**PHASE 3: Traffic Management System on IOT**

**IOT Devices:**

* Inductive Loop Detectors (ILD)
* Microwave Radar Detectors
* Infrared Sensors
* Ultrasonic Detectors
* Acoustic Detectors
* Magnetometers
* Automatic Number Plate Recognition (ANPR) Systems
* Travel Times and Speed over Distance
* LED Bulbs

**Relay Module:**

**Pin names:**

VCC - Supply voltage

GND - Ground

IN - Control signal (e.g. from micro-controller)

NC - Normally closed

COM - Common pin

**OPERATIONS:**

* Set up the circuit according to the circuit diagram If another Arduino Mega is available, multiple ultrasonic sensors can be used instead of simulating the values.
* For multiple ultrasonic sensors, use appropriate free ports and ensure appropriate modification in the code.
* If the boards used are different, ensure appropriate connection of the NRF modules with the recommended pin numbers and modify the code appropriately
* Download the transmitter and receiver codes from the link provided and run it on the respective boards.
* In case multiple ultrasonic sensors are used instead of simulation, a predesigned function called ‘ultrasound’ is available, which returns the sum total distances obtained from two ultrasonic sensors.
* Validate the results by checking the data on the Python



## Arduino Setup:

Download the latest Arduino IDE on both the computer systems.

• Include the RF24master library file. Zipped folder is available in the google drive link provided for the code.

• If RF24master is not available any equivalent suitable library can also be included provided the connections and the code are modified appropriately.

• Set the appropriate boards on both the systems



**Program:**

import numpy as np

import time

import cv2

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)

for i in (23, 25, 16, 21):

GPIO.setup(i, GPIO.OUT)

cam=cv2.VideoCapture(0)

cam.set(4,480) #Width=480

cam.set(5,480) #Height=480

cam.set(6,30) #FrameRate = 30

time.sleep(0.1)

colorLower = np.array([0,100,100]) #mid blue

colorUpper = np.array([179,255,255]) #light blue

initvert = 0

inithoriz = 0

counter = 0

xur = 0

yur = 0

xul = 0

yul = 0

xdr = 0

ydr = 0

xdl = 0

ydl = 0

t = 0

t1 = time.time()

while t < 5 and cam.isOpened():

ret,frame = cam.read()

frame = np.array(frame) #Transform frame into array

hsv = cv2.cvtColor(frame,cv2.COLOR\_BGR2HSV)

mask = cv2.inRange(hsv,colorLower,colorUpper)

mask = cv2.blur(mask,(3,3))

mask= cv2.dilate(mask,None,iterations=10)

mask= cv2.erode(mask,None,iterations=1)

mask= cv2.dilate(mask,None,iterations=5)

me,thresh = cv2.threshold(mask,127,255,cv2.THRESH\_BINARY)

cnts = cv2.findContours(thresh,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)[-2]

center = None

print("Centers")

if len(cnts) > 0:

for c in cnts:

(x,y),radius = cv2.minEnclosingCircle(c)

center = (int(x),int(y))

print(center)

radius = int(radius)

cv2.circle(frame,center,radius,(0,255,0),2)

x = int(x)

y = int(y)

if x > 240: #right

if y > 240: #up

xur = x

yur = y

if y < 240: #down

xdr = x

ydr = y

if x < 240: #left

if y > 240: #up

xul = x

yul = y

if y < 240: #down

xdl = x

ydl = y

t2 = time.time()

t = t2-t1

print("upright",xur,yur)

print("downright",xdr,ydr)

print("upleft",xul,yul)

print("downleft",xdl,ydl)

print('\n')

print("Remove calibration objects")

time.sleep(5)

while(cam.isOpened()):

ret,frame = cam.read()

frame = np.array(frame) #Transform frame into array

hsv = cv2.cvtColor(frame,cv2.COLOR\_BGR2HSV)

mask = cv2.inRange(hsv,colorLower,colorUpper)

maskhsv = cv2.resize(mask,(250,250))

mask = cv2.blur(mask,(3,3))

mask1 = cv2.resize(mask,(250,250))

#cv2.imshow("mask1",mask)

mask= cv2.dilate(mask,None,iterations=10)

mask2=cv2.resize(mask,(250,250))

#cv2.imshow("mask2",mask)

mask= cv2.erode(mask,None,iterations=1)

mask3 = cv2.resize(mask,(250,250))

#cv2.imshow("mask3",mask)

mask= cv2.dilate(mask,None,iterations=5)

mask4=cv2.resize(mask,(250,250))

#cv2.imshow("mask4",mask)

imstack = np.hstack((maskhsv,mask1,mask2,mask3,mask4))

cv2.imshow("masks",imstack)

me,thresh = cv2.threshold(mask,127,255,cv2.THRESH\_BINARY)

cv2.imshow("thresh",thresh)

cnts = cv2.findContours(thresh,cv2.RETR\_TREE,cv2.CHAIN\_APPROX\_SIMPLE)[-2]

#print(cnts)

center = None

vert = 0

horiz = 0

if len(cnts) > 0:

priority = 0

for c in cnts:

rect = cv2.minAreaRect(c)

(x,y),(width,height),angle = cv2.minAreaRect(c)

box = cv2.boxPoints(rect)

box = np.int0(box)

Area = width\*height

if Area > 16000:

priority = 1

for c in cnts:

rect = cv2.minAreaRect(c)

(x,y),(width,height),angle = cv2.minAreaRect(c)

box = cv2.boxPoints(rect)

box = np.int0(box)

Area = width\*height

x = int(x)

y = int(y)

if priority == 1 and Area > 16000:

if xul < x < xur: #vertical road

if y > yur:

GPIO.output(21,GPIO.HIGH) #Red hor

GPIO.output(23,GPIO.HIGH) #Green vert

GPIO.output(25,GPIO.LOW) #Green hor

GPIO.output(16,GPIO.LOW) #Red vert

cv2.drawContours(frame,[box],0,(0,255,0),2)

print("PRIORITY FOR VERTICAL LANE")

print ("----------------------------")

elif y < ydr:

GPIO.output(21,GPIO.HIGH) #Red hor

GPIO.output(23,GPIO.HIGH) #Green vert

GPIO.output(25,GPIO.LOW) #Green hor

GPIO.output(16,GPIO.LOW) #Red vert

cv2.drawContours(frame,[box],0,(0,255,0),2)

print("PRIORITY FOR VERTICAL LANE")

print ("----------------------------")

if ydr < y < yur: #horizontal road

if x > xur:

GPIO.output(25,GPIO.HIGH) #Green hor

GPIO.output(16,GPIO.HIGH) #Red vert

GPIO.output(21,GPIO.LOW) #Red hor

GPIO.output(23,GPIO.LOW) #Green vert

cv2.drawContours(frame,[box],0,(0,255,0),2)

print("PRIORITY FOR HORIZONTAL LANE")

print ("----------------------------")

elif x < xul:

GPIO.output(25,GPIO.HIGH) #Green hor

GPIO.output(16,GPIO.HIGH) #Red vert

GPIO.output(21,GPIO.LOW) #Red hor

GPIO.output(23,GPIO.LOW) #Green vert

cv2.drawContours(frame,[box],0,(0,255,0),2)

print("PRIORITY FOR HORIZONTAL LANE")

print ("----------------------------")

if priority == 0:

if xul < x < xur: #vertical road

if y > yur:

vert = vert +1 #up

cv2.drawContours(frame,[box],0,(0,255,0),2)

elif y < ydr:

vert = vert +1 #down

cv2.drawContours(frame,[box],0,(0,255,0),2)

else:

vert = vert

if ydr < y < yur: #horizontal road

if x > xur:

horiz = horiz +1 #right

cv2.drawContours(frame,[box],0,(0,255,0),2)

elif x < xul:

horiz = horiz +1 #left

cv2.drawContours(frame,[box],0,(0,255,0),2)

else:

horiz = horiz

if vert != initvert:

print("Cars in vertical lane: ", str(vert))

initvert = vert

print("Cars in horizontal lane: ", str(horiz))

inithoriz = horiz

print ("----------------------------")

if horiz != inithoriz:

print("Cars in vertical lane: ", str(vert))

initvert = vert

print("Cars in horizontal lane: ", str(horiz))

inithoriz = horiz

print ("----------------------------")

if vert < horiz:

GPIO.output(25,GPIO.HIGH) #Green hor

GPIO.output(16,GPIO.HIGH) #Red vert

GPIO.output(21,GPIO.LOW) #Red hor

GPIO.output(23,GPIO.LOW) #Green vert

if horiz < vert:

GPIO.output(21,GPIO.HIGH) #Red hor

GPIO.output(23,GPIO.HIGH) #Green vert

GPIO.output(25,GPIO.LOW) #Green hor

GPIO.output(16,GPIO.LOW) #Red vert

hsvim = cv2.resize(hsv,(500,500))

frameim = cv2.resize(frame,(500,500))

imstack2 = np.hstack((hsvim,frameim))

cv2.imshow("Frame + hsv",imstack2)

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cam.release()

cv2.destroyAllWindows()

GPIO.cleanup()

**OUTPUT:**

