**app.py**

import numpy as np

import cv2

from time import sleep

# import serial

# arduinoSerialPort = '/dev/ttyACM0'

sdThresh = 7

font = cv2.FONT\_HERSHEY\_SIMPLEX

def distMap(frame1, frame2):

frame1\_32 = np.float32(frame1)

frame2\_32 = np.float32(frame2)

diff32 = frame1\_32 - frame2\_32

norm32 = np.sqrt(diff32[:,:,0]\*\*2 + diff32[:,:,1]\*\*2 + diff32[:,:,2]\*\*2)/np.sqrt(255\*\*2 + 255\*\*2 + 255\*\*2)

dist = np.uint8(norm32\*255)

return dist

# camera\_stream = "http://127.0.0.1:5000/video\_feed"

# camera\_stream = "http://192.168.137.160:81/stream"

# camera\_stream = 0

camera\_stream = input("type cam id you want to stream, or url for a webcam: ")

cap = cv2.VideoCapture(int(camera\_stream))

\_, frame1 = cap.read()

\_, frame2 = cap.read()

app = False

while(1):

if app == 0:

\_, frame3 = cap.read()

cv2.imshow('dist', frame3)

print("callibration.. press c to continue")

k = cv2.waitKey(10) & 0xFF

if k == ord('c'):

app = 1

if app == 1:

# \_, frame3 = cap.read()

rows, cols, \_ = np.shape(frame3)

cv2.imshow('dist', frame3)

dist = distMap(frame1, frame3)

frame1 = frame2

\_, frame2 = cap.read()

mod = cv2.GaussianBlur(dist, (9,9), 0)

\_, thresh = cv2.threshold(mod, 100, 255, 0)

\_, stDev = cv2.meanStdDev(mod)

cv2.imshow('dist', mod)

cv2.putText(frame2, "Sensor - {}".format(round(stDev[0][0],0)), (70, 70), font, 1, (255, 0, 255), 1, cv2.LINE\_AA)

if stDev > sdThresh:

# \_, frameCallib = cap.read()

objectsDeviation = round(stDev[0][0],0)

## arduino serial

# serialData = 'road1/{}/road2/14'.format(objectsDeviation)

# serialData\_encode=serialData.encode()

# print(serialData)

# ser = serial.Serial(arduinoSerialPort)

# ser.baudrate = 9600

# ser.write(serialData\_encode)

# print(serialData\_encode)

# time.sleep(1)

# ser.close()

# print(objectsDeviation)

######

cv2.imshow('frame', frame2)

if cv2.waitKey(1) & 0xFF == 27:

break

if cv2.waitKey(33) == ord('a'):

print('callib called')

callibrate()

cap.release()

cv2.destroyAllWindows()

**microprocessor\_codes/serialEventTest.cpp**

String data;

String dataPart1;

String dataPart2;

long timeConserved1 = 0;

long timeUsing1 = 0;

long timeConserved2 = 0;

long timeUsing2 = 0;

const int redLed = 13;

const int greenLed = 6;

const int yellowLed = 7;

const int yellowLed1 = 8;

int yellowState1 = LOW;

int redState = LOW;

int yellowState = LOW;

int greenState = LOW;

unsigned long currentTime = millis();

long startTime = 0;

long elapsedTime = currentTime - startTime;

long cycleTime = 23000;

unsigned long greenTimeGreater;

unsigned long greenTimeLess;

bool updateTime = true;

bool stringComplete = false;

void setup() {

pinMode(redLed, OUTPUT);

pinMode(yellowLed, OUTPUT);

pinMode(greenLed, OUTPUT);

pinMode(yellowLed1, OUTPUT);

digitalWrite(redLed,0);

Serial.begin(9600);

data.reserve(200);

}

void loop() {

if(stringComplete)

{

data.trim();

Serial.println(data);

if (data == "hello/1"){

digitalWrite(redLed,1);

}

if (data == "hello/0"){

digitalWrite(redLed,0);

}

data = "";

stringComplete = false;

}

}

String getValue(String data, char separator, int index)

{

int found = 0;

int strIndex[] = {0, -1};

int maxIndex = data.length()-1;

for(int i=0; i<=maxIndex && found<=index; i++){

if(data.charAt(i)==separator || i==maxIndex){

found++;

strIndex[0] = strIndex[1]+1;

strIndex[1] = (i == maxIndex) ? i+1 : i;

}

}

return found>index ? data.substring(strIndex[0], strIndex[1]) : "";

}

void serialEvent() {

while (Serial.available()) {

char inChar = (char)Serial.read();

data += inChar;

if (inChar == '\n') {

stringComplete = true;

}

}

}

**multicam/twoCams\_app.py**

import numpy as np

import cv2

sdThresh = 7

font = cv2.FONT\_HERSHEY\_SIMPLEX

import serial

import time

arduinoSerialPort = '/dev/ttyACM0'

def serialEvent(serialData):

serialData += "\n"

serialData\_encode = serialData.encode()

ser = serial.Serial(arduinoSerialPort)

ser.baudrate = 9600

ser.write(serialData\_encode)

time.sleep(0.1)

ser.close()

def distMap(frame1, frame2):

frame1\_32 = np.float32(frame1)

frame2\_32 = np.float32(frame2)

diff32 = frame1\_32 - frame2\_32

norm32 = np.sqrt(diff32[:,:,0]\*\*2 + diff32[:,:,1]\*\*2 + diff32[:,:,2]\*\*2)/np.sqrt(255\*\*2 + 255\*\*2 + 255\*\*2)

dist = np.uint8(norm32\*255)

return dist

video\_capture\_0 = cv2.VideoCapture(0)

video\_capture\_1 = cv2.VideoCapture(2)

\_, frame1\_0 = video\_capture\_0.read()

\_, frame2\_0 = video\_capture\_0.read()

\_, frame1\_1 = video\_capture\_1.read()

\_, frame2\_1 = video\_capture\_1.read()

app = False

while True:

if app == False:

\_, frame3\_0 = video\_capture\_0.read()

\_, frame3\_1 = video\_capture\_1.read()

cv2.imshow('Camera\_0', frame3\_0)

cv2.imshow('Camera\_1', frame3\_1)

print("callibration.. press c to continue")

k = cv2.waitKey(10) & 0xFF

if k == ord('c'):

app = True

# ret0, frame0 = video\_capture\_0.read()

# ret1, frame1 = video\_capture\_1.read()

if app == True:

rows\_0, cols\_0, \_ = np.shape(frame3\_0)

rows\_1, cols\_1, \_ = np.shape(frame3\_1)

cv2.imshow('Camera\_0', frame3\_0)

cv2.imshow('Camera\_1', frame3\_1)

dist\_0 = distMap(frame1\_0, frame3\_0)

dist\_1 = distMap(frame1\_1, frame3\_1)

frame1\_0 = frame2\_0

\_, frame2\_0 = video\_capture\_0.read()

frame1\_1 = frame2\_1

\_, frame2\_1 = video\_capture\_1.read()

mod\_0 = cv2.GaussianBlur(dist\_0, (9,9), 0)

\_, thresh\_0 = cv2.threshold(mod\_0, 100, 255, 0)

\_, stDev\_0 = cv2.meanStdDev(mod\_0)

mod\_1 = cv2.GaussianBlur(dist\_1, (9,9), 0)

\_, thresh\_1 = cv2.threshold(mod\_1, 100, 255, 0)

\_, stDev\_1 = cv2.meanStdDev(mod\_1)

# cv2.imshow('Camera\_0', mod\_0)

# cv2.imshow('Camera\_1', mod\_1)

cv2.imshow('Camera\_0', frame2\_0)

cv2.imshow('Camera\_1', frame2\_1)

cv2.putText(frame2\_0, "Sensor\_0 - {}".format(round(stDev\_0[0][0],0)), (70, 70), font, 1, (255, 0, 255), 1, cv2.LINE\_AA)

cv2.putText(frame2\_1, "Sensor\_1 - {}".format(round(stDev\_1[0][0],0)), (70, 70), font, 1, (255, 0, 255), 1, cv2.LINE\_AA)

if (stDev\_0 > sdThresh or stDev\_1 > sdThresh):

objectsDeviation\_0 = round(stDev\_0[0][0],0)

objectsDeviation\_1 = round(stDev\_1[0][0],0)

print("deviation: {}, {}".format(objectsDeviation\_0, objectsDeviation\_1))

deviationData = "road1/{}/road2/{}".format(objectsDeviation\_0,objectsDeviation\_1)

serialEvent(deviationData)

k = cv2.waitKey(10) & 0xFF

if k == ord('c'):

app = False

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

break

video\_capture\_0.release()

video\_capture\_1.release()

cv2.destroyAllWindows()

**microprocessor\_codes/lights\_controlled.cpp**

String data;

const int redLed = 3;

const int greenLed = 4;

const int yellowLed = 5;

int redState = LOW;

int yellowState = LOW;

int greenState = LOW;

unsigned long currentTime = millis();

long startTime = 0;

long elapsedTime = currentTime - startTime;

long cycleTime = 30000;

void setup() {

pinMode(redLed, OUTPUT);

pinMode(yellowLed, OUTPUT);

pinMode(greenLed, OUTPUT);

Serial.begin(9600);

}

void loop() {

currentTime = millis();

elapsedTime = currentTime - startTime;

if (elapsedTime > cycleTime)

{

startTime = currentTime;

}

Serial.println("Start");

if (elapsedTime > 20000)

{

redState = HIGH;

Serial.println("Red");

}

else redState = LOW;

if (elapsedTime > 15000 && elapsedTime < 20000)

{

yellowState = HIGH;

Serial.println("Yellow");

}

else yellowState = LOW;

if (elapsedTime < 15000)

{

greenState = HIGH;

Serial.println("Green");

}

else greenState = LOW;

digitalWrite(redLed, redState);

digitalWrite(yellowLed, redState);

digitalWrite(greenLed, redState);

}

String getValue(String data, char separator, int index)

{

int found = 0;

int strIndex[] = {0, -1};

int maxIndex = data.length()-1;

for(int i=0; i<=maxIndex && found<=index; i++){

if(data.charAt(i)==separator || i==maxIndex){

found++;

strIndex[0] = strIndex[1]+1;

strIndex[1] = (i == maxIndex) ? i+1 : i;

}

}

return found>index ? data.substring(strIndex[0], strIndex[1]) : "";

}

**multicam/app.py**

from \_\_future\_\_ import print\_function

from detector import BasicMotionDetector

from imutils.video import VideoStream

import numpy as np

import datetime

import imutils

import time

import cv2

print("[INFO] starting cameras...")

webcam = VideoStream(src=0).start()

picam = VideoStream(src=1).start()

time.sleep(2.0)

camMotion = BasicMotionDetector()

piMotion = BasicMotionDetector()

total = 0

while True:

frames = []

for (stream, motion) in zip((webcam, picam), (camMotion, piMotion)):

frame = stream.read()

frame = imutils.resize(frame, width=400)

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

gray = cv2.GaussianBlur(gray, (21, 21), 0)

locs = motion.update(gray)

if total < 32:

frames.append(frame)

continue

frames.append(frame)

total += 1

timestamp = datetime.datetime.now()

ts = timestamp.strftime("%A %d %B %Y %I:%M:%S%p")

for (frame, name) in zip(frames, ("Webcam", "Picamera")):

cv2.putText(frame, ts, (10, frame.shape[0] - 10),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.35, (0, 0, 255), 1)

cv2.imshow(name, frame)

key = cv2.waitKey(1) & 0xFF

if key == ord("q"):

break

print("[INFO] cleaning up...")

cv2.destroyAllWindows()

webcam.stop()

picam.stop()

**multicam/detector.py**

import imutils

import cv2

class BasicMotionDetector:

def \_\_init\_\_(self, accumWeight=0.5, deltaThresh=5, minArea=5000):

self.isv2 = imutils.is\_cv2()

self.accumWeight = accumWeight

self.deltaThresh = deltaThresh

self.minArea = minArea

self.avg = None

def update(self, image):

locs = []

if self.avg is None:

self.avg = image.astype("float")

return locs

cv2.accumulateWeighted(image, self.avg, self.accumWeight)

frameDelta = cv2.absdiff(image, cv2.convertScaleAbs(self.avg))

thresh = cv2.threshold(frameDelta, self.deltaThresh, 255,

cv2.THRESH\_BINARY)[1]

thresh = cv2.dilate(thresh, None, iterations=2)

cnts = cv2.findContours(thresh, cv2.RETR\_EXTERNAL,

cv2.CHAIN\_APPROX\_SIMPLE)

cnts = imutils.grab\_contours(cnts)

for c in cnts:

if cv2.contourArea(c) > self.minArea:

locs.append(c)

return locs

**microprocessor\_codes/trafficLightsApp.cpp**

long trafficLight1Time = millis();

bool updateTraffic1Time = true;

long traffic1Delay = 0;

long traffic1DelayConserved = 0;

void setup() {

Serial.begin(9600);

}

void loop() {

if (Serial.available() != 0){

String data = Serial.readStringUntil("\n");

data.trim();

Serial.println("catched data from serial " + data);

traffic1DelayConserved = data.toInt();

}

if (updateTraffic1Time == true){

traffic1Delay = traffic1DelayConserved;

}

controlTraffic();

}

void controlTraffic(){

if(trafficLight1Time + traffic1Delay > millis()){

updateTraffic1Time = false;

Serial.println("light is on " + String(traffic1Delay));

updateTraffic1Time = true;

}

}

**serialEvents.py**

import serial

import time

from random import randint

arduinoSerialPort = '/dev/ttyACM0'

def serialEvent(serialData):

serialData += "\n"

serialData\_encode = serialData.encode()

print(serialData)

ser = serial.Serial(arduinoSerialPort)

ser.baudrate = 9600

ser.write(serialData\_encode)

print(serialData\_encode)

time.sleep(1)

ser.close()

for i in range(10):

# data = "road1/{}/road2/{}".format(randint(4,15), randint(4,15))

data = "hello/{}".format(randint(0,1))

serialEvent(data)

**microprocessor\_codes/led\_strip\_traffic.cpp**

#include <FastLED.h>

#define NUM\_LEDS\_1 4

#define NUM\_LEDS\_2 4

#define TRAFFIC1\_DATA 3

#define TRAFFIC2\_DATA 4

CRGB traffic1[NUM\_LEDS\_1];

CRGB traffic2[NUM\_LEDS\_2];

String data;

String dataPart1;

String dataPart2;

unsigned long currentTime = millis();

unsigned long greenTimeGreater;

unsigned long greenTimeLess;

long timeConserved1 = 0;

long timeUsing1 = 0;

long timeConserved2 = 0;

long timeUsing2 = 0;

long startTime = 0;

long elapsedTime = currentTime - startTime;

long cycleTime = 23000;

bool updateTime = true;

bool stringComplete = false;

void setup()

{

FastLED.addLeds<NEOPIXEL, TRAFFIC1\_DATA>(traffic1, NUM\_LEDS\_1);

FastLED.addLeds<NEOPIXEL, TRAFFIC2\_DATA>(traffic2, NUM\_LEDS\_2);

Serial.begin(9600);

}

void loop()

{

if(stringComplete)

{

dataPart1 = getValue(data, '/', 1);

dataPart2 = getValue(data, '/', 3);

timeConserved1 = dataPart1.toInt();

timeConserved2 = dataPart2.toInt();

if(updateTime == true)

{

timeUsing1 = timeConserved1;

timeUsing2 = timeConserved2;

updateTime = false;

}

data = "";

stringComplete = false;

}

Serial.println(timeUsing1);

Serial.println(timeUsing2);

FastLED.show();

greenTimeGreater = timeUsing1 \* 400;

greenTimeLess = timeUsing2 \* 400;

currentTime = millis();

elapsedTime = currentTime - startTime;

if(timeUsing1 >= timeUsing2)

{

if(timeUsing1 <= 20)

{

if(elapsedTime < 3000)

{

traffic1[1] = CRGB::Yellow;

traffic2[1] = CRGB::Yellow;

}

else

{

traffic1[1] = CRGB::Black;

traffic2[1] = CRGB::Black;

}

if (elapsedTime > 3000 && elapsedTime < 10000)

{

traffic1[0] = CRGB::Green;

traffic2[2] = CRGB::Red;

}

else

{

traffic1[0] = CRGB::Black;

traffic2[2] = CRGB::Black;

}

if (elapsedTime > 10000 && elapsedTime < 13000)

{

traffic1[1] = CRGB::Yellow;

traffic2[1] = CRGB::Yellow;

}

else

{

if(elapsedTime < 3000)

{

traffic1[1] = CRGB::Yellow;

traffic2[1] = CRGB::Yellow;

}

else

{

traffic1[1] = CRGB::Black;

traffic2[1] = CRGB::Black;

}

}

if (elapsedTime > 13000)

{

traffic1[2] = CRGB::Red;

traffic2[0] = CRGB::Green;

}

else

{

traffic1[2] = CRGB::Black;

traffic2[0] = CRGB::Black;

}

if (elapsedTime > cycleTime)

{

startTime = currentTime;

timeUsing1 = timeConserved1;

timeUsing2 = timeConserved2;

}

}

else

{

if (elapsedTime < 3000)

{

traffic1[1] = CRGB::Yellow;

traffic2[1] = CRGB::Yellow;

}

else

{

traffic1[1] = CRGB::Black;

traffic2[1] = CRGB::Black;

}

if (elapsedTime > 3000 && elapsedTime < (greenTimeGreater + 3000))

{

traffic1[0] = CRGB::Green;

traffic2[2] = CRGB::Red;

}

else

{

traffic1[0] = CRGB::Black;

traffic2[2] = CRGB::Black;

}

if (elapsedTime > (greenTimeGreater + 3000) && elapsedTime < (greenTimeGreater + 6000))

{

traffic1[1] = CRGB::Yellow;

traffic2[1] = CRGB::Yellow;

}

else

{

if(elapsedTime < 3000)

{

traffic1[1] = CRGB::Yellow;

traffic2[1] = CRGB::Yellow;

}

else

{

traffic1[1] = CRGB::Black;

traffic2[1] = CRGB::Black;

}

}

if (elapsedTime > (greenTimeGreater + 6000))

{

traffic1[2] = CRGB::Red;

traffic2[0] = CRGB::Green;

}

else

{

traffic1[2] = CRGB::Black;

traffic2[0] = CRGB::Black;

}

if (elapsedTime > (greenTimeGreater + 16000))

{

startTime = currentTime;

timeUsing1 = timeConserved1;

timeUsing2 = timeConserved2;

}

}

}

else

{

if(timeUsing2 <= 20)

{

if (elapsedTime < 3000)

{

traffic1[1] = CRGB::Yellow;

traffic2[1] = CRGB::Yellow;

}

else

{

traffic1[1] = CRGB::Black;

traffic2[1] = CRGB::Black;

}

if (elapsedTime > 3000 && elapsedTime < 10000)

{

traffic1[0] = CRGB::Red;

traffic2[2] = CRGB::Green;

}

else

{

traffic1[0] = CRGB::Black;

traffic2[2] = CRGB::Black;

}

if (elapsedTime > 10000 && elapsedTime < 13000)

{

traffic1[1] = CRGB::Yellow;

traffic2[1] = CRGB::Yellow;

}

else

{

if(elapsedTime < 3000)

{

traffic1[1] = CRGB::Yellow;

traffic2[1] = CRGB::Yellow;

}

else

{

traffic1[1] = CRGB::Black;

traffic2[1] = CRGB::Black;

}

}

if (elapsedTime > 13000)

{

traffic1[2] = CRGB::Green;

traffic2[0] = CRGB::Red;

}

else

{

traffic1[2] = CRGB::Black;

traffic2[0] = CRGB::Black;

}

if (elapsedTime > cycleTime)

{

startTime = currentTime;

timeUsing1 = timeConserved1;

timeUsing2 = timeConserved2;

}

}

else

{

if (elapsedTime < 3000)

{

traffic1[1] = CRGB::Yellow;

traffic2[1] = CRGB::Yellow;

}

else

{

traffic1[1] = CRGB::Black;

traffic2[1] = CRGB::Black;

}

if (elapsedTime > 3000 && elapsedTime < 13000)

{

traffic1[0] = CRGB::Red;

traffic2[2] = CRGB::Green;

}

else

{

traffic1[0] = CRGB::Black;

traffic2[2] = CRGB::Black;

}

if (elapsedTime > 13000 && elapsedTime < 16000)

{

traffic1[1] = CRGB::Yellow;

traffic2[1] = CRGB::Yellow;

}

else

{if(elapsedTime < 3000)

{

traffic1[1] = CRGB::Yellow;

traffic2[1] = CRGB::Yellow;

}

else

{

traffic1[1] = CRGB::Black;

traffic2[1] = CRGB::Black;

}

}

if (elapsedTime > 16000 && elapsedTime < (greenTimeLess + 16000))

{

traffic1[2] = CRGB::Green;

traffic2[0] = CRGB::Red;

}

else

{

traffic1[2] = CRGB::Black;

traffic2[0] = CRGB::Black;

}

if (elapsedTime > (greenTimeLess + 16000))

{

startTime = currentTime;

timeUsing1 = timeConserved1;

timeUsing2 = timeConserved2;

}

}

}

}

String getValue(String data, char separator, int index)

{

int found = 0;

int strIndex[] = {0, -1};

int maxIndex = data.length()-1;

for(int i=0; i<=maxIndex && found<=index; i++){

if(data.charAt(i)==separator || i==maxIndex){

found++;

strIndex[0] = strIndex[1]+1;

strIndex[1] = (i == maxIndex) ? i+1 : i;

}

}

return found>index ? data.substring(strIndex[0], strIndex[1]) : "";

}

void serialEvent() {

while (Serial.available()) {

char inChar = (char)Serial.read();

data += inChar;

if (inChar == '\n') {

stringComplete = true;

}

}

}