



Faculty of Engineering  
Credit Hours Engineering Program

# RC Car Control using: Hand Gestures

## Project #03

### CSE468: Machine Vision

<b>Submitted by:</b>	
<b>Name</b>	<b>ID</b>
Omar Mohamed Bahaa	14p8067
Zain El-Abidin Khaled	14p8017
Moustafa Ahmed Abd El Mageed	14p8200
Omar Mohamed Abd El Kader	14p8128
Khaled Samir Mohamed	14p8159
<b>Submitted to:</b>	
Prof. Hossam Hassan	
Eng. Mohamed Ashraf	

Due Date: 15/11/2018

Submission Date: 14/11/2018

# Contents

Project #03.....	1
CSE468: Machine Vision.....	1
Abstract .....	3
System Constrains.....	3
Skin color detection constrains: .....	3
Hardware Constrains: .....	3
Hardware System Overview .....	5
System Components .....	5
Technical Discussion .....	6
1 <sup>st</sup> : Open Source Libraries used .....	6
2 <sup>nd</sup> : Code Algorithm overview .....	6
Code implementation .....	9
Code execution command .....	16
Results .....	17
CAD Hardware Module .....	17
Final Project Output.....	17
Conclusion .....	18
Appendix.....	18
References: .....	18

## Abstract

Today, Mobile Robots are used in many applications that helps human in doing certain tasks. And here, Computer vision played a great role for adding features to our robots, and more sensing inputs made them reach these autonomy levels that we see today, one of the struggles is hand gestures detection, that could be used by human to control the robot for doing certain tasks.

Our Project is implementing a hand gesture technique on a Raspberry pi and embed it on an RC car to be controlled by human hand gesture to let the car follow his commands.

## System Constrains

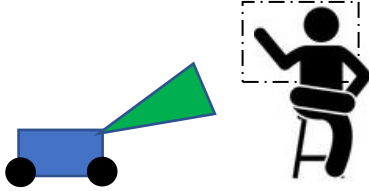
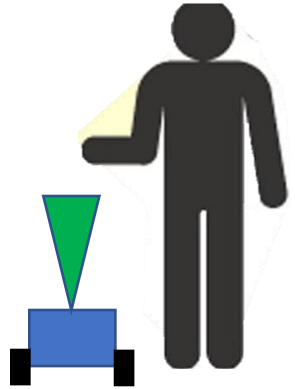
There are many techniques used to detect hand gestures, one of them is detecting skin color. But there is a lot of struggles in doing so. Also, there is some struggles related to the hardware.

### Skin color detection constrains:

- Illumination
- Background subtraction

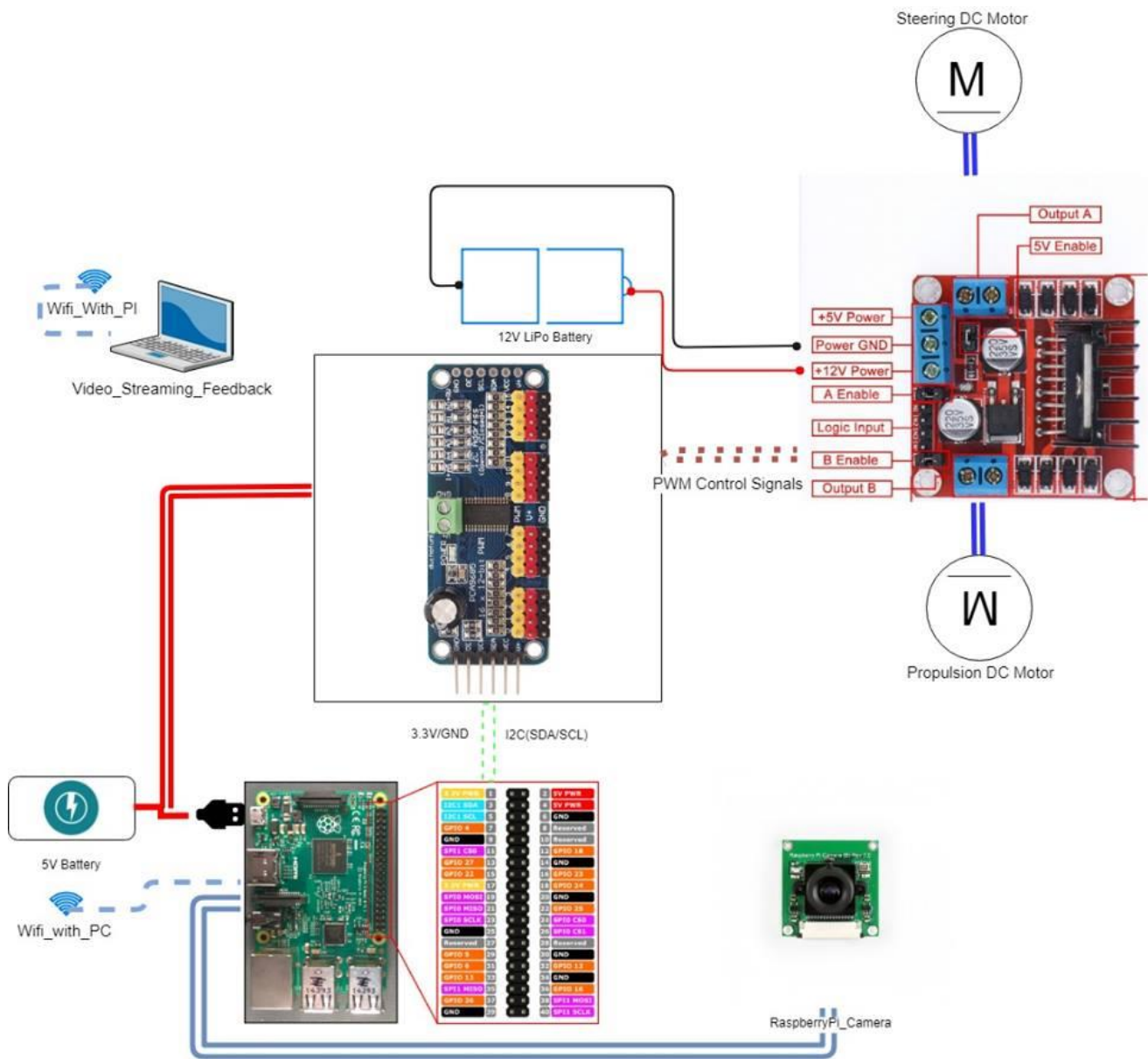
### Hardware Constrains:

- Raspberry pi Board has NO hardware PWM pins
- Our version of Raspberry pi Camera had a very narrow angle of view
- The RC Car had a very sensitive suspension system

Constrains	Problem	Explanation	Our Solution
Skin Color detection Constrains	Illumination	Depending on your environment and the illumination source, the skin color varies so much which results in unfixed RGB color ranges, which makes the setting of fixed color Intensities is impossible and not accurate at all. And not dynamic with the environment.	Our Solution is to do histogram masking depends on a sample taken in the actual testing environment each time you run the code, then filter the image with this histogram to detect the hand contour.
	Background subtraction	<p>As there are other body parts with the same histogram of the hand, so we needed to clear the scene from other body parts. So, if we let the camera to look in the forward direction it will detect or face easily. And this tried on several positions was hard mainly when the RC car tacking a steering input, so it changes its field of view and easily got confused by the different backgrounds.</p> 	<p>Our Solution was to clear the scene from other body parts by letting the camera to just look upwards to the ceil and by this we get almost clear background and we can easily control our hands in the field of view of the camera, even if the camera detects a part form the face side, the algorithm neglects it as it filters the scene and the maximum contour only is detected, which couldn't be done in other positions.</p> 
Hardware Constrains	Raspberry pi board has NO hardware PWM pins	Our application mainly requires a control action from the main controller "raspberry pi", and the actuators are 3 DC motors, so we needed a motor driver that needs a PWM input signal for speed control.	Our Solution: connecting the pi with a (16-channel, 12bit resolution) PWM module on I2C bus. And This even gave us the flexibility to use these channels instead of using the pi's GPIO pins by giving a full duty cycle or zero to the channel, so it acts as a high or low digital output pins for direction control.
	Narrow Camera cone "angle of view" and the sensitive suspension.	Our hand's comfort position is very large with respect to the camera frame, as the angle of view is very narrow, so the centroid of the detected contour doesn't change very much. So undesired input and false calls happens. And, With the Sensitive suspension this narrow angle keeps tilting and our hand sometimes got absent from the scene giving false calling and undesired input.	This was our largest problem, which we solved by small percentage in our tests. But still we need a new camera with larger angle of view. while, the suspension system of the car was controlled by mechanically limit the stroke length to minimum.

# Hardware System Overview

- After selecting all the hardware components, this is our system schematic.



## System Components

- Raspberry Pi Model 3
- Pi Camera
- 16 Channel, 12 Bit PWM Module
- L298 Dual H-Bridge Motor Driver
- 12V LiPo Battery
- 5V Battery
- RC Car

# Technical Discussion

## 1<sup>st</sup>: Open Source Libraries used


OpenCV → open source computer vision functions

Adafruit\_PCA9685 → (16-channel, 12bit-resolution) PWM module adafruit libraries

Picamera → For setting Camera parameters

## 2<sup>nd</sup>: Code Algorithm overview

### Step A: Drawing the sampling boxes

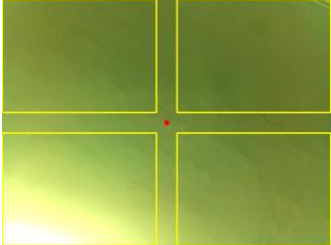
Inputs	➤ <b>Single Frame → image</b>
Processing	<b>1) image = draw_rect(image)</b> → this function takes the image and draw 9 small rectangles of size 10x10 each and save the pixel values inside some global variables. →Note: Samples are taken, not the whole hand pixel values to fasten the process.
Outputs	➤ frame with drawn 9 small rectangles in it and waiting the user input to press "x" to capture the hand's skin histogram. 

**Step B:** Taking the Histogram Samples from the hand skin color and do the histogram masking.

<b>Inputs</b>	<ul style="list-style-type: none"> <li>➤ <b>Single Frame</b> → <b>image</b></li> <li>➤ <b>(Wait key) == X</b></li> <li>➤ <b>Intensity Samples</b></li> </ul>
<b>Processing</b>	<p><b>1) hand_hist = hand_histogram(image)</b>  → Here function transforms the input frame to HSV. Using numpy arrays, we create an image of size <b>[90 * 10]</b> with <b>3</b> color channels, It then takes the 900-pixel values from the sample rectangles and puts them in the matrix. Then calc the histogram, and finally normalize them.</p>
<b>Output</b>	<ul style="list-style-type: none"> <li>➤ <b>Global Variable</b> → <b>hand_hist</b></li> </ul>

**Step C:**

<b>Inputs</b>	<ul style="list-style-type: none"> <li>➤ <b>Single Frame</b> → <b>image</b></li> <li>➤ <b>Global Variable</b> → <b>hand_hist</b></li> </ul>
<b>Processing</b>	<p><b>1) control(image, hand_hist)</b>  → this is the main function where the following functions are being called from it...</p> <p><b>2) hist_mask_image = hist_masking(frame, hand_hist)</b>  → function that changed the input frame to HSV and then applied <b>cv2.calcBackProject</b> with the skin color histogram <b>hist</b>. Following that, I have used Filtering and Thresholding function to smoothen the image. Lastly, I masked the input frame using the <b>cv2.bitwise_and</b> function. This final frame should just contain skin color regions of the frame.</p> <p><b>3) contour_list = contours(hist_mask_image)</b>  → here we convert to gray scale, then we take the threshold, then find the contours.</p> <p><b>4) max_cont = max_contour(contour_list)</b>  → determines the maximum contour in the image.</p>

Processing	<p><b>5) insideQuad(quad_F, xy_centroid)</b>  → determines whether the centroid is inside the determined quad or not, if so, it returns true, otherwise false...  → Note: this function is implemented to calculate the angle between two vectors represents the vector between each two successive vertices and the centroid, then calculate the angle between them finally add up all the angles "total 4", if the sum is equal to 360 then the point inside the quad, if the sum is zero then the point is outside the rectangle.</p> <p><b>6) d_angle(x1, y1, x2, y2)</b>  → calculates the angle between two vectors that calculated by the previous function.</p> <p><b>7) dir_Control(state)</b>  → this is actually the function that determines how the Robot moves, it receives the quad that it works in, then it gives the PWM values needed to each channel of the PWM module. Using the function:  pwm.set_pwm()  → Note: some pwm.set_pwm() functions used with full duty cycle input value, so used as a digital output pins instead of using the actual pi's GPIO pins, so keeping the pi safer, as the I2C bus is high impedance bus that the module is connected on it.</p>
Outputs	<p>➤ Frame with 4 large quads and in between other 3 smaller quads determines our control field, with a default red dot determines the idle brake position of the RC-Car.</p>  <p>➤ If the scene contains a hand "or any object with close histogram : 'D ' it updates the centroid position and keeps tracking it.</p>



## Step D: main Loop

Idle state is "**Step A**" → till the used inputs(X) to capture the histogram of his hand "**Step B**" → Repeating → **step C**, in a loop → Video Streaming using Raspberry pi camera → till user inputs a close command "ESC" to cancel the capturing process to exit the loop, then → close the program.

➤ Continuous capturing of frames...

→ **Condition**

**Step: A**

→ **Condition**

**Step: B**

→ **Condition**

**Step: C**

if pressed\_key == 27:

break

## Code implementation

```
from __future__ import division
import cv2
import numpy as np
from picamera.array import PiRGBArray
from picamera import PiCamera
import time
import math
import sys
#import RPi.GPIO as GPIO
import Adafruit_PCA9685
```

```
#####
```

```
#GPIO initialization
```

```
#GPIO.setmode(GPIO.BCM)
```

```
#GPIO.setup(17, GPIO.OUT)
```

```
#GPIO_17: is the main motors pin
```

```
#GPIO.setup(27, GPIO.OUT)
```

```
#GPIO_27: is the steering direction pin
```

```
# Initialise the PCA9685 using the default address (0x40).
```

```
pwm = Adafruit_PCA9685.PCA9685()
```

```

# Set frequency to 60hz
pwm.set_pwm_freq(60)

#####
hand_hist = None
total_rectangle = 9
hand_rect_one_x = None
hand_rect_one_y = None

hand_rect_two_x = None
hand_rect_two_y = None

#####
#Define Vertices
A = [320, 240]
B = [320, 0]
C = [320, 480-1]
D = [640-1, 240]
E = [0, 240]
F = [640-1, 0]
G = [0, 0]
H = [0, 480-1]
I = [640-1, 480-1]

#Define Quads for control
quad_F = np.array([ [B[0]-20, B[1]],
                    [B[0]+20, B[1]],
                    [A[0]+20, A[1]-20],
                    [A[0]-20, A[1]-20] ], np.int32)
#quad_F = quad_F.reshape((6))

quad_B = np.array([ [A[0]-20, A[1]+20],
                    [A[0]+20, A[1]+20],
                    [C[0]+20, C[1]],
                    [C[0]-20, C[1]] ], np.int32)
#quad_B = quad_B.reshape((6))

quad_1 = np.array([ [B[0]+20, B[1]],
                    F,
                    [D[0], D[1]-20],
                    [A[0]+20, A[1]-20] ], np.int32)
#quad_1 = quad_1.reshape((6))

quad_2 = np.array([ G,
                    [B[0]-20, B[1]],
                    [A[0]-20, A[1]-20],
                    [E[0], E[1]-20] ], np.int32)
#quad_2 = quad_2.reshape((6))

quad_3 = np.array([ [E[0], E[1]+20],
                    [A[0]-20, A[1]+20],
                    [C[0]-20, C[1]],
                    H ], np.int32)
#quad_3 = quad_3.reshape((6))

quad_4 = np.array([ [A[0]+20, A[1]+20],
                    [D[0], D[1]+20],
                    I,
                    [C[0]+20, C[1]] ], np.int32)
#quad_4 = quad_4.reshape((6))

```

```

quad_Idle = np.array([ [E[0], E[1]-20],
                      [D[0], D[1]-20],
                      [D[0], D[1]+20],
                      [E[0], E[1]+20] ], np.int32)
#quad_Idle = quad_Idle.reshape((6))

#####
#functions:

#def rescale_frame(frame, wpercent=100, hpercent=100):
#    width = int(frame.shape[1] * wpercent / 100)
#    height = int(frame.shape[0] * hpercent / 100)
#    return cv2.resize(frame, (width, height), interpolation=cv2.INTER_AREA)

def contours(hist_mask_image):
    gray_hist_mask_image = cv2.cvtColor(hist_mask_image, cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(gray_hist_mask_image, 0, 255, 0)
    _, cont, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    return cont

def max_contour(contour_list):
    max_i = 0
    max_area = 0

    for i in range(len(contour_list)):
        cnt = contour_list[i]

        area_cnt = cv2.contourArea(cnt)

        if area_cnt > max_area:
            max_area = area_cnt
            max_i = i

    return contour_list[max_i]

def draw_rect(frame):
    rows, cols, _ = frame.shape
    global total_rectangle, hand_rect_one_x, hand_rect_one_y, hand_rect_two_x, hand_rect_two_y

    hand_rect_one_x = np.array(
        [6 * rows / 20, 6 * rows / 20, 6 * rows / 20, 9 * rows / 20, 9 * rows / 20, 9 * rows / 20, 12 *
rows / 20,
        12 * rows / 20, 12 * rows / 20], dtype=np.uint32)

    hand_rect_one_y = np.array(
        [9 * cols / 20, 10 * cols / 20, 11 * cols / 20, 9 * cols / 20, 10 * cols / 20, 11 * cols / 20,
9 * cols / 20,
        10 * cols / 20, 11 * cols / 20], dtype=np.uint32)

    hand_rect_two_x = hand_rect_one_x + 10
    hand_rect_two_y = hand_rect_one_y + 10

    for i in range(total_rectangle):
        cv2.rectangle(frame, (hand_rect_one_y[i], hand_rect_one_x[i]),
                      (hand_rect_two_y[i], hand_rect_two_x[i]),
                      (0, 255, 255), 1)

```

```

return frame

def hand_histogram(frame):
    global hand_rect_one_x, hand_rect_one_y

    hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    roi = np.zeros([90, 10, 3], dtype=hsv_frame.dtype)

    for i in range(total_rectangle):
        roi[i * 10: i * 10 + 10, 0: 10] = hsv_frame[hand_rect_one_x[i]:hand_rect_one_x[i] + 10,
            hand_rect_one_y[i]:hand_rect_one_y[i] + 10]

    hand_hist = cv2.calcHist([roi], [0, 1], None, [180, 256], [0, 180, 0, 256])
    return cv2.normalize(hand_hist, hand_hist, 0, 255, cv2.NORM_MINMAX)

def hist_masking(frame, hist):
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    dst = cv2.calcBackProject([hsv], [0, 1], hist, [0, 180, 0, 256], 1)

    disc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (31, 31))
    cv2.filter2D(dst, -1, disc, dst)

    ret, thresh = cv2.threshold(dst, 150, 255, cv2.THRESH_BINARY)

    # thresh = cv2.dilate(thresh, None, iterations=5)

    thresh = cv2.merge((thresh, thresh, thresh))

    return cv2.bitwise_and(frame, thresh)

def centroid(max_contour):
    moment = cv2.moments(max_contour)
    if moment['m00'] != 0:
        cx = int(moment['m10'] / moment['m00'])
        cy = int(moment['m01'] / moment['m00'])
        return cx, cy
    else:
        cx=320          #image centre x
        cy=240          #image centre y
        return cx, cy

#####
#Motor Control Pins diagram using PWM channels (0-5) from 16 channel PWM module
EN0 = 0                # channel_0 -> Driving motors -> Note: (0 <> 2^12) PWM range for speed control
EN1 = 5                # channel_1 -> Steering Motor -> Note: (0 <> 2^12) but used only 50%
#
#Direction Control pins on the motor driver, Note: always max PWM 2^12, thus used as digital pins
N1 = 1                # channel_2 -> N1
N2 = 2                # channel_3 -> N2
N3 = 3                # channel_4 -> N3
N4 = 4                # channel_5 -> N4
#
max_pulseW = 4095      #100% duty cycle
min_pulseW = 0         # 0% duty cycle
pwm_redu = 500
#
#####

```

```

def dir_Control(state):                                #RC-Car direction control

    if state == 'F':                                  #forward
        #GPIO.output(17, GPIO.HIGH)
        pwm.set_pwm(EN0, 0, max_pulsew-pwm_redu) #50% duty cycle
        pwm.set_pwm(N1, 0, max_pulsew)           #N1: HIGH
        pwm.set_pwm(N2, 0, min_pulsew)           #N2: LOW

        pwm.set_pwm(EN1, 0, min_pulsew)          # 0% duty cycle
        pwm.set_pwm(N3, 0, min_pulsew)           #N1: LOW
        pwm.set_pwm(N4, 0, min_pulsew)           #N1: LOW

    elif state == 'B':                                  #backward
        #GPIO.output(17, GPIO.LOW)
        pwm.set_pwm(EN0, 0, max_pulsew-pwm_redu) #50% duty cycle
        pwm.set_pwm(N1, 0, min_pulsew)           #N1: LOW
        pwm.set_pwm(N2, 0, max_pulsew)           #N2: HIGH

        pwm.set_pwm(EN1, 0, min_pulsew)          # 0% duty cycle
        pwm.set_pwm(N3, 0, min_pulsew)           #N1: LOW
        pwm.set_pwm(N4, 0, min_pulsew)           #N1: LOW

    elif state == '1':                                  #forward and right
        #GPIO.output(17, GPIO.HIGH)
        pwm.set_pwm(EN0, 0, max_pulsew-pwm_redu) #50% duty cycle
        pwm.set_pwm(N1, 0, max_pulsew)           #N1: HIGH
        pwm.set_pwm(N2, 0, min_pulsew)           #N2: LOW

        pwm.set_pwm(EN1, 0, max_pulsew-pwm_redu) #50% duty cycle
        pwm.set_pwm(N3, 0, max_pulsew)           #N1: HIGH
        pwm.set_pwm(N4, 0, min_pulsew)           #N1: LOW

    elif state == '2':                                  #forward and left
        #GPIO.output(17, GPIO.HIGH)
        pwm.set_pwm(EN0, 0, max_pulsew-pwm_redu) #50% duty cycle
        pwm.set_pwm(N1, 0, max_pulsew)           #N1: HIGH
        pwm.set_pwm(N2, 0, min_pulsew)           #N2: LOW

        pwm.set_pwm(EN1, 0, max_pulsew-pwm_redu) #50% duty cycle
        pwm.set_pwm(N3, 0, min_pulsew)           #N1: LOW
        pwm.set_pwm(N4, 0, max_pulsew)           #N1: HIGH

    elif state == '3':                                  #backward and left
        #GPIO.output(17, GPIO.LOW)
        pwm.set_pwm(EN0, 0, max_pulsew-pwm_redu) #50% duty cycle
        pwm.set_pwm(N1, 0, min_pulsew)           #N1: LOW
        pwm.set_pwm(N2, 0, max_pulsew)           #N2: HIGH

        pwm.set_pwm(EN1, 0, max_pulsew-pwm_redu) #50% duty cycle
        pwm.set_pwm(N3, 0, min_pulsew)           #N1: LOW
        pwm.set_pwm(N4, 0, max_pulsew)           #N1: HIGH

    elif state == '4':                                  #backward and right
        #GPIO.output(17, GPIO.LOW)
        pwm.set_pwm(EN0, 0, max_pulsew-pwm_redu) #50% duty cycle
        pwm.set_pwm(N1, 0, min_pulsew)           #N1: LOW
        pwm.set_pwm(N2, 0, max_pulsew)           #N2: HIGH

        pwm.set_pwm(EN1, 0, max_pulsew-pwm_redu) #50% duty cycle
        pwm.set_pwm(N3, 0, max_pulsew)           #N1: HIGH

```

```

        pwm.set_pwm(N4, 0, min_pulsew)          #N1: LOW

elif state == 'X':                             #Idle Zone
    pwm.set_pwm(EN0, 0, min_pulsew)            # 0% duty cycle
    pwm.set_pwm(N1, 0, min_pulsew)            #N1: LOW
    pwm.set_pwm(N2, 0, min_pulsew)            #N2: LOW

    pwm.set_pwm(EN1, 0, min_pulsew)            # 0% duty cycle
    pwm.set_pwm(N3, 0, min_pulsew)            #N1: LOW
    pwm.set_pwm(N4, 0, min_pulsew)            #N1: LOW

else:                                           #Default Brake Zone
    pwm.set_pwm(EN0, 0, min_pulsew)            # 0% duty cycle
    pwm.set_pwm(N1, 0, min_pulsew)            #N1: LOW
    pwm.set_pwm(N2, 0, min_pulsew)            #N2: LOW

    pwm.set_pwm(EN1, 0, min_pulsew)            # 0% duty cycle
    pwm.set_pwm(N3, 0, min_pulsew)            #N1: LOW
    pwm.set_pwm(N4, 0, min_pulsew)            #N1: LOW

#####

def d_angle(x1, y1, x2, y2):

    theta_1 = np.arctan2(y1, x1)
    theta_2 = np.arctan2(y2, x2)

    d_theta = np.subtract(theta_1, theta_2)

    while (d_theta > np.pi):
        d_theta = d_theta - 2*np.pi
    while (d_theta < -np.pi):
        d_theta = d_theta + 2*np.pi

    return d_theta

def insideQuad(quad, target):

    angle = 0.0
    p1 = [0,0]
    p2 = [0,0]

    for i in range(4):
        for j in range(2):
            p1[j] = quad[i,j] - target[j]
            p2[j] = quad[(i+1)%3,j] - target[j]

        angle = angle + d_angle(p1[0],p1[1],p2[0],p2[1])

    if (abs(angle) < np.pi):
        return False
    else:
        return True

def control(frame, hand_hist):
    hist_mask_image = hist_masking(frame, hand_hist)

```

```

contour_list = contours(hist_mask_image)
max_cont = max_contour(contour_list)

xy_centroid = centroid(max_cont)
cv2.circle(frame, xy_centroid, 5, [0, 0, 255], -1)
#print "(x,y)= ", xy_centroid

#direction control
if insideQuad(quad_F, xy_centroid) == True:
    dir_Control('F')
    print "Forward"

elif insideQuad(quad_B, xy_centroid) == True:
    dir_Control('B')
    print "Backward"

elif insideQuad(quad_1, xy_centroid) == True:
    dir_Control('1')
    print "1st Quad"

elif insideQuad(quad_2, xy_centroid) == True:
    dir_Control('2')
    print "2nd Quad"

elif insideQuad(quad_3, xy_centroid) == True:
    dir_Control('3')
    print "3rd Quad"

elif insideQuad(quad_4, xy_centroid) == True:
    dir_Control('4')
    print "4th Quad"

elif insideQuad(quad_Idle, xy_centroid) == True:
    dir_Control('X')
    print "Idle Region"

else:
    dir_Control('S')
    print "Stop"

```

```
#####
```

```

def main():

    # initialize the camera and grab a reference to the raw camera capture
    camera = PiCamera()
    camera.resolution = (640, 480)
    camera.framerate = 32
    camera.rotation = 180
    rawCapture = PiRGBArray(camera, size=(640, 480))

    #Global Variables

    global hand_hist
    global quad_1
    quad_1_draw = quad_1.reshape((-1,1,2))

    global quad_2
    quad_2_draw = quad_2.reshape((-1,1,2))

```

```

global quad_3
quad_3_draw = quad_3.reshape((-1,1,2))

global quad_4
quad_4_draw = quad_4.reshape((-1,1,2))

is_hand_hist_created = False

#Main loop

for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):

    pressed_key = cv2.waitKey(1)
    image = frame.array

    if pressed_key & 0xFF == ord('x'):
        is_hand_hist_created = True
        hand_hist = hand_histogram(image)

    if is_hand_hist_created:
        control(image, hand_hist)
        #Quads Drawing
        cv2.polylines(image,[quad_1_draw],True,(0,255,255), 2)
        cv2.polylines(image,[quad_2_draw],True,(0,255,255), 2)
        cv2.polylines(image,[quad_3_draw],True,(0,255,255), 2)
        cv2.polylines(image,[quad_4_draw],True,(0,255,255), 2)

    else:
        image = draw_rect(image)

    #image = cv2.flip(image, -1)
    cv2.imshow("Car Controller", image)

    # clear the stream in preparation for the next frame
    rawCapture.truncate(0)

    if pressed_key == 27:
        break

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print 'Interrupted'
    #finally:
        #GPIO.cleanup()

```

## Code execution command

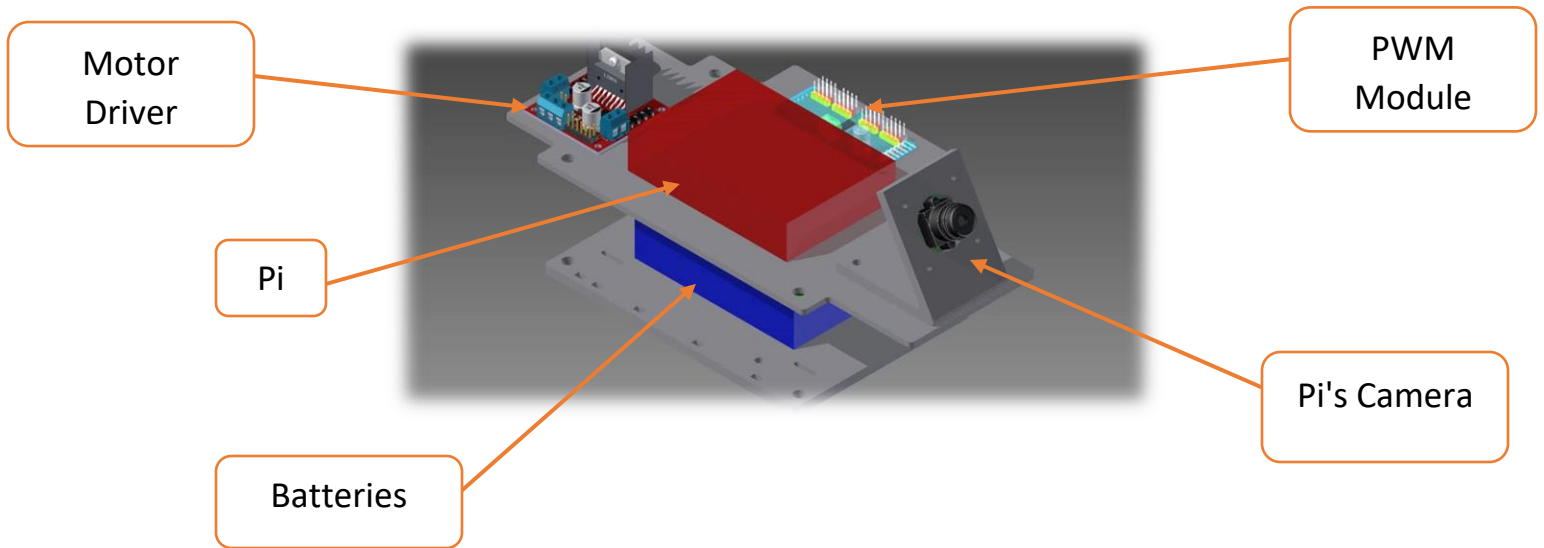
>> python CSE489\_Project\_3.py



# Results

## CAD Hardware Module

➤ This Module is attached to the RC car, holding all the electronics used.



## Final Project Output



## Conclusion

The Project needs some optimization in the code algorithm, due to the long processing time, this results in accuracy near 70% of our desired output which we could increase up to 92% after this optimizations, and this could be done by optimizing the Centroid tracking technique. Also, The camera we used has a very narrow angle of view, and this limits our project's desired output, so its recommended to try another wide angle camera.

## Appendix

### References:

>> [https://learn.adafruit.com/adafruit-16-channel-servo-driver-with-raspberry-pi/hooking-it-up?fbclid=IwAR2\\_nDJPG5PvKZnEwYcxaXUJwGf06BaMI7vc7fQvG9bYqBDBnnAjmnSEhQ](https://learn.adafruit.com/adafruit-16-channel-servo-driver-with-raspberry-pi/hooking-it-up?fbclid=IwAR2_nDJPG5PvKZnEwYcxaXUJwGf06BaMI7vc7fQvG9bYqBDBnnAjmnSEhQ)

>> <https://github.com/amarlearning/Finger-Detection-and-Tracking>