Mechatronics Engineering and Automation Program

CSE468: Machine Vision

Project #03: Self Driving RC Car

SelfDrivingRCCar

Abstract

This Project aims to equip standard remote control (RC) car with improved control since commonly used (RC) car control joystick is replaced with certain controlling system based on camera.

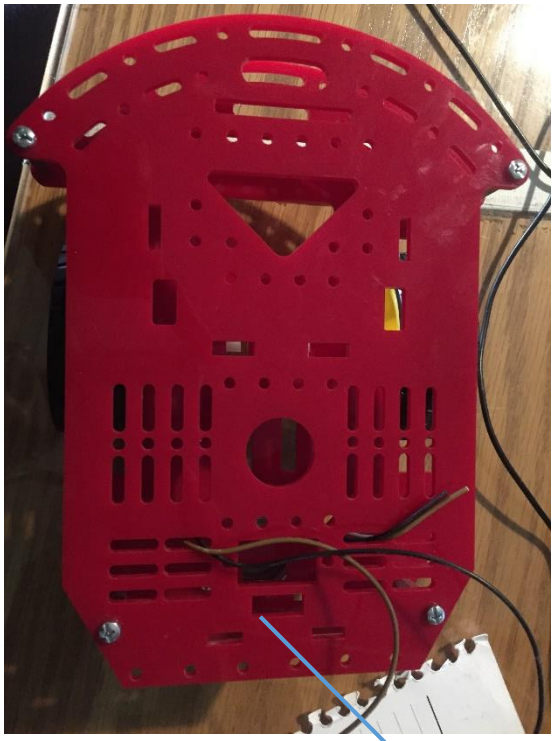
This project also aims to learn how to integrate between mechanical, electronics and vision so that we are able to integrate what we have learned in our machine vision course with our core as a mechatronics engineer but in this project we focused on how to control our car by vision as this our main target.

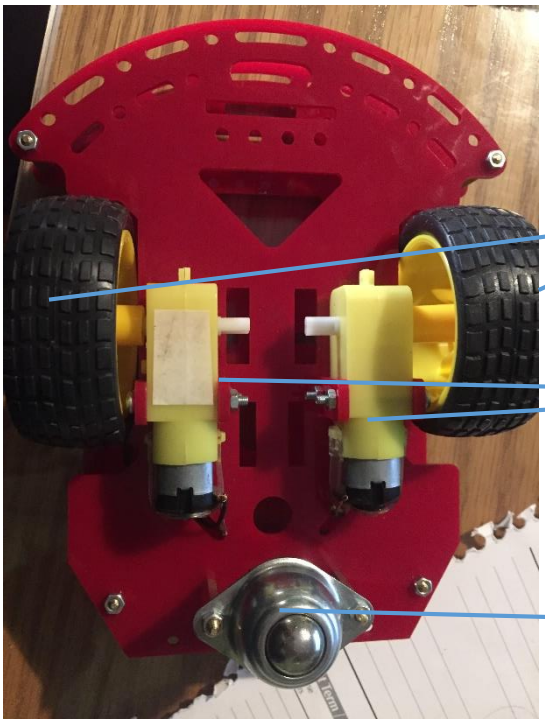
The camera that is attached to car enables view from perspective of (RC) car. (RC) car can be used in many applications such as scouting.

Our car is controlled by getting arrows direction as input from our camera.

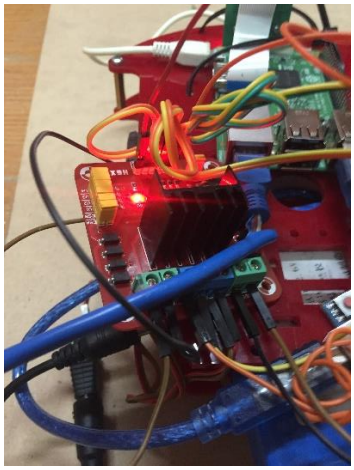
Technical Discussion:

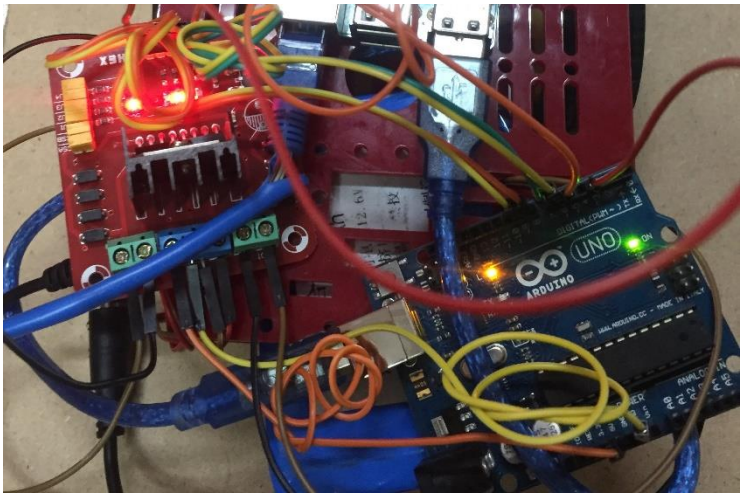
- First of all, we will start by the mechanical part of the project. We have to construct our car, so we need a body for our car and we used two front wheels and one castor back wheel.
  So we need two motors for the front two wheels as shown in the next figures.

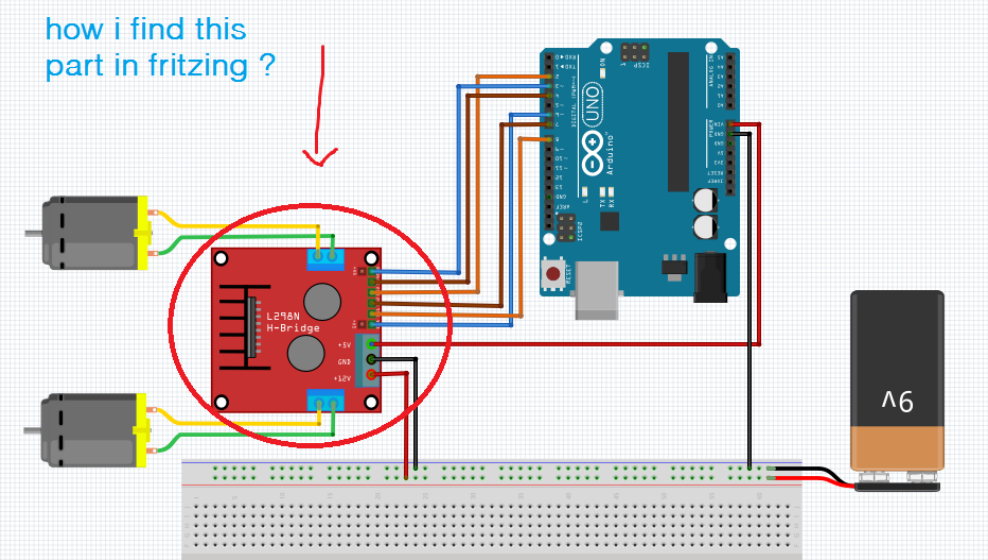




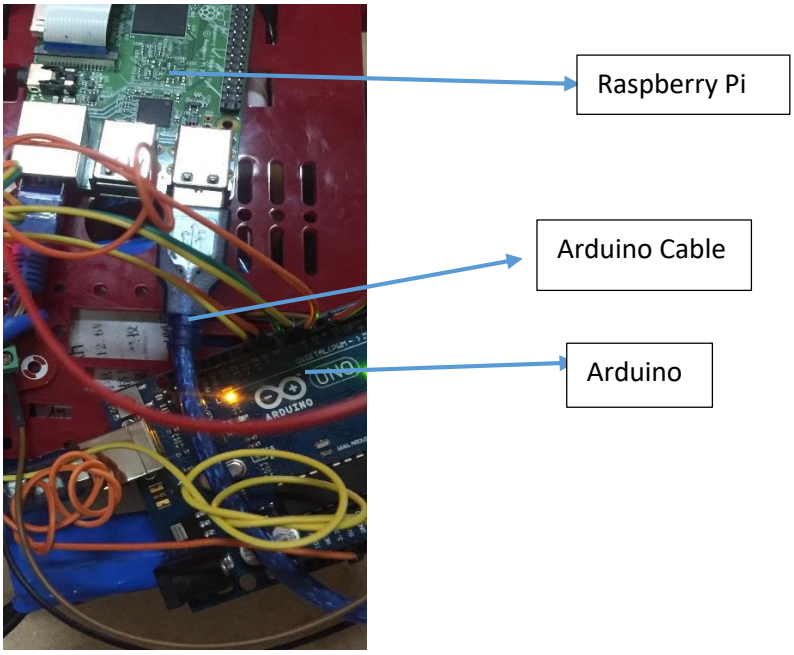
- Secondly, we will g[illegible]ctronics part. We need Raspberry Pi microcontroller to control the Raspbe[illegible]a so we also need a pi camera. We also need an Arduino microcontroller to control the motors and an L298 Dual H-Bridge Motor Driver to control the two motors speed and direction. We also need power bank to give 5V for the Raspberry Pi so we are able to make it work without PC. We also need a battery to make the Arduino able to work also without a connection with the PC. We also need net cable to make the Raspberry Pi connected to the net.
    - We need one driver for this two motors and this driver is (L298 Dual H-Bridge Motor Driver). This Motor Driver module board, using ST's L298N can directly drive two 3-30V DC motor. You can easily control DC motor speed and direction. And the next figures will show you the L298N dual H-bridge Motor Driver and also the connections of this driver with the two motors and the arduino



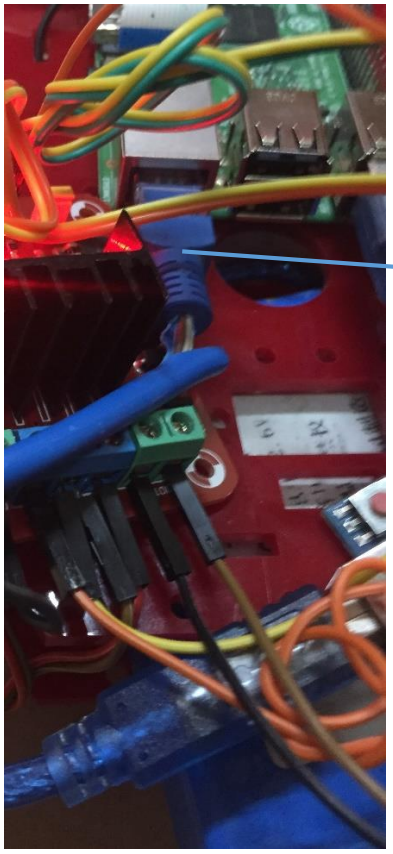




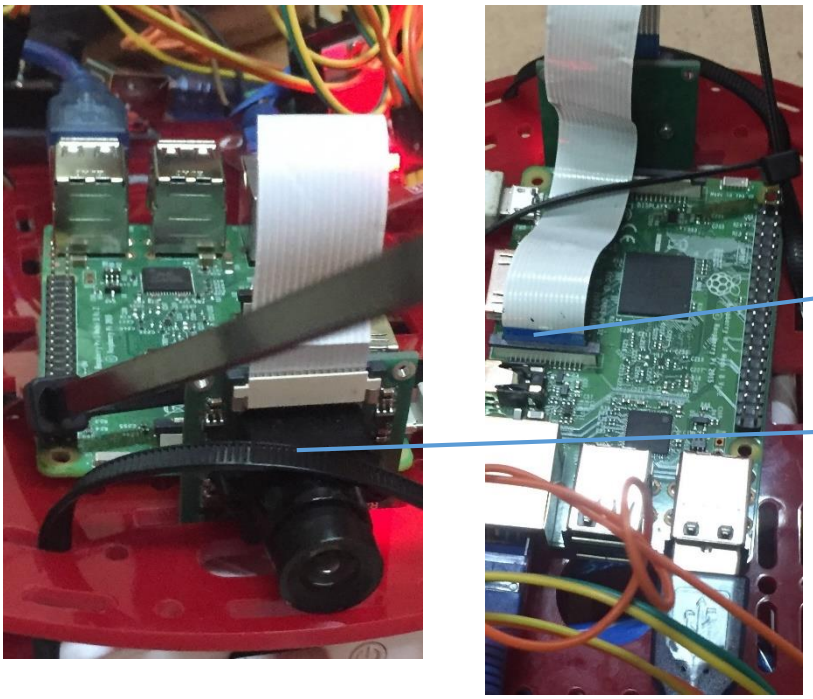
- Then, we connected the Arduino to the raspberry Pi by the Arduino cable as shown in the next figure.



- Next, the raspberry Pi must be connected to the net through wireless by configuring this in the vns command or by connecting the raspberry Pi to router using net cable as shown in the next figure.



- Next, Connect the camera to the raspberry Pi camera to the raspberry Pi microcontroller as shown in the next figures.



- Then we connected the raspberry Pi to the power bank to get power (5V) from it.

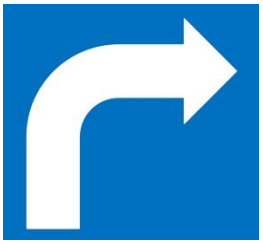
Power Bank Cable

Summary of Connections:

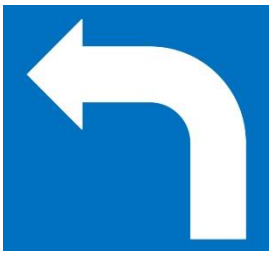
- Arduino Connected to the Dual h-bridge motor driver which is connected to the two motors to control its speed and direction as discussed above which gives signal to the raspberry Pi by the Arduino cable connecting the Arduino to the raspberry Pi.
- The raspberry Pi Camera is connected to the raspberry Pi as shown above and the raspberry Pi is working by Power bank.

- Third, we will start talking about software part of the project.
  - First of all, we have to download the vnc viewer and enter with the ip address of your raspberry Pi from puty and set up your raspberry Pi using “ping -4 raspberry Pi.local”) then start writing your code.
  - Then you have to make install to the Arduino , the raspberry Pi camera is setted with the raspberry Pi by the command “sudo raspi-config” and open cv 3.4 inside the command window of the vnc.
  - Install Open cv 3.4 from (https://www.pyimagesearch.com/2017/09/04/raspbian-stretch-install-opencv-3-python-on-your-raspberry-pi/)
  - Also you have to install nanpy library of the Arduino to set it with the raspberry Pi, pip 3 and imutils.
  - We used Sign recognition system to be our way to control our RC car as when the direction of the arrow is to the front the car will move forward, when the direction of the arrow is to the back the car will move backward, when the direction of the arrow is to the right the car will turn right and when the direction of the arrow is to the left the car will move to the left.

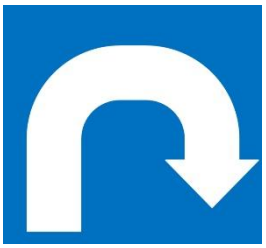
Move Forward



Turn Right



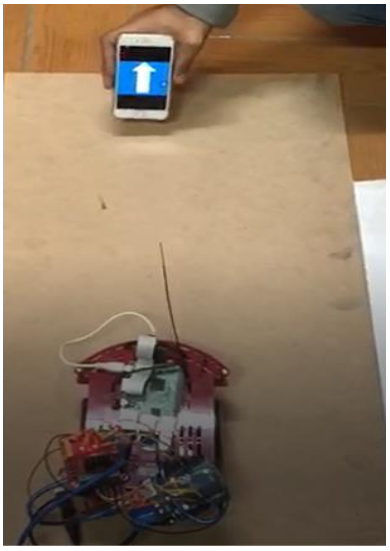
Turn Left



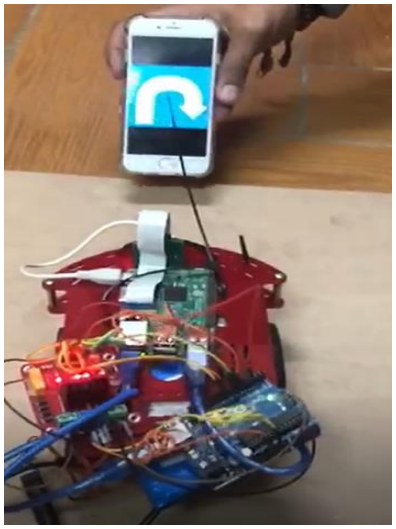
Move Backward

Discussion of Results

- First we started with the assembly of the car and the components as discussed in the technical discussion above and the results that the centre of gravity of the car was not very accurate as the weight was a little bit high on the body of the car which leads to a small deflection when the car moves straight or turns back but to control this we have to make PID control on the two motors but this is too hard and need much more time to make it in an accurate way.
- For the electronics part it was very easy to control the speed and direction of the two motors using the dual h-bridge motor driver and to connect the Arduino to raspberry pi.
  In the beginning, we had a problem as the raspberry Pi couldn't be used to control the two motors because of the pwm. First I thought it would be difficult to take a signal from the Arduino to raspberry Pi but after searching how to do this we found that it is easy by connecting the Arduino cable to the raspberry Pi and also we have to install the nanpy of the Arduino in the command window of the vnc.
  The camera was easy connected to the hdmi port in raspberry Pi and we didn't have any problem with connecting the camera to the raspberry Pi.
- We had some problems in installing some libraries in the beginning and we had a lot of errors because of downloading wrong libraries and writing wrong commands but at the end we were able to get the right commands and libraries which we need in our code to make it run successfully.
- The results of the sign recognition was really good but the performance would have been better if we did PID control but still it was very good results and did what is required.

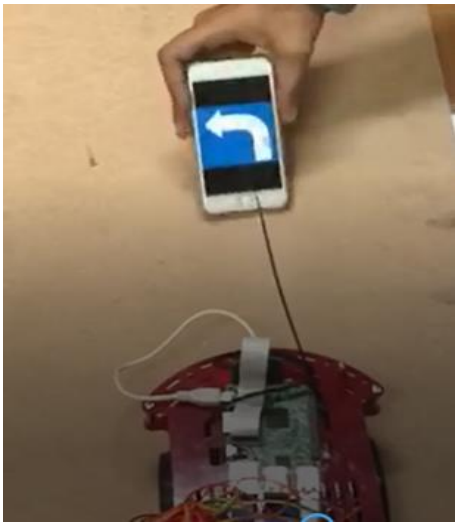


Move Forward, we used this sign to make our car moves forward.



Move Backward, we used this sign to make our car moves backward

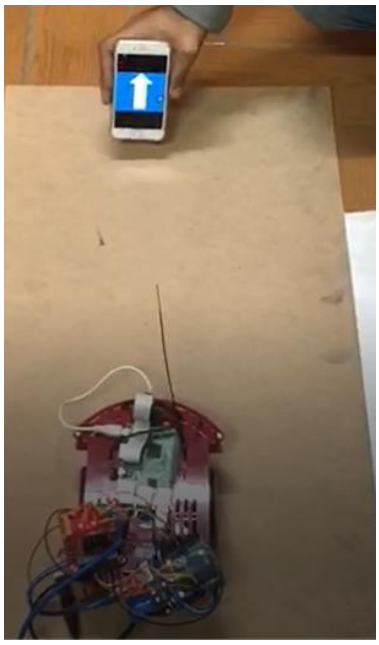
Turn Right, we used this sign to make our car moves to the right.

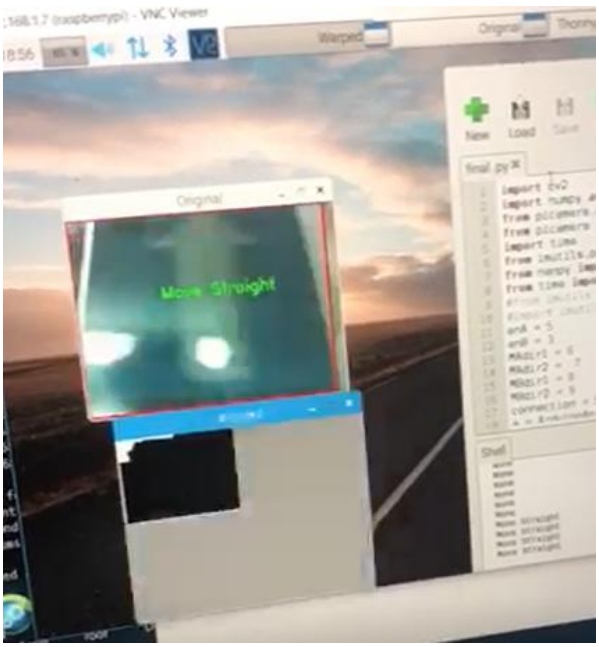


Turn Left, we used this sign to make our car moves to the left.

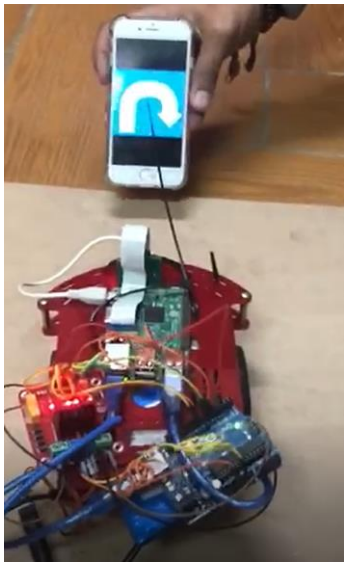
y good as we were able to do the function of the RC car which is to imple based camera control and to integrate between mechanical, so that we are able to integrate what we have learned in our machine vision course with our core as a mechatronics engineer but in this project we focused on how to control our car by vision as this our main target.

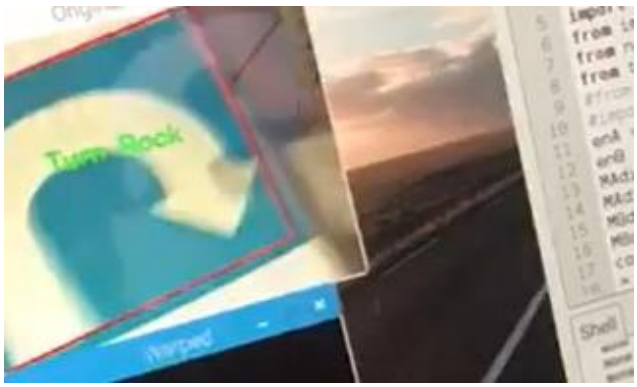
- This is the result of camera recognizing this sign which is moving straight “Move Straight” as written in the green colour in the right figure below.



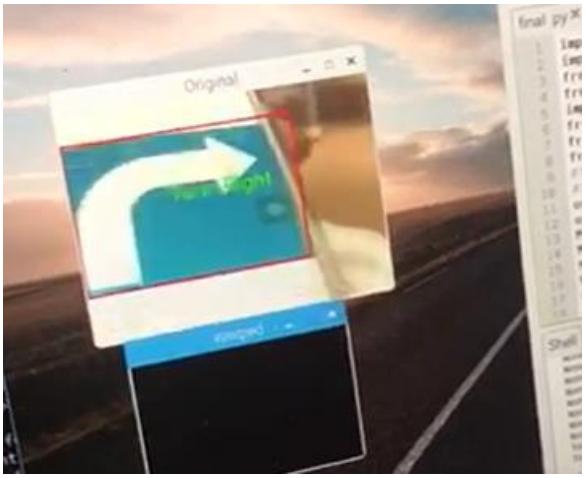


- This is the result of the camera recognizing this sign which is moving backward “Turn Back” as written in the green colour in the right figure below.

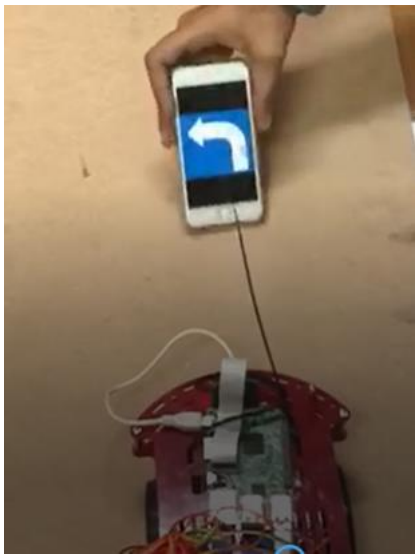


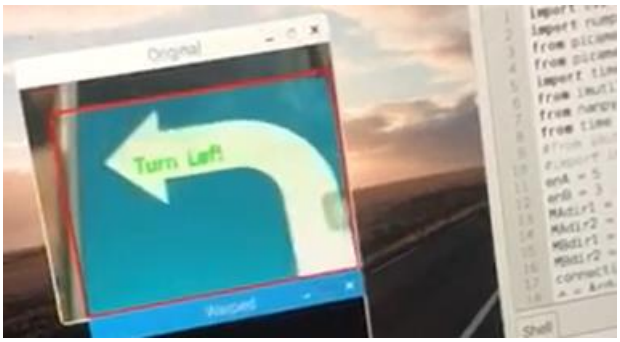


- This is the result of the camera recognizing this sign which is turning right “Turn Right” as written in the green colour in the right figure below.



- This is the result of the camera recognizing this sign which is turning left “Turn Left” as written in the green colour in the right figure below.





Components

1. Car Kit
2. Raspberry Pi
3. Raspberry Pi Camera
4. Arduino
5. L298 Dual H-Bridge Motor Driver
6. Power Bank
7. Ethernet Cable
8. Jumpers (wires)

Appendix

```
import cv2

import numpy as np

from picamera.array import PiRGBArray

from picamera import PiCamera

import time

from imutils.perspective import four_point_transform

from nanpy import (ArduinoApi, SerialManager)

from time import sleep

#from imutils import contours

#import imutils

enA = 5

enB = 3

MAdir1 = 6

MAdir2 =  7

MBdir1 = 8

MBdir2 = 9

connection = SerialManager()

a = ArduinoApi(connection = connection)

print("failed")

a.pinMode(MAdir1,a.OUTPUT)
```

```
a.pinMode(MAdir2,a.OUTPUT)
a.pinMode(MBdir1,a.OUTPUT)
a.pinMode(MBdir2,a.OUTPUT)
a.pinMode(enA,a.OUTPUT)
a.pinMode(enB,a.OUTPUT)

cameraResolution = (320, 240)

# initialize the camera and grab a reference to the raw camera capture
camera = PiCamera()
camera.resolution = cameraResolution
camera.framerate = 32
camera.brightness = 60
camera.rotation = 180
rawCapture = PiRGBArray(camera, size=cameraResolution)

# allow the camera to warmup
time.sleep(2)

def findTrafficSign():
    '''
    This function find blobs with blue color on the image.
    After blobs were found it detects the largest square blob, that must be the sign.
    '''


    # define range HSV for blue color of the traffic sign
```

```
lower_blue = np.array([90,80,50])
upper_blue = np.array([110,255,255])

while True:
    # The use_video_port parameter controls whether the camera's image or video port is used
    # to capture images. It defaults to False which means that the camera's image port is used.
    # This port is slow but produces better quality pictures.
    # If you need rapid capture up to the rate of video frames, set this to True.
    camera.capture(rawCapture, use_video_port=True, format='bgr')

    # At this point the image is available as stream.array
    frame = rawCapture.array

    frameArea = frame.shape[0]*frame.shape[1]

    # convert color image to HSV color scheme
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # define kernel for smoothing
    kernel = np.ones((3,3),np.uint8)
    # extract binary image with active blue regions
    mask = cv2.inRange(hsv, lower_blue, upper_blue)
    # morphological operations
    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)

    # find contours in the mask
```

```
cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[-2]

# defite string variable to hold detected sign description
detectedTrafficSign = None

# define variables to hold values during loop
largestArea = 0
largestRect = None

# only proceed if at least one contour was found
if len(cnts) > 0:
    for cnt in cnts:
        # Rotated Rectangle. Here, bounding rectangle is drawn with minimum area,
        # so it considers the rotation also. The function used is cv2.minAreaRect().
        # It returns a Box2D structure which contains following detals -
        # ( center (x,y), (width, height), angle of rotation ).
        # But to draw this rectangle, we need 4 corners of the rectangle.
        # It is obtained by the function cv2.boxPoints()
        rect = cv2.minAreaRect(cnt)
        box = cv2.boxPoints(rect)
        box = np.int0(box)

        # count euclidian distance for each side of the rectangle
        sideOne = np.linalg.norm(box[0]-box[1])
        sideTwo = np.linalg.norm(box[0]-box[3])
        # count area of the rectangle
        area = sideOne*sideTwo
```

```
            # find the largest rectangle within all contours
            if area > largestArea:
                largestArea = area
                largestRect = box

    if largestArea > frameArea*0.02:
        # draw contour of the found rectangle on  the original image
        cv2.drawContours(frame,[largestRect],0,(0,0,255),2)

        # cut and warp interesting area
        warped = four_point_transform(mask, [largestRect][0])

        # show an image if rectangle was found
        #cv2.imshow("Warped", cv2.bitwise_not(warped))

            # use function to detect the sign on the found rectangle
        detectedTrafficSign = identifyTrafficSign(warped)
        #print(detectedTrafficSign)



            # write the description of the sign on the original image
        cv2.putText(frame, detectedTrafficSign, (100, 100), cv2.FONT_HERSHEY_SIMPLEX, 0.65, (0,
255, 0), 2)

    # show original image
    cv2.imshow("Original", frame)
    print(detectedTrafficSign)
    if detectedTrafficSign == 'Turn Back':
```

```
    a.analogWrite(enA,150)
    a.analogWrite(enB,100)
    a.digitalWrite(MAdir1,a.HIGH)
    a.digitalWrite(MAdir2,a.LOW)
    a.digitalWrite(MBdir1,a.HIGH)
    a.digitalWrite(MBdir2,a.LOW)


elif detectedTrafficSign == 'Move Straight':
    a.analogWrite(enA,150)
    a.analogWrite(enB,100)
    a.digitalWrite(MAdir1,a.LOW)
    a.digitalWrite(MAdir2,a.HIGH)
    a.digitalWrite(MBdir1,a.LOW)
    a.digitalWrite(MBdir2,a.HIGH)

elif detectedTrafficSign == 'Turn Right':
    a.analogWrite(enA,100)
    a.analogWrite(enB,100)
    a.digitalWrite(MAdir1,a.HIGH)
    a.digitalWrite(MAdir2,a.LOW)
    a.digitalWrite(MBdir1,a.LOW)
    a.digitalWrite(MBdir2,a.HIGH)

elif detectedTrafficSign == 'Turn Left':
    a.analogWrite(enA,100)
```

```
            a.analogWrite(enB,100)
            a.digitalWrite(MAdir1,a.LOW)
            a.digitalWrite(MAdir2,a.HIGH)
            a.digitalWrite(MBdir1,a.HIGH)
            a.digitalWrite(MBdir2,a.LOW)



        elif detectedTrafficSign == None:
            a.analogWrite(enA,0)
            a.analogWrite(enB,0)
            detectedTrafficSign = None

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

        # if the q key was pressed, break from the loop
        if cv2.waitKey(1) & 0xFF is ord('q'):
            cv2.destroyAllWindows()
            print("Stop programm and close all windows")
            break



def identifyTrafficSign(image):
    '''
    In this function we select some ROI in which we expect to have the sign parts. If the ROI has more
active pixels than threshold we mark it as 1, else 0
    After path through all four regions, we compare the tuple of ones and zeros with keys in dictionary
SIGNS_LOOKUP
```

```
    '''

    # define the dictionary of signs segments so we can identify
    # each signs on the image
    SIGNS_LOOKUP = {
        (1, 0, 0, 1): 'Turn Right', # turnRight
        (0, 0, 1, 1): 'Turn Left', # turnLeft
        (0, 1, 0, 1): 'Move Straight', # moveStraight
        (1, 0, 1, 1): 'Turn Back', # turnBack
    }

    THRESHOLD = 150

    image = cv2.bitwise_not(image)
    # (roiH, roiW) = roi.shape
    #subHeight = thresh.shape[0]/10
    #subWidth = thresh.shape[1]/10
    (subHeight, subWidth) = np.divide(image.shape, 10)
    subHeight = int(subHeight)
    subWidth = int(subWidth)

    # mark the ROIs borders on the image
    #cv2.rectangle(image, (subWidth, 4*subHeight), (3*subWidth, 9*subHeight), (0,255,0),2) # left block
    #cv2.rectangle(image, (4*subWidth, 4*subHeight), (6*subWidth, 9*subHeight), (0,255,0),2) # center block
    #cv2.rectangle(image, (7*subWidth, 4*subHeight), (9*subWidth, 9*subHeight), (0,255,0),2) # right block
```

```
    #cv2.rectangle(image, (3*subWidth, 2*subHeight), (7*subWidth, 4*subHeight), (0,255,0),2) # top
block


    # substract 4 ROI of the sign thresh image
    leftBlock = image[4*subHeight:9*subHeight, subWidth:3*subWidth]
    centerBlock = image[4*subHeight:9*subHeight, 4*subWidth:6*subWidth]
    rightBlock = image[4*subHeight:9*subHeight, 7*subWidth:9*subWidth]
    topBlock = image[2*subHeight:4*subHeight, 3*subWidth:7*subWidth]

    # we now track the fraction of each ROI
    leftFraction = np.sum(leftBlock)/(leftBlock.shape[0]*leftBlock.shape[1])
    centerFraction = np.sum(centerBlock)/(centerBlock.shape[0]*centerBlock.shape[1])
    rightFraction = np.sum(rightBlock)/(rightBlock.shape[0]*rightBlock.shape[1])
    topFraction = np.sum(topBlock)/(topBlock.shape[0]*topBlock.shape[1])

    segments = (leftFraction, centerFraction, rightFraction, topFraction)
    segments = tuple(1 if segment > THRESHOLD else 0 for segment in segments)

    cv2.imshow("Warped", image)



    if segments in SIGNS_LOOKUP:
        return SIGNS_LOOKUP[segments]
    else:
        return None



def main():
```

```
    findTrafficSign()



if _name_ == '_main_':
    main()
```