Self-Correcting Ping Pong Ball Launcher

A Technical Report submitted to the Department of Electrical and Computer Engineering

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Kai Barzdukas

Spring, 2023

Technical Project Team Members

Angus Chang

David Chen

Jake Coughlin

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Harry Powell, Department of Electrical and Computer Engineering

Statement of Work:

Kai Barzdukas

The primary responsibility assigned to me was the embedded software on the microcontroller. This includes the pulse-width modulation (PWM) commands to aim and shoot the balls, the Kalman filter to self-correct miss-shot balls, and dealing with user input from the user interface (UI) with the universal asynchronous receiver-transmitter (UART) communication protocol. These were all written using C-language code in Code Composer Studio (CCS). Relevant libraries include the MSP432 source files and Texas Instruments (TI) driverlib for function declarations or standard functions for use with the TI MSP432P401R microcontroller.

Another responsibility of mine was the assembly and testing of the electronic hardware – these being the printed circuit board (PCB), microcontroller, and pan/tilt servo system. A good portion of my time on this project was spent wiring and debugging various stages and configurations of the PCB, embedded, and launching systems. This includes adding or modifying components on the PCB to get expected signal response, reconfiguring the pin placement at the various stages of development, tweaking PWM or Kalman values to aim and shoot properly, and the complete refactoring of the embedded code during the switch from our initial microcontroller – the TI Simplelink CC3220SF – to the MSP432. As a result of working closely with these sections of the project, I was largely involved with the final testing process – ensuring the launcher was accurate when shooting balls into the target grid.

Angus Chang

My main project area was the UI and object detection aspects, as well as the encompassing system that would combine them and communicate from the laptop to the microcontroller. The UI runs using the Tkinter package in Python and provides target selection capabilities to the user. When a box has been selected, it relays this to the microcontroller which initiates a launch. For object detection, it also uses a Python library called OpenCV to detect where the ping pong ball lands after being launched. It uses a live video feed from a webcam to pinpoint which of the nine boxes has the ball. The combined system connects these sections together into one flow of actions: the user selects a box to target, which is relayed to the microcontroller. After launching, the video portion detects which box the ball landed in, and the result is also relayed to the microcontroller to make aiming adjustments if needed.

Communication between the laptop system and the microcontroller was another area that required some planning. We chose to use the UART protocol and used a USB to serial adapter in order to allow serial communication to originate from the laptop. I was responsible for choosing the adapter part to use, and getting it integrated with our software system and the header board. On the software side, the pySerial library was used to allow serial communication using Python, transmitting out through where the adapter was plugged in. On the hardware side, the adapter had certain pins that needed to be routed correctly, connecting to the correct inputs on the microcontroller.

David Chen

My main role in this team was designing the mechanical enclosure and frame of our device. I started off by designing the frame in which the pan/tilt mount would operate in CAD using SolidWorks. Upon creating the initial CAD design and acquiring parts for the build. I constructed the aluminum frame and the platform for which the mount would rest on. This was done using leftover 80/20 aluminum from previous capstone groups and wood purchased from Lowes. Along with the mechanical frame of the launching side, I also created the design and implementation for the receive side which was a target grid box made entirely of wood. The initial design was developed in CAD via SolidWorks. I was primarily responsible for all building and construction of all designs. These two mechanical pieces directly affected the project's overall look and how the electrical components would be harnessed for the project.

In addition, to the mechanical design of the frame, I also designed the track and launching platform for the push solenoid. Initially, the design was created in CAD and then 3D printed. After going through many versions, the final solenoid track was decided on and implemented onto the final prototype. The track was critical in allowing the ball to be reloaded and for the plunger to return to its primed state efficiently. Once all the designs were set up, the solenoid track was placed as well so that final testing and prototyping could occur.

Along with my main role as the mechanical designer, I also was involved in the selection of electrical parts and components for the PCB. I worked hand in hand with Jake to properly integrate components onto the board to ensure that all parameters matched, and all drivers would work as intended. This also included testing and evaluation of the board to ensure that the hardware worked as intended.

Jake Coughlin

My role was to design the schematics and layout of our PCB. I used a hierarchical design with several subcircuits connected to one power supply providing power to the entire board. The PCB is a header to a microcontroller used to control our overall system, so I had to ensure the design was correctly interfaced between the PCB and the layout of the microcontroller. My design process was to effectively start from the top and work down. I knew the basic requirements to our system was to control two servo motors and one electromagnetic solenoid. These devices had specific input and power requirements. From there, I researched the electronic parts necessary to control these devices, ensured that they were available, and compiled a bill of materials necessary for the PCB. With this design, I formulated a test plan for each section of the design. This plan primarily involved using test points to verify that the correct voltages are observed and that input signals from a signal generator are also observed where they are needed.

I then designed the PCB layout using KiCad. This involved arranging the component footprints in an orientation that could satisfy the design rules and functional requirements of the PCB. I generated the files necessary for manufacturing and submitted them. Once the board arrived, I worked with David, Kai, and Angus to populate the board and test within the system. The testing involved connecting the board to power, testing the power levels at specific points on the board, connecting each of our servo and solenoid devices, and testing their functionality.

Table of Contents

Table of Figures

Table of Tables

Abstract

This project is a user-controlled ping-pong ball launching mechanism which plays against a human opponent. Both sides aim to shoot balls into 1 of 9 target squares with better accuracy than the opponent. The mechanism uses a combination of a pan/tilt mount controlled by servo motors, a kick solenoid, and a computer user interface (UI) as a control mechanism for the in/out (I/O) system that aims at the targets and launches the ping-pong balls. This system is controlled by a MSP432 microcontroller [1] board communicating with a laptop running a python3-based [2] user interface via the Universal Asynchronous Receiver/Transmitter (UART) communication protocol [3]. To automatically provide feedback and adjustments, an image processing module observes the result of the launch. This communicates back to the computer and then to the microcontroller which adjusts programmatically and physically.

Background

From the conditions that cause a baseball to sail over the outfield wall [4], to the long-term evolution of the development of ballistics for combat [5], the study of projectile motion is extensive and important. To continue this study in the context of a simple game, we created an electronic control, launching, and feedback system to play a game wherein the objective is to land the ball into a specified target square.

A similar project that launched racquet balls into targets was developed by Hujae Choi, Woohyun Jung, and Nakwan Kim in 2017 [6]. Another similar project was developed by Kober, Mulling, and Kromer through 2010 to 2013 [7]. This design did not have any error-correction methods, as it was used in a competition-style environment with only one chance to hit targets. Our variation allows the launcher to incorporate small adjustments to the aiming trajectory if a ball misses its target the first time. In 1997, Yamada et al. published a paper detailing the use of a solenoid coil to launch a steel ball [8]. Another paper detailing the launching of ping-pong balls in robotics is described by Acosta et al. in 2004 [9]. Our launching mechanism uses a similar launching mechanism of a solenoid coil with an internal pushing plunger to strike the ball and propel it forward.

The entire design encompasses a full set of hardware and software areas. Using microcontrollers to send movement instructions to the motors as well as the launcher involved our embedded system design knowledge from ECE 3430 “Introduction to Embedded Computer Systems". Our project uses the MSP432 microcontroller and its capability for pulse width modulation (PWM) to set the current flowing to the solenoid as well as control the servo motors. The user input portion involves some UI and software design which draws upon material from CS 3240 “Advanced Software Development Techniques” and CS 2150 “Program and Data Representation”. CS 3240 was important for the use of front-end development such as buttons the user can select in an interface, while CS 2150 was important for being a bridge between high and low-level coding. These aspects were essential for the UI to communicate with the microcontroller.

Finally, our solenoid coil launching mechanism required a circuit board to control and power it, so this needed printed circuit board (PCB) design knowledge from ECE 3750 “ECE Fundamentals III”. and ECE 2660 “ECE Fundamentals II”. Understanding the solenoid push

mechanism and the physics behind it also draws information from ECE 3209 “Electromagnetic Fields”. These courses were necessary for proper PCB design and the understanding of crucial electronic components such as integrated circuits (ICs), transistors, diodes, and power supplies.

Physical Constraints

Design Constraints

The MSP432 Microcontroller is limited to 256kB of flash memory at a speed of 16MHz [10]. This is more than sufficient for our project. The software necessary for this project were written in Integrated Development Environments (IDEs) called Code Composer Studio [11] and PyCharm [12], using Python language [2]. These tools were all free to use.

The only manufacturing limitation relevant to our PCB design was the inability for Advanced Circuits to make non-circular through holes for our student boards. This affects one piece, the DC Barrel Jack [13]. The PCB also used components all components that could be soldered by hand. Another design constraint was the space we had to work in. Given that our project launches ping-pong balls, they could only travel the length of the tables in the NI Lab. There was also the design constraint of part orders occurring at specific times. In order to give enough time for testing, we had to order certain parts on our own.

When discussing design constraints for the mechanical design, most of the constraint was due to the lack of excess 80/20 aluminum present that could be used. Using leftover pieces from previous capstone groups made it so that the design had to be slightly altered due to the limited amount of aluminum frame present. The target boxes, dimensionally, was constraint based on the field of view of the webcam used which was roughly 65 degrees.

Cost Constraints

The cost constraint was a budget of $500. No single item posed a large strain on cost, but many small items together added up. Equipment like Virtual Bench, soldering irons, and other items were available for use in the NI lab for free. The budget means that there was not a lot of room for purchasing many backup items.

Tools Employed

The entirety of the main system UI, including the video capture and serial communication, is coded in the Python programming language. This was chosen for its simplicity and wide selection of compatible libraries which offer the extra functionality needed for the project. For example, object detection came from the OpenCV library [14], and serial communication came from the pySerial library [15]. The software used to compile and run this code was the PyCharm IDE. This allows each component of the system (webcam/object detection, UI, and serial communication) to be coded in separate files, and then imported and called when necessary.

Code for the microcontroller was developed in the C programming language and compiled using the Code Composer Studio software, which acts as both an IDE and the mechanism to flash code onto the microcontroller chip. The MSP432 source code [10] and TI’s driverlib [16] were used in this project for general things such as MSP432-specific variable

declaration for pin initialization or clock-register manipulation. Our original microcontroller, the CC3220SF, used similar tools with the inclusion of TI's CC3220SF source code [17].

For the mechanical design, two specific software tools were used for design and implementation. SolidWorks was used to create the initial design for both the mechanical frame and the target grid box. It was also used to design the solenoid track that was then 3D printed. Cura software was used to slice the STL files of the solenoid track in preparation of it being printed. Various power tools like jigsaw, power drill, and bandsaw were used to construct the wood portions of the project.

For designing schematics and PCB layout, KiCad and its libraries were used to create hierarchical designs and fully implement the layout of the PCB. The actual manufacture of the PCB was done using a company called WWW and the population was done by hand by team members in the NI lab.

Societal Impact Constraints

Environmental Impact

The project did not have a large carbon footprint as no nonrenewable resources were used to power our robot. The plastic ping-pong balls were reused for the entirety of the semester – meaning the project did not produce extraneous amounts of waste outside of the approximately 3 grams of carbon emissions per gram of plastic used in our project [18]. With many of the primary components such as the MSP432 microcontroller board and the 80/20 steel recycled from projects in previous semesters, we believe that this trend of reusing larger-ticket items which have a larger environmental manufacturing cost in coming years will continue. This helps mitigate the environmental impact of around 400 grams of carbon dioxide emission in the production of microcontrollers, for example [19].

The electronic components, including the launching mechanism, were all electrically powered passive elements and motors/servos – not requiring nonrenewable resources such as pressurized gas. Unfortunately, there is no guarantee that the electricity used in this project was clean. With approximately 60% of electric power in the State of Virginia derived from natural gas, it is highly likely that our project has used energy sourced from nonrenewable sources [20].

Sustainability

It was determined that the deliverable was environmentally safe and can be disposed of relatively easily, if not re-used for future projects. For disposal of other parts, electronics abided by the City of Charlottesville's E-recycling policy and other plastic parts were properly recycled or disposed of [21]. Furthermore, as many of the components such as the 80/20 aluminum used in this project were reused from prior years, it is likely that the same components will be recycled in future projects of ECE 4991. The parts with a high environmental impact will be used longer than a single semester or year and thus will prevent the creation of waste stemming from parts manufacturing.

Health and Safety

From a health and safety aspect, our project was relatively safe. Containing no heavy machinery or sharp pieces, there was no danger of users or bystanders getting severely injured by our device. The motors and servos are low-powered devices and did not pose a danger for any persons near the device. The solenoid exerts less than a pound of force meaning direct contact with an extending solenoid will not cause any harm. The solenoid used (McMaster-Carr 12V Push Linear Solenoid [22]) is a UL Recognized Component, which certifies that it meets UL safety standards [23] as a device.

The projectile being launched was a ping pong ball, which has very low safety risks associated with it. Nittaku, a ping pong ball manufacturer, cites only a couple items of interest regarding safety [24]. The balls are flammable and should be kept away from sources of ignition. Although difficult to ingest, medical help should be sought out immediately in that event. Finally, these plastic balls should not be released into the environment and instead be properly disposed of in the trash. These safety constraints are straightforward, and our group was able to adhere to them throughout the project.

Ethical, Social, and Economic Concerns

Aside from physical safety concerns, there were also ethical concerns which led to the scrapping of our original idea for the Nerf dart shooting device, as it contained a model gun which was deemed insensitive given recent events surrounding violence in schools. The new iteration of our project idea featured a solenoid ball launcher instead, which has a neutral image that is not associated with any weaponry. We believe this change has addressed the concerns and presents an ethically sound game-playing robot.

External Considerations

External Standards

In terms of interconnects and industrial harnessing, we abided by the IPC/WHMA-A-620 [25] standard. For board design, our PCB was required to abide by the Institute for Printed Circuits (IPC) and its regulations [26]. Following these standards helped our project stay consistent with industry board practices. Even though the project involved transmitting data, there were no wireless components involved, so wireless communication standards such as 802.11 [27] were out of the scope of considered standards.

Intellectual Property Issues

To determine whether the project is patentable, it was compared to 3 similar US-patented designs. The first of the three patents are a ‘Mechanical projectile and target game’ [28] based on the same idea of a player versus player ping pong ball launching game. In this design, players take turns launching ping pong balls from mounted, aimable launchers. It is a purely mechanical design, with players launching the balls using spring-like apparatuses. There are several claims in this patent – all of which are independent design choices such as the positioning of the goals or launcher. While the general idea of our project was based off the same idea of launching ping pong balls into cups, we believe the idea of playing against a robotically assisted player with

nearly perfect accuracy and the various electronic components make the two projects significantly different from one another regarding the gameplay loop.

The second and third patents are a ‘Table tennis robot’ [29] and ‘Table tennis ball serving device’ [30] which both use a set of spinning wheels to launch a ping pong ball for a player to hit back. These are essentially pitching machines to practice playing ping pong against. Like the ‘Mechanical projectile and target game,’ the claims for both are all independent, only relating to the functionality of the physical design of the system. While similar in the fact that they launch ping pong balls, this and other relevant mechanical ping pong ball launching mechanisms seem to prefer a flywheel system compared to our kick solenoid design to launch the ball. This is likely due to their ability to apply spin and a lesser emphasis on the accuracy of shots.

After looking through the US patents for something more comparable to the project than these designs, we can comfortably say that our design is patentable. Compared to other designs, our launcher has several unique components such as the image processing to provide user feedback which was nonexistent in any other designs. The choice to use a solenoid to launch a ball was also unique. Furthermore, our design fulfills a niche as a perfect opponent which showcases nearly flawless play when playing Beirut (the official name of launching ping pong balls into cups).

Detailed Technical Description of Project

In order to achieve the goal of a self-targeting and self-correcting ping-pong ball launcher, three major electrical systems are necessary on the PCB. They are the servo motor-controlled pan/tilt mount that aims the launching mechanism towards the target, the solenoid-powered launching mechanism itself, and the real-time communication between the computer user interface and the microcontroller. The power, protection, and connections to these major systems are all housed in a Texas Instruments MSP432 microcontroller [1] header board. A full bill of materials for PCB parts can be seen in Appendix A. Included with the electrical systems is an efficient mechanical design that encloses all necessary components safely.

Mechanical Design

The overall mechanical design of the project covers multiple aspects. The first subsystem is the mechanical frame for the launching side. The initial CAD design can be shown on the next page.

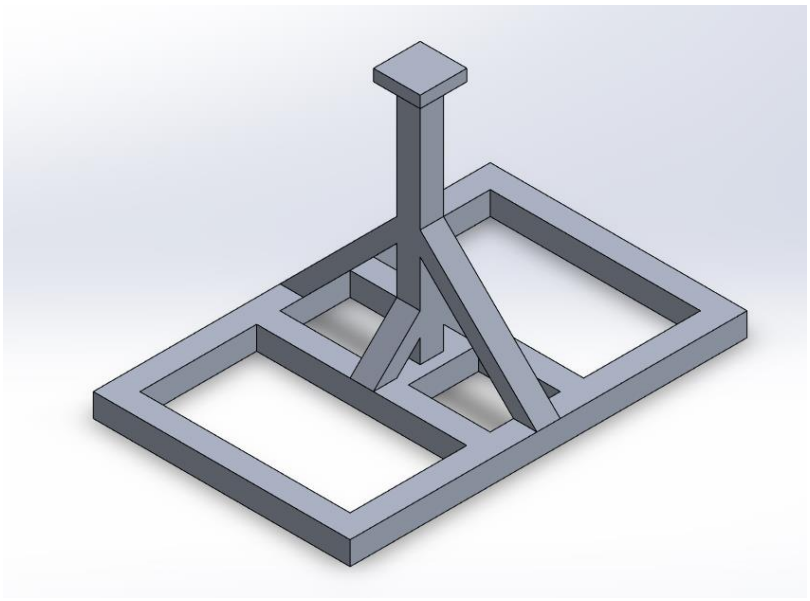


Figure 1: Initial Design Frame

Originally, the frame dimension was 16” x 10” x 10” and consisted predominantly of reused 80/20 aluminum slots. Angled and T-shaped mounting brackets were to be used to fix the pieces of 80/20 together. The top square housing would house both the pan/tilt mount and the solenoid launcher. Initially, angled trusses were designed to create additional support and prevent any components from moving. Once the design was created, construction began and as result the final design is shown below.

Figure 2: Final Design Frame

In the finalized version there are some slight differences. Due to insufficient 80/20, there were no longer angled trusses supporting the frame. Instead, four parallel supports were constructed so that the mounting platform can lay perpendicular to the supports like a square. The initial dimensions also changed drastically due to the size of 80/20 that was available.

The second subsystem for the mechanical design is the receiving target grid box. Figure 3 below shows the initial CAD design for the target grid box.

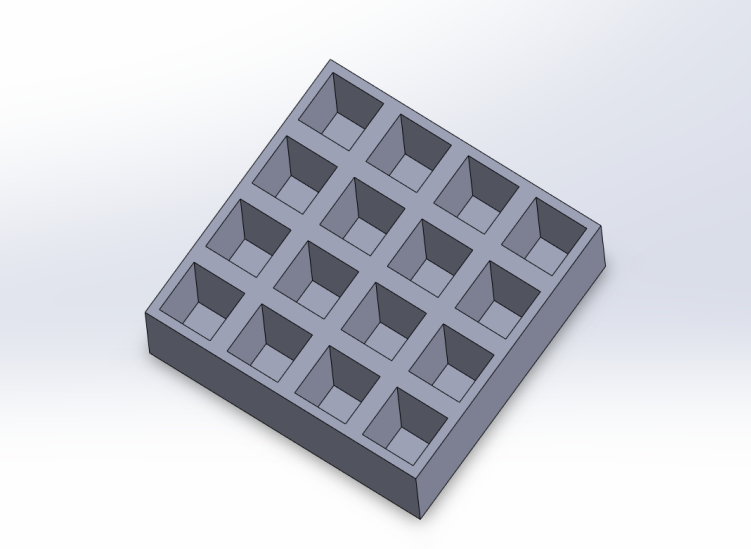


Figure 3: Initial CAD Design for Target Box

Originally, the grid box was to be comprised of a 4” x 4” design with 16 individual target spaces. Due to table constraints in the NI lab, this was changed to a 3” x 3” design that had 9 spaces instead. The dimensions of each target space, however, stayed the same at 3.5” x 3.5”. To stay consist with dimensions, the depth of the box was also similar. The initial plan for the target box was to construct it out of plexiglass and PVC, t, this was later changed to all wood to make it easier to and construct.

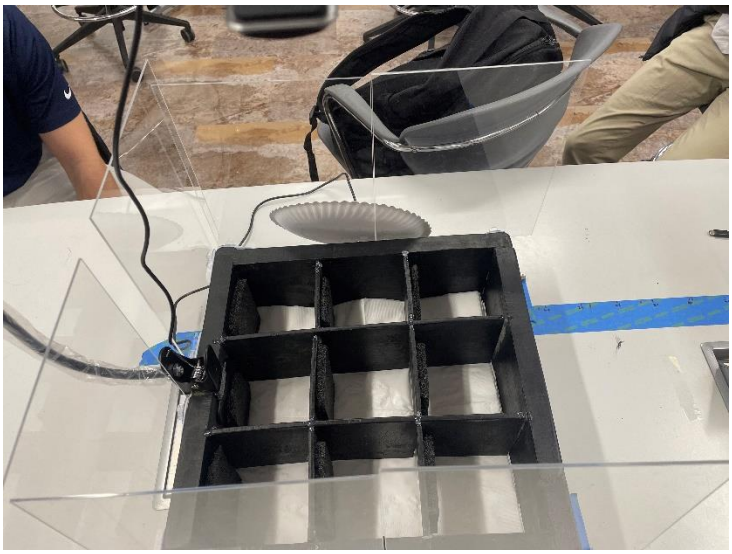


Figure 4: Final Target Box Design

Figure 4 above shows the finalized wooden target box. As discussed earlier, the dimensions stayed the same as the initial design. The plexiglass inserts around the enclosure were added so that the ping pong ball does not go flying in different directions when launched. It can also be seen in the photo that foam inserts were glued on the back side of each target to soften and prevent additional bouncing from the ball when it is launched into the target box. The target box was painted black to create a color gradient so that the image processing could detect the orange ball. Along with that, white padding was placed on the bottom of the box to soften the bouncing of the ball and to create a color gradient for easier detection from the camera.

The third mechanical subsystem that was implemented was the solenoid track. For the ball to be launched properly and effectively, there was a need to fix the solenoid onto the pan/tilt mount as well as have a track for the ping pong ball to sit on. The initial CAD design had a housing for the solenoid as well as a track and was meant to be mounted on the flat side of the pan/tilt mount. Figure 5 on the next page shows a side-by-side of the initial CAD design and the 3D printed part with the intended components on it.

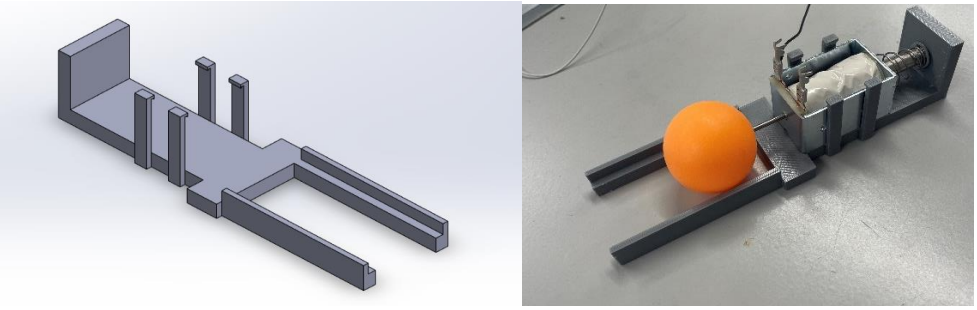


Figure 5: Initial CAD Design and 3D Part of Solenoid Track V1

The reasoning behind the solenoid track design was that it would enclose the solenoid and fix it onto the pan/tilt mount. The design had a back to it so that the plunger would maximize its distance traveled and strike the ball with a full stroke length. There were a couple of issues

with the first version of the track. The first being that there were no through holes to screw on top of the pan and tilt mount. Also, the four supports were not able to fix the solenoid in place during launching. The biggest issue was that the strike location of the ball was too close to the solenoid, meaning that the plunger did not have enough speed which meant that the ball did not travel very far. Versions of the design afterwards improved on these issues and ultimately three versions were created as pictured in Figure 6.

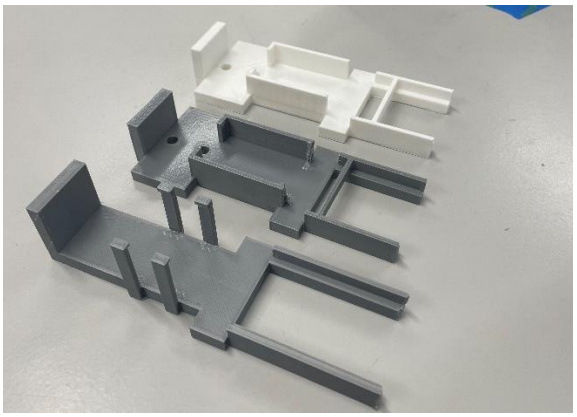


Figure 6: Versions of Solenoid Track

Once the final version was finalized, the track was fixed onto the pan/tilt mount. Figure 7 and Figure 8 on the next page show the finalized mechanical design of the project that includes all components fixed onto it.

Figure 7: Solenoid Track Final Version – Mounted

Figure 8: Fully Harnessed Frame

In Figure 8, the square mount on the top house the pan and tilt mount as expected and the wood piece on the left of the image is where the PCB and Microcontroller are housed. This image is of the frame during demo day and is the final fully harnessed frame.

Servo Motor Driver

The first subsystem housed on the header board will be the servo motor driver. Given that there are two servo motors needed to control the pan/tilt aiming mount, two of these drivers are present on the header board.

To design the servo motor driver circuit, the first consideration is the servo motor that will be used. The motor of choice is the Hitec Commercial Solutions Continuous Rotation DC Motor Servomotor HS-485HB [31]. These motors were chosen as specification of the pan/tilt aiming mount. This mount of choice is the Servocity SPT200H [32]. This mount was chosen for its capability to "pan" and "tilt". These functions create the ability to, at the command of servo motors, rotate vertically and horizontally. Thus, there will be an ability to "aim" the launcher header. The pan/tilt kit specifies: "The SPT200 will accept any standard size servo with a 24 or 25 tooth spline (2 servos are required to assemble the kit). For loads up to 1lb, the HS-485HB or HS-645MG servos work well." Given that the load is the McMaster-Carr 12V Push Linear

Solenoid (its purpose explained in a subsequent section) which weighs 0.355lb [22], the HS-485HB servo motor was chosen due to the quality of it fitting the 1lb recommendation and its availability.

With the selection of this specific servo motor, its necessary peripheral electronic components can be chosen for the header board design. The important characteristics of this device pertinent to the driver header board are noted in Table 1.

Table 1: Servo Motor Characteristics Necessary for PCB Header Design

Operating voltage range	4.8V	6.0V
Standing current	8.0mA	10.0mA
No load running current	180mA	200mA
Stall current	1000mA	12000mA

First, a transistor will be used as a switch between the microcontroller signal and the servo motor. The function of this is such that when the microcontroller is not sending an active signal to the servo motor, it is in an off-state and can't be controlled. The transistor chosen is the Diodes Incorporated ZVN3306A N-Channel MOSFET [33]. It has a gate-source threshold voltage of 0.8-2.4V. This will be sufficient to operate as a switch for the 3.3V control signal from the microcontroller.

Connected to the gate of the transistor is the voltage signal from the microcontroller. Since the motor can operate on 5V as specified in Figure 1, the transistor switch will need to perform the voltage translation to 5V between the microcontroller and the 5V supply. In between the supply and the drain is a pull up resistor with the value of 10kΩ. This value was chosen to ensure that the transistor switch does not float and when turned on is pulled up to 5V. The relationship between the transistor and the 5V power supply is that the transistor is simply a switch for the low voltage Pulse Width Modulation (PWM) input that controls the rotation of the servo motors. This will be further described in the *Pulse Width Modulation Motor Commands* **s**ection. The power supply is from where the servo motors draw the necessary current to operate. The full servo motor driver schematic can be seen in Figure 9.

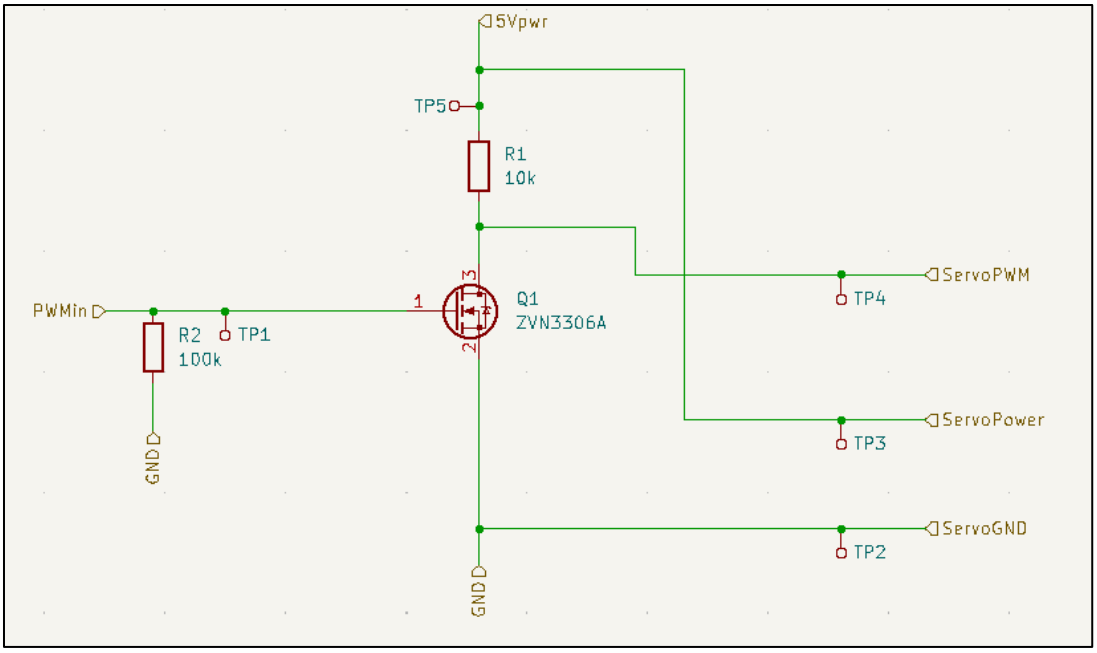


Figure 9: Servo Motor Driver Circuit

From **Error! Reference source not found.**, the "PWMin" port is the input for the pulse width modulation signal from the microcontroller. The length of the pulses from the microcontroller determines the behavior of the servo. As stated in the datasheet, the motor will rotate 40° per one side pulse traveling 400usec [31]. This is sourced from a General-Purpose Input/Output (GPIO) pin on the microcontroller. The specific GPIO pin choice is arbitrary but will be configured with the PWM control setup. The "5Vpwr" input port is the 5V power supply to the servo motor. This is sourced from the PCB power supply that will be described in the *PCB Power Supply* section. Similarly sourced is the GND port.

The three output ports on the driver board in Figure 9 all connect to the servo motor connector wires. The connector wires are for the PWM signal, the Power, and the GND. The connecter from the motor is pictured in Figure 10.

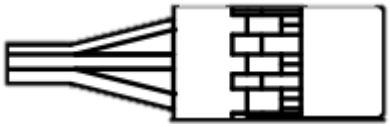


Figure 10: Servo Motor Connector

To connect to the servo connector, the PCB will have a 3-terminal block connecter (Phoenix Contact) [34] that is routed to the three ports: PWM, Power, and GND. This entire driver setup is duplicated with a second driver for the second servomotor. A different GPIO pin from the microcontroller is used for the second driver's PWM input, and a second 3-terminal block connector is used to connect to the second driver.

Solenoid Driver

The second subsystem to be housed on the header board is the solenoid driver circuit. There will be one solenoid in the overall system. As previously mentioned, it is the McMaster-Carr 12V Push Linear Solenoid [22], and its function is to push or "kick" the ping-pong ball into the air towards the target. To design the solenoid driver circuit, its main important characteristics are noted in Table 2.

Table 2: Solenoid Characteristics Relevant to PCB Design

Voltage	12V DC
Electrical connection	Quick-Disconnect Terminals
Current @ retracted stroke	0.67A
Power draw	8W

To drive this device, there are specific chips that already exist for this purpose. The selected chip is the Texas Instruments ULN2003AIN Power Switch/Driver NPN 500mA 16-PDIP [35]. This is a Darlington transistor array with each pair having the capability to output 500mA. The datasheet states that a typical application for this device, "includes motors, solenoids, and relays." In section 9.2, Figure 9-1 of the datasheet, it provides an example of a typical application. This is pictured in Figure 11 below.

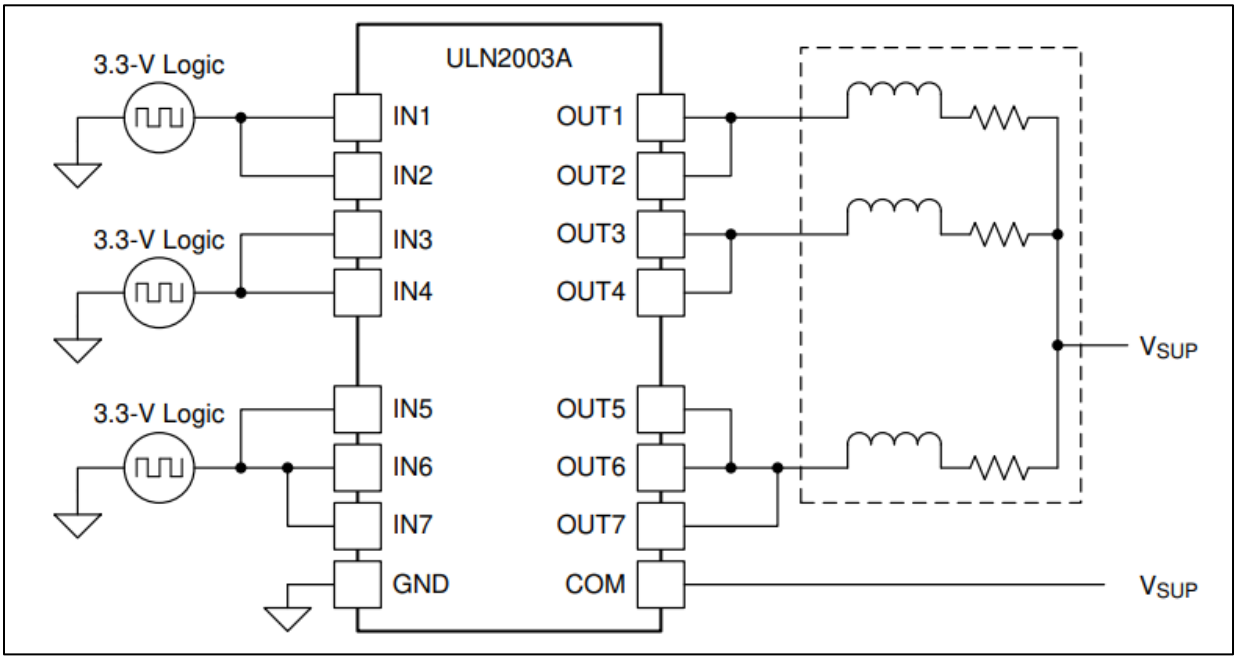


Figure 11: ULN2003 Driver Chip

Their specified application closely resembles the desired application for this system. However, instead of three inductive loads in the example, this system only requires one. The current output is driven by the 3.3V logic input which will be sourced from the microcontroller. A graph of the device current output vs. input duty cycle – % from datasheet Figure 6-5 can be seen in Figure 12 below.

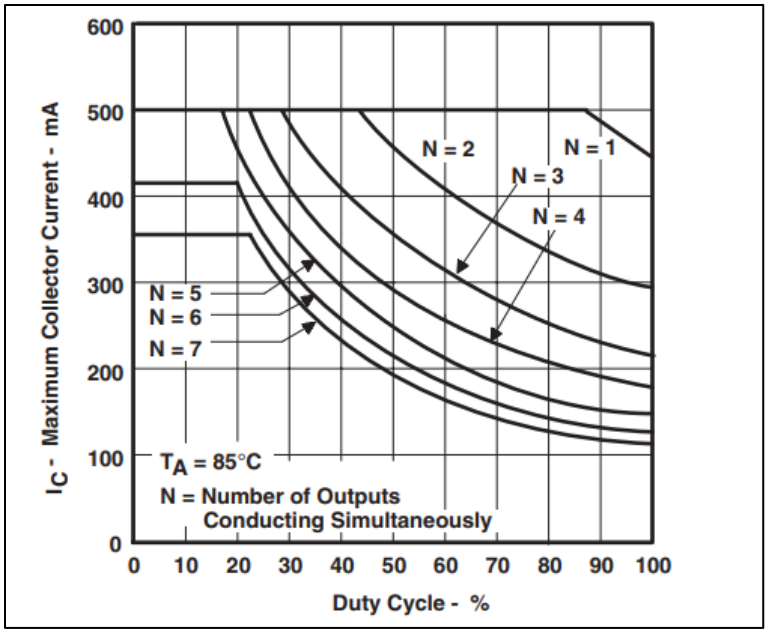


Figure 12: Maximum Collector Current vs. Duty Cycle

By the specification in Table 2, the current at retracted stroke (0.67A), the driver chip needs to provide a current surrounding this value depending on the desired force which is a variable value depending on how far the launch needs to go. Since there is only one inductive load necessary to drive, all of the outputs of the ULN2003AIN can be paralleled. Thus, the curve the chip will operate on will be the N=7 curve in Figure 12. Based on the duty cycle of the input, the amount of current provided to the solenoid can be controlled. With this driver chip there are several important characteristics to ensure proper functionality. By specifications of the solenoid, the chip will need to provide a drive current. This will lead to a level of power dissipation in the chip. The following calculations show the expected drive current and power dissipation. These equations are found in the device datasheet [35].

Drive Current

The coil voltage (V_{SUP}), coil resistance (R_{COIL}), and low-level output voltage ($V_{CE(SAT)}$ or V_{OL}) determine the coil current.

$$I_{COIL} = (V_{SUP} - V_{CE(SAT)})/R_{COIL}$$

$V_{CE(SAT)}$ is determined by I_C. In this calculation I_C is set at ~120mA. From the datasheet, $V_{CE(SAT)}$ = 1V. V_{sup} = 12V. R_{COIL} is not specified, however the datasheet does specify 12V and 8W power draw. From this, R_{COIL} can be calculated as ~18 Ohms. Thus,

$$I_{COIL} = \frac{12 - 1}{18} = 0.61A$$

when I_C = 120mA. This value is determined by Figure 12. Thus, I_{COIL} can be increased or decreased with the duty cycle chosen.

Power Dissipation and Temperature

To calculate ULN2003A device on-chip power dissipation PD:

$$P_D = \sum_{i=1}^{N} V_{OLi} * I_{Li}$$

Where:

- N is the number of channels active together
- V_{OLi} is the OUT_i pin voltage for the load current I_{Li}. This is the same as $V_{CE(SAT)}$

So, for this application, N = 7, V_{Oli} ~= 1V. I_L for each will again be ~100mA at most. So, the on-chip power dissipation is:

$$P_D = \sum_{i=1}^{7} 1V * 0.1A = 0.7W$$

To ensure reliability of ULN2003A device and the system, the on-chip power dissipation must be lower than or equal to the maximum allowable power dissipation (PD(MAX))

$$PD_{MAX} = \frac{T_{J(MAX)} - T_A}{\theta_{JA}}$$

Where:

- TJ(max) is the target maximum junction temperature
- TA is the operating ambient temperature
- RθJA is the package junction to ambient thermal resistance
- θ_{JA} is found to be 67 from the datasheet

Thus,

$$PD_{MAX} = \frac{125 - 25}{67^*} = 1.49W$$

We can confirm here that the power dissipation required by the solenoid will not surpass the maximum allowed power dissipation of the ULN2003A. The solenoid driver circuit can be seen in Figure 13.

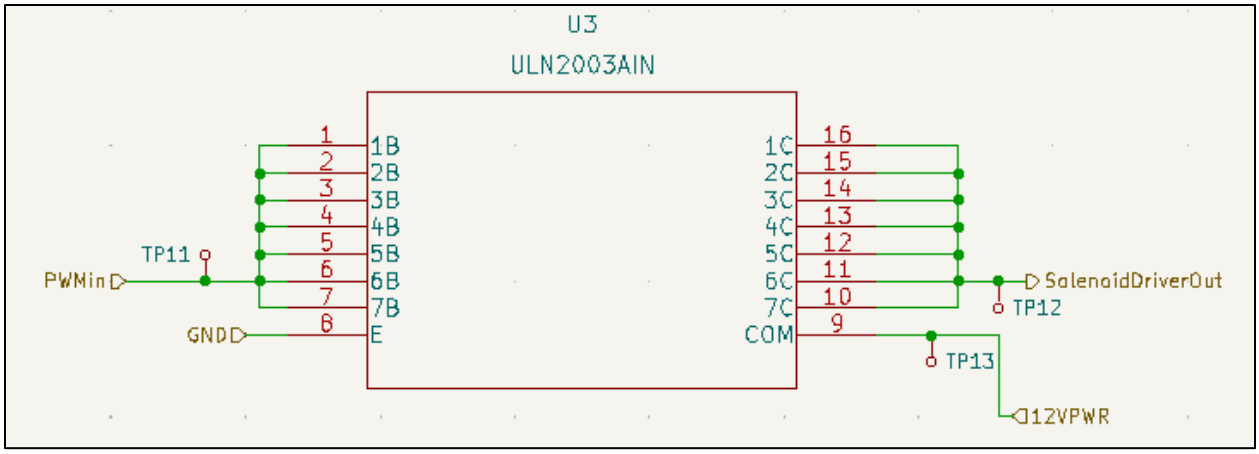


Figure 13: Solenoid Driver Circuit

PCB Power Supply

The power supply is required to provide power to the MSP432 Microcontroller, the two servo motor drivers, the solenoid driver chip, and the solenoid. The MSP432 requires a 3.3V supply, the servo motors require a 5V supply each, the solenoid driver chip requires a 12V supply, and the solenoid itself also requires a 12V supply. The 12V is supplied from a wall adaptor that delivers power through a 12V DC power jack. The power jack of choice is the Adam Tech ADC-028-1-T/R-PA10T [13]. This delivers 12V at 1A. The 12V is run directly to one terminal of the solenoid using another 3-terminal block connector. The 12V is also routed to the ULN2003AIN chip supply pin. The servo motors require 5V with as much as 1A. Therefore, the Mean Well USA Inc. SKM10A-05 DC-DC [36] converter is used to convert 12V to 5V with the ability to deliver 2A. This is sufficient for the servo motors. To supply power to the MSP432, the Microchip Technology MCP1700-3302E/TO Linear Voltage Regulator [37] is used to convert 5V to 3.3V. The power is routed to the 3.3V supply pin of the MSP432. The barrel jack, SKM10A-05, and MCP1700-3302E/TO can be seen in Figure 14, Figure 15, and Figure 16, respectively.

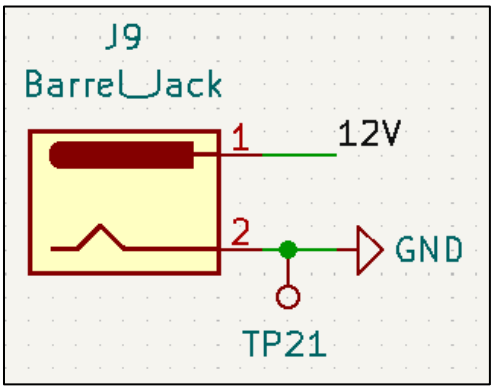


Figure 14: DC Barrel Jack

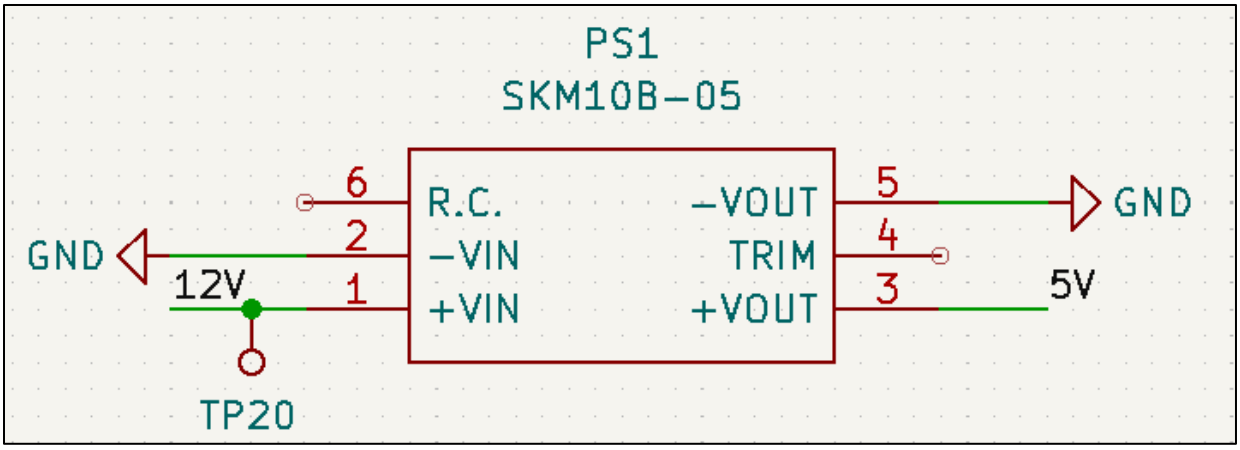


Figure 15: 12V to 5V Converter

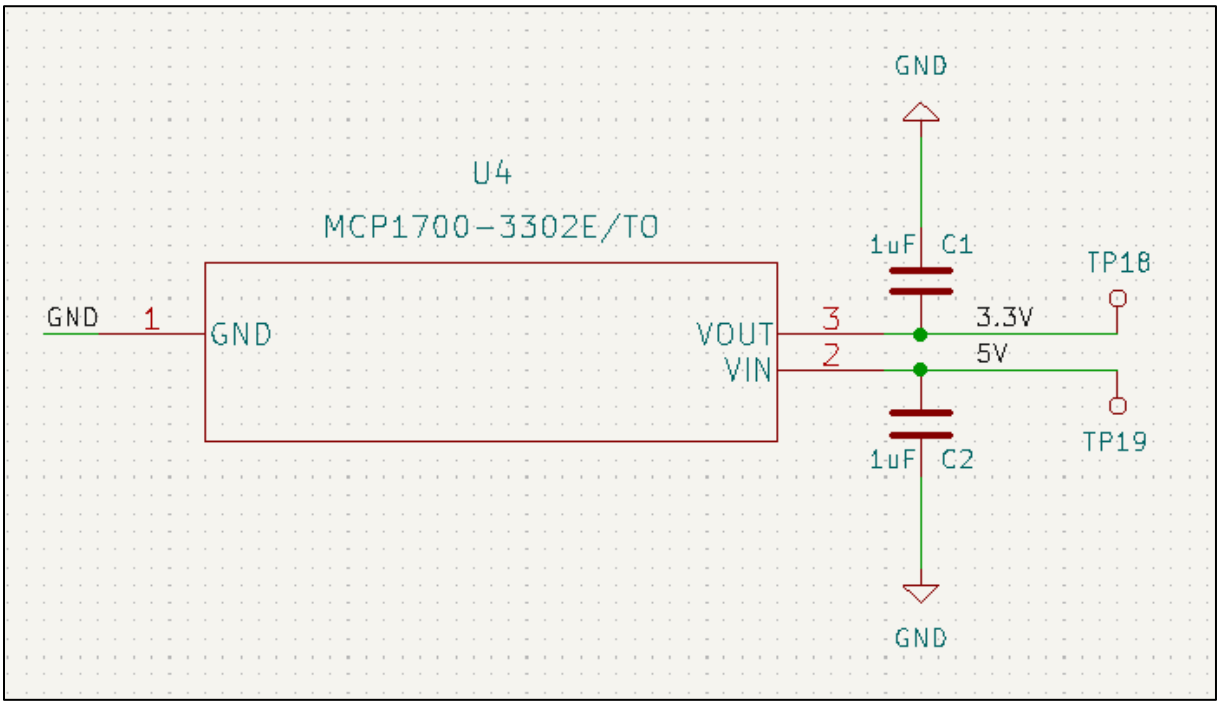


Figure 16: 5V to 3.3V Regulator

Finally, the PCB also includes a 6-pin male connector that is used in the USB to serial communication. This will be described in a subsequent section. The overall layout for the PCB header can be seen in Figure 17. The pin connections from the PCB to the microcontroller can be seen in Figure 18. The RX and TX connections will be described in a future section. D36 through D38 are pulse width modulation outputs.

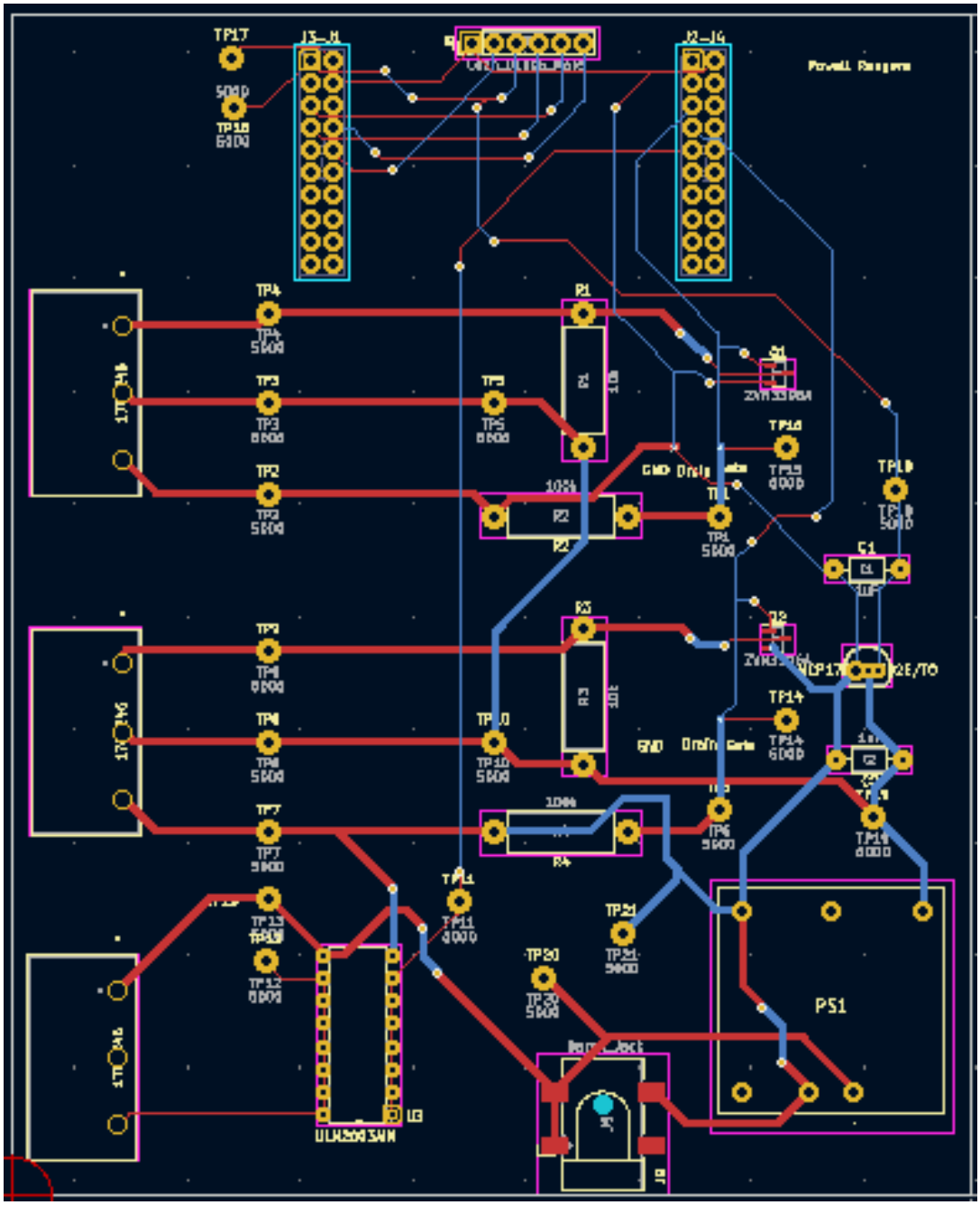


Figure 17: PCB Layout

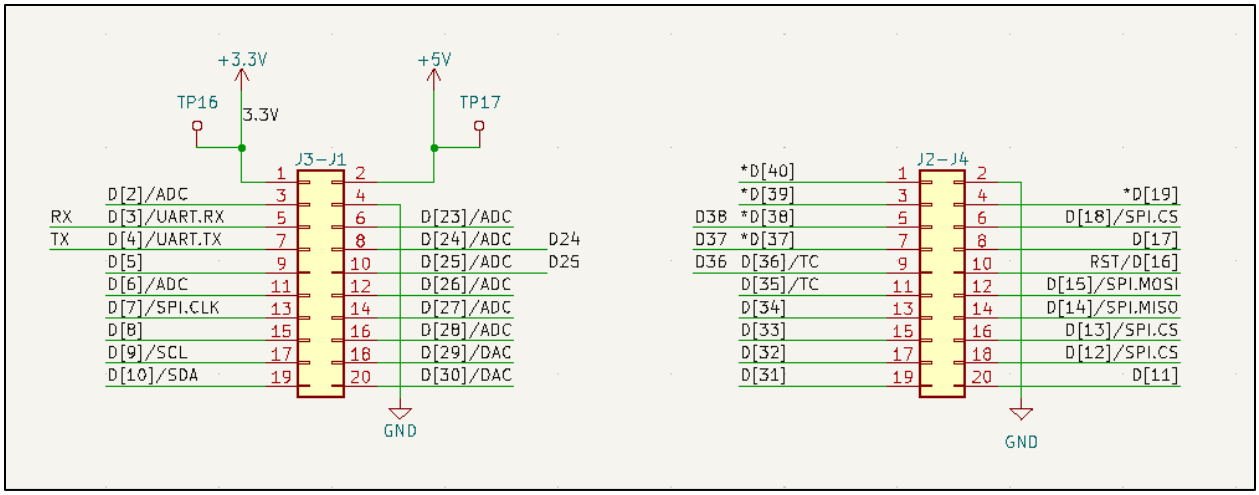


Figure 18: Pin Connections to MSP

Object Detection

The object detection subsystem is how our project determines whether the ping pong ball was successfully launched into the correct box. This starts with the live video feed from a webcam that is mounted to have a bird's-eye view of all nine boxes. Our code reads in frames from the webcam and applies a couple filters to make object detection easier. The first is a gaussian blur, which reduces high frequency noise and makes structural objects more apparent. The second is a conversion from a Red Green Blue (RGB) to a Hue Saturation Value (HSV) color space. This means that color representation changes from values of Red/Green/Blue on a scale of 0-255, to values of Hue (0-179), Saturation (0-255), and Value (0-255).

```
blurred = cv2.GaussianBlur(frame, (11, 11), 0)
hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
```

The conversion is necessary to proceed to the next step, where color boundaries are defined. A lower and upper bound are set in a range around the color of the ping pong ball. The OpenCV library [14] requires these to be defined with HSV values rather than RGB values, hence the color space changes. Using the lower and upper bounds, the program looks for areas of the frame where pixel color falls within the range. Since the target boxes are painted black with a white base, nothing else within the video feed resembles the orange of a ping pong ball. Thus, the only area that is singled out is the ball.

```
yellowLower = (10, 100, 100)
yellowUpper = (30, 255, 255)
```

Finally, the singled-out area is masked, meaning everything except the object of interest in the frame is removed. Now the program can easily detect a circular shape, assuming that the ball is in the frame, and the area is marked out with a circle. The center point of the ball is also tracked, as the value of it will determine which box the ball is in. An example of the video feed is shown in Figure 19 below. Lines are drawn to mark out roughly the shape of each target box.

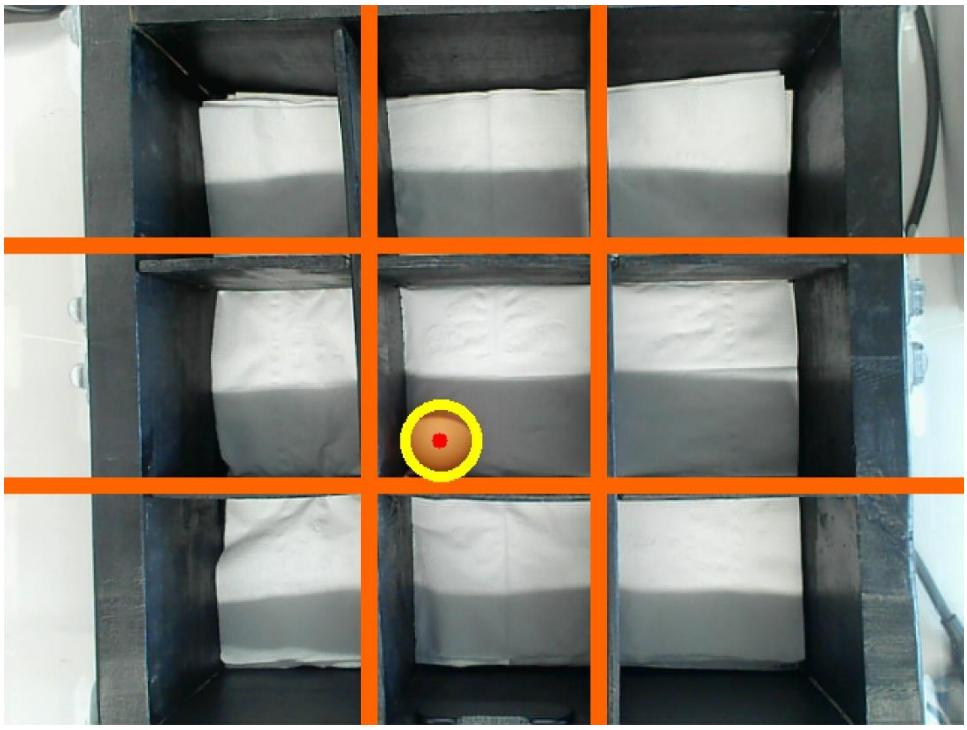


Figure 19: Live Video Feed of Target Boxes

The red dot tracks the center of the ball and has a pair of coordinates associated with it. The value of those coordinates determines which box the ball is in, and that information is saved for future communication with the microcontroller.

User Interface

Although its appearance is simple, behind the scenes the UI acts as the central point of the whole software subsystem. Information about where the ball lands is fed to the UI, and the UI initiates communication with the microcontroller (more detail on the communication method is provided in the section below). There are nine buttons on the interface that when pressed, initiate a launch to that box in the targets. The display is shown in Figure 20 below.

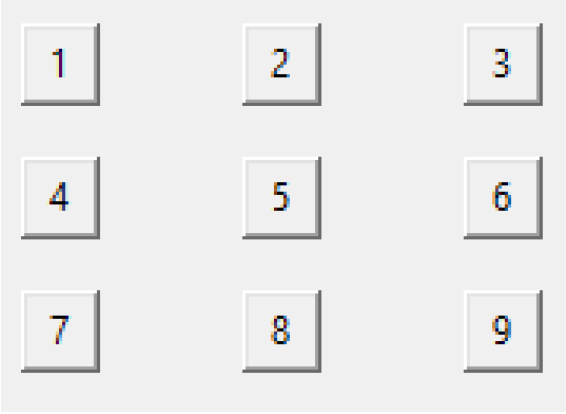


Figure 20: UI for Target Selection

When the user clicks a button, for example, number 4, the system uses serial communication to send the number 4 to the microcontroller. After a time delay allows for the aiming and launching of the ping pong ball, the object detection system is activated. Once the location of the ball is determined, the UI system receives that information and then again communicates it to the microcontroller. If both the initial launch target and the actual landing box are the same, this is regarded as a successful launch. Otherwise, aiming adjustments will have to be made.

USB to Serial Communication

In order to create a more professional connection between the UI and Microcontroller, we chose to use a USB to Serial converter which allows UART communication between the laptop where our system UI is running, and the microcontroller that powers our solenoid and servos. There are several devices which perform this function, ranging from header boards to cables. Both will require connections to the UART TX/RX pins on our MSP432 microcontroller. There is a header board situated on the microcontroller, so that board needed the corresponding pins to be extended above allowing another adapter board to be plugged in on top.

The chosen adapter was a SparkFun Electronics DEV-09873 USB Bridge [38]. Figure 21 below shows the 6-pin setup which allows it to plug into an Arduino device normally. We did not use an Arduino, but the outputs still correspond to pins on the MSP432.

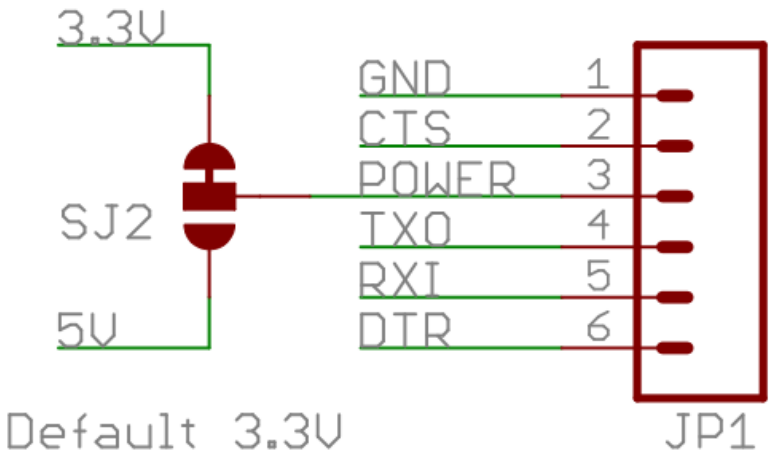


Figure 21: DEV-09873 Header Pins

The header board which contains circuitry for the servo and solenoid also features pins for this adapter board to connect to, which is the TX channel needed for UART communication as well as ground and power. Once connected to the laptop, a program called Docklight can interface with the COM port and communicate with the launchpad. This is as specified in the official MSP432 datasheet published by Texas Instruments [10]; however, in order to integrate serial communication with the UI system, the Pyserial library [15] was called to allow Python code to interface with the COM port and send data to the microcontroller.

To further simplify communication, we kept it as a one-way transfer from the laptop to the microcontroller so that flow control was not needed. This means that only the ground (GND) and TX (TXO) pins needed to be connected. RX was not needed as the laptop would not be receiving any response, and lack of flow control means the CTS and DTR were also unneeded. The aforementioned Pyserial library sends a character that indicates where the user chose to launch, and then after launching it sends another character corresponding to where the ball actually lands. Parameters are available to match the baud rate, byte size, parity, and stop bit settings of the MSP. Our communication has a baud rate of 9600, 8-bit data size, no parity, and one stop bit, as shown below.

```
SerialObj = serial.Serial('COM3')
SerialObj.bytesize = serial.EIGHTBITS
SerialObj.parity = serial.PARITY_NONE
SerialObj.stopbits = serial.STOPBITS_ONE
```

Once this function receives a character from the main UI system, it encodes the character and sends it through the laptop's COM port where the USB to serial adapter is plugged in. The data travels through the USB cable to the adapter, and then exits from the TX output of the adapter and through the header board before it is received at the RX pin of the microcontroller.

```
box_encoded = str(box_num).encode('utf-8')
SerialObj.write(box_encoded)
```

Pulse Width Modulation Motor Commands

Pulse width modulation is the reduction of electrical signals by chopping up a DC signal into periods of on/off phases. By taking the average of the signal over a predetermined time frame, the voltage level will functionally be representative of this average, also known as the duty cycle of the PWM signal. For example, a duty cycle of 50% of a 5V DC signal will effectively be a 2.5V DC signal.

PWM inputs were required to control the servos used in our project. By stepping up or down the duty cycle of the inputs, the servos would turn clockwise or counterclockwise respectively. In order to generate a fine-tunable square wave representative of a PWM signal, early versions of the project incorporating the TI Simplelink CC3220SF microcontroller [17] and the corresponding software development kit (SDK) used a system of PWM instances. Besides the eventual switch to a different hardware scheme, an issue that arose with this method was the lack of modifiability – particularly for pin configuration. The use of macros was another issue – overcomplicating function manipulation and removing abstraction.

```
/* Call driver init functions. */
PWM_init();

PWM_Params_init(&params);
params.dutyUnits = PWM_DUTY_US;
params.dutyValue = angle;
params.periodUnits = PWM_PERIOD_US;
params.periodValue = pwmPeriod;

if(motor == 0)
{
```

```
pwm1 = PWM_open(CONFIG_PWM_0, &params);
if (pwm1 == NULL) {
/* CONFIG_PWM_0 did not open */
while (1);
}

PWM_start(pwm1);

…
```

Following the shift from the CC3220SF to the MSP432, the PWM commands were updated to directly manipulate the timers instead of using prepackaged driver functions. This method of enabling pins on the microcontroller and using the corresponding timer registers is like work done in ECE 3430. For our servos, the period value was set to around 166 Hz (or 3 MHz/18000) as frequencies greater than 10000 Hz or less than 100 Hz would not work with the servos. Higher frequency divisors would also allow for a larger number of steps – allowing the servos to turn more smoothly. Therefore, PWM frequencies closer to the 100 Hz minimum were used in our project. In our final design, we found that duty cycles between 5-10% were ideal in preventing signal clipping from the transistors on the PCB which were more prevalent at higher duty cycles.

```
    //servo 1 (pan servo)
    if(motor == 0)
    {
        //Configure P2.4 as Timer A0.1 (pin 38, 5th bit
for 2.4) The number after decimal can be changed by
doubling/halving hex values
        P2->SEL0 |= 0x10;
        P2->SEL1 &= ~0x10;
        P2->DIR |= 0x10;

        //PWM values: CCR0 is PWM period, CCR1 is initial
duty cycle, do not touch the other two - these are the
clock mode settings
        //WILL NOT MOVE WHEN ATTACHED TO BOARD UNLESS
PERIOD IS LESS THAN 10ms (FREQUENCY > 100Hz)
        TIMER_A0->CCR[0] = 17999;
        TIMER_A0->CCR[1] = 17999 - angle;
        TIMER_A0->CCTL[1] = 0xE0;
        TIMER_A0->CTL = 0x0214;

…
```

While the solenoid used a PWM signal, the best results distance-wise were when the duty cycle was set near 100%. We found that the duty cycle had a direct relationship on the force – as duty cycle was increased, the force exerted on the plunger increased. The solenoid would perform a plunging action by switching between a predetermined duty cycle for the ‘on’ state and an ‘off’ state with a duty cycle of 0. For this reason, the period did not have as large of an impact on the solenoid as it did on the servos.

```
//solenoid
    if(motor == 2)
```

```
{
    P6->SEL0 |= 0x40;
    P6->SEL1 &= ~0x40;
    P6->DIR |= 0x40;

    TIMER_A2->CCR[0] = 24000 - 1;
    TIMER_A2->CCR[3] = angle * 240 - 1;
    TIMER_A2->CCTL[3] = 0xE0;
    TIMER_A2->CTL = 0x0214;

    delayMs(100*time);

    TIMER_A2->CCR[3] = 0;
}
```

Kalman Filter

Although the motor commands were accurate enough to reliably land balls into the target boxes, a self-correction algorithm was implemented in the event a shot managed to land into an incorrect box. To accomplish this, the UART connection from the UI would send a character to the MSP432 microcontroller corresponding to the landing location of the ball after a short delay following the launch of the ball. In most cases, the numerical instruction sent by the UI to the microcontroller would align with the character and no changes would occur within the system. In cases where the instruction was not consistent with the landing location, however, the Kalman filter algorithm would update the set of motor commands corresponding to the aimed location to improve the accuracy of future launches.

The Kalman filter, also known as linear quadratic estimation, is a common and simple estimation algorithm. At its most basic level, the Kalman filter calculates unknown variables in a system using prior measurement data, noise levels in a system, and the current state of the system [39]. While traditional applications include object tracking or image detection, Kalman filters can be as simple as maintaining data values and updating functions on data input. In our project, the motor commands were to be updated on any discrepancy of launch and location.

```
 move(pan2 + panAd*randHor + panAd*values[0][4], 0);

//waits for 1 second
delayMs(delay);

move(92 + tiltAd*randVert + tiltAd*values[1][4], 2);

moveR(pan2 + panAd*randHor + panAd*values[0][4], 0);
 delayMs(delay);
```

By treating the target boxes as a grid of coordinate boxes, the distance between the shot instruction and actual location can be stored in an array. On subsequent shots, this array is accessed to slightly vary the pan, tilt, or solenoid strength to shoot closer to the intended location.

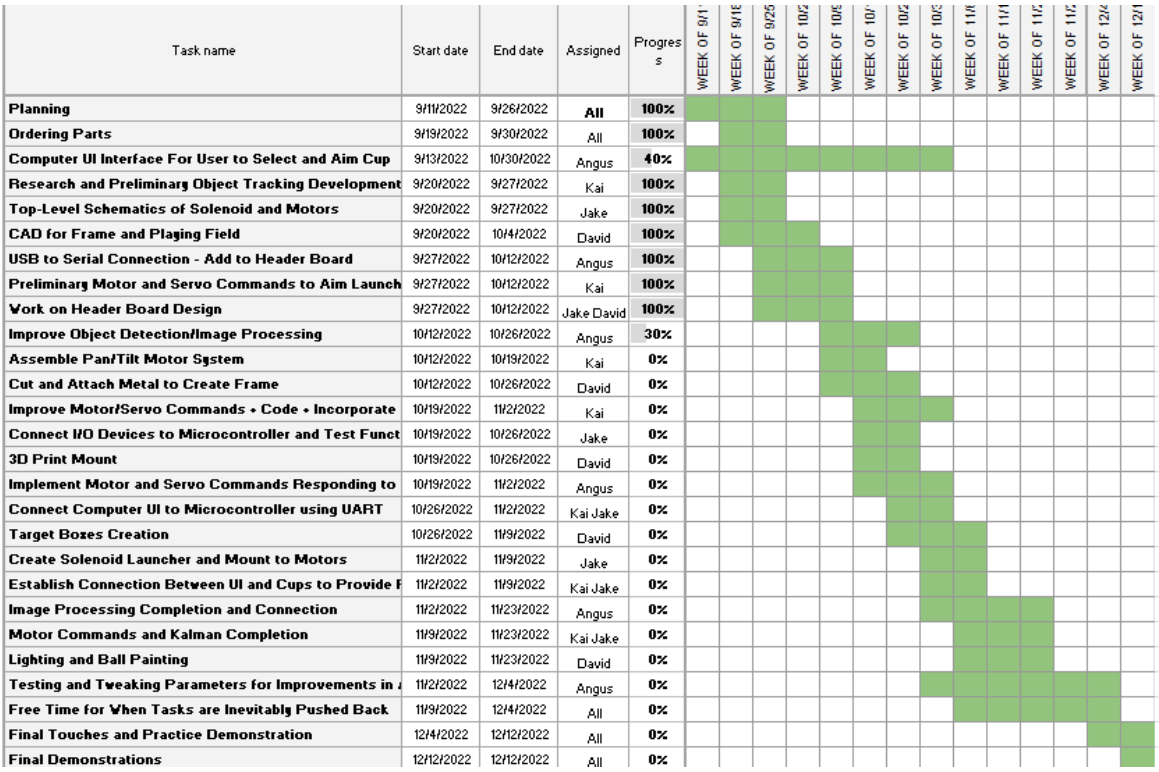
```
int xDif = (((aimed - 1)%3)+1)-xInt;
```

```
    int yDif = (int) floor((aimed - 1)/3)-yInt;

    switch(xDif)
    {
         case -2:
            //printf("Ball landed far right of aim\n");
            col = false;
            //printf("Shifting launcher 2 units
left...\n");
            values[0][aimed-1] -= 2;
            break;

        case -1:
            //printf("Ball landed right of aim\n");
            col = false;
            //printf("Shifting launcher 1 unit
left...\n");
            values[0][aimed-1] -= 1;
            break;

        case 0:
            //printf("Ball landed in same column\n");
            col = true;
            break;

...
```

Project Timeline

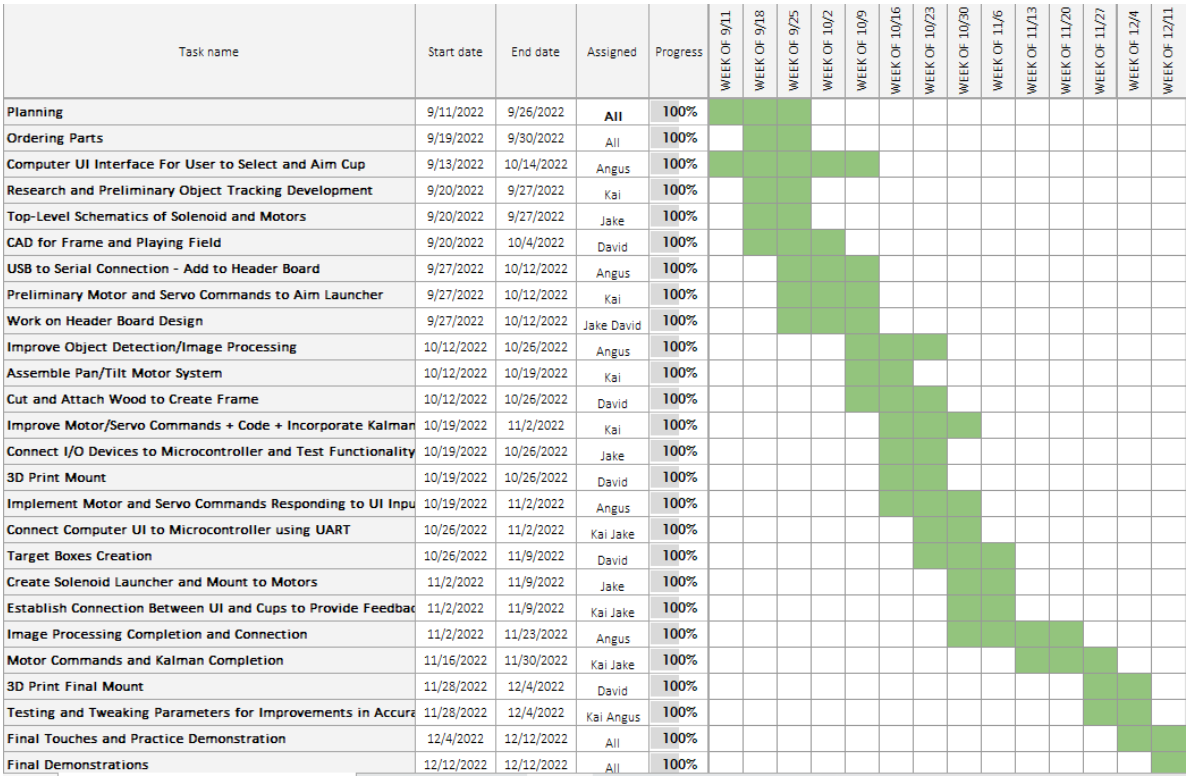
Task name	Start date	End date	Assigned	Progress
Planning	9/11/2022	9/26/2022	All	100%
Ordering Parts	9/19/2022	9/30/2022	All	100%
Computer UI Interface For User to Select and Aim Cup	9/13/2022	10/30/2022	Angus	40%
Research and Preliminary Object Tracking Development	9/20/2022	9/27/2022	Kai	100%
Top-Level Schematics of Solenoid and Motors	9/20/2022	9/27/2022	Jake	100%
CAD for Frame and Plaging Field	9/20/2022	10/4/2022	David	100%
Improved Object Tracking and Detect Object Entry	9/27/2022	10/4/2022	Angus	0%
Preliminary Motor and Servo Commands to Aim Launch	9/27/2022	10/12/2022	Kai	100%
PCB for Solenoid (Servos if Necessary)	9/27/2022	10/12/2022	Jake	100%
Cut and Attach Metal to Create Frame	9/27/2022	10/12/2022	David	0%
Create Cups Containing Photoresistors	10/4/2022	10/12/2022	Angus	0%
Connect Sensors to Computer - Incorporate Object Tra	10/12/2022	10/19/2022	Angus	30%
Assemble Pan/Tilt Motor System	10/12/2022	10/19/2022	Kai	0%
Create Solenoid Launcher and Mount to Motors	10/12/2022	10/19/2022	Jake	0%
PCB for Photoresistors	10/12/2022	10/21/2022	David	0%
Improve Motor/Servo Commands + Code	10/19/2022	10/26/2022	Kai	0%
Connect I/O Devices to Microcontroller and Test Funct	10/19/2022	10/26/2022	Jake	0%
3D Print Mount	10/19/2022	10/26/2022	David	0%
Implement Motor and Servo Commands Responding to	10/19/2022	11/2/2022	Angus	0%
Connect Computer UI to Microcontroller using UART	10/26/2022	11/2/2022	Kai and Jake	0%
Connect Photoresistors to PCB and Microcontroller	10/26/2022	11/9/2022	David	0%
Establish Connection Between UI and Cups to Provide F	11/2/2022	11/9/2022	Kai and Jake	0%
Testing and Tweaking Parameters for Improvements in	11/2/2022	12/4/2022	Angus	0%
Free Time for When Tasks are Inevitably Pushed Back	11/9/2022	12/4/2022	All	0%
Final Touches and Practice Demonstration	12/4/2022	12/12/2022	All	0%
Final Demonstrations	12/12/2022	12/12/2022	All	0%

Figure 22: Original Gantt Chart



Task name	Start date	End date	Assigned	Progress
Planning	9/11/2022	9/26/2022	**All**	**100%**
Ordering Parts	9/19/2022	9/30/2022	All	**100%**
Computer UI Interface For User to Select and Aim Cup	9/13/2022	10/30/2022	Angus	**40%**
Research and Preliminary Object Tracking Development	9/20/2022	9/27/2022	Kai	**100%**
Top-Level Schematics of Solenoid and Motors	9/20/2022	9/27/2022	Jake	**100%**
CAD for Frame and Playing Field	9/20/2022	10/4/2022	David	**100%**
USB to Serial Connection - Add to Header Board	9/27/2022	10/12/2022	Angus	**100%**
Preliminary Motor and Servo Commands to Aim Launch	9/27/2022	10/12/2022	Kai	**100%**
Work on Header Board Design	9/27/2022	10/12/2022	Jake David	**100%**
Improve Object Detection/Image Processing	10/12/2022	10/26/2022	Angus	**30%**
Assemble Pan/Tilt Motor System	10/12/2022	10/19/2022	Kai	**0%**
Cut and Attach Metal to Create Frame	10/12/2022	10/26/2022	David	**0%**
Improve Motor/Servo Commands + Code + Incorporate	10/19/2022	11/2/2022	Kai	**0%**
Connect I/O Devices to Microcontroller and Test Funct	10/19/2022	10/26/2022	Jake	**0%**
3D Print Mount	10/19/2022	10/26/2022	David	**0%**
Implement Motor and Servo Commands Responding to	10/19/2022	11/2/2022	Angus	**0%**
Connect Computer UI to Microcontroller using UART	10/26/2022	11/2/2022	Kai Jake	**0%**
Target Boxes Creation	10/26/2022	11/9/2022	David	**0%**
Create Solenoid Launcher and Mount to Motors	11/2/2022	11/9/2022	Jake	**0%**
Establish Connection Between UI and Cups to Provide F	11/2/2022	11/9/2022	Kai Jake	**0%**
Image Processing Completion and Connection	11/2/2022	11/23/2022	Angus	**0%**
Motor Commands and Kalman Completion	11/9/2022	11/23/2022	Kai Jake	**0%**
Lighting and Ball Painting	11/9/2022	11/23/2022	David	**0%**
Testing and Tweaking Parameters for Improvements in	11/2/2022	12/4/2022	Angus	**0%**
Free Time for When Tasks are Inevitably Pushed Back	11/9/2022	12/4/2022	All	**0%**
Final Touches and Practice Demonstration	12/4/2022	12/12/2022	All	**0%**
Final Demonstrations	12/12/2022	12/12/2022	All	**0%**

Figure 23: Midterm Review Gantt Chart



Task name	Start date	End date	Assigned	Progress
Planning	9/11/2022	9/26/2022	**All**	100%
Ordering Parts	9/19/2022	9/30/2022	All	100%
Computer UI Interface For User to Select and Aim Cup	9/13/2022	10/14/2022	Angus	100%
Research and Preliminary Object Tracking Development	9/20/2022	9/27/2022	Kai	100%
Top-Level Schematics of Solenoid and Motors	9/20/2022	9/27/2022	Jake	100%
CAD for Frame and Playing Field	9/20/2022	10/4/2022	David	100%
USB to Serial Connection - Add to Header Board	9/27/2022	10/12/2022	Angus	100%
Preliminary Motor and Servo Commands to Aim Launcher	9/27/2022	10/12/2022	Kai	100%
Work on Header Board Design	9/27/2022	10/12/2022	Jake David	100%
Improve Object Detection/Image Processing	10/12/2022	10/26/2022	Angus	100%
Assemble Pan/Tilt Motor System	10/12/2022	10/19/2022	Kai	100%
Cut and Attach Wood to Create Frame	10/12/2022	10/26/2022	David	100%
Improve Motor/Servo Commands + Code + Incorporate Kalman	10/19/2022	11/2/2022	Kai	100%
Connect I/O Devices to Microcontroller and Test Functionality	10/19/2022	10/26/2022	Jake	100%
3D Print Mount	10/19/2022	10/26/2022	David	100%
Implement Motor and Servo Commands Responding to UI Inpu	10/19/2022	11/2/2022	Angus	100%
Connect Computer UI to Microcontroller using UART	10/26/2022	11/2/2022	Kai Jake	100%
Target Boxes Creation	10/26/2022	11/9/2022	David	100%
Create Solenoid Launcher and Mount to Motors	11/2/2022	11/9/2022	Jake	100%
Establish Connection Between UI and Cups to Provide Feedbac	11/2/2022	11/9/2022	Kai Jake	100%
Image Processing Completion and Connection	11/2/2022	11/23/2022	Angus	100%
Motor Commands and Kalman Completion	11/16/2022	11/30/2022	Kai Jake	100%
3D Print Final Mount	11/28/2022	12/4/2022	David	100%
Testing and Tweaking Parameters for Improvements in Accura	11/28/2022	12/4/2022	Kai Angus	100%
Final Touches and Practice Demonstration	12/4/2022	12/12/2022	All	100%
Final Demonstrations	12/12/2022	12/12/2022	All	100%

Figure 24: Final Gantt Chart

As seen in all major iterations of the Gantt chart, there were minimal changes to the scheduling of tasks between the beginning of the semester to shortly after the midterm review. This is due to these events occurring before any consideration for rescheduling could occur and many of these tasks such as the ordering of parts, research, or software being doable prior to relevant hardware arriving. The free time allotted for tasks which were serial such as the testing of the complete system or the finalization of component designs can be seen being redistributed in Figure 24, the final Gantt chart. Another noticeable change is the shortening of time allotted for certain tasks and the condensing of tasks to maintain parallelization as best as possible

Generally, the Gantt charts and schedule were designed to keep parallelizable tasks to be done simultaneously and serial tasks that require previous stages of work to be complete to follow the necessary parallelized prerequisites. This can be seen in the various primary tasks such as the PCB design, hardware design, embedded coding, and UI development being followed by the connection of these systems or the assembly of these components. Due to the modularity of the four primary tasks, they tended to be parallel tasks that each member would work on individually to showcase progress at team meetings.

Of the primary tasks mentioned, Jake was responsible for the PCB design, David was responsible for the hardware and mechanical design, Kai was responsible for the embedded coding, and Angus was responsible for the software. Looking at the Gantt charts, one may notice that each team member is delegated to a specific task or two at any point in time. This method of splitting up work naturally reduces serial tasks which can act as blockers for the team and project.

While secondary tasks were not directly outlined, team members generally worked on side tasks tangential to their primary task. Some of the examples include assembly tasks being done by David or motor calibration and testing being performed by Kai. Regarding the dates discussed in class, the overall schedule was designed to have tasks start and end on team meeting dates – Tuesdays for our team. Of course, many tasks had hard deadlines for major events such as the PCB design being finalized by board send out dates, testing and calibration being the final task before the final demonstration, and major demo-able subsystems being completed near the midterm review.

Test Plan

PCB Test Plan

For the header board, there are test points placed in numerous locations. For the servo drivers, they are placed on the gate of the transistor as well as the servo power, PWM, and GND. For the solenoid driver, there are test points on the microcontroller input, the 12V supply, and the output port. At the highest-level, there are also test points on the microcontroller signal outputs, power supplies, and GND pins.

In order to ensure correct functionality, specific voltage levels must be observed at specific locations on the board. The chart on the next page lists where DC voltage levels should be observed.

Table 3: PCB DC Voltage Test Points

Location	Expected DC Voltage
Barrel jack in +	12V
Barrel jack in -	0V (GND)
12V to 5V DC/DC converter in +	12V
12V to 5V DC/DC converter in -	0V (GND)
12V to 5V DC/DC converter out +	5V
12V to 5V DC/DC converter out -	0V (GND)
5V to 3.3V Regulator in	5V
5V to 3.3V Regulator out	3.3V
5V to 3.3V Regulator GND	0V (GND)
MSP432 3.3V pin	3.3V
Servo Driver 1 5Vpwr	5V
Servo Driver 1 GND	0V (GND)
Servo Driver 2 5Vpwr	5V
Servo Driver 2 GND	0V (GND)
Solenoid Driver 12Vpwr	12V
Solenoid Driver GND	0V (GND)

These values were experimentally observed with one exception. We misidentified where the 12V barrel jack ground pin and 12V pins were. This created a -12V signal on the board. The solution to this was to simply reorient the barrel jack to be in accordance with its geometry as opposed to our expected board layout.

Solenoid Test Plan

The solenoid is a two-terminal device. One terminal is connected to the 12V supply, and one is connected to the output of the driver chip. To test it, these connections were made. The goal was ultimately to maximize the distance that the ping-pong ball traveled after being struck by the solenoid plunger. Given that the current output from the driver chip is determined by input duty cycle, the input to the chip was first tested to see if it was receiving a PWM input signal. Then, this signal was varied, and the solenoid's behavior observed. It was determined that simply a 100% duty cycle input created the strongest force in the extension of the solenoid plunger. Additionally, the solenoid was packaged with two plungers. The lighter of the two was selected because it was observed to propel forward with more speed since it was lighter. Finally, the solenoid also included a return spring. This spring brought the solenoid plunger back to a starting position after extending. However, it was also observed that the return spring slowed down the extension velocity. Thus, it was removed from the device.

Servo Motor Test Plan

To test the servo motors, their behavior was observed experimentally while testing according to the datasheet. The datasheet specified that the motor will rotate 40° per one side pulse traveling 400usec [31]. With the pulse width modulation code configured in Code Composer Studio, functions were written to vary this PWM in a controlled way and input to the servo motors. After verifying that the PWM signal was being correctly provided to the servo

motor inputs on the PCB driver board, the remaining testing was trial and error. With the small added complication that the transistor in our servo motor drivers inverted the PWM signal, we tweaked the code and observed the servo motor behavior. Eventually, we settled on the optimum commands to turn the motors in the way we desired.

Overall System Test Plan

Upon completion of the individualized test plans form the PCB, servos, and solenoid, we underwent full integration testing. The PCB was mounted onto the MSP432 and the UART module connected to the board. Once communication between the UI and MSP was established, testing of movement and launching commenced.

Part of full system testing was implementing the required calibration for the device so that all launches would accurately land into the correct target box. Calibration began with creating divisions in the three vertical columns and finding the respective rotational angle required. The aiming from an X-Y standpoint was done through inspection, seeing specifically where the track was aimed based on the intended box designated from the UI. The accuracy of column aiming was strictly based on the PWM signal from the servos. The accuracy of row aiming was determined based on the varying current from PWM to the solenoid. Changes in the PWM to the solenoid would affect the strike force, this meant that consistently calibrating the row accuracy required legitimate launches. During calibration, it was found that there were instances where the servo mount would not return to its resting position. The original design had the resting position of the launcher aimed at the left side. This was changed so that the resting position was centered on the middle column. This was done partially to alleviate the mount from moving too much but more importantly to reduce the amount of angle required for the servos to move to aim properly. During testing, it was found that the PWM waveform clipped at the sides and thus was causing some inaccuracies when aiming at the rightmost column. It was deduced that the transistor was causing the clipping, but we did not remove it to protect it from overcurrent on the board. It was decided that despite the inconsistencies, if the ball was relaunched into the same box, it would always hit that requested target.

Once calibration finished further system testing was done on the self-correction algorithm. In order to test the Kalman filter, a controlled experiment was done where a random target was selected on the grid and launched at. The intention was for the ball to launch in the middle target box and when relaunching at the target box it would recalibrate based on given information from image processing and launch into the correct box. Doing this system test validated the software design of the Kalman filter. Initially, the Kalman was implemented as a part of the main method when running through the servo commands, to isolate a Kalman demonstration this was removed and a Kalman processes was created through code that ran the function when the Kalman experiment button was pressed on the UI side.

Final Results

The final iteration of the project meets the main criteria for accuracy that was initially set out - for the launcher to beat out a human player. This means that given a fixed set of shots where the target box must be called beforehand, the launcher should successfully make more

shots than the human player, but not necessarily have 100% accuracy. A few test trials were performed to confirm this, and the results are shown in Table 4 below.

Table 4: Launcher vs. Human Player Trials

Total shots taken	Shots made by launcher	Shots made by player
9	8	4
9	9	3
10	9	4
15	13	6

Since the launcher is consistently more accurate than a human player, the accuracy requirement is met. Other aspects of the project also have grading standards that were set out in the proposal, and these are detailed in

Table 5 below. As evidenced by its ability to beat a human player, the launcher does have a fully functional launching mechanism and pan/tilt mount. The solenoid is activated when a launch command is issued and successfully propels the ball toward the target boxes. The pan/tilt mount swivels and aims when a launch command is issued and aims the ball in the right direction for launch.

On the software side, the UI and camera feed are both fully functional. The UI allows the user to specify a target box for the launcher to aim at and can issue a launch command to the microcontroller when a choice is made. Video from the webcam is also integrated into the system, as it feeds into the object detection portion which determines the location of the ball and relays this to the microcontroller.

Lastly, the launcher can self-correct its aiming if the ball misses and lands in a different target box than intended. For example, if the user selects box five to aim at and the video feed shows that it landed in box 7, the microcontroller can perform an adjustment to aim farther right and provide more power to the solenoid. The next shot that aims for box five will successfully land in the correct box. Each subsystem is functioning as intended and meets the criteria listed for a letter grade of A for the project.

Table 5: Letter Grade Criteria

Letter Grade	Criteria
A	• Fully functional launching mechanism • Fully functional pan/tilt mount • Fully functional UI and integration with camera feed • User specified accuracy, i.e., targeting a specific box • Self-correction of the launch trajectory if a ball misses due to unforeseen variables
B	• Fully functional throwing mechanism • Fully functional pan/tilt mount • Limited UI functionality with only target selection • Some level of user specified accuracy, i.e., targeting a cluster of boxes
C	• Fully functional throwing mechanism • Fully functional pan/tilt mount • Bare minimum UI with only launch activation • Bare minimum accuracy (not user specified).
D and under	• No Functional Prototype • Unable to fulfill any of the aforementioned criteria

Costs

As shown in Appendix B, the total cost of the project was roughly $474.82. A large part of the budget was purchasing the necessary power tools to build the necessary mechanical features. Assuming that a professional manufacturer would not need to purchase additional tools, this reduces the overall material cost to roughly $400. However, due to the reuse of 80/20 aluminum, mounting brackets, and microcontrollers from past capstone projects, this would increase the overall manufacturing price back to around $470. When looking at DigiKey and other distribution companies that we used, purchasing components for over 10000-unit quantities drastically decreases the overall cost of our project. Overall, there was roughly a 60% decrease in the price of components. If we were to manufacture our product in 10000-unit quantities, it would only cost $188. This price would further decrease by using automated equipment when manufacturing. All 3D printed parts would be cheaper when using automated equipment instead of 3D printers. Also included in our detailed budget in Appendix B there was a plethora of backup components and items that were purchased just in case of an emergency. By optimizing how much of each component we use, we can further drive the price down when considering adding high volume manufacturing for our project.

Future Work

One of the biggest limitations to the system's self-correction was the design of the target boxes. Ping-pong balls could easily bounce off walls of the box and fly completely off course from where they would have landed had the targets been on a flat surface. Because of this, the system could only self-correct if the ball landed properly inside one of the boxes. If future iterations of this project are worked on, a flat substitute for the target boxes should be developed. This could use some sort of extremely adhesive material to ensure that the ball stays exactly where it first contacts the target. This allows the system to catch that location and perform a more robust self-correction. The ball bouncing off the target grid made it so that the image processing could not detect where it landed first, by using a flat target system with a way to stick the ping pong balls onto the surface we can implement the Kalman filter to always have micro adjustments.

Another unforeseen difficulty was ensuring that the target boxes stayed in the right location relative to the launcher. It was known that this distance needed to stay constant, but in practice the system was much more sensitive to small movements than anticipated. For example, bumping the table could shift the targets slightly and cause some shots to become uncalibrated and miss their boxes. A more solid way of securing these two parts at a fixed distance would reduce the amount of extra time spent recalibrating the launcher during each new work session.

An additional shortcoming of our project is the launch distance. Initially, based on the datasheet provided by McMaster-Carr we believed that the push solenoid would exert enough force to launch the ball comfortably across the table. As we later tested the push solenoid was only able to launch the ball around ¾ of the table’s length. This is in part due to the force at a desired stroke length. One thing to reconsider is the push solenoid that was decided and to potentially find one that has a larger force. The reason we were unable to change solenoid is to do the power management done on the PCB. Because the solenoid proved to be an extremely consistent launching mechanism, the device could be greatly improved if it were more powerful and launched a further distance.

References

[1] "MSP430G2553," Texas Instruments, [Online]. Available: http://www.ti.com/product/MSP430G2553. [Accessed 01 December 2022].

[2] "Python," Python Software Foundation, [Online]. Available: https://www.python.org/. [Accessed 24 September 2022].

[3] E. Peňa and M. G. Legaspi, "UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter," *AnalogDialogue,* vol. 54, no. 4, December 2020.

[4] R. K. Adair, "The Physics of Baseball," *Physics Today,* vol. 48, no. 5, p. 26, 1995.

[5] M. Denny, Their Arrows Will Darken the Sun: The Evolution and Science of Ballistics, Johns Hopkins University Press, 2011.

[6] H. Choi, W. Jung and N. Kim, "Development of autonomous ball launcher system on surface vehicle for detect and deliver mission in 2016 Maritime RobotX Challenge," in *IEEE Underwater Technology*, Busan, 2017.

[7] J. Kober, K. Mülling, O. Krömer, H. C. Lampert, B. Schölkopf and J. Peters, "Movement templates for learning of hitting and batting," in *IEEE International Conference on Robotics and Automation*, Anchorage, 2010.

[8] H. Yamada, M. Yamaguchi, Y. Gohdo, H. Ota, K. Takeuchi and H. Yamagami, "Continuous electromagnetic launcher using soleniod coil," *IEEE Transactions on Magnetics,* vol. 23, no. 5, pp. 3263-3265, 1987.

[9] L. Acosta, J. Rodrigo, J. Mendez, G. Marichal and M. Sigut, "Ping-pong player prototype," *IEEE Robotics & Automation Magazine,* vol. 10, no. 4, pp. 44-52, 2003.

[10] "MSP432P401R, MSP432P401M Mixed-Signal Microcontrollers," July 2016. [Online]. Available: https://www.ti.com/lit/ds/slas826e/slas826e.pdf. [Accessed 24 September 2022].

[11] "Code Composer Studio™ integrated development environment (IDE)," Texas Instruments, [Online]. Available: https://www.ti.com/tool/CCSTUDIO. [Accessed 24 September 2022].

[12] "PyCharm," JetBrains, [Online]. Available: https://www.jetbrains.com/pycharm/. [Accessed 24 September 2022].

[13] "ADC-028-1-T/R-PA10T," Digi-Key Electronics, [Online]. Available: https://www.digikey.com/en/products/detail/adam-tech/ADC-028-1-T-R-PA10T/9832093. [Accessed 25 September 2022].

[14] "OpenCV: Open Source Computer Vision," OpenCV, [Online]. Available: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html. [Accessed 10 October 2022].

[15] C. Liechti, "pySerial," pySerial, [Online]. Available: https://pyserial.readthedocs.io/en/latest/. [Accessed 1 November 2022].

[16] "MSP Driver Library," Texas Instruments, [Online]. Available: https://www.ti.com/tool/MSPDRIVERLIB. [Accessed 10 October 2022].

[17] "CC3220 SimpleLink™ Wi-Fi® LaunchPad™ Development Kit Hardware," March 2020. [Online]. Available: https://www.ti.com/lit/ug/swru463c/swru463c.pdf?ts=1670845285702. [Accessed 10 October 2022].

[18] ICF International, "Documentation for Greenhouse Gas Emission and Energy Factors Used in the Waste Reduction Model (WARM)," U.S. Environmental Protection Agency, 2016.

[19] "Footprint of a Microcontroller," STMicroelectronics, [Online]. Available: https://www.st.com/content/st_com/en/about/st_approach_to_sustainability/sustainability-priorities/sustainable-technology/eco-design/footprint-of-a-microcontroller.html. [Accessed 26 September 2022].

[20] "U.S ENERGY ATLAS WITH TOTAL ENERGY LAYERS," U.S. Energy Information Administration, [Online]. Available: https://www.eia.gov/state/?sid=VA. [Accessed 26 September 2022].

[21] "Recycling," City of Charlottesville, [Online]. Available: https://charlottesville.gov/397/Recycling. [Accessed 26 September 2022].

[22] "McMaster-Carr Linear Solenoid Continuous, Push, DC Volts, 0.5" Stroke, 15 oz. Force," [Online]. Available: https://www.mcmaster.com/70155k121/.

[23] "NDERSTANDING UL STANDARDS FOR INDUSTRIAL ELECTRICAL CONTROLS," [Online]. Available: https://www.c3controls.com/white-paper/understanding-ul-standards-for-industrial-electrical-controls/ .

[24] "Nittaku," [Online]. Available: https://www.nittaku.com/wp-content/uploads/2019/11/152.pdf. [Accessed 26 September 2022].

[25] "IPC/WHMA-A-620," Wiring Harness Manufacturer's Association, [Online]. Available: https://whma.org/ipcwhma-a-620/. [Accessed 24 September 2022].

[26] "IPC Standards," IPC, [Online]. Available: https://www.ipc.org/ipc-standards. [Accessed 24 September 2022].

[27] "IEEE 802.11-2016," IEEE Standards Association, [Online]. Available: https://standards.ieee.org/ieee/802.11/5536/. [Accessed 26 September 2022].

[28] D. G. Dahl and S. D. Buss, "Mechanical projectile and target game". United States of America Patent US20150137452A1, 21 May 2015.

[29] G. Berliner, "Table tennis robot". Canada Patent CA1047341A, 30 January 1979.

[30] G. Gatchel, J. E. Newgarden and G. E. Lynn, "Table tennis ball serving device". United States of America Patent US4854588A, 8 August 1989.

[31] "HS-485HB General Specification," [Online]. Available: https://media.digikey.com/pdf/Data%20Sheets/Hi-Tech/HS-485HB.pdf.

[32] "Servocity SPT200 Pan & Tilt Kit," [Online]. Available: https://www.servocity.com/spt200-pan-tilt-kit/ .

[33] "Diodes Incorporated ZVN3306A MOSFET N-CH 60V 270MA TO92-3," [Online]. Available: ZVN3306A N-channel enhancement mode vertical DMOS FET datasheet (diodes.com).

[34] "TERM BLOCK HDR 3POS 90DEG 7.62MM," [Online]. Available: https://www.digikey.com/en/products/detail/phoenix-contact/1766246/348813.

[35] "Texas Instruments ULN200x, ULQ200x High-Voltage, High-Current Darlington Transistor Arrays," [Online]. Available: https://www.ti.com/lit/ds/symlink/ulq2004a.pdf?HQS=dis-dk-null-digikeymode-dsf-pf-null-wwe&ts=1663852020165&ref_url=https%253A%252F%252Fwww.ti.com%252Fgeneral%252Fdocs%252Fsuppproductinfo.tsp%253FdistId%253D10%2526gotoUrl%253Dhttps%253A%252F%252Fwww.ti.co.

[36] "DC DC CONVERTER 5V 10W," [Online]. Available: https://www.meanwellusa.com/upload/pdf/SKM10/SKM10,DKM10-spec.pdf.

[37] "IC REG LINEAR 3.3V 250MA TO92-3," [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/MCP1700-Data-Sheet-20001826F.pdf.

[38] "DEV-09873," Digi-Key Electronics, [Online]. Available: https://www.digikey.com/en/products/detail/sparkfun-electronics/DEV-09873/5318746. [Accessed 10 October 2022].

[39] "Kalman Filter Explained Simply," The Kalman Filter, [Online]. Available: https://thekalmanfilter.com/kalman-filter-explained-simply/.. [Accessed 26 September 2022].

Appendix

Appendix A: Bill of Materials for PCB Header

Capstone Team Powell Rangers Microcontroller Header Board Bill of Materials

Rev A

Line	Quantity	Reference	Value	MANU PART#	MANU
1	2	C1, C2	1uF	C315C105K3R5TA	KEMET
2	3	J5,J6,J7	1766246	1766246	Phoenix Contact
3	1	J8	Conn01x06Male	2011-1X06G00SI025B	Oupiin
4	1	J9	BarrelJack	ADC-028-1-T/R-PA10T	Adam Tech
5	1	PS1	SKM10B-05	SKM10A-05	MEAN WELL USA Inc.
6	2	Q1, Q2	ZVN3306A	ZVN3306A	Diodes Incorporated
7	19	TP1-TP19	TP	5000	Keystone Electronics
8	2	R1,R3	10k Ω	Supplied, axial, horizontal	Sourced from FUN kit
9	2	R2,R4	100k Ω	Supplied, axial, horizontal	Sourced from FUN kit
11	1	U3	ULN2003AIN	ULN2003AIN	Texas Instruments
12	1	U4	MCP1700-3302E/TO	MCP1700-3302E/TO	Microchip Technology
13	10	N/A	390088-2-ND	390088-2-ND	TE Connectivity AMP Connectors

Appendix B: Bill of Materials and Costs of Project

Digikey Part #	McMaster Part #	Qty Req'd	Per Unit Price	Cost
	70155K121	1	46.3	46.3
296-16971-5-ND		2	0.94	1.88
296-26053-5-ND		1	5.62	5.62
2589-32645S-ND		2	34.99	69.98
296-CSD18537NKCS-ND		4	1.42	5.68
3046-9V-MN1604-ND		4	2.46	9.84
36-2242-ND		4	1.16	4.64
296-16971-5-ND		2	0.94	1.88
296-26053-5-ND		1	5.62	5.62
2589-32645S-ND		2	34.99	69.98
296-CSD18537NKCS-ND		4	1.42	5.68
3046-9V-MN1604-ND		4	2.46	9.84
36-2242-ND		4	1.16	4.64
A24732CT-ND		4	0.18	0.72
2057-ADC-028-1-T/R-PA10TTR-ND		1	0.81	0.81
277-5943-ND		1	1.62	1.62
277-5830-ND		1	4.82	4.82
36-5000-ND		25	0.42	10.5
2368-02-P4R7-1-ND		10	2.49	24.9
277-5994-ND		2	2.33	4.66
277-5783-ND		2	7.28	14.56
2057-PA-013-ND		1	9.66	9.66
1568-1104-ND		1	16.95	16.95
2011-1X06G00SI025B		1	0.18	0.18
1866-4619-ND		1	22.91	22.91
MCP1700-3302E/TO-ND		4	0.5	2
399-9714-ND		4	0.8	3.2
S6106-ND		2	1.26	2.52
277-5783-ND		2	7.28	14.56
277-5994-ND		2	2.33	4.66
296-35013-ND		4	1.88	7.52
Additional Costs				
Wood and Tools from Lowes				86.49
			Total Cost	474.82