Design and Implementation of Anti-Ballistic Missile System using Video Motion Detection and a Nerf Gun

California Polytechnic State University

Senior Project for the Physics Department

Author:

Steven Bowman

Advisor:

Dr. Tom Bensky

Spring 2014

Table of Contents

Section	Page
Abstract	3
Acknowledgments	4
1 Introduction	5
2 Experimental Design: Motion Detection and Trajectory	6
2.1 The Camera and Coding of Motion Detection	6
2.2 Trajectory Calculation	8
3 Experimental Design: Launching System	9
3.1 Arduino Mega 2560	9
3.2 Parallax Continuous Rotation Servo Motors	10
3.3 Nerf Elite Firestrike Dart Gun	10
3.4 The ABM Dart Launching System	13
3.5 Complete System	13
4 Physics of Firing Angle and Timing	15
4.1 Determination of Launch Angle	15
4.2 Trigger Delay	17
4.3 Firing Mechanism	17
5 Results	18
6 Conclusion	19
Project Cost	20
Appendix A: Project Flowchart	21
Appendix B: Wiring Diagrams	22
Appendix C: Processing Code	23
Bibliography	29

Abstract

The goal of this senior project was to use a video camera and a dart gun to create an anti-ballistic missile dart launcher. I created a motion detecting and trajectory calculating program with a webcam and linked it to a Nerf dart gun to fire Nerf darts at airborne projectiles. Despite the creation of successful trajectory calculating and dart launching systems, my best efforts have resulted in an inconsistent anti-ballistic system where a very small number of projectiles are actually hit.

Acknowledgments

The inspiration for this project was drawn from several places and it could not have been made possible without contributions from several people. First and foremost the motivation and primary inspiration for this type of project came from my senior project advisor Dr. Tom Bensky and the Physics 357 course I took with him in the spring of 2013. Possibly the greatest course I have ever taken, I was introduced to the world of physical computing and Dr. Bensky made it apparent how powerful and wide a field it was. Intrigued by the incredible possibilities the field held and Dr. Bensky's undying enthusiasm for doing anything in it, I was drawn to choose him for my senior project advisor before I even knew officially what I wanted to do.

Dr. Bensky drew me to the field, however the inspiration for this project in particular came from my uncle Bernard Smith. Bernard, or Bernie as I call him, is a member of the U.S. military and works on anti-missile defense systems. Although he could not give me details about what exactly he does, he told me about how his team works to track an incoming missile and fire an anti-ballistic missile to counter it by disabling the incoming missile. Talking with him about what he does gave me the idea of doing something similar on a much smaller (and less stressful) scale.

In addition to thanking Dr. Bensky for inspiration, I would like to thank him and a few others for the motivation and ability to actually see the project through. My roommate of 4 years Brian Yabuki has been a great help to me in carrying out this project, providing me with materials and supplies I needed throughout the creation process and he was also there to lend me an extra hand when needed. On top of being my supply closet he was also of assistance when I had simple questions about my Processing code. A computer scientist himself, he was able to answer my questions and give me suggestions for ways to clean up my code. I'd also like to thank my good friend from back home Chris Forkner for his assistance with Processing code related problems.

Of course I also need to thank my parents David and Lisa Bowman for constantly nagging on me to get my project done, and also for maintaining an interest in everything I do. Without their financial and moral support I would most definitely not be here in California going to such a great school, pursuing a degree I want and doing things I love.

Chapter 1: Introduction

The concept for this senior project was essentially to create a miniature version of an anti-aircraft gun designed to launch a miniature "missile" at an airborne projectile, hitting it mid air. To accomplish this task required the design, construction, and connecting of two systems: a tracking system and a projectile launching system. Each of the processes for both systems will be described in depth in later sections of this report, but at its core the anti-aircraft system I created focused on producing a projectile tracking system capable of detecting an airborne object, determining the path of its motion, and extrapolating its path through time to give the launching mechanism coordinates for when and where to fire its "anti-ballistic missile."

The motion tracking system was created using a Creative 720p 5.7MP Live! webcam (model VF0790) for video capture and Processing 2.1 (which uses an environment and syntax very similar to Java) to determine motion and calculate a flight path. The ABM launching system was based around a Nerf Elite Firestrike dart gun firing a Nerf dart whose launch angle and trigger were controlled by two Parallax continuous rotation servo motors. Based on information from the webcam, the Processing code was able to send information to an external Arduino Mega 2560 board which controlled the two servo motors to launch the dart at a given angle at a given time. Processing was chosen to do the programming in because of its versatility and compatibility with external devices. Because it is open source, libraries for a wide array of tasks were free and available to use making the connections between the webcam, computer, and arduino/servos possible without having to do a lot of programming.

Figure 1.1: The webcam I used for motion detection.

I chose to do this project because it combines fundamental physics theory, equipment calibration, and physical construction all with a good amount of coding to supplement it. I had little experience with programming in the Processing language prior to this project, but had wanted to give it a try after a previous project I had worked on that used the language. In the rest of this report, I will show how I combined all of the aspects I just mentioned into my final product, and explain in detail how each system interacts with the others.

Chapter 2: Experimental Design: Motion Detection and Trajectory

The first component of the anti-aircraft system was the motion detection system. Initially the thought of some type of echolocation or laser detection was considered to locate a projectile in the air by positioning a detector in real time so that signals were maximized, however after evaluating what that would take to implement and realizing that a video camera could be used to more accurately locate a projectile with less processing power required, the route of video motion detection was taken. To implement the motion detection system I set out a series of checkpoints I would need to reach to successfully track the motion of a projectile.

2.1 The Camera and Coding of Motion Detection

The first step was probably the simplest because of the versatility of the Processing language and that was to get Processing to register the pixels of the webcam and put them into an array for analysis. This was accomplished using the processing.video.* library, which is included with the standard download of Processing 2.1. The library was used to indicate to the program that a video capture device would be connected to the computer and additionally allowed for many aspects of the captured image to be customized. The library allowed for the resolution and frames per second to be specified because the camera had many different shooting options available, and assigned a color value from 0 to 255 to each individual pixel for red, green and blue (RGB values) depending on how much of the color each pixel contained. The color value for each color pixel was then stored into a one dimensional array, where the first indexed value corresponded to the upper left most pixel in the frame, and the last indexed value represented the color value for the bottom right most pixel. This array of pixel values made the detection of motion possible and actually allowed for more control than I initially thought I would have.

Before I had started working with Processing, I thought that I would have to program the system so that it looked for a distinct color (like the yellow color from a tennis ball) and recognize that pixels with that color represented where the projectile was. This would have meant that the camera would have had to be set against a solid background and could only track objects of a certain color in specific lighting conditions. However the pixel array created by the Processing video library for the RGB colors allowed for motion to be detected without strict background parameters. By storing each pixel array and comparing the subsequent pixel array from each new frame to the previous one, a motion detector was created by looking for a difference in the RGB values from frame to frame. The RGB values for each pixel were essentially a three dimensional coordinate (see Fig. 2.1.1) and to determine whether or not there was motion at any individual pixel, the distance between the coordinate for the current frame and the previous frame was calculated and compared to the threshold.

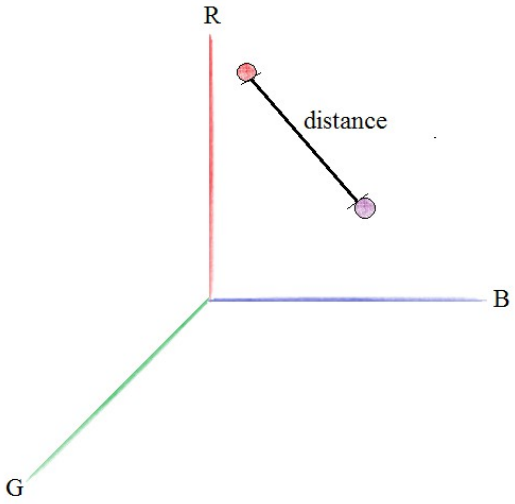


Figure 2.1.1: Distance between two RGB coordinates based on their color

If the difference exceeded the threshold, then the value for that pixel was to be stored as black in a new pixel array that I had set up on the screen to visualize motion known as the canvas in Processing. Pixels where no motion was detected were by default set to display white on the canvas. Figure 2.1.2 shows a screen shot of the motion detector at work. The threshold for motion detection was adjustable to accommodate different lighting conditions or sensitivity to change. As an extra detection buffer, I included code to only count motion in the array of pixels if an adjacent pixel also detected motion, so that the motion detector only displayed motion if at least two subsequent pixels exceeded the difference threshold.

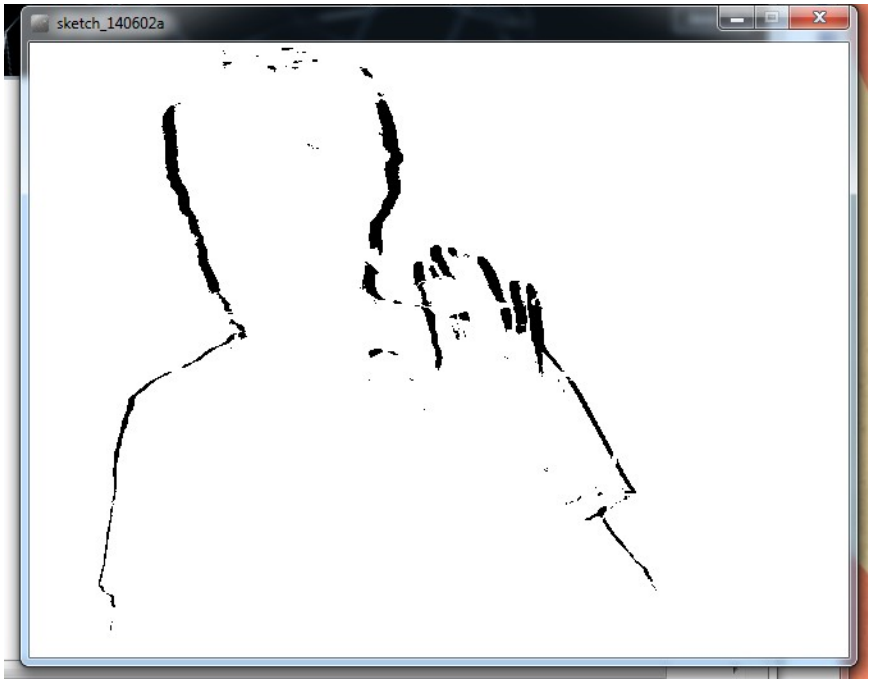


Figure 2.1.2: Screenshot of motion detector as I move my body across the screen.

2.2 Coding Trajectory Calculation

The next step was to track the motion of a projectile in two dimensions. This was done by looking at the pixels in the 1-D array that detected motion and determining an x and y location for each. Then the position of the projectile was determined by taking the average of all the x values and of all the y values for a frame that detected motion and putting the object as the average position of the moving pixels. I had initially wanted to locate the object in three dimensions, and experimented with determining a z-position (distance from the camera) by taking the standard deviation of the pixels (average distance from the center point to determine a pixel radius of the ball) however decided against pursuing this approach because I wanted to get the 2-D version to work first.

Now with the ability to locate the position of the projectile in any given frame in two dimensions I needed to be able to use information from several adjacent frames to determine a path of motion. There will be more on this in sections 4.1 and 4.2 in terms of physics, but as far as programming and implementation goes, I accomplished this task by setting up an integer list for x and y positions. An integer list is like in array in that it stores values, however its size does not need to be preallocated as you can continuously add or remove values from the list. To get a flight path for the projectile, I programmed the list so that it would start accepting values once motion was detected at or beyond a specific x coordinate. I chose the x-coordinate near the edge of the frame to be a sort of trigger to the program so that it knew when to start collecting data points. The triggering occurred when a projectile entered the frame from the left side (although this easily could have been programmed to trigger from the right side as well), and all entries that showed an average x coordinate at or beyond the previous coordinate were recorded. In this way, a series of x and y data points was collected as the projectile traveled across the screen in a specific direction. Once the projectile reached a point near the right edge of the screen, Processing was programmed to begin the determination of when and where to fire the ABM dart.

Details on how the ABM launcher timing and positioning are calculated can be found in sections 4.1 and 4.2. As far as programming goes the timing and launch angle were determined by Processing and stored as values that were sent to the Arduino via the Firmata library for Arduino and a library called cc.arduino for Processing as well as the serial library. These libraries and their functions will be discussed in section 3.1, but at this point the role of the programming is done. For a flowchart of how the complete program works once motion is detected, reference Appendix A.

Chapter 3: Experimental Design: The Launcher

In section 2.2 I described how coordinates for launch angle and timing were determined by Processing using a video capture device when a projectile was thrown across its screen. In this section I will discuss how those coordinates are used by an Arduino Mega 2560, two Parallax continuous rotation servo motors and a Nerf Elite Firestrike dart gun to launch an ABM dart at the projectile and hit it mid air.

3.1 Arduino Mega 2560

I chose to use the Arduino Mega 2560 board primarily because I already owned it and it would allow for communication between Processing and the servo motors I had. To make the board programmable directly from Processing without having to send serial commands to an external Arduino programming environment I used Firmata library for Arduino and the Arduino library for Processing. The Firmata library is a free to use library that is included in the standard download of the Arduino programming environment. It works by uploading a program to the board so that it will respond to commands from Processing through serial information when the Arduino library is included in the Processing environment.

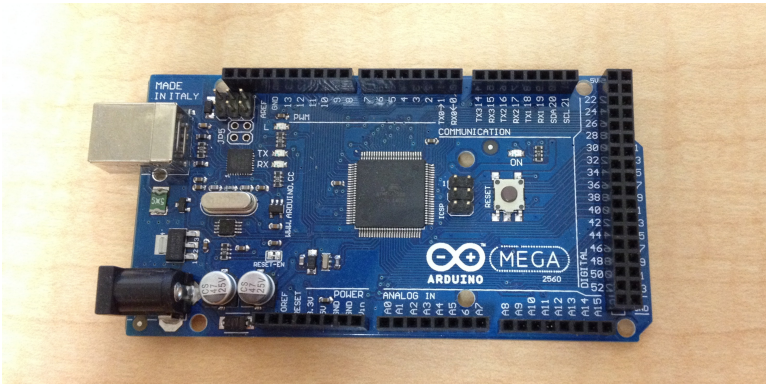


Figure 3.1: The Arduino Mega 2560 board used for the project.

The Arduino library for processing is also free to use but is not included as a library in the standard download of the Processing environment. Instead it can be found at *http://playground.arduino.cc/Interfacing/Processing* and once included in the Processing environment allows for certain commands to be sent to the Arduino from the Processing environment just as if the board was being programmed from the Arduino environment. Lucky for the sake of this project, one of the serial commands the firmware recognizes is the servoWrite function, which allows for a servo motors to be controlled through pulse width modulation out of certain ports on the Arduino Mega board. So by using the Firmata I was able to control the servo motors through the Arduino using commands from Processing without having to upload a new program to the Arduino board from the Arduino environment with directions for the motors once I determined what I wanted the motors to do. This also allowed me to essentially control the motors in real time through a serial port. Without using this type of control system the idea of tracking and shooting a projectile in real time would be unfeasible because of the time it would normally take to upload commands to the Arduino board for the motors.

3.2 Parallax Continuous Rotation Servo Motors

The motors I used to control the launch angle and trigger timing on the Nerf dart gun were Parallax continuous rotation servo motors. Despite being called servo motors, these motors lacked the ability to be set to a specific angle in exchange for the fact that they can rotate continuously in either direction. They were used in this project because of their availability and despite not being able to rotate to a desired angle they completed the tasks that were required of them because their rotation speed and orientation could be controlled. By sending pulses between 1300 and 1700 µsec to the motors, the speed and direction of rotation could be controlled or stopped and thus was used to set the launch angle and trigger timing. Instead of mapping the desired launch angle to an angle the motor must turn through directly, the angle of the motor was adjusted by sending a pulse of 1300 µsec to start the motor turning clockwise, then a delay was calculated to allow the motor to spin for a calculated amount of time to reach a certain angle before finally a 1500 µsec pulse was sent to stop the motor. The delay between starting and stopping pulses is what was mapped to control the launch angle instead of setting the angle directly. A similar process was used for the second motor to control the pulling of the trigger, however it did not require mapping to anything, instead some tests were conducted to see how long it was required to run to pull the trigger before being turned off. A delay time of 300 milliseconds was determined to work well for this motor.

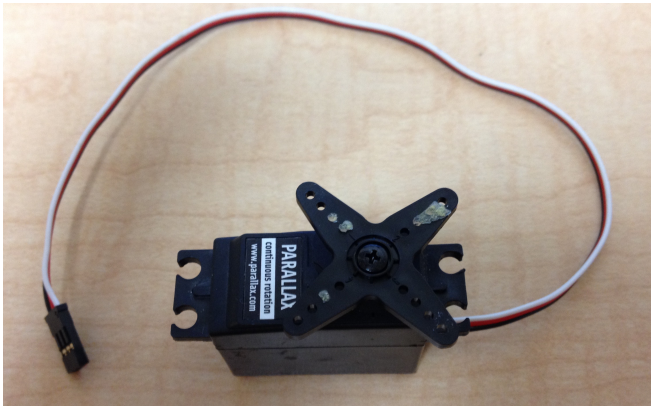


Figure 3.2: A Parallax Continuous Rotation servo motor, two of these were used in the project.

3.3 Nerf Elite Firestrike Dart Gun

With all of the calculations made and desired launching specifications made, the last component of the project was the ABM dart launcher itself. I chose to use the Nerf Elite Firestrike dart gun because of its consistency, power, and price. I picked out the dart gun over several similarly priced options (around $10) because it was single shot and focused on power and accuracy whereas other guns had unnecessary features such as multi-shot capability or larger magazines for quicker fire rates in exchange for the loss of power and accuracy while maintaining the price.

Figure 3.3.1: The Nerf Elite Firestrike dart gun used for this project (unmodified).

To give an idea of how accurately and precise the gun shoots, I set up a quarter on a box against a wall about 10 feet away from the gun being held in place by a couple of other boxes. I sighted it in, so to speak, until the gun would launch a dart that would knock over the quarter. Out of the 10 test shots I fired, the quarter was knocked over 9 times due to either being hit directly or the dart being close enough for the wind drag to knock it over. Even though the packaging for the dart gun implicitly claimed 100% accuracy for firing at objects within 15 feet, I was pleased with this outcome. The accuracy was acceptable to me considering that the projectiles I would be throwing in the project itself were going to be tennis balls with a much larger cross sectional area (diameter of quarter is about 1 inch while the diameter of a standard tennis ball is about 2.7 inches, making the cross sectional area of a tennis ball about 7 times greater) and also because I was going to try and keep the tennis balls within 5 feet of the launcher. This test was conducted before modifications to the gun were made (see Fig. 3.3.3 for the modified gun), however subsequent firings of the gun after modifications did not seem to vary whatsoever from the pre-modified version, most likely because the modifications made were to the frame and not the firing mechanism.

In addition to the impressive precision and accuracy of the dart gun, the power was also quite respectable. Drawing a lot of power from the stiff spring inside of it (see Fig. 3.3.2 and section 4.3) the darts launched from the gun moved at very high speeds, the highest I've ever seen from a Nerf gun, and traveled a great distance. The packaging claimed that the darts could be launched up to 75 feet, however I was not able to confirm that as my apartment most definitely has no enclosed space that long. I will attest that it can launch darts at least up to 20 feet (length of my apartment) and still hit the wall with quite a bit of kinetic energy left to spare. As I mentioned before though, my tests would take place within 5 feet of the gun and so this model provided enough energy to get the dart to projectile with no problems.

Figure 3.3.2: Inside the gun the trigger mechanism can be seen (circled in red) as well as the launching mechanism (circled in green). The modified launcher (Fig. 3.3.3) alters the trigger mechanism.

Figure 3.3.3: The modified gun. The trigger mechanism seen in Fig. 3.3.2 has been altered so that instead of the spring being released by the pull of the trigger, the white component circled in red in Fig. 3.3.2 is released when the servo motor attached to the gun rotates. In addition to this, holes have been drilled through the gun to enable it to be fixed to the launcher housing.

3.4 The ABM Dart Launching System

The components for the final product were described in earlier sections. Now I want to show how they were put together and modified to produce the final ABM dart launching system. Beginning with Nerf gun, I opened it up and removed any unnecessary and easily removable components to try and minimize the amount of weight the angle determining servo motor would have to move. I could not remove the launching mechanism from the frame and isolate it to minimize weight for several reasons, the first of which was that the spring used in the launching mechanism actually relies on the frame to press against (see Fig. 3.3.2). Second and more vital was that I could not drill holes in the launching mechanism to fix the motors to without destroying to some degree its ability to launch darts; instead it would have to remain in the frame of the gun which could have holes drilled in it without affecting performance.

To fix the angle determining and trigger pulling motors to the gun, holes were drilled through the frame and the motors were attached using these holes as can be seen in Fig. 3.4.1. After attaching the motors, the gun was placed in a custom cut cardboard housing with some stabilizers so that it was free to move about the axis of the angle determining motor. With the launcher and motors set up, the motors were connected to the PWM pins of the Arduino Mega 2560 board, which was then connected to the computer.

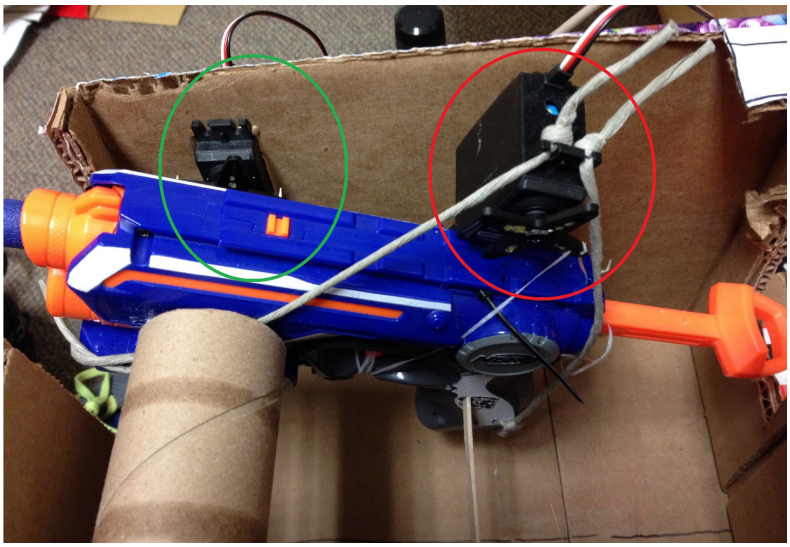


Figure 3.4.1: Nerf gun with servo motors attached and placed in its housing. The servo motor circled in green controls the launch angle and the servo motor circled in red releases the trigger.

3.5 Complete System

The last step in the creation of the anti-aircraft system was the marriage of the ABM dart launching system and the motion detection/trajectory system. I have already outlined how each system works individually, but the harmonizing of the two systems was really the crucial step in determining whether or not the whole thing would work. To do so required calibrating pixel sizes to actual lengths at different distances from the camera, determining the distance between the edge of the camera frame and the launcher's line of fire as well as the height difference between the camera frame and the launcher, and also any delay due to processing times or camera lag.

To account for each of these parameters, variables were created in the Processing code that could be adjusted to accommodate the specific conditions created by the camera placement relative to the launcher and projectile being thrown. These parameters can be found in the code in Appendix C and their determination can be found in chapter 4. The inclusion of these parameters however effectively concludes the design portion of this project.

Here is a link to a youtube video that shows a physical outline of the system:

http://www.youtube.com/watch?v=o30gCz9KoR4

Chapter 4: Theory and Implementation

The two key parameters produced by the Processing code and given to the arduino via serial communication are the launch angle of the Nerf dart gun and the trigger release timing. To determine these values there were a number of variables to consider.

4.1 Determination of Launch Angle

The determination of the launch angle comes from basic kinematics and a bit of situational analysis.

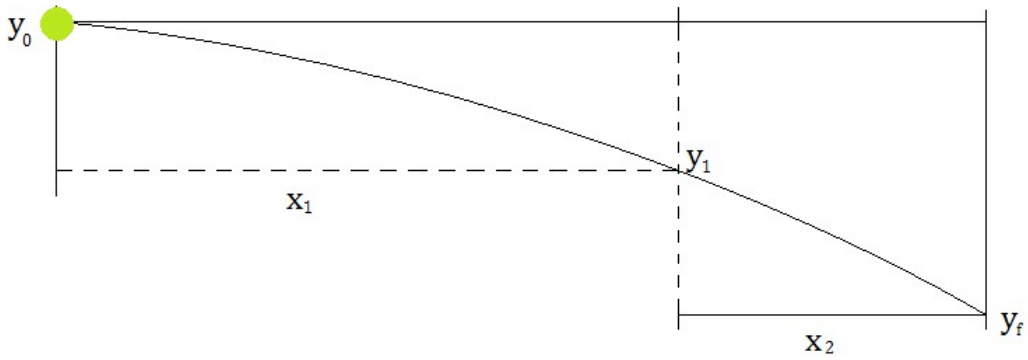


Figure 4.1.1: Theoretical trajectory of tennis ball. The distance spanned by x_1 represents the frame of the image captured by the camera.

As the ball travels across the captured image from the left boundary to the right, snapshots of its position are captured every 1/30th of second and the equation to find the desired height of the launched ABM dart is found as follows: call the height that the ball enters the image at on the left boundary y_0, the distance the ball travels across the frame of the camera x_1, the height of the ball at the right boundary of the camera y_1, the distance the ball travels from the boundary of the camera to the line of fire for the ABM dart x_2, and the final height of the ball once it has reached the line of fire for the ABM dart y_f. To get y_f, start with the equations

$$y_1 = y_0 + v_{iy} t_1 - \frac{1}{2} g t_1^2 \quad \text{and} \quad y_f = y_1 + v_{fy} t_2 - \frac{1}{2} g t_2^2 \qquad \text{(Eq.'s 1 and 2)}$$

where v_{iy} is the vertical velocity of the ball at the left boundary of the frame, v_{fy} is the vertical velocity of the ball at the right boundary of the frame, t_1 is the time it takes for the ball to travel across the entire captured image of the webcam (found by counting the number of captured frames while the ball is on screen and dividing it by the frames per second) and t_2 is the time it takes for the ball to travel from the right boundary of the captured image to the line of fire for the ABM dart (more on this in section 4.2). These equations are both drawn from basic kinematics and for the purposes of this experiment where the ball faces little air resistance, does not travel near the speed of light, and lies in a near uniform gravitational field, work fine.

Since we are trying to calculate y_f and already know y_1, y_0, and can calculate t_1 and t_2 from other known information (which we'll look at in section 4.2), all we have left to do is find

v_{fy} which comes from v_{iy} and one more equation. In this instance, v_{iy} can actually be solved for fairly easily by just using some algebra to find that

$$v_{iy} = \frac{y_1 - y_0 + \frac{1}{2} g t_1^2}{t_1}, \qquad \text{(Eq. 3)}$$

and then by using the relation that

$$v_{fy} = v_{iy} + at, \qquad \text{(Eq. 4)}$$

where a is $-g$ and t is actually t_1, giving that the final equation for the height of the ball once it reaches the line of fire for the ABM dart is

$$y_f = y_1 + \left[\frac{y_1 - y_0 + \frac{1}{2} g t_1^2}{t_1} - g t_1 \right] t_2 - \frac{1}{2} g t_2^2. \qquad \text{(Eq. 5)}$$

This method works as long as y_0 and y_1 are set relative to some coordinate on the captured image that can be identified, such as setting the bottom of the captured image to zero. Notice though, that the ball could have gone from the same starting and ending points on the screen in a different path like this:

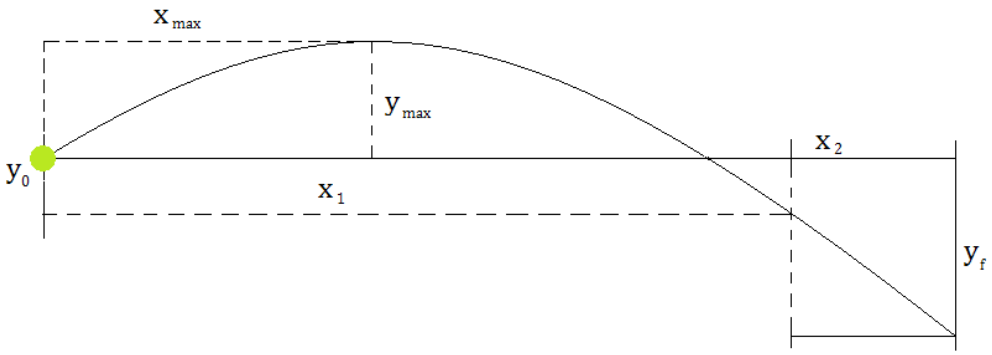


Figure 4.1.2: Alternate trajectory of tennis ball passing through same points on the left and right boundaries.

It would seem as if a different series of equations would be needed because the ball takes a completely different path. However the same equations will still apply only the time it takes the ball to travel across the screen will change. Fortunately the determination of the time it takes for the ball to travel across the screen is calculated separately, meaning that regardless of the situation, Eq. 5 will still hold.

As far as intercepting the dart with the trajectory path of the projectile, I chose to go a more practical route instead of including more equations in the Processing code. Certainly more kinematic equations could be used if the exact position of the dart gun was known in units of pixels, however instead of trying to include these equations I just mapped the desired launch height to a position of the motor. As I discussed in section 3.2, the launch angle of the dart gun was determined by a delay for how long the servo motor should run. Therefore instead of determining theoretically where the dart would strike in terms of a pixel, I used the mapping

function of Processing to produce a delay on the motor that would launch the dart to the desired height. Practically speaking however, if I were to have included equations of motion for the dart traveling over that short of a distance they would have essentially had the dart traveling in a straight line because of how fast the dart travels in that realm (less than 10 feet from the barrel) compared to the speed at which the tennis ball is moving.

4.2 Trigger delay

The second and final piece of information needed by the arduino to correctly align and shoot the ABM dart at the projectile is when to pull the trigger. The timing for the delay between the projectile leaving the screen and the pulling of the trigger is calculated by using the horizontal velocity of the projectile while it is on the screen with the distance from the right boundary of the screen to the line of fire. By dividing the distance between the right boundary and the line of fire by the velocity of the projectile a time delay is produced.

This time delay does not consider things like video capture lag, processing time and time for the dart to travel to the ball. To account for all of these parameters I included a single delay correction in the information for when to fire the dart. Practically speaking the time it takes for the dart to reach the ball is insignificant because as I mentioned previously the launcher really sends the dart screaming out of the barrel. This means that the majority of the correction term is offsetting the time it takes for Processing to gather all of the information and produce the values I requested from it. I found that the camera lag was pretty bad when I ran the camera at its maximum resolution (1280 x 720 pixels at 30 fps) but was unnoticeable when running it at 640 x 480 and 30 fps.

As an added note of practicality, because of the time delay between gathering, analyzing and finally relaying the information to the trigger, the launching mechanism had to be physically below the bottom of the captured image on the webcam and well away from it horizontally. Without adding some distance between the camera and the launching mechanism and taking into consideration the lag that I have mentioned, the trigger timing delay sent to the launching mechanism actually turned out to be negative sometimes, meaning that by the time the information got to the trigger servo it was being told it needed to have already fired the dart. This distance could of course be minimized by using computers with faster processors, although the computer I was using throughout the project had a 3.40GHz i5-3570K processor and was certainly not too slow by any means, or by increasing the speed at which the dart left the gun.

4.3 The Firing Mechanism

I have noted several times throughout this report about how powerful the Nerf Elite Firestrike is. The launching mechanism in the Nerf Elite Firestrike works by rapidly forcing air into the base of a dart using a spring and plunger to move the air. Figure 3.3.2 shows the inside of the gun where the launching mechanism can be observed. It is essentially like a bike pump on steroids, using the spring to move the plunger through the white chamber extremely fast creating a pocket of high pressure. Because the plunger tightly seals the white chamber, the easiest place for the pressure to even out is at the base of the Nerf dart, dealing a very large force over a small amount of time and area to send the dart hurtling out of the barrel of the gun.

Chapter 5: Putting it all together; The Results

With everything set in place, program written, launcher constructed, I had one last task and that was to start calibrating and entering values to the code. This process proved to be extensive as I worked with a tape measure and the webcam to try and convert physical lengths into pixel values. I produced several good results, the most important of which was that the ratio of distance seen horizontally on screen to the distance away from the camera was 0.91. From this I found the ratio of pixels/meter as a function of distance from the camera by dividing the number of horizontal pixels (640) by the aspect ratio of 0.91 multiplied by the distance away from the screen in meters.

With that information in hand, I set up the launcher below and away from the webcam and found where the edge of image was relative to the launcher. After inputting values to accommodate for the height difference and separation between the webcam and line of fire, I began tossing tennis balls and tried to map motor stop delays to launch heights and trigger timings. Unfortunately what I found was that the process was not working too well.

Despite both of the systems working together, I was unable to actually hit a tennis ball consistently in mid air. I know each system was working because I was able to get the launcher to fire a dart in response to the tennis ball traveling across the screen, I was just unable to correctly map the launching angle and timing to the parameters of each throw on a regular basis. There were successful attempts, however the vast majority of attempts were just close calls. Part of the problem lies with the motors I used. The motor controlling the launch angle struggled to position the gun with consistency. I tried to build a housing unit that would allow for the motor to freely control position of the launcher, however my attempt was not enough for the servo motor as it struggled to consistently put the launcher at the same angle despite similar manual inputs.

The trigger timing too was not exactly spot on. This was about as disappointing as the failure of the angle determining system because I spent a lot of time on getting this system to function at all. It required me at one point to bore a hole through really hard plastic using a dental pick and then carefully thread dental floss through the hole and tie tiny knots in it. After enduring the painstakingly meticulous work to replace the trigger system I had hoped this system would have worked better. In all honesty it works pretty well, its just that the window of time for when the trigger must be pulled is very small (around 0.03 seconds if the ball has a horizontal velocity of 2 m/s and it is assumed that the launch angle would orient the dart at the ball's widest point). As I mentioned before, factoring in things like camera lag as well as time it takes for the motor to get the trigger to release makes this window of time very difficult to hit.

Despite the project not actually working as well as I had hoped, here is a link to a video of the system functioning as best as it could:

https://www.youtube.com/watch?v=Mdb1LqhubX0

Chapter 6: Conclusion

Despite the lack of success in actually shooting a ball out of the air I was really happy with how the project turned out. Having very little experience with programming in Java type languages I was concerned I wouldn't be able to grasp the language fast enough to be able to complete both components of the project (the coding and the designing/construction). I learned a lot about coding in Processing and feel like I have a much better grasp on Java like languages after this project. I was also really pleased with how well the physical parts of my project turned out. The housing I built for the launcher (the second time through) was actually really sturdy and provided a lot of support for the launching mechanism as a whole. Additionally I feel like the entire launching system, gun and housing, was put together well considering the supplies I had available.

As I mentioned before the project as it is does not currently work up to my expectations, which would be hitting the ball consistently. I would not however consider the project a failure, as I believe it could be fixed with one of several modifications. The easiest and most obvious choice would be to shoot at larger projectiles. Increasing the size of the target would increase the duration of the trigger firing window and also decrease the strictness on the launching angle. If I were to continue working on this project, I do not believe I would settle for this though. Instead I would keep the same goals and focus on better execution of the systems. Most likely this would start with the replacement of the continuous rotation servo motors for actual servo motors. Additionally I would buy fishing line to use instead of dental floss and would develop better ways to fix the motors onto the frame of the launcher. Of course I would probably also try to create a better housing unit out of materials that aren't cardboard.

Looking forward, if I continue on with this project I hope to ultimately make it work in three dimensions. Currently I only have it programmed to work in the case where the ball is thrown so that its distance from the camera remains somewhat constant. I mentioned in section 2.2 that I had wanted to include a method to measure the distance from the camera based on pixel information, and actually intended to do so before I realized it was beyond the scope of this project. The last foreseeable change I would make to the program would be to replace the current trajectory calculating bit of code with a least squares approximation method. That way the model for the trajectory of the projectiles would be more accurate.

Project Cost

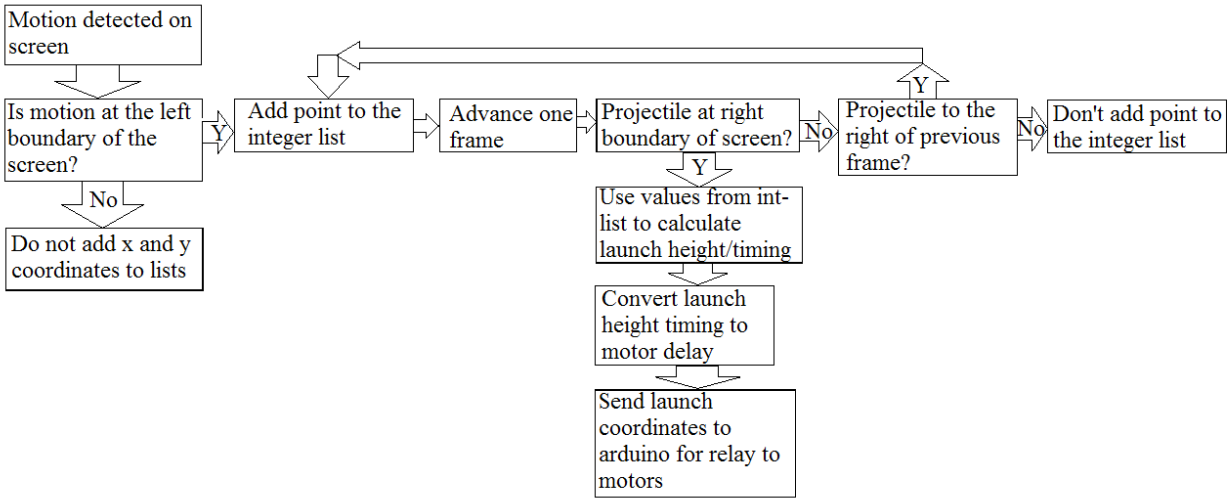
Although there were not an exceedingly large number of components for this project, the price of the components I purchased added up rather quickly. While I did try to keep costs as low as possible, there were several components I purchased that have cheaper alternatives without sacrificing functionality, I will detail those below.

Component	Cost	Comments
Creative Live! Webcam	$22.00	Bought off Amazon, could probably find cheaper webcam with similar specifications
Arduino Mega 2560 Board	$39.00	Bought off Amazon, the Uno board could accomplish the same tasks for about half the cost, I just wanted the Mega for later projects.
Parallax Continuous Rotation Servo Motor	$14.00 (2x)	Provided to me by Dr. Bensky, would not recommend using these motors for this project, instead look for actual servo motors.
Nerf Elite Firestrike Dart Gun	$10	Got it at Target, would recommend using this type of launcher for any project requiring the launching of some kind of projectile.
Total:	~$100	This is not actually what I paid because of Dr. Benksy providing me with motors, and this could be reduced by seeking cheaper alternatives.

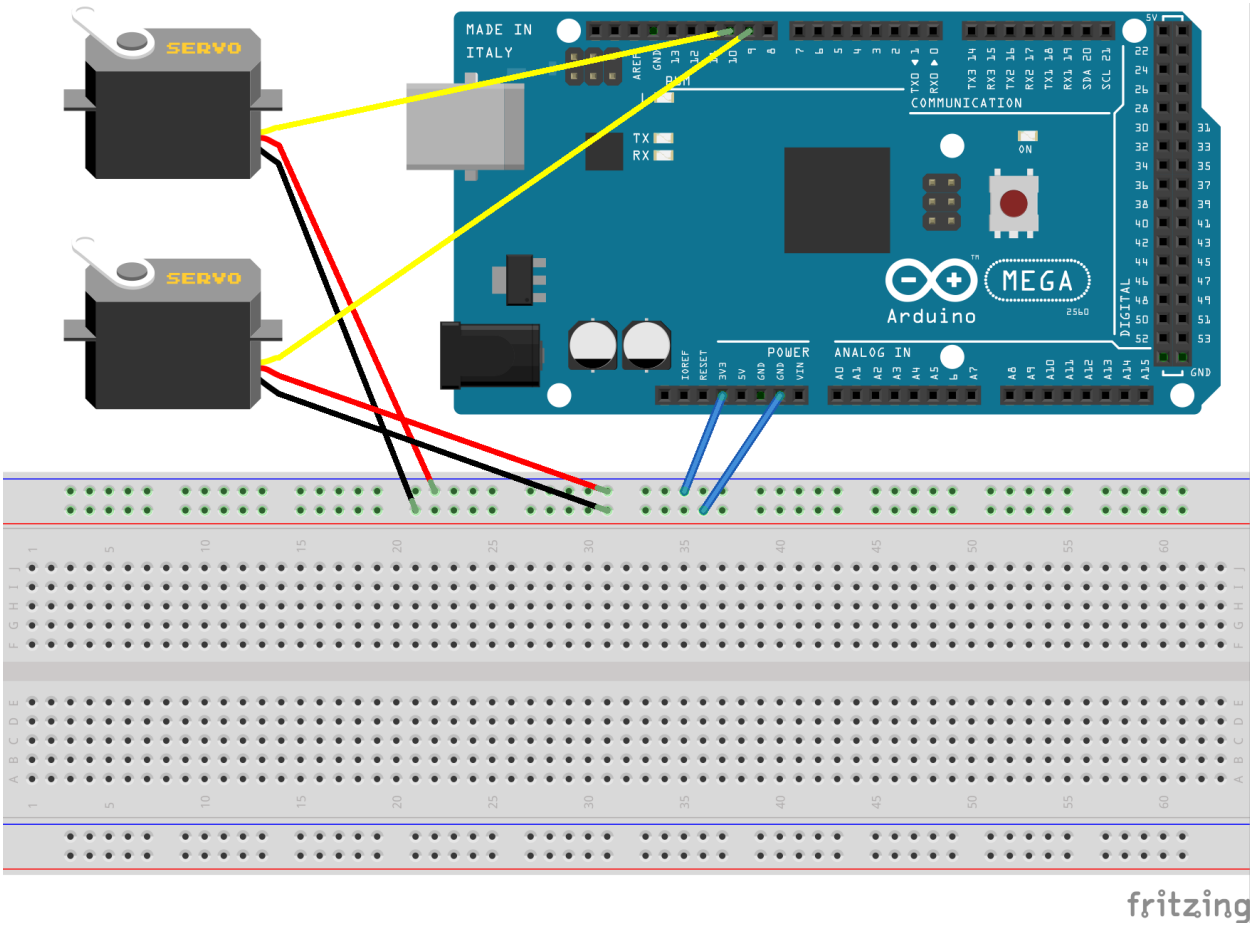
In addition to the major components I also used things that I had lying around my apartment or that could easily be found like cardboard boxes, duct tape, toothpicks, etc..

Appendix A: Project Flowchart

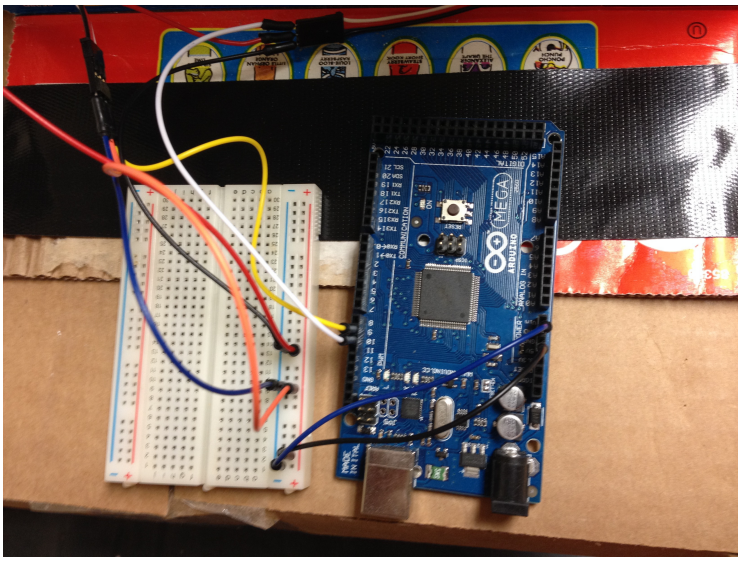
Once motion is detected by the Processing code, the following sequence of events occurs to create a series of data points that are analyzed to determine if they are projectile motion and if so, the data is converted into coordinates for the launch angle and launch timing for the Nerf ABM dart gun.



Appendix B: Wiring Diagrams



Here is the wiring diagram for the servo motors into the arduino. Notice how pins 9 and 10 have pulse width modification (PWM) capabilities. This allows for precise control of the signals being sent into the two servo motors so that their rotation speed and direction can be controlled. Beyond that, the wiring for this project is fairly limited and straightfoward.



Appendix C: Processing Code

Below is the code I ran in Processing to implement my project. It should also be noted that the standard firmata sketch needs to be uploaded to the arduino board being used for this code to work properly. I have not included that code here because it can already be found under examples in the arduino programming environment.

```
//Inlcude libraries

import processing.video.*; // allows for video capture
import processing.serial.*; // allows for communication with arduino
import cc.arduino.*; // allows the arduino to be programmed in processing

//Create objects

Arduinoarduino;// creates an arduino device
Capturevideo;// creates a video capture device
PImage prevFrame; // variable name for previous frame

//Create variables, arrays, intlists to be used in calculation of firing height and
//time

volatile int[] xvals = new int[640*480];
volatile int[] yvals = new int[640*480];
volatile int[] location = new int[640*480];
volatile int xcoord;
volatile int ycoord;
volatile int xsum;
volatile int ysum;
volatile int xsize;
volatile int ysize;
volatile float r1,r2,g1,g2,b1,b2, diff;
IntList xvalue;
IntList yvalue;
volatile int xarraysize;
volatile int yarraysize;
volatile int oldx;
volatile int oldy;
volatile int xvel; // pixels/msec
volatile int xdelay; // msec, the time to wait before firing the dart
volatile int time1; // time for ball to travel through the frame of camera
volatile int time2; // time to reach the launch path
volatile int control; // stops the motors from spinning after they have completed
                      //their task

//Create variables to represent physical parameters that can be adjusted

float threshold = 100; // limit for how different a pixel's color must be to
indicate motion
volatile int dist; // dist from cameras 0 pixel to launch path, in pixels
volatile int lag; // msecs of lag
volatile int fireheight; // desired y pixel location to be fired at a distance
"dist" away
volatile int turntime; // time for the angler to rotate before stopping
volatile int pixelratio; // pixels/meter
```

```
volatile float g; // gravity in pixels/sec^2
volatile float z; // distance from camera ball is thrown in meters
volatile float aspectratio; // ratio of horizontal distance seen in frame to
distance from lens
volatile int camoffset; // how much higher the camera is than the launcher

void setup() {
// Initialize camera and create canvas on computer screen

size(640,480); // creates a visual canvas on the screen to view image captured by
camera
frameRate(30); // captures and displays 30 frames per second
video=new Capture(this,width,height,30);
prevFrame=createImage(video.width,video.height,RGB); // Create an empty image the
same size as the video
video.start();

// Initialize Arduino and designate servo ports
println(Arduino.list()); // View available arduino devices
arduino=new Arduino(this, Arduino.list()[1], 57600);
arduino.pinMode(9,Arduino.SERVO);
arduino.pinMode(10,Arduino.SERVO);
// Set rotation speed of servos to zero
arduino.servoWrite(9,90);
arduino.servoWrite(10,90);

// Initialize x and y arrays
xvalue=new IntList();
yvalue=new IntList();
xvalue.append(640);
yvalue.append(0);

// Input physical parameters (requires measuring and calibration)
dist=1000;// pixels
lag=200;// Adjust to accommodate for processing time, dart movement time, etc.
z=1;// Adjust based on how far the ball will be thrown from the camera
aspectratio=0.91;// Measured for my camera using a tape measure and dimensions of
the frame
pixelratio=round(640/(aspectratio*z)); // pixels/meter
g=9.8*pixelratio*0.000001;// pixels/msec^2
control=0;// Allows fireheight and firetime to only be calculated once

}

void draw() {

// Capture video
if (video.available()) {
// Save previous frame for motion detection
prevFrame.copy(video,0,0,video.width,video.height,0,0,video.width,video.height); //
Before we read the new frame, we always save the previous frame for comparison
prevFrame.updatePixels();
video.read();
}

// Get access to the pixels for the current and previous
```

```
loadPixels();
video.loadPixels();
prevFrame.loadPixels();


// Begin loop to walk through every pixel and look for motion

for (int x = 0; x < video.width; x ++ ) {
for (int y = 0; y < video.height; y ++ ) {

int loc = x + y*video.width;            // what is the 1D pixel location
color current = video.pixels[loc];      // what is the current color
color previous = prevFrame.pixels[loc]; // what is the previous color

// compare colors (previous vs. current)
r1=red(current); g1 = green(current); b1 = blue(current);
r2=red(previous); g2 = green(previous); b2 = blue(previous);
diff=dist(r1,g1,b1,r2,g2,b2);

// How different are the colors?
// If the color at that pixel has changed, then there is motion at that pixel.
if (diff > threshold) {
// If motion, display black on the canvas
pixels[loc] = color(0);
location[loc]=loc;// If there is motion at this pixel, store the pixel number


} else {
// If not, display white on the canvas
pixels[loc] = color(255);
location[loc]=0;// If there is not motion at this pixel, the value at this pixel
location is 0
}
}
}

// Begin loop to determine what pixels detected motion and go from one 2-D array to
two 1-D arrays for the x and y coordinates

for ( int i=0; i<location.length-1; i++) {
if (i<video.width) {
xvals[i]=location[i];// for the first line of pixels the x location is just the
value of pixel where motion was detected
yvals[i]=0;// for the first line of pixels all of the y-coords are just 0
} else {

// The x and y coordinates of the pixels where motion was detected are determined
by division and remainder of the pixel number by the video width,
// the y value is the number of times the pixel number can be divided by the video
width and the x coordinate is the remainder.
// Also a pixel is only determined to have motion detected if either the pixel in
front of it or behind it had motion detected as well.

if (location[i-1] != 0) { // If the previous pixel detected motion
xvals[i]=location[i]%video.width;
yvals[i]=location[i]/video.width;
```

```
}else if (location[i+1] != 0) { // If the next pixel detected motion
xvals[i]=location[i]%video.width;
yvals[i]=location[i]/video.width;
}else {
xvals[i]=0;
yvals[i]=0;
}
}
}
// Proceed to the next frame
updatePixels();

// Determine the average x and y values of all the pixels where motion was detected

// Initialize and reset the sums to determine the average x and y coordinates
xsum=0;
ysum=0;
xsize=0;
ysize=0;

// Sum up all of the x values that had motion and see how many values there are
for (int i =0; i < xvals.length; i++) {
xsum+=xvals[i];
if (xvals[i] > 0) {
xsize+=1;
}
}

// Sum up all of the y values that had motion and see how many values there are
for (int j=0; j < yvals.length; j++) {
ysum+=yvals[j];
if (yvals[j] > 0) {
ysize+=1;
}
}

// Determine the average x and y coordinates assuming there was more than 0 pixels
where motion was detected, otherwise set the averages to 0
if (xsize != 0) {
xcoord=xsum/xsize;
}else {
xcoord=0;
}
if (ysize != 0) {
ycoord=ysum/ysize;
}else {
ycoord=0;
}

// Add elements to arrays to determine trajectory of projectile
xarraysize=xvalue.size();
yarraysize=xvalue.size();

// Get previous frames x and y coordinates
oldx=xvalue.get(xarraysize-1);
oldy=yvalue.get(yarraysize-1);
```

```
// If the x coordinate is not 0 and is less than the previous x coordinate, add it
to the list as well as the y coordinate
// this could be reversed if the projectile was thrown from the opposite direction,
but presumably the first average x coordinate
// will be coming from the right side and so subsequent x values should only be
added if they are less than the previous
if (xcoord > 1) {
if(xcoord <= oldx) {
xvalue.append(xcoord);
yvalue.append(ycoord);
}
}

// Default these values to 1 if the next loop is not entered
time1=1;
time2=1;

// Once the object has reached the left side of the screen, begin to calculate its
trajectory, the 40 could be changed depending on where
// the limit for "left side of the screen" is determined to be
if (oldx < 40) {
if (control == 0) {
control=1;// makes it so the following command sequence only runs 1 time through so
motors only react once
time1=((xarraysize-1)*1000)/30;// msec
xvel=1000*(xvalue.get(xarraysize-1) - xvalue.get(1))/(time1); // pixels/msec
time2=1000*dist/xvel;// msec
xdelay=time2-lag;//msec, the time it will take for the projectile to reach the line
of fire minus the lag

// Determine fireheight using kinematic equations
fireheight=round(yvalue.get(yarraysize-1) + ((((yvalue.get(yarraysize-1)) -
yvalue.get(0) + 0.5*g*(time1)*(time1))/time1) - g*time1)*time2 -
(0.5*g*time2*time2));

// Diagnostic lines of code to determine if system is behaving properly
print(time1);
print(",");
print(xvel);
print(",");
print(time2);
print(",");
print(xdelay);
print(",");
println(fireheight);


// Finally, map the fireheight and trigger timing to times to allow the motors to
spin

// Get value for how long to spin the launching angle motor
turntime=map(fireheight,start1,stop1,0,300); // start1 and stop1 are adjusted based
on the calibration of where the
// max and min

arduino.servoWrite(10,0);// Begin turning the launching angle motors
delay(turntime);// Allow it to spin for calculated period of time
```

```
arduino.servoWrite(10,90);// Stop the motor at its desired position

// Once the launch angle is set, fire the dart at the desired time
if ((xdelay-turntime)>0) { // If the time it took for the first motor to turn
wasn&amp;apos;t too long, fire the dart
delay(xdelay-turntime);// Have to subtract the time it took for the first motor to
spin
arduino.servoWrite(9,180);// Begin rotating the trigger motor
delay(300);// Allow the trigger motor to rotate enough to pull the trigger
arduino.servoWrite(9,90);// Stop the trigger motor from rotating
}else { // If the time it took for the first motor to turn to the right angle
won&amp;apos;t allow the dart to reach the projectile in
// time, don't fire it
arduino.servoWrite(9,90);
}

// If the program has already run its course, keep the motors in the stopped
position
}else {
xdelay=0;
fireheight=0;
arduino.servoWrite(9,90);
arduino.servoWrite(10,90);
}
}
}
```

Bibliography

McComb, Gordon. “Parallax Continuous Rotation Servo.” *Learn.Parallax.com – Parallax Inc.* 2012. Web. <http://learn.parallax.com/KickStart/900-00008>

Shiffman, Daniel. “Simple Motion Detection.” *Learning Processing – A Beginner's Guide to Programming Images, Animation, and Interaction.* 2008. Web. <http://www.learningprocessing.com/examples/chapter-16/example-16-13/>

“Arduino and Processing.” *Arduino Playground.* N.p., n.d. Web. <http://playground.arduino.cc/Interfacing/Processing>

“Firmata Library.” *Arduino Playground.*N.p., n.d. Web. <http://arduino.cc/en/reference/firmata>

“IntList.” *Processing.org*. N.p., May 17 2014. Web. <http://www.processing.org/reference/IntList.html>