```
//Kurtis Rickelmann
//UR2 Term Project Final Code C#/Emgu Side
//Due 12/7/2016

using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;
using Emgu.CV;
using Emgu.CV.Structure;
using System.IO.Ports;
using System.Threading;

namespace Term_Project_Final_Official
{
    public partial class Form1 : Form
    {
        SerialPort serPort;
        string comPort = "COM4"; //Com port for the Arduino Mega

        Capture _capture = null;
        Image<Gray, Byte> img = null;
        int threshold = 80;
        byte[] send_Byte = new Byte[4]; //bytes to send to the Arduino
        int shapes_picked = 0; //counter for number of shapes picked up

        public Form1()
        {
            InitializeComponent();

            serPort = new SerialPort(comPort);
            serPort.BaudRate = 115200;
            serPort.DataBits = 8;
            serPort.Parity = Parity.None;
            serPort.StopBits = StopBits.One;
            serPort.Open();
            serPort.DataReceived += new SerialDataReceivedEventHandler(serPort_DataReceived);

            this.KeyPreview = true;
            this.KeyDown += Form1_KeyDown;
        }

        //Function to deal with a serial byte received
        void serPort_DataReceived(object sender, SerialDataReceivedEventArgs e)
        {

            int num = serPort.ReadByte(); // Read the serial byte
            if (shapes_picked > 4) // If there are more than 4 shapes picked, do nothing
                                   // The robot will stop until the program is restarted
                ;
            else if (num == 1) //If there haven't been more than 4 objects picked,
                               //Send the next shape location
            {
                serPort.Write(send_Byte, 0, 1);
                serPort.Write(send_Byte, 1, 1);
                serPort.Write(send_Byte, 2, 1);
                serPort.Write(send_Byte, 3, 1);
            }
            shapes_picked++; //increment shapes picked
        }

        //Adjusts the B/W threshold when the up/down arrow keys are pressed
        void Form1_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Up)
            {
                threshold += 5;
                if (threshold >= 255)
                    threshold = 255;
            }
            if (e.KeyCode == Keys.Down)
            {
                threshold -= 5;
                if (threshold <= 0)
                    threshold = 0;
            }
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            label1.Text = null; //Initialize labels to 'null'
            label2.Text = null; //Initialize labels to 'null'
            label3.Text = null; //Initialize labels to 'null'
            label4.Text = null; //Initialize labels to 'null'
            label5.Text = null; //Initialize labels to 'null'
            label6.Text = null; //Initialize labels to 'null'
            send_Byte[3] = 255; //Initialize the send_byte to 255 for triangle

            _capture = new Capture(1);
            _capture.ImageGrabbed += Display_Captured;  // Event Handler
            _capture.Start();

            imageBox1.MouseClick += ibox_MouseClick;
        }

        //Event handler for displaying an image
        void Display_Captured(object sender, EventArgs e)
        {
            img = _capture.RetrieveGrayFrame().Resize(
                    imageBox1.Width, imageBox1.Height,
                    Emgu.CV.CvEnum.INTER.CV_INTER_LINEAR);
            img = img.ThresholdBinary(new Gray(threshold), new Gray(255));

            imageBox1.Image = img;

            img = img.SmoothGaussian(5); // Smooth the image
            img = img.Canny(1, 1); // Canny edge detection

            WarpImage(sender, e); //This function will warp the image to the necessary shape
                                  //if 4 points on the image have been selected for corners of the warp

            FindPoly(sender, e); //This finds the shapes and sets the angle values accordingly

        }

        private void FindPoly(object sender, EventArgs e)
        {
            Image<Gray, Byte> gray = img.Convert<Gray, Byte>().PyrDown().PyrUp(); // to filter out noise
            Image<Gray, Byte> cannyEdges = gray.Canny(100, 255);
            List<Triangle2DF> triangleList = new List<Triangle2DF>();
            List<MCvBox2D> boxList = new List<MCvBox2D>();
            // a contour: list of pixels that can represent a curve
            using (MemStorage storage = new MemStorage()) //allocate storage for contour approximation
            {
                for (Contour<Point> contours = cannyEdges.FindContours(); contours != null; contours =     ↙
    contours.HNext)
                {
                    Contour<Point> currentPolygon = contours.ApproxPoly(contours.Perimeter * 0.1, storage);↙
     // adjust here
                    if (contours.Area > 1000 && contours.Area < 50000) //only consider contours with area  ↙
    greater than 1000 and less than 50000
                    {
```

```
                    if (currentPolygon.Total == 3) //The contour has 3 vertices, it is a triangle
                    {
                        Point[] pts = currentPolygon.ToArray();
                        triangleList.Add(new Triangle2DF(pts[0], pts[1], pts[2]));
                    }
                    else if (currentPolygon.Total == 4) //The contour has 4 vertices.
                    {
                        Point[] pts = currentPolygon.ToArray();
                        boxList.Add(currentPolygon.GetMinAreaRect());
                    }
                }
            }
        } // end using

        Image<Bgr, Byte> bgr_display = img.Convert<Bgr, Byte>();

        bool triangle_avail = false; // Variable to indicate a triangle is in the trianglelist
        int base_angle = 90; // Default servo angle
        int shoulder_angle = 45; // Default servo angle
        int elbow_angle = 90; // Default servo angle

        foreach (Triangle2DF triangle in triangleList)
        {
            triangle_avail = true; // If there's a triangle, set true
            bgr_display.Draw(triangle, new Bgr(Color.Yellow), 2); // Draw a yellow triangle where the ↙
    triangle is
            // This will show the current triangle the robot will pick
            PointF center = triangle.Centeroid;

            double x_pos = center.X / imageBox1.Width * 11 - 5.5; // x position from the robot ↙
    centerline in inches
            double y_pos = 18 - center.Y / imageBox1.Height * 8.5; // y position from the robot base in↙
     inches


            dispStr1("X: " + x_pos.ToString()); // Display the x position
            dispStr2("Y: " + y_pos.ToString()); // Display the y position

            base_angle = Convert.ToInt32((Math.Atan(x_pos / y_pos) / 3.14 * 180) + 90); // calculate ↙
    base servo angle

            double dist_from_base = Math.Sqrt(x_pos * x_pos + (y_pos + 0.75) * (y_pos + 0.75)); // ↙
    distance from the base
            // the  "+0.75" was added during testing for more accurate picks ↙


            if (dist_from_base > 17.5) //If the distance is too large, simply set the angle to 0 or the↙
     cosine function will fail
                shoulder_angle = 0;
            else
                shoulder_angle = Convert.ToInt32(Math.Acos(dist_from_base / (2 * 8.75)) / 3.14 * 180); ↙
    //calculate shoulder servo angle

            elbow_angle = shoulder_angle * 2; //calculate elbow servo angle

            dispStr3("Base: " + base_angle.ToString()); // Display the base servo angle
            dispStr4("Shoulder: " + shoulder_angle.ToString()); // Display the shoulder servo angle
            dispStr5("Elbow: " + elbow_angle.ToString()); // Display the elbow servo angle

            //Set the values for the send_bytes for when a serial input occurs
            send_Byte[0] = Convert.ToByte(base_angle);
            send_Byte[1] = Convert.ToByte(shoulder_angle);
            send_Byte[2] = Convert.ToByte(elbow_angle);
            send_Byte[3] = Convert.ToByte(255);

            break; // break to only handle one triangle in case there are multiple
        }

        if (!triangle_avail) // If there were no triangles found, search for squares
        {
            foreach (MCvBox2D box in boxList)
            {
                bgr_display.Draw(box, new Bgr(Color.DarkOrange), 2); // Draw an orange square where the ↙
 square is
                // This will show the current square the robot will pick

                PointF center = box.center;

                double x_pos = center.X / imageBox1.Width * 11 - 5.5; // x position from the robot      ↙
centerline in inches
                double y_pos = 18 - center.Y / imageBox1.Height * 8.5; // y position from the robot     ↙
base in inches


                dispStr1("X: " + x_pos.ToString()); // Display the x position
                dispStr2("Y: " + y_pos.ToString()); // Display the y position

                base_angle = Convert.ToInt32((Math.Atan(x_pos / y_pos) / 3.14 * 180) + 90); //         ↙
calculate base servo angle

                double dist_from_base = Math.Sqrt(x_pos * x_pos + (y_pos + 0.75) * (y_pos + 0.75)); // ↙
distance from the base
                // the  "+0.75" was added during testing for more accurate picks                        ↙


                if (dist_from_base > 17.5) //If the distance is too large, simply set the angle to 0 or↙
 the cosine function will fail
                    shoulder_angle = 0;
                else
                    shoulder_angle = Convert.ToInt32(Math.Acos(dist_from_base / (2 * 8.75)) / 3.14 *    ↙
180); //calculate shoulder servo angle

                elbow_angle = shoulder_angle * 2; //calculate elbow servo angle

                dispStr3("Base: " + base_angle.ToString()); // Display the base servo angle
                dispStr4("Shoulder: " + shoulder_angle.ToString()); // Display shoulder the servo angle
                dispStr5("Elbow: " + elbow_angle.ToString()); // Display the elbow servo angle

                //Set the values for the send_bytes for when a serial input occurs
                send_Byte[0] = Convert.ToByte(base_angle);
                send_Byte[1] = Convert.ToByte(shoulder_angle);
                send_Byte[2] = Convert.ToByte(elbow_angle);
                send_Byte[3] = Convert.ToByte(254);

                break; // break to only handle one square in case there are multiple
            }
        }

        imageBox2.Image = bgr_display.Resize(imageBox2.Width, imageBox2.Height, Emgu.CV.CvEnum.INTER.   ↙
CV_INTER_LINEAR);
        //display the image with the highlighted shape to be picked next


    }

    int mouse_click_count = 0;
    PointF[] p1 = new PointF[4];
    PointF[] p2 = null;

    //This function warps the image if there have been 4 corners clicked on the original image
    void WarpImage(object sender, EventArgs e) // p trans
    {
        if (mouse_click_count < 4) // not clicked all 4.
            return;

        p2 = new PointF[]   // points of the flattened image
             { new PointF(0, 0), new PointF(imageBox1.Width, 0), new PointF(0, imageBox1.Height), new ↙
PointF(imageBox1.Width, imageBox1.Height) };
        HomographyMatrix homography = CameraCalibration.GetPerspectiveTransform(p1, p2);
        MCvScalar s = new MCvScalar(0, 0, 0); // Managed Scalar class written in C

        CvInvoke.cvWarpPerspective(img, img, homography, (int)Emgu.CV.CvEnum.INTER.CV_INTER_LINEAR, s);
    }

    //Event handler for imagebox1 mouse click
    void ibox_MouseClick(object sender, MouseEventArgs e)
    {
        if (mouse_click_count > 3)
            return;
        p1[mouse_click_count] = new PointF(e.Location.X, e.Location.Y);
        mouse_click_count++;
    }

    //Trackbars were used for testing, and I didn't delete them from the GUI
    //in case the arduino was acting up. I could switch from the image detection to the
    //trackbars to make sure the serial to arduino to servo still worked correctle
    private void trackBar1_Scroll(object sender, EventArgs e)
    {
        send_Byte[0] = Convert.ToByte(trackBar1.Value);
    }
    private void trackBar2_Scroll(object sender, EventArgs e)
    {
        send_Byte[1] = Convert.ToByte(trackBar2.Value);
    }
    private void trackBar3_Scroll(object sender, EventArgs e)
    {
        send_Byte[2] = Convert.ToByte(trackBar3.Value);
    }

    //Start button, sends the most recent shape and location
    //to start the program
    private void button1_Click(object sender, EventArgs e)
    {
        serPort.Write(send_Byte, 0, 1);
        serPort.Write(send_Byte, 1, 1);
        serPort.Write(send_Byte, 2, 1);
        serPort.Write(send_Byte, 3, 1);
    }

    //Sets the angles to the home position and sends te information to the Arduino
    private void button2_Click(object sender, EventArgs e)
    {
        send_Byte[0] = 90;
        send_Byte[1] = 90;
        send_Byte[2] = 90;
        serPort.Write(send_Byte, 0, 1);
        serPort.Write(send_Byte, 1, 1);
        serPort.Write(send_Byte, 2, 1);
        serPort.Write(send_Byte, 3, 1);
    }

    //The below 6 functions are used to write to the labels in the GUI
    // shen inside of a function that can't directly write to the labels
    delegate void displayStringDelegate(String s);
    public void dispStr1(string s)
    {
        if (label1.InvokeRequired)
        {
            displayStringDelegate dispStrDel = dispStr1;
            this.BeginInvoke(dispStrDel, s);
        }
        else
            label1.Text = s;
    }
    public void dispStr2(string s)
    {
        if (label2.InvokeRequired)
        {
            displayStringDelegate dispStrDel = dispStr2;
            this.BeginInvoke(dispStrDel, s);
        }
        else
            label2.Text = s;
    }
    public void dispStr3(string s)
    {
        if (label3.InvokeRequired)
        {
            displayStringDelegate dispStrDel = dispStr3;
            this.BeginInvoke(dispStrDel, s);
        }
        else
            label3.Text = s;
    }
    public void dispStr4(string s)
    {
        if (label4.InvokeRequired)
        {
            displayStringDelegate dispStrDel = dispStr4;
            this.BeginInvoke(dispStrDel, s);
        }
        else
            label4.Text = s;
    }
    public void dispStr5(string s)
    {
        if (label5.InvokeRequired)
        {
            displayStringDelegate dispStrDel = dispStr5;
            this.BeginInvoke(dispStrDel, s);
        }
        else
            label5.Text = s;
    }
    public void dispStr6(string s)
    {
        if (label6.InvokeRequired)
        {
            displayStringDelegate dispStrDel = dispStr6;
            this.BeginInvoke(dispStrDel, s);
        }
        else
            label6.Text = s;
    }
}
}
```