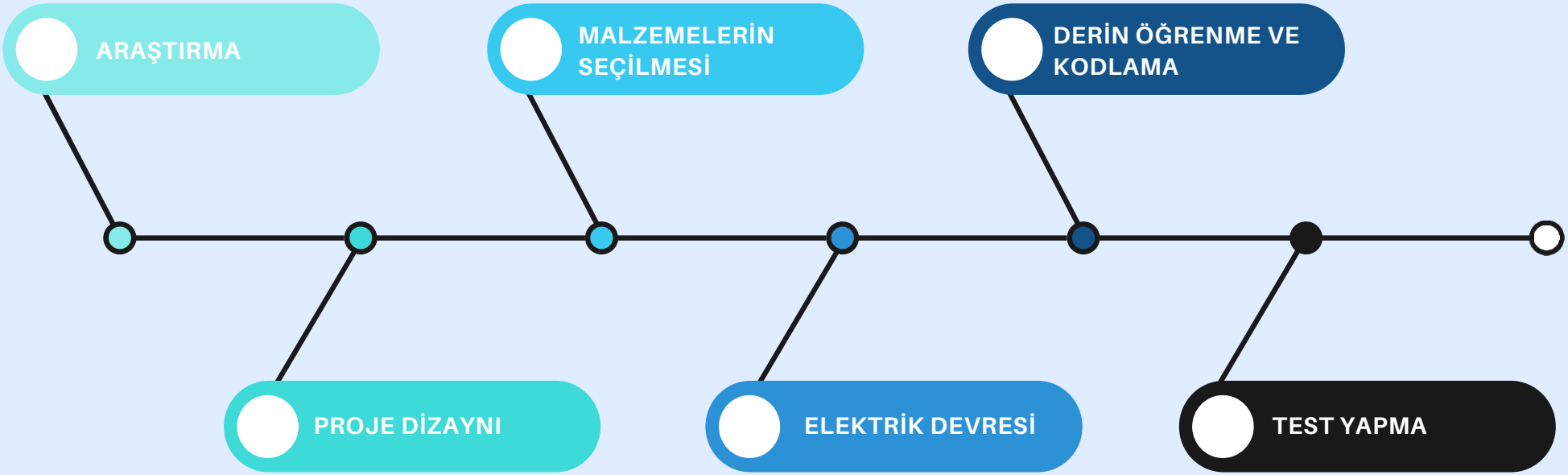
DERİN ÖĞRENME TABANLI GERÇEK ZAMANLI ATEŞ ÖLÇÜMÜ VE MASKE KONTROLÜ YAPAN OTOMATİK KAPI SİSTEMİNİN GELİŞTİRİLMESİ

FIRAT ÜNİVERSİTESİ | MÜHENDİSLİK FAKÜLTESİ | YAZILIM MÜHENDİSLİĞİ
YMÜ414 - BİTİRME PROJESİ
170290607 - AHMET MİRA

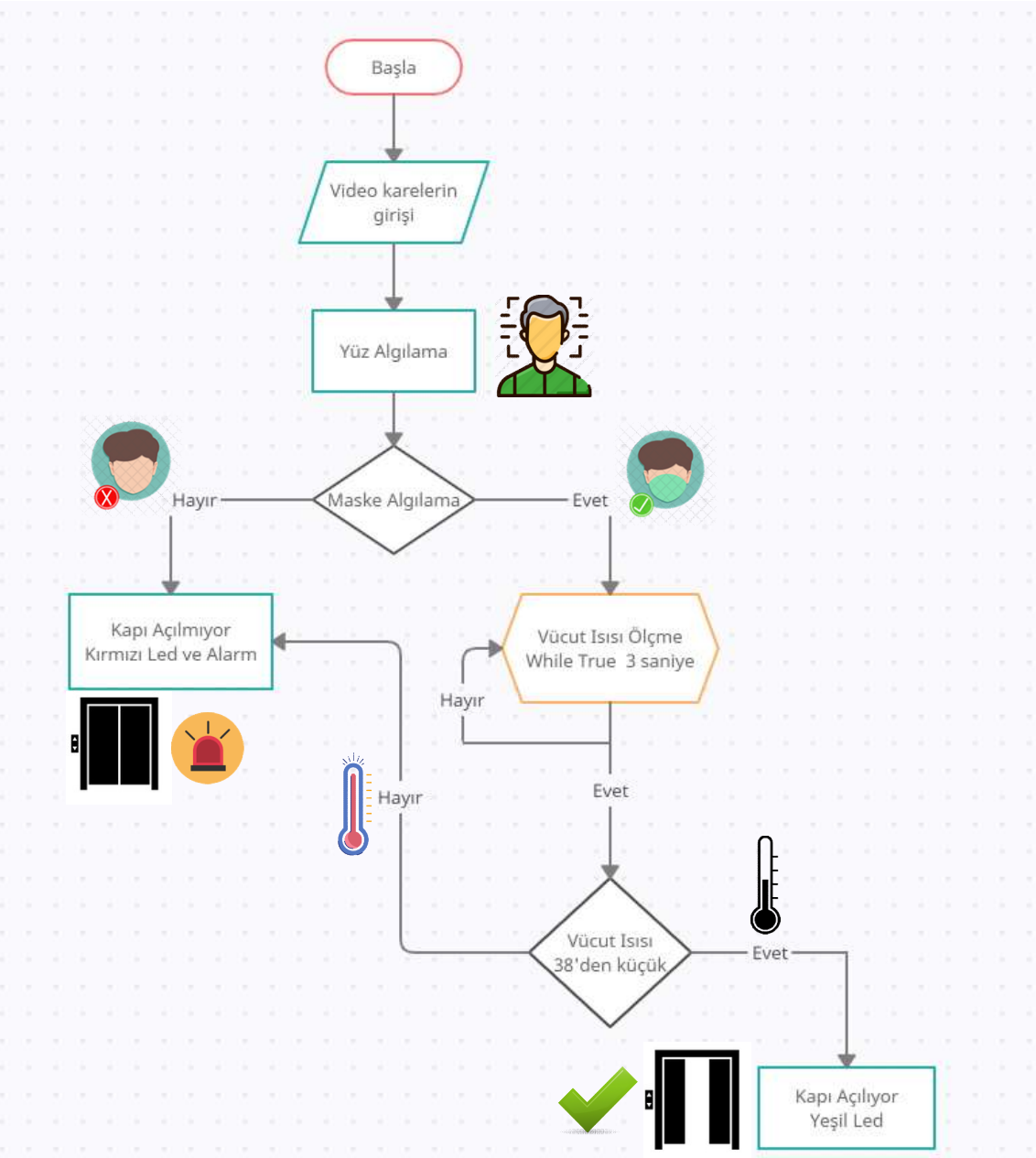
GİRİŞ

COVID-19 yayılması gerildikten sonra ve tüm dünya kontrollü normalleşme aşamasına girerken ancak bu normalleşme virüsün sona erdiği ve yayılmadığı anlamına gelmez. Yine de virüsün yayılmaması için Maske takma, Eldiven giyme ve Sosyal mesafeyi koruma gibi önlemleri alınmalı. Bu proje ile yüksek sıcaklıklara sahip insanların ve maske takmayanların tespiti yapılabilecek. Bu durumda otomatik kapıların açılması önlenebilecektir. Kullanımı sadece Covid-19 sürecinde ile sınırlı olmayıp buna benzer tüm salgın hastalıklar için geçerli olup sağlık kurumlarında da kalıcı olarak kullanabiliriz.

PROJE GERÇEKLEŞTIRME AŞAMALARI



PROJE DİZAYNI

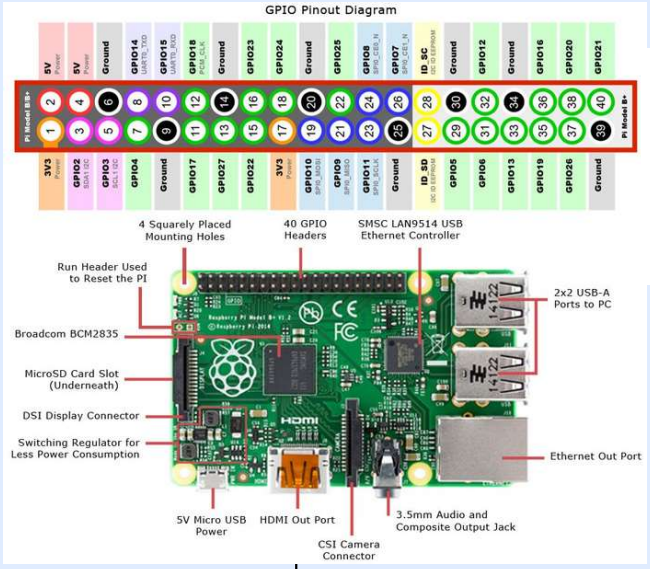


MALZEMELERİN SEÇİLMESİ

Arayüzü ve sonuçlarımızı göstermek için Raspberry Pi Ekran

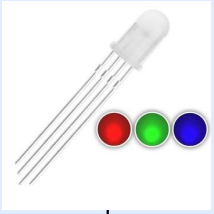
Yüz tanıma ve Maske algılamak için Raspberry Pi Kamera

ses yanıtı ve alarm verilmesi için Buzzer



Raspberry Pi Microcontroller

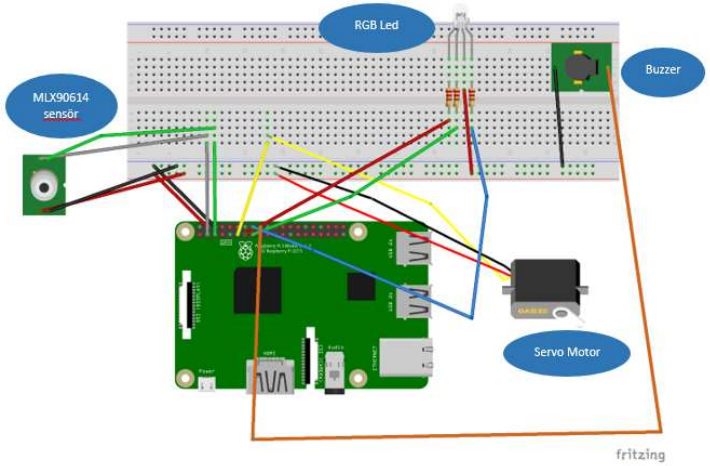
Vücut ısısı ölçmek için MLX90614 Sıcaklık Sensörü



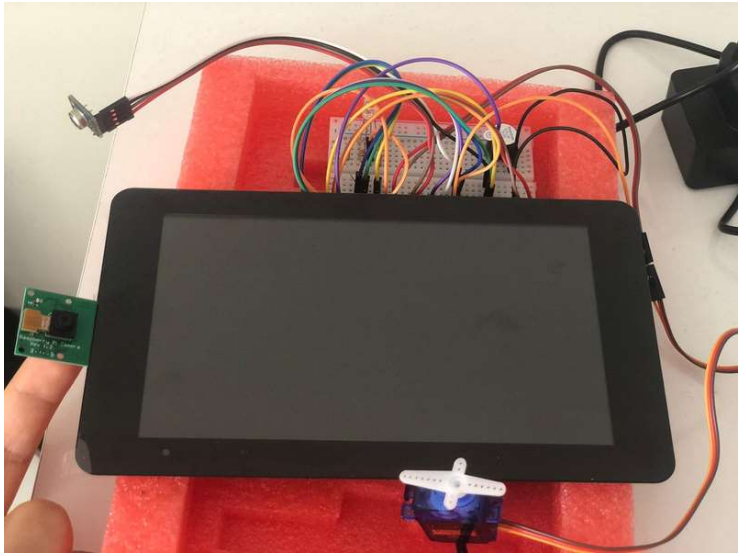
Projenin doğruluğu göstermek için RGB LED

Otomatik kapı yerine gösterim amaçlı olarak servo motor

ELEKTRİK DEVRESİ



Simülasyon programı üzerinde yapılan sanal elektrik devrisi



Gerçek Elektrik Devresi

Bağlamalar şu şekilde oluşturuldu:

- Servo Motor: GPIO-17, 5V ve Ground ile bağlıdır.
- RGB LED: GPIO27, GPIO22, 5V ve GPIO23 ile bağlıdır.
- BUZZER: GPIO24 ve Ground ile bağlıdır.
- MLX90614 Sıcaklık Sensörü: SDA, SCL, 5V ve Ground ile bağlıdır.
- Pi 7 dokunulabilir Ekran: SDA, SCL, 5V, Ground ve DSI portu ile bağlıdır.
- Pi Kamera: DSI portu ile bağlıdır.

DERİN ÖĞRENME VE KODLAMA

```python
#Gerekli Kütüphaneler
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import os
from google.colab import drive

#data çağırma
drive.mount('/content/drive')
base_dir = '/content/drive/MyDrive/data/'
# Öğrenme hızı, Epochs, batch size
INIT_LR = 1e-4
EPOCHS = 20
BS = 32
DIRECTORY = os.path.join(base_dir, 'test')
CATEGORIES = ["Masked", "No_Mask"]
```

```python
print("[Bilgi] Fotğraflar yükleniyor...")
data = []
labels = []
for category in CATEGORIES:
path = os.path.join(DIRECTORY, category)
for img in os.listdir(path):
img_path = os.path.join(path, img)
image = load_img(img_path, target_size=(224, 224))
image = img_to_array(image)
image = preprocess_input(image)
data.append(image)
labels.append(category)

# perform one-hot encoding on the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)
data = np.array(data, dtype="float32")
labels = np.array(labels)
(trainX, testX, trainY, testY) = train_test_split(data, labels,
test_size=0.20, stratify=labels, random_state=42)

# veri büyütmek için eğitim görüntü oluşturucusunu oluşturma
aug = ImageDataGenerator(
rotation_range=20,
zoom_range=0.15,
width_shift_range=0.2,
height_shift_range=0.2,
shear_range=0.15,
horizontal_flip=True,
fill_mode="nearest")
```

```python
# MobileNetV2 ağını yükleme, "FC katman setleri"
baseModel = MobileNetV2(weights="imagenet", include_top=False,
input_tensor=Input(shape=(224, 224, 3)))

#temel modelin üstüne yerleştirilecek modelin başını inşa etme
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)

#Model, eğiteceğimiz gerçek model olacaktır
model = Model(inputs=baseModel.input, outputs=headModel)

#temel modeldeki tüm katmanlar üzerinde döngü oluştuema ve ilk
for layer in baseModel.layers:
layer.trainable = False

# Modeli derleme
print("[bilgi] model derleniyor...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
metrics=["accuracy"])

# Modeli eğitmek
print("[bilgi] model eğitiliyor...")
H = model.fit(
aug.flow(trainX, trainY, batch_size=BS),
steps_per_epoch=len(trainX) // BS,
validation_data=(testX, testY),
validation_steps=len(testX) // BS,
epochs=EPOCHS)
```

```python
#Test
print("[bilgi] deeğerlendirme...")
predIdxs = model.predict(testX, batch_size=BS)
predIdxs = np.argmax(predIdxs, axis=1)

# Rapor
print(classification_report(testY.argmax(axis=1), predIdxs,
target_names=lb.classes_))

# Eğitimiz modeli kaydetme
print("[bilgi] maske_algila modeli kaydediliyor...")
model.save("maske_algila.model", save_format="h5")

# Eğitim kaybını ve doğruluğunu çizme
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig("plot.png")
```

Projeyi iki bağımsız bölüme ayrılmakta, böylece her bölüm kendi adımlarına sahip olacak:

1- Training (Eğitme): Öncellikle veri setimizi yükledik bu veri seti üzerinde Keras Kütüphanesi ve MobileNetV2 sinir ağ mimarisini kullanılarak modelimiz eğittik ve iki sınıfa sınıflandırdık. (Maskeli / Maskesiz yüz), sınıflandırdıktan ve eğittikten sonra eğitilmiş modeli hafızaya kaydederiz.

DERİN ÖĞRENME VE KODLAMA

```python
import cv2
import os
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from gpiozero import RGBLED, Buzzer, Servo
from colorzero import Color
import numpy as np
import board
import busio as io
import adafruit_mlx90614
import time

buzzer = Buzzer(24)
led = RGBLED(22, 23, 27)
servo = Servo(17)


def sensor():
    i2c = io.I2C(board.SCL, board.SDA, frequency=100000)
    mlx = adafruit_mlx90614.MLX90614(i2c)
    while True:
        ambientTemp = "{:.2f}".format(mlx.ambient_temperature)
        targetTemp = "{:.2f}".format(mlx.object_temperature)

        print("Ambient Temperature:", ambientTemp, "°C")
        print("Target Temperature:", targetTemp, "°C")
        time.sleep(3)
        return targetTemp
```

```python
video_capture = cv2.VideoCapture(0)
while True:
    # Capture frame-by-frame
    ret, frame = video_capture.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = faceCascade.detectMultiScale(gray,
                                         scaleFactor=1.1,
                                         minNeighbors=5,
                                         minSize=(60, 60),
                                         flags=cv2.CASCADE_SCALE_IMAGE)
    faces_list = []
    preds = []
    for (x, y, w, h) in faces:
        face_frame = frame[y:y + h, x:x + w]
        face_frame = cv2.cvtColor(face_frame, cv2.COLOR_BGR2RGB)
        face_frame = cv2.resize(face_frame, (224, 224))
        face_frame = img_to_array(face_frame)
        face_frame = np.expand_dims(face_frame, axis=0)
        face_frame = preprocess_input(face_frame)
        faces_list.append(face_frame)
    if len(faces_list) > 0:
        preds = model.predict(faces_list)
```

```python
    for pred in preds:
        (mask, withoutMask) = pred
        if mask > withoutMask:
            label = "Maske takili"
            color = (0, 255, 0)
            a = float(sensor())
            if (a <= 38):
                buzzer.off()
                led.color = Color('green')
                servo.max()
            else:
                label = "Atesiniz yuksek"
                color = (0, 0, 255)
                buzzer.on()
                led.color = Color('red')
                servo.min()
        else:
            label = "Maskeyi takiniz"
            color = (0, 0, 255)
            buzzer.on()
            led.color = Color('red')
            servo.min()

        label = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)
        cv2.putText(frame, label, (x, y - 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)

        cv2.rectangle(frame, (x, y), (x + w, y + h), color, 2)

    cv2.imshow('Video', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
video_capture.release()
cv2.destroyAllWindows()
```

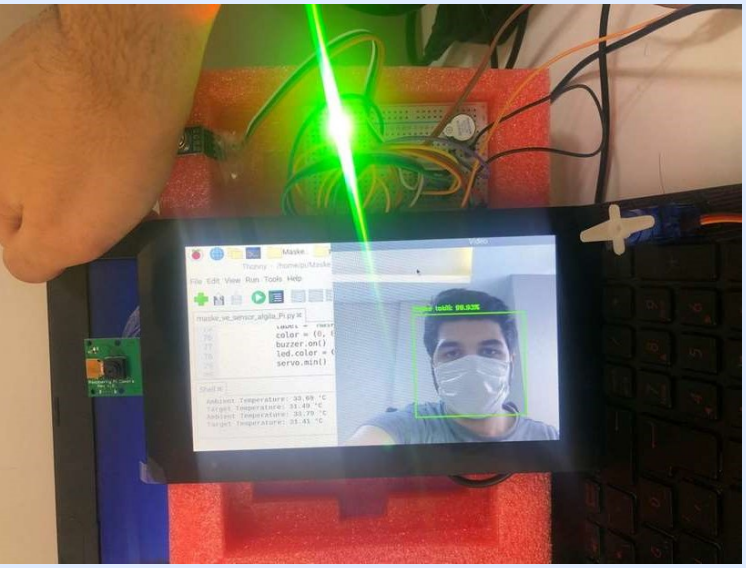
2- Deployment (Dağıtım): Bu aşamada, Raspberry Pi Kamera aracılığıyla canlı bir akıştan gerçek zamanlı Haar Cascade Modelini Kullanarak Yüz Algılanıyor sonra da eğittiğimiz maske algılama modeline dayanarak bir kişinin maske takıp takmadığını tespit etmekte.

Maske tespit edildiğinde Vücut Isısı almak için sensor metodunu çağırılıyor ve vücut ısısını kontrol ediliyor.

TEST YAPMA

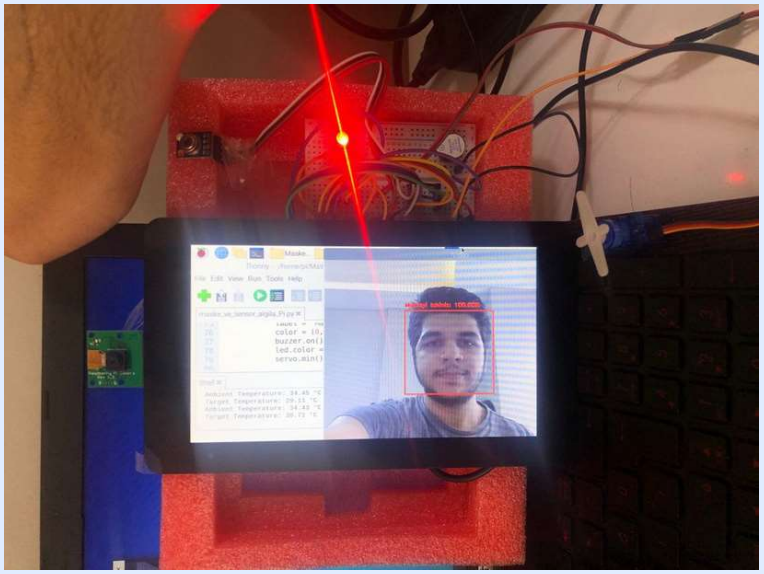
1.Durum

Maske takılı ve Vücut Isısı 38'den küçüktür.



2.Durum

Maske takılı değil ve Vücut Isısı 38'den küçüktür.



3.Durum

Maske takılı ve Vücut Isısı 38'den büyüktür.