Nesnelerin İnterneti ve Uygulamaları

**Onur GÖNÜLLÜ - 1722130013**

1-)Mqtt protokolü aracılığı ile Sisteme Tanımlı Yüzler (Yüz Tanıma) İle Açılan Kapı Projesini çalıştıran kodu yazınız.

**Arduino Kodu**

#include <MQTT.h>

#include <MQTTClient.h>

#include <ESP8266WiFi.h>

#include <MQTTClient.h>

#define led1 D2

char ssid[] = "Redmi";

char pass[] = "onur1905";

const char\* broker = "onurgonullu.cloud.shiftr.io";

char mqttUserName[] = "onurgonullu";

char mqttPass[] = "12345";

WiFiClient net;

MQTTClient client;

unsigned long lastMillis = 0;

int alinan\_mesaj\_topic1;

int alinan\_aktivasyon;

void connect() {

Serial.print("\nconnecting to wifi.");

while (WiFi.status() != WL\_CONNECTED) {

Serial.print(".");

delay(5000);

}

char clientID[] ="OnurGonullu";

Serial.print("\nconnecting to broker...");

while (!client.connect(clientID,mqttUserName,mqttPass)) {

Serial.print(".");

delay(5000);

}

Serial.println("\nconnected!");

client.subscribe("kapi\_control");

}

void messageReceived(String &topic, String &payload) {

if (topic=="kapi\_control") {

alinan\_mesaj\_topic1 = (payload.toInt());

girdi\_oku();

}

}

void setup() {

Serial.begin(115200);

pinMode(led1, OUTPUT);

WiFi.begin(ssid, pass);

client.begin(broker, net); //broker , wifi

client.onMessage(messageReceived);

connect();

}

void loop() {

client.loop();

delay(10); // <- fixes some issues with WiFi stability

if (!client.connected())connect();

if (millis() - lastMillis > 3000) {

girdi\_oku();

lastMillis = millis();

}

}

void girdi\_oku() {

if (alinan\_mesaj\_topic1 == 1) {

digitalWrite(led1, HIGH);

Serial.println("kapi acildi");

}

else {

digitalWrite(led1, LOW);

Serial.println("kapi kapatildi");

}

}

**Python(Main) Kodları**

import cv2

from simple\_facerec import SimpleFacerec

from mqtt\_adapter import mqtt\_yaz

from excel\_adapter import excel\_yaz

# Encode faces from a folder

sfr = SimpleFacerec()

sfr.load\_encoding\_images("image\_databasefile/")

cap = cv2.VideoCapture(0)

while True:

ret, frame = cap.read()

# Detect Faces

face\_locations, face\_names = sfr.detect\_known\_faces(frame)

for face\_loc, name in zip(face\_locations, face\_names):

y1, x2, y2, x1 = face\_loc[0], face\_loc[1], face\_loc[2], face\_loc[3]

cv2.putText(frame, name,(x1, y1 - 10), cv2.FONT\_HERSHEY\_DUPLEX, 1, (0, 0, 200), 2)

cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 200), 4)

mqtt\_yaz(name) #mqtt burada cagiriyoruz

excel\_yaz(name)

cv2.imshow("Yuz Tanima", frame)

key = cv2.waitKey(1) #bir tuşa basılırsa

if key == 27: # ve bu tuş esc ise

break

cap.release()

cv2.destroyAllWindows()

**Python Mqtt Kodları**

import paho.mqtt.client as mqttclient

import time

def on\_connect(client,usedata,flags,rc):

if rc==0:

print("client is connected")

global connected

connected=True

else:

print("connection failed")

#mqtt broker codes:

connected=False

mqtt\_port = 1883

mqtt\_broker = "onurgonullu.cloud.shiftr.io"

mqtt\_username = "onurgonullu"

mqtt\_password = "12345"

client = mqttclient.Client("MQTT")

client.username\_pw\_set(mqtt\_username,password=mqtt\_password)

client.on\_connect=on\_connect

client.connect(mqtt\_broker,port=mqtt\_port)

client.loop\_start()

while connected!=True:

time.sleep(0.2)

#mqqt kodları burada bitti.

def mqtt\_yaz(name): #name değiskeni mainden gönderilir

if (name == "unknown"):

client.publish("kapi\_control", 0)

client.publish("isim\_control", name)

client.loop\_stop()

else:

client.publish("kapi\_control", 1)

client.publish("isim\_control", name)

time.sleep(2)

client.publish("kapi\_control", 0)

client.publish("isim\_control", "unknown")

#client.publish("kapi\_control", "ac")

client.loop\_stop()

**Python SimpleFacerec Klasörü**

import face\_recognition

import cv2

import os

import glob

import numpy as np

class SimpleFacerec:

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

self.known\_face\_encodings = []

self.known\_face\_names = []

# Resize frame for a faster speed

self.frame\_resizing = 0.25

def load\_encoding\_images(self, images\_path):

"""

Load encoding images from path

:param images\_path:

:return:

"""

# Load Images

images\_path = glob.glob(os.path.join(images\_path, "\*.\*"))

print("{} encoding images found.".format(len(images\_path)))

# Store image encoding and names

for img\_path in images\_path:

img = cv2.imread(img\_path)

rgb\_img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

# Get the filename only from the initial file path.

basename = os.path.basename(img\_path)

(filename, ext) = os.path.splitext(basename)

# Get encoding

img\_encoding = face\_recognition.face\_encodings(rgb\_img)[0]

# Store file name and file encoding

self.known\_face\_encodings.append(img\_encoding)

self.known\_face\_names.append(filename)

print("Encoding images loaded")

def detect\_known\_faces(self, frame):

small\_frame = cv2.resize(frame, (0, 0), fx=self.frame\_resizing, fy=self.frame\_resizing)

# Find all the faces and face encodings in the current frame of video

# Convert the image from BGR color (which OpenCV uses) to RGB color (which face\_recognition uses)

rgb\_small\_frame = cv2.cvtColor(small\_frame, cv2.COLOR\_BGR2RGB)

face\_locations = face\_recognition.face\_locations(rgb\_small\_frame)

face\_encodings = face\_recognition.face\_encodings(rgb\_small\_frame, face\_locations)

face\_names = []

for face\_encoding in face\_encodings:

# See if the face is a match for the known face(s)

matches = face\_recognition.compare\_faces(self.known\_face\_encodings, face\_encoding)

name = "unknown"

# # If a match was found in known\_face\_encodings, just use the first one.

# if True in matches:

# first\_match\_index = matches.index(True)

# name = known\_face\_names[first\_match\_index]

# Or instead, use the known face with the smallest distance to the new face

face\_distances = face\_recognition.face\_distance(self.known\_face\_encodings, face\_encoding)

best\_match\_index = np.argmin(face\_distances)

if matches[best\_match\_index]:

name = self.known\_face\_names[best\_match\_index]

face\_names.append(name)

# Convert to numpy array to adjust coordinates with frame resizing quickly

face\_locations = np.array(face\_locations)

face\_locations = face\_locations / self.frame\_resizing

return face\_locations.astype(int), face\_names