

# The Catch-Bot

Final Report

Submitted to

The Faculty of Operation Catapult 100

Rose-Hulman Institute of Technology

Terre Haute, Indiana

by

Group 38

Baran Usluel

Bilkent Lab. & International School  
Ankara, Turkey

Jason Kurohara

Westlake High School  
Westlake Village, California

Nate Irwin

Petaluma High School  
Petaluma, California

Drew Kleiman

Bishop Ireton High School  
Fort Washington, Maryland

July 26, 2016

## TABLE OF CONTENTS

Introduction.....	3
User's Manual.....	4
Hardware.....	6
Software.....	12
Testing.....	15
References.....	16
Acknowledgement.....	17
Software list.....	18

## Section 1.0-Introduction

Robots improve the quality of human life through assistance in surgery, military, and industrial manufacturing. In the 21st century, robot technology is booming, and even the simplest robots are in high demand. This report describes the design, programming, mechanics, and optimization of a tennis ball catching robot called the Catch-Bot, which is the manifestation of the vast capabilities of robot technology. For humans, catching a ball is simple. For our group and the Catch-Bot, this simple task requires complex programming, hands-on building, communication, and teamwork. While the Catch-Bot will not directly benefit humanity, this machine demonstrates the potential of robotic technology as well as smaller, cheaper microcontrollers.

The Catch-Bot captures the motion of a projectile thrown at the robot using two identical cameras mounted on opposite sides of the frame. These images must then be processed by a computer that executes a C++ program. This program converts the images from RGB to HSV (Hue, Saturation, and Value) format and applies a binary threshold on the HSV values to create a new image in which the colored projectile appears white and everything else black. A Hough Circle Transform is utilized to detect the circle in the image, and the detected pixel coordinates are used to calculate a direction vector for each camera. The software then triangulates the projectile's position in 3D space by searching for the intersection or closest point of the two 3D lines (consisting of direction vectors and the positions of the cameras). At a rate of approximately 30 frames per second, the C++ program repeats this process to collect multiple data points and calculate the best fit curve of the projectile's trajectory. The computer sends the X and Y coordinates of the final destination of the projectile to the Arduino board to instruct the end-effector to intercept and catch the projectile.

The Arduino code actuates the motors to move the end-effector to the final destination. To avoid delays in the serial communication of the computer and arduino, the coordinates are read by the Arduino in the form of individual bytes and then converted to numbers, for reading them as strings is very inefficient.

The mechanics of the Catch-Bot are influenced by Cartesian coordinate robots such as 3D printers and CNC machines. To catch the falling projectile, the crate, or end-effector, must slide on two parallel rods in each of the X and Y directions using a stepper motor-driven pulley system. Mounted on one side of the 24 by 24 by 3.5 inch acrylic frame, a stepper motor and large pulley rotates the timing belt to move the primary acrylic tray horizontally. An additional stepper motor and pulley mounted on this tray rotates another belt to move an upper acrylic tray in the perpendicular direction. The end-effector begins in the center of the frame and can traverse the complete length of the frame in around 1.08 seconds.

The assembly of the Catch-Bot came with unexpected difficulties. The tension of the belt caused the weaker acrylic to bend inwards, so a wooden dowel was added to prevent bending of the frame. The rapid rotation of the stepper motor caused the belt teeth to slip from the 3D printed pulley, so the Arduino code was modified to accelerate the speed of the motor gradually. Additionally, the 3-D printed pulleys were replaced with metal pulleys. These modifications yielded more consistency in the movement.

## Section 2.0-User's Manual

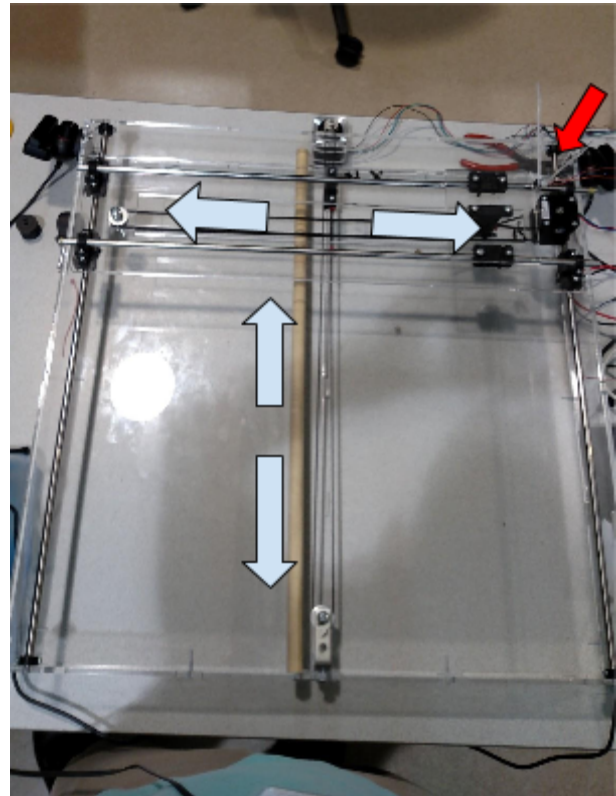
The Catch-Bot operates on a 12V power source, computer, and 5V USB connector. First, make sure that the 12V power source is unplugged (to disable the motors) and drag the crate/end-effector to the bottom left corner of the frame (marked by the red arrow). Then, connect the Arduino to the computer using the USB cable. Connect the cameras to the computer, and run the C++ program (see below). Finally, plug the 12V source back into the power jack. This initializes the system and zeros the position, allowing the Catch-Bot to orient itself in the frame.

The two cameras must remain at their precise angles, for any alteration will reset the calibration and reduce the chances of catching the ball. The angle of these cameras is approximately 45 degrees above the horizontal.

Also, the tension of the belt must be as tight as possible for reliable motion. In order to reach the maximum tension below the lower tray, loosen the L-bracket with the idler on the side of the frame, leaving 2 inches distance from the frame. Wrap the pulley around the stepper motor and the idler and fasten the pulley belt to the belt clamp using zip ties and 4-40 nuts and bolts. Compress the belt in the clamp. The tension in the belt should be tight and without slack. Then, slowly tighten the corner brace until the belt is parallel to the bottom of the frame and isn't sliding up or down the idler during motion. Do not tighten the belt too much, but it should be taut. For the upper tray, wrap the pulley belt around the stepper motor and idler. Compress one end of the pulley belt in the clamp and zip tie both ends of the belt together. A belt tightener may be used for additional tension.

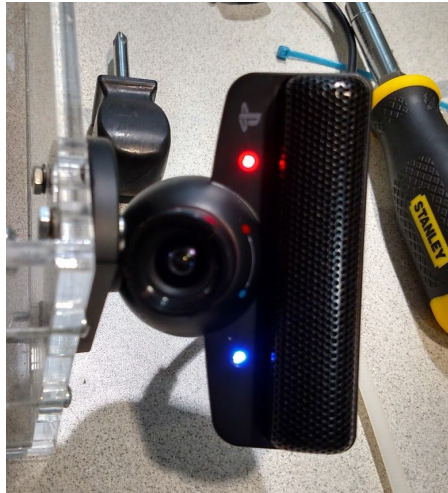
The PCB board is mounted using velcro to the side of the frame. The wires from the stepper motor that operates the upper tray, and the associated bumper switch should have excess length to slide freely.

Use a ball that greatly contrasts the other colors in the room. A blue ball will be used in this case. Stand approximately 3 meters away from the front end of the robot, which can be identified by the frame-mounted stepper motor. Then, throw the colored ball at a minimum of 2



meters above the floor at the Catch-Bot, within the fields of view of the front-facing cameras. This will allow the robot to track the ball and calculate its trajectory.

To use the software, download the Windows executable files onto your computer, from <https://github.com/BaranCODE/BallCatcher/tree/master/Releases>. Make sure the two PS3 Eye



cameras are plugged into your computer, and that you have the necessary drivers (commercially available from <https://codelaboratories.com/>). You may then run the file

called BallCatcher.exe by double-clicking on it. The program will automatically connect to the two cameras and the Arduino. You will see four windows, two of which contain the images obtained from the cameras as well as dots on the detected circles, and the others containing the images after the HSV conversion and binary threshold.

There will be one additional settings window with sliders for the HSV threshold value, a slider to toggle the triangulation, and a slider to toggle demo mode (see below). The last command-line window is for debugging purposes. To start

the triangulation and control of robot, move the “Enabled” slider to 1.

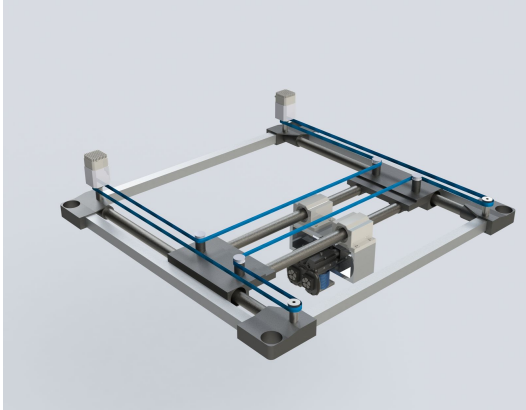
The software comes with a demo mode to be used for demonstrations. As the trajectory calculations are currently not working reliably, ~~and the robot can not catch balls as intended,~~ the demo mode disables the trajectory calculations and sends the triangulated coordinates of the ball directly to the robot. Hence, this mode lets the robot mirror the motions of the ball in one’s hand. Demo mode is enabled by default; to disable it and observe the experimental trajectory calculations in action, move the “Demo” slider in the Settings window to 0.

**Section 3.0-Hardware****Section 3.1-Components List**

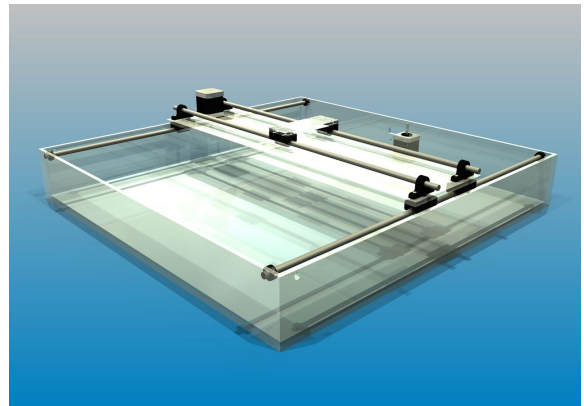
<u>Component</u>	<u>Quantity</u>
Acrylic Frame (24 x 24 x 3.5 in)	1
Computer	1
Arduino Nano board	1
12V power source	2
Power jack	1
1 100uF capacitor	1
Upper carriage (4 x 5 in)	1
Lower carriage (20 x 6 in)	1
Stainless steel rods (24 in)	4
Wooden dowel (0.629 x 24 in)	1
Nema 17 Stepper Motor 17HS19-2004S	2
L-Bracket	1
Pulley Belt (5m)	5m
Pulley	2
4-40 Nuts, 4-40 Bolts	50
Short 3D Printed Belt clamp	1
Long 3D Printed Belt clamp	1
3D printed bearing mount	6
3D printed collar	4
3D printed collar with mount	4
PlayStation™ Eye cameras	2
Ball bearings	2
Bumper switch	2
10 kOhm resistor	2

### Section 3.2-Frame

The mechanics of a 3D printer heavily influenced the design of the Catch-Bot. Both machines slide on smooth rods using a basic pulley system for optimal accuracy and precision. This design provides full mobility within an area of simple Cartesian coordinates delivered by the computer. Each motor controls one of the two axes.

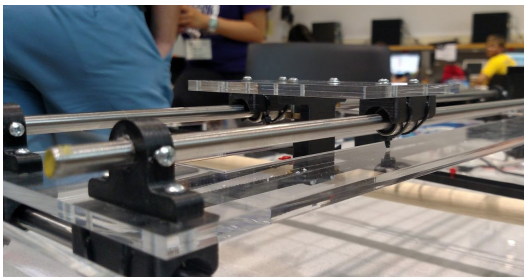


*FDM 3D printer*

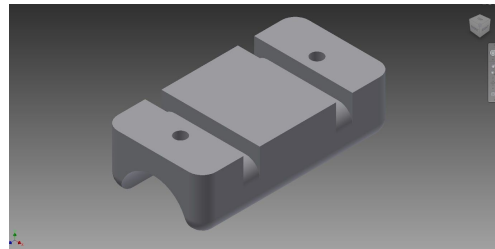


*3D rendering of the Catch-Bot*

The laser cut acrylic frame is 24 x 24 x 3.5 inches, a substantial 576 square inch range to catch a ball. We constructed four identical 3D printed collars using AutoDesk Inventor, a 3D modeling software. These collars fasten two smooth stainless steel rods to the frame.



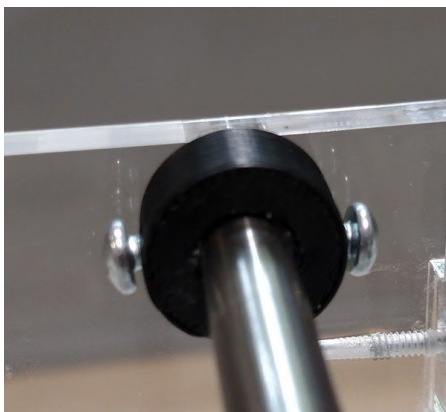
*Lower and upper trays*



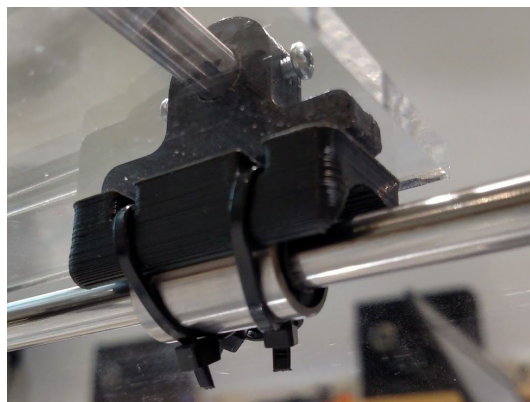
*3D model of bearing bracket*

The lower acrylic tray slides on linear bearings that are 8mm in diameter, slightly larger than the diameter of the stainless steel rods. These bearings reduce frictional force in order to maximize the speed of the crate. A pair of the linear bearings are zip tied to a pair of 3D printed brackets that attach to the lower tray. The brackets also connect to the steel rod holder on the upper tray, compressing the rods between two screws. Two more steel bearings hold the upper

tray and crate to catch the ball. Each of the 3D printed parts were “tapped” or hollowed with screw threads. However, the ABS plastic, the material used by the 3D printer, is not strong enough to hold a tightened metal screw, so crucial joints are connected by a 4-40 nut and bolt. Fluency with AutoDesk Inventor allowed us to assemble custom pieces that usually take two hours to manufacture.



*3D printed collar*



*Linear bearing with 3D printed bracket*

### Section 3.3-Stepper Motors, Pulleys, and Reinforcement

Like 3D printers and laser cutters, the Catch-Bot uses two pulleys to move the crate in the X and Y directions. The lower and upper trays are connected to a rotating pulley belt and stepper motor. The diameter of the pulley determines the linear speed of the crate, so we 3D printed pulleys of a specific size. In order to derive the linear speed, we measured the angular velocity by fastening a pencil onto the stepper motor using a zip tie and spinning it at maximum speed. The pencil hit a sheet of paper, indicating a revolution. We discovered that the motor spun at a rate of 280 rotations per minute, and we derived that a pulley of 0.865 inches in diameter resulted in a linear speed of approximately 12.7 inches per second. This is half the length of our robot, meaning that it can reach from the center to any side or corner in a second, which would be enough time to catch the ball. However, these 3D printed pulleys were inherently flawed. They were printed with supports, which ruined the surface finish and thus the effectivity of the pulley grooves. The second iteration of the 3D printed pulley didn't have this issue; however, the grooves for the pulley belt were still too shallow. Consequently, the rapid rotational acceleration of the pulley resulted in slippage between the teeth on the pulley belt and shallow teeth on the 3D printed pulley. The third and final iteration of the pulley system was smaller and made of metal, equipped with deeper grooves. Sacrificing a smaller diameter and reduced speed, this iteration does not cause any slippage, proving to be the most consistent model.

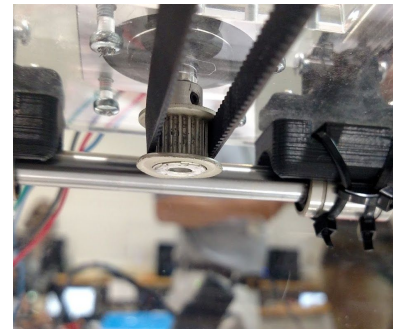




*3D printed pulley with support artifacts remaining*



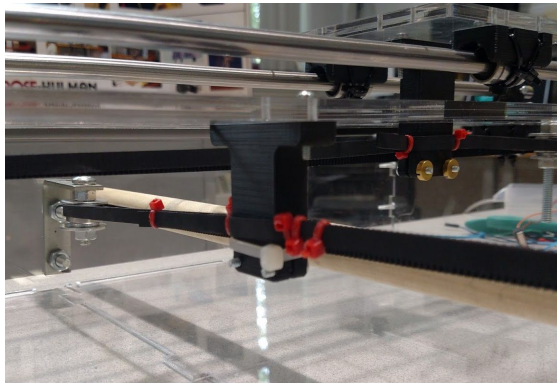
*3D printed pulley with better print quality*



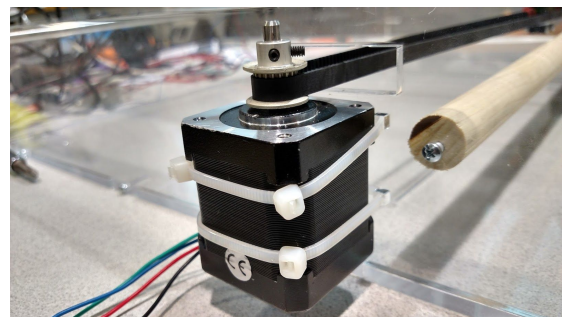
*Metal gear pulley (most effective)*

After the Arduino board receives coordinates from the computer, the motors are instructed to rotate a given amount of steps to bring the end effector to the final destination. To actuate the middle tray, the belt controlled by the pulley is attached to a 3D printed belt clamp, and is compressed by zip ties. This integral piece of the actuator must hold a large tensional force so that the belt does not slip off of the pulley gears. The opposite end to the motor and pulley is an idler, to keep the belt on track and taut.

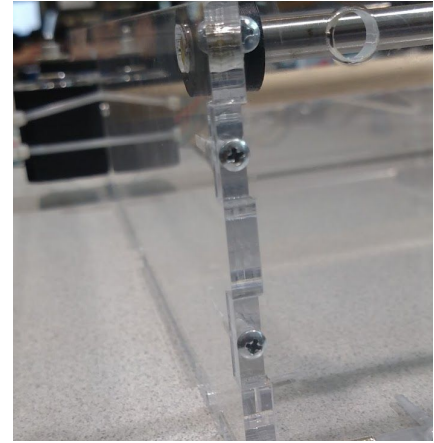
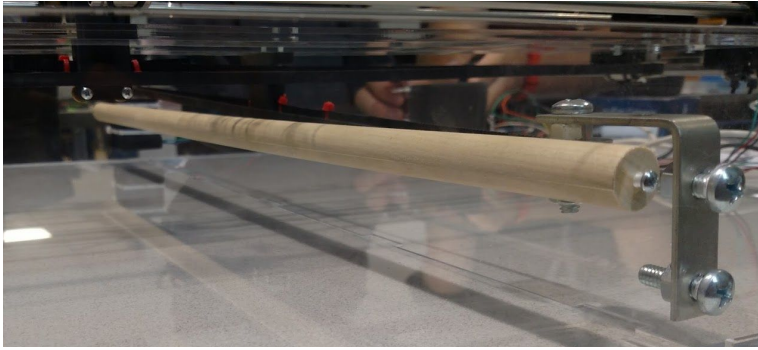
The tension of the pulley caused one side of the acrylic frame to warp dangerously inwards. To overcome this obstacle, the acrylic frame was reinforced with screws and a wooden dowel.



*3D printed clamps to fasten pulley belt*

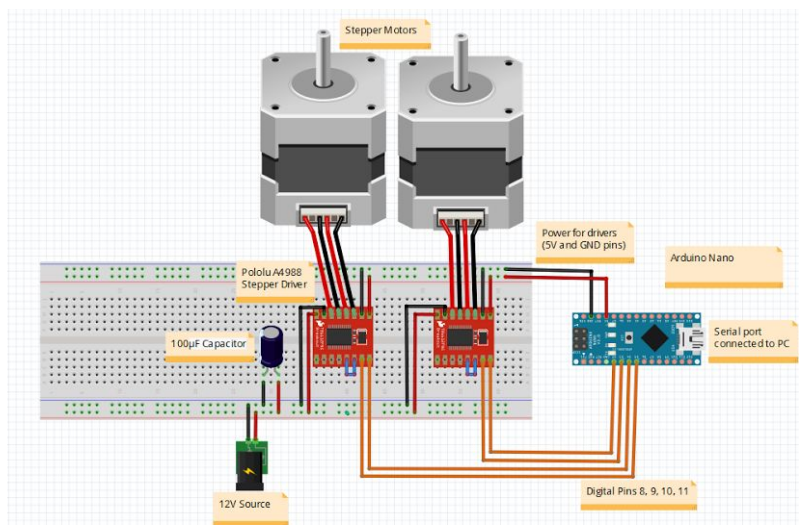


*Nema 17 Stepper Motor 17HS19-2004S*



*Reinforcements to the acrylic frame*

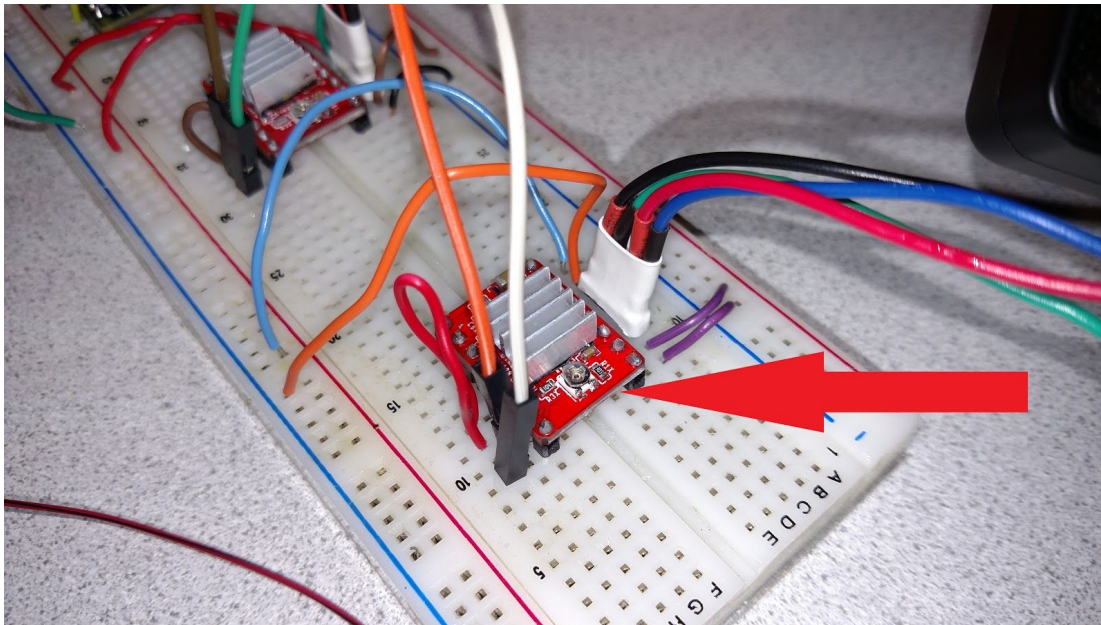
### Section 3.4-Circuit Schematic



*Circuit schematic  
designed on  
Fritzing software*

The arduino is powered by a 5V USB port connected to a computer. Connected by a serial port, the computer executes the C++ program to determine the final destination of the ball and sends the data to the arduino. Stepper motors require more voltage and current, so we used a 12V source from a wall outlet. The stepper motor drivers are connected to both the 5V on the Arduino and the 12V source, on separate pins.

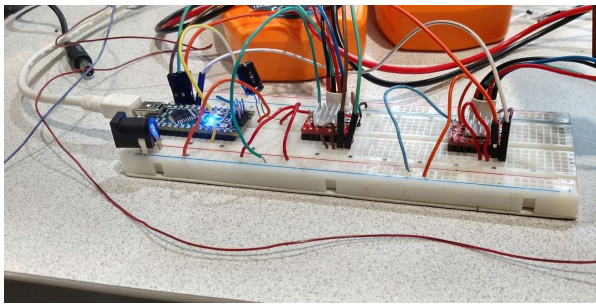
Knobs on the driver adjust the current supplied to the stepper motors. An increase in the amount of current results in more torque, at the expense of excess heat production. It is best to adjust it until it is at its “sweet-spot”. During our first test, the stepper motor controlling the X-axis would not move because of the heavy load on the rods and motor. After increasing the current with an adjustable potentiometer, the motor could rotate with ease. However, the drivers may overheat if the motors are used excessively.



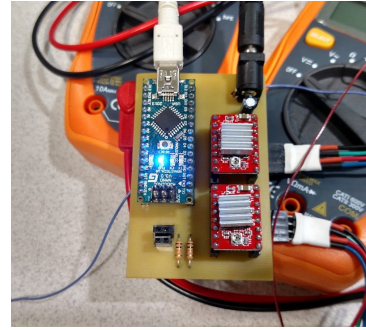
*Driver knob that controls current supplied to the stepper motor*

In order to streamline the circuitry and bundles of wires on the breadboard, we converted the messy breadboard to a much more concise version on the Printed Circuit Board (PCB). Designed on the computer using DesignSpark, we built our own breadboard, now on a thin 2 x 3 inch plate. This copper plate has integrated wires and channels to replace the wires. The Arduino Nano board and the drivers are mounted onto the PCB using pin headers, so all the electronics are consolidated. Soldering the PCB took a total of 3 hours.

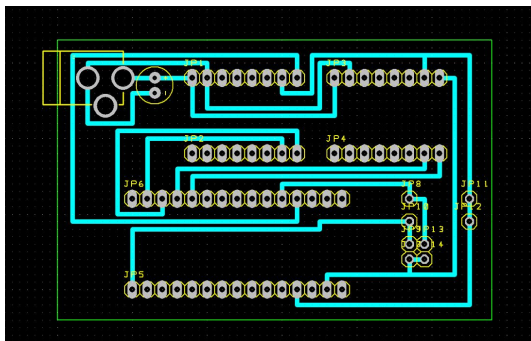




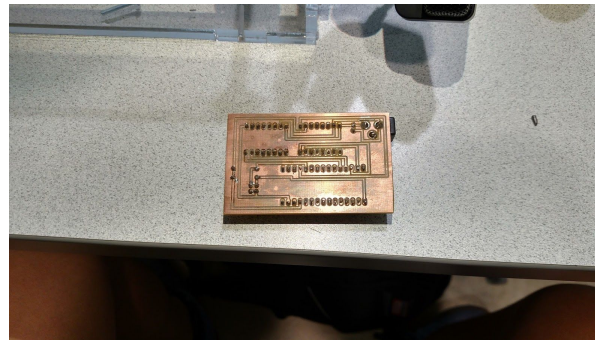
*Breadboard is cluttered with wires*



*Printed PCB is more concise and consolidated*



*Computer model on DesignSpark*



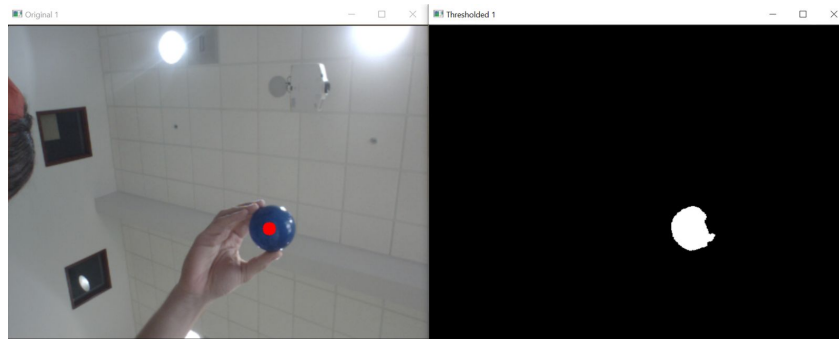
*Soldered female pins on the custom-made PCB*

## **Section 4.0-Software**

### **Section 4.1 - C++**

The most complex part of the robot is tracking the ball's mid-air motion. The computer runs a C++ program developed by our team that takes the data from the two cameras, tracks the ball, predicts the trajectory, and sends the coordinates of the final destination to the Arduino board as Cartesian coordinates.

The C++ program uses OpenCV (Open Computer Vision) for image processing and computer vision. Firstly, it converts the Red Green Blue (RGB) data of the camera to the Hue Saturation Value (HSV) format. It then picks out pixels that match the minimum and maximum HSV values, as calibrated to our ball, by applying a binary threshold. A window displays all pixels that match this data as white while the background is black. The Hough Circle Transform technique can then be used to detect the circle in the image. Finally, the program places an indicator dot over the recognized ball to indicate detection of the ball.



*Converting from RGB to HSV and  
applying a binary threshold*

Once the computer detects the ball on each camera, it converts the pixel coordinates into direction vectors relative to the camera, by utilizing a mapping and calibration technique we came up with. These direction vectors are transformed into the world's coordinate system by rotating them through the angles of the camera's rotation, with rotation matrices. Hence, we obtain two 3D lines, each comprised of a direction vector and a point on the line (the tip of the camera). The 3D coordinates of the ball can then be triangulated, by finding the intersection of these lines or the point where they are nearest.

Using multiple data points, the computer proceeds to make a line of best fit to predict the ball's trajectory and final destination. The coordinates are then sent through serial communication through a USB cable to the Arduino Uno that turns data into motor movement.

One difficulty during software development was changing the code to use the PlayStation™ Eye instead of standard webcams. The PlayStation™ Eye cameras are designed for the Playstation console, so the computer cannot register these cameras as standard web cameras. We were forced to find alternate, commercial drivers from The Code Laboratories to run a camera. These drivers were a hassle to use, but eventually worked.

Serial communication with the Arduino also proved to be much more of a problem than originally expected. The first issue we faced was that updating the target destination of one motor would reset the other motor's. This was fixed fairly easily by rewriting the code in a completely different way. The second issue, which was a lot more difficult to solve, was that there was an inherent delay issue in the Arduino's `Serial.readString` function. After identifying the culprit, we switched to the `Serial.read` function which reads a single byte from the serial port. Our code was modified so that it could read all the bytes individually, and convert them into our desired format.

Because we used two identical Playstation Eye™ cameras as opposed to standard webcams, the computer could not use both cameras simultaneously. Although we found the drivers to run one camera, the software would now not allow two cameras. Downloading a commercial SDK from Code Laboratories allowed us to use both cameras simultaneously in conjunction with our C++ program.

#### Section 4.2-Camera Calibration

Calibrating the cameras to the specific angles was crucial to the robot's design. At first we measured the orientation angles with an electronic level; however, measurement of three axis of rotation was difficult without flat surfaces parallel to the cameras' axis. Then, we centered an object on the camera's image display and measured the distance in the x, y, and z direction to the object. This allowed us to find the exact angles of the cameras. These angles were then placed in a rotation matrix in the C++ code to translate the angles based on the axis of the robot rather than the axis of the camera. While this method did work, we found we achieved much more accurate results when we changed the angle of the cameras to 45 degrees in one direction. Although we had a smaller field of view, the better accuracy produced much better results.

In addition to entering the angles of the cameras into the program, we also had to perform a calibration method to map the pixel coordinates to direction vectors relative to the camera. An object of known size is held in front of the camera so that it covers the entire image, and the distance from the camera to the object is measured. The ratio of camera distance to maximum object size observable can then be found, which is another way of representing the field of view. And finally, when the program is running in real-time, the ratio of the detected circle's pixel coordinates to the image resolution is multiplied by the maximum observable object size, yielding a direction vector relative to the camera.

#### Section 4.3-Arduino

The Arduino code uses the coordinates from the computer through serial communication and actuates the motors to intercept the ball. The coordinates from the computer are in units of motor steps. One complete revolution is composed of 200 motor steps. Our first Arduino program used these commands and processed them as strings from the serial port. The Arduino then ran two methods to move each motor. Each method had a "for loop" for the total number of steps that the motor had to move from its beginning location to its target location. This was problematic because it ran each motor one at a time. Also, because the Arduino ran each motor with a separate method, the Arduino could not take new commands from the computer while the motors were running. This first iteration of the Arduino code was too slow to catch the ball and the computer could not change the coordinates while the motors were running.

Our second iteration ran the motors together by alternating between sending pulses to the X and Y motors. Using “if loops,” we checked if there was a difference between the actual X coordinate and the set X coordinate. If the positions did not correspond, we sent the step pin to “HIGH.” We repeated this step for the Y motor. We finally set both step pins to “LOW” after a delay of one microsecond. The Arduino read one pulse for each loop. Because the entirety of the code was in one looping method, the Arduino reread the information coming from the serial port before going through the motor “if statements.” This allowed the computer to update the coordinates while the motors were running. The problem we ran into with this iteration was a huge delay when the arduino began processing the new coordinates from the computer. This was a major issue because any time the Arduino would receive a new coordinate it would stop the motors to spend time reading the new information.

The third and final iteration reduced the time that the Arduino spent reading the new coordinates. At first we tried reducing the latency times of the computer as well as the twenty millisecond buffer the Arduino had built-in initially to make sure the final byte had been received. We were not able to solve the problem with any of these methods because none of these methods matched the long delay. We then began testing the time the Arduino took to process each command in the program and realized that the delay was caused by reading the String data type. We changed the code to read each byte individually and convert that into a character. This not only left out the time it took the Arduino to read the serial data as a string, but it also let us process each byte as it came in during the gap between bytes coming from the computer. This made the Arduino much faster with a barely noticeable delay.

We finalized our code by making minor tweaks to reassure that the computer and Arduino were communicating effectively as well as compensating for any slipping that occurred during motor acceleration.

When the Arduino code is first starting it would pick up noise signals coming from the computer and interpret them as coordinates, so we had the computer send “-” and the Arduino reply with “#” so we know when we can start sending commands. We also had the Arduino send the coordinates it had received back to the computer for debugging purposes so the operator knew that the computer and Arduino were communicating with each other. To reduce the effects of motor and belt slipping, we added a speed ramp feature so that the motor does not accelerate as quickly. We changed the delay between the “HIGH” and “LOW” stepper commands from 1 microsecond to a parabolic equation.

In the future, we plan to introduce bumper switches at the bottom most right corner so know when the tray hits the outer bounds. When the crate moves to a location, the Arduino assumes made it there without issue. If any slipping of the belt or motor occurs, the actual position will be different than the Arduino position. The bumper switches counteract this by acting as coordinate reset switches that hit the tray after each catch. When the tray hits the switches, the Arduino will reset the coordinates to the known position.

## **Section 5.0-Testing**

### Section 5.1-Software

To test the C++ program, during its beginning stages, we adjusted the acceptable HSV values to identify the ball out of a crowded background. After this calibration, we began to print the coordinates of the moving ball to the display screen to confirm usable data points.

Much later, we tested the serial communication between the Arduino and the computer by sending coordinates and confirmations back and forth. This took trial and error because there was a large delay of communication between the Arduino and the computer. We also realized we needed to get rid of noise signals sent from the computer at the very beginning of running a program.

After construction, we tested the triangulation of the cameras. First, we mirrored the movements of the ball with the tray. We could visually observe the accuracy of the coordinates the robot was receiving as well as the response time of the robot. We then had the computer print out the real-time coordinates that matched the true position of the ball.

We then tested the prediction abilities of the ball by having the computer print the predicted landing location of the projectile. When we knew the coordinates were getting close we finished with throwing the ball.

### Section 5.2-Arduino

To test the Arduino, we mainly ran the program with motors that were not hooked up to the machine. We ran the motors without risk of straining the motor or breaking the frame. This was how we discovered the flaws of our first and second code iterations. This also allowed us to analyze the Arduino processing speed of coordinates and delay before movement. This is what led us to change our second iteration to process bytes at a time rather than Strings.

### Section 5.3- Hardware

Our first test of hardware was running the motors and the arduino alone. This allowed us to ensure that the circuitry connecting the arduino to the stepper motor drivers and stepper motors was correct as well as to calculate the final speed of the robot based on the rotational velocity of the motor. When we were testing the motors we realized that when there was some force against the motor as it was first starting out the motor would slip a couple steps. This was important information to us because we needed accuracy in our motors and we may not have noticed the slipping after the robot was put together. We later fixed this problem by ramping up the speed.



## Section 6.0 References

"CL - About CL Eye Platform - Code Laboratories." *CL - About CL Eye Platform - Code Laboratories*. Code Laboratories, n.d. Web. 26 July 2016.

"Polynomial Regression." - *Rosetta Code*. Rosetta Code, n.d. Web. 26 July 2016.

Cristian, Pantaziu. "Need an Algorithm for 3D Vectors Intersection." - *Stack Overflow*. N.p., n.d. Web. 26 July 2016.

"OpenCV Tutorial C++." *Color Detection & Object Tracking*. N.p., n.d. Web. 26 July 2016.

"Arduino and C++ (for Windows)." *Arduino Playground*. N.p., n.d. Web. 26 July 2016

## Section 7.0 Acknowledgement

The development of the Catch-Bot took weeks of hard work and dedication, and would not have been possible without the support of many individuals, including the Rose-Hulman faculty.

We would like to thank Dr. Song for his expertise and guidance. If it hadn't been for Dr. Song's technical support throughout the development of the project, none of this would have been possible, and we certainly would not be walking away with the amount of knowledge that we are.

We would also like to express our gratitude to the volunteers and assistants who devoted their time, knowledge, and energy towards the implementation and development of this project. Their significant contributions played a big role in the development of the project.

Lastly, we would like to thank Rose-Hulman Institute of Technology for providing us with the resources to make this possible. Resources such as the wonderful staff, state-of-the-art scientific equipment, and materials at Rose-Hulman that guided and assisted us. Their support was crucial in the development of the Catch-Bot.

## Section 8.0 Software

For the convenience of the reader, the lengthy code was not included in this report. It can be found in our repository, hosted on the Github service:

<https://github.com/BaranCODE/BallCatcher>