Connection between the host machine and target hardware device is accomplished by means of RS-232 serial communication. A serial communication protocol called minicom is opened and configured to the appropriate com port and baud rate of 115200. The boot procedure is initiated by pressing the reset button on the target device. The SD card boot process proceeds normally as outlined in the installation guide provided by RidgeRun. However, when prompted to select enter to activate the console, no interaction can be established. Unfortunately this feature of Striker was unable to be incorporated into the final design. Currently a suspected hardware issue with the Leopard Board is the reason for the inability to develop this sub system of Striker.

In lieu of this unfortunate circumstance, a seven segment numerical display will be used to keep track of scoring during game play. Ultimately the goal is to use the display to enhance Striker with an authentic gaming experience much like what you would see at an arena. Due to time constraints a launch pad evaluation board is used to configure the seven segment display.

### E. Power Supply

Power supply is one of the key elements needed for the functionality of this design. The air hockey table itself is powered through a 120V AC wall receptacle, which is converted into DC through a transformer and bridge rectifier. The bridge rectifier used for this design is KBPC6005, which is 50V/6A. Although a power system is already in place for the table, additional design is needed to power our sub-systems.

To power up our motors we decided to use a separate 24V/3A adjustable power supply. This was chosen for the motors to prevent circuit failure as both motors draw at least 1.5A with load.

For the other sub-systems, we designed a linear regulated power supply, which will provide specific output voltages of 12V, 9V, 5V and 3.3V, respectively. The 12V is used for Solenoid and Puck return. The 5V line is used for our LED display and our logic such as MCUs and drivers. The 3.3V is used for our Bluetooth module. Other key elements for the power supply design include capacitors, diodes, resistors, an LED light, and heat sinks.

The peak voltage after the rectifier is stepped down to 12V using an LM7812t voltage regulator which draw a maximum of 1A. Three other voltage regulators are used to complete the design: the LM7809, LM7805 and

LM1117v3.3. The LM7809 is a 9V/1A regulator, used as a bypass regulator. It reduces the power lost between the LM7812 and LM7805 and also reduces the heat dissipation that is coming from the LM7812 going to the LM7805. The LM7805 is our 5V/1A regulator. The LM1117V3 provides 3.3V max at 800mA. Heat sinks are placed on top or on the side of each regulator to prevent circuit from overheating. To reduce noise from the circuit we used electrolytic capacitors. The voltage rating of these capacitors must be higher than the output voltage of each regulator, that helps prevent circuit failure. The capacitor values we used were 2200  $\mu$ F, 1000 $\mu$ F, 470  $\mu$ F, and 10  $\mu$ F, respectively. The voltage of the capacitor chosen is 20% higher than the output voltage of the regulator. The voltage rating of the capacitors used in our design range from 35V to 50V.

An LED is placed in series with a 1K ohm resistor to indicate that Striker has been turned on and is receiving power. Figure 8 summarizes our power supply as described.

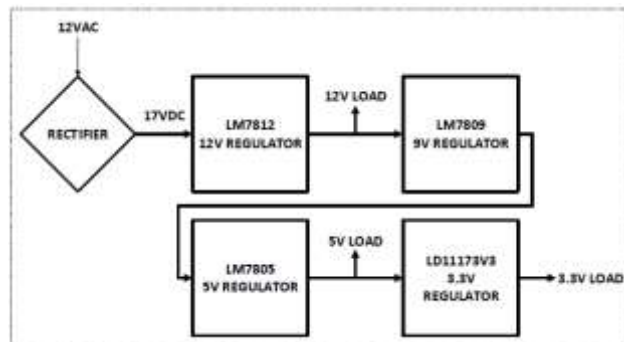


Fig. 8. Power supply circuit configuration

The circuit above, figure 8 was tested on a breadboard. After testing our design we found that our output voltages were very close to the values we were expecting (12.4V DC, 9.8VDC, 5.5VDC and 3.48VDC). Figures 9 and 10 are the screenshots of the power supply taken from an oscilloscope, respectively. As we can see, the signals have no ripple, which is due to the use of electrolytic capacitors.

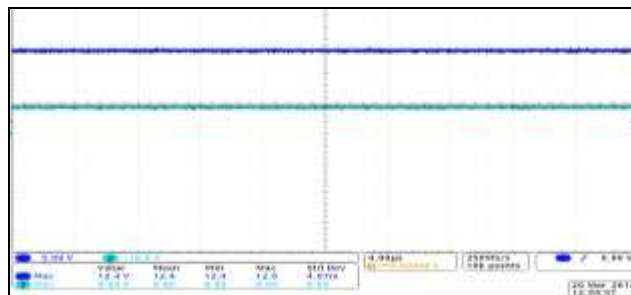


Fig. 9. 12v and 9V DC signal

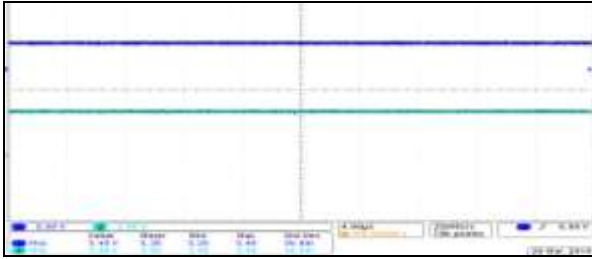


Fig. 10. 5V and 3.3V DC signal.

### E.1 Power Supply load calculation

To find the total load consumption by our design we summed up the maximum load of each sub-system and we made sure that what we are consuming is less than the maximum load that our power supply can provide. The maximum load available is calculated based on voltage and current rating of each supplier.

$$P_1 = 24V * 3A = 72 \text{ W (separate)}$$

$$P_2 = 17V * 3A = 51 \text{ W}$$

$$P_{\text{avail}} = 51 \text{ W}$$

Now, the total load consuming by our design is equal to the sum of the maximum load consumed by our LEDs, Gear motor, solenoid, communication system and microcontroller:

$$P_{\text{system}} = \sum P_i, 1 \leq i \leq 6$$

$$P_{\text{system}} = P_{\text{Gear}} + P_{\text{LEDs}} + P_{\text{MCU}} + P_{\text{Com}} + P_{\text{sol}}$$

$$P_{\text{system}} = 24V * 1A + 5V * .25A + 5V * .212A + 3.3V * .03A + 12V * 1A$$

$$P_{\text{system}} = 38.409 \text{ Watts}$$

75% of the available supply is used to feed our design.

### F. Puck Return

The main purpose of the puck return is to drive the air hockey puck back to the player when a goal is scored against the robot. For this mechanism we use the goal sensor (IR) which comes with table. This goal sensor sends an analog signal to our microcontroller which activates the motor for approximately 8 seconds. Figure 11 is a screenshot of the actual design.



Fig.11. Puck-Return Gear Motor Drive

### G. Main System

The main system uses an ATmega 328 to initiate the puck return system and to update the addressable LED strips with the current coordinates of the puck. It is named 'Main System' because originally this micro controller was going to initiate all the subsystems. However, the only system that requires initiation is the tracking system with the android application because all other subsystems are constantly in listening mode. Main system receives coordinates from Raspberry Pi through serial communication.

## III. PCB DESIGN

The entire circuit design for our main system and subsystems were designed to complement each other on one single printed circuit board. In this chapter, we will break down the design of our individual circuits by Motor control, Communications, End-Effector, and game effects.

### A. Motor Control

The motor control is driven by STMicroelectronics L29N8 dual H-bridge driver. This allows for switching of the motor directions without having to depend on electromechanical devices such as relays and contacts. The packaging we are using is the multiwatt15 vertical package. This will allow us to save space on our PCB and attach a heat sink. To compensate for back EMF to our components, we are using 1N5822 Schottky diodes. Due to the high current and voltage demands of our circuit, the trace-width for the 24V lines is set at 30 mils. This same circuit configuration will be used for the puck return as well, with 12VDC being supplied to the motor.

### B. Communications

Since communication will primarily be done through Bluetooth, we only need to use two pins of our controllers to receive and transmit data through UART. The RX and TX lines are pins 2 and 3, respectively, on the Atmega328. The RX lines will be shared by both Microcontrollers since they both need to monitor the Puck's Y-Coordinate. Our original design concept was to have the tracking system wirelessly communicate to the system

microcontroller in addition to the robot microcontroller. During testing, we found difficulties in pairing the devices through software. As a result, we are hardwiring the UART pins from the Raspberry pi to the UART pins of each Amega328. The Bluetooth will be connected to the Raspberry pi, sending signals from the user's android application to initiate a game, and manually control the robot arm, if selected.

### C. End-effector

As mentioned previously, the end-effector consists of a servo for yaw, and a solenoid for striking. The circuit configuration for the servo is quite simple, as it only requires a PWM digital output, and power lines of 5VDC. We mapped this external device's control to pin 5 of the microcontroller. Since it is strictly powering a control gate, there is no need for circuit protection from the servo. This will be connected through Molex header pins.

On the other hand, the solenoid is a bigger risk to damaging board components in our system. For this, we set our design as shown in Figure 12. In this configuration, we use the IRF620 power MOSFET driver to act as a high-powered Zener diode. This will divert the inductive kick from the solenoid to ground once the voltage exceeds 200V. Diode D11 acts as a clamping bypass diode. This will help dissipate the energy from the solenoid while in idle. 'SOLCTL' is the control line connecting to pin 13 of the robot Atmega328. The trace widths were widened to 30mil in order to accommodate for the possible high currents. The two leads on the solenoid will connect in parallel with the clamping diode, D11.

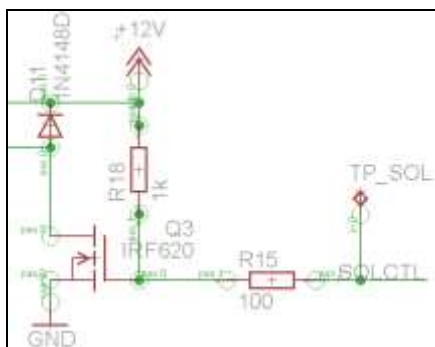


Fig.12. Solenoid Control Circuit

### D. Lighting Effects

The lighting is controlled via PWM signal as mentioned previously. We need a way to tie both strips to the same control signals so they may simultaneously animate. As

seen in Figure 13, we are using Molex header pins to connect the LED strips to the PCB, and tying both lines together to their respective control pins. The data and clock lines are connected to pins 12 and 11 on the system MCU, respectively.

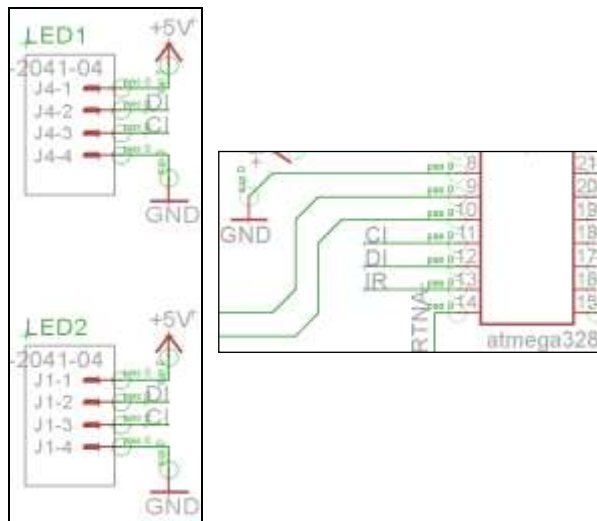


Fig.13. LED Lighting Control Connections

### E. Final PCB Design

The final PCB was designed using EAGLE 6.5 Professional. We are using a 2-Layer board in which the dimensions are 5.8" x 6". We ordered our PCB from Advanced Circuits at 4pcb.com. Figure 14 shows our final PCB.

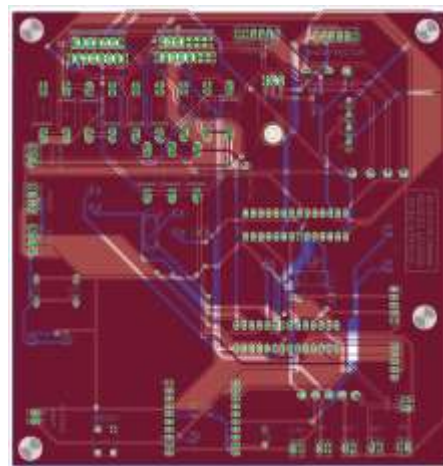


Fig.14. Striker PCB Top Layer

#### IV. ENGINEERS



**Efrain Cruz**, a senior student of the Electrical Engineering department of the University of Central Florida. He is currently an intern at Siemens Wind Power Americas, working in the Engineering Warranty department. He will be graduating May 2014.



**Loubens Decamp** is a senior Electrical engineering student at the University of Central Florida. He is a former Protection and control intern at Power Grid Engineering LLC. He will be graduating in May 2014; will start a new career with Lockheed Martin in

May 2014 in Akron, OH.



**Brian Thomas**, is an Electrical Engineering student at the University of Central Florida. He is graduating in May 2014 with a BSEE.



**Luis Narvaez** is a graduating senior Electrical Engineering student. He has been an intern for Universal Studios Engineering and Creative Engineering department since April 2013. After graduation, he will be hired as a technical consultant for Universal

Creative Engineering until he starts his job with Siemens Industry in Norcross, GA in July 2014.

#### V. CONCLUSION

Striker has been a challenging project. As with any design project, there were many obstacles that had to be overcome. Some of these obstacles included Bluetooth pairing connectivity and delays in manufacturing of tracking system components. Furthermore, testing of Striker's subsystems revealed that revisions needed to be implemented to improve overall performance. For

example, the end-effector needed to be redesigned to improve speed and overall functionality. Unfortunately, other subsystems namely the audio/video replay needed to be omitted.

These design setbacks are a part of the learning process that will acclimate our team to what is expected with our future job endeavors. The valuable lesson learned is communication has to be a premium. At this education level, everybody is individually talented. The difficult part is coalescing these individual abilities into a cohesive unit. Gladly, these obstacles were overcome using team work and solid engineering principles.

Ultimately, the goal of Striker is to have the end-effector interact with the puck. Realistically, game play is competitive with the average player (world air hockey champions are excluded). Moreover, aesthetic attributes like puck return, LED lighting, and remote game play are included to make Striker innovative. Striker as with all senior design projects are a reflection of the sum total of knowledge obtained at U.C.F.

#### VI. ACKNOWLEDGEMENT

The engineers wish to acknowledge the assistance and support of Dr. Richie. He has been a great source of knowledge and inspiration. His wealth of experience allowed us to remain focused and determined.

Additionally, the engineers would like to acknowledge our sponsors for their tremendous financial contributions. Thank you, Boeing and Power Grid Engineering.

Lastly, the engineers would like to thank their families and loved ones for all their support.

#### VII. REFERENCES

- [1] Adafruit, "Raspberry Pi Camera Board ID: 1367," Adafruit, [Online]. Available: <http://www.adafruit.com/products/1367#Description>.
- [2] T. Hirzel, "PWM," Arduino, [Online]. Available: <http://arduino.cc/en/Tutorial/PWM>.
- [3] Texas Instruments Wiki. (n.d.). *Getting Started Guide for Leopard Board DM365 and DM368*. Retrieved from [http://processors.wiki.ti.com/index.php/Getting\\_Started\\_Guide\\_for\\_Leopard\\_Board\\_DM365\\_and\\_DM368](http://processors.wiki.ti.com/index.php/Getting_Started_Guide_for_Leopard_Board_DM365_and_DM368)