Mechatronic Design Final Report

16-778 / 18-578 / 24-778

Spring 2013

Carnegie Mellon University

Team E

Neil Abcouwer

Ben Wasserman

Qi Wang

Jerry Carlson

Abstract

This report describes Mechatronic Design Team E's design of a robot that identifies, aims at, and fires disks at colored targets. It includes a description of the system, its requirements, and its performance. The design requirements include being able to do these tasks without external processing or user interaction (other than pressing a Start button), and within time and space constraints. The mechanical design was conceptualized around four subsystems (yaw, pitch, hopper, launcher), and the report provides a thorough description of these subsystems. The mechanical design revolved around dispensing disks into and aiming a curved firing track. The description of the electrical system revolves around the discrete components (different boards), and what purpose each one serves. The electrical system was built around a single board to simplify wiring. The software was built around a state machine and a series of libraries in order to enable code readability, enable easier behavior programming, and quicken the debugging process. Performance was judged on how accurately and consistently disks could be fired at the targets, with higher scores being assigned to smaller targets, and performance was determined to be high, but not as high as was originally desired.

Table of Contents

Project Overview

In Mechatronic Design, students from several disciplines (Mechanical Engineering, Electrical & Computer Engineering, and Robotics) come together as a team to design and construct a system capable of achieving some task in the physical world. For this mechatronics project, the students were tasked with creating a disk-launching robot capable of sensing multiple targets and launching individual disks at each target from 10 feet away. To accomplish this objective the team created a robot with a hopper that could hold and deploy multiple disks, a track with a rotating wheel to accelerate disks, pitch and yaw axes capable of changing the direction the robot shot, and a camera that could identify the location of colored targets.

Design Requirements

The robot designed for this project was required to fit into several specific requirements. Firstly the robot must fit within a maximum volume of 2' x 2' x 2'. The robot must use a dedicated power supply, not laboratory bench supplies. The total cost of all new components must be less than $600, with an additional allowance of $200 dollars of salvaged material. The device must be solidly constructed using nuts and bolts, cable ties, and soldering rather than tape and rats' nest wiring. Most importantly, the machine must not damage anyone or anything during operation, testing or construction. The disks used in this robot are Petco Mini Flying disks. The disks have a 6" diameter and are made from malleable plastic.

To demonstrate proper disk-throwing, the robot must pass a target test. Targets are 18" wide and 10" high rectangles colored red, green, or blue. Each red, green, and blue targets features a slit 6" high and 9", 12" and 15" wide, respectively. disks must pass through these slits to be considered successful shots. These targets are constrained to a 4' wide 3' high target zone with a black background that is 10' away from the robot.

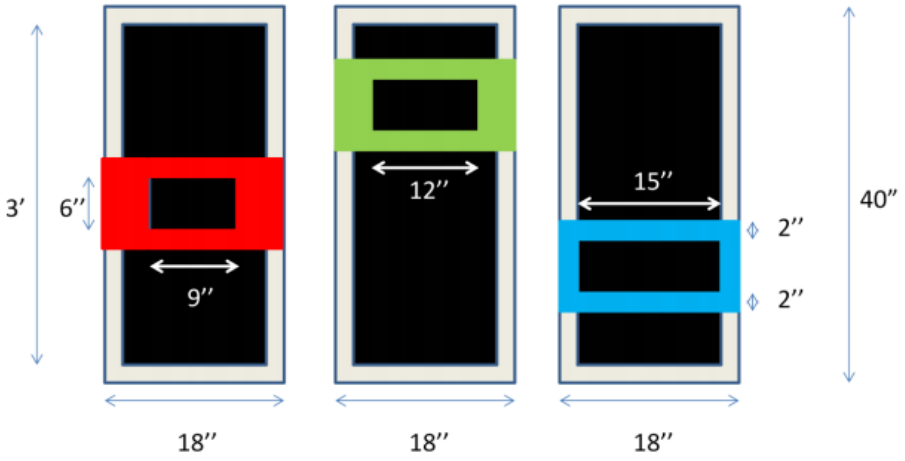

Figure 1: Target Range Setup and Dimensions

The robot is allocated 5 minutes to calibrate on the first test on a given day and up to 2 minutes of preparation for subsequent rounds. Once disk are place into the machine, 15 seconds are granted for disk sorting, target seeking, and robot movement. After this time the robot test must begin. In Test 1, the robot must be able to hit either the blue target or the green target three times in 20 seconds. In Test 2, each target has a point value and the robot has 40 seconds shoot three disks at no less than two unique targets and meet a minimum point requirement over all trials.

Design Concepts

Fundamentally our design requirements were to accurately and repeatedly deliver multiple disks to three randomly-positioned color-coded targets. The disk-shooting mechanism must then have two degrees of freedom, for the pitch and yaw (vertical and horizontal) axes. As an additional requirement, disks only "fly" in a stable manner when spinning. Thus, the shooter must simultaneously impart the disks with forward and rotational velocity. The pitch and yaw axes must have a high degree of control for sufficient accuracy, and the shooter must have the disks well-constrained during the firing sequence for meaningful amounts of accuracy repeatability (or "precision" from shot to shot). In order to shoot multiple disks within one loading sequence, some storage and loading scheme must be implemented.

To meet the aiming requirements, we designed a simple pitch/yaw system, which would occupy the bottom (for yaw) and rear (for pitch) of the robot, out of the way of the hopper and shooter. The yaw system consisted of a large, flat ring-style turntable, driven by a servo-actuated belt and pulley system, and the pitch system consisted of a lead screw coupled to a stepper motor. The servo and stepper provided excellent precision of orientation, and importantly were able to resist being back-driven, so the robot could hold a steady target. In a mix of function and form, the turntable was elevated about six inches off the base, by four stylized "legs" in keeping with our specified "Portal turret" aesthetic theme. This created space beneath the turntable for the robot's electronic systems to be stowed, centrally and out of the way of all moving mechanical parts.

The shooter needed to accomplish the task of accelerating the disk both forward and rotationally, and did this by placing the disk between a rotating wheel and a stationary, curved track. The wheel was coated in foam tape to provide better grip against the disk. This design was chosen over a belt design due to its simplicity. Unlike designs of other teams, which often featured 90° or even 180° curved tracks, after realizing that the track length was unimportant, we designed our track as a 45° curve. This allowed us to greatly economize on space and shrink the overall effective dimensions of the robot. The track constrained the disks on all four sides, to ensure repeatability of firing. Relying on gravity to hold the disks down during firing was deemed too risky due to the light weight of the disks and variable angle of the shooter. Disk storage and dispensing was handled by a hopper with two pairs of conveyor belts which would slowly lower a stack of disks one at a time, dropping them onto the rear of the track, where a servo-actuated swipe rod pushed them forwards as the track narrowed and guided the disk toward contact with the wheel.

The robot was controlled by an Arduino Mega development board, and the vision system utilized for tracking the color targets was the CMUcam4. Extensive calibration testing allowed us to map the position of a target to the desired pitch and yaw output of the robot.

Safety, both for humans, and for the robot, was a key part of the system design. One part of this was the use of the Enable line on the power supply. By controlling the voltage of this line, the power supply could be turned on and off. When the power supply is turned off, the enable lines that power the mechanical actuators are turned off, therefore "killing" the robot (safely). There were two ways to control this Enable line for safety. First, if the software detected an error condition, it could activate the enable line to turn off the power and safely disable the robot. Secondly, there was an emergency stop button that could switch the Enable line to the disable state, "killing" the robot regardless of the state of the software. This was inspired by the wise adage "There is no safety in silicon", meaning that software cannot be trusted for safety critical events. This way, if a human notices that something is behaving improperly, they can disable the robot instantly. The E-stop button was commonly used when we wanted to quickly disable the robot. The third key safety feature is the limit switches installed on the yaw and pitch axes. These were used to calibrate the axis limits during initial calibration, and then were used to trigger an emergency stop during normal operation. This way, if something caused an axis actuator to exceed the range of motion it should have during normal operation, the system will automatically shut down.

Depictions of System

The following figures show the overall design of the disk-throwing robot. Figure 2 shows the Solidworks design of the robot. The tan parts are lasercut plywood custom built for the robot. The CAD designs also includes Solidworks models for all purchase items such as motors, gears, and belts, which must be integrated with the custom-built pieces.

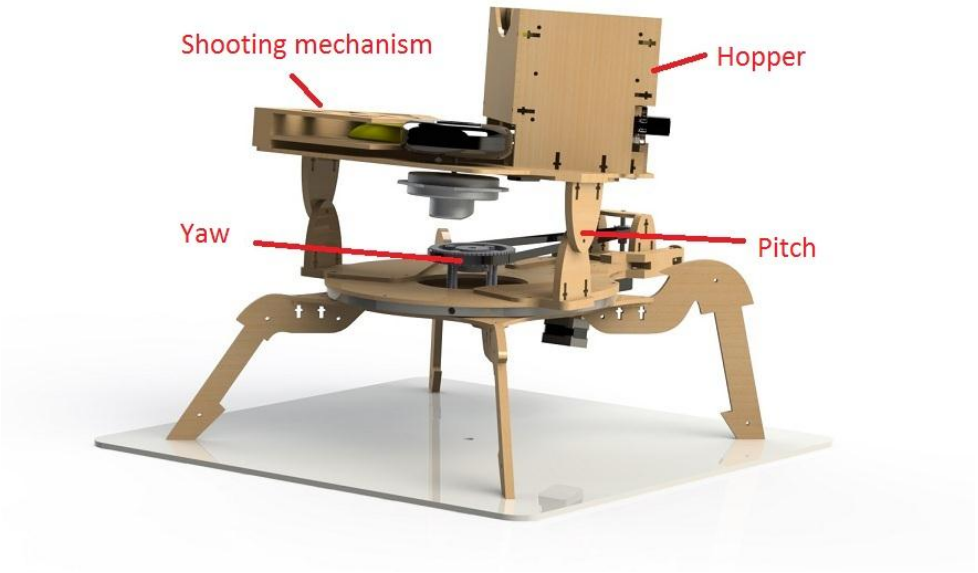

Figure 2: CAD Model of Complete System

Figure 3 below shows the final constructed robot. In the state of initialization, the pitch and yaw touch the limit switches and locate itself in the middle position. The CMUcam4 then calibrates and the Arduino controls the pitch and yaw to aim at targets. Next, disks are loaded by the hopper and are shot by the wheel.

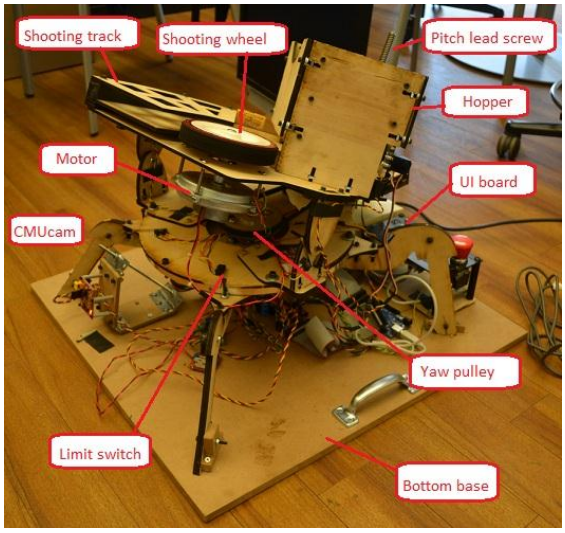

Figure 3: Complete System with Electronics

Functional Architecture

The functional architecture outlines the states of the system and the actions that must take place during each state.

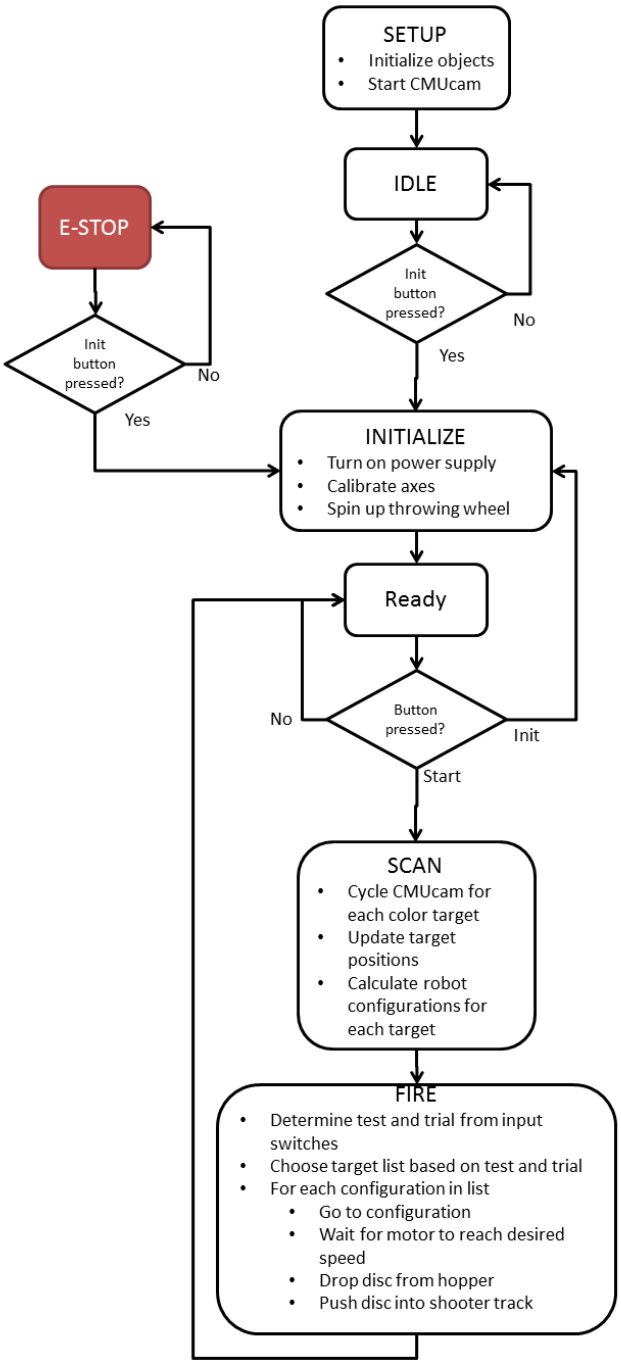

Figure 4: Functional Architecture

Physical Architecture

The physical architecture focuses on the interactions between the mechanical subsystems and the signal routing between electrical components. The firing subsystem accelerates disks received from the hopper, the pitch subsystem tilts the shooter, and the yaw subsystem turns the shooter. Meanwhile all electrical signals are routed to and from the Arduino Mega, and the PSU powers all devices.

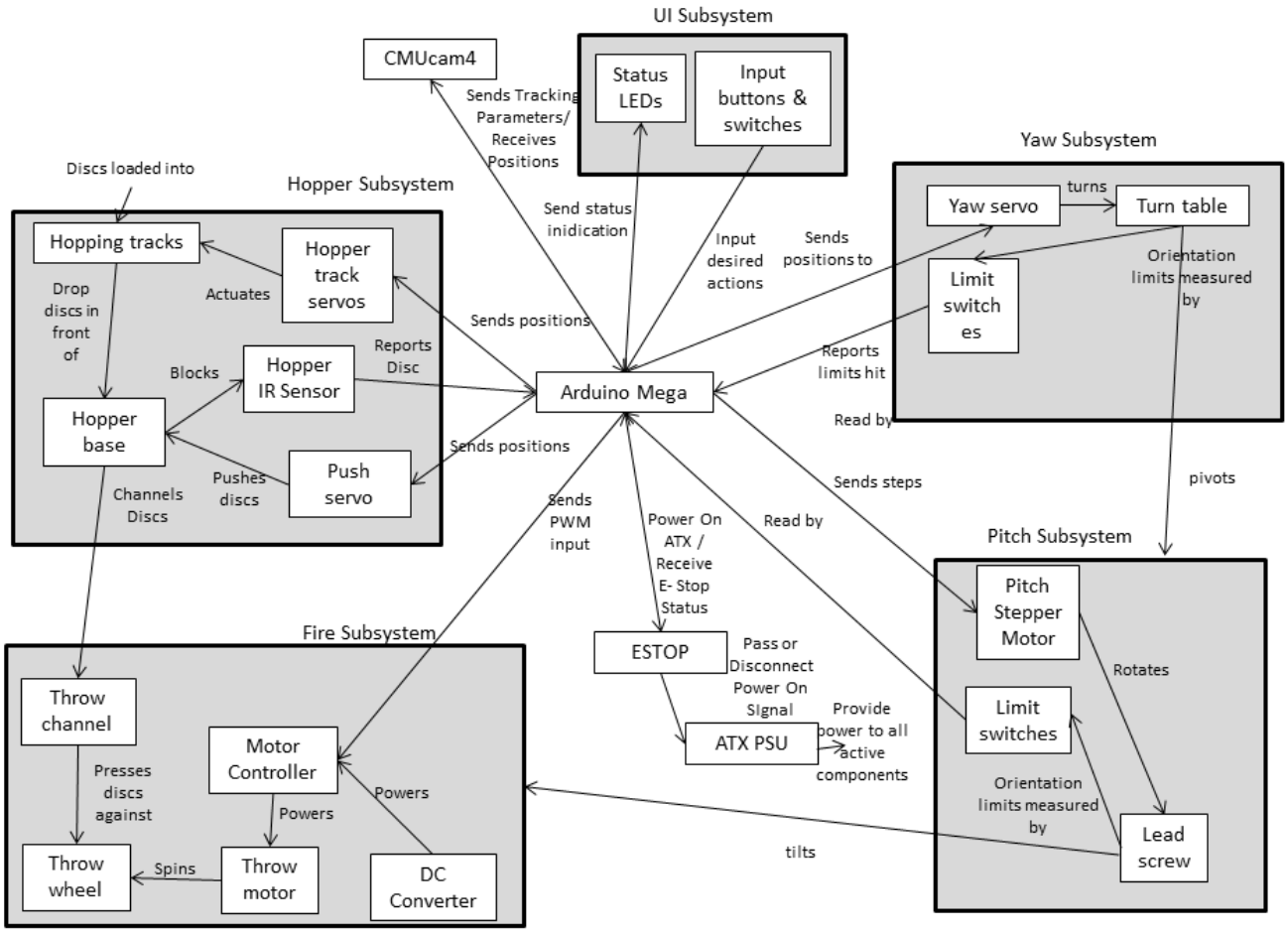

Figure 5: Physical Architecture

Subsystem Description

The robot can be broken into multiple subsystems that each serve distinct purposes. The subsystems include shooter, pitch, yaw, hopper, power, vision, user interface, sound, and processing.

Yaw

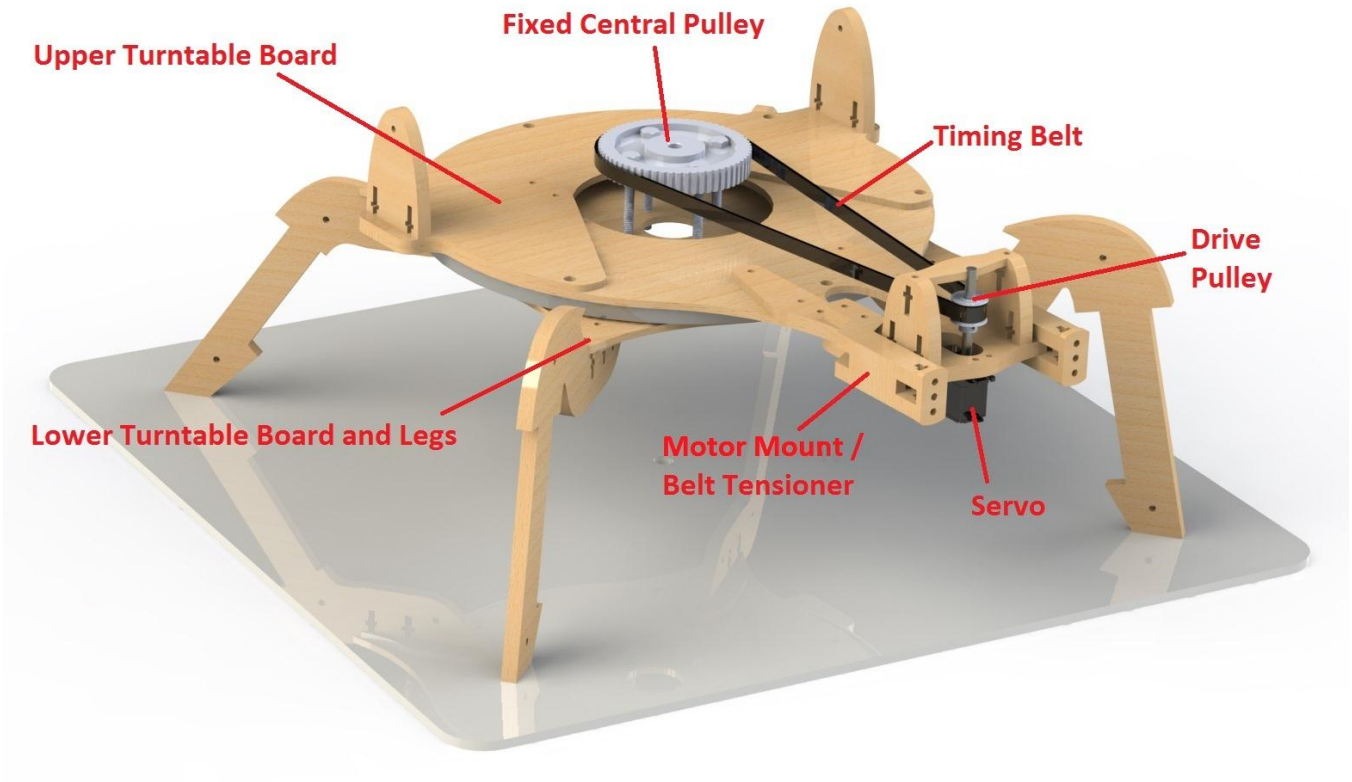

Figure 6: Yaw Subsystem

Going from the ground up, the first mechanical system of the robot is the yaw axis. The centerpiece of it is an aluminum ring-style turntable, which is essentially a very large thrust bearing, with convenient screw holes for attachment points. Attached to the upper and lower halves of the turntable are two circular boards, laser-cut from 3/16" birch plywood (like every other general structural component). The lower board attaches the turntable to the MDF base, with four stylized "legs". The appearance of the legs was intended as a coolness factor, and doubled as a functional asset by providing a convenient location for the yaw axis limit stops. The upper board serves as the mounting platform for the yaw axis motor, the pitch axis motor, and the robot's top platform. The yaw axis is driven by a two-pulley timing belt drive. The larger, stationary pulley is mounted to the lower platform and projects up through a hole in the upper platform. The smaller drive pulley is mounted to the upper platform, behind the pitch axis motor. The motor mount has build-in position adjustability, so that once installed, the belt could be adequately tensioned. The drive pulley is attached to a shaft coupled to a servo. The pulleys have a 6:1 reduction, so 180 degrees in the servo equates to 30 degrees of the turntable. This was chosen because we only need approximately 20 degrees of horizontal range, and such a reduction afforded more accuracy of lateral position. A reinforcing brace for the top of the motor mount allows us to place the belt under an adequate amount of tension without loading the shaft in a dangerous way.

Pitch

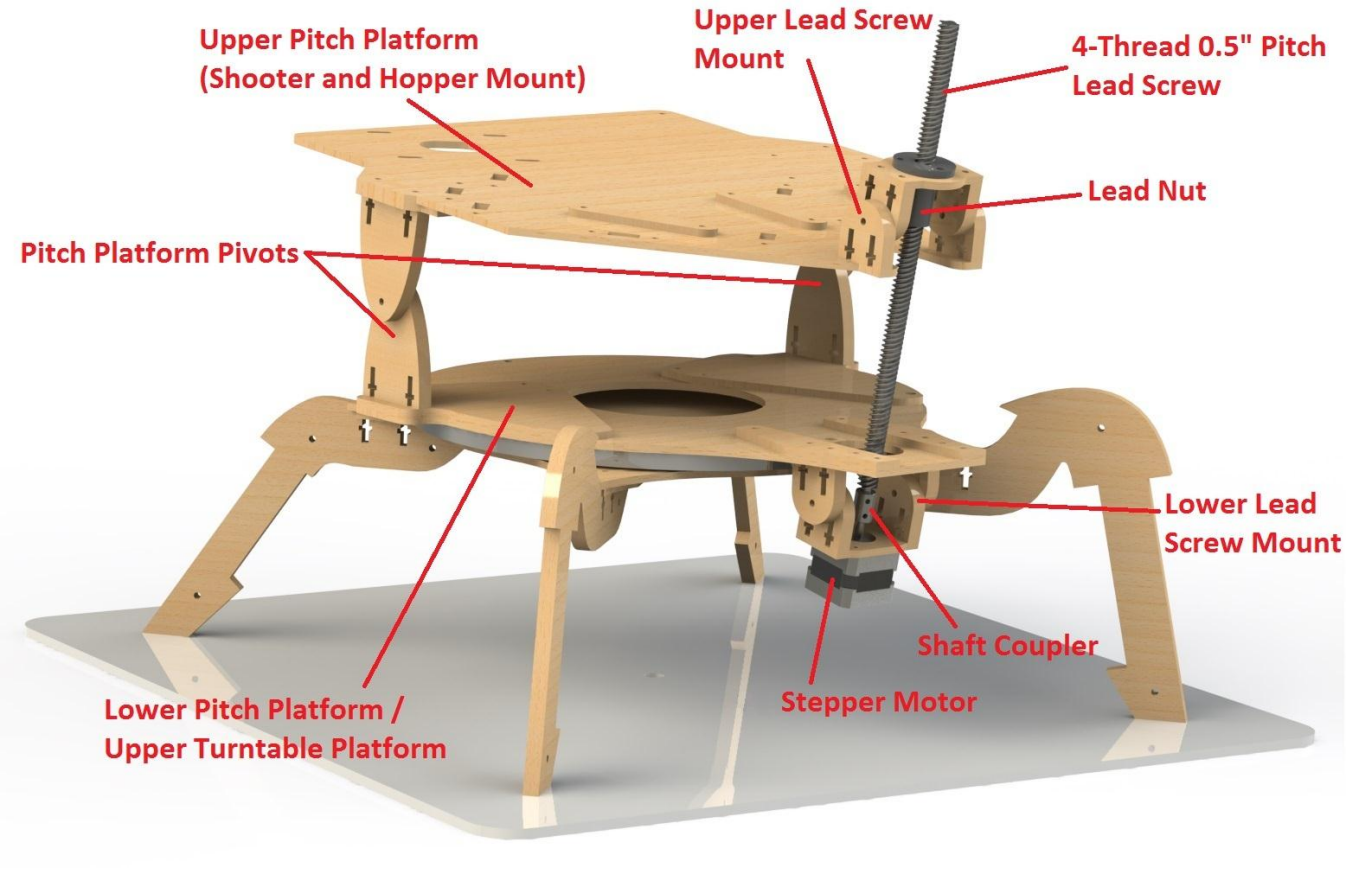

Figure 7: Pitch Subsystem

Positioned atop the turntable is the pitch subsystem. The upper pitch platform is connected to the lower pitch platform in three points: two pivots and one lead screw. These three points are spread as far apart as the robot's geometry allows, for maximum stability. The two pivots approximate a simple hinge joint, and the lead screw was chosen as the actuator for this hinge. Due to the changing geometry of the platform, both the lead screw's attachment points must also have similar (but more compact) pivoting attachments. The lead screw is driven by a stepper motor (which was used in the early class labs) which affords us a very high degree of vertical position control. The lead screw's fairly aggressive pitch of 0.5" per rotation still allowed us to quickly react to having to aim at different heights. A custom shaft coupler was manufactured for joining the stepper motor to the lead screw.

Shooter

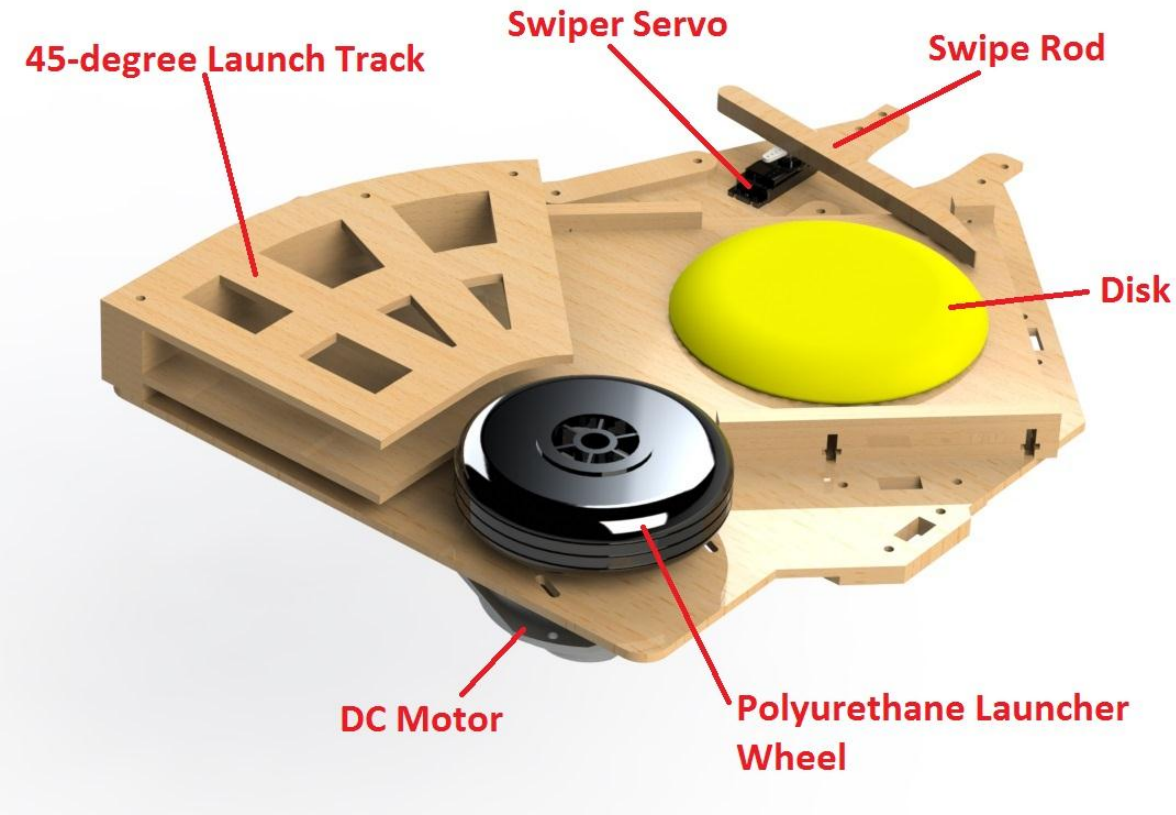

Figure 8: Shooter Subsystem

The centerpiece of the robot is the shooter. Its purpose is to receive disks dispensed by the hopper and fire them accurately out the front, with consistent linear and rotational velocity. When a disk falls into the back of the launch track, it interrupts an infrared light beam, triggering an IR sensor that lets the Arduino know that a disk has been dispensed. The swiper servo actuates, causing the swipe rod to push the disk forward. When the disk reaches the launcher wheel, it is gripped between the wheel and the sidewall of the track and propelled forward by the already-spinning wheel. The difference in radius between the wheel and the track is designed to be slightly less than the diameter of the disk, so that it is pinched between the two, and fits snugly into its rectangular slot. To adjust this amount of pinch, the attachment holes for the motor in the pitch board are actually slotted, so that it can be screwed down in a tighter or looser configuration. The track is designed so that the disk is completely enclosed and is unable to vary its position or angle within the shooting area. This ensures that the disk is always at the same orientation when it exits the shooter.

Hopper

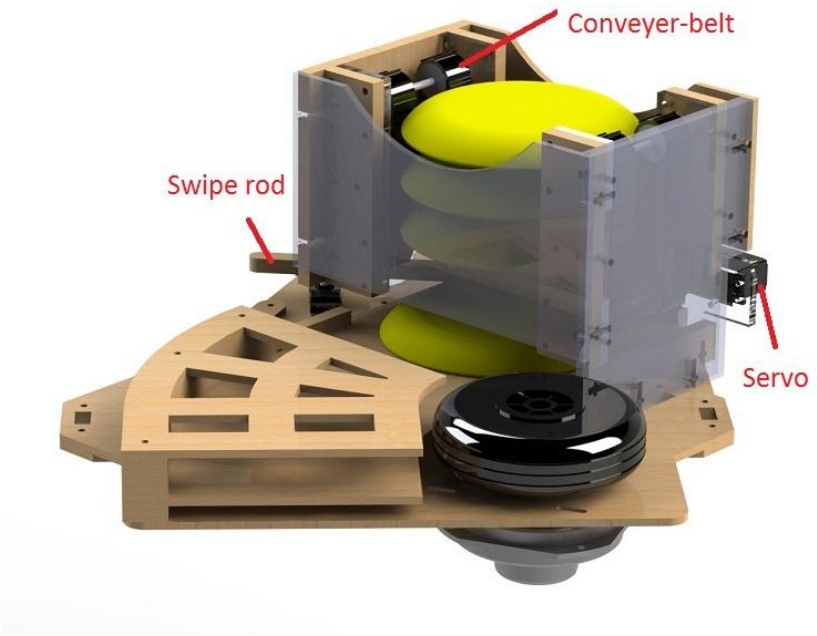

Figure 9: CAD Model of Hopper Subsystem

Figure 9 shows the CAD design model of the hopper system. There are four conveyer-belts, driven by continuous servos that transport the disks to the track. Ridges keep the disks from shifting and falling prematurely. An IR sensor at the bottom of the hopper can detects whether IR LEDs at the other side of the hopper base is blocked, indicating that a disk has been dispensed from the belts. When this signal has been detected, the software stops the belts (so as to not dump the next disk on top of the current one), and actuates the swiper arm to push the disk into the firing track.

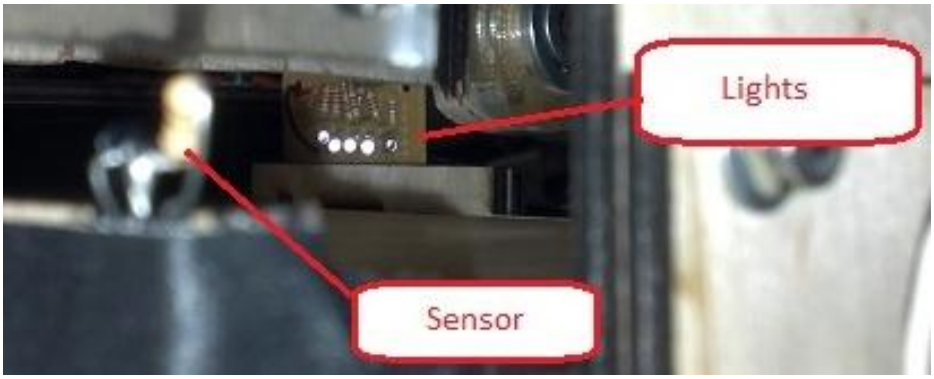

Figure 10: Hopper Sensor System

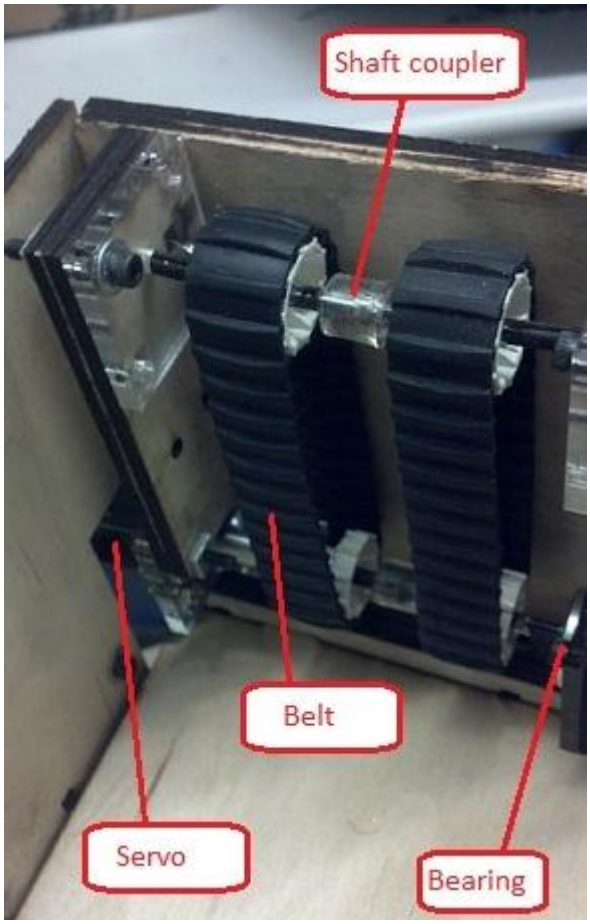

Figure 11: Hopper Conveyor Belts

Vision

A vision system was required to determine the position of the targets in front of the robot. A CMUcam4 was chosen to track the targets. The CMUcam4 was mounted on the front of the machine's base, underneath the end of the shooting track. The mount was tilted slightly upward in order to present less floor space in the shot and more viable tracking areas.

The CMUcam4 communicates with the Arduino over TTL serial. The Arduino sends configuration parameters, including color configurations and tracking windows, to the CMUcam4. The CMUcam4 uses this information to perform color tracking and returns tracking packets to the Arduino. These packets include the centroid of the target and the coordinates of the bounding box surrounding all pixels of that color.

Color tracking was done in the YUV mode presented by the CMUcam4 API. This was done based on the strong recommendation of the primary CMUcam4 developer. The advantage of YUV mode was the ability to specify one tolerance for Hue (based on the U and V parameters), and a different, wider tolerance for brightness (the Y parameter). This enabled the color tracking to be more tolerant of varying lighting conditions. Even so, it was noticed that performance identifying targets was improved by recalibrating the color selection slightly before each set of test runs.

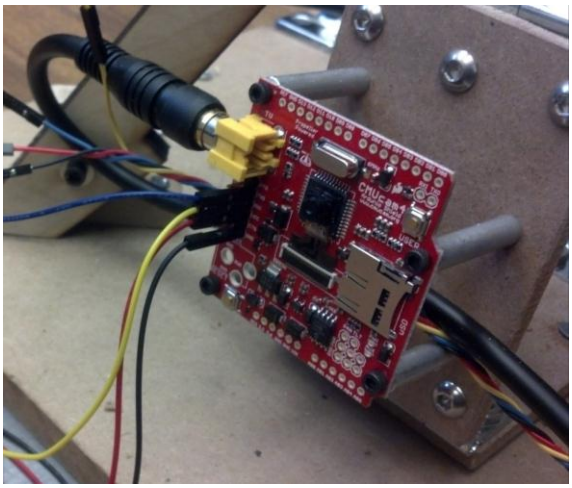

Figure 12: CMUcam4 mounted to base of Robot

Due to the noisy nature of the bitmaps returned by the CMUcam4 representing the presence of the specified color in the field of view, the "lock" on the target may not be precise. To help remove extraneous pixels not actually part of the target, we defined a scanning region for the CMUcam4 to look in, and ignore any pixels outside this region. This region was set to be approximately the size of the total target area, because any pixels outside that area cannot be part of a valid target. This significantly reduced the tracking noise, as compared to how we saw other teams target identification. Additionally, in order to improve target tracking accuracy, the size of the bounding box (around all the pixels representing the target) was compared to a reference size. This reference size was calculated to be the size of a target as viewed from a CMUcam4 at 10ft away. When the bounding box around the identified pixels matched within the margin of error to the reference size, a solid target "lock" was achieved, with the centroid returned by the CMUcam4 having a high confidence of being in the center of the target. When this match was not achieved, the scan was repeated (up to 10 times per target) in an attempt to get a better match.

Sound

One of the system's coolness factors is the ability to play sound effects. This functionality was achieved by writing a background program for the Raspberry Pi Linux board. The Raspberry Pi received I2C input from the Arduino and played stored music and sounds over the 3.5 mm audio jack. A speaker was plugged into the audio jack, allowing the robot to play sound effects.

A sounds library (custom) written for the Arduino made playing sounds simple. A play() function took the name of a sound (of which the list of available options was hard coded), and would cause the Raspberry Pi to play that sound next (unless overwritten by another call to play(). The Raspberry Pi would poll the Arduino over I2C, and get the number corresponding to the next sound to be played. The program on the Raspberry Pi would then play the specified file off the SD card. The play() function would also accept the name of a group of sounds (such as for finding a target), and randomly select one of them to play next. This made adding a new instance to play a sound at easy, as well as identifying where in the code base sounds would be triggered.

User Interface

In order to calibrate the machine, start firing, and choose between various tests and trials, a User Interface Panel was designed. The panel features the E-Stop, two buttons to enter the Initialization and Scan/Fire states, two switches to choose between tests and trials, and status LEDs to indicate the status of the machine.

Figure 13: UI Panel

The UI panel plugs into the main circuit board via ribbon cable. From there, connections are made to the GPIO pins on the Arduino and Raspberry Pi. All components on the panel share a common ground. The buttons and switches are connected to input pins and the status LEDs connect to output pins. The status LEDs indicate whether the system is in an E-Stop, Idle, Ready, or Firing state, if the Arduino is on, if the Rapberry Pi is ready to provide sound effects, and if the PSU is on. These LEDs are updated during state execution loops and in the main loop.

The use of the system via the User Interface Panel is very simple. The E-Stop button disables power and stops the system at any time, and prevents the system from continuing until the button has been reset. To calibrate, the Init button may be pressed. To begin a test (finding and shooting at targets), the Start button may be pressed. The Test and Trial switches are used to select which combination of targets will be fired at. The reset button does not reset any part of the system, but instead plays various songs from Portal or Portal 2, such as “Still Alive”. The song to be played is also selected from the Test and Trial switches.

Power

The entire system is powered by an ATX power supply unit. The PSU plugs into AC wall power and produces several different voltage outputs, including 3.3V, 5V, and 12V. An additional line provided by the ATX connector is the green Power On line. This line floats at 5V when not connected to any other voltage. This line must be grounded in order to turn the PSU on, and continually pulled low to keep the PSU on. The one exception to this is the 5V Standby line, which is at 5V regardless of the state of the Power On line. However, this line was only capable of providing 2A of current, meaning that it was only useful for powering the devices with built in microcontrollers.

The system’s power design takes advantage of these PSU features. All embedded processing, including the Arduino Mega, the CMUcam4, and the Raspberry Pi are powered by the 5V standby line. Once the system is powered, all these components turned on and the Arduino begins its main routine. The Power On line is connected to the Arduino Mega through the Normally Closed relay of the E-Stop. In order for the main power to turn on, the E-Stop must be off to keep the Power on line from floating, and the Mega output must be Low. This

allows the main power to be shut off by software errors or by the human operator, regardless of the current software state.

When the main power is connected, the 5V line is used to power all active sensors, servos, and other low voltage devices on the robot. The 12V line is used to power the speakers, the stepper motor, and the DC converter for the 24V motor that drives the shooter wheel.

All electrical power and signaling is brought together in a single circuit board at the core of the robot. The board includes a breakout for the PSU connector, ribbon connectors that connect to the dozens of Arduino pins, and male pins to plug in molex connector cables to sensors and actuators. The plans for this board are included as figure XX.

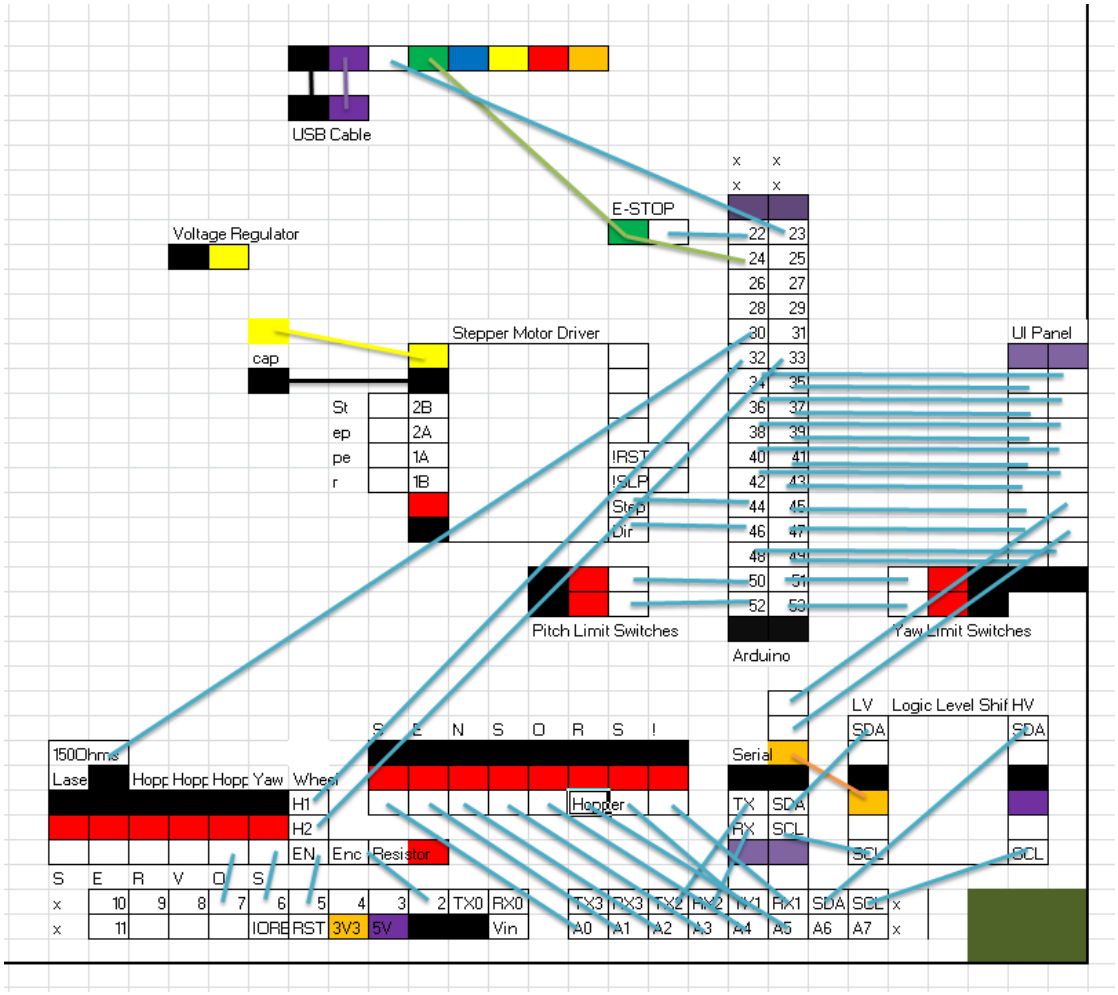

Figure 14: Circuit Board Layout

Processing

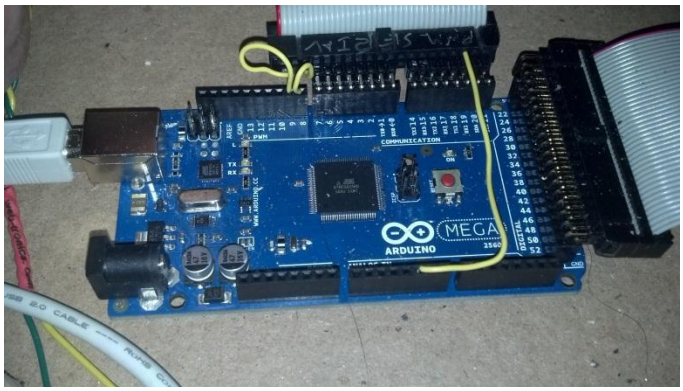

Figure 15:Arduino Mega with Ribbon Cables

An Arduino Mega was chosen to run the machine's core routine. This design decision gave several benefit. The Arduino Mega has many libraries and lots of support online, is easy to replace at a local Radioshack, and features 4 serial lines and over 50 GPIO pins, the majority of which were needed for the robot. The Arduino interfaces with the rest of the machine via ribbon connectors to the main circuit board.

Software

The software was designed with several libraries. Rather than running through individual pin actions for every action of the robot, classes were developed for each feature of the robot, including state machine, PSU, binary sensor, binary output, stepper motor, yaw axis, and sounds. Instances of each of these classes were created during the Arduino's setup, and functions of each instance were created. This allowed the main loop design to abstract away individual signals and focus on behavior. Additionally, code duplication was minimized due to not having to copy and paste code for each duplicate piece of hardware (such as multiple LEDs on the user interface panel, and the many buttons and switches).

The software architecture is built around a state machine concept. Once the arduino leaves setup and goes to its main loop, it falls into our designed state machine. Our global state variable is an enumerated type with values for each state. The main loop enters a switch statement controlled by this variable, and each branch of the switch executes a state-specific function. Based on what happens during their execution, these functions can change the state variable and affect the flow of the program.

We have several defined states for our system. The Idle state is entered after the Arduino finishes setting up objects and the CMUcam4. The system stays in Idle until the Init button is pressed and the system enters the Initialize state. The robot then turns on the power supply, calibrates the axes, and spins up the shooter wheel motor. After these tasks are complete, the system moves to Ready state. In Ready state, the system waits for button commands to either re-Initialize or to start the shooting sequence. The shooting sequence begins with a Scan state, where the robot looks for targets, and concludes with a Fire state, where the robot dispenses disks from the hopper and shoots them at the targets. After this is completed, the robot returns to Ready state. The E-stop state can be called during any state if an error condition is encountered.

Within each state, various library function calls are made in order to take the necessary actions for that state. The libraries were primarily custom written (with exceptions, such as for the CMUcam4) for this project to make the main body of code (the states) easier to write, understand, and debug.

System Modeling, Development, and Performance -

For the robot to accurately shoot targets, well-defined transformations were needed to translate the Cartesian target coordinates from the CMUcam4 into robot pitch and yaw coordinates.

The first approach was geometrical. Cartesian coordinates of the target range were compared to inch measurements of the target range to establish a conversion from CMU coordinates to inches. Next, the geometry of the triangles formed between the 10' from the robot to the target, the target's deviation from center of the range, and the hypotenuse representing the straight line shot was examined. The inverse tangent was used to determine the yaw angle required for the shot. This calculation required only a 2 degree bias in order to accurately shoot disks with regards to yaw.

A similar attempt was made to determine the proper pitch angle, involving the conversion of the stepper motor steps to inches, solving the triangle formed around the lead screw, and relating this angle to the angle required to hit the target after factoring the height of the machine and the amount of fall during the flight. This approach was unreliable due to this multitude of factors. Instead pitch angle was controlled manually over serial, proper pitch angles were found for various target heights, and a quadratic curve was fit to the data. This quadratic curve was successfully used to command pitch angles.

The performance of the robot in terms of shooting disks was passable, but not as high as was initially desired. The vision system was capable of finding targets, and to a high degree was able to identify the pixels in the target without too many extraneous pixels. Once the targets were found, aim (yaw and pitch) were reliably able to line up with the target for firing. Unfortunately, while the system was accurate for aiming at the targets, it was not the most precise. Disks would fly to within a foot of the target, varying in the direction of error. The lack of precision was discovered to be attributable to the disk being used (each disk was consistent in how it flew) and the time between shots. This led to an overall lack of precision in firing disks at the targets. It was considered to program in adjustments for the different disks, but there was not sufficient time to collect the calibration data and program it in. The error due to time between shots was guessed to be due to the launch wheel spinning at different speeds. This could have been corrected by hooking up the encoder on the launch motor and programming in velocity control for that motor. This was also not completed due to time constraints, and fear of electrically damaging the already functional system.

The disks were able to be fired at a sufficient speed. Although this was not calculated, it was judged as "fast enough" for the flight to be straight and level (with minor altitude drop over the flight path). However, we were unable to measure flight speed, so we couldn't judge consistency of that parameter.

In order to judge the precision of the aiming system, we had the system identify a target, and actuate pitch and yaw to aim at the target. We could reliably have the system arrive at the same location for the same target location, meaning that the vision system was identifying targets consistently, the calculations for position were correct, and the yaw and pitch actuators were moving to the same positions, which put the system aimed at the correct location. This correct position was tested by firing the same disk over and over, and watching it arrive in the plane of the targets in the same location each time.

Conclusions and Future Work

If this project was started from scratch again, there are several things we would have done differently. One of our major problems was precision. While our design was fairly precise, it was not the most precise in the class. We would have ordered a specific motor and wheel chosen to meet the specifications of the project. Our motor was not quite powerful enough to keep the wheel going at a uniform speed. Adding velocity feedback control would also have helped with this issue. We also would have tested different throwing track geometries earlier to determine which track geometry produced consistent disk paths. We should have also started our hopper earlier. Our design used plastic material that exhibited signs of fatigue by the end. These issues could have been resolved with more rapid prototyping.

One last problem we had to adapt to was vibration. Our quick and ugly fix was to tape a wrench to our shooting platform to change the mass and change the resonant frequency of the machine. However stronger materials with some damping built in could have aided with this issue.

Overall our biggest issue was a problem of stealing and tampering. We had multiple instances of components taken from our bench or boxes or odd things done to our machine. Going into this again, we would have gotten sorted parts boxes earlier, labeled everything as soon as we got it, and gotten a cover for our machine as a symbolic barrier.

Because our entire machine was designed from scratch and lasercut, many parts were difficult to make early or required multiple iterations. This slowed the testing process but did result in a quality product by the end. Knowing what we know now, we would have been more selective with which components needed to be designed and laser-cut. We would have prioritized critical components (shooter, hopper) over pieces that could be made by conventional shop tools (legs, base platform).

If we were building this as a startup company for public consumption, we would probably want to make more components hidden and internal to the machine. Yaw control could be switched from the top of the platform to underneath and contained. The biggest challenges would be simplifying the hopper without making it more prone to jamming and making the lead screw less exposed.

Parts List

#	Supplier	Type and description	Quantity	Unit Price	Total Price
1	SparkFun	CMUcam4	1	99.95	99.95
2	Pololu	Limit Switches	10	5.76	13.05
3	Sparkfun	Logic Level Converter	1	1.95	1.95
4	Sparkfun	Microcontroller, Arduino Mega 2560 R3	1	58.95	58.95
5	Sparkfun	ATX Power Supply Connector - Straight	1	1.50	1.50
6	Sparkfun	ATX Power Supply Connector - Right Angle	1	1.95	1.95
7	Sparkfun	Laser Card Module	1	7.95	7.95
8	Sparkfun	RCA Jack	1	0.95	0.95
9	Sparkfun	Concave Button - Yellow	1	1.95	1.95
10	Sparkfun	Concave Button - Green	1	1.95	1.95
11	Sparkfun	Toggle Switch	2	1.95	3.90
12	Sparkfun	ATX Connector Breakout Board	1	4.95	4.95
13	Sparkfun	ATX Power Supply Connector - Right Angle	1	1.95	1.95
14	Amazon	LM2587 DC Booster Converter	1	11.99	11.99
15	McMaster	Wheel for firing mechanism	1	15.20	15.20
16	SDP-SI	Timing belt for yaw mechanism	1	6.61	6.61
17	SDP-SI	10-tooth Timing pulley for yaw mechanism	1	10.34	10.34
18	McMaster	Bearings for yaw mechanism	3	4.39	13.17
19	Roton	1/2 x .500 Lead screw for pitch mechanism	1	10.25	10.25
20	Roton	1/2 x .500 Flange nut for pitch mechanism	1	20.40	20.40
21	Roton	1/2 x .200 Lead screw	2	8.53	17.06
22	Roton	1/2 x .200 Flange nut	1	20.40	20.40
23	McMaster	Aluminum turntable for yaw mechanism	1	33.02	33.02
24	SDP-SI	60-tooth Driven pulley for yaw mechanism	1	28.76	28.76
25	SDP-SI	150-tooth belt for yaw mechanism	1	8.82	8.82
26	Servocity	1/4" Servo to Shaft Coupler	1	19.98	19.98
27	McMaster	#8-32 5/8" hex cap screws, 100ct	1	12.48	12.48
28	McMaster	#8-32 1" hex cap screws, 50ct	1	7.51	7.51
29	McMaster	#8-32 nylon insert thin hex locknuts, 100ct	1	2.30	2.30
30	McMaster	#8-32 washer	1	1.42	1.42
31	McMaster	New 5" wheel for shooter	1	9.02	9.02
32	McMaster	Additional 5" wheel for shooter	1	9.02	9.02
33	McMaster	Backup 5" wheel for shooter	1	2.77	2.77
34	The Home Depot	MDF 1/4'' 2'*4'	1	6.37	6.37
35	The Home Depot	Birch handpanel 1/4'' 2'*4'	2	8.74	17.48
36	RoboClub Stock	Lego Tank Treads and Wheels	1	5.00	5.00
37	RoboClub Stock	IR LED	10	0.30	3.00
38	RoboClub Stock	QEC122 IR Detector	1	1.00	1.00
39	RoboClub Stock	3001HB Power HD Servo	2	10.00	20.00
40	Mechatronics Stock	Stepper Motor	1	20.00	20.00
41	Mechatronics Stock	Solarbotics L298 Motor Driver	1	18.00	18.00
42	Mechatronics Stock	Crescent Wrench	1	14.00	14.00
43	Salvage	Continuous Rotation Servo	2	5.00	10.00
44	Salvage	DC Motor	1	10.00	10.00
45	Salvage	ATX Power Supply	1	10.00	10.00
46	Salvage	E-Stop	1	15.00	15.00
47	Salvage	Raspberry Pi	1	35.00	35.00
48	Salvage	Computer Speaker	1	2.00	2.00
				Total:	**648.32**

Citations

"Arduino." Arduino. Arduino, n.d. Web. <http://www.arduino.cc/>.

"CMUcam: Open Source Programmable Embedded Color Vision Sensors." CMUcam: Open Source Programmable Embedded Color Vision Sensors. CMUcam, n.d. Web. <http://www.cmucam.org/>.