Ain shams university

MACHINE VISION

Project 3

Omar Khaled El Baz	14p1013
Mostafa Emad Hamdy	14p8014
Mohamed Abd El Kader	14p1056
Marwan Ahmed Wahid	14p8004
Youssef hussien	14p1024
Mohamed Ali Fathy	14p8033

Contents

List of Figures

1 Abstract

Our mega project is divided into three sections mechanical, electrical, and control the objective of the project is to make an action when it sees a sign through the camera as when it get a forward arrow on the image it moves forward and this algotherim is applied on all four signs

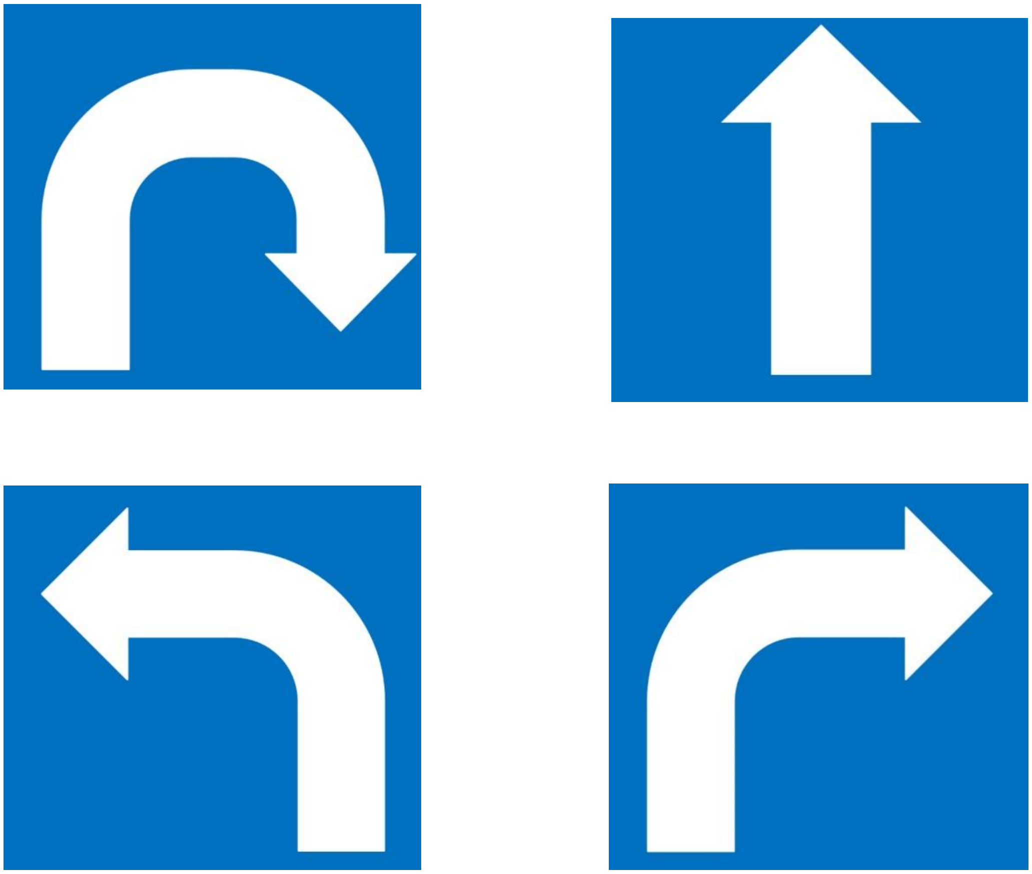
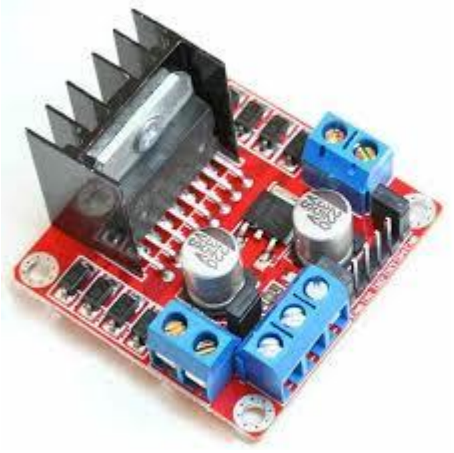


Figure 1: Signs

2 Components

Component	Picture
Raspberry pi	
Car kit	
Motor driver	

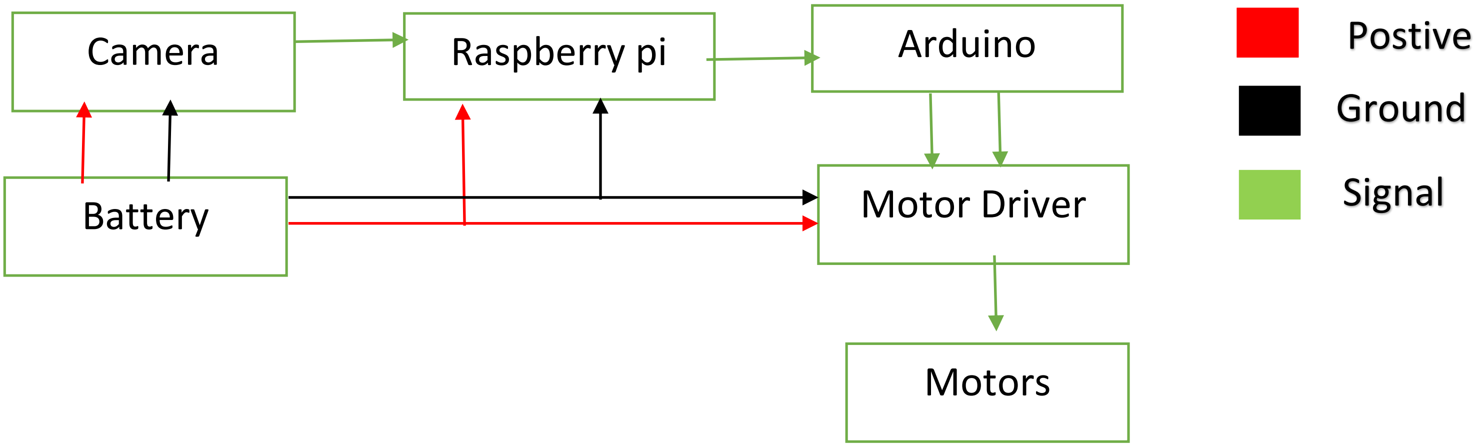
Ethernet cable	
Pi Camera	
Power bank	

Arduino	

3 Implementation

1. Setting up Raspberry pi on Windows using vnc viewer(get ip address from putty using “ping -4 raspberrypi.local”)
2. Setting Arduino with Raspberry pi (nanpy library)
3. Setting up picamera with raspberry pi (enable camera from “sudo raspi-config”)
4. Install opencv3.4 on raspberrypi (https://www.pyimagesearch.com/2017/09/04/raspbian-stretch-install-opencv-3-python-on-your-raspberry-pi/)
5. Fixing driver and pi on RC car
6. Connecting motors to driver
7. Connection between driver and raspberry pi
8. Fixing picamera on RC car
9. Testing

4 Electrical Connection SID



5 Flow Chart

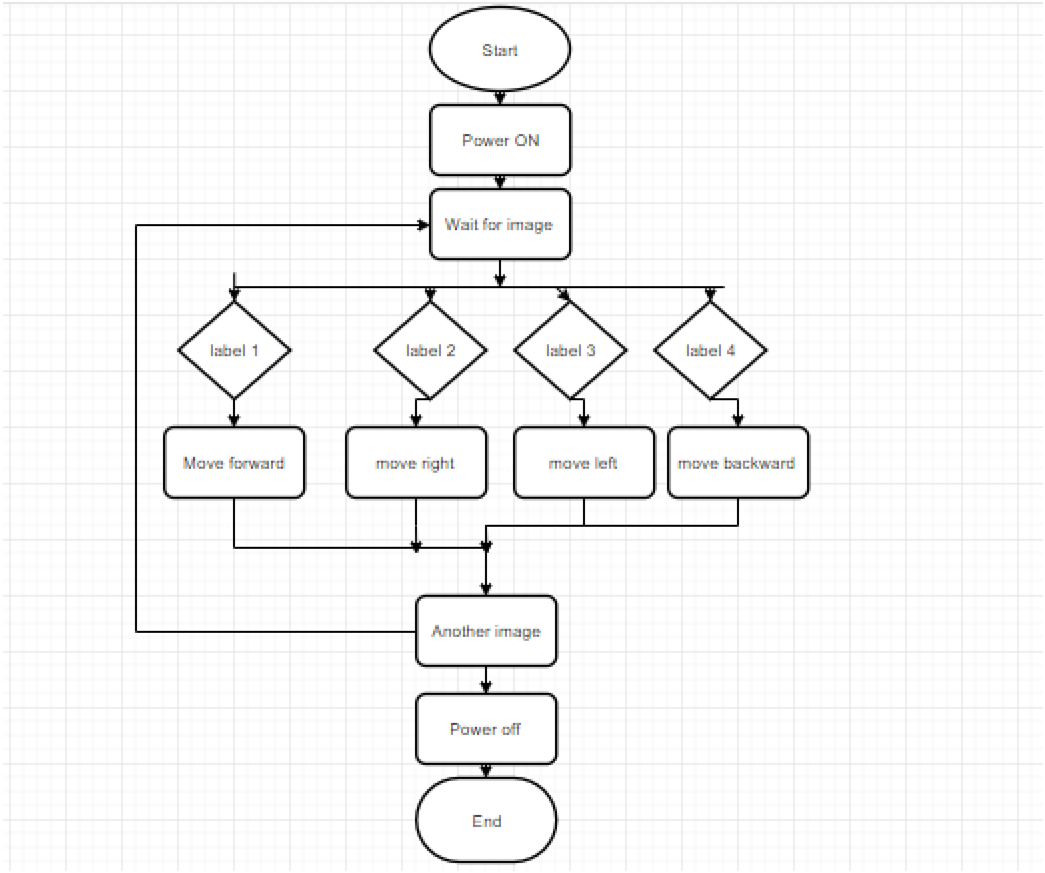


Figure 2: Flowchart

6 Technical discussion

The technique used in this project is that when we detect an image of a sign it forms the action in this image as if it sees a blue image with an arrow in it, it forms an action corresponding to this arrow when it is a forward arrow the car goes forward, right it goes right as shown in the video attached in our project we access the RC car through either net cable to monitor what the camera see when the camera detects an image it appears on the pc screen and draws a rectangle on the borders of the image then we write on the screen what is the corresponding instruction of that image for example when it is a forward arrow we write on the screen moving forward then we give instructions to our RC car as we send PWM to our motors driver related to that action for moving forward we make the two wheels moves forward for turning we make one wheel turns forward and the other turns backward so that the RC card rotates about its' center to the right or the left

7 Bill of Material

Component	Price	Quantity	Total
Raspberry Pi	1300	1	1300
Arduino	200	1	200
Webcam	150	1	150
RC kit	260	1	260
Motor Driver	200	1	200
Memory card	120	1	90

Total = 2200 L.E

8 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
enA = 5
enB = 3
MAdir1 = 6
MAdir2 = 7
MBdir1 = 8
MBdir2 = 9
connection = SerialManager()
a = ArduinoApi(connection = connection)
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()
```