**Autonomous Operation and Face Detection for Stepper Motor-Controlled Projectile Device (Nerf Gun)**

*Final Project – Advanced Engineering Computation[[1]](#footnote-1)*

*Spring 2018*

Clay Kettlewell\*, Kyle Hejtmanek\*, and Jedrzej Kosinski\*\*

Faculty Advisor: Jesse Lowe\*

\*University of Missouri Kansas City, Department of Electrical and Computer Engineering

\*\*University of Missouri Kansas City, Department of Computer Science

INTRODUCTION

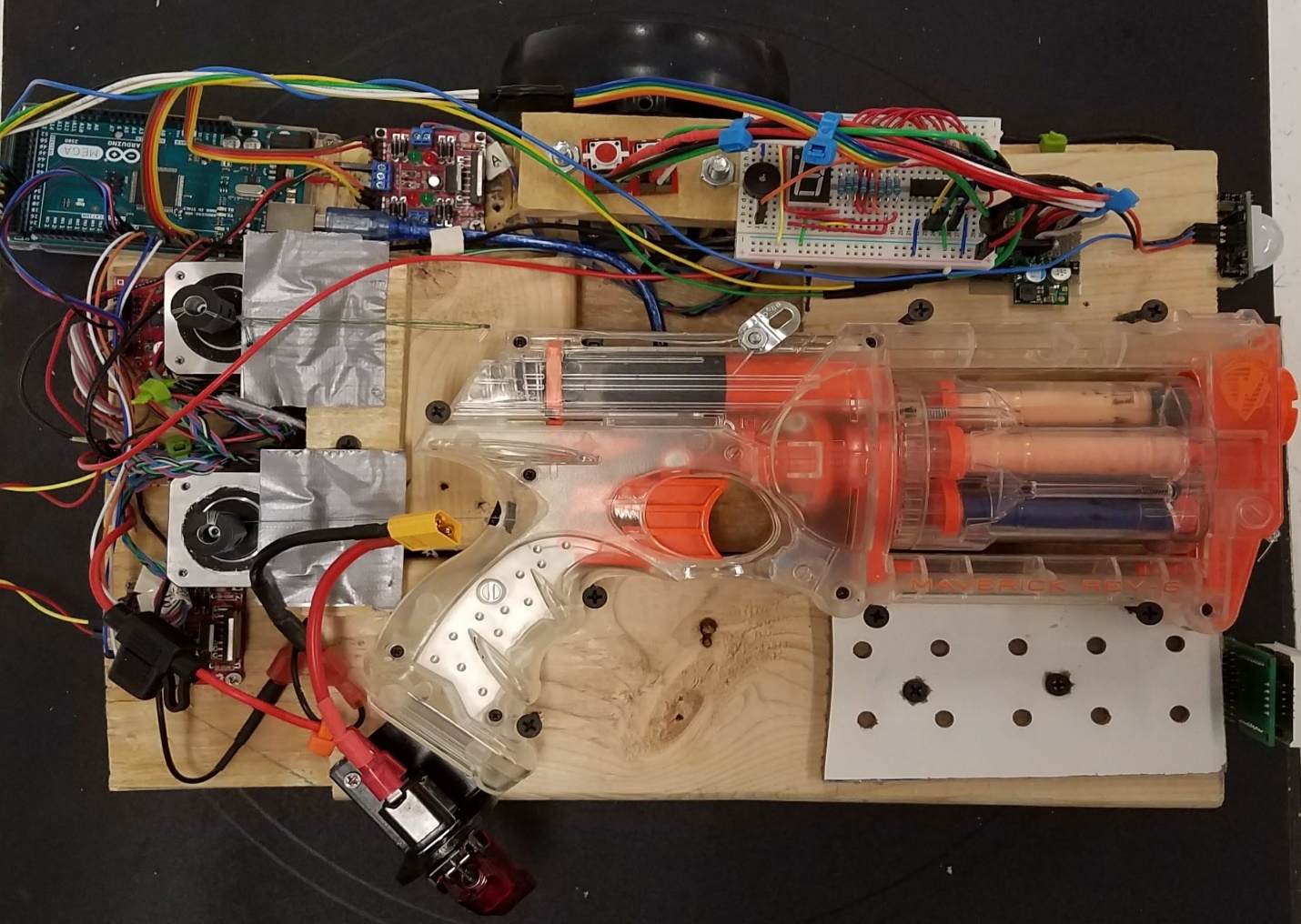
The objective for this project was to better understand the facial detection features of the opencv library using the python language, and to use a Raspberry Pi 3b+ to implement autonomous functionality into a harmless projectile device which targets detected faces.[[2]](#footnote-2)

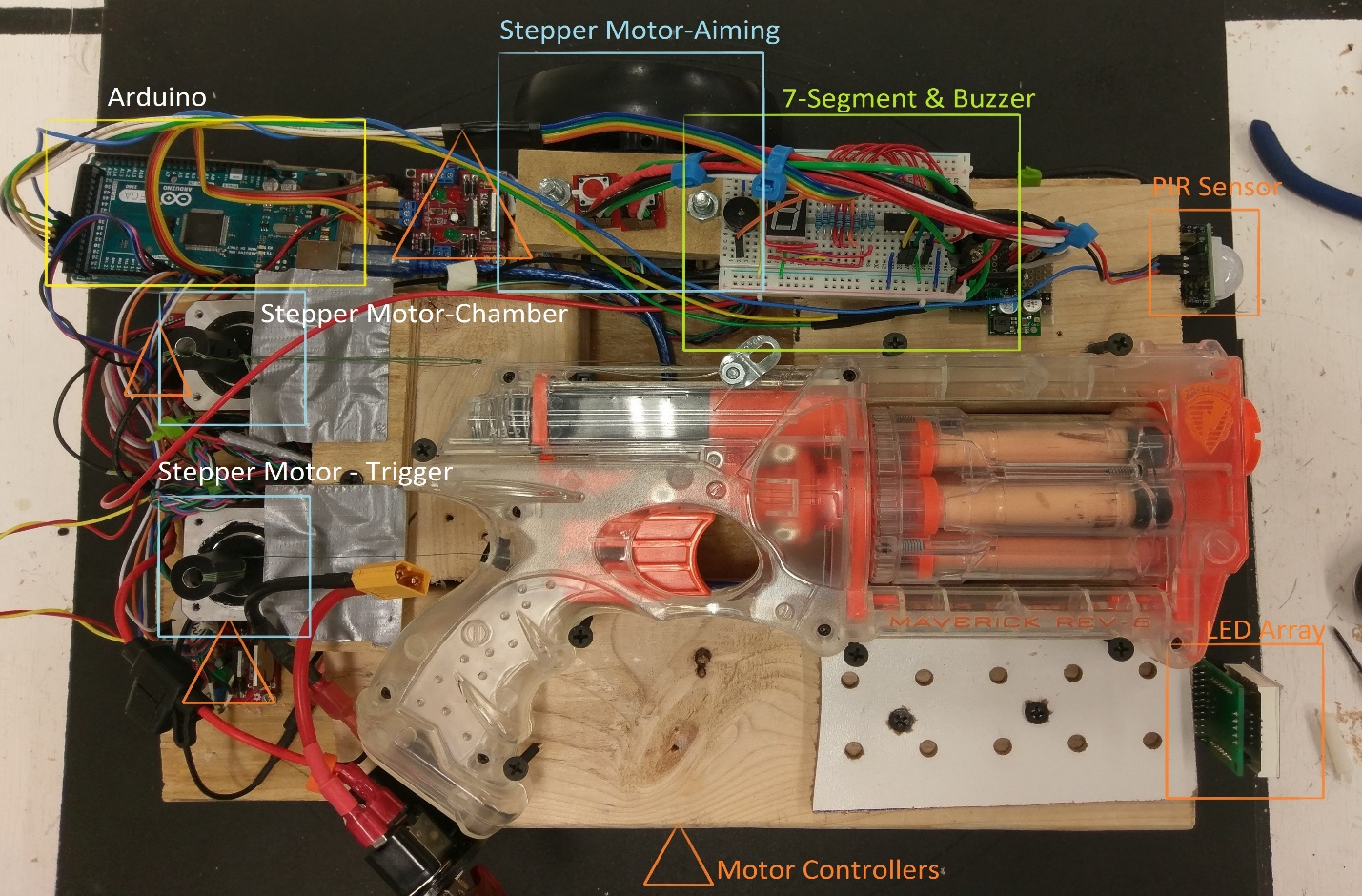
The bulk of this project (40+ hours,) was spent learning how to use the Pi, learning opencv code and its facial detection functions, and developing code to implement that onto a mechanical system. While the mechanical system itself is impressive, my experience in robotics made that section of the project relatively easy.

As a whole, this project is the greatest thing that I have ever built. The system, the design, the construction, and the ideas were mine. This final product represents everything that I have learned in my college education up to this point (currently a Junior.) It is important to mention that the code was built off the work of other programmers, but it was bent to the purpose of this project, and that method is at the heart of all programming.

IMPLEMENTATION

**Overview.** This robot is designed to detect faces in the immediate vicinity, rotate its entire frame to point at the target, and activate the trigger and chambering mechanisms of the nerf gun. A Raspberry Pi and Arduino are used to perform all previously mentioned functions autonomously. When all nerf darts have been exhausted, a buzzer and 7-segment display light up informing the user that it must be reloaded. Once the user finishes this task, a green button is pressed and the robot resumes its processes.





**Software and Logic Flow.** An Arduino Mega and a Raspberry Pi were both used in this implementation. They communicate with each other via USB Serial, using simple write/read commands.

The Raspberry Pi has one main task: detect faces. It accomplishes this through the use of a python program built using the opencv library and haar face cascades. A dnn (deep neural net) module was also tested here because it is much more reliable, however it took up far too much processing power and could only process 1 frame per second. The haar cascade method was much faster, allowing the robot to function at an appropriate speed. I also decreased the image window to 200x200 pixels, which gave a significant gain in frames processed per second. As it is, face detection is very computationally-intensive and the Pi is using all of its processing power when active.

When a face is detected (or multiple faces,) the Pi detects the horizontal distance from the center of the camera and chooses the nearest face. Since the robot has no vertical movement, it shoots underneath of whichever face it picks (generally in the thoracic region.) Thus the vertical distance is irrelevant. It uses simple subtraction to determine if the face is to the left or the right, and it has a threshold for the “center” position.

The Pi and Arduino communicate by sending character bytes back and forth. When the pi determines a face is in the center, it sends a character to the Arduino which interprets the command to start pulling the trigger, and then it immediately chambers the gun for the next pass. If a face is on either side, it sends different characters, and the Arduino understands it needs to rotate the robot until another command is sent to stop it.

The Arduino is in command of all the mechanical systems. It sends signals to the motor drivers, which drive the stepper motors that operate the gun and rotate the robot. It also has within its function loops the commands to activate the LED arrays, the buzzer, and the buttons. It also sends a signal back to the Pi to verify it is ready to operate at the beginning of operation, and every time after reloading.



*Example of how the pi determines what is a face. For debugging purposes only.*

**Hardware.**

1. Raspberry Pi 3B+ -- Used with OpenCv to process images to detect faces and communicate with the Arduino Mega 2560 Rev3.
   1. Camera Module 2 – Used to capture images to be used with OpenCv
2. Arduino Mega 2560 Rev3 – Used to control the motor controllers and all external devices.
3. Lin Engineering 4118 Series Stepper Motors x3 – Used to control the turning, chambering, and trigger of the Nerf gun.
4. Stepper Motor Drivers x3 – Used to control the stepper motors.
5. 7-Segment Display
   1. 1 Digit 7-Segment Display – Used to display the remaining ammo count.
   2. 74HC595 IC – Used to convert the input from the Arduino Mega into binary values to display on the 7-Segment Display.
6. HC-SR501 PIR Sensor – Used to detect motion in order to wake the Nerf gun and have it begin scanning for targets.
7. MAX7219 LED Dot Matrix Module – Used to display different smiley faces depending on which stage of shooting the Nerf gun is in.
8. LED Tactile Button
   1. Green: Used to tell the Arduino that the Nerf Gun has been reloaded.
   2. Red: Used as a safety that stops all motor functions without removing power from the entire circuit.
9. Toggle Switch – Used to cut all battery power from the circuit.
10. 5v Voltage Regulator – Used to regulate battery power from 7.4v to 5v in order to power the Raspberry Pi which powers the Arduino.
11. 7.4v 4000 mAh Li-PO Battery – Used to power the entire circuit.

APPENDIX A- PYTHON CODE

#I’m using the opencv and imutils library, which allows multi-threading of the analysis processes

from imutils.video.pivideostream import PiVideoStream

from picamera.array import PiRGBArray

from picamera import PiCamera

import imutils

import time

import serial

import cv2

#This sets up Serial communication with the Arduino

ser = serial.Serial('/dev/ttyACM0',9600)

#This sets up the face cascades (one of which is unused) and the image resolution to process

cascadePathFront = "/home/pi/Desktop/NerfGun/haarcascade\_frontalface\_default.xml"

cascadePathProfile = "/home/pi/Desktop/NerfGun/haarcascade\_profileface.xml"

faceCascade = cv2.CascadeClassifier(cascadePathFront);

profileCascade = cv2.CascadeClassifier(cascadePathProfile);

font = cv2.FONT\_HERSHEY\_SIMPLEX

widthImage=200

positionHolder=0;

#from the imutils library. The pi can read in raw image matrices and also process them at the same time.

vs = PiVideoStream().start()

time.sleep(2.0)

centerImage = widthImage/2 #x coordinate of center of image

#Waiting for the PIR sensor to activate before doing anything.

loop\_break='a'

while loop\_break!='s':

loop\_break=ser.read(1)

#main loop of the program. It continuously scans for faces here, and sends instructions as appropriate to the #Arduino.

while(1):

img =vs.read()

#resizes it here in order to decrease processing time. A lot of the other options here are just to get the image right #side up and gray.

img = imutils.resize(img, width=widthImage)

img = imutils.rotate(img, angle=90)

#img = cv2.flip(img, -1) # Flip vertically

gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

#Used within the loops to be used for logic later.

distance=widthImage;

xf=0

yf=0

wf=0

hf=0

#I am using two separate Haar cascades to get both the front and side of the face.

faces = faceCascade.detectMultiScale(

gray,

scaleFactor = 1.2,

minNeighbors = 4,

minSize = (int(20), int(20)),

)

#The other cascade works well enough and removing this increases the speed.

# profiles = profileCascade.detectMultiScale(

# gray,

# scaleFactor = 1.2,

# minNeighbors = 4,

# minSize = (int(20), int(20)),

# )

for(x,y,w,h) in faces:

cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)

distance\_temp=centerImage-(x+w/2)

if abs(distance\_temp)<abs(distance):

distance=distance\_temp

xf=x

yf=y

wf=w

hf=h

# for(x,y,w,h) in profiles:

# cv2.rectangle(img, (x,y), (x+w,y+h), (0,255,0), 2)

# distance\_temp=centerImage-(x+w/2)

# if abs(distance\_temp)<abs(distance):

# distance=distance\_temp

# xf=x

# yf=y

# wf=w

# hf=h

#using only the x-distance from the center will allow me to position the turret in

#optimal position for shooting.

cv2.rectangle(img, (xf,yf), (xf+wf,yf+hf), (0,0,255), 2)

loop\_break='a'

if distance<widthImage:

if abs(distance)<25: #20 seemed to be an accurate value that will also allow the

#program to operate without infinitely correcting itself

cv2.putText(img, "\*\*Bang\*\*", ((centerImage-75),30), font, 1, (0,0,255), 2)

ser.write(b't')

positionHolder=0;

while loop\_break!='h':

loop\_break=ser.read(1)

elif distance>0:

cv2.putText(img, "<--", ((centerImage-75),30), font, 1, (0,255,0), 2)

if positionHolder!=1:

ser.write(b'l')

positionHolder=1;

else:

cv2.putText(img, "-->", (centerImage,30), font, 1, (0,255,0), 2)

if positionHolder!=2:

ser.write(b'r')

positionHolder=2;

cv2.imshow('camera',img)

xf = 0

yf = 0

wf = 0

hf = 0

k = cv2.waitKey(10) & 0xff # Press 'ESC' for exiting video

if k == 27:

break

# Do a bit of cleanup

print("\n [INFO] Exiting Program and cleanup stuff")

cv2.destroyAllWindows()

vs.stop()

APPENDIX B- ARDUINO CODE

//Pre-set-up

byte ammoCount=6;

char temp;

char serial\_input;

bool b\_trigger=false;

bool b\_cock=false;

bool b\_left=false;

bool b\_right=false;

//The stepper motors gave me the most trouble of //the entire project.

#include <Stepper.h>

Stepper myStepper(200, 5, 6, 7, 8);

Stepper travelMotor(200, 9, 10, 11, 12);

//Caitlyn's LED array

#include "LedControl.h"

int DIN = 36;

int CS = 38;

int CLK = 40;

LedControl lc=LedControl(DIN,CLK,CS,0);

byte dir=2;

byte stp=3;

byte en=4;

byte PIR=34;

//Kyle's 7-segment additions

byte seven\_seg\_digits[8] = { B11111100, // = 0

B01100000, // = 1

B11011010, // = 2

B11110010, // = 3

B01100110, // = 4

B10110110, // = 5

B10111110, // = 6

B00000000 // = OFF

};

int dataPin = 22; //2A to 13C

int latchPin = 24; //3A to 10C

int clockPin = 26; //4A to 9C

int buzzerPin = 28; //5A to +Buzzer

int reloadPin = 30; //9A to B1 Ground

//define the LED array faces

byte smile[8]= {0x3C,0x42,0xA5,0x81,0xA5,0x99,0x42,0x3C};

byte neutral[8]= {0x3C,0x42,0xA5,0x81,0xBD,0x81,0x42,0x3C};

byte frown[8]= {0x3C,0x42,0xA5,0x81,0x99,0xA5,0x42,0x3C};

void setup() {

Serial.begin(9600);

pinMode(dir, OUTPUT); //dir

pinMode(stp, OUTPUT); //step

pinMode(en,OUTPUT); //enable

digitalWrite(dir, LOW);

digitalWrite(stp, LOW);

digitalWrite(en,HIGH);

//7-segment and buzzer

pinMode(latchPin, OUTPUT);

pinMode(clockPin, OUTPUT);

pinMode(dataPin, OUTPUT);

pinMode(buzzerPin, OUTPUT);

pinMode(reloadPin, INPUT);

//stop all button 32

//motion sensor 34

sevenSegWrite(ammoCount);

//Robot stays here until PIR sensor goes off

//David's PIR sensor

pinMode(PIR, INPUT);

int PIR\_read=LOW;

while(PIR\_read==LOW){

PIR\_read= digitalRead(PIR);

Serial.print('s');

}

//LED array

lc.shutdown(0,false);

//The MAX72XX is in power-saving mode on startup

lc.setIntensity(0,15);

// Set the brightness to maximum value

lc.clearDisplay(0); // and clear the display

//pre-cock the gun before face-detection starts

cock();

}

**//\*\*MAIN LOOP\*\*\*\*\*\*\*\*//**

void loop() {

//constantly checking the pi for updates.

//the pi is the brain. The arduino just executes commands, like a perfect soldier.

serial\_read\_and\_bool\_determination();

if(ammoCount==0){

while(1){

//use an interrupt or poll and #break. Probably will use pin 1.

//ammoCount=6;

}

}

if(b\_trigger)

trigger();

if(b\_cock)

cock();

if(b\_right)

motorRight();

if(b\_left)

motorLeft();

}

**//\*\*END MAIN LOOP\*\*\*\*\*\*\*\*//**

//Here are all the serial read commands and most //of the logic that determines the robots entire //entire functionality.

void serial\_read\_and\_bool\_determination(){

if(Serial.available()>0){

temp=Serial.read();

if((temp!='\n')&&(temp!='\r')){

serial\_input=temp;

digitalWrite(9,LOW);

digitalWrite(10,LOW);

digitalWrite(11,LOW);

digitalWrite(12,LOW);

switch (serial\_input){

case 'c':

b\_cock=true;

break;

case 't':

b\_trigger=true;

b\_right=false;

b\_left=false;

break;

case 'r':

b\_right=true;

b\_left=false;

break;

case 'l':

b\_left=true;

b\_right=false;

break;

break;

}

}

}

}

//the rest of this code is pretty self-explanatory //through the simple fact that I used very //descriptive and relevant variable/function names.

void cock(){

int delayCock=800;

int steps=620;

digitalWrite(dir,HIGH);

digitalWrite(en,LOW);

for(int i=0;i<steps;i++){

digitalWrite(stp, HIGH);

delayMicroseconds(delayCock);

digitalWrite(stp, LOW);

delayMicroseconds(delayCock);

}

digitalWrite(dir,LOW);

for(int i=0;i<steps;i++){

digitalWrite(stp, HIGH);

delayMicroseconds(delayCock);

digitalWrite(stp, LOW);

delayMicroseconds(delayCock);

}

digitalWrite(en,HIGH);

b\_cock=false;

Serial.print('h');

printByte(smile);

}

void trigger(){

printByte(frown);

myStepper.setSpeed(80);

myStepper.step(270);//(stepsPerRevolution / 100);

myStepper.step(-270);//(stepsPerRevolution / 100);

digitalWrite(5,LOW);

digitalWrite(6,LOW);

digitalWrite(7,LOW);

digitalWrite(8,LOW);

b\_trigger=false;

b\_cock=true;

ammoCount=ammoCount-1;

sevenSegWrite(ammoCount);

if(ammoCount==0){

while(digitalRead(reloadPin)!=HIGH){

sevenSegWrite(0);

delay(100);

digitalWrite(buzzerPin, HIGH);

delay(100);

sevenSegWrite(7);

delay(100);

digitalWrite(buzzerPin, LOW);

delay(100);

}

ammoCount = 6;

sevenSegWrite(ammoCount);

digitalWrite(buzzerPin,LOW);

}

}

void motorRight(){

travelMotor.setSpeed(10);

travelMotor.step(1);

}

void motorLeft(){

travelMotor.setSpeed(10);

travelMotor.step(-1);

}

void sevenSegWrite(unsigned char digit)

{

digitalWrite(latchPin, LOW);

shiftOut(dataPin, clockPin, LSBFIRST, seven\_seg\_digits[digit]);

digitalWrite(latchPin, HIGH);

}

void printByte(byte character [])

{

int i = 0;

for(i=0;i<8;i++)

{

lc.setRow(0,i,character[i]);

}

}

1. This is much more to me than simply a Final Project. I have been designing and learning about these concepts all semester, using much of my free time to bring this to life, even before I was aware of the project requirements. This project represents my passion for robotics, and it was only good fortune that I was able to use it as a grade. [↑](#footnote-ref-1)
2. I must give an incredible amount of credit to Adrian Rosebrock at Pyimagesearch.com, for freely supplying tutorials and explanations of these features, and for using his imutils library, which gives the pi a significant increase in speed by utilizing its 4 cores for image processing tasks. [↑](#footnote-ref-2)