Written Proposal

Shaun Holtzman

Draw-O-Matic 9000

EEL 4665/5666 Intelligent Machines Design Laboratory

Instructors: Dr. A. Antonio Arroyo, Dr. Eric M. Schwartz

TAs: Andy Gray, Jake Easterling

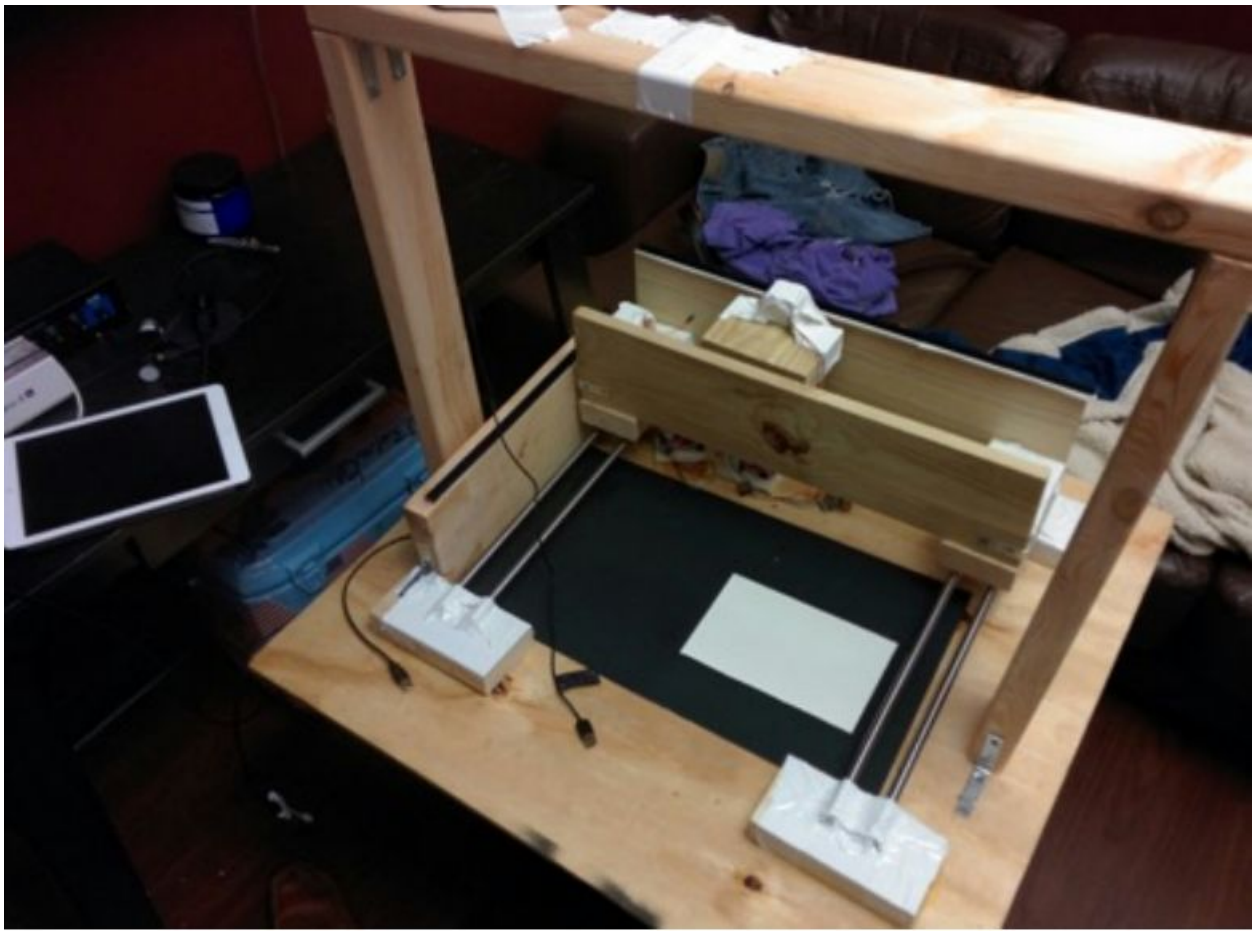

Table of Contents

Abstract……..2

Executive Summary……...2

Introduction…….…...3

Integrated System…….….3

Mobile Platform……....4

Actuation……...4

Sensors……..5

Special Sensor/System……..5

Behaviors……..10

Experimental Layout and Results…….10

Conclusion……11

Appendices……...11

Abstract

My robot, Draw-O-Matic 9000, is designed to have the ability to recognize certain shapes and draw them. The main idea is to allow the user to approach the robot, hold up a shape for the camera to detect, and for the robot to draw it on a piece of paper. One of the most important aspects of my robot is the ability for it scale the picture to the size of the paper being used. The drawing mechanism is controlled by a gantry type system with one rail moving the pen vertically across the paper while the other moves it horizontally across the paper and a solenoid to control the pen. OpenCV is being used to detect the shapes as well as detecting where the paper is located and its size.

Executive Summary

Draw-O-Matic 9000 can detect shapes using a laptop camera and OpenCV. It draws them on a piece of paper, scaling them to the size of the paper. It looks at the properties the shapes in order to determine what shape it is, including if it is convex, how many sides, and the angles between the sides. I have a camera mounted on top of my robot that is used to detect the size of the paper as well as its location and size. This information is used by my computer to calculate how far to move the stepper motors when finding the paper as well as when drawing the image. Once the calculations are done, the proper data is sent to the Arduino to control the motors and the pen.

My robot uses my MacBook laptop for the image processing and an Arduino Due for controlling the motors. In total I am using two stepper motors, one solenoid, two motor drivers, a transistor, a mosfet, two limit switches, an Arduino Due, a MacBook Pro, a Microsoft LifeCam, and a handful of resistors. This is in addition the the wood and metal rods used to construct the physical appearance of the robot. The two stepper motors have wheels with teeth and sit on a platform attached to linear bearings on a set of rods and use a track to rotate the wheels on which moves the platforms across the rods.

The robot starts by letting the user hold up a shape from a set of shapes I have determined up to the camera. Once the user has the shape in the camera's view, they will be able to see their shape on the screen, outlined by a green line indicating it has been detected. Once their shape is detected, they hit the s button to save the image. Afterwards, OpenCV switches the the camera attached to the top of the robot and detects the size and location of the paper being drawn on. Once completed, the robot will draw the shape and then calibrate itself by returning the starting position. The robot then waits for another shape to draw.

Introduction

Edge detection is one of the most widely used applications in image processing and is often the first thing that is done to an image before more complicated processing is done on the image. I have experience with the edge detection algorithm particularly with the Reconfigurable Computing class I took where I did the algorithm through an FPGA. Since an edge detected picture has the pixel data of either a black pixel or a white pixel, I thought it would be an interesting way to draw the outline of features onto a piece of paper. I had originally decided to draw the image that the camera took onto the piece of paper. However, complications led to me drawing shapes instead. The shapes are detected by initially using a canny edge detector and then sent through other image processing that will be discussed in this paper to determine the shape being detected. This report will go through a detailed summary of the robot's design, and how it all works.

Integrated System

My robot uses my MacBook Pro's cam to detect the shape the user wants drawn. Once the picture is taken, the Microsoft LifeCam detects the location and size of the paper being drawn on. After the calculations using this information is finished, the finalized data is sent to the Arduino Due using serial communication over USB. The Arduino Due controls the rails system as well as the solenoid that is attached to the pen. The rails have linear rail bearings on them, with the stepper motor sitting on top. The step motors have a gear with teeth on them that and a track with teeth for the gear to rotate across. The motor rotates the gear that will catch the teeth of the rail, pushing the platform across the rail. At the end of the rail there are limit switches that will help the system identify when it has reach one end of the rail and will help with calibrating after every drawing. The pen is controlled by a push solenoid that moves pen up and down. Each stepper motor is driven using a motor driver and the solenoid is controlled by a mosfet and transistor circuit. Figure 1 contains a picture of the circuitry that controls my robot.

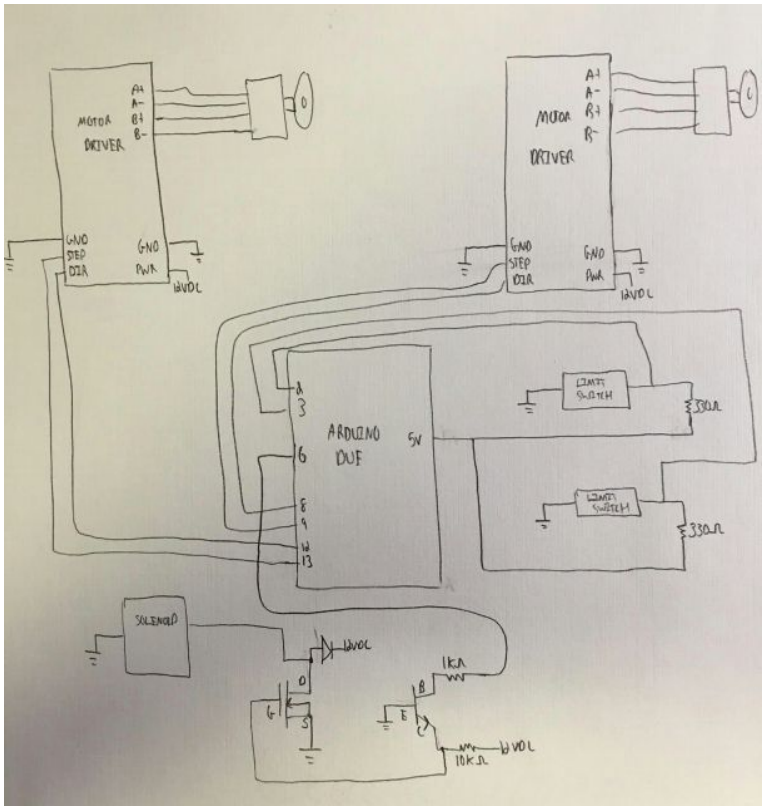

Figure 1. Circuitry for Draw-O-Matic 9000

Mobile Platform

My robot is stationary so it does not have a mobile platform. The platform I am using is just a sheet of wood in order to hold all the components of the gantry. The gantry consists of an X and Y axis with stepper motors moving each platform for each axis. A camera is mounted above the gantry in order to do image processing on the paper being drawn on.

Actuation

The motors being used to control the rail system are CanaKit Stepper Motors and driven by an EasyDriver stepper motor driver with a 12V input. Using this stepper motor allows me to move the rail in small enough increments (1/8 of a rotation being the smallest) so I can be relatively accurate with placing the pen at specific locations A solenoid is used to move the pen up and down. Moving the pen up and down is done using a solenoid. The pen is attached to the solenoid using what I have pictured below in figure 2. Attaching the pen in this manner stabilizes the pen for better drawings and allows me to draw using lines rather than pixels by holding the pen down when drawing. The circuit for controlling the solenoid can be seen above in figure 1, which contains a mosfet and a transistor. Using the mosfet and transistor allows me to control the 12V signal that goes to the solenoid with the the digital output pins of the Arduino.

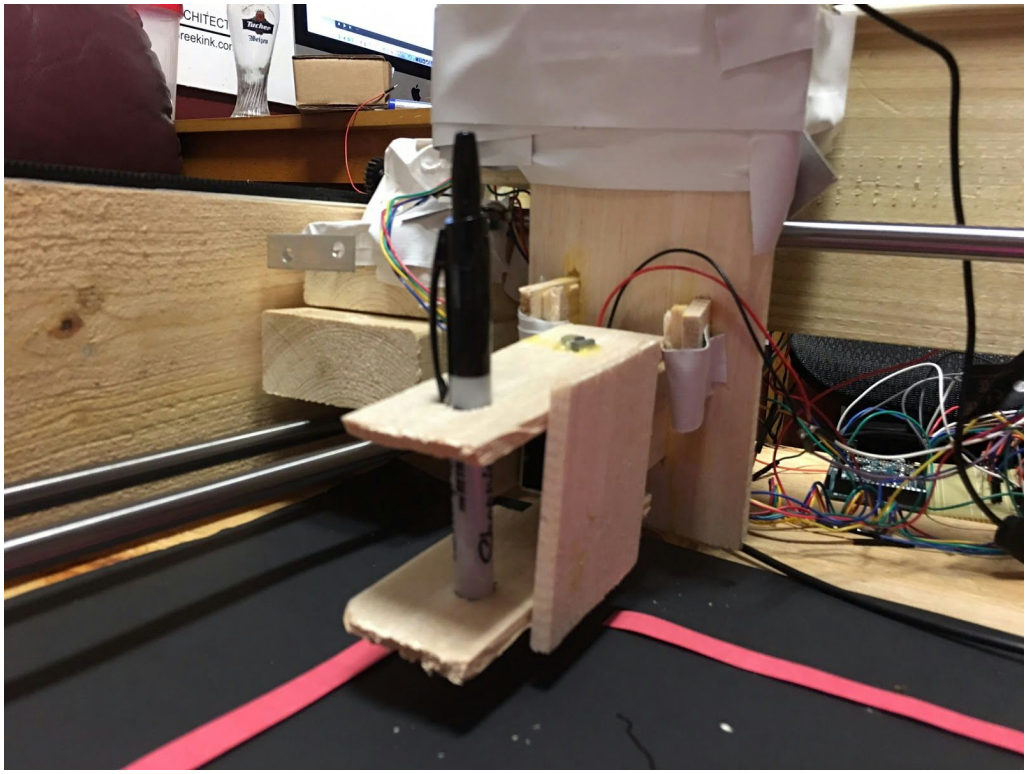

Figure 2. How the pen was attached to the solenoid

Sensors

For the shape detection, my MacBook Pro's cam is being used. The paper size and location detection is done using a Microsoft LifeCam. The MacBook's cam is more than clear enough for shape detection, and the LifeCam takes pictures at a max resolution of 720p and is good enough for detecting the size and location of the paper being drawn on. In order to identify when the rail system has met one end of the rail, limit switches will be used. This is crucial for calibrating each time a picture is drawn. The rails move toward the limit switches and one they are hit they will stop moving. This gives the robot a point of origin before moving to the paper to draw the shape.

Special Sensor/System

My special system design is two-fold. First, it includes the ability to track the location of the paper on the platform; Secondly, it is able to scale the image being drawn based on the size of the paper. In order to achieve these goals, OpenCV was used along with some calculations.

The first step was mounting my camera. I had to make sure it was in a fixed position above the gantry in order to know that the calculations I am making will be consistent throughout. The first step I took was taking a picture of the platform with the camera and recording the pixel location of both the X and Y axis. Afterward, I moved each axis an arbitrary number of steps using the stepper motors, and recorded the new pixel location of each axis. Using the distance each platform moved in pixels along with the amount of steps taken by the motor allowed me to calculate a ratio of pixels to steps. This allows me to easily move the pen to wherever the paper is since all I need to know is the X and Y pixel coordinates of the paper along with the calculated ratio.

Once I had this ratio, I had to figure out how I was going to detect the piece of paper. I first put a black background behind where the paper goes in order to make sure the piece of paper pops out in the image. I applied a blur to reduce noise, and then ran it through a Canny edge detector to detect the edges of the paper. Afterwards, I found all the contours within the image. The contours are used to calculate a polygonal approximation, which is in turn used to determine if the shape being detected is convex as well as how many sides the shape has and the angles between its sides. Using this method allows me to cut out extraneous contours and shapes that the camera might pick up and helps keep the paper detection consistent. Afterwards, I used OpenCV's rectangle class to draw a box around the paper. Using the rectangle class allowed me to easily find the X and Y coordinates as well as the length and the width of the paper since both values are properties of the rectangle class. As shown below in figures 3, 4, and 5, the program is able to find the paper within the designated area, regardless of orientation.

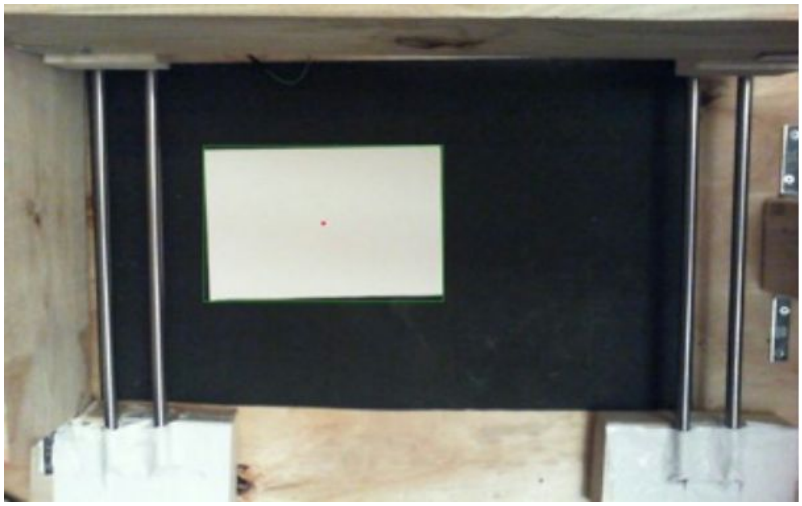

Figure 3. Paper location detection part one

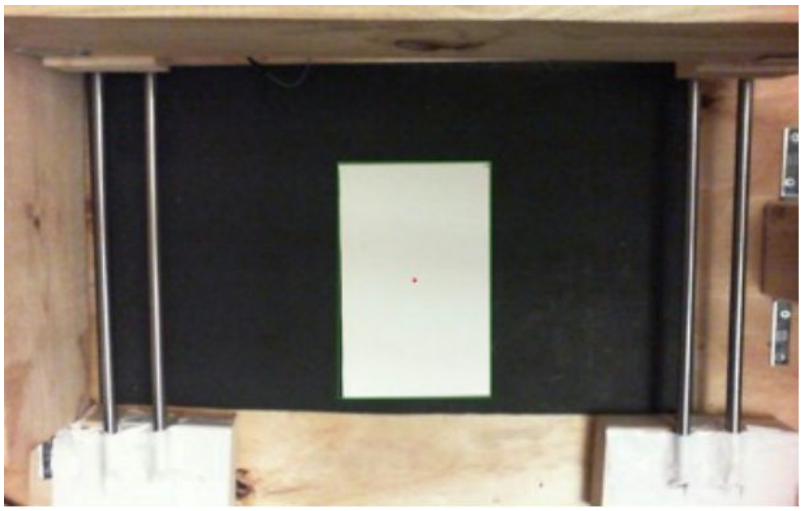

Figure 4. Paper location detection part two

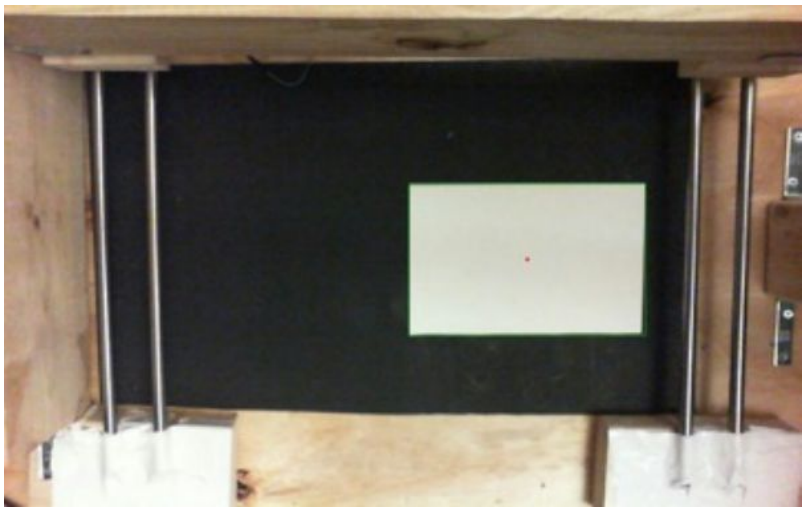

Figure 5. Paper location detection part three

In the examples shown in figures 3, 4, and 5, the center of the paper is calculated along with the length and the width. This center point is outputted by the program and is used alongside the calculated ratio to tell the motors how many steps to take. The ability to move the pen to where the paper is located is an important part of my project and was one of the main things I wanted to achieve. Doing this allows me to just put the paper on the platform without having it to be in a specific position every time and adds to the autonomy of my system. Also, since I am using the contours of the paper for detection, I am not limited to any specific color. However, using something dark that will blend into the black background could cause some issues with the edge detection.

While figuring out how to get the pen to move to where the paper is, I ran into quite a few issues. One of the first issues I had was finding the contours properly. With the above images you can see there is a lot more in the picture than just the drawing region of where the paper is. This cause a lot of random contours to popup. To work around this, I made sure to use the polygonal approximation to cutout non-convex shapes and shapes that did not meet the criteria of a rectangular or square piece of paper. This includes having four sides and approximately 90 degrees between each side.

I also had some issues with calculating the ratio properly. The main mistake I made was something I just overlooked. When recording the pixel distance moved by each axis, I was first only recording the final position it was in, not taking into account the starting position. Doing so threw the ratio off. It was not until later that I realized I need to record the difference in position, not just the final position, since that does not give me the correct distance traveled. After getting through these two problems, I was able to successfully and continuously move the pen to the paper.

In order to detect the shape before drawing it on the piece of paper, I had to run the image of the shape through a series of steps. First the image is converted to grayscale and then a Gaussian blur is applied to reduce noise. Afterwards, the image is sent through a Canny edge detector and then the contours of the output of the Canny edge detector is calculated. The contours are used to for the polygonal approximation. Using the polygonal approximation allows me to determine if the shape is convex, how many sides it has, as well as the angles between the sides. These steps allow me to differentiate between two shapes that have the same number of sides as can be seen below in figures 6 and 7 with the detection of a square and a diamond. Even though a diamond and a square have the same number of sides, there are obviously different shapes.

Figure 6. Detecting a square

Figure 7. Detecting a diamond

The image scaling is done similar to finding the location of the center of the paper. I first put an arbitrarily sized paper down and recorded how many steps it took to go across the length and the width of the paper. This amount of steps along with the pixel dimensions of the paper are

used to make a ratio of steps per pixels. This allows me to find the dimensions of the paper in pixels, multiply it by this ratio, and have the motors use this value when drawing the shape. I made sure to leave out a portion of the edges of the paper when finding this ratio to avoid drawing off the sides of the paper if something happens to be off.

Behaviors

My robot has two distinct behaviors. The first is detecting the shape to draw on the piece of paper and the second is the actual drawing of the shape. It starts out in the first behavior and waits until a shape is detected. Once the shape is detected, it goes into the second behavior of actually drawing the shape. After that it is completed, the robot recalibrates and goes back into the first behavior and waits for the next shape.

One lesson I learned in particular was with the shape detection. I had make sure that I was able to not begin drawing if the wrong shape was detected or if not shape was detected at all. This lesson was learned during demo day when my robot failed to detect a shape at one point, which caused the Arduino to receive incorrect information and forced me to restart the program over manually. Since then, I have made sure the drawing will not begin until a shape is detected correctly. In addition, stabilizing the camera is another aspect I overlooked for demo day. Everything worked fine at home, but since the camera was not particularly stable and the robot was transported in a car, it was off a bit from where I originally had it. This threw off finding the location of the paper and I have made sure the camera is more stable since.

Experimental Layout and Results

Getting the pen to consistently move to the center of the paper took a bit of experimenting. When taking a picture of the paper location, the paper is not always right underneath the camera so my ratio made the pen either go too far past the center or come up short depending on where the paper is located. To rectify this problem, I made a ratio for certain quadrants. Depending on where the paper is, I would use the ratio for that specific quadrant which helped cut down on the inconsistency.

I also had issues with serial communication. When sending the data from my computer to the Arduino, my Arduino did not seem to read the data quick enough before it was removed from the buffer. I had to mess around with delays in order to make sure the Arduino received the right information at the right time.

When I first began drawing images I had issues with moving both stepper motors at the same time. This was also an issue with delays and I just had to find the right delay time between

sending a high and low signal to the motors. Being able to move both stepper motors at the same time let me draw shapes with angled sides, such as a triangle and a diamond, using lines rather than pixels like I originally was doing.

Conclusion

In conclusion, even though I was not able to achieve my original goal of directly drawing an image the camera took a picture of, I am happy with what I have achieved. I am able to consistently detect shapes, find the piece of paper, and scale the shape to the size of the paper. I learned a lot of OpenCV that I did not already know and had my first experience with controlling motors. The whole idea of making up a project and spending a semester on one project was quite an experience and I am glad I decided to take this class. The only thing I regret is the size I made my robot. My robot's size makes it difficult to move it from one location to another and always took one other person to help me bring it in for the demonstrations.

Appendices

Final Arduino Code:

```
int pinXdir = 8;
int pinXstep = 9;

int pinYdir = 12;
int pinYstep = 13;

int buttonPinX = 3;
int buttonPinY = 2;

int solenoidPin = 6;

int X = 1;
int Y = 1;

int d = 0;

char shape;
char garbage;
```

```
char test1 = 'x';
char test2 = 'x';
char test3 = 'x';

char x[4];
char y[4];
char paper[4];

void setup() {
  shape = 'x';
  pinMode(pinXdir, OUTPUT);
  pinMode(pinXstep, OUTPUT);
  digitalWrite(pinXdir, LOW);
  digitalWrite(pinXstep, LOW);
  pinMode(buttonPinX, INPUT);

   pinMode(pinYdir, OUTPUT);
  pinMode(pinYstep, OUTPUT);
  digitalWrite(pinYdir, HIGH);
  digitalWrite(pinYstep, LOW);
  pinMode(buttonPinY, INPUT);

  pinMode(solenoidPin, OUTPUT);
   digitalWrite(solenoidPin, HIGH);
   Serial.begin(9600);
}

void loop() {

//Calibrate X axis
  setup();
  while(digitalRead(buttonPinX) == HIGH) {
   digitalWrite(pinXstep, HIGH);
  delayMicroseconds(250);
  digitalWrite(pinXstep, LOW);
  delayMicroseconds(250);
  }
```

```
//Calibrate Y axis

   while(digitalRead(buttonPinY) == HIGH){
     digitalWrite(pinYstep, HIGH);
     delayMicroseconds(250);
     digitalWrite(pinYstep, LOW);
     delayMicroseconds(250);
     }


while(Serial.available() > 0) {
    garbage = Serial.read();
  }

    while(!Serial.available());

      shape = Serial.read();


delay(1000);

 while(Serial.available() > 0) {
    garbage = Serial.read();
  }

    while(!Serial.available());
      x[0] = Serial.read();


      delay(1000);
      while(Serial.available() > 0) {
    garbage = Serial.read();
  }

  while(!Serial.available());
      x[1] = Serial.read();

       delay(1000);
      while(Serial.available() > 0) {
```

```
  garbage = Serial.read();
}

while(!Serial.available());
    x[2] = Serial.read();
    x[3] = '\0';

    delay(1000);

    while(Serial.available() > 0) {
  garbage = Serial.read();
}

  while(!Serial.available());
    y[0] = Serial.read();


    delay(1000);
    while(Serial.available() > 0) {
  garbage = Serial.read();
}

while(!Serial.available());
    y[1] = Serial.read();

     delay(1000);
    while(Serial.available() > 0) {
  garbage = Serial.read();
}

while(!Serial.available());
    y[2] = Serial.read();
    y[3] = '\0';

    delay(1000);

    while(Serial.available() > 0) {
  garbage = Serial.read();
}
```

```
   while(!Serial.available());
    paper[0] = Serial.read();


     delay(1000);

      while(Serial.available() > 0) {
   garbage = Serial.read();
  }

   while(!Serial.available());
    paper[1] = Serial.read();


     delay(1000);

      while(Serial.available() > 0) {
   garbage = Serial.read();
  }

   while(!Serial.available());
    paper[2] = Serial.read();
    paper[3] = '\0';


     delay(1000);


int xtemp = atoi(x);
int ytemp = atoi(y);
int paperfinal = atoi(paper);

double xtemp2 = (xtemp-65-341)*14.7959184;
double ytemp2 = (ytemp+14-261)*16.8181818;

int xfinal = (int)xtemp2;
int yfinal = (int)ytemp2;
```

```
  digitalWrite(pinYdir, LOW);
    while(d < yfinal) {
    digitalWrite(pinYstep, HIGH);
     delayMicroseconds(250);
     digitalWrite(pinYstep, LOW);
     delayMicroseconds(250);
    d++;
  }
  d = 0;

  digitalWrite(pinXdir, HIGH);
    while(d < xfinal) {
      digitalWrite(pinXstep, HIGH);
     delayMicroseconds(250);
     digitalWrite(pinXstep, LOW);
     delayMicroseconds(250);
    d++;
    }
    d = 0;

delay(750);

double stepcalc = paperfinal*10;
int stepcalcfinal = (int)stepcalc;

if(shape == 's') {

digitalWrite(pinXdir, HIGH);
while(d<(stepcalcfinal/2)) {
     digitalWrite(pinXstep, HIGH);
     delayMicroseconds(400);
     digitalWrite(pinXstep, LOW);
     delayMicroseconds(400);
     d++;
  }
  d = 0;
  delay(750);
```

```
  digitalWrite(solenoidPin,LOW);
  delay(750);
digitalWrite(pinYdir, LOW);
  while(d<(stepcalcfinal/2)) {
    digitalWrite(pinYstep, HIGH);
     delayMicroseconds(400);
     digitalWrite(pinYstep, LOW);
     delayMicroseconds(400);
     d++;
  }
  d = 0;
delay(750);
  digitalWrite(pinXdir, LOW);
  while(d<stepcalcfinal) {
     digitalWrite(pinXstep, HIGH);
     delayMicroseconds(400);
     digitalWrite(pinXstep, LOW);
     delayMicroseconds(400);
     d++;
  }
  d = 0;
delay(750);
  digitalWrite(pinYdir, HIGH);
  while(d<stepcalcfinal) {
    digitalWrite(pinYstep, HIGH);
     delayMicroseconds(400);
     digitalWrite(pinYstep, LOW);
     delayMicroseconds(400);
     d++;
  }
  d = 0;
delay(750);
  digitalWrite(pinXdir, HIGH);
  while(d<stepcalcfinal) {
     digitalWrite(pinXstep, HIGH);
     delayMicroseconds(400);
     digitalWrite(pinXstep, LOW);
     delayMicroseconds(400);
     d++;
```

```
  }
  d = 0;
delay(750);
  digitalWrite(pinYdir, LOW);
  while(d<(stepcalcfinal/2)) {
    digitalWrite(pinYstep, HIGH);
     delayMicroseconds(400);
     digitalWrite(pinYstep, LOW);
     delayMicroseconds(400);
     d++;
  }
  d = 0;
  delay(750);
  digitalWrite(solenoidPin, HIGH);
}

if(shape == 't') {

digitalWrite(pinYdir, HIGH);

 while(d<(stepcalcfinal/2)) {
    digitalWrite(pinYstep, HIGH);
     delayMicroseconds(400);
     digitalWrite(pinYstep, LOW);
     delayMicroseconds(400);
     d++;
  }
  d = 0;
  delay(750);
  digitalWrite(solenoidPin, LOW);
  delay(750);
  digitalWrite(pinXdir, HIGH);
  digitalWrite(pinYdir, LOW);
  while(d<stepcalcfinal/1.5) {
    digitalWrite(pinYstep, HIGH);
    digitalWrite(pinXstep,HIGH);
    delayMicroseconds(400);
    digitalWrite(pinYstep, LOW);
    digitalWrite(pinXstep,LOW);
```

```
  delayMicroseconds(400);
  d++;
 }
 d=0;
 delay(750);
 digitalWrite(pinXdir,LOW);
 while(d<stepcalcfinal*1.35){
  digitalWrite(pinXstep, HIGH);
  delayMicroseconds(400);
  digitalWrite(pinXstep, LOW);
  delayMicroseconds(400);
  d++;
 }
 d=0;
 delay(750);
 digitalWrite(pinXdir,HIGH);
 digitalWrite(pinYdir,HIGH);
 while(d<stepcalcfinal/1.5){
   digitalWrite(pinYstep, HIGH);
  digitalWrite(pinXstep,HIGH);
  delayMicroseconds(400);
  digitalWrite(pinYstep, LOW);
  digitalWrite(pinXstep,LOW);
  delayMicroseconds(400);
  d++;

 }
 delay(750);
 digitalWrite(solenoidPin, HIGH);


}

if(shape == 'd') {

 digitalWrite(pinYdir, HIGH);

 while(d<(stepcalcfinal/2)) {
  digitalWrite(pinYstep, HIGH);
```

```
   delayMicroseconds(400);
   digitalWrite(pinYstep, LOW);
   delayMicroseconds(400);
   d++;
}
d = 0;
delay(750);
digitalWrite(solenoidPin, LOW);
delay(750);
digitalWrite(pinXdir, HIGH);
digitalWrite(pinYdir, LOW);
while(d<stepcalcfinal/2) {
  digitalWrite(pinYstep, HIGH);
  digitalWrite(pinXstep,HIGH);
  delayMicroseconds(400);
  digitalWrite(pinYstep, LOW);
  digitalWrite(pinXstep,LOW);
  delayMicroseconds(400);
  d++;
}
d=0;
delay(750);
digitalWrite(pinXdir, LOW);
 while(d<stepcalcfinal/2) {
  digitalWrite(pinYstep, HIGH);
  digitalWrite(pinXstep,HIGH);
  delayMicroseconds(400);
  digitalWrite(pinYstep, LOW);
  digitalWrite(pinXstep,LOW);
  delayMicroseconds(400);
  d++;
}
d=0;
delay(750);
digitalWrite(pinYdir, HIGH);
 while(d<stepcalcfinal/2) {
  digitalWrite(pinYstep, HIGH);
  digitalWrite(pinXstep,HIGH);
  delayMicroseconds(400);
```

```
  digitalWrite(pinYstep, LOW);
  digitalWrite(pinXstep,LOW);
  delayMicroseconds(400);
  d++;
 }
 d=0;
 delay(750);
 digitalWrite(pinXdir,HIGH);
  while(d<stepcalcfinal/2) {
  digitalWrite(pinYstep, HIGH);
  digitalWrite(pinXstep,HIGH);
  delayMicroseconds(400);
  digitalWrite(pinYstep, LOW);
  digitalWrite(pinXstep,LOW);
  delayMicroseconds(400);
  d++;
 }
 d=0;
 delay(750);
 digitalWrite(solenoidPin, HIGH);


}

 delay(3000);
 //while(1);

}
```

Final C++ Code:

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/core/core.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <cv.h>
#include <highgui.h>
#include <iostream>
```

```
#include <fstream>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <fcntl.h>   /* File control definitions */
#include <errno.h>   /* Error number definitions */
#include <termios.h> /* POSIX terminal control definitions */
#include <iostream>
#include <string.h>
#include <unistd.h>
#include <stdint.h>
#include <fcntl.h>
#include <termios.h>
#include <errno.h>
#include <math.h>
#include <sys/ioctl.h>

using namespace cv;
using namespace std;

/*

 This program detects specfic
 shapes from the camera feed.

 Square
 ------

 Find the contours
 Approximate a polygonal curve for the contours
 Polygonal curve makes a square if:
 1) It is convex
 2) It has 4 vertices
 3) All angles are ~90 degrees

 */

void onMouse( int event, int x, int y, int, void* )
{
```

```
    if( event != CV_EVENT_LBUTTONDOWN )
        return;

    Point pt = Point(x,y);
    std::cout<<"x="<<pt.x<<"\t y="<<pt.y<<"\n";

}


static double angle( Point pt1, Point pt2, Point pt0 )
{
    double dx1 = pt1.x - pt0.x;
    double dy1 = pt1.y - pt0.y;
    double dx2 = pt2.x - pt0.x;
    double dy2 = pt2.y - pt0.y;
    return (dx1*dx2 + dy1*dy2)/sqrt((dx1*dx1 + dy1*dy1)*(dx2*dx2 + dy2*dy2) + 1e-10);
}




int main(int argc, char* argv[])
{
    while(1) {
    char buffer[2];

    int fd; //The device label for the Arduino
    struct termios toptions; //Store the serial port settings
    //Open the serial port
    fd = open("/dev/cu.usbmodem1461", O_RDWR | O_NOCTTY);
    //printf("fd opened as %i\n", fd);
    usleep(3500000);
    // Wait for the Arduino to reboot
    tcgetattr(fd, &toptions);
    //Get current serial port settings
    //Set 9600 baud both ways
    cfsetispeed(&toptions, B9600);
    cfsetospeed(&toptions, B9600);
```

```
// Serial settings: 8 bits, no parity, no stop bits
toptions.c_cflag &= ~PARENB;
toptions.c_cflag &= ~CSTOPB;
toptions.c_cflag &= ~CSIZE;
toptions.c_cflag |= CS8;
// Canonical mode
toptions.c_cflag &= ~CRTSCTS;
toptions.c_cflag |= CREAD | CLOCAL;
toptions.c_iflag &= ~(IXON | IXOFF | IXANY);
toptions.c_iflag &= ~(ICANON | ECHO | ECHOE | ISIG);
toptions.c_oflag &= ~OPOST;
//Commit the serial port settings
tcsetattr(fd, TCSANOW, &toptions);




char shape;
int square = 0;
int exit = 0;

Mat picture;
Mat calcShape;
Mat gray;
Mat blurredImage;
Mat edges;

double width;
double height;

int largest_area = 0;
int largest_contour_index = 0;

Rect bounding_rect;
   Rect bounding_rect2;

VideoCapture cap(1);

width = cap.get(CV_CAP_PROP_FRAME_WIDTH);
```

```
    height = cap.get(CV_CAP_PROP_FRAME_HEIGHT);

    cvStartWindowThread();



    while(exit == 0){

        cap.read(picture);



        calcShape = picture;



        cvtColor(calcShape, gray, CV_BGR2GRAY);


        blur(calcShape, blurredImage, Size(3,3));
        Canny(gray, edges, 50, 150, 3);

        vector<vector<Point> > contours;
        vector<Vec4i> hierarchy;


        findContours(edges, contours, hierarchy, CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE, Point(0,0));
        Mat finalImage = Mat::zeros(edges.size(), CV_8UC3);

        vector<Point> approx;
        vector<vector<Point> > squares;
        vector<vector<Point> > triangles;
        vector<vector<Point> > diamonds;
        vector<vector<Point> > octagons;
        vector<vector<Point> > pentagons;


        for(int i=0; i<contours.size(); i++) {
```

```
        approxPolyDP(Mat(contours[i]), approx, arcLength(Mat(contours[i]), true)*0.02, true);

        if(fabs(contourArea(contours[i])) < 1000 || !isContourConvex(approx))
           continue;

        if(approx.size() == 3) {
           triangles.push_back(approx);
        }
        else if(approx.size() == 5) {
           pentagons.push_back(approx);
        }
        else if(approx.size() == 8) {
           octagons.push_back(approx);
        }
        else if(approx.size() == 4 && fabs(contourArea(Mat(approx))) > 1000 &&
isContourConvex(Mat(approx))) {

           double maxCosine = 0;

           for(int j=2; j<5; j++){

              double cosine = fabs(angle(approx[j%4], approx[j-2], approx[j-1]));
              maxCosine = MAX(maxCosine, cosine);

           }

           if(maxCosine<0.3) {
              squares.push_back(approx);
           }
           else{
              diamonds.push_back(approx);
           }

        }
```

```
}

for( int i = 0; i < squares.size(); i++ )
{
   const Point* p = &squares[i][0];
   int n = (int)squares[i].size();
   polylines(calcShape, &p, &n, 1, true, Scalar(0,255,0), 3, LINE_AA);
}

for( int i = 0; i < triangles.size(); i++ )
{
   const Point* p = &triangles[i][0];
   int n = (int)triangles[i].size();
   polylines(calcShape, &p, &n, 1, true, Scalar(0,255,0), 3, LINE_AA);
}
for( int i = 0; i < diamonds.size(); i++ )
{
   const Point* p = &diamonds[i][0];
   int n = (int)diamonds[i].size();
   polylines(calcShape, &p, &n, 1, true, Scalar(0,255,0), 3, LINE_AA);
}
for( int i = 0; i < pentagons.size(); i++ )
{
   const Point* p = &pentagons[i][0];
   int n = (int)pentagons[i].size();
   polylines(calcShape, &p, &n, 1, true, Scalar(0,255,0), 3, LINE_AA);
}
for( int i = 0; i < octagons.size(); i++ )
{
   const Point* p = &octagons[i][0];
   int n = (int)octagons[i].size();
   polylines(calcShape, &p, &n, 1, true, Scalar(0,255,0), 3, LINE_AA);
}



imshow("Final Image", calcShape);
```

```
        char ch = waitKey(25);

        if(ch == 27) {
           break;
        }
        else if(ch == 's') {

           cvDestroyWindow("Final Image");

          // imwrite("shape.jpg", calcShape);

           if(squares.size() > 0) {

              putText(calcShape, "Square detected", cvPoint(30,30),
                    FONT_HERSHEY_COMPLEX_SMALL, 0.8, cvScalar(200,200,250), 1,
CV_AA);
              shape = 's';
              sprintf(buffer, "%c", shape);
              write(fd, buffer, 2);


              exit = 1;







           }
           else if(triangles.size() > 0) {

              putText(calcShape, "Triangle detected", cvPoint(30,30),
                    FONT_HERSHEY_COMPLEX_SMALL, 0.8, cvScalar(200,200,250), 1,
CV_AA);
              shape = 't';
              sprintf(buffer, "%c", shape);
              write(fd, buffer, 2);
```

```
            char what = '3';
            sprintf(buffer, "%c", what);
            write(fd, buffer, 2);



            exit = 1;


        }
        else if(diamonds.size() > 0){
            putText(calcShape, "Diamond detected", cvPoint(30,30),
                FONT_HERSHEY_COMPLEX_SMALL, 0.8, cvScalar(200,200,250), 1,
CV_AA);
            shape = 'd';
            sprintf(buffer, "%c", shape);
            write(fd, buffer, 2);


            exit = 1;
        }
        else if(pentagons.size() > 0) {
            putText(calcShape, "Irregular pentagon detected", cvPoint(30,30),
                FONT_HERSHEY_COMPLEX_SMALL, 0.8, cvScalar(200,200,250), 1,
CV_AA);
            shape = 'p';
            sprintf(buffer, "%c", shape);
            write(fd, buffer, 2);


            exit = 1;
        }
        else if(octagons.size() > 0) {
            putText(calcShape, "Octagon detected", cvPoint(30,30),
                FONT_HERSHEY_COMPLEX_SMALL, 0.8, cvScalar(200,200,250), 1,
CV_AA);
            shape = 'o';
            sprintf(buffer, "%c", shape);
```

```
            write(fd, buffer, 2);


            exit = 1;
         }
         else {
            putText(calcShape, "No shape detected", cvPoint(30,30),
                  FONT_HERSHEY_COMPLEX_SMALL, 0.8, cvScalar(200,200,250), 1,
CV_AA);




         }



      }

   }



   imshow("Final Image", calcShape);
      usleep(10000000);
   cvDestroyWindow("Final Image");



   Mat paper;
   Mat gray2;
   Mat bw;

   //while(1) {

      int matchx = 0;
      int matchy = 0;
      VideoCapture cap2(0);
```

```
        Point center;
        Point center2;

        int done = 0;
        int number;
       // while(matchx == 0 || matchy == 0) {

//        while(1) {
        cap2.read(paper);

//        imshow("Paper", paper);
//
//        char c = waitKey(25);
//
//        if(c == 's') {
//            break;
//        }
//
//    }
//
//    cvDestroyWindow("Paper");



        cvtColor(paper, gray2, CV_BGR2GRAY);
        blur(paper,paper, Size(3,3));
        Canny(gray2, bw, 50, 150, 3);

        vector<vector<Point> > contours;
        vector<Vec4i> hierarchy;

        findContours(bw, contours, hierarchy, CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE, Point(0,0));
//        vector<Point> approx;
        vector<vector<Point> >  squares;
        int thresh = 50, N = 11;
        Mat pyr, timg, gray0(paper.size(), CV_8U), g;
        pyrDown(paper, pyr, Size(paper.cols/2, paper.rows/2));
        pyrUp(pyr, timg, paper.size());
```

```
    for(int c = 0; c<3; c++) {
       int ch[] = {c,0};
       mixChannels(&timg, 1, &gray0, 1, ch, 1);
       for(int l = 0; l<N; l++) {
          if(l==0) {
             Canny(gray0, g, 0, thresh, 5);
             dilate(g,g,Mat(),Point(-1,-1));
                }
                else {
                   g = gray0 >= (l+1)*255/N;
                }
                findContours(g, contours, hierarchy, CV_RETR_TREE,
CV_CHAIN_APPROX_SIMPLE);
                vector<Point> approx;


                for(int i=0; i<contours.size(); i++) {

                   approxPolyDP(Mat(contours[i]), approx, arcLength(Mat(contours[i]),
true)*0.02, true);

                   if(approx.size() == 4 && fabs(contourArea(Mat(approx))) > 1000 &&
isContourConvex(Mat(approx))) {


                      double maxCosine = 0;

                      for(int j=2; j<5; j++){

                         double cosine = fabs(angle(approx[j%4], approx[j-2], approx[j-1]));
                         maxCosine = MAX(maxCosine, cosine);

                      }

                      if(maxCosine<0.3) {

                         squares.push_back(approx);
                      }
```

```
                    }
                }


            }
        }


//        for(int i=0; i<contours.size(); i++) {
//
//            approxPolyDP(Mat(contours[i]), approx, arcLength(Mat(contours[i]), true)*0.02, true);
//
//            if(approx.size() == 4 && fabs(contourArea(Mat(approx))) > 1000 &&
isContourConvex(Mat(approx))) {
//
//
//                double maxCosine = 0;
//
//                for(int j=2; j<5; j++){
//
//                    double cosine = fabs(angle(approx[j%4], approx[j-2], approx[j-1]));
//                    maxCosine = MAX(maxCosine, cosine);
//
//                }
//
//                if(maxCosine<0.3) {
//                    cout << fabs(contourArea(Mat(approx))) << endl;
//                    squares.push_back(approx);
//                }
//
//            }
//        }
//        cout << squares.size() << endl;
        for(int i=0; i <squares.size(); i++) {

//            if( (boundingRect(squares[i]).width*boundingRect(squares[i]).height) < 9000) {
//                bounding_rect = boundingRect(squares[i]);
```

```
//          }

          if( (boundingRect(squares[i]).width*boundingRect(squares[i]).height) > 9000 &&
boundingRect(squares[i]).width > 120) {
             bounding_rect = boundingRect(squares[i]);
          }
      // bounding_rect2 = boundingRect(squares[squares.size()-1]);
//          rectangle(paper, bounding_rect, Scalar(0,255,0), 1, 8, 0);

      }


      rectangle(paper, bounding_rect, Scalar(0,255,0), 1, 8, 0);
//       rectangle(paper, bounding_rect2, Scalar(0,255,0), 1, 8, 0);

      center = Point((bounding_rect.x + (bounding_rect.width/2)), (bounding_rect.y +
bounding_rect.height/2));

//          if(done == 0) {
//       center2 = Point((bounding_rect2.x + (bounding_rect2.width/2)), (bounding_rect2.y +
bounding_rect2.height/2));
//          done = 1;
//          }

      circle(paper, center, 1, CV_RGB(255,0,0),3);
      //circle(paper, center2, 1, CV_RGB(255,0,0),3);
//          if(matchy == 0) {
//          if(abs((center.y+14)-center2.y) <= 5){
//            cout << "MATCH Y" << endl;
//             matchy = 1;
//             char toSend = 'y';
//             for(int i = 0; i <100; i++) {
//             sprintf(buffer, "%c", toSend);
//             write(fd, buffer, 2);
//             }
//             //usleep(1000000);
//          }
//          }
//          if(matchx == 0) {
```

```
//          if(center2.x>760) {
//
//             if(abs((center.x-30)-center2.x) <=5) {
//                cout << "MATCH X 1" << endl;
//                matchx = 1;
//                char toSend = 'x';
//                sprintf(buffer, "%c", toSend);
//                write(fd, buffer, 2);
//                usleep(1000000);
//             }
//          }
//
//
//          if(center2.x<670) {
//          if(abs((center.x+30)-center2.x) <=5) {
//             cout << "MATCH X 2" << endl;
//             matchx = 1;
//             char toSend = 'x';
//             sprintf(buffer, "%c", toSend);
//             write(fd, buffer, 2);
//             usleep(1000000);
//          }
//          }
//
//          if(center2.x<=760 && center2.x>=670) {
//             if(abs(center.x-center2.x) <=5) {
//                cout << "MATCH X 3" << endl;
//                matchx = 1;
//                char toSend = 'x';
//                sprintf(buffer, "%c", toSend);
//                write(fd, buffer, 2);
//                usleep(1000000);
//
//             }
//               }
//          }
```

```
            cout << "Width: " << bounding_rect.width << " Height: " << bounding_rect.height <<
endl;
//            cout << "Width: " << bounding_rect2.width << " Height: " << bounding_rect2.height
<< endl;
 //           cout << "Center at: (" << center.x << ", " << center.y << ")" << endl;
            cout << "Paper center at: (" << center.x << ", " << center.y << ")" << endl;




        //}
//        matchx = 0;
//        matchy = 0;

        namedWindow("Final");
        //setMouseCallback("Final", onMouse, 0);


        imshow("Final", paper);

        double tempx = (center.x-65-341)*14.7959184;
        //int finalx = (int)tempx;
        int finalx = center.x;

        double tempy = (center.y+14-261)*16.8181818;
       // int finaly = (int)tempy;
        int finaly = center.y;


        int testWidth = 398;

        int temp = finaly;

        do{
            temp /= 10;
            finalx *= 10;

        } while(temp>0);
```

```
        int final = finalx + finaly;

    int choose;
    if(bounding_rect.width < bounding_rect.height) {
        choose = bounding_rect.width;
    }
    else {
        choose = bounding_rect.height;
    }

    int temp2 = choose;

    do{
        temp2 /= 10;
        final *= 10;
    }
        while(temp2>0);

        int newfinal = final +choose;

    cout << newfinal << endl;


    //char buffer[2];
char test1 = '3';
char test2 = '9';
char test3 = '8';

char target[9];
    char target2[3];
    //char target3[6];
sprintf(target, "%03d", newfinal);

    //sprintf(target2, "%03d", finaly);




    sprintf(buffer, "%c", target[0]);
```

```
    write(fd, buffer, 2);
usleep(1000000);



sprintf(buffer, "%c", target[1]);
write(fd, buffer, 2);
usleep(1000000);



sprintf(buffer, "%c", target[2]);
write(fd, buffer, 2);
    usleep(1000000);


    sprintf(buffer, "%c", target[3]);
    write(fd, buffer, 2);
    usleep(1000000);



    sprintf(buffer, "%c", target[4]);
    write(fd, buffer, 2);
    usleep(1000000);



    sprintf(buffer, "%c", target[5]);
    write(fd, buffer, 2);
    usleep(1000000);

    sprintf(buffer, "%c", target[6]);
    write(fd, buffer, 2);
    usleep(1000000);

    sprintf(buffer, "%c", target[7]);
```

```
        write(fd, buffer, 2);
        usleep(1000000);

        sprintf(buffer, "%c", target[8]);
        write(fd, buffer, 2);
        usleep(1000000);









//        cout << "Width: " << bounding_rect.width << " Height: " << bounding_rect.height <<
endl;
//        cout << "Width: " << bounding_rect2.width << " Height: " << bounding_rect2.height <<
endl;
//        cout << "Center at: (" << center.x << ", " << center.y << ")" << endl;
//        cout << "Center2 at: (" << center2.x << ", " << center2.y << ")" << endl;


        usleep(10000000);

        cvDestroyWindow("Final");









    }

    return 0;


}
```