Advanced Robot Control Project
Team ROBOCAR

By Ian Cairns

Introduction:

For this project I wanted to work on trying to extract information from images. I have been very interested in trying to build a robot that could in some way incorporate using images to gain information. Before this class I had never used a computer to capture an image or try to analyze one. For this project I decided to try and build a car that follows someone around. I wanted to be able to walk away and have it follow me as well as walk towards the robot and it would back away. I specifically wanted to do this by using vision instead of using some sort of sensor and receiver. I spent some time researching different language options for doing this. It seemed to boil down to a couple good options; Matlab, Labview, OpenCv (python or C). I chose to go with Matlab because I had been working with it most in school although I would love to try in both OpenCV and Labview. Matlab also had great documentation for working with their vision tool box libraries. I had a chassis at home that I thought would work as a good platform and all the components required to control it.

Methods:

To run the vision software I wanted to use my computer thus the robot chassis would have to be large enough to do this. I had an RC truck at home that I purchased a while ago that I though would be perfect for this. The truck is a standard hobby grade car. It comes with a RC controller and receiver. It is extremely easy to take the receiver out of the car and use an Arduino to control it. Receivers send out a signal that is the same as the signal used to run a hobby servo. It sends a pulse that ranges between 1 and 2 ms with a neutral position at 1.5ms. Arduino conveniently has libraries that will send out this pulse on certain pins. Thus by placing the Arduino where the receiver was the Arduino can control the car. I then used Matlab to tell the Arduino how to steer the car. In this case the Arduino is essentially just a cheap DAQ. A diagram of the system information flow is shown below.

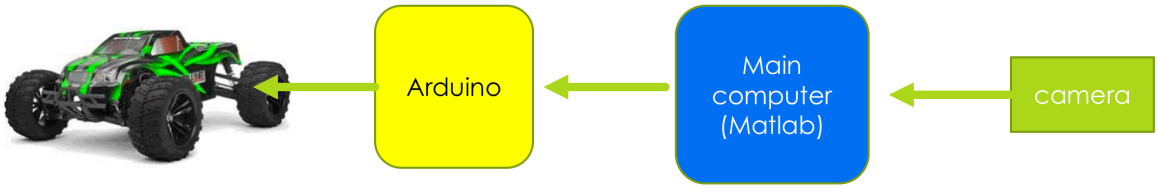

FIGURE 1: Information flow

The wiring to put the Arduino in the system is pretty simple. I used a bread board jumper cable that had 3 connections to connect the grounds of the Arduino, ESC and servo for steering. Then I connected the 5v power coming out of the ESC to the servo. Originally this 5v would have been connected to the receiver which passes it directly to the servo. The next step was to connect the signal wires of both the ESC and the Servo to the Arduino. Two jumper wires were used for this. Both signal wires were connected to Arduino pins capable of PWM. An image of this is shown below along with the wiring diagram.

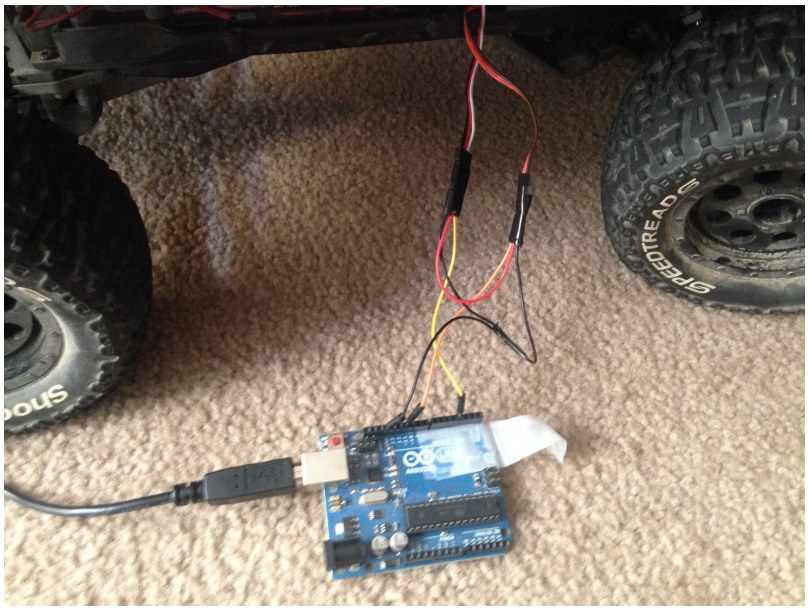

FIGURE2: Arduino wired into car

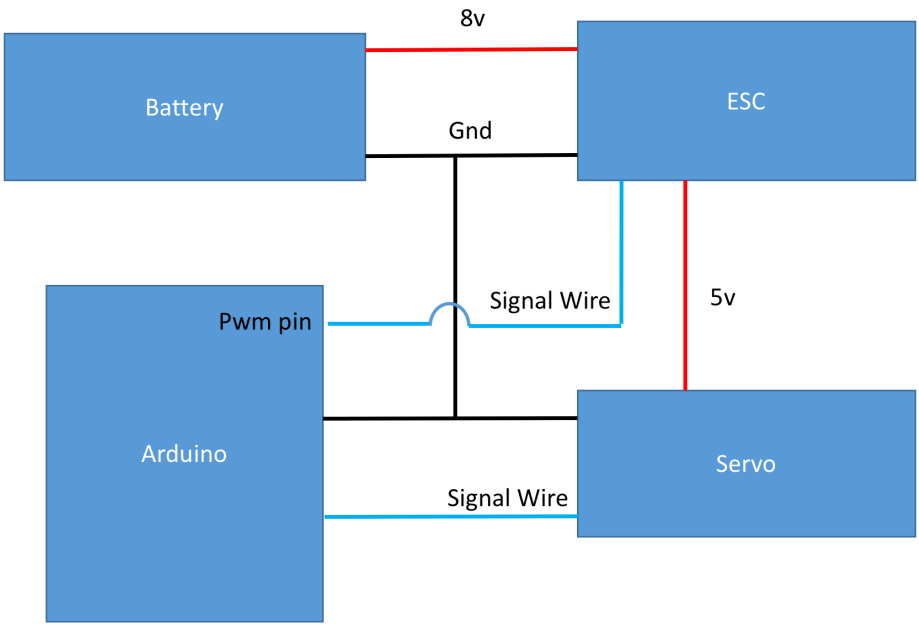

FIGURE 3: Electrical Schematic

Once the Arduino was wired into the car the computer was connected by USB cable which powered the Arduino. A piece of cardboard was mounted onto the top of the car so that the computer could sit without falling off. A photo of the computer on the car is shown below.

FIGURE 4: Computer on top of car

At this point the car was ready to be controlled. Now code needed to be developed that would allow Matlab to send information over serial to the Arduino which would then drive the car. To do this I created a code for the Arduino that checks information on the serial port. The numbers come in as an ascii string where two number are separated by a comma. The Arduino code then parses out these two numbers and converts then into integer values which it then uses as its goal for what to set the speed to. Instead of directly writing that number the Arduino increments from the number it is currently at to the number that is its goal. I did this because if the computer tried to have the car go between forward and reverse too quickly the esc would stop the motors. The Arduino code also checks if it hasn't received information from Matlab. If it hasn't received information for 2 seconds the Arduino tells the motors to stop. This helps prevent any run away cars!! This finished the Arduino code and then I was ready to move onto the Matlab side of things.

The first step I took with Matlab was to create three functions that would handle the car. The first one opens the serial port for the Arduino. The second one is the main one. It takes the values passed to it and caps them at the cars max allowed speed. Again, this prevents run away robots. After that it puts the numbers into a string and sends them out the serial port. The last function closes the serial port.

After this the car was fully controllable by Matlab and I was ready to begin the image processing portion of the project. To begin the image processing portion of the project I spent time researching the different detection algorithms in Matlab. I ended up thinking three methods could work well. They had a checkerboard detection library, a color detection library and a person detection library. I began with the person detection code but I did not have good luck detecting a person. To do this there is a function in Matlab that runs a people detection algorithm. Below is an image produced using code I wrote trying to detect people.

FIGURE 5: People Detection

As you can see the red box is not over any of the people. I then tried the Matlab face detection software and found it worked much better. Below is an image of my code finding my face.

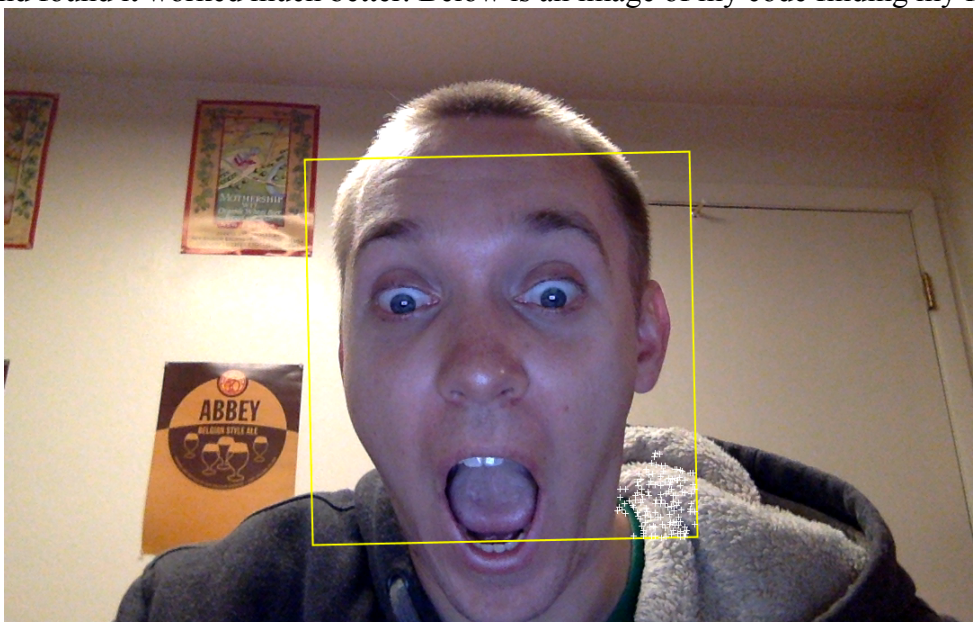

FIGURE 6: Face Detection

This code works by first making a gray image and then running the Matlab function for finding a face. This function then outputs a series of points used to track the face. Then once the code finds a face in the future frames it tracks the points found. It does this till it loses those points and then starts looking for a face again.

After getting the code to pick out my face I then found the center of the face by finding the center of the bounding box. Using the known center of the face I decided the easiest way to control the car was to have the car move back if the face is in the upper portion of the picture and move forward if the face is in the lower portion of the picture. This scheme was decided upon

based on geometry. Because the camera is facing up around 25 degrees to see a face, if I stand farther back from the car I will be lower in the picture. Turning took the same approach. If I was right in the picture and down it turns right as it goes forward. If it is down and left it goes forward and left. For turning the streaming is reverse when going backwards. If the face is in the left upper picture it turns the wheels right and backs up. This keeps the face in the picture no matter how you move. The whole scheme operates in a loop and every incoming picture is analyzed and sets a new target speed and turn.

This scheme worked pretty well, but was not perfect. It takes a long time to process and would frequently pick out things that were not faces. Once this happened it would hold on to that wrong solution until something, like my hand, would move in front of the object. This is because once it finds what it thinks is a face it tracks the points it found until they disappear.

After using the face detection method I decided to try using checkerboard detection to find where the person was. In this case the person can carry the checkerboard or have one on their back.

To make this code I used the “detectcheckerboardpoints” function. This comes back with a matrix of points if there is a checkerboard in the image. I was then able to use the size of the checkerboard detected to determine if it was my checkerboard. This allowed me to have almost zero false detections. For this project false detections are far worse then false negatives. A false positive can lead to a run away robot. False negatives do nothing. Matlab will also calculate the distance to the checkerboard. To use this, you must first calibrate the camera in Matlab. Below is a screen shot of this.

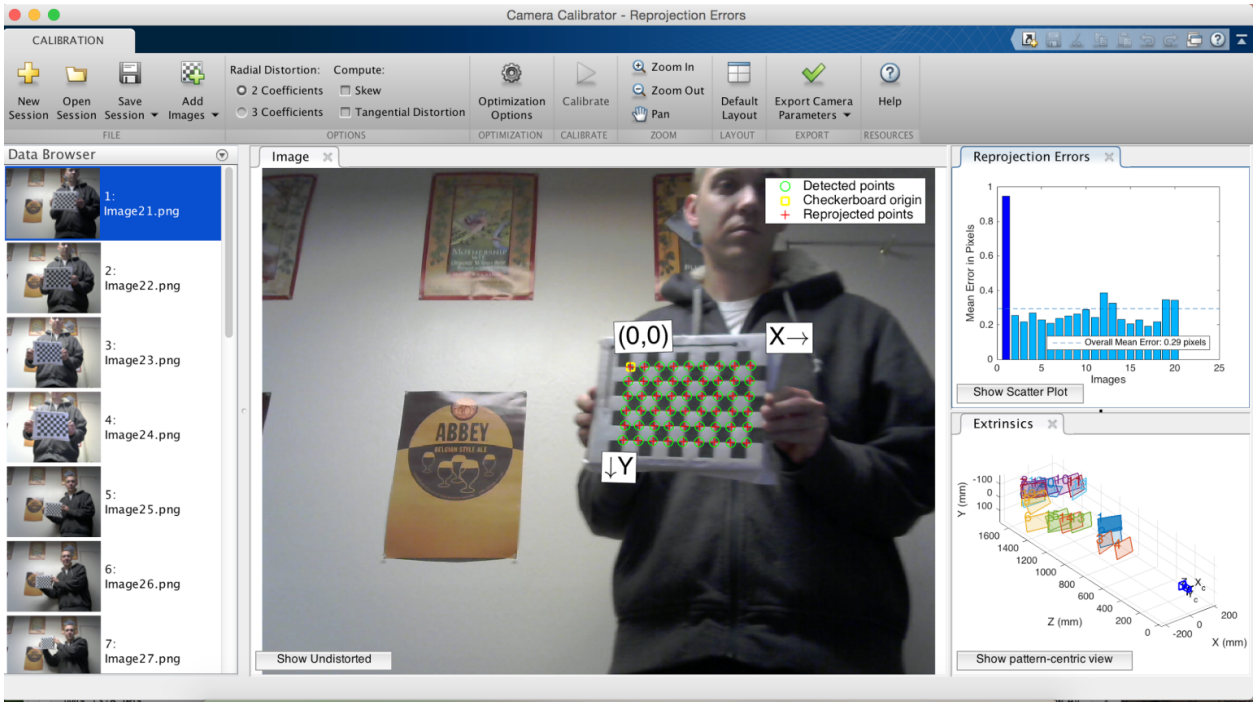

FIGURE 7: Matlab Camera Calibration

This software is pretty cool. You need to take pictures with the checkerboard in different locations in each picture. It then develops camera parameters. It also will tell you the statistical

deviation of each picture (blurry pictures are the main reason for high deviations). It also makes a 3d plot of where each checkerboard was in each picture.

Once you have saved the camera parameters created using this software you can use the checker board extrinsic function to determine the location of the checker board. In my code I just print this out. It is not used for driving the robot. My code still works off determining the center of the checker board and then using the same algorithm as the face detection to drive the robot. Below is an image of my code finding the checkerboard and determining the distance to the object.

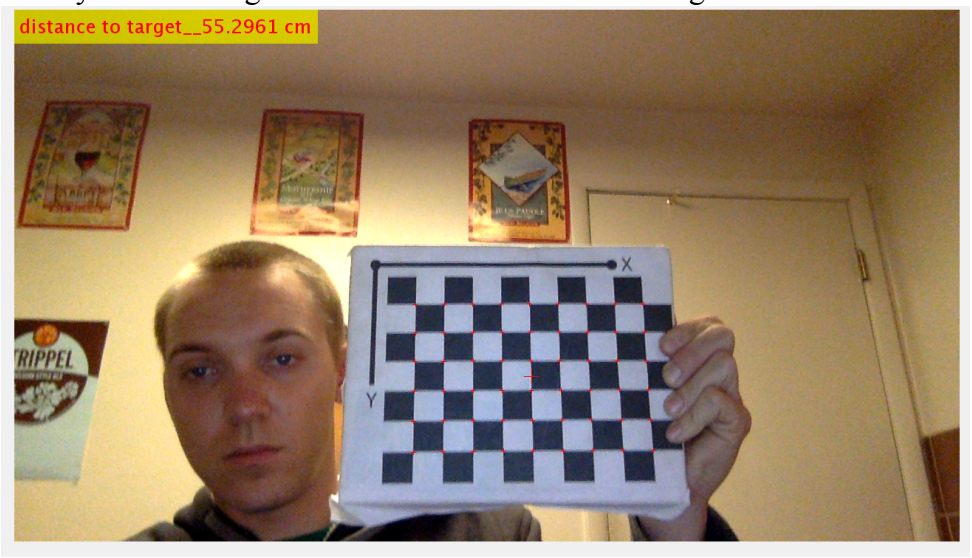

FIGURE 8: My code

It may be hard to see but there are red crosses at the edges of each square and there is a red plus in the center. There is also a distance measurement printed to the top of the screen. The distance measurements ended up being very accurate. I could not find an error using a tape measure. I would need a more precise measurement system.

This code ended up working pretty well with almost no false positives. The biggest issue is processing time for finding the checkerboard. The other big issue is that it can't find the checkerboard in blurry images. This could be mitigated if I had a quicker exposure camera. I used the webcam built into my mac.

The last scheme I tried out was color detection. My thought was I could wear a construction vest that would hopefully be unique enough to spot. To do this I split the incoming photo into each color band (red, blue, green). Then the code checks for the areas in the photo where each color band is within the tolerances defined for the target color. It then gets rid of all the small areas detected and fills in any small holes in the bigger objects. It then finds the biggest blob using the "regionprops" function. Then the code finds the blob center and uses this as the image location used to dive the car. Once the center is found it uses the same code used in the face and check board detection to driver the car. Below is a photo of the code finding the construction vest color.

FIGURE 9: Color Detection

This code is extremely touchy. Picking out a color in a single image is easy. The code works well when you know the color of the object. The biggest problem is that depending on the lighting the construction vest has very different color properties. If I open up the tolerances enough to always see it then I get tons of false positives. And if I set them up too tight then it needs the perfect lighting to see it. I played around with it for a while but in the end the limits need to be dynamic where it senses how light the picture is and then changes it. It does not work well by setting ranges of color to look for.

Results

When the car was completed it could do the objectives I had set. It was able to successfully follow a person by extracting data from an image. The best scheme for doing this was the checker board detection. This software worked pretty well. The biggest problems were the processing speed for detecting the checkerboard as well as blurry images from the camera. If I had a faster exposure time for the camera I think the system would work even better. Currently the person can only move slowly because of these problems, but if the speed of the detection were faster it could eventually be possible to follow someone on a bike or something faster.

The face detection software was the second best way to detect and the color detection did not work well enough to effectively use. Face detection did have false positives but it was not that frequent. If there was a person in the image the software would correctly work about 85 to 90 percent of the time. The color detecting software basically wasn't usable because it had so many false positives. It was extremely difficult to find color thresholds that worked in different lighting. Even in the same room shadows and position with respect to the light made a huge difference. I think some sort of learning algorithm for setting color thresholds would be needed. It is not a simple problem because even in the same room if the object is in different locations the color is notably different and requires significantly different threshold values.

Future Work:

In this project I learned a lot and developed an even stronger passion for working on robots. I would like to continue working on using some of the skills I gained in this project. My next big goal is to start working with navigation. I began working on a navigation code that can navigate a robot through a room of unknown obstacles to a desired location. I came up with an algorithm I

thought could work while sitting in Advanced Robots when we first started talking about navigation. Using a grid type world where you can move up, down, right, left, and diagonal the robot calculates the shortest direction to go to get to the goal. Then it checks if there is anything in front of it (this would be like looking at the distance sensors). If there is an obstacle, it checks the locations +-45,+-90 and if any of those positions are open it picks the closest one. Using this algorithm, it can negotiate many different obstacle scenarios and get to the destination. A couple screen shots are shown below of the code in action.

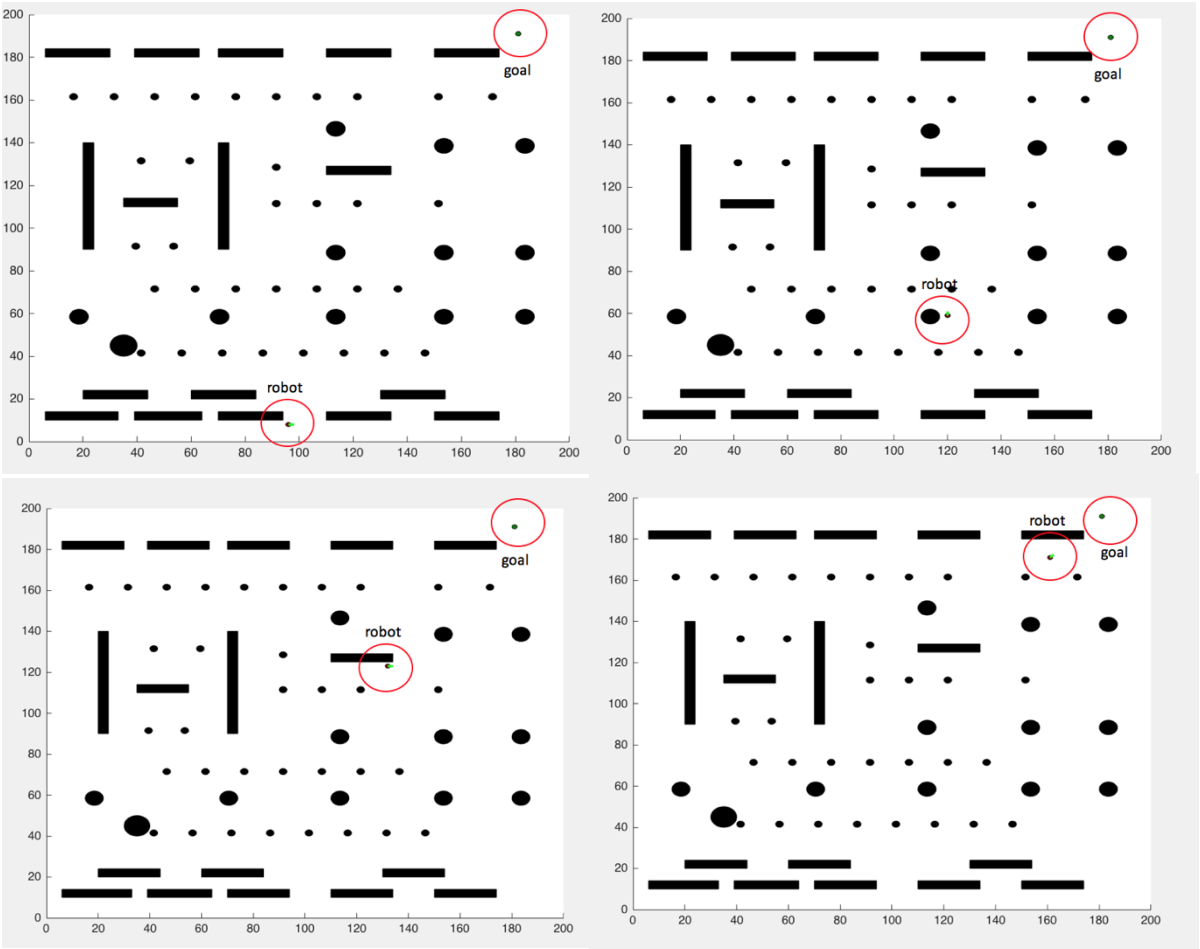

FIGURE 10: Navigation Code

I would like to use this navigation scheme to navigate a real robot. My plan is to use the checker board detection in Matlab for robot localization. I would like to mount a camera on the ceiling facing down. Then put a checkerboard on the robot and use Matlab checkerboard detection to find where it is. I could then control the car from Matlab the same way I did in this current project. I could use a wireless serial device such as Zigbee or Bluetooth. I would have several IR sensors mounted on the vehicle and/or sweep the IR sensors with a servo so it can see 180. This will be used as the check of whether anything is at the next position.

My algorithm would be for completely unknown environments, but I would also like to start working with some of the other algorithms, such as A* and wildfire, once I map the environment.

Reference:

- Shows the syntax and functions for the Arduino coding language.

"Arduino - Reference." *Arduino - Reference*. Web. 8 May 2016. <https://www.arduino.cc/en/Reference/HomePage>.

- Shows many examples for matlab vision software. I used the checker board detection, face detection and color detection examples.

"Matlab - Examples" *Image Processing and Computer Vision Examples*. Web. 8 May 2016. <http://www.mathworks.com/examples/product-group/matlab-image-processing-and-computer-vision>.

- This citation shows how to control an ESC with an arduino

"ESC Programming on Arduino (Hobbyking ESC)." *Instructables.com*. Web. 8 May 2016. <http://www.instructables.com/id/ESC-Programming-on-Arduino-Hobbyking-ESC/>.

- Shows how to control a servo using an arduino

"Arduino - Servo." *Arduino - Servo*. Web. 8 May 2016. <https://www.arduino.cc/en/Reference/Servo>.

Appendix:
Arduino CODE

```
#include <Servo.h>

Servo thrservo;
Servo turnservo;
int thr=1500;
int turn=1500;
int thr2=1500;
int turn2=1500;
int DIR[2];
int i=0;

void setup() {
  Serial.begin(9600);

  thrservo.attach(11);
  turnservo.attach(3);
  thrservo.write(1500);
  turnservo.write(1500);

}

void loop() {
  i=i+1;
  while (Serial.available() > 0) {
     //decifer the incoming serial string
     String thrstr = Serial.readStringUntil(',');
     String turnstr = Serial.readStringUntil('\n');

     thr= thrstr.toInt() ;//string to integer
     turn= turnstr.toInt() ;

     //cap velocity to prevent run away robot
     if(thr>1680){
      thr=1680;
     }
     else if(thr<1300){
      thr=1300;
     }
     if(turn>2000){
      turn=2000;
     }
```

```
   else if(turn<1000){
    turn=1000;
   }


   //thrservo.write(thr);
   //turnservo.write(turn);
   Serial.println(turn);
   i=0;
}
delay(3);

//iterate to get to goal this keeps the
//ESC from faulting
if(thr!=thr2){
    thr2=((thr-thr2)/abs(thr-thr2))+thr2;
}
if(turn!=turn2){
    turn2=((turn-turn2)/abs(turn-turn2))+turn2;
   }
//if serial is lost set throttle to 1500
if (i>650){
  thr=1500;
  thr2=1500;
  //thrservo.write(thr);
}
    //write to servo and esc
   thrservo.write(thr2);
   turnservo.write(turn2);
}
```

Matlab Checkerboard Detection CODE

```
%main code
function main
load cameraParams
cam = webcam('FaceTime HD Camera');
s=serial_init;
thr=1500;
turn=1500;
squareSize = 20;


% Capture one frame to get its size.
videoFrame = snapshot(cam);
frameSize = size(videoFrame);

% Create the video player object.
videoPlayer = vision.VideoPlayer('Position', [100 100 [frameSize(2),
frameSize(1)]+30]);
runLoop = true;
numPts = 0;
frameCount = 0;
Xcenter=frameSize(2)/2;
Ycenter=frameSize(1)/2;

while runLoop && frameCount < 200
    frameCount = frameCount + 1;

    % Read in image into an array.
    [rgbImage storedColorMap] = snapshot(cam);
    [rows columns numberOfColorBands] = size(rgbImage);
    videoFrame = rgbImage;

    [imagePoints,boardSize] = detectCheckerboardPoints(rgbImage);
    S=std(imagePoints);
    if size(imagePoints)>0 & S(1)>10 & S(2)>10 & max(boardSize)==10 &
min(boardSize)==7
        videoFrame = insertMarker(videoFrame, imagePoints, '+', 'Color',
'red');
        centroids=mean(imagePoints);
        centroidsX = centroids(1);
        centroidsY = centroids(2);
        Xdelta=centroidsX-Xcenter;
        Ydelta=centroidsY-Ycenter;

        thr=1500-Ydelta;
        turn=1500+Xdelta*sign(Ydelta);
        ArdWrite(s,thr,turn);
        videoFrame = insertMarker(videoFrame, [centroidsX, centroidsY], '+',
'Color', 'red','size',10);


        worldPoints = generateCheckerboardPoints(boardSize, squareSize);
```

```
        [rotationMatrix, translationVector] = extrinsics(imagePoints,
worldPoints, cameraParams);
        checkerdis=translationVector(3)/10;
        disch=num2str(checkerdis);
        text=strcat('distance to target__',' ',disch,' cm');
        position=[0,0];

        videoFrame =
insertText(videoFrame,position,text,'FontSize',26,'TextColor','red');
    else
        ArdWrite(s,1500,1500);

    end

    % Display the annotated video frame using the video player object.
    step(videoPlayer, videoFrame);

    % Check whether the video player window has been closed.
    runLoop = isOpen(videoPlayer);



end

ArdWrite(s,1500,1500);
clear cam;
serial_close(s)

end

function y=ArdWrite(s,thr,turn)
    if thr>1640
        thr=1640;
    elseif thr<1380
        thr=1380;
    end
    if turn>2000
        turn=2000;
    elseif turn<1000
        turn=1000;
    end

    thrSTR=num2str(thr);
    turnSTR=num2str(turn);
    strout = strcat(thrSTR,',',turnSTR);
    fprintf(s,strout);
end

%set up serial
function s=serial_init
    s = serial('/dev/tty.usbmodem1421'); % Be certain to know which port you
are connected to.
    set(s,'BaudRate',9600);
    fopen(s);
    %del
    for i=1:150
```

```
        pause(.01);
    end
end

function serial_close(s)
    fclose(s);
    delete(s);
    clear s
end
```

Matlab Face Detection CODE

```
%main code
function main
s=serial_init;
thr=1500;
turn=1500;


% Create the face detector object.
faceDetector = vision.CascadeObjectDetector();

% Create the point tracker object.
pointTracker = vision.PointTracker('MaxBidirectionalError', 2);

% Create the webcam object.
cam = webcam('FaceTime HD Camera');

% Capture one frame to get its size.
videoFrame = snapshot(cam);
frameSize = size(videoFrame);

% Create the video player object.
videoPlayer = vision.VideoPlayer('Position', [100 100 [frameSize(2),
frameSize(1)]+30]);

runLoop = true;
numPts = 0;
frameCount = 0;
Xcenter=frameSize(2)/2;
Ycenter=frameSize(1)/2;

while runLoop && frameCount < 200

    % Get the next frame.
    videoFrame = snapshot(cam);
    videoFrameGray = rgb2gray(videoFrame);
    frameCount = frameCount + 1;

    if numPts < 10
        % Detection mode.
        bbox = faceDetector.step(videoFrameGray);

        if ~isempty(bbox)

            % Find corner points inside the detected region.
            points = detectMinEigenFeatures(videoFrameGray, 'ROI', bbox(1,
:));

            % Re-initialize the point tracker.
            xyPoints = points.Location;
            numPts = size(xyPoints,1);
```

```
            release(pointTracker);
            initialize(pointTracker, xyPoints, videoFrameGray);

            % Save a copy of the points.
            oldPoints = xyPoints;

            % Convert the rectangle represented as [x, y, w, h] into an
            % M-by-2 matrix of [x,y] coordinates of the four corners. This
            % is needed to be able to transform the bounding box to display
            % the orientation of the face.
            bboxPoints = bbox2points(bbox(1, :));

            % Convert the box corners into the [x1 y1 x2 y2 x3 y3 x4 y4]
            % format required by insertShape.
            bboxPolygon = reshape(bboxPoints', 1, []);

            % Display a bounding box around the detected face.
            videoFrame = insertShape(videoFrame, 'Polygon', bboxPolygon,
'LineWidth', 3);

            % Display detected corners.
            videoFrame = insertMarker(videoFrame, xyPoints, '+', 'Color',
'white');


        end

    else

        % Tracking mode.
        [xyPoints, isFound] = step(pointTracker, videoFrameGray);
        visiblePoints = xyPoints(isFound, :);
        oldInliers = oldPoints(isFound, :);

        numPts = size(visiblePoints, 1);

        if numPts >= 10
            % Estimate the geometric transformation between the old points
            % and the new points.
            [xform, oldInliers, visiblePoints] =
estimateGeometricTransform(...
                oldInliers, visiblePoints, 'similarity', 'MaxDistance', 4);

            % Apply the transformation to the bounding box.
            bboxPoints = transformPointsForward(xform, bboxPoints);

            % Convert the box corners into the [x1 y1 x2 y2 x3 y3 x4 y4]
            % format required by insertShape.
            bboxPolygon = reshape(bboxPoints', 1, []);

            % Display a bounding box around the face being tracked.
            videoFrame = insertShape(videoFrame, 'Polygon', bboxPolygon,
'LineWidth', 3);

            % Display tracked points.
```

```
            videoFrame = insertMarker(videoFrame, visiblePoints, '+',
'Color', 'white');

            % Reset the points.
            oldPoints = visiblePoints;
            setPoints(pointTracker, oldPoints);

            %find center of face in video

Xcurrent=(bboxPolygon(1)+bboxPolygon(3)+bboxPolygon(5)+bboxPolygon(7))/4;

Ycurrent=(bboxPolygon(2)+bboxPolygon(4)+bboxPolygon(6)+bboxPolygon(8))/4;

            Xdelta=Xcurrent-Xcenter;
            Ydelta=Ycurrent-Ycenter;

            thr=1500-Ydelta;
            turn=1500+Xdelta*sign(Ydelta);
            ArdWrite(s,thr,turn);


        end

    end



    % Display the annotated video frame using the video player object.
    step(videoPlayer, videoFrame);

    % Check whether the video player window has been closed.
    runLoop = isOpen(videoPlayer);
end

% Clean up.
clear cam;
release(videoPlayer);
release(pointTracker);
release(faceDetector);




%ArdWrite(s,thr,turn);


serial_close(s)

end

function y=ArdWrite(s,thr,turn)
    if thr>1640
        thr=1640;
    elseif thr<1380
```

```
        thr=1380;
    end
    if turn>2000
        turn=2000;
    elseif turn<1000
        turn=1000;
    end

    thrSTR=num2str(thr);
    turnSTR=num2str(turn);
    strout = strcat(thrSTR,',',turnSTR);
    fprintf(s,strout);
end

%set up serial
function s=serial_init
    s = serial('/dev/tty.usbmodem1421'); % Be certain to know which port you
are connected to.
    set(s,'BaudRate',9600);
    fopen(s);
    %del
    for i=1:150
        pause(.01);
    end
end

function serial_close(s)
    fclose(s);
    delete(s);
    clear s
end
```

Matlab Color Detection CODE

```
%main code
function main
cam = webcam('FaceTime HD Camera');
%s=serial_init;
thr=1500;
turn=1500;
% Assign the low and high thresholds for each color band.
        redThresholdLow = 0;
        redThresholdHigh = 120;
        greenThresholdLow = 100;
        greenThresholdHigh = 255;
        blueThresholdLow = 0;
        blueThresholdHigh = 40;


% Capture one frame to get its size.
videoFrame = snapshot(cam);
frameSize = size(videoFrame);

% Create the video player object.
videoPlayer = vision.VideoPlayer('Position', [100 100 [frameSize(2),
frameSize(1)]+30]);
runLoop = true;
numPts = 0;
frameCount = 0;
Xcenter=frameSize(2)/2;
Ycenter=frameSize(1)/2;

while runLoop && frameCount < 200
    frameCount = frameCount + 1;

    % Read in image into an array.
    [rgbImage storedColorMap] = snapshot(cam);
    [rows columns numberOfColorBands] = size(rgbImage);
    videoFrame = rgbImage;


    if strcmpi(class(rgbImage), 'uint8')
        % Flag for 256 gray levels.
        eightBit = true;
    else
        eightBit = false;
    end
    % Extract out the color bands from the original image
    % into 3 separate 2D arrays, one for each color component.
    redBand = rgbImage(:, :, 1);
    greenBand = rgbImage(:, :, 2);
    blueBand = rgbImage(:, :, 3);

    % Compute the red histogram.
    [countsR, grayLevelsR] = imhist(redBand);
    maxGLValueR = find(countsR > 0, 1, 'last');
```

```
    maxCountR = max(countsR);
    % Compute and plot the green histogram.
    [countsG, grayLevelsG] = imhist(greenBand);
    maxGLValueG = find(countsG > 0, 1, 'last');
    maxCountG = max(countsG);
    % Compute and plot the blue histogram.
    [countsB, grayLevelsB] = imhist(blueBand);
    maxGLValueB = find(countsB > 0, 1, 'last');
    maxCountB = max(countsB);


    maxGL = max([maxGLValueR,  maxGLValueG, maxGLValueB]);
    if eightBit
            maxGL = 255;
    end
    maxCount = max([maxCountR,  maxCountG, maxCountB]);

    maxGrayLevel = max([maxGLValueR, maxGLValueG, maxGLValueB]);

    if eightBit
        xlim([0 255]);
    else
        xlim([0 maxGrayLevel]);
    end

    if eightBit
            redThresholdLow = uint8(redThresholdLow * 255);
            greenThresholdHigh = uint8(greenThresholdHigh * 255);
            blueThresholdHigh = uint8(blueThresholdHigh * 255);
    end

    % Now apply each color band's particular thresholds to the color band
    redMask = (redBand >= redThresholdLow) & (redBand <= redThresholdHigh);
    greenMask = (greenBand >= greenThresholdLow) & (greenBand <=
greenThresholdHigh);
    blueMask = (blueBand >= blueThresholdLow) & (blueBand <=
blueThresholdHigh);

    % Combine the masks to find where all 3 are "true."
    % Then we will have the mask of only the red parts of the image.
    redObjectsMask = uint8(redMask & greenMask & blueMask);

    smallestAcceptableArea = 600; % Keep areas only if they're bigger than
this.

    % Get rid of small objects.  Note: bwareaopen returns a logical.
    redObjectsMask = uint8(bwareaopen(redObjectsMask,
smallestAcceptableArea));


    % Smooth the border using a morphological closing operation, imclose().
    structuringElement = strel('disk', 4);
    redObjectsMask = imclose(redObjectsMask, structuringElement);

    % Fill in any holes in the regions, since they are most likely red also.
    redObjectsMask = uint8(imfill(redObjectsMask, 'holes'));
```

```
    % You can only multiply integers if they are of the same type.
    % (redObjectsMask is a logical array.)
    % We need to convert the type of redObjectsMask to the same data type as
redBand.
    redObjectsMask = cast(redObjectsMask, class(redBand));

    % Use the red object mask to mask out the red-only portions of the rgb
image.
    maskedImageR = redObjectsMask .* redBand;
    maskedImageG = redObjectsMask .* greenBand;
    maskedImageB = redObjectsMask .* blueBand;

    % Concatenate the masked color bands to form the rgb image.
    maskedRGBImage = cat(3, maskedImageR, maskedImageG, maskedImageB);

    % Measure the mean RGB and area of all the detected blobs.

        bw = rgb2gray(maskedRGBImage);
        labeledImage = bwlabel(bw, 8);
        blobMeasurements = regionprops(labeledImage, bw, 'all');
        if size(blobMeasurements)>0
            allAreas = [blobMeasurements.Area];
            [sortedAreas, sortIndexes] = sort(allAreas, 'descend');
            if size(sortIndexes)>0
                biggestBlob = ismember(labeledImage, sortIndexes(1));
                bob=regionprops(biggestBlob, bw, 'all');
                blobarea=[bob.Area];


               % allBlobCentroids = [blobMeasurements.Centroid];
                %[len, wid]=size(allBlobCentroids);


                if blobarea>2000
                    centroids = bob.Centroid;
                    centroidsX = centroids(1);
                    centroidsY = centroids(2);
                    Xdelta=centroidsX-Xcenter;
                    Ydelta=centroidsY-Ycenter;

                    thr=1500-Ydelta;
                    turn=1500+Xdelta*sign(Ydelta);
                    %ArdWrite(s,thr,turn);
                    videoFrame = insertMarker(videoFrame, [centroidsX,
centroidsY], '+', 'Color', 'red','size',30);
                end
            end
        end

    % Display the annotated video frame using the video player object.
    step(videoPlayer, videoFrame);

    % Check whether the video player window has been closed.
    runLoop = isOpen(videoPlayer);
```

```
end


clear cam;
%serial_close(s)

end

function y=ArdWrite(s,thr,turn)
    if thr>1640
        thr=1640;
    elseif thr<1380
        thr=1380;
    end
    if turn>2000
        turn=2000;
    elseif turn<1000
        turn=1000;
    end

    thrSTR=num2str(thr);
    turnSTR=num2str(turn);
    strout = strcat(thrSTR,',',turnSTR);
    fprintf(s,strout);
end

%set up serial
function s=serial_init
    s = serial('/dev/tty.usbmodem1421'); % Be certain to know which port you
are connected to.
    set(s,'BaudRate',9600);
    fopen(s);
    %del
    for i=1:150
        pause(.01);
    end
end

function serial_close(s)
    fclose(s);
    delete(s);
    clear s
end
```

Matlab Navigation CODE

```
function main
global thetapastmove;
global robotypast;
global robotxpast;
global goalx;
global goaly;
global offset;
world=[200,200]; %size of the world
start=[70,4];%robot starting location
goal=[180,190];%goal location
goalx=goal(1);
goaly=goal(2);
robotx=start(1);
roboty=start(2);
xdis=1;
ydis=1;
robotxpast=start(1);
robotypast=start(2);
robotsize=2;
offset=2;

%list of rectangle obstacles
rect1=  [6,10,27,4;
        39,10,25,4;
        70,10,24,4;
        39,180,24,4;
        70,180,24,4;
        6,180,24,4;
        150,180,24,4;
        110,180,24,4
        110,125,24,4

        70,90,4,50;
        20,90,4,50;
        35,110,20,4;
         150,10,24,4;
         130,20,24,4;
         60,20,24,4;
         20,20,24,4;
        110,10,24,4];


%list of circle obstacles
circ1= [30,40,10,10;
        67,55,7,7;
        15,55,7,7;
        110,55,7,7;
        150,55,7,7;
        180,55,7,7;
        110,85,7,7;
        150,85,7,7;
        180,85,7,7;
        110,143,7,7;
```

```
        150,135,7,7;
        180,135,7,7;
        40,40,3,3;
        55,40,3,3;
        70,40,3,3;
        85,40,3,3;
        100,40,3,3;
        115,40,3,3;
        130,40,3,3;
        145,40,3,3;
        45,70,3,3;
        60,70,3,3;
        75,70,3,3;
        90,70,3,3;
        105,70,3,3;
        120,70,3,3;
        135,70,3,3;
        90,110,3,3;
        90,127,3,3;
        38,90,3,3;
        52,90,3,3;
        105,110,3,3;
        120,110,3,3;
        150,110,3,3;
        170,160,3,3;
        105,160,3,3;
        120,160,3,3;
        150,160,3,3;
        170,160,3,3;
        15,160,3,3;
        30,160,3,3;
        45,160,3,3;
        60,160,3,3;
        75,160,3,3
        40,130,3,3
        58,130,3,3
        90,160,3,3];


%Obstacle array
global Obs;
Obs = ones(world(1),world(2));
%add rectangles
for i=1:length(rect1(:,1))
    addrec(rect1(i,:));
end
%add cirlces
for i=1:length(circ1(:,1))
    addcirc(circ1(i,:))
end

%start navigation loop
while (abs(xdis)>0)|| (abs(ydis)>0)


    %find where to go
    theta=atan2(ydis,xdis);%direction to point
```

```
height=sin(theta);
width=cos(theta);
%set next point to clossest point to the destination
if abs(height)>=((sqrt(2)/2)-.01)
    roboty=roboty+abs(sin(theta))/sin(theta);
end
if abs(width)>=((sqrt(2)/2)-.01)
    robotx=robotx+abs(cos(theta))/cos(theta);
end

%Check if obstical is in the new location
%in real life this could be done by a laser sensor or something
if Obs(robotx,roboty)==0
    points=[0,0,2^17];
    %Check position 180 deg from current position and orientation
    [robotxtest1,robotytest1,dis1]=discalc(pi()/2);
    %set first check as best if it is open
    if Obs(robotxtest1,robotytest1)==1
        points=[robotxtest1,robotytest1,dis1];
    end
    %check postion 45 deg away
    [robotxtest2,robotytest2,dis2]=discalc(pi()/4);
    if Obs(robotxtest2,robotytest2)==1
        if dis2<points(3);
            %if its open and closer than the other point use it.
            points=[robotxtest2,robotytest2,dis2];
        end

    end
    %do the same checks for the angles from 180-(-180) including 0;
    [robotxtest3,robotytest3,dis3]=discalc(-pi()/4);
    if Obs(robotxtest3,robotytest3)==1
        if dis3<points(3);
        points=[robotxtest3,robotytest3,dis3];
        end

    end
    [robotxtest4,robotytest4,dis4]=discalc(-pi()/2);
    if Obs(robotxtest4,robotytest4)==1
        if dis4<points(3);
        points=[robotxtest4,robotytest4,dis4];
        end

    end
    [robotxtest5,robotytest5,dis5]=discalc(0);
    if Obs(robotxtest5,robotytest5)==1
        if dis5<points(3);
        points=[robotxtest5,robotytest5,dis5];
        end

    end

    %best point to chose is now points so set it to the next position
    robotx=points(1);
    roboty=points(2);
end
```

```
    %find orientation
    %find direction of past move
    pastmovex=robotx-robotxpast;
    pastmovey=roboty-robotypast;
    %find theta of past move
    thetapastmove=atan2(pastmovey,pastmovex);
    %make points for the line indicating theta
    Arrowx=[robotx+robotsize/2,robotx+robotsize/2+2*cos(thetapastmove)];
    Arrowy=[roboty+robotsize/2,roboty+robotsize/2+2*sin(thetapastmove)];
    clf %start with new figure
    figure(1)
    axis([0,world(1),0,world(2)])

    %make robot postion vecotor for plotting a circle
    pos = [robotx roboty robotsize robotsize];
    %make goal postion vecotor for plotting a circle
    goalpos=[goal(1) goal(2) robotsize robotsize];
    %plot all rectangle obstacles
    for i=1:length(rect1(:,1))
     rectangle('Position',rect1(i,:),'FaceColor',[0 0 0])
    end
    %plot all circle obstacles
    for i=1:length(circ1(:,1))
        rectangle('Position',circ1(i,:),'FaceColor',[0 0 0],'Curvature',[1
1])
    end
    %plot goal
    rectangle('Position',goalpos,'FaceColor',[0 .5 0],'Curvature',[1 1])
    %plot robot cirlce
    rectangle('Position',pos,'FaceColor',[.5 0 0],'Curvature',[1 1])
    hold on
    %plot robot orientation
    plot(Arrowx,Arrowy,'g','LineWidth',2)
    %delay so it doesnt happen instantly and plays like a movie
    pause(.1)

    %find new distance to goal this will end the loop when it gets to the
    %goal
    xdis=goal(1)-robotx;
    ydis=goal(2)-roboty;
    %set past distances
    robotxpast=robotx;
    robotypast=roboty;
end

end


%finds the distances to the goal for different points around the current
%location
function  [robotxtest1,robotytest1,dis1]= discalc(testangle)
global thetapastmove;
global robotypast;
global robotxpast;
global goalx;
global goaly;
```

```
testtheta1=thetapastmove+testangle;


        heighttest1=sin(testtheta1);
        widthtest1=cos(testtheta1);
        if abs(heighttest1)>=((sqrt(2)/2)-.01)
            robotytest1=robotypast+abs(sin(testtheta1))/sin(testtheta1);
        else
            robotytest1=robotypast;
        end
        if abs(widthtest1)>=((sqrt(2)/2)-.01)
            robotxtest1=robotxpast+abs(cos(testtheta1))/cos(testtheta1);
        else
            robotxtest1=robotxpast;
        end
        dis1=((goalx-robotxtest1)^2+(goaly-robotytest1)^2);
end
%adds a rectangle to the obstacle array
function addrec(X)
    global Obs;
    global offset;
    Obs(X(1)-offset:X(1)+X(3)+offset,X(2)-offset:X(2)+X(4)+offset)=0;
end
%adds circle to obstacle array
function addcirc(X)
    global Obs;
    global offset;
    for i=-X(3)-offset:X(3)+offset
        for j=-X(3):X(3);

            if (i^2+j^2)^.5<=(X(3)/2)+offset+3
                Obs(X(1)+i,X(2)+j)=0;
            end

        end
    end

end
```