RV COLLEGE OF ENGINEERING®
BENGALURU – 560059
(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

"Smart Covid 19 Door using RPi"

Experiential Learning REPORT
Fundamentals of Computer System Design (18CS35)
III SEMESTER

2021-2022

Submitted by

Malavika Hariprasad	**1RV20CS080**
Neha N	**1RV20CS094**
Nimisha Dey	**1RV20CS098**
Pratiksha Narasimha Nayak G	**1RV20CS123**

Under the Guidance of

Badarinath K B,
Associate Professor
Department of CSE, RVCE,
Bengaluru - 560059

CERTIFICATE

Certified that the **Experiential Learning** work titled "Smart Covid 19 Door using RPi" has been carried out by **Malavika Hariprasad (1RV20CS080), Neha N (1RV20CS094), Nimisha Dey (1RV20CS098), Pratiksha Narasimha Nayak G (1RV20CS123),** bonafide students of RV College of Engineering, Bengaluru, have submitted in partial fulfillment for the **Assessment of Course: Fundamentals of Computer System Design (18CS35) – Experiential Learning** during the year 2021-2022. It is certified that all corrections/suggestions indicated for the internal assessment have been incorporated in the report.

Faculty Incharge
Department of CSE,
RVCE., Bengaluru –59

Head of Department
Department of CSE,
RVCE, Bengaluru–59

ACKNOWLEDGEMENT

It gives us immense pleasure to be associated with this project titled “Smart Covid ”. The project was a joyous learning process that helped us in linking our theoretical knowledge to practical knowledge. We got to learn a lot from this project and gain in depth knowledge about the implementation and practical application of Discrete Mathematics.

Firstly, we would like to express our sincere gratitude to our teacher Prof. Badarinath K B as well as our Principal Dr. K N Subramanya who gave us the golden opportunity to do this wonderful project. I extend my heartfelt thanks to our teacher who has helped us in this endeavor and has always been very cooperative in providing us with the necessary information regarding the project.

Secondly, we would like to thank our family and friends for providing us with unfailing support and continuous encouragement through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Last but not the least, we would like to thank all those who had helped us (directly or indirectly) towards the completion of this project within the limited time frame. It was a great learning experience for us.

Table of Contents

1. Abstract

With the spread of COVID-19 and the emergence of different variants, recent guidelines have advised avoiding crowded places. The use of face masks and sanitizers are essential to prevent the further spread of COVID. When the rate of infections are high, public places like malls and offices carry out manual screening of temperature and sanitizer to increase safety. However, this produces a negative impact since it only leads to long lines and increased risk. Even the guards tasked with these jobs are at high risk of getting the virus. Hence, to solve this problem, we propose an automatic door system which carries out face mask detection using a Raspberry Pi 3B+.

In this project, we have incorporated a face-mask detection module, a MLX90614 temperature sensor, a passive IR sensor with motor, to detect the motion of the person when he/she is allowed to walk by the door and open the door; and a hand-sanitisation module. We were able to assemble these sensors with a Raspberry PI B+ Model. We have integrated this with a cloud platform by name ThingSpeak, wherein, we were able to send the temperature data from the model to the cloud.

After the compilation of the model, we were able to get a high accuracy of face-mask detection using CNN-based machine learning model called MobileNetV2. The other functionalities of the smart door were also tested by simulating different scenarios, such as the person not wearing a mask, the person covering their face, the person wearing a mask but having high temperature and the person wearing a mask and having normal body temperature. We plotted a graph of the temperature recorded by the sensor with respect to time in ThingSpeak. The main goal of this project was to integrate the pre-existing sensors with an Rpi microcontroller in order to automate the previously mentioned process so as to minimize human intervention in these processes, as far as possible, so as to reduce the risk of spreading the infection.

The developed model is an approach to improvise the health monitoring systems in various commercial and medical units. The prototype has scope for the future with the inclusion of additional features to it such as measuring the blood pressure, weight, oxygen levels in the blood, etc, especially in hospitals. This model was also developed by keeping in mind the affordability of the units.

This project demonstrates how smart systems can be successfully designed, with the integration of microcontroller and sensors, driven by artificial intelligence and machine learning technologies. The developement of the said smart door would help in minimizing the risk of infection during the pandemic, thereby benefiting the society, and especially specific groups, at large.

2. Introduction

2.1 Need of the project

As new variants of COVID 19 have been looming, it is important to keep our health and safety in mind. Although many countries have begun relaxing rules related to masks and temperature screening, medical organizations still advise wearing masks to public places. In this scenario, it is cumbersome and risky to have security personnel who perform temperature checks and mask checks for visitors in a public place. Automation of this process could ensure smooth facilitation of people in public places and help in avoiding large crowds which further put people at risk of getting infected.

2.2 Project Description

In this respect, we aim to develop a smart door system which can carry out artificial intelligence based face mask detection, automated temperature screening, door opening and hand sanitizer dispensing. Further, data collected from the people at the door is sent to a cloud network which can later be used to study the trends in rising or coming down of cases and average temperatures of people in a region. The main goal of this project was to integrate the pre-existing sensors with an Rpi microcontroller in order to automate the previously mentioned process so as to minimize human intervention in these processes, as far as possible, so as to reduce the risk of spreading the infection. A user-friendly visual interface has been developed using the tkinter library and the outputs of the system can be displayed on an lcd screen or monitor. It is a standalone system that can be setup anywhere and is easy to use.

3. Literature Survey

[1] This paper covers the development of a smart door system using Arduino UNO which has the features of face mask detection, automatic door opening, sanitization and digital and audio outputs. This paper defined the framework and requirements of a smart door system. However, a separate system is required to run the mask detection code.

[2] This paper talks about the importance of early detection of COVID-19 and the main risk factors of the virus. A smart door system which aids in detection of these factors has been implemented. The system has been implemented using Arduino UNO. It collects body temperature and sends the collected data to a cloud network using an API called Obloq mqtt.

[3] This paper focuses on the design and implementation of an artificial intelligence based face mask detection system using Raspberry Pi which detects if a person is wearing a mask and sends an alert via a mobile app. The system lets the door open only if a mask is detected. No other data is collected in this project.

[4] This paper implements a face mask detection system. This project makes use of OpenCV, Caffe-based face detector, Keras, TensorFlow and MobileNetV2 for the detection of face masks on humans. It examines the method of designing a code to detect if a person is wearing a mask and draw a bounding box over the face of the person.

[5] This paper primarily focuses on building a simple Smart Door for the COVID 19 pandemic. The key features of this project are measuring the temperature of the person, hand sanitisation and gate control module. An add-on in this project was the availability of a switch, which on pressing could open the door immediately in case of emergencies. The proposed system comprises an Arduino UNO, MLX90614 Contactless Temperature Sensor, PIR Sensor, IR Sensor, and a Servo Motor.

[6] This paper has also proposed a simple Smart Door for COVID 19 pandemic. The proposed system comprises a MLX90614 temperature sensor, a proximity sensor, a 0V7670 camera module, servo motor and a LCD display; all integrated with Arduino UNO. However, an additional feature incorporated in this project is that when the temperature of the person is greater than the prescribed one, the camera module captures an image of the ID/Aadhar card of the person and sends an alarm email to the designated IDs.

4. Block Diagram

The proposed system architecture is depicted in Fig 1. It diagrammatically explains the integration of the various modules that are used to develop the smart door.

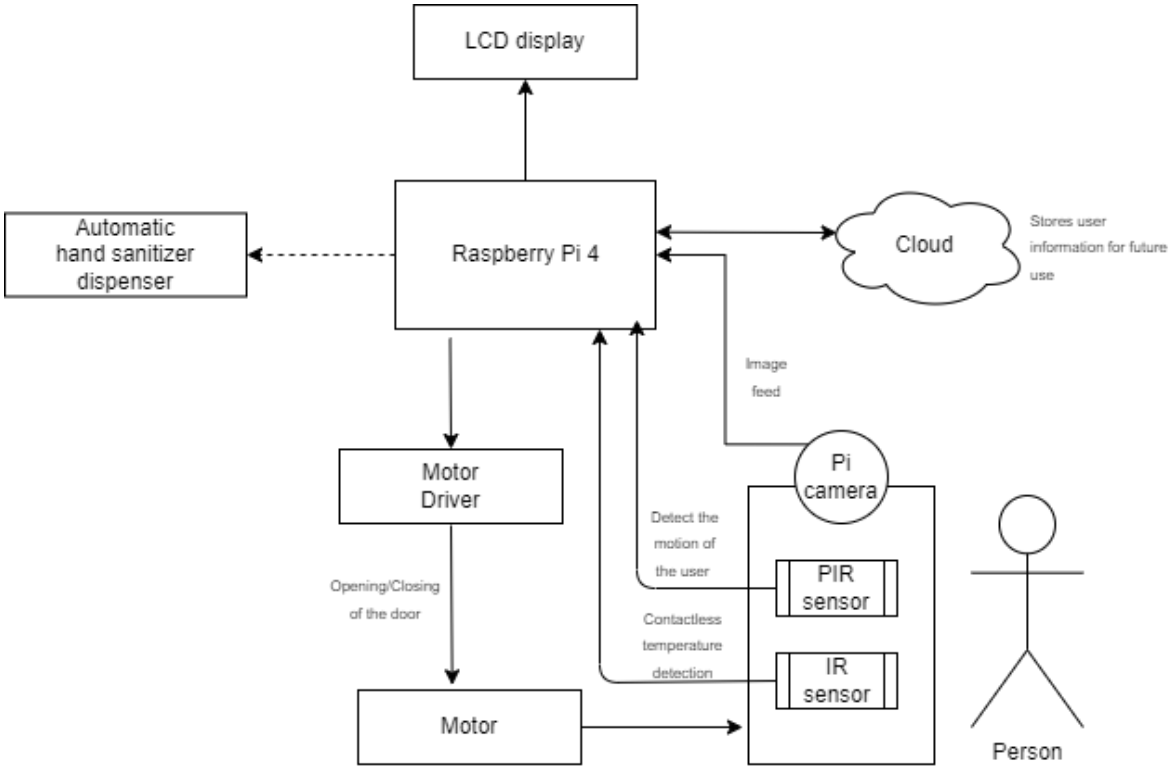

Fig 1: Architecture

The methodology and the complete working of the smart door is as follows.
The image obtained from the Pi camera is sent to R Pi to be processed using the OpenCV image processing library installed in Raspberry Pi. The region of interest is obtained and fed to the pre-trained model which is trained using datasets and saved. The lcd screen is used to show the video stream from the R Pi Camera. If the mask is not detected, the red box highlights the face on the LCD, and Tkinter is used to display appropriate messages, and the door remains closed. If the mask is detected, a green box is used to highlight the face, and an appropriate message is displayed using Tkinter. The infrared temperature sensor MLX90614 measures the temperature of the person standing in front of the door. If the temperature is above 37C the door still remains closed and an appropriate message is displayed using Tkinter. If the temperature is below 37 C the door opens and an appropriate message is displayed using Tkinter. The door remains open until motion is detected by the PIR sensor, else the door closes. The ultrasonic sensor is used to detect distance of hand from the sanitiser bottle and gives appropriate prompts using Tkinter. The servo motor is used to dispense sanitiser. Temperature data is sent to the ThingSpeak cloud and a graph of the data is plotted.The methodology is depicted using a flowchart in Fig.2

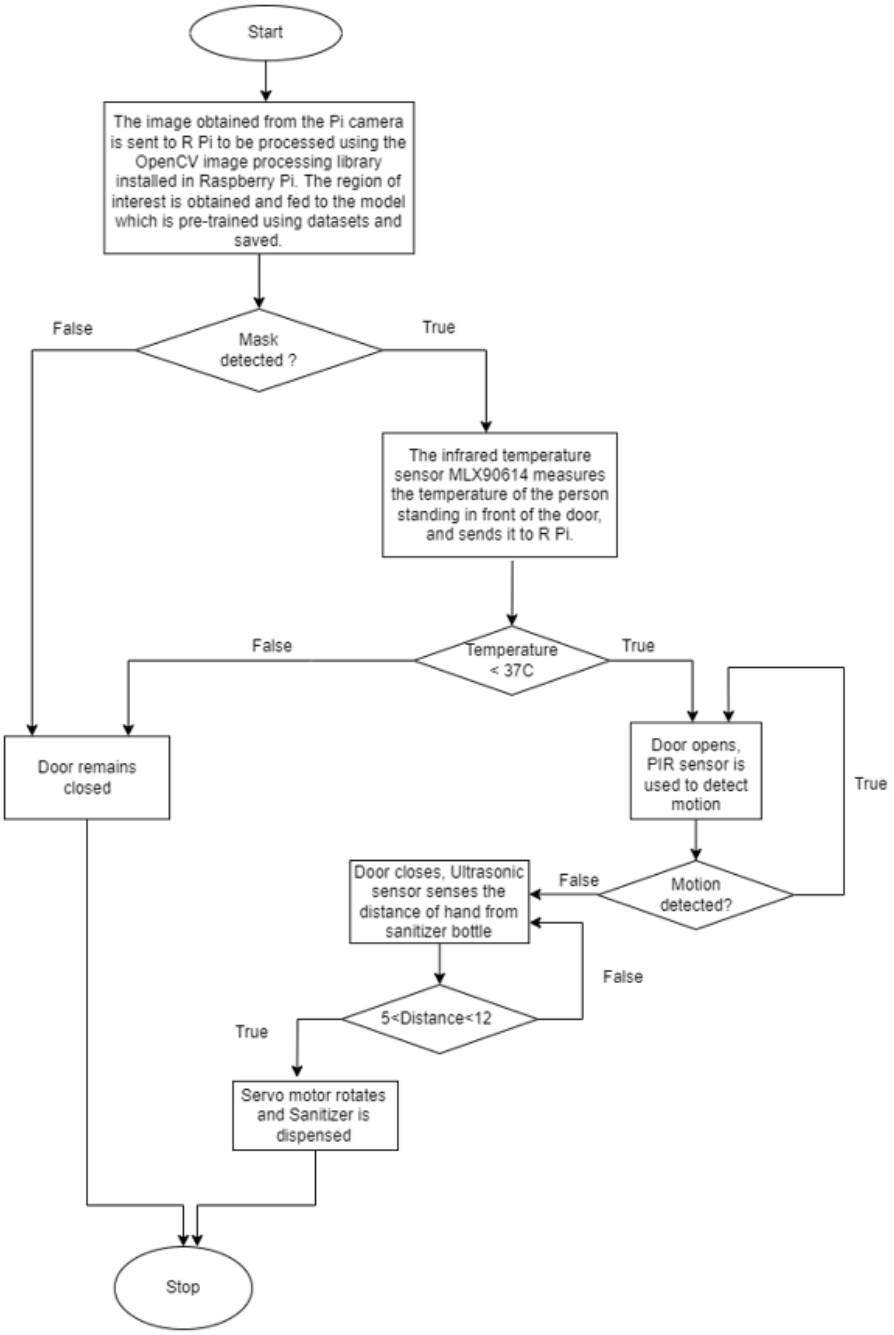

Fig 2: Methodology flowchart

5. Methodology/System Design/Implementation

5.1. Description of components

1. Raspberry Pi

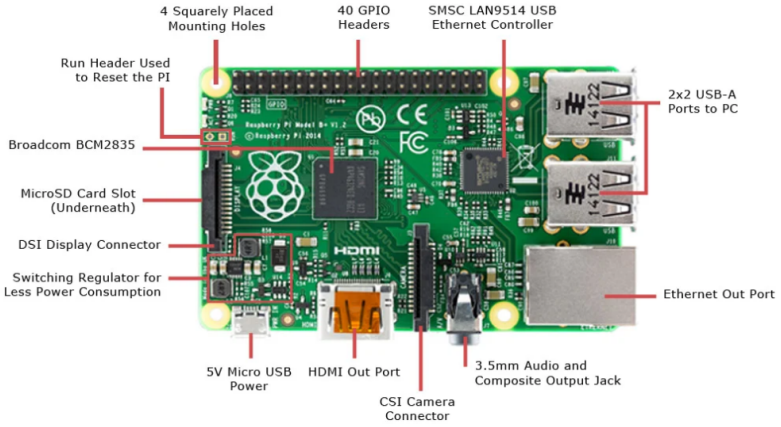

Fig 3: Raspberry Pi 3B+

Board specification

- Quad-Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
- 100 Base Ethernet
- 4 USB 2 ports
- 4 Pole 3.3mm stereo output and composite video port
- Full-size HDMI CSI (Camera Serial Interface) camera port for connecting a camera
- DSI (Display Serial Interface) display port for connecting a touchscreen display
- Micro SD port
- Micro USB power port (up to 2.5A)
- 40-pin extended GPIO

Fig 4: Pin details and configuration

Pin details and configuration

The raspberry pi 4 consists of 12 power pins - two 5V pins, two 3V3 pins, and 8 ground pins (0V).

- 5V: The 5v pin outputs the 5 volts coming from the USB Type-C port.
- 3.3V: The 3v pin is used to provide a stable 3.3v supply to external components.
- GND: The ground pin is commonly referred to as GND.

The raspberry pi 4 has 28 GPIO pins (GPIO 0 - GPIO 27)

A pin that can be set as an input or output and is controlled in run time is called a GPIO pin. A GPIO pin set as input allows the signal transmitted by any external device to be received by the Raspberry Pi. Input voltage between 1.8V and 3.3V is read as HIGH by the Raspberry pi. And when the input voltage is lower than 1.8V, it is read as LOW. A GPIO pin set as output delivers HIGH/3.3V or LOW/0V.

Apart from Input/Output, the GPIO pins can also perform a variety of other functions like. Some of these functions/pins are:

- PWM pins
- SPI pins
- I2C pins
- UART pins

* PWM pins

- PWM stands for “Pulse Width Modulation”. It means that an analog value is being modulated on a digital signal.
- Software PWM is available on all pins.
- Hardware PWM is available on these pins only: GPIO12, GPIO13, GPIO18, GPIO19

*SPI pins

- SPI (Serial Peripheral Interface) is a type of serial communication protocol. It is used by the Raspberry Pi for master-slave communication to quickly communicate between one or more peripheral devices.
- 5 pins are required for SPI communication:

    - GND: Connect the GND pin from all the slave components and the Raspberry Pi 4 board together.

- SCLK: Clock for SPI communication.

- MOSI: It stands for Master Out Slave In. This pin is used to send data from the master to a slave.

- MISO: It stands for Master In Slave Out. This pin is used to receive data from a slave to the master.

- CE: It stands for Chip Enable. We need to connect one CE pin per slave (or peripheral devices) in our circuit.

*I2C pin

- I2C pins on the Raspberry Pi board are used to communicate with peripheral devices that are compatible with Inter-Integrated Circuit (a low-speed two-wire serial communication protocol).
- This serial communication protocol requires master-slave roles between both, the board and the peripheral devices.
- I2C protocol requires two connections: SDA (Serial Data) and SCL (Serial Clock). They work by transmitting data using the SDA connection, and the speed of data transfer is controlled via the SCLK pin.

*UART pins

- The UART (Universal Asynchronous Receiver / Transmitter) is an asynchronous protocol that provides a way to communicate between two microcontrollers or devices.
- TX pin transmits the serial data to the RX pin of another device and RX pin receives the serial data coming from TX pin of the other device.

2. LCD

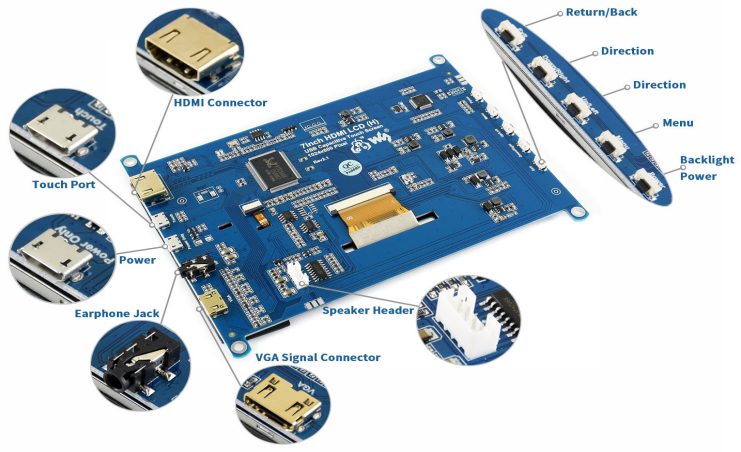

Fig 5: 7 inch HDMI LCD display

It is a 7inch, 1024x600, Capacitive Touch Screen LCD, having HDMI interface, supporting Multi mini-PCs and Multi Systems. The HDMI interface of the 7 inch LCD display is connected to the HDMI port of Rpi; and the USB cable is used to connect the LCD display with the Rpi.

3. Pi Camera

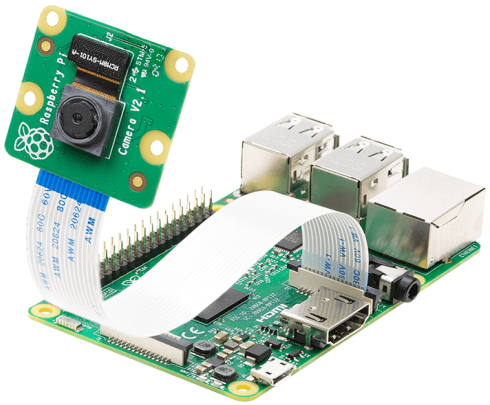

Fig 6: Rpi Pi camera module

Pi Camera module is a camera which can be used to take pictures and high definition video. Raspberry Pi Board has CSI (Camera Serial Interface) interface to which we can attach the PiCamera module directly. This Pi Camera module can attach to the Raspberry Pi's CSI port using a 15-pin ribbon cable.

4. Temperature sensor

Fig 7: MLX90614 Temperature sensor

We have incorporated a MLX90614 Temperature Sensor in our project to measure the temperature of the person. It is an infrared contactless temperature measurement device that has an operating voltage of about 3.6V to 5V. We know that every object above zero Kelvin emits heat in the range of the infrared spectrum. This incident infrared radiation is measured by the temperature sensor. The sensor is powered with 5V and we use the SCL pin to enable sending the measured temperature to the Rpi through the SDA pin. The sensor is capable of , measuring 2 types of temperature: 1) Ambient temperature i.e., temperature of its surroundings (environment), with a range from -70° C to 382.2°C. and; 2) Object temperature with a range from -40° C to 125°C. The Table 1 gives a description of the pin details of the sensor.

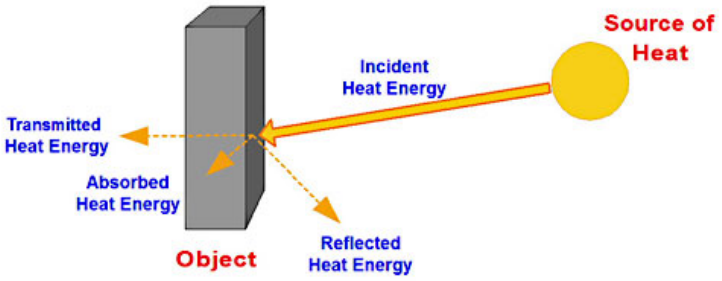

Fig 8: Temperature sensor principle

Table 1: Pin details of MLX90614 IR temperature sensor

Pin name	Description
VIN	Power to temperature sensor
GND	Ground pin
SCL	To control the rate of data transfer
SDA	To transfer the measured temperature

5. Motor driver

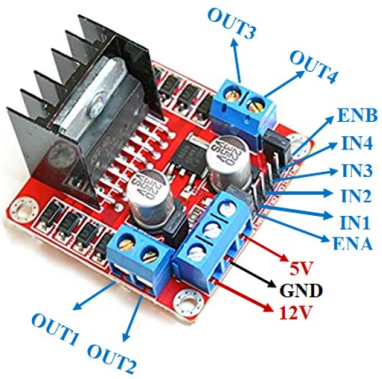

Fig 9: Motor driver

For this project, we have used an L298N motor driver. It is a dual H-Bridge motor driver which allows speed and direction control of two DC motors at the same time. The module can drive DC motors that have voltages between 5 and 35V, with a peak current up to 2A. It requires an input of 3V-40V. The pins of the motor driver are described in Table 2. The Enable A and Enable B pins are used for enabling and controlling the speed of the motor. If direct voltage is given to this pin, the motor rotates with a constant velocity. The speed of the motors can be varied by connecting a PWM pin to the enable pins. The motor has 4 input pins (2 for each motor) that can be used to control the direction of rotation of the motors. IN1 and IN2 correspond to motor1 and IN3 and IN4 correspond to motor2. If input 1 is LOW and input 2 is HIGH, the motor will move forward, if input 1 is HIGH

and input 2 is LOW the motor will move backward. In case both inputs are the same (i.e., either HIGH and HIGH or LOW and LOW), the motor will stop.

Table 2 : Pin details of Motor driver L298N

Pin Name	Description
IN1 & IN2	Motor A input pins. Used to control the spinning direction of Motor A
IN3 & IN4	Motor B input pins. Used to control the spinning direction of Motor B
ENA	Enables PWM signal for Motor A
ENB	Enables PWM signal for Motor B
OUT1 & OUT2	Output pins of Motor A
OUT3 & OUT4	Output pins of Motor B
12V	12V input from DC power Source
5V	Supplies power for the switching logic circuitry inside L298N IC
GND	Ground pin

6. PIR sensor

Fig 10: PIR sensor

We have integrated a HC-SR501 PIR Motion Sensor Detector Module to detect the motion of the person passing by, so as to automate the opening and closing of the door. It works on a principle similar to the temperature sensor i.e., all objects with a temperature above Absolute Zero emit heat energy in the form of infrared radiation, including human bodies. The hotter an object is, the more radiation it emits. PIR sensor is specially designed to detect such levels of infrared radiation. It has two rectangular slots in it made of a material that allows the infrared radiation to pass. Behind these, are two separate infrared sensor electrodes. If one half sees more or less IR radiation than the other, the output will swing high or low. A description of the pin details is given in Table 3.

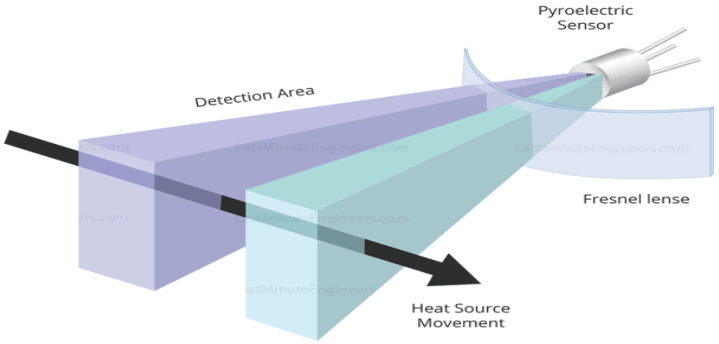

Fig 11: Working of a passive infrared motion sensor (PIR)

Table 3: Pin details of the PIR sensor:

Pin name	Description
VCC	Power to the PIR sensor
OUT	HIGH or LOW; sends a HIGH signal if motion is detected; else, sends a LOW signal
GND	Ground pin

7. Ultrasonic sensor

Fig 12: Ultrasonic sensor

An ultrasonic sensor is an instrument that measures the distance to an object using ultrasonic sound waves. In this project, we have used it to detect the presence of the hand

so that the sanitizer can be dispensed without wasting any of it. The pin details of the ultrasonic sensor are as shown in Table 4. The VCC pin provides power to the ultrasonic sensor and is connected to the 3V3 pin of the Raspberry Pi. The TRIG pin is used to trigger the ultrasonic pulses and the ECHO pin produces a pulse when the reflected signal is received. The length of the pulse is proportional to the time it took for the transmitted signal to be detected. It works on the same principle as RADAR, i.e. sound wave reflection. The radio-frequency (rf) energy is transmitted to and reflected from the reflecting object. The time taken for this is recorded and distance is calculated using the formula:

Distance= (time * speed of sound)/2

While connecting the ultrasonic sensor to the RPi, The resistors are used to reduce the output of the ECHO pin from 5V to 3.3V since the GPIO pins of the RPi can handle only a maximum of 3.3V. Here, the principle of a voltage divider circuit is followed. Connections are to be made according to Fig. 4. Vin is ECHO, and Vout is connected to the RPi as it has been reduced to 3.3V.

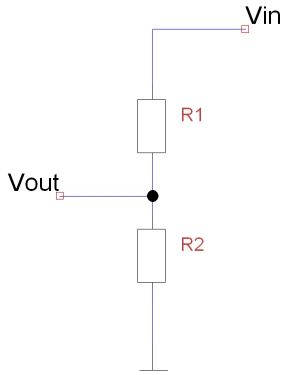

Fig 13: Voltage divider circuit

Table 4: Pin details of ultrasonic sensor

Pin Name	Description
VCC	Power to ultrasonic sensor
GND	Ground pin
TRIG pin	Input pin
ECHO pin	Output pin

8. Servo motor

Servo is a general term for a closed loop control system. A closed loop control system uses the feedback signal to adjust the speed and direction of the motor to achieve the

desired result. RC servo motors work on the same principle. It contains a small DC motor connected to the output shaft through the gears. The output shaft drives a servo arm and is also connected to a potentiometer (pot). The potentiometer provides position feedback to the servo control unit where the current position of the motor is compared to the target position. According to the error, the control unit corrects the actual position of the motor so that it matches the target position. The pin details are shown in table 5.

Table 5: Pin details of servo motor

Pin Name	Description
VCC	Power to servo motor
GND	Ground pin
Control	Input pin

5.2. Circuit Diagram

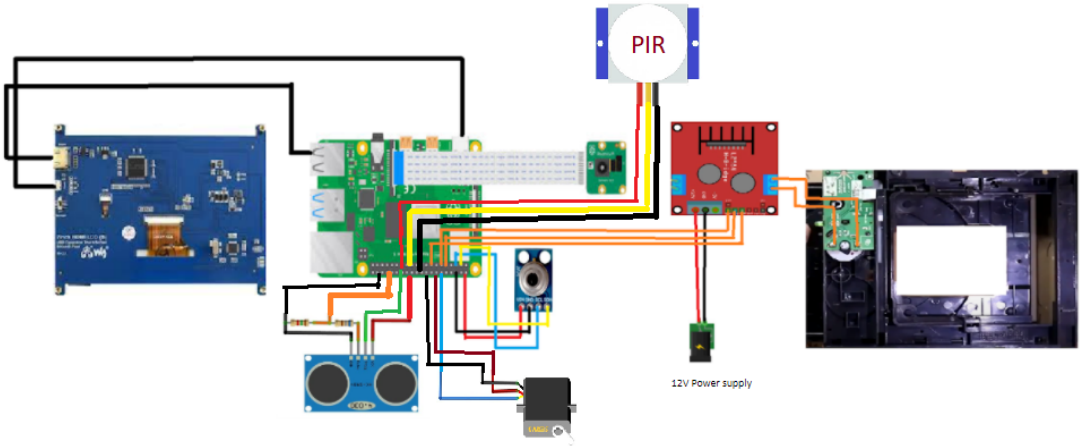

Fig 14: Circuit diagram

The circuit diagram of our project is as shown in Fig 14. Note that the following description of connections with the Rpi are given with respect to its BOARD pin numbers.

- The Rpi camera module is attached to the CSI port using a 15-pin ribbon cable.
- The SDA (data transfer) pin of the temperature sensor is connected to Pin 3 of the Rpi. Its SCL (rate of transfer) pin is connected to Pin 5 of the Rpi.
- The 12V DC source to the motor driver assembly could be provided with a 9V or 12V battery. The inputs IN1 and IN2, as shown in Fig 9, are connected to the Pins 36 and 38 respectively. These pins are set to HIGH or LOW depending on the direction in which the gate has to be opened. The ENA Pin of the motor drive, connected to Pin 40 of Rpi, is used to enable the corresponding motor. The output pins of the motor driver assembly are connected to the terminals of the motor of the CD driver, which we have used as the door assembly.
- For the PIR sensor, the OUT (to detect motion) Pin is connected to Pin 10 of the

Rpi. This pin is used in output mode i.e., the sensor sends a HIGH signal through this pin to the Rpi if motion is detected; else, it sends a LOW signal.

- The TRIG and ECHO Pins of the ultrasonic sensor, used in the hand sanitization module, are connected to Pins 16 and 18 of Rpi respectively. 1KΩ and 2KΩ resistors are used to supply the required voltage (about 3.3V) to the ultrasonic sensor.
- For the servo motor, for the hand sanitisation module, its control Pin is connected to the Pin 15 of the Rpi. The duty cycle of the servo motor is changed as and when the gate has to be opened or closed.
- The HDMI interface of the 7 inch LCD display is connected to the HDMI port of Rpi; and the USB cable is used to connect the LCD display with the Rpi.

The temperature sensor and the PIR sensor are powered using Pins 2 and 4 of the Rpi that supply 5V. The ultrasonic sensor is also powered with 5V from the Rpi, but it is reduced to about 3.3V using the resistors. The servo motor is powered with 3.3V, obtained from Pin 1 of the Rpi.

5.3. Software Code

5.3.1 Code for FaceMask Detection

This section explains the code written to perform the face mask detection. It examines the entire process of loading the data, creating the model and finally fitting the model on our dataset and saving the trained model as “MaskDetector.h5” so that it can be used later by the main program to test if any user has a mask. For this model, tensorflow libraries are used. TensorFlow is a commonly used software library for ML and AI projects. It focuses on training and inference of deep neural networks. In this paper, it has been used to train the model using a CNN- based algorithm known as MobileNetV2. This model has been used since it is much faster than the standard CNN and fast detection is essential to our project. As seen in Fig 15, the necessary libraries needed for pre-processing, training and testing are first imported.

```
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
```

Fig 15: Importing the necessary libraries

The dataset used to train the model, consisting of images of people with and without masks have been downloaded in which there are 2 folders- one of images of people with

masks, and the other without. As seen in Fig 16, each image in the sub folders are obtained and loaded into a list called “data”. Before loading the images, they are converted into arrays and resized into the standard (224,224) size which is the input shape of the model. The corresponding label (Mask or No_Mask) is obtained and stored in a list called “label”. Since the model requires numerical input, the string labels are One Hot Encoded and converted into binary form. In this method, if a mask is detected the label is of the form [1 0] and if no mask is detected it is of the form [0 1]. Both lists are converted into numpy arrays to follow the format needed for the deep learning model. FInally, the obtained dataset is ready to be split into train and test. In this code, we have taken 80% of the model to train and only 20% to test.

```
INIT_LR = 1e-4
EPOCHS = 20
BS = 32
print("[INFO] loading images...")
#imagePaths = list(paths.list_images(args["dataset"]))
imagePaths = r"/home/pi/PyMLX90614-0.0.4/Face-Mask_detection/dataset"
data = []
labels = []
for directory in os.listdir(imagePaths):
        label = os.path.join(imagePaths, directory)
        if not os.path.isdir(label):
                continue
        for item in os.listdir(label):
                if item.startswith("."):
                    continue
                image = load_img(os.path.join(label, item), target_size=(224, 224))
                image = img_to_array(image)
                image = preprocess_input(image)
                data.append(image)
                labels.append(label)
data = np.array(data, dtype="float32")
labels = np.array(labels)
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)
(trainX, testX, trainY, testY) = train_test_split(data, labels,
test_size=0.20, stratify=labels, random_state=42)
```

Fig 16: Loading images and splitting dataset

Since there are a limited number of images in the dataset, it is passed to an ImageDataGenerator which makes slight changes to the images and enlarges the dataset. After this, the model is defined. The base model (MobileNetV2) is defined. The head model is then defined which consists of a pooling layer which converts the images from size (224,224) to (7,7). The data then passes through a Flatten layer and a Dense layer with 128 neurons. Further, a Dropout layer is added to avoid overfitting. The final model takes the input as the input of the base model and output as the output of the head model. The model is optimized using the Adam optimizer, which is among the preferred optimizers. After this, the model is compiled and the metrics chosen is accuracy. The model is then fitted on the training data and passed for 20 epochs. This is depicted in Fig 17.

```
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")
baseModel = MobileNetV2(weights="imagenet", include_top=False,
input_tensor=Input(shape=(224, 224, 3)))
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 = Model(inputs=baseModel.input, outputs=headModel)

for layer in baseModel.layers:
layer.trainable = False

print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
metrics=["accuracy"])
```

Fig 17: Defining the model

Before saving, it was tested on the test dataset and a classification report can be obtained to evaluate and improve its performance. Once satisfactory results are obtained, this final, trained model is then saved and can further be used to classify the images obtained from the RPi camera module as shown in Fig 18.

```
print("[INFO] training head...")
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)

print("[INFO] saving mask detector model...")
model.save("MaskDetector.h5")
```

Fig 18: Fitting and saving the model

5.3.2 Code for Mask Detection using RPi video

Fig.19 shows the main function wherein the face detector model is loaded. faceNet is the file for face detection. The OpenCV DNN module is used to detect the face. The model explained in the previous section is then loaded in maskNet. VideoStream is used to load the camera and capture the frames.

```
if __name__=="__main__":

    prototxtPath = "/home/pi/PyMLX90614-0.0.4/Face-Mask_detection/face_detector/deploy.prototxt"
    weightsPath = "/home/pi/PyMLX90614-0.0.4/Face-Mask_detection/face_detector/res10_300x300_ssd_iter_140000.caffemodel"
    faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
    print("[INFO] Loading the model");

    maskNet = load_model("/home/pi/PyMLX90614-0.0.4/Face-Mask_detection/MaskDetector.h5")

    print("[INFO] starting video stream...")
    vs = VideoStream(src=0, framerate=30).start()

    run_video(detect_and_predict_mask)
```

Fig 19: Main function

Then the run_video() function is called which is used to loop over the frames obtained from the video stream as shown in Fig 20. Then each frame from the video stream is resized to have a maximum width of 1000 pixels. The faces in the frames are then detected using detect_and_predict_mask() function which takes in frame, faceNet and maskNet and is then determined if wearing a mask or not. Then detect_mask() function is

called to loop over the obtained face locations and their corresponding locations. To show the output frame imshow() function under cv2 is used. Then applyLogic() function which takes in the label as the parameter is called.

```
def run_video(detect_and_predict_mask):
    while True:

        frame = vs.read()
        frame = imutils.resize(frame, width=1000)

        (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)

        label = detect_mask(locs, preds, frame)

        cv2.imshow("Frame", frame)
        key = cv2.waitKey(1) & 0xFF

        applyLogic(label)

        if key == ord("q"):
            break

    cv2.destroyAllWindows()
    pwm.stop()
    GPIO.cleanup()
    vs.stop()
```

Fig 20: run_video() function

In the detect_and_predict_mask() function as shown in Fig 21, the dimensions of the frame are determined and the blogfromImage function is used to preprocess the images and a blog is created. A blog is potentially a collection of images with same spatial dimensions i.e same width and height, same depth(number of channels), and also preprocessed in the same manner. The blog is then passed through the network and the face detections are obtained. Then for each detection confidence i.e, probability associated with each detection is extracted. Weak detections are filtered out by ensuring that the confidence is greater than minimum confidence. Then the x and y coordinates of the bounding box are computed. It is then ensured that the bounding boxes fall within the dimensions of the frame. Then some modifications are done and then face and bounding boxes are appended to their respective lists. The predictions are made only when a face is detected. This function returns face locations and their respective predictions.

```
def detect_and_predict_mask(frame, faceNet, maskNet):
        sleep(2)
        (h, w) = frame.shape[:2]
        blob = cv2.dnn.blobFromImage(frame, 1.0, (224, 224),
                (104.0, 177.0, 123.0))

        faceNet.setInput(blob)
        detections = faceNet.forward()
        print(detections.shape)

        faces = []
        locs = []
        preds = []

        for i in range(0, detections.shape[2]):

                confidence = detections[0, 0, i, 2]

                if confidence > 0.5:

                        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                        (startX, startY, endX, endY) = box.astype("int")

                        (startX, startY) = (max(0, startX), max(0, startY))
                        (endX, endY) = (min(w - 1, endX), min(h - 1, endY))

                        face = frame[startY:endY, startX:endX]
                        face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
                        face = cv2.resize(face, (224, 224))
                        face = img_to_array(face)
                        face = preprocess_input(face)

                        faces.append(face)
                        locs.append((startX, startY, endX, endY))

        if len(faces) > 0:
                faces = np.array(faces, dtype="float32")
                preds = maskNet.predict(faces, batch_size=32)

        return (locs, preds)
```

Fig 21: detect_and_predict_mask() function

In the detect_mask() function as shown in Fig 22, for each bounding box the label is printed including the probability. Then temperature sensor data is collected and is printed along with the bounding box. The color of the box is green if the mask is detected, else it is red in color. It then returns the label.

```
def detect_mask(locs, preds, frame):
    for (box, pred) in zip(locs, preds):
        (startX, startY, endX, endY) = box
        (mask, withoutMask) = pred

        label = "Mask" if mask > withoutMask else "No Mask"

        color = (0, 255, 0) if label == "Mask" else (0, 0, 255)

        print(label)

        label_out = "{}: {:.2f}%".format(label, max(mask, withoutMask) * 100)

        temp = getTempData()
        person_temp = "Temp: {:.1f}".format(temp)

        cv2.putText(frame, label_out, (startX, startY - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
        cv2.putText(frame, person_temp, (endX-10, endY), cv2.FONT_HERSHEY_SIMPLEX, 0.45, (255, 0, 0), 2)
        cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)

        return label
```

Fig 22: detect_mask() function

In the applyLogic() function, logic for opening and closing of the gate is coded as shown in Fig 23. When the label is equal to “No mask” then “No Mask. Cannot open gate” message is displayed on the LCD display on tkinter GUI. Otherwise when a mask is detected, then “Mask detected” is displayed on the screen. Then temperature sensor data is collected as shown in Fig 25 and sent to the ThingSpeak cloud platform as shown in Fig 31. If the temperature is greater than 37 then “High temperature, Gate closed.” is displayed and the gate does not open and when the temperature is within acceptable range then “Temperature is within acceptable range. Gate opening” is displayed as shown in fig 24. Then the opengate() function is called which is used to turn the motor using the GPIO pins as shown in Fig 26 and the gate opens. Then sanitize_hand() function is called which is for the automatic hand sanitization when the temperature is within the range. PIR sensor data is collected to detect motion. If motion is detected then “Motion detected. Gate is still open” is displayed on the screen, else “Gate is closing” is displayed and then closeGate() function is called as defined in Fig 28.

```
def applyLogic(label):
    if (label=="No Mask"):
        msg="No Mask. Cannot open gate"
        master.after(2000,message(msg))
        master.after(2000,delete())
    else:
        msg="Mask detected"
        master.after(2000,message(msg))
        master.after(2000,delete())
        temp = getTempData()
        temp=int(sensor.get_obj_temp())
        params = urllib.parse.urlencode({'field1': temp, 'key':key })
        headers = {"Content-typZZe": "application/x-www-form-urlencoded","Accept": "text/plain"}
        conn = hp.HTTPConnection("api.thingspeak.com:80")
        try:
            conn.request("POST", "/update", params, headers)
            response = conn.getresponse()
            print(temp)
            print(response.status, response.reason)
            data = response.read()
        except:
            print("connection failed")
        task=threading.Thread(target=temp)
        var=tk.StringVar()
        master.update()
```

Fig 23: applyLogic() function

```
if temp >= 37:
    print(temp," High temp")
    msg=" is your temperature. High temperature, Gate closed."
    master.after(2000,message(msg))
    master.after(2000,message_temp(temp))
    master.after(2000,delete())
    master.after(3000,lambda:msg.delete(0,END))
    sleep(1)
else:
    print(temp," is within acceptable range. Gate opening")
    #count+=1
    msg=" is your temperature. Temperature is within acceptable range. Gate opening"
    master.after(2000,message(msg))
    master.after(2000,message_temp(temp))
    master.after(2000,delete())
    openGate()
    flag=0
    while(flag==0):
        flag=sanitize_hand()
    while(GPIO.input(ir)):
        print(temp,"  Motion detected. Gate is still open")
    else:
        print(temp," Motion not detected. Gate was open for 2 secs")
        print("Gate is closing")
        msg="Gate closing"
        master.after(2000,message(msg))
        master.after(2000,delete())
        master.after(3000,lambda:msg.delete(0,END))
        closeGate()
```

Fig 24: Opening and closing of gate based on temperature

```
def getTempData():
    temp = sensor.get_obj_temp()
    return temp
```

Fig 25: getTempData() to collect temperature sensor data

```
def openGate():
        print ("Turning motor on")
        GPIO.output(Motor1A,GPIO.LOW)
        GPIO.output(Motor1B,GPIO.HIGH)
        GPIO.output(Motor1E,GPIO.HIGH)

        sleep(7)
        GPIO.output(Motor1A,GPIO.LOW)
        GPIO.output(Motor1B,GPIO.LOW)
        GPIO.output(Motor1E,GPIO.LOW)
```

Fig 26: openGate() function

The hand sanitization module consists of the ultrasonic sensor and the servo motor. The ultrasonic sensor detects the presence of hands of the person; and if detected, the servo motor is to be activated so as to push down the sanitizer spray for the liquid to come out. The code for the same is shown in Fig 27. The TRIG Pin of the ultrasonic sensor is first set HIGH so as to trigger ultrasonic pulses for about 10μs. Then, it is set to LOW. The ECHO pin detects if an obstacle is present or not. So, until the ECHO Pin becomes 1 (HIGH - obstacle (here, hands) detected), the timestamp (from being LOW to HIGH) is calculated. This timestamp gives the amount of time required for the ultrasonic sound waves to travel from the sensor, to the obstacle, and back to the sensor. This would be used to calculate the distance between the hands of the person and the sensor. If the calculated distance is less than 12cm and greater than 5cm, we would start the servo motor to push down the sanitizer spray. A change in duty cycle represents the change in the angle of the servo motor head. If the distance is greater than 12cm, then a message saying “Please keep your hand near the sanitizer” is displayed.

```
def sanitize_hand():
    flag1=0
    GPIO.output(trig,True)
    time.sleep(0.00001)
    GPIO.output(trig,False)
    while GPIO.input(echo)==0:
        pulse_start=time.time()
    while GPIO.input(echo)==1:
        pulse_end=time.time()
    pulse_duration=pulse_end-pulse_start
    distance=pulse_duration*17150
    distance=round(distance+1.15,2)
    if(distance<12 and distance>5):
        print("Distance= ", distance, "cm. Dispensing sanitizer")
        pwm.start(2.5)
        pwm.ChangeDutyCycle(5)
        time.sleep(0.5)
        pwm.ChangeDutyCycle(7.5)
        time.sleep(0.5)
        pwm.ChangeDutyCycle(10)
        time.sleep(0.5)
        pwm.ChangeDutyCycle(12.5)
        time.sleep(0.5)
        pwm.ChangeDutyCycle(10)
        time.sleep(0.5)
        pwm.ChangeDutyCycle(7.5)
        time.sleep(0.5)
        pwm.ChangeDutyCycle(5)
        time.sleep(0.5)
        pwm.ChangeDutyCycle(2.5)
        time.sleep(0.5)
        pwm.stop()
        flag1=1
    else:
        print("Please keep your hand near the sanitizer")
        msg="Please keep your hand near the sanitizer"
        master.after(2000,message(msg))
        master.after(2000,delete())
        master.after(3000,lambda:msg.delete(0,END))
        sleep(1)
    return flag1
```

Fig 27: sanitize_hand() function

```
def closeGate():
        print ("Closing door")
        GPIO.output(Motor1A,GPIO.HIGH)
        GPIO.output(Motor1B,GPIO.LOW)
        GPIO.output(Motor1E,GPIO.HIGH)

        sleep(2)
        GPIO.output(Motor1A,GPIO.LOW)
        GPIO.output(Motor1B,GPIO.LOW)
        GPIO.output(Motor1E,GPIO.LOW)
```

Fig 28: closeGate() function

6. Result snapshots and Conclusion

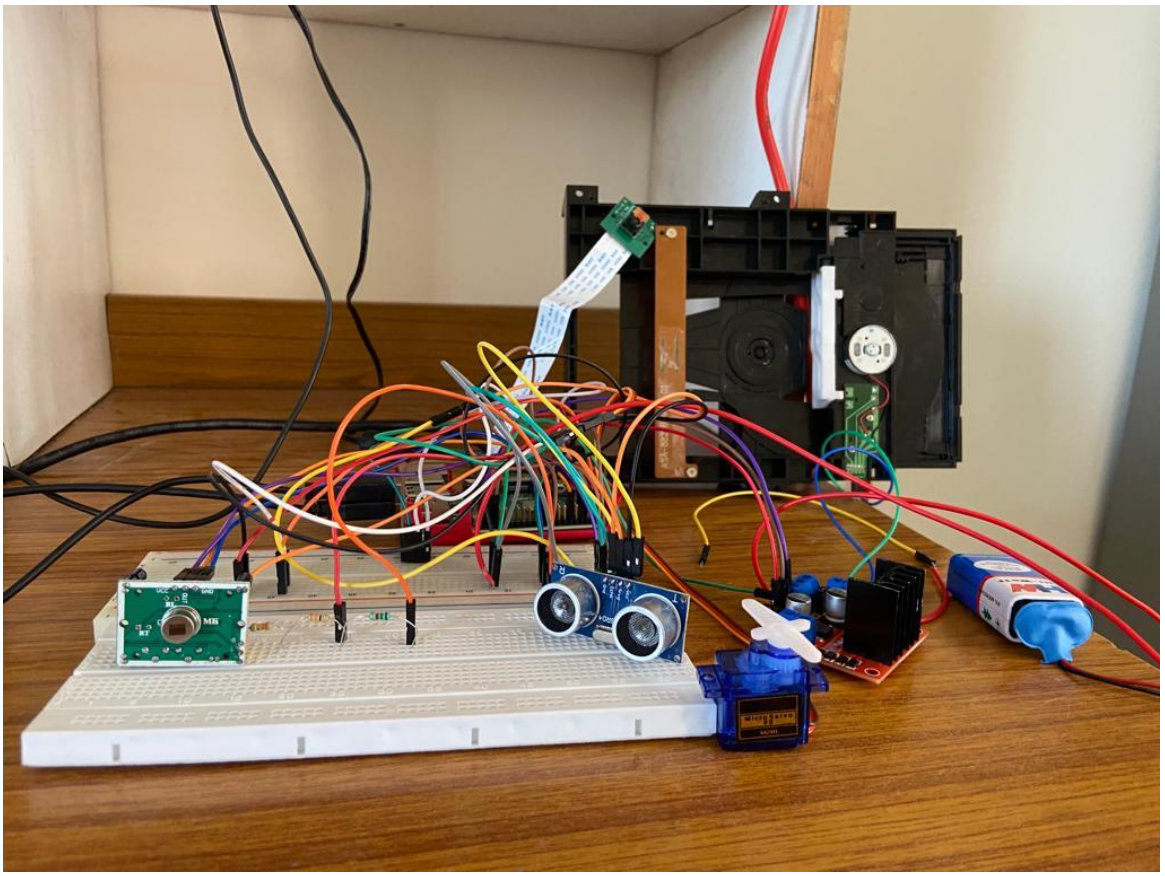

Fig 29: Final hardware setup

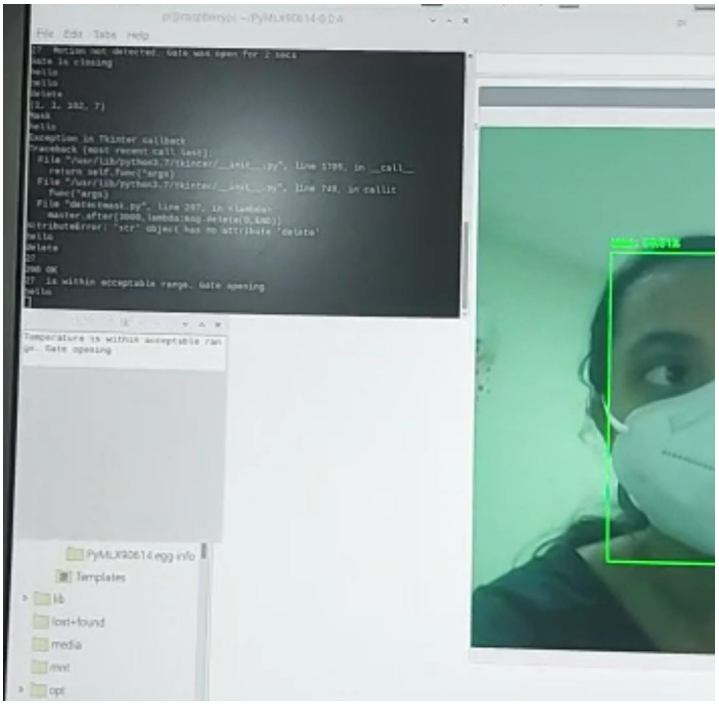

Fig 30: Display on the screen when mask is detected

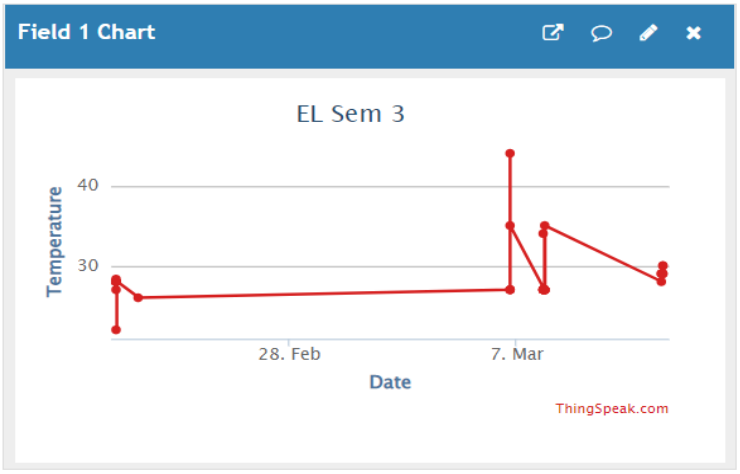

Fig 31: ThingSpeak graph of temperature received from the Rpi

The developed model of the Smart Covid Door using Rpi has been able to automate the tasks of face mask detection, temperature measurement, hand sanitization and door control. The model was also integrated with a cloud platform, ThingSpeak, to send the data - temperature of the person - from the Rpi. This prototype is an effort to improvise the safety of guards, who otherwise would be incharge of performing the above mentioned tasks. It is an approach to enhance the health care systems in hospitals. There are, however, scopes of future improvement in the model as mentioned below.

- Implementing the smart door in hospitals that can store the information like weight, BP, SPO2, of the people entering in the cloud using Azure IoT Hub which can connect to the RPi. The messages from and to the device can be monitored and managed using VS Code.
- Using facial recognition features to identify patients and fetch hospital records.

7. Bibliography

[1] C. Sathvik, S. Moorthy, K. Nidhi and K. Badari Nath, "SmartZ COVID-19 Smart Door," 2021 IEEE International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS), 2021, pp. 1-5, doi: 10.1109/CSITSS54238.2021.9683586.
[2] A. D. K. -T. Lam, K. Zhang and X. -Y. Zeng, "Design of COVID-19 Smart Door Detection Device for Risk Factor Detection," 2021 7th International Conference on Applied System Innovation (ICASI), 2021, pp. 130-133, doi: 10.1109/ICASI52993.2021.9568463.
[3] K. N. Baluprithviraj, K. R. Bharathi, S. Chendhuran and P. Lokeshwaran, "Artificial Intelligence based Smart Door with Face Mask Detection," 2021 International Conference on Artificial Intelligence and Smart Systems (ICAIS), 2021, pp. 543-548, doi: 10.1109/ICAIS50930.2021.9395807.
[4]H. Adusumalli, D. Kalyani, R. K. Sri, M. Pratapteja and P. V. R. D. P. Rao, "Face Mask Detection Using OpenCV," 2021 Third International Conference on Intelligent Communication Technologies and Virtual Mobile Networks (ICICV), 2021, pp. 1304-1309, doi: 10.1109/ICICV50876.2021.9388375.
[5] M Logeshwaran; J. Joselin Jeya Sheela, "Contactless Door System with Temperature Detection for Covid-19", 2022 4th International Conference on Smart Systems and Inventive Technology (ICSSIT), 2022, pp.1269-1273, doi: 10.1109/ICSSIT53264.2022.9716560
[6] Sumeeth Asnani; Ashwini Kunte; Mohammed Hasan Charoliya; Nivaan Gupta, "Temperature actuated non-touch automatic door", 2021 2nd International Conference on Smart Electronics and Communication (ICOSEC), 2021, pp. 669-675, doi: 10.1109/ICOSEC51865.2021.9591951
[7]https://circuitdigest.com/microcontroller-projects/raspberry-pi-motion-detector-pir-sensor
[8]https://circuitdigest.com/microcontroller-projects/face-mask-detection-using-raspberry-pi-and-opencv
[9]https://circuitdigest.com/microcontroller-projects/iot-based-contactless-body-temperature-monitoring-using-raspberry-pi-with-camera-and-email-alert
[10]https://thepihut.com/blogs/raspberry-pi-tutorials/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi
[11] https://projects.raspberrypi.org/en/projects/parent-detector/1
[12]https://www.electronicshub.org/raspberry-pi-l298n-interface-tutorial-control-dc-motor-l298n-raspberry-pi/
[13] https://tutorials-raspberrypi.com/raspberry-pi-servo-motor-control/
[14] https://www.youtube.com/watch?v=C603Asi6bk0
[15] https://www.youtube.com/watch?v=RcVn9NUtq0c
[16]https://www.youtube.com/watch?v=TYzOhgUmZc