import firebase\_admin

from firebase\_admin import credentials

from firebase\_admin import db

import serial

import time

# http://picamera.readthedocs.io/en/latest/recipes2.html#web-streaming

import io

import picamera

import logging

import socketserver

from threading import Condition

from http import server

import Adafruit\_DHT

from mpu6050 import mpu6050

import RPi.GPIO as GPIO

trial = "france"

PAGE="""\

<html>

<head>

<title>Raspberry Pi - Surveillance Camera</title>

</head>

<body>

<center><img src="stream.mjpg" width="640" height="480"></center>

</body>

</html>

"""

#mpu = mpu6050(0x68)

GPIO.setwarnings(False)

#GPIO Pins Buzzer

buzzPin =22

#GPIO Mode (BOARD / BCM)

GPIO.setmode(GPIO.BCM)

#GPIO Pins(Straight)

GPIO\_TRIGGER\_ST = 18

GPIO\_ECHO\_ST = 24

#GPIO Pins(Right)

GPIO\_TRIGGER\_RI = 20

GPIO\_ECHO\_RI = 21

#GPIO Pins(Left)

GPIO\_TRIGGER\_LE = 5

GPIO\_ECHO\_LE = 6

#GPIO Pins(Back)

GPIO\_TRIGGER\_BK = 7

GPIO\_ECHO\_BK = 8

GPIO.setup(buzzPin, GPIO.OUT)

GPIO.setup(GPIO\_TRIGGER\_BK, GPIO.OUT)

GPIO.setup(GPIO\_ECHO\_BK, GPIO.IN)

#GPIO Pin (IN / OUT)

GPIO.setup(GPIO\_TRIGGER\_ST, GPIO.OUT)

GPIO.setup(GPIO\_ECHO\_ST, GPIO.IN)

GPIO.setup(GPIO\_TRIGGER\_RI, GPIO.OUT)

GPIO.setup(GPIO\_ECHO\_RI, GPIO.IN)

GPIO.setup(GPIO\_TRIGGER\_LE, GPIO.OUT)

GPIO.setup(GPIO\_ECHO\_LE, GPIO.IN)

arduino = serial.Serial(port='/dev/ttyACM0', baudrate=115200, timeout=.1)

def write\_read(x):

print(x)

arduino.write(bytes(x, 'utf-8'))

time.sleep(0.05)

data = arduino.readline()

return data

# Fetch the service account key JSON file contents

cred = credentials.Certificate('fyp1-86727-firebase-adminsdk-hbuz9-8abb2d102c.json')

def listener(event):

flag = 0

#print(event.event\_type) # can be 'put' or 'patch'

temp=str(event.path)

temp=temp.split("/")

#print(event.path) # relative to the reference, it seems

#print(event.data) # new data at /reference/event.path. None if deleted

value=str(event.data)

if value=="1":

if temp[1]=="Left":

dir = "4"

value = write\_read(dir)

print(value)

print('moving left!')

elif temp[1]=="Right":

dir = "3"

value = write\_read(dir)

print(value)

print('moving right!')

elif temp[1]=="Backward":

dir = "2"

value = write\_read(dir)

print(value)

print('moving backward!')

elif temp[1]=="Forward":

dir = "1"

value = write\_read(dir)

print(value)

print('moving forward!')

if value=="0":

dir = "0"

value = write\_read(dir)

print(value)

print('not moving!')

distStraight = distance\_straight()

print ("Straight Distance = %.1f cm" % distStraight)

distRight = distance\_right()

print ("Right Distance = %.1f cm" % distRight)

distLeft = distance\_left()

print ("Left Distance = %.1f cm" % distLeft)

distBack = distance\_back()

print ("Back Distance = %.1f cm" % distBack)

if flag%5 == 0:

humidity, temperature = Adafruit\_DHT.read\_retry(11, 17)

print('Temp: {0:0.1f} C Humidity: {1:0.1f} %'.format(temperature, humidity))

print("")

flag += 1

# print("Temp : "+str(mpu.get\_temp()))

# print()

# accel\_data = mpu.get\_accel\_data()

# print("Acc X : "+str(accel\_data['x']))

# print("Acc Y : "+str(accel\_data['y']))

# print("Acc Z : "+str(accel\_data['z']))

# print()

# gyro\_data = mpu.get\_gyro\_data()

# print("Gyro X : "+str(gyro\_data['x']))

# print("Gyro Y : "+str(gyro\_data['y']))

# print("Gyro Z : "+str(gyro\_data['z']))

print()

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

time.sleep(1)

GPIO.output(buzzPin, GPIO.LOW)

time.sleep(1)

GPIO.output(buzzPin, GPIO.HIGH)

time.sleep(1)

# firebase\_admin.db.reference('https://fyp1-86727-default-rtdb.firebaseio.com').listen(listener)

# Initialize the app with a service account, granting admin privileges

firebase\_admin.initialize\_app(cred, {

'databaseURL': 'https://fyp1-86727-default-rtdb.firebaseio.com'

})

# As an admin, the app has access to read and write all data, regradless of Security Rules

ref=firebase\_admin.db.reference('').listen(listener)

# ref = db.reference('')

# print(ref)

def distance\_straight():

# set Trigger to HIGH

GPIO.output(GPIO\_TRIGGER\_ST, True)

# set Trigger after 0.01ms to LOW

time.sleep(0.00001)

GPIO.output(GPIO\_TRIGGER\_ST, False)

StartTime\_S = time.time()

StopTime\_S = time.time()

# save StartTime

while GPIO.input(GPIO\_ECHO\_ST) == 0:

StartTime\_S = time.time()

# save time of arrival

while GPIO.input(GPIO\_ECHO\_ST) == 1:

StopTime\_S = time.time()

# time difference between start and arrival

TimeElapsed\_S = StopTime\_S - StartTime\_S

# multiply with the sonic speed (34300 cm/s)

# and divide by 2, because there and back

distance\_st = (TimeElapsed\_S \* 34300) / 2

return distance\_st

def distance\_right():

# set Trigger to HIGH

GPIO.output(GPIO\_TRIGGER\_RI, True)

# set Trigger after 0.01ms to LOW

time.sleep(0.00001)

GPIO.output(GPIO\_TRIGGER\_RI, False)

StartTime\_R = time.time()

StopTime\_R = time.time()

# save StartTime

while GPIO.input(GPIO\_ECHO\_RI) == 0:

StartTime\_R = time.time()

# save time of arrival

while GPIO.input(GPIO\_ECHO\_RI) == 1:

StopTime\_R = time.time()

# time difference between start and arrival

TimeElapsed\_R = StopTime\_R - StartTime\_R

# multiply with the sonic speed (34300 cm/s)

# and divide by 2, because there and back

distance\_r = (TimeElapsed\_R \* 34300) / 2

return distance\_r

def distance\_back():

# set Trigger to HIGH

GPIO.output(GPIO\_TRIGGER\_BK, True)

# set Trigger after 0.01ms to LOW

time.sleep(0.00001)

GPIO.output(GPIO\_TRIGGER\_BK, False)

StartTime\_B = time.time()

StopTime\_B = time.time()

# save StartTime

while GPIO.input(GPIO\_ECHO\_BK) == 0:

StartTime\_B = time.time()

# save time of arrival

while GPIO.input(GPIO\_ECHO\_BK) == 1:

StopTime\_B = time.time()

# time difference between start and arrival

TimeElapsed\_B = StopTime\_B - StartTime\_B

# multiply with the sonic speed (34300 cm/s)

# and divide by 2, because there and back

distance\_b = (TimeElapsed\_B \* 34300) / 2

return distance\_b

def distance\_left():

# set Trigger to HIGH

GPIO.output(GPIO\_TRIGGER\_LE, True)

# set Trigger after 0.01ms to LOW

time.sleep(0.00001)

GPIO.output(GPIO\_TRIGGER\_LE, False)

StartTime\_L = time.time()

StopTime\_L = time.time()

# save StartTime

while GPIO.input(GPIO\_ECHO\_LE) == 0:

StartTime\_L = time.time()

# save time of arrival

while GPIO.input(GPIO\_ECHO\_LE) == 1:

StopTime\_L = time.time()

# time difference between start and arrival

TimeElapsed\_L = StopTime\_L - StartTime\_L

# multiply with the sonic speed (34300 cm/s)

# and divide by 2, because there and back

distance\_l = (TimeElapsed\_L \* 34300) / 2

return distance\_l

class StreamingOutput(object):

def \_\_init\_\_(self):

self.frame = None

self.buffer = io.BytesIO()

self.condition = Condition()

def write(self, buf):

if buf.startswith(b'\xff\xd8'):

# New frame, copy the existing buffer's content and notify all

# clients it's available

self.buffer.truncate()

with self.condition:

self.frame = self.buffer.getvalue()

self.condition.notify\_all()

self.buffer.seek(0)

return self.buffer.write(buf)

class StreamingHandler(server.BaseHTTPRequestHandler):

def do\_GET(self):

if self.path == '/':

self.send\_response(301)

self.send\_header('Location', '/index.html')

self.end\_headers()

elif self.path == '/index.html':

content = PAGE.encode('utf-8')

self.send\_response(200)

self.send\_header('Content-Type', 'text/html')

self.send\_header('Content-Length', len(content))

self.end\_headers()

self.wfile.write(content)

elif self.path == '/stream.mjpg':

self.send\_response(200)

self.send\_header('Age', 0)

self.send\_header('Cache-Control', 'no-cache, private')

self.send\_header('Pragma', 'no-cache')

self.send\_header('Content-Type', 'multipart/x-mixed-replace; boundary=FRAME')

self.end\_headers()

try:

while True:

with output.condition:

output.condition.wait()

frame = output.frame

self.wfile.write(b'--FRAME\r\n')

self.send\_header('Content-Type', 'image/jpeg')

self.send\_header('Content-Length', len(frame))

self.end\_headers()

self.wfile.write(frame)

self.wfile.write(b'\r\n')

except Exception as e:

logging.warning(

'Removed streaming client %s: %s',

self.client\_address, str(e))

else:

self.send\_error(404)

self.end\_headers()

class StreamingServer(socketserver.ThreadingMixIn, server.HTTPServer):

allow\_reuse\_address = True

daemon\_threads = True

with picamera.PiCamera(resolution='640x480', framerate=24) as camera:

output = StreamingOutput()

#Uncomment the next line to change your Pi's Camera rotation (in degrees)

camera.rotation = 180

camera.start\_recording(output, format='mjpeg')

try:

address = ('', 8000)

server = StreamingServer(address, StreamingHandler)

server.serve\_forever()

except:

camera.stop\_recording()

finally:

camera.stop\_recording()