Spring 2020

BitBilliards

Grant Reinbolt
gar20@zips.uakron.edu

Bit Billiards

Senior Design Project Final Report

Design Team #08

David Milostan (CpE)

Grant Reinbolt (EE)

Kyle Stevenson (EE)

Rodney Morgan (CpE)

Faculty Advisor: Osama Alkhateeb

Date Submitted: April 24th, 2020

Table of Contents

List of Figures

List of Table

Abstract (RM)

The goal of the project is to create a billiards table capable of keeping track of the score while simultaneously racking the sunk balls under the table. To achieve this, image processing will be utilized to detect and determine which balls are above and below the table. Upon leaving the table, the balls will be held by gates to prevent the cue ball from entering the reracking track. The balls will then enter the track and form a queue to be placed back into the triangle underneath the table. A gantry system consisting of an x and y axis will place the waiting billiard balls into their corresponding positions in the triangle. Throughout this process, data will be sent to a remote server which will provide the live game updates to the app. The app will be capable of showing the current score and status of the game, including images of the table. All of these improvements will help bring more viewers to the competitive pool scene while decreasing the delay between games.

1. Problem Statement

1.1: Need (GR/RM)

The American Poolplayers Association has over 250,000 members across 300 APA leagues in the US, Canada and Japan. These tournaments range from junior leagues to professional world championships with monetary payouts of over $2 million. However, the game of pool has not changed since its creation. This means there is no way to automatically display the score of each pool game to the viewers. Viewers of the tournament have no way to know the score of any game besides the game they are currently watching. Each table also must re-rack their own balls, slowing down the speed of the game. By taking advantage of current technology, the game of pool can be more interactive with its audience while speeding up the pace of the game.

1.2: Objective (RM)

The objective of this project is to design and prototype a modified billiards table that re-racks pool balls during tournament play and allows pool fans to receive live updates of each game from an app. To accomplish automated re-racking, a mounted camera will send live video to a microcontroller that will use image processing to determine when balls are sunk, and make a decision on whether or not they should be re-racked. If balls are determined legally out of play, they will be sent to a queue to be racked for the next game. By knowing the order of the balls in the queue, they can be re-racked properly. The rack will be placed on a motorized bed similar to an X-Y cartesian 3D printer, so it can accurately reposition itself under the hopper for each ball that gets re-racked. The images sent to the microcontroller will also determine the game's status in real time. As the game progresses, the positions of the balls will be sent from the microcontroller to a server, so pool followers can use an app to view an animated pool table that updates as the game progresses.

1.3: Background

(RM) One objective of this project is to use image processing to keep track of the balls that have been sunk and record the position of the balls currently on the table during each move. To accomplish this, a camera must be placed above the table and a live video feed will be sent to an embedded system that implements OpenCV, an image recognition API, to understand the placement of all balls. According to OpenCV's API, the service must run on either Linux, MacOS, or Windows. There's an example of this being completed with an ARM9 Processor using the OpenCV C++ API in a facial recognition environment [1]. Using the ARM9 processor provides the benefit of high frequency rates and a five-stage pipeline, allowing for fast and reliable image processing. One limitation may be the frame rate of the video feed. The document

did not include information about the number of video frames sent to the embedded system each second, so the time to receive and process each frame may exceed the time that it takes to stream each frame. Also, this implementation did not discuss sending the data to a database. As a result, the embedded system must have a Wi-fi module to stream data to the database.

(DM) Another implementation of image recognition and video analysis in a billiards environment is seen in Billiards Wizard [2]. This proof of concept uses image recognition and video analysis to teach people how to play pool based on videos of professionals playing the game. Similar to Billiards Wizard, the proposed billiards table outlined in this document will also use a camera to determine the position of the balls based on color and coordinates. In contrast, Billiards Wizard is limited in how it uses the video feed sent to it. There is no way to broadcast or analyze live gameplay, which limits its functionality.

(GR) Currently there is no way to automatically count score for those playing billiards. All score is kept track manually by the participants. This slows down the game because it takes the participants focus away from the game. Several projects have been attempted to alleviate this such as the project by Tang and Wang [3]. These different methods include: RFID readers, manually toggling LED scoreboards, and magnetic tracks which automatically track score. However, each of these methods has its own flaws. RFID readers are accurate, but the big issue is how the RFID chips are implemented in the balls. In the document by Wang and Tang, the billiards balls are simulated with Ping-Pong balls. This does not accurately represent the implementation because the RFID tags used would not fit in the balls. Any other form of tag or chip would require modifying the balls in a way that could affect how they affect play. Manually toggling an LED scoreboard still requires manual operation, defeating the purpose of automatic score tracking. Designing magnetic tracks to keep score would require many hours of design and

deliberation, assuming the design even functions in the end. Using a camera to track score eliminates the need to modify the balls or create a mechanical system that tracks score. This is the most efficient way to track score while adding little distraction to the participants.

(GR/DM) A third objective of this project is to design an automatic ball re-racking system. This solves the issue of manually emptying the pockets and re-racking the balls. The closest thing to an automated re-racking system is the commonly seen coin-operated tables located in bars and pool halls. This mechanism locks the sunk balls in a hopper underneath the table. If the cue ball is sunk, a magnet inside the ball allows the it to bypass the normal collection method. Another example of automatically re-racking billiards balls was introduced in 1917 [4]. The concept suggested by Russell is like the modern coin-operated tables. In this concept, tubes are ran from each pocket to a hopper at the end of the table, where sunken balls collect. The limitation of this technology is that there is no method to place the balls into the racking device. The balls would just be dropped into a bucket or some other collection device. This patent is similar to what is being proposed, however the goal is to re-rack the balls before the next game. This adds one more step to the process. Due to this, a solution will be designed that takes the collected balls and accurately places them into a billiards rack underneath the table. The most difficult part of this is to figure out how to get the balls in the proper spot in the rack, as there are requirements for how the balls are to be racked depending on the game being played. The solution to this would be a cartesian motion system for the bed that the rack will be sitting on. 3-D printers currently use this system to move designs on the two-dimensional plane, while the extruder moves vertically. In the publication “Dynamic Modeling and Characterization of the Core-XY Cartesian Motion System” it can be seen exactly how this system works and why it is one of the most accurate systems to use for positioning [5]. While this project is not to create a 3-

D printer, the cartesian system is relevant to the design of the re-racking system because it will allow a billiards rack to move under a hopper in a way that each dropped ball will land in the desired position for the next game.

(KS) A fourth objective of this project is to use a motorized gate to release billiard balls from the pocket to the track. Motorized gate systems are able to be used in many different types of scenarios. Automatic doors and gate systems are used in everyday life from handicap doors to garage doors. In a project by Hao Wang in 2015, Wang designed a scale model of a door system used in the Spartan Superway automated transit network [6]. The way this gate system will work in this design project is that the system will stay normally closed and when billiard balls are pocketed, a motor will open the door and release the balls below into the track, then into the automatic re-racking system. In the case that the cue ball is pocketed, the door system will remain closed so the cue ball may be removed by a player. There will be many similarities and differences between the design by Wang and the design implemented into this proposed design project. A design similarity to Wang's is that "Basic dimensions of door opening need to meet requirements for easy accessing for passengers with disability, emergency egress, and rescue access." [6]. This similarity comes into play such that the door must be located deep enough into the pocket in order to allow at least three balls to fall into any pocket at any point in time. The case where three balls would be in a pocket at one certain time would come when a player makes a combination shot and pockets the cue ball. In Wang's design, there are many requirements that would be unnecessary in the design of a modified billiards table. In the design of Wang's door system, there are standards for gap space between doors, platforms, and door frames in order to prevent crushing hazards. In the design of a door system for a billiards table, the only precautionary gap would come between the gate and the track. A factor that will need to be

calculated in this design project is the force being exerted onto the door in the system. The design of an automatic door system by Wang accounts for an external pressure force being applied between the roller and base plate. In this design project, there will be a pressure force exerted directly onto the door due to the balls that were pocketed. The gate will be controlled by a spring that will allow the billiard balls to fall into the racking system and then return back to level after the balls have been released.

(KS) Another objective of this project is monitoring gameplay, a useful tool for coaches or anyone viewing the game at play. There is currently a patent from James W. and James V Bacus for an LED light fixture that is mounted above the pool table that may support one or more cameras [7]. The main concept behind monitoring the game is that the recordings may be reviewed in order to achieve a higher level of play. This device records game play, offers replay, review, analysis of stored video, and light dimming controls. This device would be very similar to the recording device used in this design project. A difference is that the device used in the design project will not be required to be mounted from the ceiling. The device created for the design project will be freestanding around the table. Having lighting and the monitoring device mounted above the table restricts the table from use in an open room. With the freestanding device, the table can essentially be located anywhere as long as there is an electrical source readily available and there is enough clearance for the freestanding aperture. The device created by James W. and James V Bacus uses a grid system that aids in locating and determining the position of each ball. This grid system will also be used by the freestanding device as well so the balls can be easily located and tracked.

(RM) A final project objective is viewing live game data from an app. For this project, React Native has been selected as the programming language of choice, because it can be written

in one language, and function on both Android and iOS operating systems instead of writing two separate apps. For real time applications, Firebase is an appropriate database to implement in this project due to its real-time capabilities. Game data will be streamed to the Firebase Cloud after the software interprets the game's progress on the embedded system. The mobile app will listen for changes on the Firebase database and update the mobile app when it's found. An example of an embedded system streaming data to Firebase for a mobile app to interpret can be found in the publishing by Li, Yen, Lin, Tung, and Huang [8]. In both implementations, embedded systems interpret data and send it to Firebase storage. Then, a mobile app analyzes the data in real-time. This article discusses the different methods of streaming data to Firebase from the microprocessor. The authors of this publication utilized a REST API for sending data, and a JavaScript SDK to read data. In the modified billiards table, these concepts will be the same. Data will stream from the embedded system via REST API and read via the JavaScript SDK. In this implementation, data is sent from an Arduino (a microprocessor without internet capabilities) to an "intelligence server" before sending data to Firebase Storage. For this implementation, server integration could be eliminated by adding internet capabilities to the embedded system so data can be streamed straight to Firebase.

1.4: Marketing Requirement (DM)

This product should automatically rack the balls, give fans live updates of tournament play, identify which balls are on the table, individually identify which ball has been pocketed, and determine the score and winner of each game.

2. Design Requirements Specifications (ALL)

Table 2.1.1: Engineering Requirements

Marketing Requirement	Engineering Requirements	Justification

2, 3, 4	The system will be able to differentiate each of the balls on the table from one another other.	The algorithm will be able to successfully detect each of the pool balls on the table, and determine which ones have been pocketed by each player.
6	The system will be able to remotely display the position of each ball on the table.	The algorithm will create an (X,Y) coordinate for each of the balls on the table, and send a JSON
1,3,4,5	The system will be able to determine which balls are in the pockets or no longer on the table.	The program will determine which balls are in the table so that re-racking commands can be sent to the microcontroller below.
4,5	The gate system will be able to be delayed a maximum of 10 seconds until the ball in the pocket is identified.	The microcontroller will send a signal to the servo motor telling the motor to open or close depending on which ball was pocketed.
1,4,5	The racking system will be able to place the 8-ball and one solid and one stripe in their desired positions within the rack.	The racking system will be told by the microcontroller which ball was pocketed and determine which position in the rack to place the ball in.
2,3,4,5,6,7	The application will be able to offer a live look into the game being played.	The server will send live updates out to a mobile application that will track ball movement and location, score and winner of each game.
1,4,5	The racking system will be able to place a ball into its desired spot within 30 seconds of the ball being placed in the start position.	The stepper motors used to drive the system will move the position arm quickly into place upon receiving a ball.
2,3	The live game play will not be affected by the system in any way.	High speed camera will be mounted above table to provide a clear view of table with constant light applied.

6,7	The system will be able to determine which player's turn it is therefore, determining which balls should be being aimed for.	The camera will determine which player is up based on ball placement and by the amount of time delay,7 between shots.
1	The re-racking system will be complete within 2 minutes of game completion.	The racking system is used to help speed up the process and decrease the time in between games.
Marketing Requirements 1. The system will automatically re-rack the pool balls 2. The system will provide live updates 3. The system will identify balls on the table 4. The system will identify balls in the pockets 5. The system will track the status of the balls 6. The system will determine the score of each game 7. The system will determine the winner of each game		

3. Accepted Technical Design

3.1. Hardware Design (GR/KS):

Figure 3.1.1: Level 0 Hardware Block Diagram

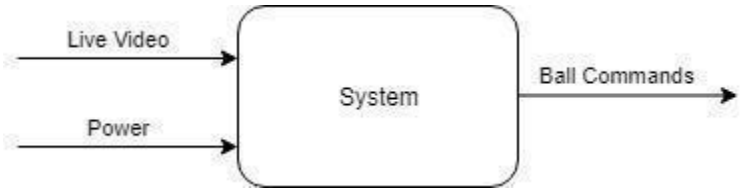


Table 3.1.1: System Fundamental Requirement Table

Module	System
Inputs	Power
	Live video from the table
Outputs	Ball command signals
Description	The system will analyze live images from the table to determine what ball commands to be executed.

Figure 3.1.2: Level 1 Hardware Block Diagram

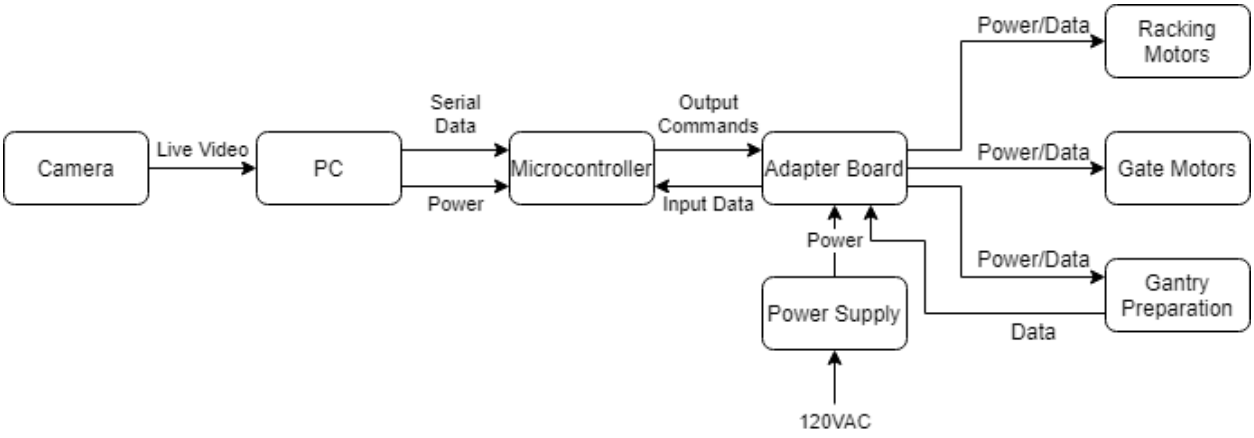


Table 3.1.2: Camera Fundamental Requirement Table

Module	Camera
Inputs	Power over USB
	Visual input from table
Outputs	Live video to the PC
Description	The camera will send live video to the PC for processing.

Table 3.1.3: PC Fundamental Requirement Table

Module	PC
Inputs	Live video from camera
Outputs	Serial commands to microcontroller
	Power over USB to the microcontroller
Description	The PC will use image processing to determine ball positions on the table and inside the pockets. The PC will then send this data over serial to the microcontroller. The PC will also power the microcontroller over USB.

Table 3.1.4: Microcontroller Fundamental Requirement Table

Module	Microcontroller
Inputs	Power over USB from the PC

	Serial commands from the PC
	Input data from the Adapter Board
Outputs	Output commands to the Adapter Board
Description	The microprocessor will receive serial commands from the PC telling the microprocessor which balls are in the pockets.
	The microprocessor will receive serial commands from the PC telling the microprocessor which balls need re-racked under the table.
	The microprocessor will then send output commands to the Adapter Board.

Table 3.1.5: Adapter Board Fundamental Requirement Table

Module	Adapter Board
Inputs	Power from the Power Supply
	Output commands from the Microcontroller
	Data from the Gantry Preparation
Outputs	Output any received input data from the stages
	Output power and data to the Racking Motors
	Output power and data to the Gate Motors
	Output power and data to the Gantry Preparation
Description	The Adapter Board serves as an interface between the Microcontroller and the sensors and actuators in the system.
	The Adapter Board receives data and power while also outputting data and power to various stages under the table.

Table 3.1.6: Racking Motors Fundamental Requirement Table

Module	Racking Motors
Inputs	Power and data from the Adapter Board
Outputs	Mechanical movement
Description	The racking motors will be used to move a gantry to re-rack pocketed balls into a triangle.
	This system will be located under the table to allow the pockets to feed into the racking solution.

Table 3.1.7: Gate Motors Fundamental Requirement Table

Module	Gate Motors
Inputs	Power and data from the Adapter Board
Outputs	Mechanical movement
Description	The gate motors will be used to hold the sunk balls in each pocket to check and see if the cue ball was pocketed.
	The gate motors will then release the sunk balls into the re-racking solution under the table.

Table 3.1.8: Gantry Preparation Fundamental Requirement Table

Module	Gantry Preparation
Inputs	Power and data from the Adapter Board
Outputs	Mechanical movement
	Input data from sensors in the track
Description	The gantry preparation tube will be used to index and hold the balls in the track before the gantry system.
	This system will allow for one ball at a time to be released and indexed into the gantry to be re-racked.

Table 3.1.9: Power Supply Fundamental Requirement Table

Module	Power Supply
Inputs	Power: 120VAC
Outputs	Power: 12VDC, 6VDC, 5VDC
Description	The power supply will provide the necessary voltages and currents to each component of the system.

Figure 3.1.3: Level 2 Hardware Block Diagram

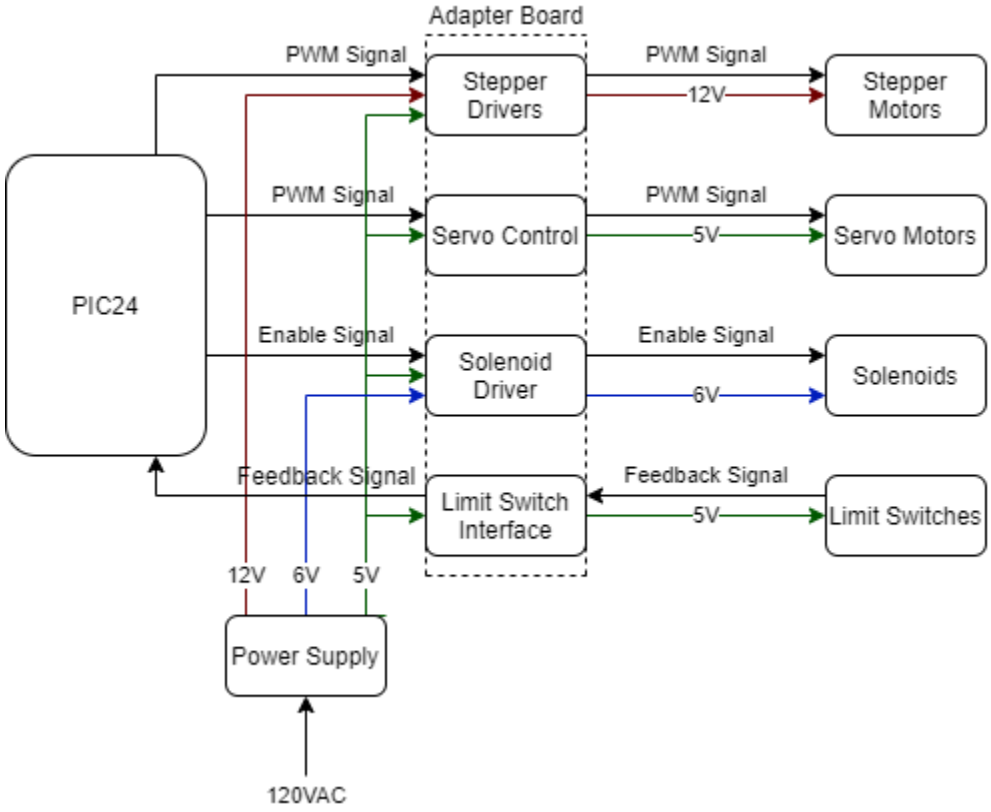


Table 3.1.10: PIC24 Fundamental Requirement Table

Module	PIC24
Inputs	Feedback signal from Limit Switch Interface
Outputs	PWM signal to Stepper Drivers
	PWM signal to Servo Control
	Enable signal to Solenoid Drivers
Description	The PIC24 will output various signals to different driver/controller circuits on the adapter board to activate the corresponding devices.
	The PIC24 will receive a feedback signal from the Limit Switch Interface when the limit switches have been triggered.

Table 3.1.11: Power Supply Fundamental Requirement Table

Module	Power Supply

Inputs	120VAC
Outputs	12V to Stepper Drivers
	6V to Solenoid Drivers
	5V to Stepper Drivers, Servo Control, Solenoid Driver, Limit Switch Interface
Description	The Power Supply

Table 3.1.12: Stepper Drivers Fundamental Requirement Table

Module	Stepper Drivers
Inputs	12V from Power Supply
	5V from Power Supply
	PWM Signal from PIC24
Outputs	12V to Stepper Motors
	PWM Signal to Stepper Motors
Description	The Stepper Drivers are used to provide the necessary power and signal to the Stepper Motors.
	The PWM Signal will be converted into a modified PWM to move the Stepper Motors.
	The Drivers will pass 12V to power the Stepper Motors. The 5V is used to select and enable options built into the Driver chips.

Table 3.1.13: Servo Control Fundamental Requirement Table

Module	Servo Control
Inputs	5V from the Power Supply
	PWM signal from the PIC24
Outputs	5V to the Servo Motors
	PWM signal to the Servo Motors
Description	The Servo Control will be used to open and close the Servo Motors in the pocket gates. Depending on the PWM signal received from the PIC24, the gates will open or close. The 5V is used to power the Servo Motors.

Table 3.1.14: Solenoid Driver Fundamental Requirement Table

Module	Solenoid Driver
Inputs	6V from the Power Supply
	5V from the Power Supply
	Enable signal from the PIC24
Outputs	6V to Solenoids
	Enable signal to Solenoids
Description	The Solenoid Driver is used to engage and disengage the solenoids via a signal from the PIC24. However, this Driver is needed because the PIC24 cannot power or control the solenoids on its own.

Table 3.1.15: Limit Switch Interface Fundamental Requirement Table

Module	Limit Switch Interface
Inputs	5V from Power Supply
	Feedback Signal from Limit Switches
Outputs	5V to Limit Switches
	Feedback Signal to PIC24
Description	The Limit Switch Interface is used to power the attached Limit Switches. The Interface will also limit the feedback signal to protect the PIC24 from damage.

Table 3.1.16: Stepper Motors Fundamental Requirement Table

Module	Stepper Motors
Inputs	12V from Stepper Drivers
	PWM signal from Stepper Drivers
Outputs	Mechanical movement
Description	The Stepper Motors are used to move the rerack gantry. The motors will move the x and y axis of the gantry. The 12V is used to power the motors and the signal determines the speed of the motors.

Table 3.1.17: Servo Motors Fundamental Requirement Table

Module	Servo Motors
Inputs	5V from Servo Control
	PWM signal from Servo Control
Outputs	Mechanical movement
Description	The Servo Motors are used to control the gates in the pockets. These gates hold balls temporarily before releasing them into the track below. The 5V is used to power the motors and the PWM determines the motors position.

Table 3.1.18: Solenoids Fundamental Requirement Table

Module	Solenoids
Inputs	6V from Solenoid Driver
	Enable signal from Solenoid Driver
Outputs	Mechanical movement
Description	The Solenoids are used to stop and index balls in the gantry preparation track. These solenoids allow one ball at a time to be released and control the flow of balls. The 6V is used to power the solenoids while the enable signal engages or disengages the solenoids.

Table 3.1.19: Limit Switches Fundamental Requirement Table

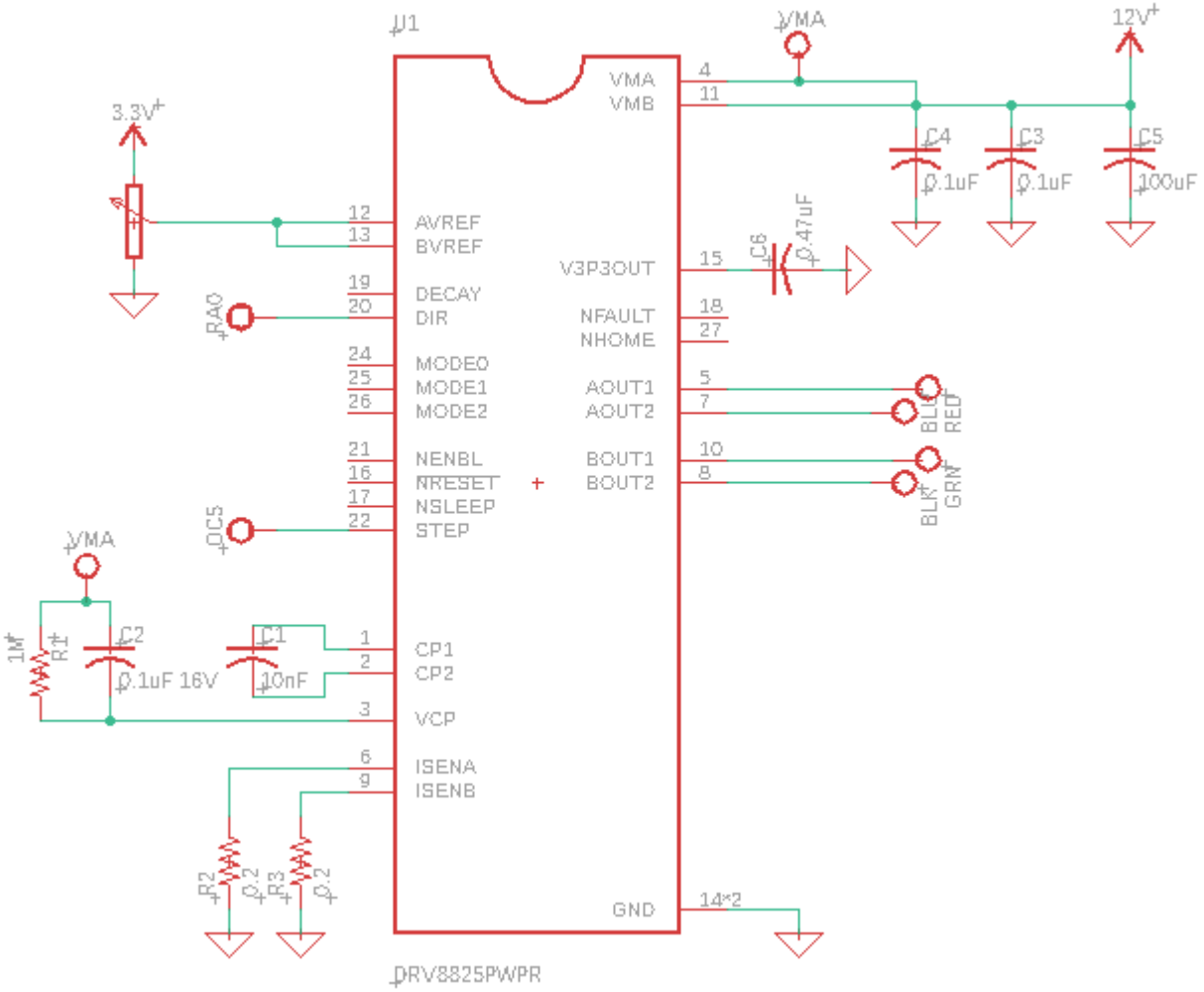
Module	Limit Switches
Inputs	5V from Limit Switch Interface
Outputs	Mechanical movement
	Feedback Signal
Description	The Limit Switches are used to determine the position of a ball in the gantry preparation track and the position of the re-rack gantry. The limit switches are powered by 5V. When the limit switch is activated, a feedback signal is sent to the Limit Switch Interface to tell the PIC24 the limit switch has been tripped.

Table 3.1.20: Stepper Motor Driver Parts Table

Reference Designator	Part Description
U1	DRV8825 Stepper Motor Chip

C1	10nF Capacitor
C2	0.1uF 16V Capacitor
C3, C4	0.1uF Capacitor
C5	100uF Capacitor
C6	0.47uF Capacitor
R1	1M ohm resistor
R2, R3	0.2 ohm resistors
UNLABELED	10k ohm potentiometer

Figure 3.1.4: Stepper Motor Driver Schematic



Shown in Figure 3.1.4 is the schematic created to build the stepper motor drivers. These stepper motor drivers are necessary to move the attached stepper motors, as stepper motors are not operated the same way as traditional brushless motors. The Texas Instruments DRV8825 was selected to be used as the driver chip. The DRV8825 can output 2.5A per phase, more than enough to power the 2A per phase stepper motors selected. The DRV8825 is capable of 1/32 micro stepping, which can be used to reduce the 1.8 degree step into a smaller, more precise movement. The schematic shown above was designed using the typical application selection on page 18 of the DRV8825 datasheet [10]. VMA and VMB are the Bridge A and Bridge B power supplies [10]. These are connected to the motor supply voltage of 12 Volts and bypassed to GND with a 0.1uF capacitor. The 100uF capacitor also tied to VMA/VMB is used to safely block any parasitic from the attached power supply. VMA/VMB are also connected to VCP via a 1M ohm resistor and 0.1uF capacitor. VCP is the high-side gate drive voltage [10]. DIR is connected to pin RA0 of the PIC24 microcontroller. This pin changes the direction of the stepper motor depending on if the pin is set high or low. STEP is connected to OC5 of the PIC24 microcontroller. This pin is used to move the indexer one step every time a rising edge signal occurs. CP1 and CP2 are listed as charge pump flying capacitors [10]. These pins are tied together using a 0.01uF capacitor. ISENA and ISENB are Bridge A and Bridge B current sensing resistors [10]. These resistors are chosen such that the denominator in Figure 5.1.5 is set equal to 1. This means that the values for the current sensing resistors needs to be 0.2 ohms.

Figure 3.1.5: Chopping Current Equation [10]

$$I_{CHOP} = \frac{V_{(xREF)}}{5 \times R_{ISENSE}}$$

The other variable used in Figure 3.1.5 is AVREF and BVREF. These pins are used to obtain a reference voltage for Bridge A and Bridge B. They are connected to a 10k ohm

potentiometer connected to 3.3V. The center tap is then connected to both AVREF and BVREF. This reference voltage is then used as the numerator for Figure 3.1.5. The value of the voltage determines the chopping current because the denominator is set to 1. This chopping current is the limit on the current for each bridge. By varying the chopping current, more or less current can be used to drive the stepper motors. For the stepper motors selected, the VREF values are set to 2V so that 2A of chopping current is set as the limit. V3P3OUT is the internal 3.3V regulator on the chip. However, it will not be used so it is bypassed to GND with a 0.47uF capacitor. AOUT1, AOUT2, BOUT1, and BOUT2 are connected to the four wires of the stepper motor. The color connection is shown in the schematic. Each set of outputs, AOUT/BOUT, are connected to a single phase of the stepper motor. The remaining unconnected pins are used to select different features such as the micro stepping, sleep, home, or reset. These pins all contain internal pulldown resistors so if no voltage is applied, they are logic level 0's.

Table 3.1.21: PIC Adapter Board Parts Table

Reference Designator	Part Description
5V, 6V, 12V	Screw-Terminal Connector
C1, C2, C3, C4	100uF 50V Capacitor
22-23-2021	Molex 2-Pin Connector
22-23-2031	Molex 3-Pin Connector
22-23-2041	Molex 4-Pin Connector
22-23-2051	Molex 5-Pin Connector
22-23-2061	Molex 6-Pin Connector
JP1, JP2	2 Row 6-Pin Header
UNLABELED	8 Position Connector Receptacle
U1, U2	IC Gate AND 4 Channel 2 Input

S1, S2, S3, S4	MOSFET N-CH 60V 200MA TO-92
D1, D2, D3, D4	Diode General Purpose 1000V 1A
Q1, Q2, Q3, Q4	MOSFET, 50V, 30A, TO-220 pkg
R1, R2, R3, R4, R5, R6, R13, R14, R15, R16, R17, R18	10kΩ Resistor
R7, R8, R9, R10, R11, R12	20kΩ Resistor
F1, F2, F3	FUSE BLOCK BLADE 500V 30A PCB
F1, F2, F3	FUSE AUTOMOTIVE 7.5A 32VDC BLADE

Figure 3.1.6: PIC Adapter Board Schematic

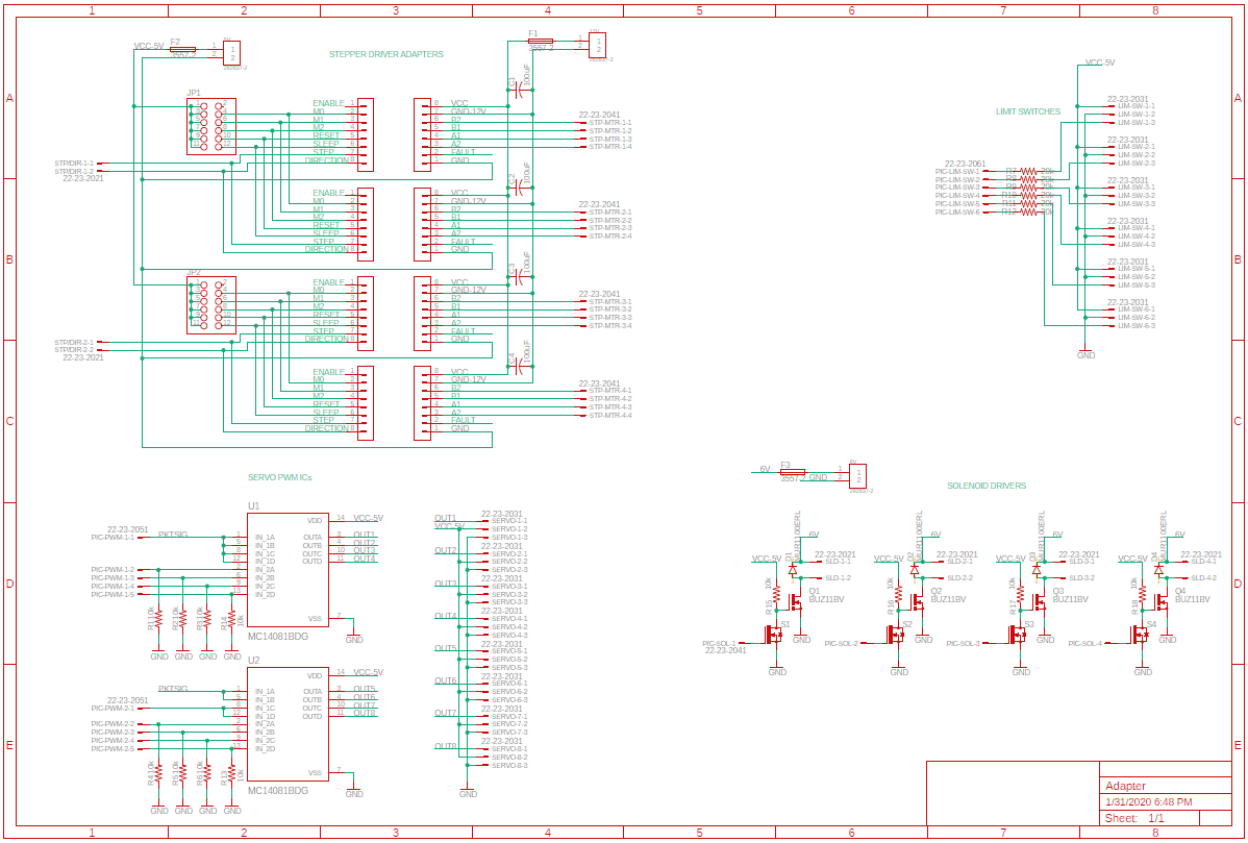


Figure 3.1.7: Adapter Board PCB

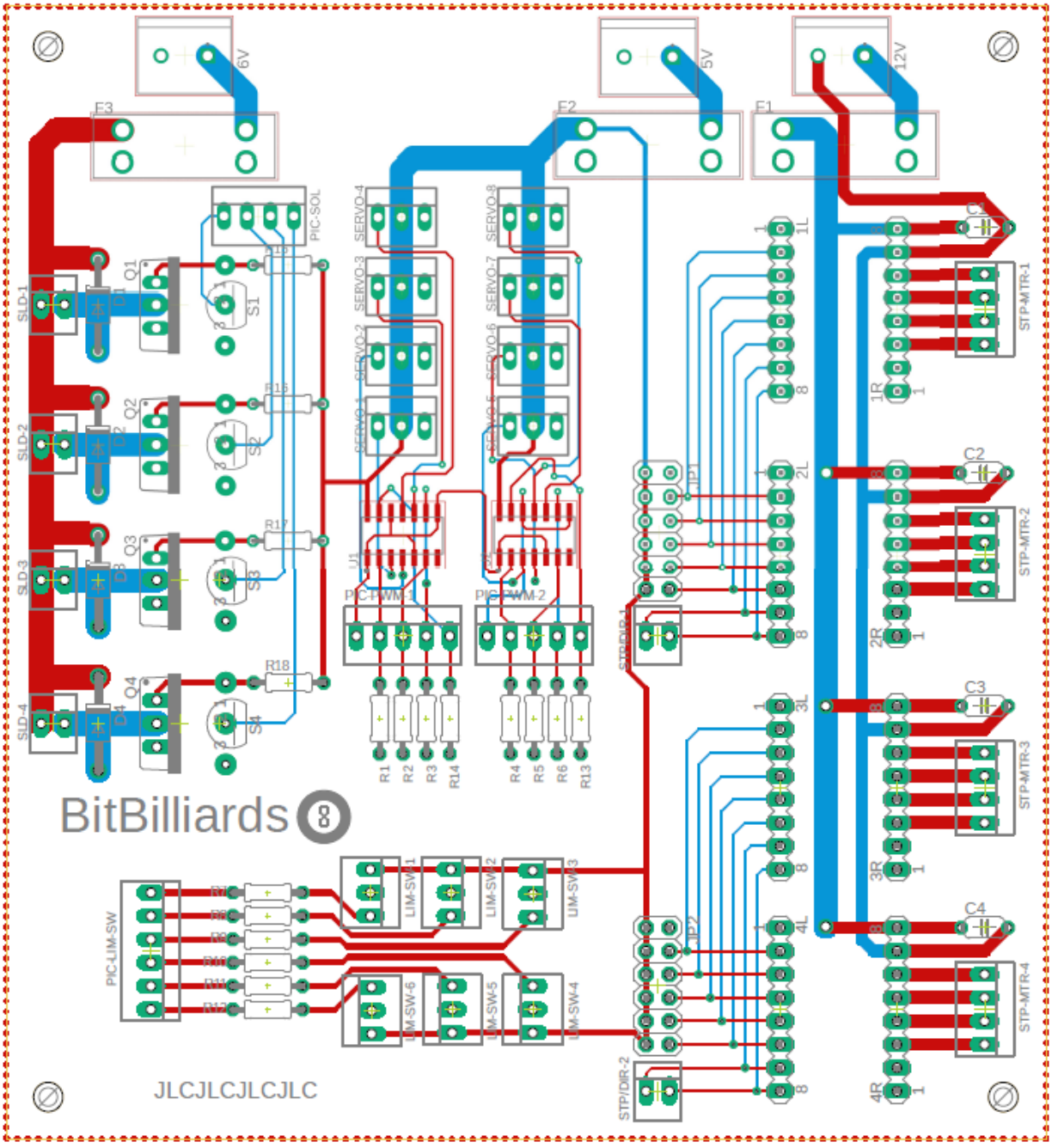
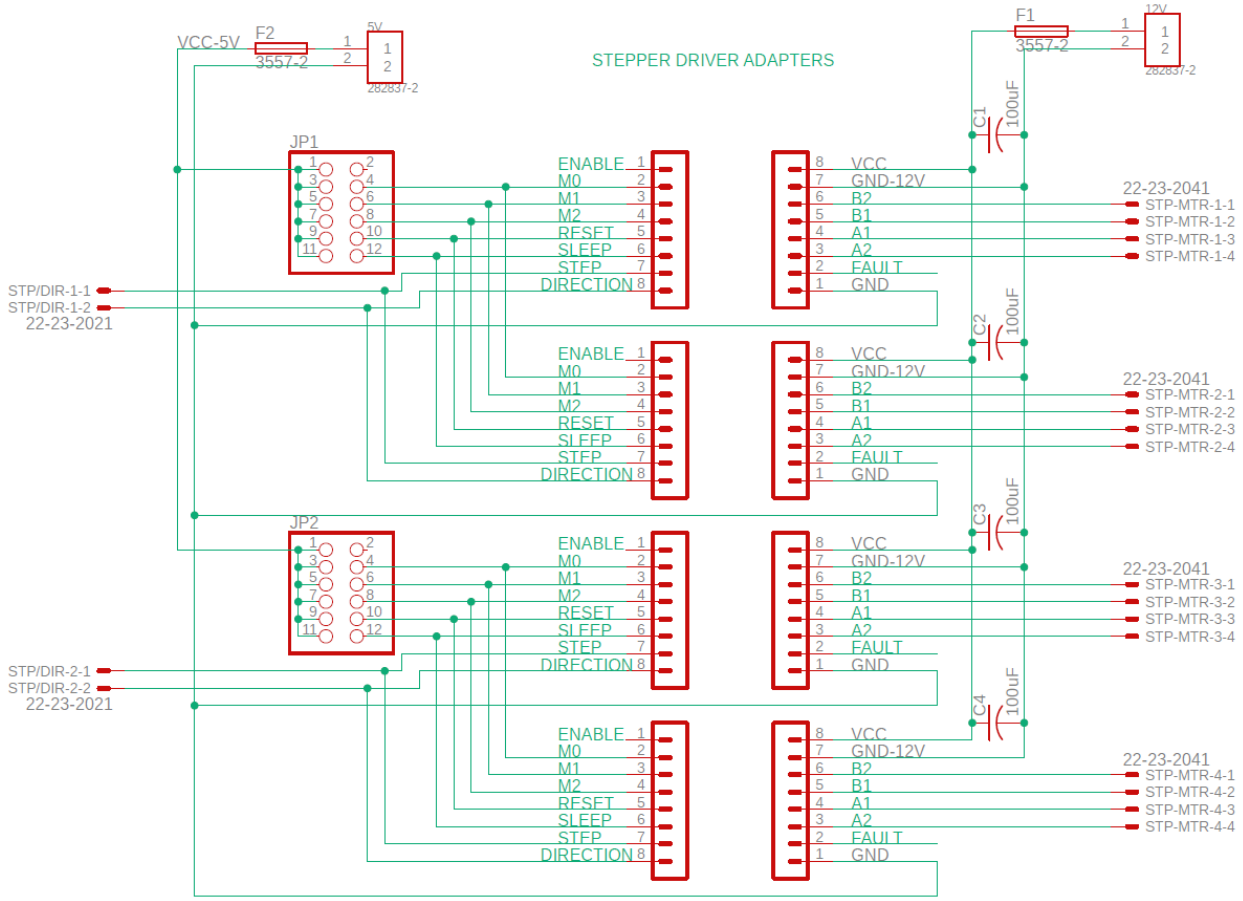


Figure 3.1.6 shows the designed PIC Adapter Board for the project. The goal of this PCB was to create a PCB that housed all the necessary components to be an interface between the PIC24 controller and the billiards table. The PCB layout is shown in Figure 3.1.7. The stepper drivers, solenoid drivers, PWM servo control, and limit switch interface were all designed and

placed on the Adapter Board. This allows for the signals of the PIC24 to be sent to the adapter, and then for the adapter to send the signals to the billiards table to trigger the corresponding components. The schematic and PCB layout are broken into four sections: Stepper Driver, Servo Control, Solenoid Driver, and Limit Switch interface. 5V, 6V, 12V, and GND are all provided to the Adapter Board. Each different voltage was fused separately. The Adapter Board is a 2-layer PCB.

Figure 3.1.8: Adapter Board Stepper Motor Schematic



The Stepper Motor section of the Adapter Board shown in Figure 3.1.8 was designed to allow plug and run functionality while also allowing options to be selected. The four stepper driver boards plug into two sets each of eight pin receptacles. The pins are all labeled with their corresponding functions as seen in Figure 3.1.8. These functions include providing 12V,

connecting to the motor outputs, and selecting the desired options on the stepper drivers. The 12V and 5V rails are both fused to protect all components. A 100uF capacitor is then connected across the 12V rail to protect each stepper driver from any irregularities from the power supply. The four motor outputs from the steppers are connected each to a Molex 4-Pin header. Each set of two steppers have their additional options and step/direction signal wired together. This is because each set of two stepper drivers corresponds to either the X or Y axis of the gantry re-racking system. One motor was used to drive each side of each axis, allowing the load to be distributed evenly across the motors. By being wired together, each axis of motors is always receiving the same options and the same step/direction signal. This prevents the two motors on the same axis from becoming out of sync. The additional option pins are routed to a 2 Row 8-Pin header (shown as JP1 & JP2) which allows for a jumper to be placed across the pins to set the desired pin high. 5V is connected to the other side of the jumper headers JP1 and JP2. Some of the additional selectable options include micro stepping mode, sleep mode, enabling of the steppers, and reset. The SLEEP and RESET jumpers were connected to enable the steppers to operate. The step/direction pins are routed to a Molex 2-Pin header, which directly connects to the output from the PIC24 controller.

Figure 3.1.9: Adapter Board Servo Schematic

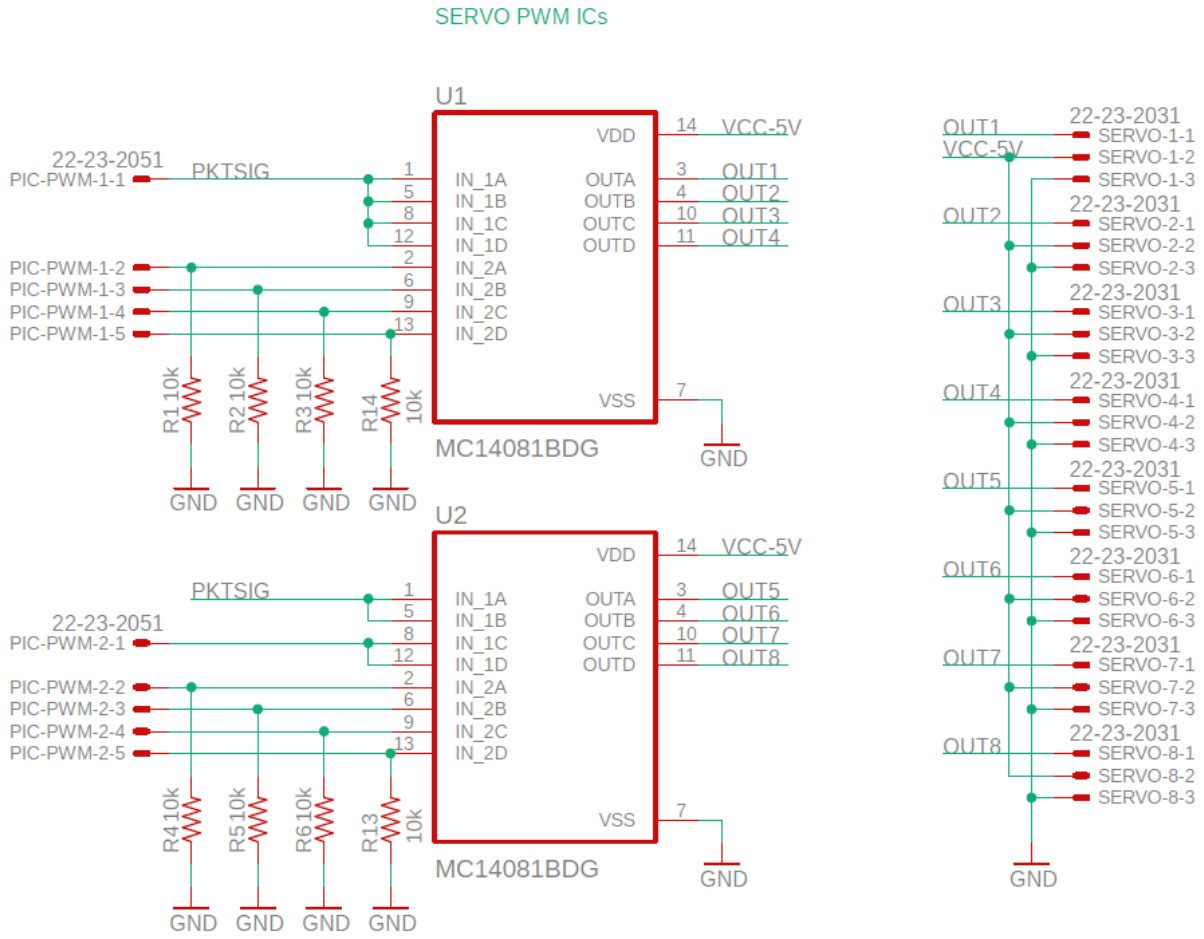


Figure 3.1.9 shows the schematic design of the servo control on the adapter board. The servo control had to be planned out because the servos operate on PWM signals. The PIC24 has a limited number of PWM outputs, and with the stepper drivers also using PWM the remaining number of PWM outputs were limited. To get around this issue, two AND IC chips, labeled U1 and U2 in Figure 3.1.9, are utilized. The AND ICs used added very little delay to the response time of the signals. An AND gate outputs a HIGH signal when both inputs are toggled HIGH. So, for this situation, one input on each gate was tied to a PWM output from the PIC24. The other inputs were then tied to a normal output pin on the PIC24. This allowed for one PWM to drive up to six servos. These six servos would each be used in the pockets of the tables to act as

gates. If servo one was required to be opened, the second AND input, 2A in this situation, was set HIGH. By setting 1A to the PWM, and 2A to a constant HIGH signal, the resulting output would be the desired PWM signal. The six pocket gate servos were connected to the 3-Pin Molex Headers labeled Servo 1 to Servo 6. 3-Pin Molex Headers were used to connect all eight servos to the adapter board. The remaining two available AND gate inputs were connected to a second PWM output from the PIC24. This allows for these two servos to be controlled separately from the pocket gate servos. One of these servos, Servo 7, was used to release the balls into their final positions in the gantry re-rack system. The other servo output, Servo 8, was unused. A 10kΩ resistor was connected to each non-PWM input and then to ground. This prevented the enable pins from floating, as the resistors acted as pull-down resistors. 5V and GND were provided to all servos and the AND IC chips.

Figure 3.1.10: Adapter Board Solenoid Schematic

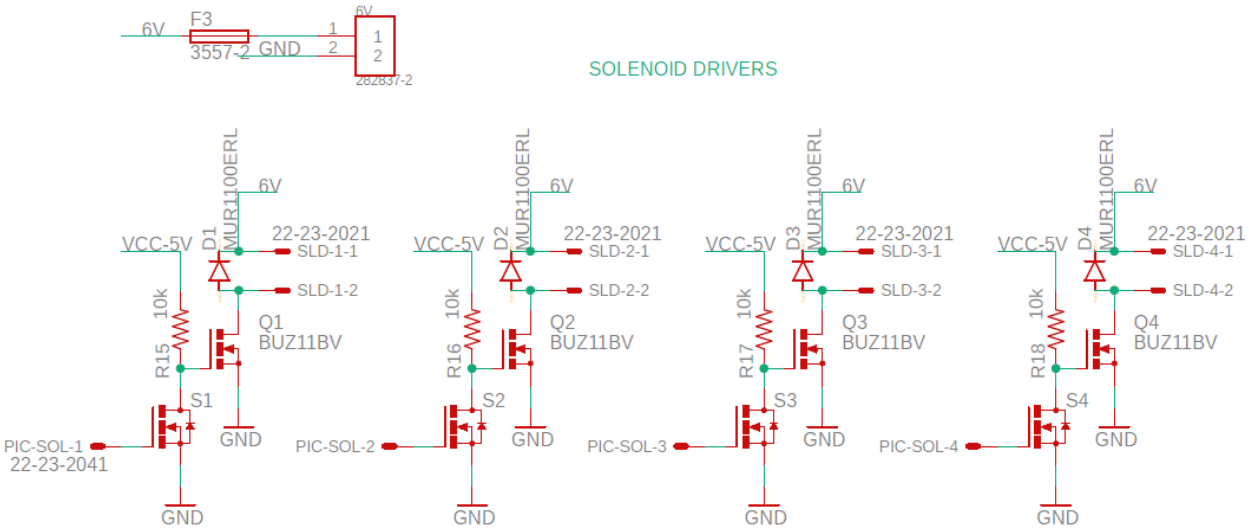


Figure 3.1.10 shows the circuit used to drive the solenoids in the project. Four solenoid driver circuits were constructed, with two being used for the gantry staging track and the other two free to be used as desired. The solenoids could be operated at 5V or 6V, however at 5V it

was observed that the solenoid would occasionally fail to fully retract, so 6V was used to operate the solenoids. A power MOSFET was required because at 6V the solenoid pulled 1.2 amps of current. The power MOSFET chosen, the BUZ11, could operate at 50V and pull 30A of current [11]. The BUZ11 MOSFETS are indicated in Figure 3.1.10 by Q1, Q2, Q3, and Q4. A major issue that occurred was that the PIC24 voltage output was too low to surpass the gate threshold voltage of the BUZ11 MOSFET. The PIC24 output pins are capable of outputting 3.3V. The gate threshold voltage of the BUZ11 is 4V [11]. To fix this issue, a N-Channel MOSFET was used to trigger the required voltage to control the BUZ11 MOSFET. This N-Channel MOSFET used was the 2N7000. The 2N7000 has a gate threshold voltage of 3V [12]. The 2N7000 is designated as S1, S2, S3 and S4 in Figure 3.1.10. 5V was connected to a 10kΩ resistor, which is connected to the gate of the BUZ11 and the drain of the 2N7000. When the gate of S1 was set HIGH by the PIC24, 5V was set to the gate of Q1. When this occurred, current was pulled through the attached solenoid. The solenoid had one lead connected to 6V and the other lead connected to the drain of Q1. A diode was wired across the solenoid to prevent and protect against any back EMF generated by the inductor in the solenoid. When tested, the circuit operated exactly as expected with no issues occurring. The 6V source was fused and connected to the correct power supply.

Figure 3.1.11: Adapter Board Limit Switch Schematic

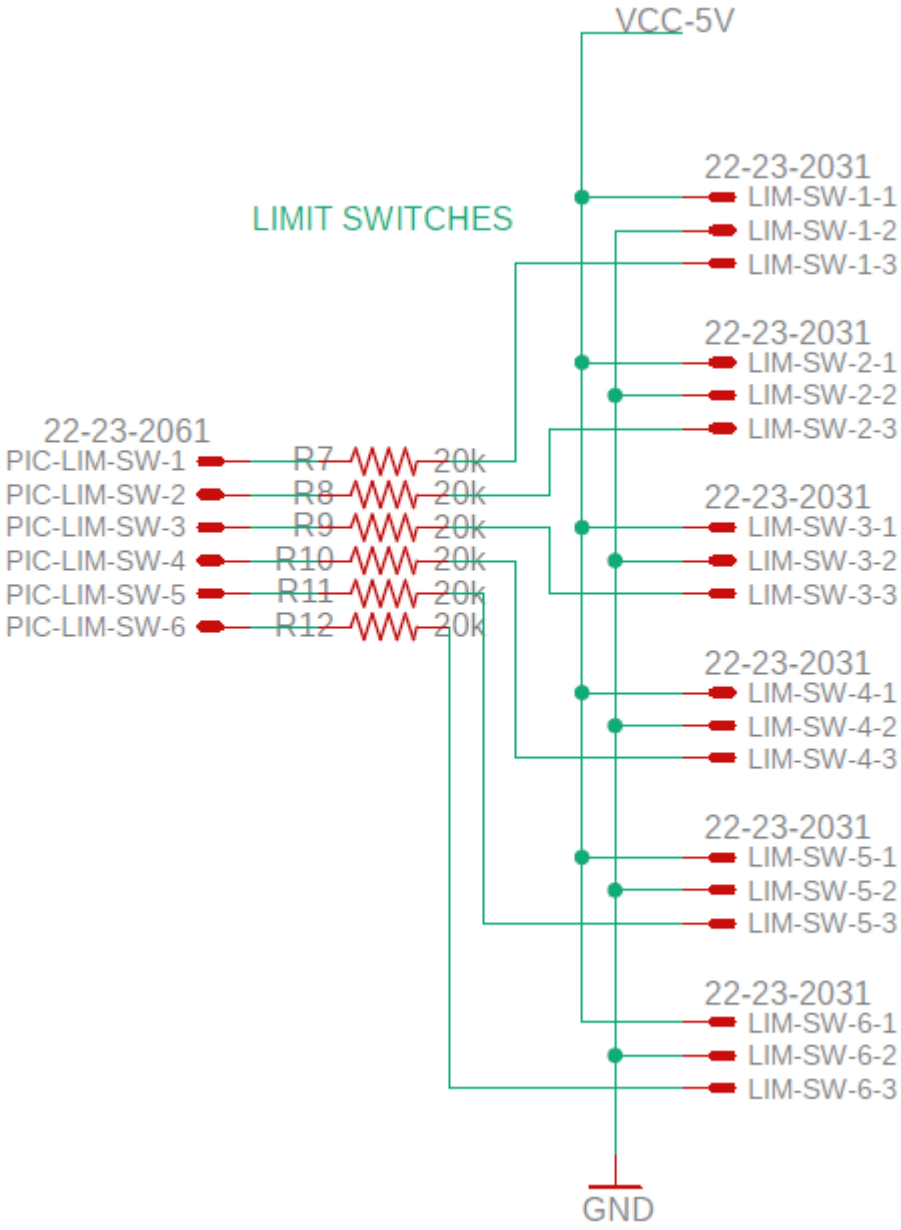


Figure 3.1.11 shows how the limit switches were connected to the adapter board. The limit switches were used in various instances such as: gantry system end of arm travel for both the X and Y axis, ball present in gantry, ball present in preparation tube, and ball index position present. The circuit itself is the simplest of the four sections on the adapter board. The limit switches each contained three pins, one for each of the following: power, signal, ground. A 3-Pin

Molex Header was used to connect each limit switch to the adapter board. The limit switches operated on +5V. +5V and GND were connected to the supplies on the adapter board. A 20kΩ resistor was connected in series with the signal output of the limit switch. This is due to how the limit switches operated. When connected to the normally open contact, depressing the limit switch actuator will complete the circuit. When the power is connected to +5V and the signal output to the PIC24, this results in connecting +5V directly to a PIC24 input. The 20kΩ resistor in series prevents the current from damaging the PIC. The current becomes 0.25mA, enough for the PIC24 to detect a signal but low enough to prevent damage. The six output signals were connected to a 6-Pin Molex Header which was then wired to the corresponding PIC24 pins.

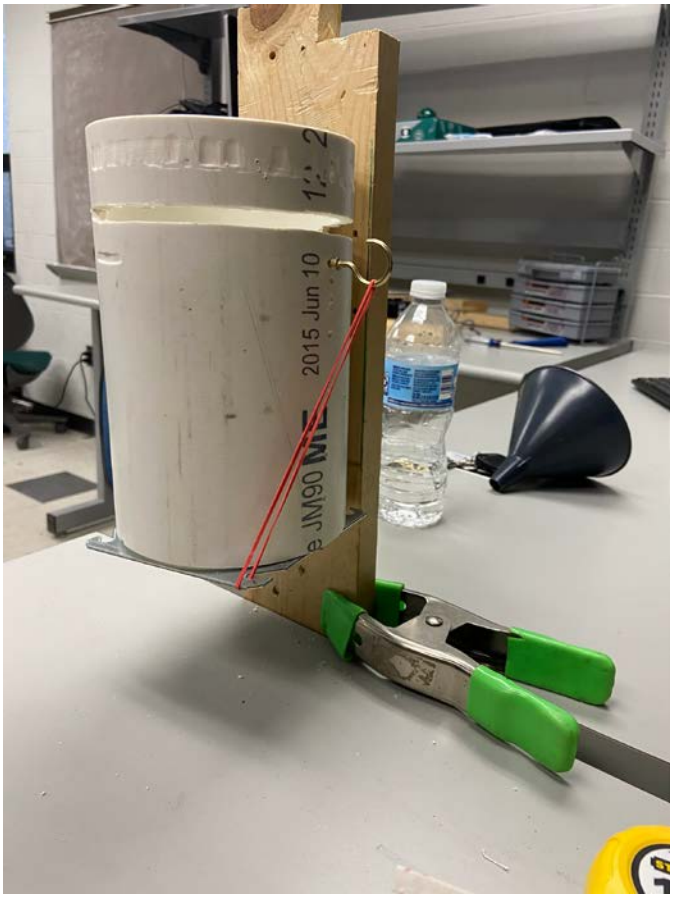
Figure 3.1.12: Pool Table Top View

Figure 3.1.13: Pool Table Underside View

As shown in Figure 3.1.12 and 3.1.13, the track system was made of 3-inch PVC pipe that was cut in half in order for the balls to fall from the pockets directly into the track system. The track was then hung underneath the table and angled so that the ball was able to roll continuously without gaining too much speed.

One main problem that we ran into when integrating the pocket gates and track system together was how the balls were falling out of the pockets when the servo motor was triggered. It was determined that the pocket gates needed a way to allow the balls to fall the same way each time into the track, or essentially aim the ball. To fix the problem, a funnel was attached inside of the pvc pocket such that only one ball would fall straight down at a time rather than have the balls off-centered in the pocket and release at the same time.

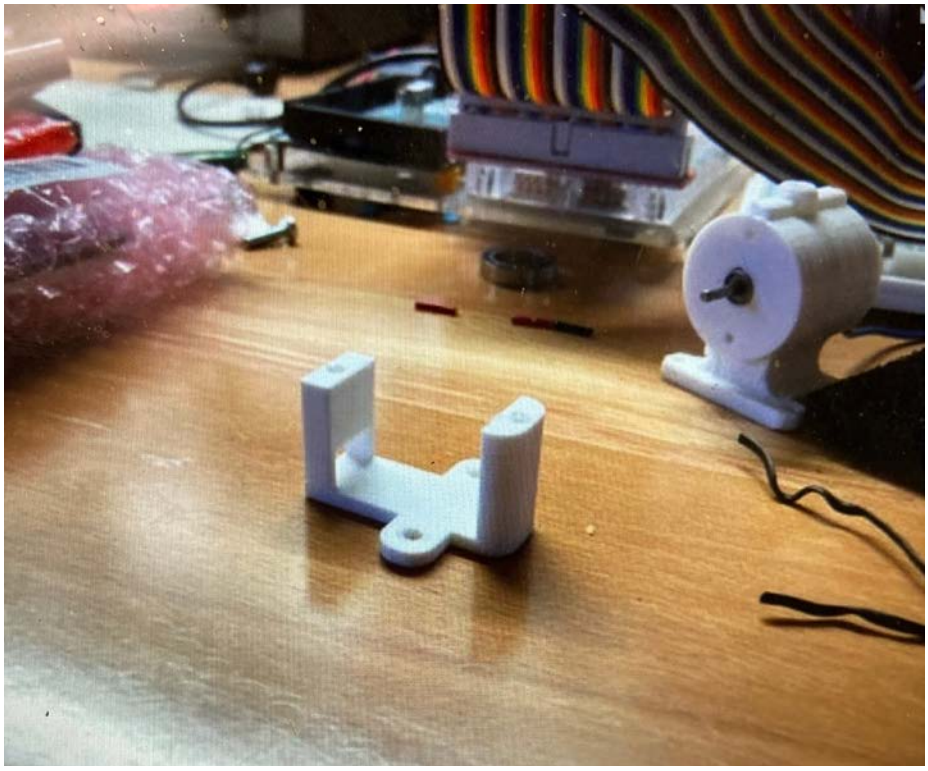
Figure 3.1.14: Pocket Gate Test Stand



The implementation of the pocket gates became rather tough. In the final design for the pocket gates, as seen in Figure 3.1.14, a steel plate was attached to a small piece of wood with a hinge that allowed the plate to move freely. Attached to the side of the pvc pocket was a cup hook which then, a rubber band was hooked onto the cup hook and the metal plate. This rubber band was implemented in order for the plate to return back to level so that the servo motor, which acted as a latch, would be able to lock the plate back into place.

Figure 3.1.15: Pocket Gate Implementation

Figure 3.1.16: Servo Motor Mounts



Shown in Figure 3.1.15, each servo was then mounted next to the pvc pocket such that the servo could act as a latching mechanism. Each servo motor was mounted using a 3D printed motor mount, as shown in Figure 3.1.16, that was made through the 3D printing services offered by the university. Attached to each servo was a steel arm that would extend under the steel plate to take stress off of the motor itself. From here, as a ball was pocketed, there was a delay by the microcontroller in order to determine whether the ball was a solid, stripe or the cue ball. Once the ball was determined to be solid or stripe, the servo motor would rotate 180°, releasing the balls into the track that runs to the gantry system.

Figure 3.1.17: Gantry System Complete View

Figure 3.1.17 shows the completed construction of the gantry re-racking system. The gantry, the device with the funnel, rides on the x and y axis to move to the desired positions. The axis are ball screws driven by the stepper motors. The balls screws are 25 inches long, but due to the design of the re-rack system, only 18 inches of length are utilized in the x and y direction. The pool table triangle measures 14 inches wide by 12 ¼ inches tall so 18 inches is more than enough room to re-rack the billiard balls. The x-axis arm was cut to a longer length to account for the extra space needed to mount the stepper motors. This also helped balance the load on the ball screws below by allowing the arm to position the motors directly above the ball screws. End and motor mounts were utilized to secure the stepper motors and the ball screws to the wooden frame. A flexible coupler was used to attach the ball screw to the stepper motors. Limit switches are located on one side of each axis, with one limit switch at the start and stop position.

Figure 3.1.18: Side View of Re-racking System

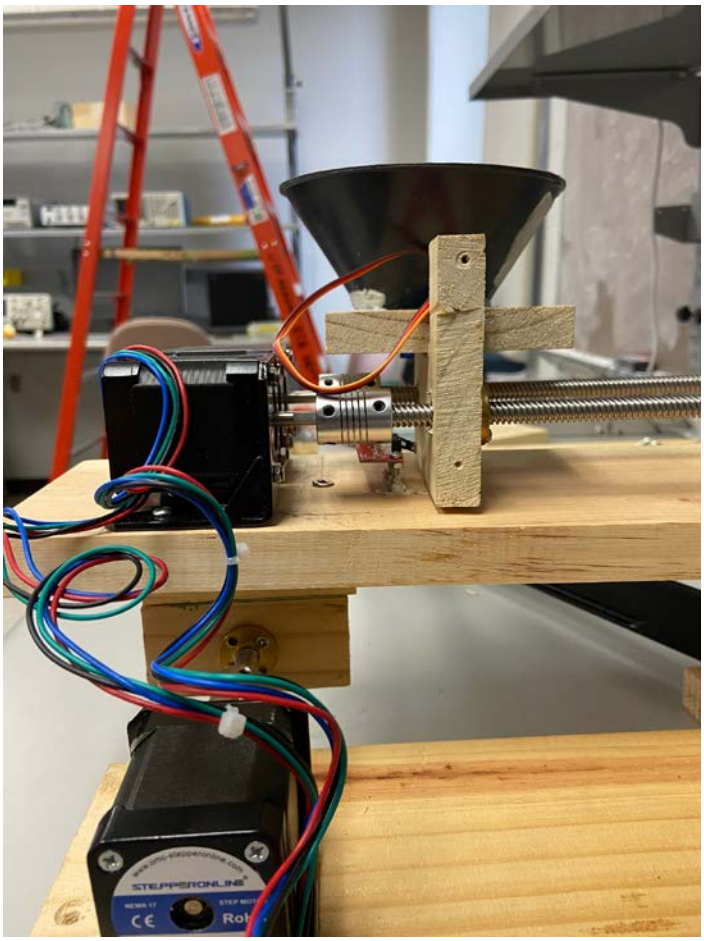


Figure 3.1.18 shows a side view of the re-racking system. The x-axis home limit switch is shown. This prevents the stepper motors from binding by stopping the gantry from coming into contact with the coupler. The x-axis was mounted on 2-inch spacers so that the y-axis could return back to its starting position without colliding with the x-axis.

Figure 3.1.19: Y-Axis Limit Switch

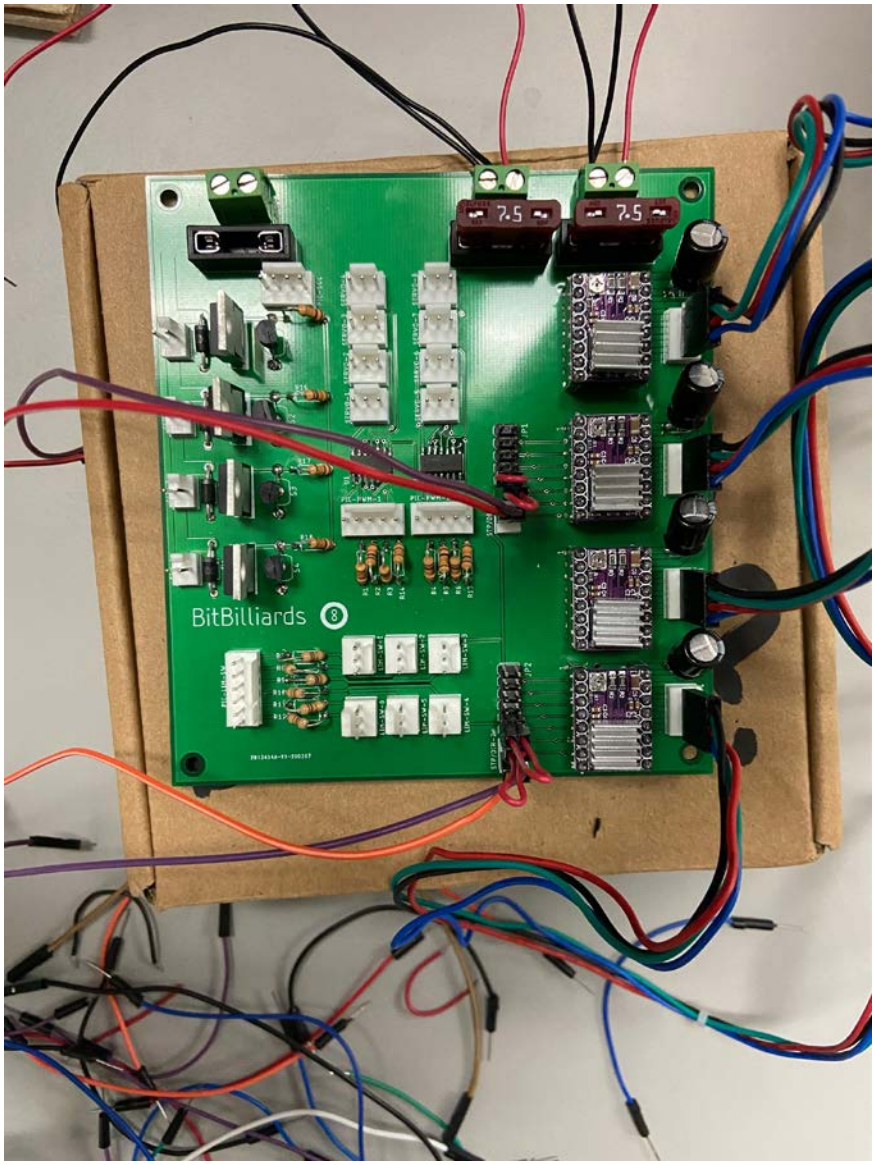
Figure 3.1.19 shows the y-axis stop limit switch. The limit switches were mounted on standoff screws which were then drilled and glued into the wooden frame. Also pictured is one of the end mounts which secures the ball screw in place. The end mount needed to be attached to a spacer to help keep the ball screw level with the stepper motor. This helped reduce the wear and stress on the motor.

Figure 3.1.20: Re-rack Gantry Construction

Figure 3.1.20 shows the constructed gantry for the re-racking system. This consisted of a funnel cut to allow a ball to pass through attached to a wooden plate that is moved on the x-axis. A servo motor is mounted under the plate, and a limit switch is built into the funnel. This allows

for the funnel to detect when a ball is placed inside, and the servo is then used to release the contained ball into the pool triangle below. A piece of metal tube is attached to the servo arm to provide the necessary reinforcement to hold a billiard ball.

Figure 3.1.21: Assembled Adapter Board



3.2. Software Design (DM/RM):

Table 3.2.1: Microcontroller Fundamental Requirement Table

Byte array	Microcontroller	Servo Position
		Stepper Motor X and Y Position

Module	Microcontroller
Designer	David Milostan
Inputs	Byte array
Outputs	Servo Position, Stepper Motor X and Y Position
Description	Send servo motor position to go to based on which pocket a ball has been made in and send the stepper motor an x and y coordinate to move to based on which ball has been made.

Table 3.2.2: Server Fundamental Requirement Table

Power	Server	App Data Signal
Microprocessor		
App Data Request		

Module	Server
Designer	David Milostan, Rodney Morgan
Inputs	Power: 120VAC
	Data from microprocessor
	App Data request signal
Outputs	Data signal to microprocessor
Description	The camera will be used to send live video of the pool table to a microprocessor.
	The video will contain ball placement and movement on the table.

Table 3.2.3: Mobile App Fundamental Requirement Table

Server	Mobile App	Server

Module	Mobile Application
Designer	David Milostan, Rodney Morgan
Inputs	Server data signal
Outputs	Server data signal
Description	The mobile app will take the data received from the server and display it on an infographic to show the status of games in progress.

Figure 3.2.1: Gate System Software Flowchart

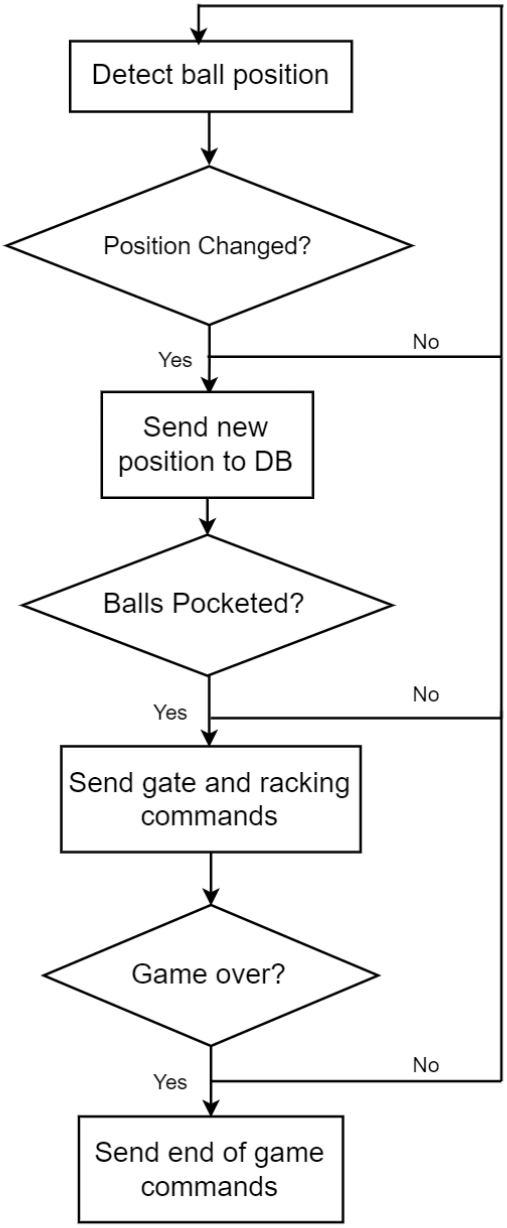


Figure 3.2.2: Racking System Software Flowchart

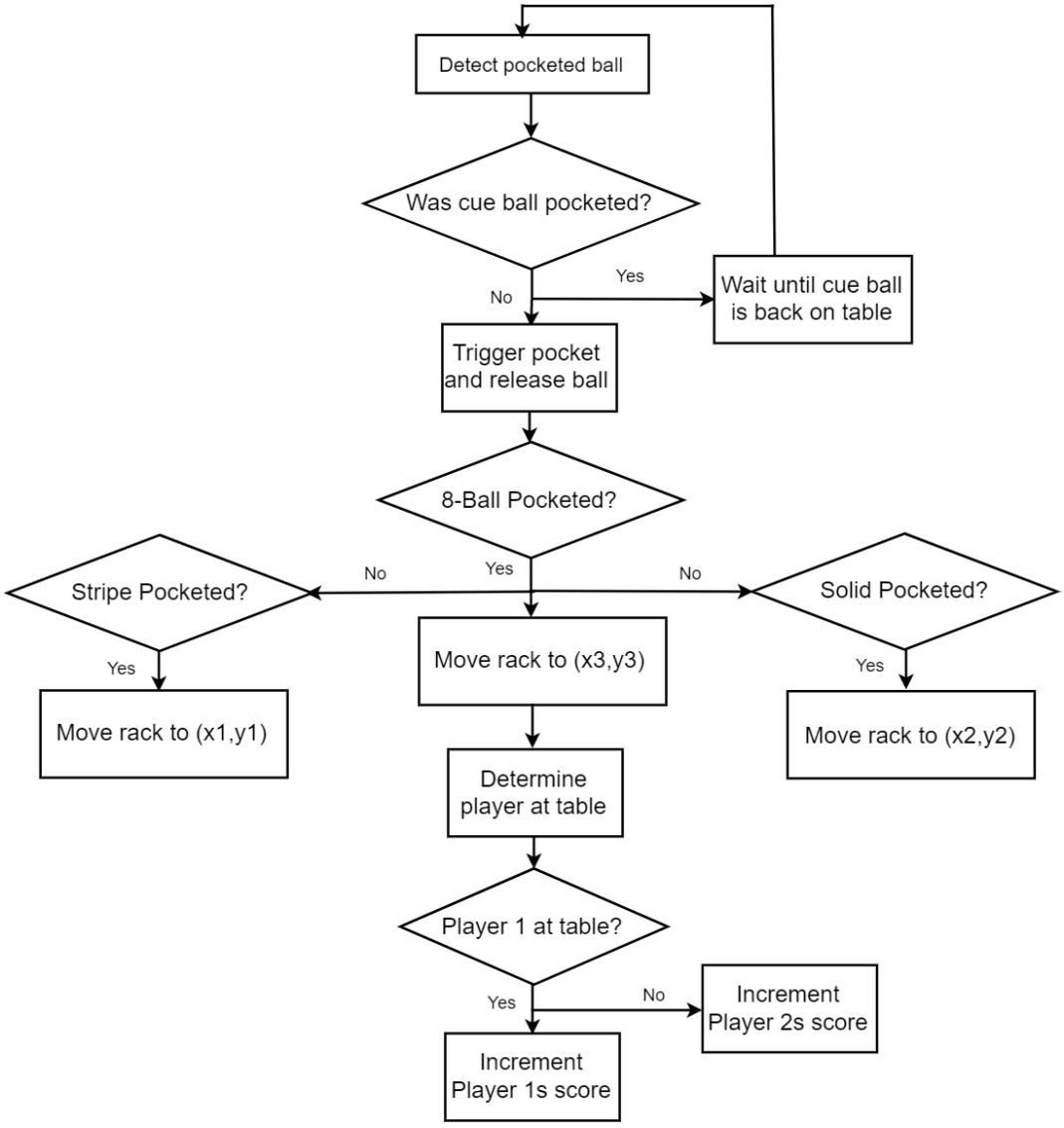


Figure 3.2.3: Serial Communication Software Python - Opening Serial Port

```
def open_serial_port():
    ser = serial.Serial(port='COM3', timeout=0, rtscts=0)
    print("Connected to ", ser.name)
```

The first thing that needs to be done to communicate from the PC to the PIC24 via serial communication is to open the communication port that the data is going to be sent through. This is done in the python code by assigning the port COM3 and setting the flow control to RTSCTS (Ready To Send and Clear To Send). After that has been set up the last thing to do is send the data. This is done as follows.

Figure 3.2.4: Serial Communication Software Python - Transmitting Byte

```
def sendSerialData(ID, Pocket):
    data = hex(Pocket | ID)[2:]
    ser.write(data.encode())
```

Figure 3.2.5: Serial Communication Software C - Setting Registers for UART

```
void InitU1(void) {
    U1BRG = 103; //415 //Baud rate with 9600
    U1MODE = 0x8000; //8008 //Enable UART2, BRGH = 0
    // Idle state = 1, 8 data, No parity, 1 Stop bit
    U1STA = 0x0400; // See data sheet, pg. 150, Transmit Enable
    // Following lines pertain Hardware handshaking
}
```

On the microcontroller side UART communication needs to be set up to receive data. To first enable UART only bit 15 is set to high in U1MODE and everything else is set low giving the hexadecimal value 0x8000. Then the UART has to be set up to receive that data which is setting the receive enabled bit in U1STA which is bit 12 giving the value of 0x1000. Finally,

the baud rate has to be set which is how many bits can be transferred per second. A baud rate of 9600 is good for this application. U1BRG is the baud rate generator. Setting this to 103 will give a baud rate of 9600. U1BRG can be calculated with the formula

U1BRG = ((FCY/Desired Baud Rate)/16) – 1.

Figure 3.2.6: Serial Communication Software C - Receiving Bytes

```
while(1)
{
    data = getU1();
}
 char getU1(void) {
    while (!U1STAbits . URXDA);
    return U1RXREG;
 }
```

In the main loop getU1 is called and the program sits in this function until a byte is received in the receive register, U1RXREG, and then that byte is returned to the character variable data.

Figure 3.2.7: Gate System Software

```
while(1)
{
    data = getU1();
    moveServo(data);
}
```

```
void moveServo(char cmd)
{
    if(cmd == '0')  //S3
    {
        Pocket1Open();
        ms_delay(1000);
        ms_delay(1000);
        Pocket1Close();
        moveStepperFWD(5);
        ms_delay(500);
        moveStepperREV(5);
    }
    else if(data == '1') //S5
    {
        Pocket1Open();
        ms_delay(1000);
        ms_delay(1000);
        Pocket1Close();
        moveStepperFWD(2);
        ms_delay(500);
        moveStepperREV(2);
    }
    else if(data == '2') //S4
    {
        Pocket2Open();
        ms_delay(1000);
        ms_delay(1000);
        Pocket2Close();
        moveStepperFWD(5);
        ms_delay(500);
        moveStepperREV(5);
    }
```

```
void Pocket_Open(data[i])
{
    switch(data[i])
    {

        case 1:
            //Open Pocket 1;
            break ;

        case 2:
            //Open Pocket 2;
            break;

        case 3:
            //Open Pocket 3;
            break;

        case 4:
            //Open Pocket 4;
            break;

        case 5:
            //Open Pocket 5;
            break;

        case 6:
            //Open Pocket 6;
            break ;

        default:
            break;
    }
}
```

Once the data is received and stored in the data variable it is sent to the moveServo function. Depending on the data sent different scenarios will take place. For an example if a ‘0’ is received then Pocket1Open will be called to turn the servo motor on pocket one 180 degrees and stay there for two seconds to let the balls drop into the racking system. Then Pocket1Close will be called to turn the servo motor back to degree 0 which will stop any balls from moving further than the pocket. Moving the servo to different positions is done by setting the duty through the output compare register which is shown below in figure 3.2.8. The code shown

below is exactly the same for pocket one and two except a different output compare module is used so the signals can be triggered separately.

Figure 3.2.8: Gate System Software - Set Servo Degree

```
void Pocket1Open(void)
{
    OC2R = 5000; //180 degrees WORKING //pin 23
}

void Pocket1Close(void)
{
    OC2R = 1000; //0 degrees WORKING //pin23
}
```

Figure 3.2.9: Racking System Software - Move Stepper to Coordinate

```
void moveServo(char cmd)
{
    if(cmd == '0')  //S3
    {
        Pocket1Open();
        ms_delay(1000);
        ms_delay(1000);
        Pocket1Close();
        moveStepperFWD(5);
        ms_delay(500);
        moveStepperREV(5);
    }
    else if(data == '1') //S5
    {
        Pocket1Open();
        ms_delay(1000);
        ms_delay(1000);
        Pocket1Close();
        moveStepperFWD(2);
        ms_delay(500);
        moveStepperREV(2);
    }
    else if(data == '2') //S4
    {
        Pocket2Open();
        ms_delay(1000);
        ms_delay(1000);
        Pocket2Close();
        moveStepperFWD(5);
        ms_delay(500);
        moveStepperREV(5);
    }
```

```
void moveStepperFWD_xaxis(int Count)
{
    int i;
    PORTAbits.RA0 = 1;      //DIR pin, PIN 17
    for(i = 0; i < Count; i++)
    {
        OC2RS = 14400;      //PIN 23 //New 16M
        ms_delay(1000);     //1 sec delay
    }
    OC2RS = 0;
    i = 0;
}

void moveStepperREV_xaxis(int Count)
{
    int i;
    PORTAbits.RA0 = 0;      //DIR pin, PIN 17
    for(i = 0; i < Count; i++)
    {
        OC2RS = 14400;      //PIN 23 //New 16M
        ms_delay(1000);     //1 sec delay
    }
    OC2RS = 0;
    i = 0;
}
```

Moving the stepper motor happens after the ball is released from the pocket and is in the racking tube that will move to the proper position in the rack to drop the ball. The function called to do this is moveStepperFWD which sets RA0 which is connected to the direction pin. This will spin clockwise if high and counterclockwise if low. The output compare register will trigger a rising edge to move the stepper as many times as the value passed into the function along with a one second delay for each trigger so the stepper will move for five seconds if the value five is passed into the function. The moveStepperREV works the same exact way except in the opposite direction. It will run for the same amount of time so the stepper motor will go back to the home position to go and pick up the next ball.

Figure 3.2.10: Code Snippet for Detecting the Pool Table

```
def calibrate_table():
    #Grab the HSV threshold of table perimeter
    MIN_HSV = np.array([0, 0, 0],np.uint8)
    MAX_HSV = np.array([255, 255, 204],np.uint8)

    while 1:
        #Capture a frame from camera
        ret, calibrate = cap.read()

        if calibrate is not None:
            #Blurr image to get rid of imperfections
            #and convert to hsv so image can be masked
            blurred = cv2.GaussianBlur(calibrate, (11, 11), 0)
            hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
            mask = cv2.inRange(hsv, MIN_HSV, MAX_HSV)

            #Find the contours in the masked image
            cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
            cnts = imutils.grab_contours(cnts)

            if len(cnts) > 0:
                #Get only the biggest contours from the image
                c = max(cnts, key=cv2.contourArea)
                if len(c) > 4:
                    #Get the approximation of the tables edges
                    #and resize the frame to those edges
                    epsilon = 0.01 * cv2.arcLength(c, True)
                    approx = cv2.approxPolyDP(c, epsilon, True)
                    cv2.drawContours(calibrate, [approx], 0, (0,255,), 2)

                    if approx[2,0,0] < approx[0,0,0]:
                        calibrate = calibrate[approx[0,0,1]:approx[2,0,1],approx[2,0,0]:approx[0,0,0]]
                    else:
                        calibrate = calibrate[approx[0,0,1]:approx[2,0,1],approx[0,0,0]:approx[2,0,0]]

                    cv2.imshow("Calibration", calibrate)

                    #Calculate the standard deviation of pixels in grayscaled image
                    #the lower the standard deviation means the pixels are blurred together
                    #and the image is not focused
                    temp = cv2.cvtColor(calibrate, cv2.COLOR_BGR2GRAY)
                    lap = cv2.Laplacian(temp, cv2.CV_16S)
                    mean, stddev = cv2.meanStdDev(lap)

                    #Call this function again to get camera in focus
                    #when the standard deviation is below a certain threshold
                    #else return the coordinates for the approximation of the tables edges
                    k = cv2.waitKey(30) & 0xff
                    if stddev[0,0] < 9:
                        print('cam not focused')
                    elif stddev[0,0] >= 9 and k == 27:
                        break
    return(approx)
```

For this implementation, a Logitech C920 will be used as a video input device. When the

program is started, the program will verify that the pool table can be recognized. This can be accomplished by masking out the pool table's perimeter, finding its area, and verifying that it matches the dimensions of the table area. To do this, the HSV (hue saturation value) threshold for the table's perimeter must be found and entered in line 26-27 in Figure 3.2.10. The approxPolyDP function will then get the area of the table based on the HSV threshold. The x and y values can then be used from this to resize the image coming from the video feed, so the main area of focus is around only the pool table. This will make processing the images quicker and also prevent false readings later on from outside sources around the pool table. The standard deviation of the pixels is also calculated on line 66 to make sure the camera is in focus. This will continuously run until the user presses enter verifying that they are happy with the calibration of the table.

Figure 3.2.11: Detecting Pocket Boundaries

```
78  def pocket_boundaries(coord):
79      POCKET_MIN_HSV = np.array([83, 153, 0],np.uint8)
80      POCKET_MAX_HSV = np.array([255, 255, 255],np.uint8)
81
82      ret, calibrate = cap.read()
83      # only proceed if at least one contour was found
84      if calibrate is not None:
85          calibrate = calibrate[coord[0,0,1]:coord[2,0,1],(coord[2,0,0]):(coord[0,0,0])]
86          #calibrate = calibrate[coord[0,0,1]:coord[2,0,1],(coord[0,0,0]):(coord[2,0,0])]
87          height, width = calibrate.shape[:2]
88          blurred = cv2.GaussianBlur(calibrate, (11, 11), 0)
89          hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
90          mask = cv2.inRange(hsv, POCKET_MIN_HSV, POCKET_MAX_HSV)
91
92          cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
93          cnts = imutils.grab_contours(cnts)
94
95          for c in cnts:
96              # draw the contour and show it
97              cv2.drawContours(calibrate, [c], -1, (0, 255, 0), 2)
98              cv2.imshow("Image", calibrate)
99          return cnts
```

The coordinates of the pocket boundaries can be found by masking out the HSV values for everything but the table border. These coordinates are used to crop the frame and understand

when balls have been pocketed. The code in Figure 3.2.11 shows the process of grabbing the coordinates of the border, storing the coordinates in local variables, and drawing the border of the table to the program's live output.

Figure 3.2.12.1: Creating Each Ball to be Tracked

```
156   #Initialize each ball that is supposed to be on the table
157   #and store them all in an array
158   cue_ball = Ball("cue")
159   one_ball = Ball("one")
160   two_ball = Ball("two")
161   three_ball = Ball("three")
162   four_ball = Ball("four")
163   five_ball = Ball("five")
164   six_ball = Ball("six")
165   seven_ball = Ball("seven")
166   eight_ball = Ball("eight")
167   nine_ball = Ball("nine")
\david\Documents\School\Spring 2020\Senior Design Project II\bitBilliardsRunners_Final\hsvFin
168   ten_ball = Ball("ten")
169   eleven_ball = Ball("eleven")
170   twelve_ball = Ball("twelve")
171   thirteen_ball = Ball("thirteen")
172   fourteen_ball = Ball("fourteen")
173   fifteen_ball = Ball("fifteen")
174
175   ball_objs = [cue_ball,one_ball,two_ball,
176       three_ball,four_ball,five_ball,
177       six_ball,seven_ball,eight_ball,
178       nine_ball,ten_ball,eleven_ball,
179       twelve_ball,thirteen_ball,fourteen_ball,
180       fifteen_ball
181   ]
```

Each ball on the table has a ball object created for them that way each one can have properties of its own. This is important for not only being able to see each ball and track it, but also later on being able to determine which ball is moving or has been pocketed during the game.

Figure 3.2.12.2: Ball Object

```
7   class Ball:
8       def __init__(self, ballNumber):
9           MIN_HSV = {
10              "cue": np.array([0, 0, 200]),
11              "one": np.array([28, 28, 230],np.uint8),
12              "two": np.array([107, 220, 178], np.uint8),
13              "three": np.array([137, 98, 125],np.uint8), #
14              "four": np.array([120, 100, 178], np.uint8), # tweak this value
15              "six": np.array([74, 118, 0], np.uint8),
16              "seven": np.array([171, 62, 16], np.uint8),
17              "eight": np.array([97, 98, 0], np.uint8),
18              "nine": np.array([0, 0, 0],np.uint8),
19              "ten": np.array([0, 0, 0], np.uint8),
20              "eleven": np.array([0, 0, 0],np.uint8),
21              "twelve": np.array([0, 0, 0], np.uint8),
22              "thirteen": np.array([0, 0, 0], np.uint8),
23              "fourteen": np.array([0, 0, 0], np.uint8),
24              "fifteen": np.array([0, 0, 0], np.uint8)
25          }
26
27          MAX_HSV = {
28              "cue": np.array([11, 9, 255]),
29              "one": np.array([30, 77, 255], np.uint8),
30              "two": np.array([121, 251, 255], np.uint8),
31              "three": np.array([186, 255, 255], np.uint8), # 186, 255, 255
32              "four": np.array([132, 142, 206], np.uint8), # tweak this value
33              "six": np.array([102, 208, 164], np.uint8),
34              "seven": np.array([211, 178, 211], np.uint8),
35              "eight": np.array([134, 255, 111], np.uint8),
36              "nine": np.array([0, 0, 0],np.uint8),
37              "ten": np.array([0, 0, 0], np.uint8),
38              "eleven": np.array([0, 0, 0],np.uint8),
39              "twelve": np.array([0, 0, 0], np.uint8),
40              "thirteen": np.array([0, 0, 0], np.uint8),
41              "fourteen": np.array([0, 0, 0], np.uint8),
42              "fifteen": np.array([0, 0, 0], np.uint8)
43          }
```

School\Spring 2020\Senior Design Project II\bitBilliardsRunners_Final\extractHSV.py

```
    ID = {
        "cue": 0x00,
        "one": 0x01,
        "two": 0x02,
        "three": 0x03,
        "four": 0x04,
        "five": 0x05,
        "six": 0x06,
        "seven": 0x01,
        "eight": 0x08,
        "nine": 0x09,
        "ten": 0x0A,
        "eleven": 0x0B,
        "twelve": 0x0C,
        "thirteen": 0x0D,
        "fourteen": 0x0E,
        "fifteen": 0x0F
    }

    NAME = {
        "cue": "cueball",
        "one": "oneball",
        "two": "twoball",
        "three": "threeball",
        "four": "fourball",
        "five": "fiveball",
        "six": "sixball",
        "seven": "sevenball",
        "eight": "eightball",
        "nine": "nineball",
        "ten": "tenball",
        "eleven": "elevenball",
        "twelve": "twelveball",
        "thirteen": "thirteenball",
        "fourteen": "fourteenball",
        "fifteen": "fifteenball"
    }
```

```
83          POCKETED = {
84              "cue": False,
85              "one": False,
86              "two": False,
87              "three": False,
88              "four": False,
89              "five": False,
90              "six": False,
91              "seven": False,
92              "eight": False,
93              "nine": False,
94              "ten": False,
95              "eleven": False,
96              "twelve": False,
97              "thirteen": False,
98              "fourteen": False,
99              "fifteen": False
100         }
101
102         self.MIN_HSV = MIN_HSV.get(ballNumber, None)
103         self.MAX_HSV = MAX_HSV.get(ballNumber, None)
104         self.ID = ID.get(ballNumber, None)
105         self.NAME = NAME.get(ballNumber, None)
106         self.POCKETED = POCKETED.get(ballNumber, None)
107         self.mask = None
108         self.dbUpdated = False
109         self.ballMoving = False
110         self.ballPocketed = False
111         self.buffer = deque(maxlen=64)
112
113         self.center = (0,0)
114         self.rad = 0
```

The Ball class shown above in figure 3.2.12.2 shows each of the properties that each individual ball will have. The main property is the MIN_HSV and MAX_HSV values. These are used to mask out everything in the image except for that particular ball. So, for each ball that should be the only thing seen in the image at that moment. The next properties ID and NAME are for identification purposes. The ID is a hex value that will later be combined with the hex value for the pocket the ball was made in. For example, pocket 1 (Hex value 0x10) and the three ball (Hex value 0x03) will combine to be 0x13. The hex values work out really well since they can represent sixteen 0-15 as 0-F and there are only six pockets and exactly sixteen balls. The NAME property is for naming purposes in the database. The final property is the deque which keeps track of the 64 most recent coordinates of the ball.

Figure 3.2.13.1: Masking Each Ball

```
223             #Construct a mask for each ball on the table
224             masks =[]
225             for ball in ball_objs:
226                 ball.setMask(hsv, boundaries)
227                 masks.append(ball.getMask())
```

The mask for each ball is made with the MIN_HSV and MAX_HSV values in the ball class shown in figure 3.2.12.2 so each ball can individually be identified. The pocket boundaries are passed to the setMask function for later use. Each individual mask is then stored in an array for processing.

Figure 3.2.13.2: Masking Continued

```
299     def setMask(self, hsv, bounds):
300         mask = cv2.inRange(hsv, self.MIN_HSV, self.MAX_HSV)
301         kernel = np.ones((11,11),np.uint8)
302         mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
303         self.mask = mask
304         self.updateBuffer(bounds)
```

The setMask function is called in figure 3.2.13.1 and is shown in detail here. After the mask is applied the pocket bounds are passed to the updateBuffer function where the main tracking is done.

Figure 3.2.14.1: Tracking Algorithm

```
116     # analyzes the lastFrame ORIGINAL
117     def updateBuffer(self, bounds):
118         ball_cnts = cv2.findContours(self.mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
119         ball_cnts = imutils.grab_contours(ball_cnts)
120         ball_center = None
121 
122         if len(ball_cnts) > 0:
123             # find the largest contour in the mask, then use
124             # it to compute the minimum enclosing circle and
125             # centroid
126             c = max(ball_cnts, key=cv2.contourArea)
127             ((x, y), radius) = cv2.minEnclosingCircle(c)
128             M = cv2.moments(c)
129 
130             # epsilon = 0.01*cv2.arcLength(c, True)
131             # approx = cv2.approxPolyDP(c, epsilon, True)
132             if M["m00"] != 0.0:
133                 ball_center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
134                 self.center = ball_center
135                 self.rad = radius
136 
137                 top = self.center[1] - self.rad
138                 bottom = self.center[1] + self.rad
139                 left = self.center[0] - self.rad
140                 right = self.center[0] - self.rad
141                 testPoints = [top, bottom, left, right]
142                 overCount = 0
143 
144                 for c in bounds:
145                     for j in range(0, 3, 1):
146                         if j == 0 or j == 1:
147                             overBoundary = cv2.pointPolygonTest(c, (self.center[0], testPoints[j]), False)
148                         else:
149                             overBoundary = cv2.pointPolygonTest(c, (testPoints[j], self.center[1]), False)
150                         if overBoundary == -1.0:
151                             overCount += 1
152 
153                 if overCount >= 3:
154                     self.ballPocketed = True
155                 else:
156                     self.ballPocketed = False
157         self.isBallMoving()
158         self.buffer.appendleft(ball_center)
```

With the mask that was just applied to the image the contours of the ball can be found with the findContours function that is able to determine the shape of the object based on the hsv threshold previously applied. The next step is to find the largest contour or the value that is closest to an exact match of the HSV values and pass that contour to the minEnclosingCircle function to get a perfect circle with an x and y coordinate as well as a radius. Next an image moment is captured which gets a weighted average of the pixels which are used to calculate the center of the ball. Now that there is a center and a radius the top, bottom, left, and right edge of

the ball need to be calculated. Each edge of the ball can then be passed to the pointPolygonTest. This function will take a point and determine if it is inside, on, or outside of a polygon shape. The polygon shape in question is the boundary for the pockets. If it has been determined that 3 or more of the edges or out of the boundaries, then it is safe to say the ball has been pocketed.

Figure 3.2.14.2: Tracking Movement

```
355     def isBallMoving(self):
356         if len(self.buffer) >= 10 and not None in self.buffer:
357             std = np.std(self.buffer, axis=0)
358             #print(std)
359             if std[0] <= 0.7 or std[1] <= 0.7:
360                 if not self.dbUpdated:
361                     print(str(self.ID) + "-ball is not moving")
362                     self.ballMoving = False
363                     self.dbUpdated = True
364             else:
365                 if self.dbUpdated:
366                     print(str(self.ID) + "-BALL IS MOVING!!!")
367                     self.ballMoving = True
368                     self.dbUpdated = False
369         elif all(elem == None for elem in list(self.buffer)[:10]):
370             self.ballMoving = False
371             self.dbUpdated = True
372
```

To determine if a ball has moved all the stored points in the buffer are passed to a standard deviation function. If the standard deviation of those coordinates varies by more than 0.7 pixels then it is safe to say that the ball is moving. Once all of the balls have stopped moving then it will tell us that the database has to be updated with the new coordinates.

Figure 3.2.14.2: Decision Making

```
237         for ball in ball_objs:
238             pts = ball.getBuffer()
239 
240             if ball.getBallMoving():
241                 ballsMoving[ball.getID()] = True
242                 dbUpdated = False
243             else:
244                 if pts[0] is not None:
245                     bottom, left = ball.getTopLeft()
246                     positionDictionary['coordinates'].update({ball.getName():{
247                             "xPos": ((100 - (bottom/height)*100)),
248                             "yPos": ((left/width)*100)
249                         }})
250                 ballsMoving[ball.getID()] = False
251 
252                 if ball.getBallPocketed() and not pocketedDictionary[ball.getName()]:
253                     if(len(pts) > 11):
254                         if pts[11] is not None:
255                             xPos = ((100 - (pts[11][1]/height)*100))
256                             yPos = ((pts[11][0]/width)*100)
257                             pocket = getPocket(xPos, yPos)
258                             sendSerialData(ball.getID(), pocket)
259                             pocketedDictionary[ball.getName()] = True
260                 elif not ball.getBallPocketed():
261                     pocketedDictionary.update({ball.getName(): False})
262 
263             for i in range(1, len(pts)):
264                 if pts[i-1] is None or pts[i] is None:
265                     continue
266                 thickness = int(np.sqrt(len(pts) / float(i+1)) * 2.5)
267                 cv2.line(frame, pts[i-1], pts[i], (0, 0, 255), thickness)
268 
269         if not any(ballsMoving) and not dbUpdated:
270             data = json.loads(json.dumps(positionDictionary))
271             db.update(data)
272             dbUpdated = True
```

This is the main loop that makes decisions on whether or not the database needs to be updated and if serial data needs to be sent to the microcontroller. The first condition is if a ball is moving. If this is the case, then nothing happens because all balls have to stop before updates are sent out. If that ball is not moving, then its coordinates are stored in a dictionary that contains the coordinates of all the balls which will be sent to the database as an object for a bulk update. Then if the ball has been pocketed the most recent x and y coordinate are sent to the getPocket

function that returns the hex value of the pocket the ball was made into. The serial data with the ball id and pocket is then sent to the microcontroller and the ball is set as pocketed in the dictionary to send as a bulk update to the database.

Figure 3.2.15: Implementation of Python to Firebase Communication

```
132   #Set up database connection
133   config = {
134     "apiKey": "FIREBASE_API_KEY",
135     "authDomain": "bitbilliards.firebaseapp.com",
136     "databaseURL": "https://bitbilliards.firebaseio.com/",
137     "storageBucket": "bitbilliards.appspot.com"
138   }
139   firebase = pyrebase.initialize_app(config)
140   db = firebase.database()

272   db.update(data)
```

Once new position data is found, or a ball has been pocketed, the Python application will bundle the data into a JSON format and send it to a Firebase database via POST request. Firebase has been chosen for this project, because React Native built Mobile Apps can re-render its UI when Firebase synchronizes the application state. This makes it easy to program the live changes on the app, since React Native has functions that listen for database updates. React Native will use a GET request to retrieve the grid information of the balls and display it on a virtual pool table by comparing the grid to the pixels on the phone. To accomplish this, set up a real-time database from *firebase.google.com*, and use the credentials to configure the config information in Figure 3.2.15. Also note that Pyrebase is used as a helper class in this implementation for data management in Firebase.

Figure 3.2.16: Full Python Code


```python
import numpy as np
from cv2 import cv2
from collections import deque
import imutils
import math

class Ball:
    def __init__(self, ballNumber):
        MIN_HSV = {
            "cue": np.array([0, 0, 200]),
            "one": np.array([28, 28, 230],np.uint8),
            "two": np.array([107, 220, 178], np.uint8),
            "three": np.array([137, 98, 125],np.uint8), #
            "four": np.array([120, 100, 178], np.uint8), # tweak this value
            "six": np.array([74, 118, 0], np.uint8),
            "seven": np.array([171, 62, 16], np.uint8),
            "eight": np.array([97, 98, 0], np.uint8),
            "nine": np.array([0, 0, 0],np.uint8),
            "ten": np.array([0, 0, 0], np.uint8),
            "eleven": np.array([0, 0, 0],np.uint8),
            "twelve": np.array([0, 0, 0], np.uint8),
            "thirteen": np.array([0, 0, 0], np.uint8),
            "fourteen": np.array([0, 0, 0], np.uint8),
            "fifteen": np.array([0, 0, 0], np.uint8)
        }

        MAX_HSV = {
            "cue": np.array([11, 9, 255]),
            "one": np.array([30, 77, 255], np.uint8),
            "two": np.array([121, 251, 255], np.uint8),
            "three": np.array([186, 255, 255], np.uint8), # 186, 255, 255
            "four": np.array([132, 142, 206], np.uint8), # tweak this value
            "six": np.array([102, 208, 164], np.uint8),
            "seven": np.array([211, 178, 211], np.uint8),
            "eight": np.array([134, 255, 111], np.uint8),
            "nine": np.array([0, 0, 0],np.uint8),
            "ten": np.array([0, 0, 0], np.uint8),
            "eleven": np.array([0, 0, 0],np.uint8),
            "twelve": np.array([0, 0, 0], np.uint8),
            "thirteen": np.array([0, 0, 0], np.uint8),
            "fourteen": np.array([0, 0, 0], np.uint8),
            "fifteen": np.array([0, 0, 0], np.uint8)
        }

        ID = {
            "cue": 0x00,
            "one": 0x01,
            "two": 0x02,
            "three": 0x03,
            "four": 0x04,
            "five": 0x05,
            "six": 0x06,
            "seven": 0x01,
            "eight": 0x08,
            "nine": 0x09,
            "ten": 0x0A,
            "eleven": 0x0B,
            "twelve": 0x0C,
            "thirteen": 0x0D,
            "fourteen": 0x0E,
```

```
            "fifteen": 0x0F
        }

        NAME = {
            "cue": "cueball",
            "one": "oneball",
            "two": "twoball",
            "three": "threeball",
            "four": "fourball",
            "five": "fiveball",
            "six": "sixball",
            "seven": "sevenball",
            "eight": "eightball",
            "nine": "nineball",
            "ten": "tenball",
            "eleven": "elevenball",
            "twelve": "twelveball",
            "thirteen": "thirteenball",
            "fourteen": "fourteenball",
            "fifteen": "fifteenball"
        }

        POCKETED = {
            "cue": False,
            "one": False,
            "two": False,
            "three": False,
            "four": False,
            "five": False,
            "six": False,
            "seven": False,
            "eight": False,
            "nine": False,
            "ten": False,
            "eleven": False,
            "twelve": False,
            "thirteen": False,
            "fourteen": False,
            "fifteen": False
        }


        self.MIN_HSV = MIN_HSV.get(ballNumber, None)
        self.MAX_HSV = MAX_HSV.get(ballNumber, None)
        self.ID = ID.get(ballNumber, None)
        self.NAME = NAME.get(ballNumber, None)
        self.POCKETED = POCKETED.get(ballNumber, None)
        self.mask = None
        self.dbUpdated = False
        self.ballMoving = False
        self.ballPocketed = False
        self.buffer = deque(maxlen=64)

        self.center = (0,0)
        self.rad = 0

        self.shape = []
        self.cum_area = []

    # analyzes the lastFrame ORIGINAL
```

```
121     def updateBuffer(self, bounds):
122         ball_cnts = cv2.findContours(self.mask.copy(), cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)
123         ball_cnts = imutils.grab_contours(ball_cnts)
124         ball_center = None
125 
126         if len(ball_cnts) > 0:
127             # find the largest contour in the mask, then use
128             # it to compute the minimum enclosing circle and
129             # centroid
130             c = max(ball_cnts, key=cv2.contourArea)
131             ((x, y), radius) = cv2.minEnclosingCircle(c)
132             M = cv2.moments(c)
133 
134             # epsilon = 0.01*cv2.arcLength(c, True)
135             # approx = cv2.approxPolyDP(c, epsilon, True)
136             if M["m00"] != 0.0:
137                 ball_center = (int(M["m10"] / M["m00"]), int(M["m01"] /
    M["m00"]))
138                 self.center = ball_center
139                 self.rad = radius
140 
141                 top = self.center[1] - self.rad
142                 bottom = self.center[1] + self.rad
143                 left = self.center[0] - self.rad
144                 right = self.center[0] - self.rad
145                 testPoints = [top, bottom, left, right]
146                 overCount = 0
147 
148                 for c in bounds:
149                     for j in range(0, 3, 1):
150                         if j == 0 or j == 1:
151                             overBoundary = cv2.pointPolygonTest(c,
    (self.center[0], testPoints[j]), False)
152                         else:
153                             overBoundary = cv2.pointPolygonTest(c,
    (testPoints[j], self.center[1]), False)
154                         if overBoundary == -1.0:
155                             overCount += 1
156 
157                 if overCount >= 3:
158                     self.ballPocketed = True
159                 else:
160                     self.ballPocketed = False
161         self.isBallMoving()
162         self.buffer.appendleft(ball_center)
163 
164     def setMask(self, hsv, bounds):
165         mask = cv2.inRange(hsv, self.MIN_HSV, self.MAX_HSV)
166         kernel = np.ones((11,11),np.uint8)
167         mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
168         #mask = cv2.dilate(mask,None, iterations=4)
169         #mask = cv2.erode(mask, None, iterations=1)
170         self.mask = mask
171         self.updateBuffer(bounds)
172 
173     def getMask(self):
174         return(self.mask)
175 
176     def getBuffer(self):
```

```
177         return(self.buffer)
178 
179     def getID(self):
180         return(self.ID)
181 
182     def getName(self):
183         return(self.NAME)
184 
185     def getDbUpdated(self):
186         return(self.dbUpdated)
187 
188     def setDbUpdated(self, value):
189         self.dbUpdated = value
190 
191     def getBallMoving(self):
192         return(self.ballMoving)
193 
194     def getPocketedStatus(self):
195         return(self.POCKETED)
196 
197     def getBallPocketed(self):
198         return(self.ballPocketed)
199 
200     def setBallPocketed(self):
201         ball_cnts = cv2.findContours(self.mask.copy(), cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)
202         ball_cnts = imutils.grab_contours(ball_cnts)
203 
204         if len(ball_cnts) > 0:
205             # find the largest contour in the mask
206             c = max(ball_cnts, key=cv2.contourArea)
207 
208             epsilon = 0.01*cv2.arcLength(c, True)
209             approx = cv2.approxPolyDP(c, epsilon, True)
210 
211             if len(approx) <= 9:
212                 print(self.ballPocketed)
213                 self.ballPocketed = True
214             else:
215                 self.ballPocketed = False
216 
217     def getTopLeft(self):
218         bottom = self.center[1] + self.rad
219         left = self.center[0] - self.rad
220         return(bottom, left)
221 
222     def isBallMoving(self):
223         if len(self.buffer) >= 10 and not None in self.buffer:
224             std = np.std(self.buffer, axis=0)
225             #print(std)
226             if std[0] <= 0.7 or std[1] <= 0.7:
227                 if not self.dbUpdated:
228                     print(str(self.ID) + "-ball is not moving")
229                     self.ballMoving = False
230                     self.dbUpdated = True
231             else:
232                 if self.dbUpdated:
233                     print(str(self.ID) + "-BALL IS MOVING!!!")
234                     self.ballMoving = True
235                     self.dbUpdated = False
```

```
236         elif all(elem == None for elem in list(self.buffer)[:10]):
237             self.ballMoving = False
238             self.dbUpdated = True
239
```

```python
#! python3
from cv2 import cv2
from imutils.video import VideoStream
import imutils
import time
import numpy as np
from collections import deque
import pyrebase
import json
import serial
from Ball import Ball

def combine_images(array_of_images):
    #As long as there is more than one image in array combine first image with last
    #and pop last image from the array then recursivly call this function until
    #only one image is left
    if len(array_of_images) > 1:
        array_of_images[0] =  cv2.addWeighted(array_of_images[0], 1.0, array_of_images[len(array_of_images)-1], 1.0, 0)
        array_of_images.pop()
        return(combine_images(array_of_images))
    else:
        return(array_of_images[0])

def calibrate_table():
    #Grab the HSV threshold of table perimeter
    MIN_HSV = np.array([0, 0, 0],np.uint8)
    MAX_HSV = np.array([255, 255, 204],np.uint8)

    while 1:
        #Capture a frame from camera
        ret, calibrate = cap.read()

        if calibrate is not None:
            #Blurr image to get rid of imperfections
            #and convert to hsv so image can be masked
            blurred = cv2.GaussianBlur(calibrate, (11, 11), 0)
            hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
            mask = cv2.inRange(hsv, MIN_HSV, MAX_HSV)

            #Find the contours in the masked image
            cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
            cnts = imutils.grab_contours(cnts)

            if len(cnts) > 0:
                #Get only the biggest contours from the image
                c = max(cnts, key=cv2.contourArea)
                if len(c) > 4:
                    #Get the approximation of the tables edges
                    #and resize the frame to those edges
                    epsilon = 0.01 * cv2.arcLength(c, True)
                    approx = cv2.approxPolyDP(c, epsilon, True)
                    cv2.drawContours(calibrate, [approx], 0, (0,255,), 2)

                    if approx[2,0,0] < approx[0,0,0]:
                        calibrate = calibrate[approx[0,0,1]:approx[2,0,1],approx[2,0,0]:approx[0,0,0]]
```

```
56                      else:
57                          calibrate =
calibrate[approx[0,0,1]:approx[2,0,1],approx[0,0,0]:approx[2,0,0]]
58
59                      cv2.imshow("Calibration", calibrate)
60
61                      #Calculate the standard deviation of pixels in grayscaled
image
62                      #the lower the standard deviation means the pixels are
blurred together
63                      #and the image is not focused
64                      temp = cv2.cvtColor(calibrate, cv2.COLOR_BGR2GRAY)
65                      lap = cv2.Laplacian(temp, cv2.CV_16S)
66                      mean, stddev = cv2.meanStdDev(lap)
67
68                      #Call this function again to get camera in focus
69                      #when the standard deviation is below a certain threshold
70                      #else return the coordinates for the approximation of the
tables edges
71                      k = cv2.waitKey(30) & 0xff
72                      if stddev[0,0] < 9:
73                          print('cam not focused')
74                      elif stddev[0,0] >= 9 and k == 27:
75                          break
76      return(approx)
77
78  def pocket_boundaries(coord):
79      POCKET_MIN_HSV = np.array([83, 153, 0],np.uint8)
80      POCKET_MAX_HSV = np.array([255, 255, 255],np.uint8)
81
82      ret, calibrate = cap.read()
83      # only proceed if at least one contour was found
84      if calibrate is not None:
85          calibrate = calibrate[coord[0,0,1]:coord[2,0,1],(coord[2,0,0]):
(coord[0,0,0])]
86          #calibrate = calibrate[coord[0,0,1]:coord[2,0,1],(coord[0,0,0]):
(coord[2,0,0])]
87          height, width = calibrate.shape[:2]
88          blurred = cv2.GaussianBlur(calibrate, (11, 11), 0)
89          hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
90          mask = cv2.inRange(hsv, POCKET_MIN_HSV, POCKET_MAX_HSV)
91
92          cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
93          cnts = imutils.grab_contours(cnts)
94
95          for c in cnts:
96            # draw the contour and show it
97            cv2.drawContours(calibrate, [c], -1, (0, 255, 0), 2)
98            cv2.imshow("Image", calibrate)
99          return cnts
100
101 def getPocket(x, y):
102     Pocket = 0x00
103     if x > 75:
104         if y > 85:
105             Pocket = 0x10
106             print("Pocket 1")
107         elif y < 12:
108             Pocket = 0x50
```

```
            print("Pocket 5")
        elif y > 40 and y < 55:
            Pocket = 0x60
            print("Pocket 6")
    elif x < 15:
        if y > 85:
            Pocket = 0x20
            print("Pocket 2")
        elif y < 12:
            Pocket = 0x40
            print("Pocket 4")
        elif y > 40 and y < 55:
            Pocket = 0x30
            print("Pocket 3")
    return(Pocket)

def sendSerialData(ID, Pocket):
    data = hex(Pocket | ID)[2:]
    print(data)
    # data = bytes.fromhex(data)
    #print(data)
    ser.write(data.encode())

#Set up database connection
config = {
  "apiKey": "AIzaSyBdS5j90Qx02WghwEolK6sLFGr7lu1QN0U",
  "authDomain": "bitbilliards.firebaseapp.com",
  "databaseURL": "https://bitbilliards.firebaseio.com/",
  "storageBucket": "bitbilliards.appspot.com"
}
firebase = pyrebase.initialize_app(config)
db = firebase.database()
print("Connected to firebase")

ser = serial.Serial(port='COM3', timeout=0, rtscts=0)
print("Connected to ", ser.name)

#Set up camera to capture images
cap = cv2.VideoCapture(0,  cv2.CAP_DSHOW)
cap.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc(*"MJPG"))
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
cap.read()
print('cam initialized')
print('calibrating cam')
coordinate = calibrate_table()
print('calibration succesful!')
boundaries = pocket_boundaries(coordinate)

#Initialize each ball that is supposed to be on the table
#and store them all in an array
#cue_ball = Ball("cue")
one_ball = Ball("one")
two_ball = Ball("two")
three_ball = Ball("three")
#four_ball = Ball("four")
 #five_ball = Ball("five")
#six_ball = Ball("six")
#seven_ball = Ball("seven")
#eight_ball = Ball("eight")
```

```
169 #ball_objs = [one_ball, two_ball, three_ball, four_ball,  six_ball,
    seven_ball, eight_ball]
170 ball_objs = [one_ball, two_ball, three_ball]
171
172 ballsMoving = [False] * 16
173 dbUpdated = False
174 postionDeleted = False
175 positionDictionary = {'coordinates': {}}
176 pocketedDictionary = {
177     'oneball': one_ball.getPocketedStatus(),
178     'twoball': two_ball.getPocketedStatus(),
179     'threeball': three_ball.getPocketedStatus(),
180     'fourball': True,
181     'fiveball': True,
182     'sixball': True,
183     'sevenball': True,
184     'eightball': True,
185     'nineball': True,
186     'tenball': True,
187     'elevenball': True,
188     'twelveball': True,
189     'thirteenball': True,
190     'fourteenball': True,
191     'fifteenball': True
192     }
193
194 data = json.loads(json.dumps(pocketedDictionary))
195 db.child('pocketedballs').update(data)
196
197 while 1:
198     #Grab the current frame
199     ret, frame = cap.read()
200     tempDict = dict(pocketedDictionary)
201
202     if frame is not None:
203         #Resize the frame, blur it, and convert it to the HSV
204         #color space
205         frame = frame[coordinate[0,0,1]:coordinate[2,0,1],
    (coordinate[2,0,0]):(coordinate[0,0,0])]
206         height, width = frame.shape[:2]
207         blurred = cv2.GaussianBlur(frame, (11 ,11), 0)
208         hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
209
210         #Construct a mask for each ball on the table
211         masks =[]
212         for ball in ball_objs:
213             ball.setMask(hsv, boundaries)
214             masks.append(ball.getMask())
215
216         #Combine each seperate mask into a single masked image
217         if len(masks) > 1:
218             mask = combine_images(masks)
219             cv2.imshow("Mask", mask)
220         elif len(masks) == 1:
221             mask = masks[0]
222             cv2.imshow("Mask", mask)
223
224         for ball in ball_objs:
225             pts = ball.getBuffer()
226
```

```
227             if ball.getBallMoving():
228                 ballsMoving[ball.getID()] = True
229                 dbUpdated = False
230             else:
231                 if pts[0] is not None:
232                     bottom, left = ball.getTopLeft()
233                     positionDictionary['coordinates'].update({ball.getName():
    {
234                             "xPos": ((100 - (bottom/height)*100)),
235                             "yPos": ((left/width)*100)
236                         }})
237                 ballsMoving[ball.getID()] = False
238 
239                 if ball.getBallPocketed() and not
    pocketedDictionary[ball.getName()]:
240                     if(len(pts) > 11):
241                         if pts[11] is not None:
242                             xPos = ((100 - (pts[11][1]/height)*100))
243                             yPos = ((pts[11][0]/width)*100)
244                             pocket = getPocket(xPos, yPos)
245                             sendSerialData(ball.getID(), pocket)
246                             pocketedDictionary[ball.getName()] = True
247                 elif not ball.getBallPocketed():
248                     pocketedDictionary.update({ball.getName(): False})
249 
250             for i in range(1, len(pts)):
251                 if pts[i-1] is None or pts[i] is None:
252                     continue
253                 thickness = int(np.sqrt(len(pts) / float(i+1)) * 2.5)
254                 cv2.line(frame, pts[i-1], pts[i], (0, 0, 255), thickness)
255 
256         if not any(ballsMoving) and not dbUpdated:
257             data = json.loads(json.dumps(positionDictionary))
258             db.update(data)
259             dbUpdated = True
260 
261         for k2, v2 in pocketedDictionary.items():
262                 if k2 in tempDict:
263                     if v2 != tempDict[k2]:
264                         positionDictionary['coordinates'].pop(k2, None)
265                         postionDeleted = True
266 
267         if postionDeleted:
268             data = json.loads(json.dumps(pocketedDictionary))
269             db.child('pocketedballs').update(data)
270             data =  json.loads(json.dumps(positionDictionary))

271             db.update(data)
272             postionDeleted = False
273 
274         cv2.imshow('Frame', frame)
275         k = cv2.waitKey(30) & 0xff
276         if k == 27:
277             break
278 
279 cap.release()
280 cv2.destroyAllWindows()
281 #ser.close()
```

Figure 3.2.17: Implementation of React Native to Firebase Communication

```
8   const firebaseConfig = {
9     "apiKey": "FIREBASE_API_KEY",
10    "authDomain": "bitbilliards.firebaseapp.com",
11    "databaseURL": "https://bitbilliards.firebaseio.com/",
12    "storageBucket": "bitbilliards.appspot.com"
13  }

22    async componentDidMount() {
23      await Font.loadAsync({
24        Buldano: require("./assets/fonts/buldano.otf")
25      });
26      await firebase.initializeApp(firebaseConfig);
27      this.setState({renderedFont: true});
28    }
```

For the React Native implementation, Expo is the framework of choice for this project. To set up Expo, the following steps should be taken:

- Download NodeJS
- Run "`npm install expo-cli -global`"
- Type the command "`expo init bitBilliards`"
- "`cd bitBilliards`"
- "`expo start`"

Figure 3.2.17 shows the initial connection from the app to Firebase.

Figure 3.2.18: React Native Listening for Pocketed Balls

```
15    async componentDidMount() {
16      firebase.database().ref('pocketedballs').on('value', (snapshot) => {
17        this.setState({ballStatusObject: snapshot.val(), hasMounted: true})
18      })
19    }
```

When the Python application detects that balls are pocketed, the "pocketedballs" children are updated with new data. Each billiard ball is assigned a value of true or false based on its pocketed status. Figure 3.2.18 shows React Native asynchronously listening for these updates. The scoring markers on the screen are rendered as opaque if the ball is playable and translucent if the ball is pocketed.

Figure 3.2.19: Ball Coordinate Rendering

```
20      function getCoordinates(){
21          firebase.database().ref('coordinates').on('value', (snapshot) => {
22              setCoordinates(snapshot.toJSON());
23          })
24      }
```

Figure 3.2.19 shows the implementation of React Native listening for changes to the children of the “coordinates” database reference. When the database is updated from the Python application, the new pool table coordinates are stored in the React Native app and displayed on the screen.

Figure 3.2.20: BitBilliards App

Each sprite seen on the screen was created in Adobe Illustrator and saved as .SVG file for icon scalability. The app listens for changes to the Firebase database and displays the results.

Figure 3.2.21: Full BitBilliards React Native Code


```
1  import React from 'react'
2  import * as Font from 'expo-font';
3  import * as firebase from 'firebase';
4  import { StyleSheet, Text, View } from 'react-native';
5  import PocketedBallsIndicator from
   './components/PocketedBallsIndicator/PocketedBallsIndicator';
6  import LiveTable from './components/LiveTable/LiveTable';
7  const firebaseConfig = {
8    "apiKey": "FIREBASE_API_KEY",
9    "authDomain": "bitbilliards.firebaseapp.com",
10   "databaseURL": "https://bitbilliards.firebaseio.com/",
11   "storageBucket": "bitbilliards.appspot.com"
12 }
13 export default class App extends React.Component{
14   constructor(props){
15     super(props);
16     this.state = {
17       renderedFont: false,
18       ballStatus: {}
19     }
20   }
21   async componentDidMount() {
22     await Font.loadAsync({
23       Buldano: require("./assets/fonts/buldano.otf")
24     });
25     await firebase.initializeApp(firebaseConfig);
26     this.setState({renderedFont: true});
27   }
28   render(){
29     if(this.state.renderedFont){
30       return (
31         <View style={styles.container}>
32           <View style={styles.titleContainer}>
33             <Text style={styles.title}>BitBilliards</Text>
34           </View>
35           <View style={styles.bodyContainer}>
36             <PocketedBallsIndicator />
37             <LiveTable />
38           </View>
39
40         </View>
41       );
42     }
43     else{
44       return (
45         <View style={styles.container}>
46         </View>
47       );
48     }
49   }
50 }
51
52 const styles = StyleSheet.create({
53   container: {
54     flex: 1,
55     backgroundColor: '#0a6c03',
56     alignItems: 'center',
57     justifyContent: 'center',
58
59   },
```

```
  titleContainer: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center'
  },
  title: {
    fontSize: 56,
    fontFamily: 'Buldano',
    color: '#fff'
  },
  bodyContainer: {
    flex: 3,
    alignItems: 'center',
    justifyContent: 'flex-start',
    width: '100%',
    height: '100%',
  },
  scoringView: {
    flex: 1
  },
  gameStats: {
    backgroundColor: '#d2a'
  }
});

```

```
1  import React, { useEffect, useRef, useState } from 'react'
2  import { View, Text, StyleSheet, Dimensions} from 'react-native'
3  import PoolTable from '../../assets/sprites/PoolTable.svg';
4  import PoolBall from '../PoolBall/PoolBall';
5  import * as firebase from 'firebase';
6
7  const LiveTable = () => {
8
9      const loadedCoordinates = useRef(false);
10     const [coordinates, setCoordinates] = useState({});
11
12     useEffect(() => {
13         getCoordinates();
14
15         return () => {
16             loadedCoordinates.current = true;
17         };
18     }, []);
19
20     function getCoordinates(){
21         firebase.database().ref('coordinates').on('value', (snapshot) => {
22             setCoordinates(snapshot.toJSON());
23         })
24     }
25
26     let ballsToRender = Object.keys(coordinates).map((ballName) => {
27       return <PoolBall key={ballName} ball={ballName} coordinates=
   {coordinates[ballName]} />
28     })
29
30     if(loadedCoordinates){
31       console.log(coordinates);
32       return (
33         <View style={styles.poolTable}>
34             <PoolTable width={"100%"} height={"100%"} />
35             {ballsToRender}
36         </View>
37       )
38     }
39     else{
40       return(
41         <View style={styles.poolTable}>
42           <PoolTable width={"100%"} height={"100%"} />
43         </View>
44       )
45     }
46 }
47
48 const styles = StyleSheet.create({
49   poolTable: {
50     flex: 1,
51     width: '65.5%',
52     position: 'relative',
53     backgroundColor: 'white'
54   }
55 })
56
57 export default LiveTable
58
```

```
1  import React, { Component } from 'react'
2  import { Text, View, Image, Dimensions, StyleSheet} from 'react-native'
3  import * as firebase from 'firebase';
4  const {width, height} = Dimensions.get('window');
5  export default class PocketedBallsIndicator extends Component {
6
7      constructor(props){
8          super(props);
9          this.state = {
10             hasMounted: false,
11             ballStatusObject: {}
12         }
13     }
14
15     async componentDidMount() {
16         firebase.database().ref('pocketedballs').on('value', (snapshot) => {
17           this.setState({ballStatusObject: snapshot.val(), hasMounted: true})
18         })
19     }
20
21     getStyle(ballNumber){
22         obj = this.state.ballStatusObject;
23         if(this.state.ballStatusObject[ballNumber] == false){
24             return ({
25                 width: width* 0.1,
26                 height: width * 0.1,
27                 margin: width * 0.005
28             });
29         }
30         else{
31             return({
32                 width: width* 0.1,
33                 height: width * 0.1,
34                 margin: width * 0.005,
35                 opacity: 0.4
36             });
37         }
38     }
39
40     render() {
41         if(this.state.hasMounted){
42             console.log(this.state.ballStatusObject);
43             return (
44                 <View style={{display: 'flex', flexDirection: 'row',
justifyContent: 'space-between', position: 'absolute', width: '100%'}}>
45                     <View style={styles.solidContainer}>
46                         <Image style={this.getStyle("oneball")} source=
{require('../../Sprites/OneBall.png')} />
47                         <Image style={this.getStyle('twoball')} source=
{require('../../Sprites/TwoBall.png')} />
48                         <Image style={this.getStyle('threeball')} source=
{require('../../Sprites/ThreeBall.png')} />
49                         <Image style={this.getStyle('fourball')} source=
{require('../../Sprites/FourBall.png')} />
50                         <Image style={this.getStyle('fiveball')} source=
{require('../../Sprites/FiveBall.png')} />
51                         <Image style={this.getStyle('sixball')} source=
{require('../../Sprites/SixBall.png')} />
52                         <Image style={this.getStyle('sevenball')} source=
{require('../../Sprites/SevenBall.png')} />
```

```
53                          <Image style={this.getStyle('eightball')} source=
{require('../../Sprites/EightBall.png')} />
54                      </View>
55                      <View style={styles.stripeContainer}>
56                          <Image style={this.getStyle('nineball')} source=
{require('../../Sprites/NineBall.png')} />
57                          <Image style={this.getStyle('tenball')} source=
{require('../../Sprites/TenBall.png')} />
58                          <Image style={this.getStyle('elevenball')} source=
{require('../../Sprites/ElevenBall.png')} />
59                          <Image style={this.getStyle('twelveball')} source=
{require('../../Sprites/TwelveBall.png')} />
60                          <Image style={this.getStyle('thirteenball')} source=
{require('../../Sprites/ThirteenBall.png')} />
61                          <Image style={this.getStyle('fourteenball')} source=
{require('../../Sprites/FourteenBall.png')} />
62                          <Image style={this.getStyle('fifteenball')} source=
{require('../../Sprites/FifteenBall.png')} />
63                          <Image style={this.getStyle('eightball')} source=
{require('../../Sprites/EightBall.png')} />
64                      </View>
65                  </View>
66
67              )
68          }
69          else{
70              return(<View></View>)
71          }
72
73      }
74  }
75
76  const styles = StyleSheet.create({
77      solidContainer:{
78          left: width * 0.02
79      },
80      stripeContainer: {
81          right: width * 0.02
82      }
83  })
```

```
1  import React from 'react'
2  import { View, Dimensions } from 'react-native'
3  import OneBall from '../../assets/sprites/OneBall.svg';
4  import TwoBall from '../../assets/sprites/TwoBall.svg';
5  import ThreeBall from '../../assets/sprites/ThreeBall.svg';
6  import FourBall from '../../assets/sprites/FourBall.svg';
7  import FiveBall from '../../assets/sprites/FiveBall.svg';
8  import SixBall from '../../assets/sprites/SixBall.svg';
9  import SevenBall from '../../assets/sprites/SevenBall.svg';
10 import EightBall from '../../assets/sprites/EightBall.svg';
11 import NineBall from '../../assets/sprites/NineBall.svg';
12 import TenBall from '../../assets/sprites/TenBall.svg';
13 import ElevenBall from '../../assets/sprites/ElevenBall.svg';
14 import TwelveBall from '../../assets/sprites/TwelveBall.svg';
15 import ThirteenBall from '../../assets/sprites/ThirteenBall.svg';
16 import FourteenBall from '../../assets/sprites/FourteenBall.svg';
17 import FifteenBall from '../../assets/sprites/FifteenBall.svg';
18 import CueBall from '../../assets/sprites/CueBall.svg';
19
20 const {width, height} = Dimensions.get('window');
21
22
23 const PoolBall = (props) => {
24
25     function getBallPosition(){
26         if(props.coordinates != undefined){
27             return({
28                 position: 'absolute',
29                 height: '5.61798%',
30                 width: '5.61798%',
31                 left: getLeftPadding(props.coordinates['xPos']),
32                 top: getTopPadding(props.coordinates['yPos']),
33
34             })
35         }
36     }
37
38     function getLeftPadding(xPos){
39         if(xPos < 7.5){
40             return('7.5%');
41         }
42         else if(xPos > 87){
43             return('87%');
44         }
45         else{
46             return(xPos + '%');
47         }
48     }
49
50     function getTopPadding(yPos){
51         if(yPos < 2.5){
52             return('2.5%');
53         }
54         else if(yPos > 92){
55             return('92%');
56         }
57         else{
58             return(yPos + '%');
59         }
60     }
```

```
61
62      function getBall(){
63          switch(props.ball){
64              case('oneball'):
65                  return( <View style={getBallPosition()}>
66                              <OneBall width={"100%"} height={"100%"} />
67                          </View>
68                  )
69              case('twoball'):
70                  return( <View style={getBallPosition()}>
71                              <TwoBall width={"100%"} height={"100%"} />
72                          </View>
73                  )
74              case('threeball'):
75                  return( <View style={getBallPosition()}>
76                              <ThreeBall width={"100%"} height={"100%"} />
77                          </View>
78                  )
79              case('fourball'):
80                  return( <View style={getBallPosition()}>
81                              <FourBall width={"100%"} height={"100%"} />
82                          </View>
83                  )
84              case('fiveball'):
85                  return( <View style={getBallPosition()}>
86                              <FiveBall width={"100%"} height={"100%"} />
87                          </View>
88                  )
89              case('sixball'):
90                  return( <View style={getBallPosition()}>
91                              <SixBall width={"100%"} height={"100%"} />
92                          </View>
93                  )
94              case('sevenball'):
95                  return( <View style={getBallPosition()}>
96                              <SevenBall width={"100%"} height={"100%"} />
97                          </View>
98                  )
99              case('eightball'):
100                 return( <View style={getBallPosition()}>
101                             <EightBall width={"100%"} height={"100%"} />
102                         </View>
103                 )
104             case('nineball'):
105                 return( <View style={getBallPosition()}>
106                             <NineBall width={"100%"} height={"100%"} />
107                         </View>
108                 )
109             case('tenball'):
110                 return( <View style={getBallPosition()}>
111                             <TenBall width={"100%"} height={"100%"} />
112                         </View>
113                 )
114             case('elevenball'):
115                 return( <View style={getBallPosition()}>
116                             <ElevenBall width={"100%"} height={"100%"} />
117                         </View>
118                 )
119             case('twelveball'):
120                 return( <View style={getBallPosition()}>
```

```
                            <TwelveBall width={"100%"} height={"100%"} />
                        </View>
                )
            case('thirteenball'):
                return( <View style={getBallPosition()}>
                            <ThirteenBall width={"100%"} height={"100%"} />
                        </View>
                )
            case('fourteenball'):
                return( <View style={getBallPosition()}>
                            <FourteenBall width={"100%"} height={"100%"} />
                        </View>
                )
            case('fifteenball'):
                return( <View style={getBallPosition()}>
                            <FifteenBall width={"100%"} height={"100%"} />
                        </View>
                )
            case('cueball'):
                return( <View style={getBallPosition()}>
                            <CueBall width={"100%"} height={"100%"} />
                        </View>
                )
        }
    }

    if(!props.coordinates){
        return(null)
    }
    return (
        getBall()
    )
}


export default PoolBall


```

4. Financial Budget (GR)

Table 4.1.1: Original Budget

Supplies for table construction	$200
Electrical components	$200
Miscellaneous supplies	$200

The original budget for the project was a very simple 3-way split among table supplies, electrical components, and misc. supplies. The cost of PVC and other plywood was estimated and used to estimate the $200 need for the general track construction, gantry and repair of the table. The PCB design was still in progress so a rough estimate of $200 was used for the PCB cost, and then the PCB components. The remaining $200 would be used on the camera for image recognition and any other cables or etc. needed.

Table 4.1.2: Actual Budget

Qty.	Refdes	Part Num.	Description	Suggested Vendor	Vendor Part Num.	Catalog #/Page #/Hyperlink	Cost	Total Cost
1		C920	Camera for image processing	Amazon	B006JH8T3S	https://www.amazon.com/Lo	$63.43	$63.43
1			Full HD Webcam	Amazon	B07925CYP5	https://www.amazon.com/Wi	$36.91	$36.91
1		000151black	USB to RS-232	Amazon	B0753HBT12	https://www.amazon.com/Se	$7.25	$7.25
1			USB to TTL Serial Cable	Adafruit	954	https://www.adafruit.com/pro	$9.95	$9.95
4			Stepper motor	Amazon	17HS19-2004S1	https://www.amazon.com/gp	$13.99	$55.96
5			Stepper Motor Driver	Digikey	296-29503-1-ND	https://www.digikey.com/prod	$3.84	$19.20
1			Nema 17 Stepper Motor Mount	Amazon	43237-2	https://www.amazon.com/An	$14.99	$14.99
4			Ball Screw Kit	Amazon	THSL-500-8D	https://www.amazon.com/Sc	$10.99	$43.96
1			Shaft coupler	Amazon	7.50441E+11	https://www.amazon.com/YO	$5.99	$5.99
1			Pillow Block Bearing	Amazon	6.11982E+11	https://www.amazon.com/Eo	$8.69	$8.69
4			Solenoids	Digikey	1528-2797-ND	https://www.digikey.com/pro	$7.50	$30.00
3			Limit Switch	Digikey	P10870S-ND	https://www.digikey.com/pro	$0.28	$0.84
3			Limit Switch	Digikey	P11151SCT-ND	https://www.digikey.com/pro	$0.30	$0.90
2			Adapter board for Stepper Driver	Sparkfun	BOB-00500	https://www.sparkfun.com/pr	$3.95	$7.90
1			4Pcs SG90 9g Micro Servos	Amazon	B07MLR1498	https://www.amazon.com/Mic	$9.99	$9.99
1			SG90 Servos 5 Pcs	Amazon	B07Q6JQWNV	https://www.amazon.com/Do	$9.99	$9.99
1		0db1db7bd68ff3	Felt	Amazon	0db1db7bd68ff3f8772df82d8ab8b8b1	https://www.amazon.com/To	$33.99	$33.99
2			3"x10' PVC pipe	Lowes	Item # 23834 Model # PVC 04300 0600	https://www.lowes.com/pd/C	$10.80	$21.60
2			4"x2' PVC pipe	Lowes	Item # 294919 Model # PVC 04400 0200	https://www.lowes.com/pd/C	$12.96	$25.92
4			3" 90deg PVC elbows	Home Depot	Model # C4807LTHD3	https://www.homedepot.com	$6.98	$27.92
1			PVC Glue	Lowes	Item # 23467 Model # 31018L	https://www.lowes.com/pd/C	$5.78	$5.78
4	5V, 6V, 12V		Screw-terminal Connector	Digikey	A113320-ND	https://www.digikey.com/pro	$1.04	$4.16
5	C1, C2, C3, C4		100uF 50V Cap	Digikey	P15369CT-ND	https://www.digikey.com/pro	$0.50	$2.50
8			Molex 2-pin connector	STOCK	22-23-2021	LOCATION: H17		
16			Molex 3-pin connector	STOCK	22-23-2031	LOCATION: H18		
7			Molex 4-pin connector	STOCK	22-23-2041	LOCATION: I01		
3			Molex 5-pin connector	Digikey	22-23-2051	https://www.digikey.com/pro	$0.34	$1.02
2			Molex 6-pin connector	STOCK	22-23-2061	LOCATION: I01		
8			Molex 2-pin housing	STOCK	22-01-2027	LOCATION: H17		
16			Molex 3-pin housing	STOCK	22-01-2037	LOCATION: H18		
7			Molex 4-pin housing	STOCK	22-01-2047	LOCATION: I01		
3			Molex 5-pin housing	Digikey	22-01-2057	https://www.digikey.com/pro	$0.23	$0.69
2			Molex 6-pin housing	STOCK	22-01-2067	LOCATION: I02		
130	N/A		Molex pins	STOCK	08-50-0114	LOCATION: H16		
2	JP1, JP2	67997-212HLF	2 row 6 pin header	Digikey	609-3237-ND	https://www.digikey.com/pro	$0.82	$1.64
10	N/A	215297-8	CONN RCPT 8POS 0.1 GOLD PCB	Digikey	A106654-ND	https://www.digikey.com/pro	$1.72	$17.15
3	U1, U2	MC14081BDG	IC GATE AND 4CH 2-INP 14SOIC	Digikey	MC14081BDGOS-ND	https://www.digikey.com/pro	$0.51	$1.53
5	S1, S2, S3, S4	2N7000	MOSFET N-CH 60V 200MA TO-92	STOCK, Digikey	2N7000FS-ND	https://www.digikey.com/pro	$0.36	$1.80
5	D1, D2, D3, D4	MUR1100ERLG	Diode Gen Purpose 1000V 1A	STOCK, Digikey	MUR1100ERLGOSCT-ND	https://www.digikey.com/pro	$0.41	$2.05
5	Q1, Q2, Q3, Q4	BUZ11	MOSFET, 50V, 30A, TO-220 pkg	STOCK, Digikey	BUZ11-NR4941-ND	https://www.digikey.com/pro	$1.13	$5.65
15	R1, R2, R3, R4, R5, R6, R13, R14, R15, R16, R17, R18		10k-ohm resistor	STOCK				
8	R7, R8, R9, R10, R11, R12		20k-ohm resistor	STOCK				
4	F1, F2, F3	3557-2	FUSE BLOCK BLADE 500V 30A PCB	Digikey	36-3557-2-ND	https://www.digikey.com/pro	$1.03	$4.12
4	F1, F2, F3	028707.5PXCN	FUSE AUTOMOTIVE 7.5A 32VDC BLADE	Digikey	F4198-ND	https://www.digikey.com/pro	$0.28	$1.12
							Total	$484.59

The purchased parts are shown in Figure 4.1.2. The total price was $484.59. The biggest costs of the project were the cameras purchased, one at the beginning and then another camera to replace the initial faulty camera, and the PVC to construct the re-racking track under the table.

Another major cost was the re-racking system. It was underestimated how much it would cost to build from scratch. The motors and ball screws combined were over $100. This is not factoring in the motor mounts, end pieces, or limit switches. Most of the budget was spent on mechanical systems that were needed to be constructed. The adapter board PCB and components was the cheapest subsystem to build. It was assumed that components and PCBs cost much more, but due to the simplicity of the final design no expensive components were needed. The only component not purchased was the power supply to power the system. This point was not reached in the project.

7. Project Schedule (GR)

Figure 7.1.1: Design Gantt Chart Part 1

ID	Task Mode	Task Name	Duration	Start	Finish	Predec	Resource Names	% Complete
1		SDP I 2019						0%
2		**Project Design**	**89 days**	**Fri 8/30/19**	**Wed 11/27/19**			**3%**
3		**Midterm Report**	40 days	Fri 8/30/19	Wed 10/9/19			10%
4		Cover page	40 days	Fri 8/30/19	Wed 10/9/19		Everyone	0%
5		T of C, L of T, L of F	40 days	Fri 8/30/19	Wed 10/9/19		Everyone	0%
6		**Problem Statement (USE PRELIM)**	**40 days**	**Fri 8/30/19**	**Wed 10/9/19**		**Everyone**	**0%**
7		Need	40 days	Fri 8/30/19	Wed 10/9/19			0%
8		Objective	40 days	Fri 8/30/19	Wed 10/9/19			0%
9		Background	40 days	Fri 8/30/19	Wed 10/9/19			0%
10		Marketing Requirements	40 days	Fri 8/30/19	Wed 10/9/19			0%
11		Engineering Requirements Specification	40 days	Fri 8/30/19	Wed 10/9/19			0%
12		**Engineering Analysis**	**40 days**	**Fri 8/30/19**	**Wed 10/9/19**		**Everyone**	**0%**
13		**Circuits (DC, AC, Power, ...)**	**39.38 days**	**Fri 8/30/19**	**Tue 10/8/19**			**0%**
14		Power Supply					Kyle Steveson,Grant Reinbolt	0%
15		Electronics (analog and digital)	39.38 days	Fri 8/30/19	Tue 10/8/19			0%
16		**Signal Processing**	**39.38 days**	**Fri 8/30/19**	**Tue 10/8/19**			**0%**
17		Image Processing			Tue 10/8/19		Rodney Morgan	0%
18		**Communications (analog and digital)**	**39.38 days**	**Fri 8/30/19**	**Tue 10/8/19**			**0%**
19		Camera to PC			Tue 10/8/19		Rodney Morgan	0%
20		PC to Controller			Tue 10/8/19		David Milostan	0%
21		**Electromechanics**	**39.38 days**	**Fri 8/30/19**	**Tue 10/8/19**			**0%**
22		Re-rack system			Tue 10/8/19		Grant Reinbolt	0%
23		Gate System			Tue 10/8/19		Kyle Steveson	0%
24		**Computer Networks**	**40 days**	**Fri 8/30/19**	**Wed 10/9/19**			**0%**
25		Server Database			Wed 10/9/19		Rodney Morgan	0%
26		App			Wed 10/9/19		Rodney Morgan	0%
27		**Embedded Systems**	**40 days**	**Fri 8/30/19**	**Wed 10/9/19**			**0%**
28		Microcontroller					Grant Reinbolt,Kyle Steveson	0%
29		**Accepted Technical Design**	**40 days**	**Fri 8/30/19**	**Wed 10/9/19**			**0%**
30		**Hardware Design: Phase 1**	**39.38 days**	**Fri 8/30/19**	**Tue 10/8/19**		**Grant Reinbolt,Kyle Steveson**	**0%**
31		Hardware Block Diagrams Levels 0 thru N (w/ FR tables)	40 days	Fri 8/30/19	Wed 10/9/19			0%
32		**Software Design: Phase 1**	**39.38 days**	**Fri 8/30/19**	**Tue 10/8/19**		**David Milostan,Rodney Morgan**	**0%**
33		Software Behavior Models Levels 0 thru N (w/FR tables)	40 days	Fri 8/30/19	Wed 10/9/19			0%
34		**Mechanical Sketch**	**39.38 days**	**Fri 8/30/19**	**Tue 10/8/19**		**Grant Reinbolt,Kyle Steveson**	**0%**
35		Pool Table			Tue 10/8/19			0%
36		**Team Information**	40 days	Fri 8/30/19	Wed 10/9/19			0%
37		**Project Schedules**	**39.38 days**	**Fri 8/30/19**	**Tue 10/8/19**			**50%**
38		Time Chart	0.38 days	Mon 10/7/19	Tue 10/8/19		David Milostan	100%
39		Midterm Design Gantt Chart	39.38 days	Fri 8/30/19	Tue 10/8/19		Grant Reinbolt	50%
40		**References**	40 days	Fri 8/30/19	Wed 10/9/19			0%
41		Midterm Parts Request Form	40 days	Fri 8/30/19	Wed 10/9/19			0%
42		**Preliminary Design Presentations**	0 days	Thu 9/19/19	Thu 9/19/19			0%
43		Project Poster	13 days	Thu 10/10/19	Wed 10/23/19			0%
44		Final Design Report	48 days	Thu 10/10/19	Wed 11/27/19			0%
45		**Abstract**	48 days	Thu 10/10/19	Wed 11/27/19			0%
46		**Hardware Design: Phase 2**	**48 days**	**Thu 10/10/19**	**Wed 11/27/19**			**0%**

Page 1

Figure 7.1.2: Design Gantt Chart Part 2

ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors	Resource Names	% Complete
47		**Modules**	**48 days**	**Thu 10/10/19**	**Wed 11/27/19**			0%
48		Simulations	48 days	Thu 10/10/19	Wed 11/27/19			0%
49		**Schematics**	**48 days**	**Thu 10/10/19**	**Wed 11/27/19**			0%
50		Gate System	48.38 days	**Thu 10/10/19**	**Wed 11/27/19**		Kyle Steveson	0%
51		Power Supply	48.38 days	**Thu 10/10/19**	**Wed 11/27/19**		Kyle Steveson	0%
52		Gantry System	48.38 days	**Thu 10/10/19**	**Wed 11/27/19**		Grant Reinbolt	0%
53		Microcontroller	48.38 days	**Thu 10/10/19**	**Wed 11/27/19**		Grant Reinbolt	0%
54		**Software Design: Phase 2**	**48 days**	**Thu 10/10/19**	**Wed 11/27/19**			0%
55		**Modules**	**48 days**	**Thu 10/10/19**	**Wed 11/27/19**			0%
56		Microcontroller communication	47.38 days	Thu 10/10/19	Tue 11/26/19		David Milostan	0%
57		Image Processing	47.38 days	Thu 10/10/19	Tue 11/26/19		Rodney Morgan	0%
58		**Parts Lists**	**48 days**	**Thu 10/10/19**	**Wed 11/27/19**			0%
59		Parts List(s) for Schematics	47.38 days	Thu 10/10/19	Tue 11/26/19		Grant Reinbolt,Rodney Morgan	0%
60		Materials Budget list	48 days	Thu 10/10/19	Wed 11/27/19			0%
61		**Proposed Implementation Gantt Chart**	48 days	Thu 10/10/19	Wed 11/27/19			0%
62		**Conclusions and Recommendations**	48 days	Thu 10/10/19	Wed 11/27/19			0%
63		Final Parts Request Form	13 days	Tue 10/15/19	Mon 10/28/19			0%
64		Final Design Presentations Part 1	0 days	Thu 11/14/19	Thu 11/14/19			0%
65		Final Design Presentations Part 2	0 days	Thu 11/21/19	Thu 11/21/19			0%
66		Parts Request Form for Spring Semester	9 days	Wed 11/27/19	Fri 12/6/19			0%

Page 2

Figure 7.1.3: Implementation Gantt Chart

ID	Task Mode	Task Name	Duration	Start	Finish	Resource Names
1		**SDPII Implementation 2020**	**103 days**	**Mon 1/13/20**	**Fri 4/24/20**	
2		Revise Gantt Chart	14 days	**Mon 1/13/20**	Sun 1/26/20	
3		**Implement Project Design**	**96 days**	**Mon 1/13/20**	**Fri 4/17/20**	
4		**Hardware Implementation**	**49 days**	**Mon 1/13/20**	**Sun 3/1/20**	
5		Finalize PCB Design	8 days	Sun 1/26/20	Sun 2/2/20	Grant
6		Finalize Re-racking gantry design	8 days	Sun 2/2/20	Sun 2/9/20	Grant
7		Finish construction of re-racking gantry	8 days	Sun 2/9/20	Sun 2/16/20	Grant
8		Finish assembly of PCBs	8 days	Sun 2/16/20	Sun 2/23/20	Grant
9		***MIDTERM: Demonstrate Hardware***	6 days	Sun 2/23/20	Fri 2/28/20	Grant
10		Correct any issues from PCBs	8 days	Fri 2/28/20	Fri 3/6/20	Grant
11		Finish and test solenoid/limit switch code	8 days	Sun 3/8/20	Sun 3/15/20	Grant
12		Finish and test stepper code	8 days	Sun 3/15/20	Sun 3/22/20	Grant
13		Finish and test embedded code	8 days	Sun 3/22/20	Sun 3/29/20	Grant
14		Test and debug table	8 days	Sun 3/29/20	Sun 4/5/20	Grant
15		Test and debug table	8 days	Sun 4/5/20	Sun 4/12/20	Grant
16		***Project Demonstration and Presentation***	0 days	Mon 4/20/20	Mon 4/20/20	
17		Finish mounting PVC track	7 days	Mon 1/27/20	Sun 2/2/20	Kyle
18		Finalize design of pocket gates	8 days	Sun 2/2/20	Sun 2/9/20	Kyle
19		Finish construction of pocket gates	8 days	Sun 2/9/20	Sun 2/16/20	Kyle
20		Finish mounting of solenoids/limit switches	8 days	Sun 2/16/20	Sun 2/23/20	Kyle
21		***MIDTERM: Demonstrate Hardware***	6 days	Sun 2/23/20	Fri 2/28/20	Kyle
22		Finish mounting gantry system	8 days	Fri 2/28/20	Fri 3/6/20	Kyle
23		Route and cable manage everything under the table	8 days	Sun 3/8/20	Sun 3/15/20	Kyle
24		Finish mechanical issues with tables	8 days	Sun 3/15/20	Sun 3/22/20	Kyle
25		Finish and test servo code	8 days	Sun 3/22/20	Sun 3/29/20	Kyle
26		Finish and test servo code	8 days	Sun 3/29/20	Sun 4/5/20	Kyle
27		Test and debug table	8 days	Sun 4/5/20	Sun 4/12/20	Kyle
28		***Project Demonstration and Presentation***	0 days	Mon 4/20/20	Mon 4/20/20	
29		**Software Implementation**	**49 days**	**Mon 1/13/20**	**Sun 3/1/20**	
30		Splicing the pool table out of each frame and adjust x,y coordinates of each ball to send to Firebase DB	14 days	Mon 1/27/20	Sun 2/9/20	Rodney,David
31		Optimize frequency of data collection in firebase and BitBillards App	8 days	Sun 2/9/20	Sun 2/16/20	Rodney,David
32		Optimize detection algorithm to avoid external influence and separate Solids from Stripes	8 days	Sun 2/16/20	Sun 2/23/20	Rodney,David
33		Remove false positives/negatives from buffer algorithmically	6 days	Sun 2/23/20	Fri 2/28/20	Rodney,David
34		***MIDTERM: Demonstrate Software***	5 days	Mon 2/24/20	Fri 2/28/20	Rodney,David
35		Sync balls with table in real time	8 days	Sun 3/1/20	Sun 3/8/20	Rodney,David
36		Sync balls with table in real time	8 days	Sun 3/8/20	Sun 3/15/20	Rodney,David
37		Sync balls with table in real time	8 days	Sun 3/15/20	Sun 3/22/20	Rodney,David
38		Implementing score tracking on app and with Python application	8 days	Sun 3/22/20	Sun 3/29/20	Rodney,David
39		Test and debug table	8 days	Sun 3/29/20	Sun 4/5/20	Rodney,David
40		Test and debug table	8 days	Sun 4/5/20	Sun 4/12/20	Rodney,David
41		***Project Demonstration and Presentation***	0 days	Mon 4/20/20	Mon 4/20/20	
42		**Develop Final Report**	**103 days**	**Mon 1/13/20**	**Fri 4/24/20**	
43		Write Final Report	103 days	Mon 1/13/20	Fri 4/24/20	
44		Submit Final Report	0 days	Fri 4/24/20	Fri 4/24/20	
45		Spring Recess	7 days	Mon 3/23/20	Sun 3/29/20	
46		***Project Demonstration and Presentation***	0 days	Mon 4/20/20	Mon 4/20/20	

The goal of the Gantt Chart shown in Figure 7.1.3 was to have all major systems built and implemented by April 5th. This would allow for several weeks of using the table to work out and squash any bugs or issues in gameplay. The image recognition was also planned to be finished before any else, so that the embedded code could be tested without interfering with any image recognition issues. The schedule was fairly aggressive in terms of completion dates so that extra time to fix issues was available.

Figure 7.1.4: Actual Gantt Chart

ID	Task Mode	Task Name	Duration	Start	Finish	Resource Names	% Complete
1		**SDPII Implementation 2020**	**103 days**	**Mon 1/13/20**	**Fri 4/24/20**		60%
2		Revise Gantt Chart	14 days	**Mon 1/13/20**	Sun 1/26/20		0%
3		**Implement Project Design**	**96 days**	**Mon 1/13/20**	**Fri 4/17/20**		63%
4		**Hardware Implementation**	**49 days**	**Mon 1/13/20**	**Sun 3/1/20**		59%
5		Finalize PCB Design	8 days	Sun 1/26/20	Sun 2/2/20	Grant	100%
6		Finalize Re-racking gantry design	8 days	Sun 2/2/20	Sun 2/9/20	Grant	100%
7		Finish construction of re-racking gantry	8 days	Sun 2/9/20	Sun 2/16/20	Grant	100%
8		Finish assembly of PCBs	8 days	Sun 2/16/20	Sun 2/23/20	Grant	100%
9		***MIDTERM: Demonstrate Hardware***	6 days	Sun 2/23/20	Fri 2/28/20	Grant	100%
10		Correct any issues from PCBs	8 days	Fri 2/28/20	Fri 3/6/20	Grant	100%
11		Finish and test solenoid/limit switch code	8 days	Sun 3/8/20	Sun 3/15/20	Grant	85%
12		Finish and test stepper code	8 days	Sun 3/15/20	Sun 3/22/20	Grant	25%
13		Finish and test embedded code	8 days	Sun 3/22/20	Sun 3/29/20	Grant	10%
14		Test and debug table	8 days	Sun 3/29/20	Sun 4/5/20	Grant	0%
15		Test and debug table	8 days	Sun 4/5/20	Sun 4/12/20	Grant	0%
16		***Project Demonstration and Presentation***	0 days	Mon 4/20/20	Mon 4/20/20		0%
17		Finish mounting PVC track	7 days	Mon 1/27/20	Sun 2/2/20	Kyle	85%
18		Finalize design of pocket gates	8 days	Sun 2/2/20	Sun 2/9/20	Kyle	100%
19		Finish construction of pocket gates	8 days	Sun 2/9/20	Sun 2/16/20	Kyle	90%
20		Finish mounting of solenoids/limit switches	8 days	Sun 2/16/20	Sun 2/23/20	Kyle	10%
21		***MIDTERM: Demonstrate Hardware***	6 days	Sun 2/23/20	Fri 2/28/20	Kyle	100%
22		Finish mounting gantry system	8 days	Fri 2/28/20	Fri 3/6/20	Kyle	10%
23		Route and cable manage everything under the table	8 days	Sun 3/8/20	Sun 3/15/20	Kyle	0%
24		Finish mechanical issues with tables	8 days	Sun 3/15/20	Sun 3/22/20	Kyle	50%
25		Finish and test servo code	8 days	Sun 3/22/20	Sun 3/29/20	Kyle	80%
26		Finish and test servo code	8 days	Sun 3/29/20	Sun 4/5/20	Kyle	80%
27		Test and debug table	8 days	Sun 4/5/20	Sun 4/12/20	Kyle	0%
28		***Project Demonstration and Presentation***	0 days	Mon 4/20/20	Mon 4/20/20		0%
29		**Software Implementation**	**49 days**	**Mon 1/13/20**	**Sun 3/1/20**		71%
30		Splicing the pool table out of each frame and adjust x,y coordinates of each ball to send to Firebase DB	14 days	Mon 1/27/20	Sun 2/9/20	Rodney,David	100%
31		Optimize frequency of data collection in firebase and BitBillards App	8 days	Sun 2/9/20	Sun 2/16/20	Rodney,David	100%
32		Optimize detection algorithm to avoid external influence and separate Solids from Stripes	8 days	Sun 2/16/20	Sun 2/23/20	Rodney,David	85%
33		Remove false positives/negatives from buffer algorithmically	6 days	Sun 2/23/20	Fri 2/28/20	Rodney,David	85%
34		***MIDTERM: Demonstrate Software***	5 days	Mon 2/24/20	Fri 2/28/20	Rodney,David	100%
35		Sync balls with table in real time	8 days	Sun 3/1/20	Sun 3/8/20	Rodney,David	100%
36		Sync balls with table in real time	8 days	Sun 3/8/20	Sun 3/15/20	Rodney,David	100%
37		Sync balls with table in real time	8 days	Sun 3/15/20	Sun 3/22/20	Rodney,David	100%
38		Implementing score tracking on app and with Python application	8 days	Sun 3/22/20	Sun 3/29/20	Rodney,David	0%
39		Test and debug table	8 days	Sun 3/29/20	Sun 4/5/20	Rodney,David	0%
40		Test and debug table	8 days	Sun 4/5/20	Sun 4/12/20	Rodney,David	0%
41		***Project Demonstration and Presentation***	0 days	Mon 4/20/20	Mon 4/20/20		0%

Figure 7.1.4 shows what was accomplished before the project was put on hold. For the electrical side of the project, all major systems were completed and built with some minor issues. The gantry construction was finished, but it was never implemented under the table. The PVC track under the table and pocket gates had all been mounted minus the preparation piece. The gantry prep piece of PVC was not constructed as progress was stopped before a method of attaching the solenoids was developed. Small issues in the pocket gates and PVC track were being fixed also. The PCB worked as intended. The embedded code phase had been started but not too much progress was made as classes were moved online right around that time. Most concepts and major parts of the code had been written; however they were not assembled into the final step by step process needed to re-rack the balls. The stepper motors, solenoids, limit switches, and servo motors all functioned separately, they were just never put together besides the sample code created and used for the midterm demo. Functions such as stepper movement to fixed positions were never finished. Small bugs and issues were present in the solenoid and limit switch code. A few servo motors could open and close, the remaining were not configured in the code to operate.

For the software side of the project, the Python application detected the position of the balls, determined when balls were pocketed, and tracked the score of a game in real time. The React Native application displayed live scores and the positions of the balls. At the time, this worked very well for solid colored balls, but struggled to distinguish striped balls. Also, the game was unable to determine whose turn it was at a given point.

8. Team Information

David Milostan (CpE), Archivist

Grant Reinbolt (EE), Team Leader

Kyle Stevenson (EE), Hardware Lead

Rodney Morgan (CpE), Software Lead

9. Conclusions and Recommendations (GR)

Unfortunately, the project was not able to be completed due to COVID-19. This was disappointing to the group as it was believed that the project was on track to be completed and be successful. Lots of time and effort went into this project so it is disappointing to not be able to finish it. When the project was originally proposed to the advisors, it was stated that the project was too mechanical in design. At the time the group disagreed with this statement and went ahead as planned. Looking back on the project, it is now understood why this project was believed to be too mechanical. The first two months of the implementation phase were spent solely on mechanical design and working out mechanical issues such as track mount, gate mounting, pocket mounting, and gantry construction. While this may seem counterintuitive to the purpose of the design project, it is believed this helped expand the ability and skills of the group. It helped expand cross discipline knowledge and skills by having to come up with solutions for non-electrical issues. This encouraged creativity and can be seen in several designs including the gate pockets using rubber bands. From the software side, it is always underestimated how much code or time goes into a program. Most of the time spent on the embedded side was spent trying to get components to be functional, as making a motor move seemed like an impossible task for the three weeks it took to make happen. The image recognition also took a lot more time and effort than expected. It was assumed to have some

degree of difficulty, but that difficulty was much higher than expected. Overall the team dynamic was very good and healthy, all parties worked together very well and would work together in the future. Our recommendation for future students is that just because a project may seem simple, doesn't mean it will be. A simple project is not a bad idea, as it allows for the project to be completed and for the concepts in the project to be fully mastered.

10. References

[1] V. Singh, “An ARM Based Hardware Architecture for Image Processing,” International Journal of Scientific & Engineering Research, Volume 4, Issue 10, pp 1266-1274, Oct. 2013

[2] Chen-Wei Chou, Ming-Chun Tien, Ja-Ling Wu “Billiards wizard: A tutoring system for broadcasting nine-ball billiards videos,” *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 1921-1924, Apr. 2009.

[3] J. Tang, P.K. Wang, “An Auto-scoring billiards system,” In Proc. Eighth International Conference on Machine Learning and Cybernetics, Baoding, 2009, pp. 3305-3309.

[4] J.B. Russell, “Automatic Ball-Racking Device for Billiard-Tables,” U. S. Patent 1,227,833, 14 May., 1917.

[5] Mingzhou Yin, Yue Chen, Kit-Hang Lee, Denny K.C. Fu, Zion Tsz Ho Tse, and Ka-Wai Kwok, “Dynamic Modeling and Characterization of the Core- XyCartesian Motion System,” *2018 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pp. 206-211, Aug. 2018.

[6] Wang, Hao, “Design of an Automatic Door System for an Automated Transit Network Vehicle,” Dec. 2015, pp. 1-105

[7] Bacus, James W. and Bacus, James V., “Billiard Table lighting and Game Play Monitoring” U. S. Patent 9.485,399 B2, 1 Nov. 2016.

[8] W. Li, C. Yen, Y. Lin, S. Tung, and S. Huang, “JustIoT Internet of Things based on the Firebase real-time database,” 2018 IEEE International Conference on Smart Manufacturing, Industrial & Logistics Engineering (SMILE), pp. 43–47, 2018.

[9] Motor Sizing Calculations, ORIENTAL MOTOR USA CORP, accessed 6 October, 2018, https://www.orientalmotor.com/technology/motor-sizing-calculations.html

[10] Texas Instruments, “DRV8825 Stepper Motor Controller IC” DRV8825 datasheet, Apr. 2010 [Revised July. 2014].

[11] Semiconductor Components Industries, “BUZ11 N-Channel Power MOSFET 50V, 30A, 40 mΩ” BUZ11 Datasheet, September 2013 [Revised October 2017]

[12] Semiconductor Components Industries, “2N7000 / 2N7002 / NDS7002A N-Channel Enhancement Mode Field Effect Transistor” 2N7000 Datasheet, 1998 [Revised October 2017]