

ABU DHABI UNIVERSITY



جامعة أبوظبي
Abu Dhabi University

Self-Driving Mask Detection Robot

by

Amer Barhoush - 1064958

Omar Farag - 1061007

Sohrab Setoodeh - 1067187

Zayed Aslam - 1065528

A thesis submitted in partial fulfillment for the
degree of Bachelor of Science in Computer Engineering

in the
Electrical and Computer Engineering Department
College of Engineering

August 2021

Declaration of Authorship

We, Amer Barhoush, Omar Farag, Sohrab Setoodeh, Zayed Aslam, declare that this thesis titled, ‘Self-Driving Mask Detection Robot’ and the work presented in it are our own. We confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where we have consulted the published work of others, this is always clearly attributed.
- Where we have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely our own work.
- We have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Signed:

Signed:

Signed:

Date:

“We haven’t got the money, so we’ll have to think.”

Ernest Rutherford

“Why don’t they have a light bulb that only shines on things worth looking at?”

George Carlin

“Education is a progressive discovery of our own ignorance.”

Will Durant

“If we knew what it was we were doing, it would not be called research, Would it?”

Albert Einstein

ABU DHABI UNIVERSITY

Abstract

College of Engineering
Electrical and Computer Engineering Department

by Amer Barhoush - 1064958

Omar Farag - 1061007

Sohrab Setoodeh - 1067187

Zayed Aslam - 1065528

Innovation and creativity always thrive when we face difficulties and crises, and what our world is currently facing is a global health crisis known as Coronavirus (Covid-19). Covid-19 is spreading quite fast, and it has already taken many lives all over the world. Thus, countries, including the United Arab Emirates, started to take action by issuing new rules and regulations to stop Covid-19 from spreading any further. Wearing face masks properly in public places is one of the most important rules the UAE government has issued, but some people tend to break this rule. Therefore, we have proposed the idea of designing a self-driving mask detection robot that can be deployed in public or closed places such as universities. The robot uses AI and machine learning to detect if people are wearing face masks correctly or not. If the robot detects people wearing face masks incorrectly or not wearing face masks at all, it will take images of them that can be used to punish those who violate safety rules. Face mask detection is implemented by detecting people's faces at first using a pre-trained face detector then applying a trained face mask detector on the detected faces. The face mask detector was trained using Tensorflow and OpenCV, and the neural networks we have used for our detection models are FaceNet and MobileNetV2. The robot also includes another main feature which is autonomous driving; this feature is implemented using Robot Operating System (ROS) which has autonomous navigation applications using Simultaneous Localization and Mapping (SLAM). The objective of SLAM in mobile robotics is to construct and update the map of an unexplored environment with the help of the available sensors attached to the robot which will be used for autonomous exploring. The robot will also be able to avoid any obstacles on the way while moving freely on the environment drawn by Simultaneous Localization and Mapping.

Acknowledgements

First of all, we are gratefully thankful to Allah the Almighty for blessing us with the knowledge, health, patience, and guidance throughout working on this project. Next, we would like to thank Dr. Mohammed Ghazal, Professor and Chair of the Electrical, Computer and Biomedical Engineering Department at Abu Dhabi University, and Dr. Huma Zia, Associate Professor in Computer Engineering Department in Abu Dhabi University for their teachings and continuous feedback and guidance that led us to grow throughout the semesters. Finally, we would like to thank the Office of Research and Sponsored Programs for funding this project.

Contents

Declaration of Authorship	i
Abstract	iii
Acknowledgements	iv
List of Figures	viii
List of Tables	xi
Abbreviations	xii
1 Introduction	1
1.1 Motivation	2
1.2 Problem Statement	3
1.3 Literature Review	3
1.3.1 YOLOv2 based Real Time Object Detection	3
1.3.2 Real-Time Object Detection Using TensorFlow	4
1.3.3 Obstacle Avoidance Algorithms for Autonomous Navigation system in Unstructured Indoor areas	6
2 Design	7
2.1 Requirements, Constraints, and Considerations	7
2.1.1 Requirements	7
2.1.2 Design Constraints	8
2.1.3 Considerations	8
2.1.4 Cost Breakdown	9
2.2 Design Process	9
2.2.1 Face Detection Software	9
2.2.2 Robot Design	10
2.3 Component Design	10
2.3.1 Nvidia Jetson Xavier NX	10
2.3.2 Turtlebot3	11
2.3.2.1 OpenCR	12
2.3.2.2 HLS-LFCD2 (LIDAR sensor)	13
2.3.2.3 11.1V 1800mAh Li-PO Rechargeable Battery	13

2.3.3	Logitech C920	13
2.3.4	Li-ion 18650	14
2.3.5	MicroSD	14
2.3.5.1	Nvidia Jetson Xavier NX Operating System	15
2.3.5.2	Raspberry Pi 3B+ Raspbian Operating System	15
2.3.6	Bluetooth speaker	16
2.3.7	Face mask detection design process	16
2.3.7.1	Software requirements	16
	Jetson OS: Ubuntu 18.04	16
	Python Modules	19
	Dataset	21
	Serialized Face Detector Model	21
2.3.8	Face mask detection model training	22
2.3.9	Face mask model detection process	25
2.3.10	Autonomous navigation design process	27
2.3.10.1	Software requirements	27
2.3.10.2	TurtleBot3 Assembly	31
2.3.10.3	Map drawing and autonomous navigation using SLAM	32
2.4	System Overview	33
3	Experimental Testing, Results and Analysis	35
3.1	Experimental Testing & Results	35
3.1.1	Test 1: Face mask detection using YOLOv2 object detector	35
3.1.2	Test 2: Face mask detection using Tensorflow and OpenCV on two categories	40
3.1.3	Test 3: Face mask detection using tensorflow and openCV on three categories	50
3.1.4	Test 4: Auto image capturing and Alert sound system when detecting people without face masks or wearing them incorrectly	52
3.1.5	Test 5: Jetson Xavier and Turtlebot3 operating time	54
3.1.6	Test 6: Face mask detection model accuracy on different face mask colors	55
3.1.7	Test 7: Autonomous navigation using ROS	58
3.2	Results and Discussion	60
3.2.1	Data set size	61
3.2.1.1	Data set 1	61
3.2.1.2	Data set 2	64
3.2.1.3	Data set 3	66
3.2.1.4	Data set 4	68
3.2.1.5	Data set 5	70
3.2.2	Number of epochs and Batch size	72
3.2.3	Distance and Face angle	72
3.3	Analysis and Interpretation of Data	73
3.3.1	Face mask detection model evaluation	73
3.3.1.1	Metrics for performance evaluation	74
3.3.1.2	Model evaluation and comparison	75
3.3.2	Summary of Analysis and Interpretation of Data	79

4	Conclusion	83
4.1	Summary	83
4.2	Future Improvements and Takeaways	85
4.3	Learned Lessons	85
4.4	Team Dynamics	86
4.4.1	Team Members	86
4.4.2	Tasks and Work Division	87
4.4.3	Team Communication and Gantt Chart	87
4.5	Impact Statement	90
4.5.1	Environmental Impact	90
4.5.2	Economic Impact	91
4.5.3	Social Impact	91
A	Python code for face mask detection testing	94
	Bibliography	99

List of Figures

1.1	YOLOv2 Detection of multiple object in single frame [1]	4
1.2	Object Detection using tensorflow [2]	5
2.1	Nvidia Jetson Xavier NX [3]	11
2.2	TurtleBot3 Burger [4]	12
2.3	OpenCR [5]	12
2.4	HLS-LFCD2 (Lidar sensor) [6]	13
2.5	Li-PO Rechargeable Battery [7]	13
2.6	Logitech C920 camera [8]	14
2.7	Li-ion 18650 [9]	14
2.8	MicroSD [10]	15
2.9	Raspberry pi 3B+ [11]	15
2.10	Bluetooth speaker [12]	16
2.11	Where to find the Jetson NX Developer Kit SD Card Image in their website	16
2.12	SD Card Formatter Software	17
2.13	Etcher's Last Step for Writing into a microSD	18
2.14	Where to insert the microSD in NVIDIA Jetson Xavier NX Developer Kit	18
2.15	First Boot-up Screen	19
2.16	Design Process for face mask detection model training	22
2.17	Creating altered images using Image Data Generator	23
2.18	Splitting Dataset into 70% Training, 20% Testing, and 10% validation	24
2.19	Different Initial Training Rates Vs Loss [13]	24
2.20	Design Process for face mask detection model detection	25
2.21	Getting the middle point from the face detected	26
2.22	Tracking the number of faces wearing the mask incorrectly or no mask. [14]	27
2.23	Audio Phrases for wearing mask incorrectly and not wearing a mask.	27
2.24	SD Card Formatter Software	29
2.25	Etcher's Last Step for Writing into a microSD	29
2.26	50-cloud-init.yaml file content	30
2.27	Where you need to change the IP Addresses	31
2.28	System Overview Diagram	33
2.29	Face mask detection System Diagram	34
2.30	TurtleBot3 System Diagram	34
3.1	Image labeling session in MATLAB	36
3.2	Training data exported from image labeling session	36
3.3	Training the YOLOv2 detector	38
3.4	Full face mask detection using YOLOv2 object detection	39

3.5	Full face mask detection using YOLOv2 object detection	39
3.6	Nosed exposed detection using YOLOv2 object detection	40
3.7	Mask under chin detection using YOLOv2 object detection	40
3.8	Face mask detection using tensorflow and openCV on one person without face mask and the other with	49
3.9	Face mask detection using tensorflow and openCV on two people without face masks	49
3.10	Face mask detection using tensorflow and openCV on two people with face masks	50
3.11	Face mask detection using tensorflow and openCV on two people (one wearing face mask correctly and the other is not)	51
3.12	Face mask detection using tensorflow and openCV on two people (one wearing face mask incorrectly and the other without a mask)	51
3.13	Face mask detection using tensorflow and openCV on two people wearing face mask incorrectly	52
3.14	Face mask detection using tensorflow and openCV training process on three categories	52
3.15	Training loss and accuracy plot for training three categories using Tensorflow and openCV	52
3.16	Wearing blue face mask incorrectly with an accuracy of 72.81%	56
3.17	Wearing blue face mask correctly with an accuracy of 93.32%	56
3.18	Wearing pink face mask incorrectly with an accuracy of 71.67%	57
3.19	Wearing pink face mask correctly with an accuracy of 67.88%	57
3.20	Wearing black face mask incorrectly with an accuracy of 34.38%	57
3.21	Wearing black face mask correctly with an accuracy of 93.03%	58
3.22	Wearing white face mask incorrectly with an accuracy of 81.54%	58
3.23	Wearing white face mask correctly with an accuracy of 83.02%	58
3.24	Drawing the IoT Lab in Abu Dhabi University using SLAM	59
3.25	Autonomous navigation and obstacle avoidance using TurtleBot3 in the IoT Lab	60
3.26	Training process and classification report for Data set 1	62
3.27	Training loss and accuracy plot for Data set 1	63
3.28	Face mask detection model accuracy of Wearing face mask incorrectly using Data set 1	63
3.29	Face mask detection model accuracy of Wearing face mask incorrectly using Data set 1	64
3.30	Training process and classification report for Data set 2	65
3.31	Training loss and accuracy plot for Data set 2	65
3.32	Face mask detection model accuracy of Wearing face mask incorrectly using Data set 2	66
3.33	Face mask detection model accuracy of Wearing face mask correctly using Data set 2	66
3.34	Training process and classification report for Data set 3	67
3.35	Training loss and accuracy plot for Data set 3	67
3.36	Face mask detection model accuracy of Wearing face mask incorrectly using Data set 3	67
3.37	Face mask detection model accuracy of Wearing face mask correctly using Data set 3	68

3.38	Training loss and accuracy plot for Data set 4	69
3.39	Training process and classification report for Data set 5	69
3.40	Face mask detection model accuracy of Wearing face mask correctly and without using Data set 4	69
3.41	Face mask detection model accuracy of Wearing face mask incorrectly using Data set 4	70
3.42	Training process and classification report for Data set 5	70
3.43	Training loss and accuracy plot for Data set 5	71
3.44	Face mask detection model accuracy of Wearing face mask incorrectly and correctly using Data set 5	71
3.45	Face mask detection model accuracy of Wearing face mask incorrectly using Data set 5	71
3.46	Face mask detection model accuracy of Wearing face mask correctly and without using Data set 5	72
3.47	Incorrect classification form the model due to the angle of the face	73
3.48	Confusion matrix along with the most widely-used metrics [15]	74
3.49	Testing instances up to 29 frames, Probability of correctly worn masks, True class, Prediction, TP, FP, TPR, FPR, Recall and Precision for the base model	75
3.50	ROC curve of the base model shown in green line with the default classifier shown in red	76
3.51	ROC Interpretation [15]	77
3.52	Testing instances up to 30 frames, Probability of correctly worn masks, True class, Prediction, TP, FP, TPR, FPR, Recall and Precision for the better performance model	78
3.53	ROC curve of the better performance model shown in blue line with the default classifier shown in red	78
3.54	ROC curve of the better performance model shown in blue line, base model shown in green line and the default classifier shown in red	79
4.1	Caption in landscape to a figure in landscape.	88
4.2	MS Team Group	89
4.3	Whats app Group	89
4.4	Home development	89
4.5	Environmental Impact Tool Screenshot	90
4.6	Economic Impact Tool Screenshot 1	91
4.7	Economic Impact Tool Screenshot 2	91
4.8	Social Impact Tool Screenshot 1	92
4.9	Social Impact Tool Screenshot 2	93
4.10	Social Impact Tool Screenshot 3	93

List of Tables

2.1	Cost Breakdown	9
3.1	Base model Vs. Better performance model	80
3.2	Dataset size, No. of epochs, Batch size, Distance and Face angle effect on accuracy	81
3.3	Summary of data set size, training time, distance and performance	82

Abbreviations

COVID-19	Corona Virus Disease 2019
YOLO	You Only Look Once
SD	Secure Digital
OpenCR	Open-source Control module
ROS	Robot Operating System
SLAM	Simultaneous Localization And Mapping
RPi 3B+	Raspberry Pi 3B+
LiPo	Lithium Polymer battery
Li-Ion	Lithium Ion
CNN	Convolutional Neural Network
TP	True Positive
FP	False Positive
TN	True Negative
FN	False Negative
FPR	False Positive Rate
TPR	True Positive Rate
ROC	Receiver Operating Characteristic

Chapter 1

Introduction

Due to the coronavirus (COVID-19) outbreak, many countries including the UAE have issued new rules and regulations to stop the spreading of Covid-19. One of the main regulations the UAE government has issued is to wear face masks in public places since COVID-19 is spread through airdrops and close contact. Wearing face masks will lower the rate of transmission and spreading of COVID-19, and if everyone took responsibility and wore face makes properly, this pandemic will end sooner.

Although wearing face masks in public and closed places is mandatory by the UAE government and critical to stopping the spread of COVID-19, some people do not wear masks or improperly wear them outside their homes which increases the risk of spreading or catching the virus from the people around them. The UAE has set strict fines on those who are spotted not wearing face masks or improperly wearing them. Also, to ensure that people are always wearing face masks properly in public, monitoring them is considered to be one of the most effective ways.

Having an individual (e.g. a security guard) monitor other people in public for wearing face masks or not can be quite hard. In crowded places, monitoring multiple people at once would be hard for us humans; because we could miss a few people in the progress of looking at every person in our sights at the same time. Also, if we happened to spot someone wearing a face mask improperly, that someone could quickly adjust his/her face mask properly as if nothing has happened. On the other hand, having an Artificial Intelligence (AI) robot monitor people can eliminate the incapability humans have. AI

machine can be both fast and accurate in identifying whether people are wearing face masks properly or not.

Having a camera that can monitor people and detect face masks using machine learning and computer vision can be efficient and accurate enough to spot people who are not wearing face masks properly or not at all. What is even better is to have a face mask detection camera patrol through specified areas and take pictures of everyone who is violating the face mask rules while also avoiding different obstacles in the way.

The main purpose of this project is to design a robot capable of detecting face masks in public and closed places such as malls, parks, restaurants, markets, workplaces, universities, etc. The robot will also be able to navigate autonomously around the place it has been deployed to. AI and machine learning will be used to detect face masks then classify if people are wearing them properly or not. If the robot detects one person or multiple people not wearing their face masks correctly or without face masks at all, it will capture their image and alert them using recorded voice alerts. Navigation and localization will be used to assist the robot in driving autonomously while avoiding any objects on the way.

1.1 Motivation

The main motivation behind this project is to reduce the spread of the Covid-19 virus in closed and small open areas such as universities and malls.

As we know, the UAE government made great efforts to reduce the spread of this virus by using modern technologies and development. These developments include using thermal cameras and face mask detection at public places entrances. These efforts have been made to ensure the public good and safety. We as a team want to be a part of this effort by designing this solution.

In this project, The robot will help to notify the people who are not wearing a mask or wearing it improperly by capturing an image of the person and then generating a sound in both Arabic and English languages to warn the person to wear the mask. The Robot will function and run autonomously using AI and machine learning. The benefit

of this solution is to ensure people's safety and well being as it's our main objective in this project.

1.2 Problem Statement

The main problem the team is trying to tackle is to find a way to increase the safety of well-being by reducing the spread of the Coivd-19 virus. Unfortunately, The virus has already infected many people and took some lives as we know from now this is a really big issue on our society health. In this project, we are attempting to develop a self-driving mask detection robot that will detect individuals that are either not wearing a mask or wearing it improperly and then alert them to wear their masks properly. The robot will be controlled fully using AI, machine learning and computer vision. For the first part, the robot will navigate autonomously and move while avoiding obstacles using ROS and SLAM. As for the face mask detection part, the robot will use computer vision and machine learning to detect people's faces then apply a face mask detection model to detect face masks on their faces and classify them. This method will be implemented using a pre-trained face detector (FaceNet), and a trained face mask detector using tensorflow and openCV.

1.3 Literature Review

1.3.1 YOLOv2 based Real Time Object Detection

Object detection is one of the classical problems in computer vision where we try to let the computer see and recognize objects inside an image. There are many different algorithms used to detect objects in images and many researchers are doing their best to get the best AI model. However, In this research paper, YOLOv2 is used to detect objects on real-time footage. Yolov2 is an upgrade of the original YOLO algorithm which has some enhancements in its computing time and speed. What also makes it better is that we can use GPU to reinforce the training speed by processing 40 fps. Moreover, As we know YOLOv1 starts its processing procedure by dividing the image into $S \times S$ grids and usually $s=7$ and when the desired objects fall in the center of a grid that grid will be responsible to detect that object [16]. However, in the second version

of YOLO, they used convolution with anchor boxes to detect multiple objects per grid this enhances the algorithm by allowing it to detect multiple objects in less processing time [1].



FIGURE 1.1: YOLOv2 Detection of multiple object in single frame [1]

In the paper, They trained a YOLOv2 object detector using a COCO dataset of 80 classes of random objects such as traffic lights and cars. The model is trained using GPU and CUDA to detect and localize objects using anchor boxes which helps also in detecting small and far objects in an image. Some of the results are shown in Figure 1.1 . The detector was tested on a real-time video and the result of the detection was obtained and stored in the .avi file after the testing is done [1].

1.3.2 Real-Time Object Detection Using TensorFlow

Object detection is one of the main challenges in computer vision, where we use the computer to detect, label, and locate objects. In this research, An object detection algorithm is developed using TensorFlow which is a free and open-source machine learning that is offered by Google. It allows the users to create graphs computations to see the performance of the model and improve it. Python is used as the front-end language and in the backend, they are using C++. Moreover, The algorithm will detect the object and point it out to its location by drawing a bounded box around the object. In Tensorflow there is a pre-trained object detector that can be used directly and the developer can also develop its model from scratch. What even makes it better than Tensorflow offers Object detection API that contains pre-trained machine learning and object detection

model that uses open-source data sets such as; COCO, KITTI, and Open Images Data sets, which makes this model performance much accurate with less error. Those models are used in many different companies to increase their growth and technologies. Facebook uses the image recognition system to target people with their Ads, It is used in voice recognition by Apple's Siris and more. To start with the API it's not mandatory to have good knowledge about neural networks and machine learning. The API can simply be used by importing the model file and start testing the algorithm on your test data. As we can see in [Figure 1.2](#) a random image from google was used to test the object detection model. We can see that it was able to detect objects and bound them with a bounding box. It also shows the accuracy of the detection which means how confident id the detector about the detect object [2] .

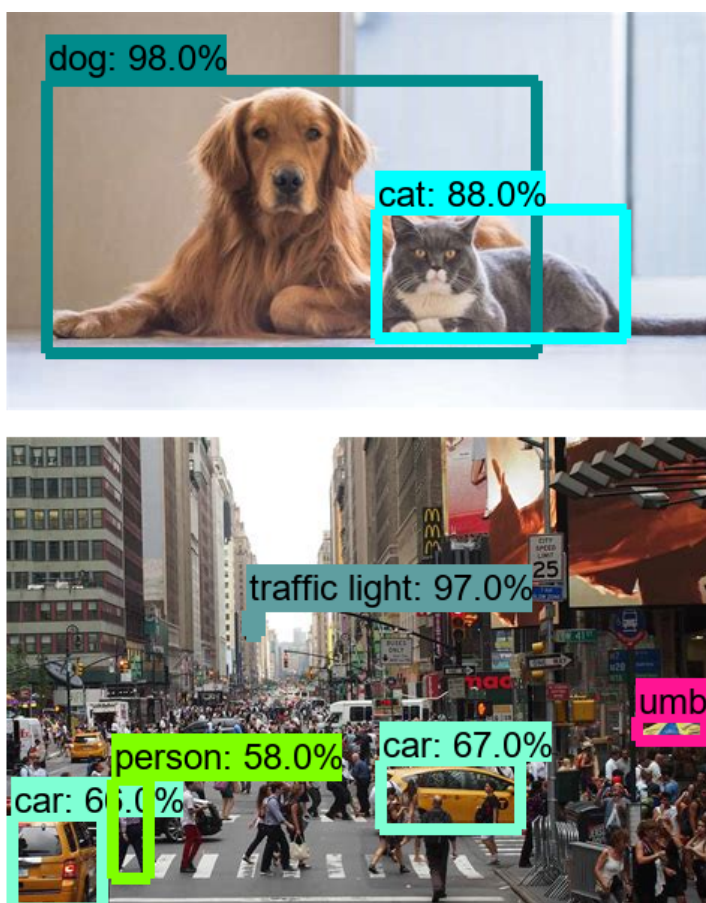


FIGURE 1.2: Object Detection using tensorflow [2]

1.3.3 Obstacle Avoidance Algorithms for Autonomous Navigation system in Unstructured Indoor areas

The concept of robotization has become one of the hottest topics in research fields, because of the development of technology and the availability of open-source software's this led the big companies to take the advantage of robots to develop a new solution to automate most of their tasks. In this work, They studied the problem of autonomous navigation in an indoor environment in which the Robot will move in a path and avoid any obstacles in the way. The TurtleBot3 robot was used alongside a LIDAR sensor which was responsible for the obstacle detection the task of the sensor was to scan 180 degrees in front of the robot to draw an MAP in RViz software which is a 3d visualization software available in ROS environment. This software was used to visualize the data that are coming from the LIDAR sensor to be able to control the behavior of the Robot [17].

Chapter 2

Design

The main design processes to implement self-driving face mask detection robot include:

- Turtlebot3 Design and Configuration.
- NVIDIA Jetson Xavier NX Configuration and Deployment.
- Face mask detection model training and testing.
- Auto image capture and sound alert system.

In this section, we will discuss each design process needed for implementing a self-driving face mask detection in addition to the overall system overview.

2.1 Requirements, Constraints, and Considerations

2.1.1 Requirements

- Design and implement a robot that can move autonomously in closed environment while avoiding any obstacles in the way.
- Develop a machine learning model that can detect face masks on people's faces and classify them accordingly.
- Design and Implement a sound alerting system that alerts people who are not wearing face masks or wearing them incorrectly.

- Design and Implement a sound alerting system that alerts people who are not wearing face masks or wearing them incorrectly.

2.1.2 Design Constraints

- Cost: no more than 4500 AED.
- Weight: The TurtleBot3 Burger can handle maximum payload of 15kg.
- Performance: The Robot can detect face masks on at least 2 people in a single frame.
- Sustainability: The Robot power consumption should be as low as possible to provide more sustainable and environmental Robot.
- Usability: The Robot should capture the image of people who violate the face mask rule in public places. In addition to an alerting sound system that alerts people who violates the face mask rule in both Arabic and English.
- Aesthetic: Since the Robot can operate in public and closed places, it should be visually elegant to the public.
- Functionality:
 - It must be able to navigate autonomously in closed environments or public places.
 - It must be able to detect and avoid any obstacles in its path.
 - It must detect and classify people in three categories (Wearing mask correctly, Wearing mask incorrectly, Not wearing Mask).
 - The robot should operate at least 2 hours without running out of charge.
- Accuracy: The robot should be able to detect and classify face masks on people's faces with high accuracy.

2.1.3 Considerations

The following standards and codes were considered and incorporated in our design:

- IEEE 3652.1-2020 - IEEE Guide for Architectural Framework and Application of Federated Machine Learning
- P2986 - Recommended Practice for Privacy and Security for Federated Machine Learning
- P1900.8 - Standard for Training, Testing, and Evaluating Machine-Learned Spectrum Awareness Models
- P2671 - Standard for General Requirements of Online Detection Based on Machine Vision in Intelligent Manufacturing

We used IEEE 3652.1-2020 and P2986 to obtain privacy and security for our data set by not sharing it to any unauthorized people. We also used P1900.8 standard for training, testing and cross-validation methods to be used in machine learning.

2.1.4 Cost Breakdown

Table 2.1 shows all the cost breakdown of all used components in this project:

TABLE 2.1: **Cost Breakdown**

Item	Cost
Turtlebot3 Burger	2178.75 AED
Jetson Xavier NX	1999 AED
Bluetooth Speaker	70 AED
Logitech Webcam C920	320 AED
x3 4.2V Li-on Batteries	45 AED
Long Nuts & Spacer	50 AED
Plastic Holders	20 AED
MicroSD Card 64GB	35 AED
<i>Total</i>	4717.75 AED

2.2 Design Process

2.2.1 Face Detection Software

When we first started in our project we all agreed to use YOLO v2 and Matlab for our machine learning because it was what we learn by Dr. Mohammed Ghazal during

his course "Special Topic: Computer Vision and Machine Learning". We began with using Matlab to create the dataset by using a video we captured using our phones of each other where we change our face position and face masks then label each frame with the corresponding label. While one of us was working on labeling the others researched for ways to make this easier since it was very time consuming and will result in a dataset including only our faces. We came across a website called Kaggle which has a large amount of datasets for face masks; However, it wasn't compatible with YOLO v2 so through further research we found out and learned Python's Tensorflow along with its other modules which are compatible with the datasets we found. After further discussion we came into the conclusion to use Python's Tensorflow since it will further our knowledge and allow us access to more datasets. Learning and getting Tensorflow to work was a challenge because it had many requirements and wasn't as user friendly as YOLO v2 but with it came versatility.

2.2.2 Robot Design

2.3 Component Design

2.3.1 Nvidia Jetson Xavier NX

After some research, we chose to deploy our mask detection model into the Nvidia Jetson Xavier NX which is a powerful microprocessor board that is designed especially for AI applications. The Nvidia Jetson NX comes with Nvidia Volta GPU which contains 48 Tensor cores, 6-core Nvidia Carmel CPU, and 8 GB memory shared between CPU and GPU which can run our mask detection model with acceptable performance. Moreover, the Nvidia Jetson Xavier NX has an advantage in processing computer vision applications using high-quality cameras which can give us up to 60 frames per second in full HD (1080p).



FIGURE 2.1: Nvidia Jetson Xavier NX [3]

2.3.2 Turtlebot3

TurtleBot3 is a small, affordable, programmable, ROS-based mobile robot for use in education, research, hobby, and product prototyping. The TurtleBot3's core technology is SLAM, Navigation and Manipulation, making it suitable for home service robots. The TurtleBot can run SLAM(simultaneous localization and mapping) algorithms to build a map and can drive around your room. Also, it can be controlled remotely from a laptop, joypad or Android-based smart phone. TurtleBot3 Burger achieves this goal by using the following components Raspberry pi 3, OpenCR, HLS-LFCD2, Dynamixel XL430 x2, and Li-PO 11.1V 1800 mAh Battery.

TurtleBot3 Burger

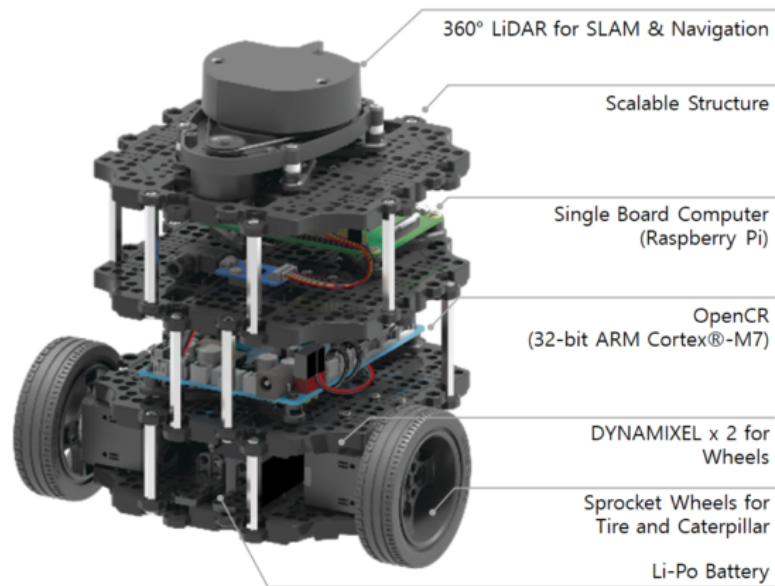


FIGURE 2.2: TurtleBot3 Burger [4]

2.3.2.1 OpenCR

OpenCR is used for its ROS capabilities, which will allow us to connect many different hardware components on it and control them easily using the Raspberry pi 3. We will use the OpenCR to connect the motors and control the navigation and movement of the robot.



FIGURE 2.3: OpenCR [5]

2.3.2.2 HLS-LFCD2 (LIDAR sensor)

We use a Lidar sensor that comes with the TurtleBot3 burger bundle. The Lidar has better Accuracy when it comes to obstacle detection from other components such as a camera or an ultrasonic sensor. What also makes it better for our work is that the Lidar can cover 360 Degrees to avoid any obstacles from the near environment.



FIGURE 2.4: HLS-LFCD2 (Lidar sensor) [6]

2.3.2.3 11.1V 1800mAh Li-PO Rechargeable Battery

To power the Raspberry pi 3 and TurtleBot3 actuators we used an 11.1V Li-PO Battery which provides us with a good amount of voltage and current, with a capacity of 1800mAh which can run the Robot up to 2 Hours.



FIGURE 2.5: Li-PO Rechargeable Battery [7]

2.3.3 Logitech C920

To have better result in mask detection we choice Logitech C920 camera which can give us up to full HD resolution with 78 degree view. It give us clear picture which will be process in our mask detection code so whenever we have clearer picture we get better

result in mask detection. It is flexible camera that we can adjust it in different angle because we are installing it below the leader so we need to adjust in in 45 degree.



FIGURE 2.6: Logitech C920 camera [8]

2.3.4 Li-ion 18650

To power up our Nividea Jetson Xavior NX we connect three Li-on 4.2V 18650 battery's in series to give us 12.6 Voltage which can start the Nividea Jetson Xavior NX. Li-on 18650 it has advantage because it is high capacity battery which have around 3600 mAh.



FIGURE 2.7: Li-ion 18650 [9]

2.3.5 MicroSD

Both Nvidia Jetson Xavier NX and Raspberry pi needs an SD card as main storage of the system so we get two of 64 GB Micro SD card to use it as main storage for our operating system and some of other application. We get 64 GB because it will give us

enough space to install as many as python liberty's on it and other application such as slam and python compiler.



FIGURE 2.8: MicroSD [10]

2.3.5.1 Nvidia Jetson Xavier NX Operating System

After following Nvidia guide line for Jetson Xavier Nx we have seen that it use Linux operating system which design specially for it. It is operating system has very similar interface as Ubuntu interface and has additional applications and drivers that designed for Jetson Xavier

2.3.5.2 Raspberry Pi 3B+ Raspbian Operating System

Also when we are following the turtle bot initial setup we found that there are many operating systems for raspberry pi such as melodic and kinetic. After some reading we choice a melodic because we can control it using our Jetson Xavier operating system.



FIGURE 2.9: Raspberry pi 3B+ [11]

2.3.6 Bluetooth speaker

We used a Bluetooth Speaker for our alerting system. The speaker will alert the people who are either not wearing their mask properly or not wearing it at all. The speaker is connected to the Jetson Xavier through a Bluetooth connection.

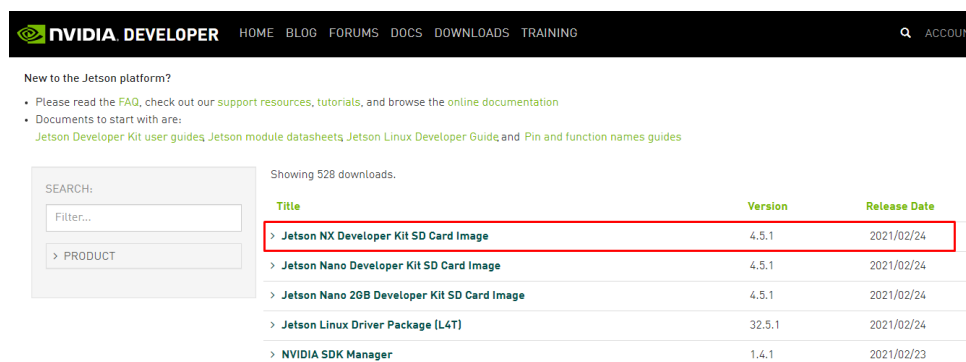


FIGURE 2.10: Bluetooth speaker [12]

2.3.7 Face mask detection design process

2.3.7.1 Software requirements

Jetson OS: Ubuntu 18.04 In order to be able to use the NVIDIA Jetson Xavier NX Developer Kit you will have it's Jetson NX Developer Kit SD Card Image written in the microSD to do that you will need to connect the microSD to your computer and download the Jetson NX Developer Kit SD Card Image from the [nvidia's developer official website](#) as seen in the Figure 2.11.



The screenshot shows the NVIDIA Developer website with a navigation bar and a search bar. Below the navigation bar, there is a section for 'New to the Jetson platform?' with links to FAQ, support resources, tutorials, and online documentation. A search bar is present with a 'Filter...' input and a 'PRODUCT' dropdown. Below the search bar, there is a table of downloads with 528 items shown. The table has columns for Title, Version, and Release Date. The first row, 'Jetson NX Developer Kit SD Card Image', is highlighted with a red box.

Title	Version	Release Date
> Jetson NX Developer Kit SD Card Image	4.5.1	2021/02/24
> Jetson Nano Developer Kit SD Card Image	4.5.1	2021/02/24
> Jetson Nano 2GB Developer Kit SD Card Image	4.5.1	2021/02/24
> Jetson Linux Driver Package (L4T)	32.5.1	2021/02/24
> NVIDIA SDK Manager	1.4.1	2021/02/23

FIGURE 2.11: Where to find the Jetson NX Developer Kit SD Card Image in their website

After you download the image you will need to download [SD Memory Formatter](#) and launch it and select the card drive and then select quick format and leave the volume label as blank then click format and click yes in the warning dialog as seen in [Figure 2.12](#).

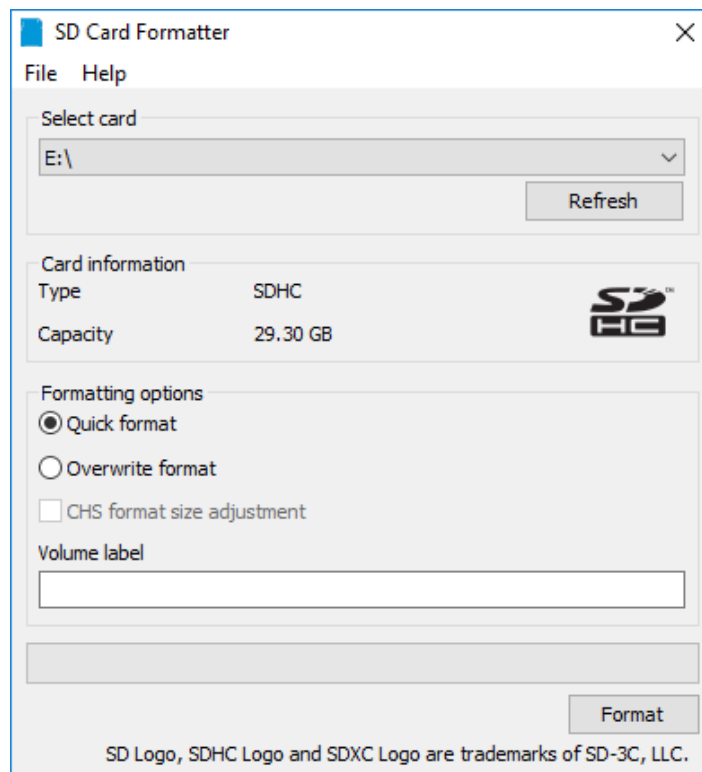


FIGURE 2.12: SD Card Formatter Software

After you are done formatting the microSD you will need to download [Etcher](#) and launch it and click "Select Image" and choose the Jetson NX Developer Kit SD Card Image you downloaded earlier and then select drive for the microSD and click "Flash!" as seen in [Figure 2.13](#).

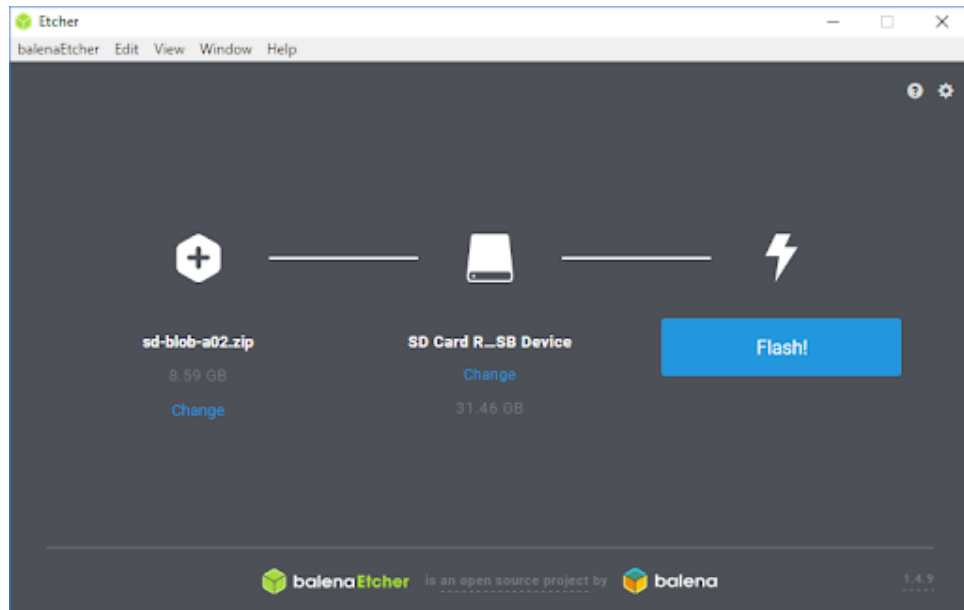


FIGURE 2.13: Etcher's Last Step for Writing into a microSD

once it is done writing to the microSD, you can remove the microSD and connect it to NVIDIA Jetson Xavier NX Developer Kit as shown in the Figure 2.14.

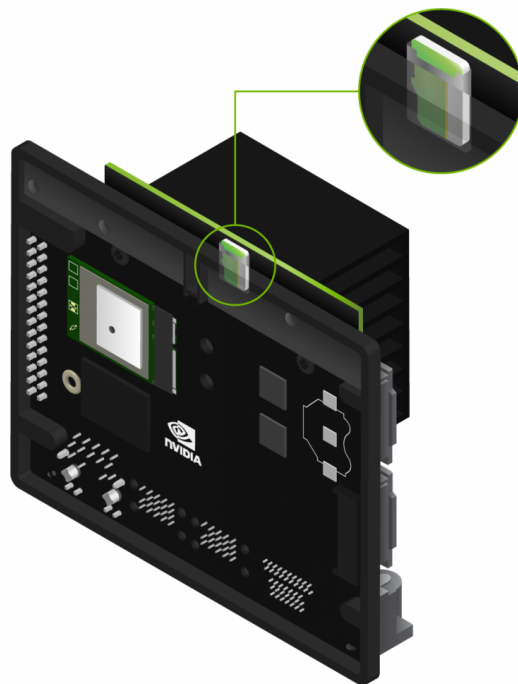


FIGURE 2.14: Where to insert the microSD in NVIDIA Jetson Xavier NX Developer Kit

Now you will need to connect keyboard, mouse to the USB ports of NVIDIA Jetson Xavier NX Developer Kit and monitor to it's HDMI Port and connect NVIDIA Jetson Xavier NX Developer Kit to a power supply. Now boot up the NVIDIA Jetson Xavier NX Developer Kit and review and accept "NVIDIA Jetson software EULA" then select the system language, keyboard layout, and time zone. Connect to the internet using wire or wireless network and create a username, password, and computer name then log in which will land you in this screen as Figure 2.15.

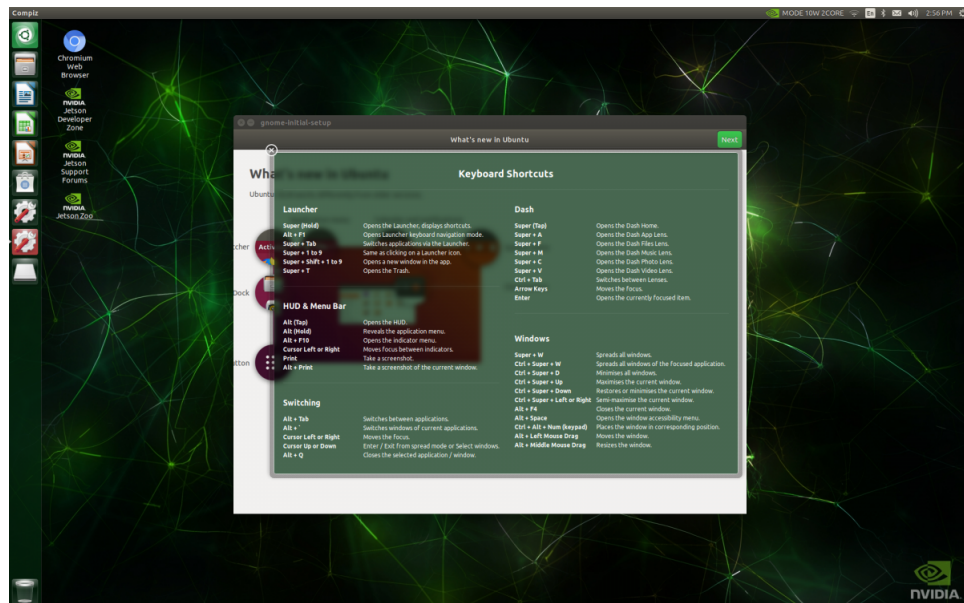


FIGURE 2.15: First Boot-up Screen

Python Modules Fortunately the Jetson NX Developer Kit SD Card Image comes prepacked with python, CUDA, and CUDNN which are needed to run tensorflow in GPU instead of CPU. To be able to run our python code for face and mask detection you will need a couple of python modules like tensorflow, keras, imutils, and numpy to name a few. We will show which version you need for each and how to install them.

First you will need make sure the pip installer and default modules are up to date to do that run the following three commands.

```
sudo apt-get upgrade
sudo apt-get update
sudo pip install --upgrade pip
```

Tensorflow is an open-source made by Google for machine learning and it provides a number of useful libraries that will help us achieve detection. For our code you will

need a Tensorflow with a version of 1.15.2 or higher and to install that you will need to open up the terminal in the Ubuntu operating system of the NVIDIA Jetson Xavier NX Developer Kit and type the following command.

```
sudo pip install tensorflow>=1.15.2
```

Keras is a high level API which acts as the interface for tensorflow library and it can be installed for python using the command below.

```
sudo pip install keras==2.3.1
```

Imutils is a python module which offers many image processing options like resizing, rotation, and etc. You will need to add it to your python module library by writing the following command in the terminal.

```
sudo pip install imutils==0.5.3
```

NumPy is a Python package that adds support for big, multi-dimensional arrays and matrices to the python language. To add it to your python module library you will need to write the command below in the Ubuntu Terminal.

```
sudo pip install numpy==1.18.2
```

The OpenCV-Python library is a set of Python bindings for solving computer vision issues. OpenCV-Python makes use of Numpy Python Module to solve computer vision issues so it essential to have them both installed. In the previous step you installed numpy, so to install OpenCV-Python you need to run this command in the terminal of Ubuntu.

```
sudo pip install opencv-python==4.2.0.*
```

Matplotlib is the last module you need to install and it is a plotting library for the Python programming language and its numerical mathematics extension NumPy. Matplotlib is another module that makes use of numpy and to include it in your python module library by running the following command.

```
sudo pip install scipy==1.4.1
```

Threading is a default python module which will allows us to increase the speed in which we capture camera frames by running it in a separate thread, and threading will

also allow us to call playsound function that can play voice alerts in Arabic and English from mp3 files.

Time is another default python module that helps capture current time during the execution of our code.

OS is the operating system python module that can be used to write the images into files so we can keep a folder of images of people who were spotted not wearing a mask or wearing it incorrectly.

Math python module is useful because using it we can calculate the distance between faces detected each frame by using the distance formula [2.1](#).

$$Distance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (2.1)$$

Dataset In order to train our mask detector model we will need a large enough dataset of images with each categorized into their respective label. Fortunately a website [Kaggle](#) provides many datasets for various machine learning training needs, so with the lack of manpower we have and the pandemic we are in, creating a good large dataset is almost impossible to do. We were able to find three Kaggle Datasets that will help us create our mask model. The three kaggle datasets we used and had to manually categorized each image are [Face Mask Detection](#), [Face Mask Detection 12K Images Dataset](#), and [Face Mask Detection Dataset](#); However, in order to save you the time of having to manually download them and categorized them into three folders, we have uploaded them ready for use in this [dataset.zip](#) file.

Serialized Face Detector Model The way we put our code is that first we will detect a face then pass that face into our face mask detector which will label it into Correctly worn mask, Incorrectly worn mask, Without mask. There are many prebuilt face detection models that are available in the internet so, instead of having to create a face detection model from scratch we used one from the internet with high accuracy. In order to detect masks using our python code you will need to download [face_detector.rar](#) file and it consists of a prototxt and caffemodel file.

prototxt file has image classification or segmentation model that used to be trained in the caffe. Prototxt is considered a prototype machine learning model created for use using caffe and are used to create the .caffemodel file.

caffemodel file is the model that was created using caffe along with large datasets in order to have high accuracy with detecting faces.

2.3.8 Face mask detection model training

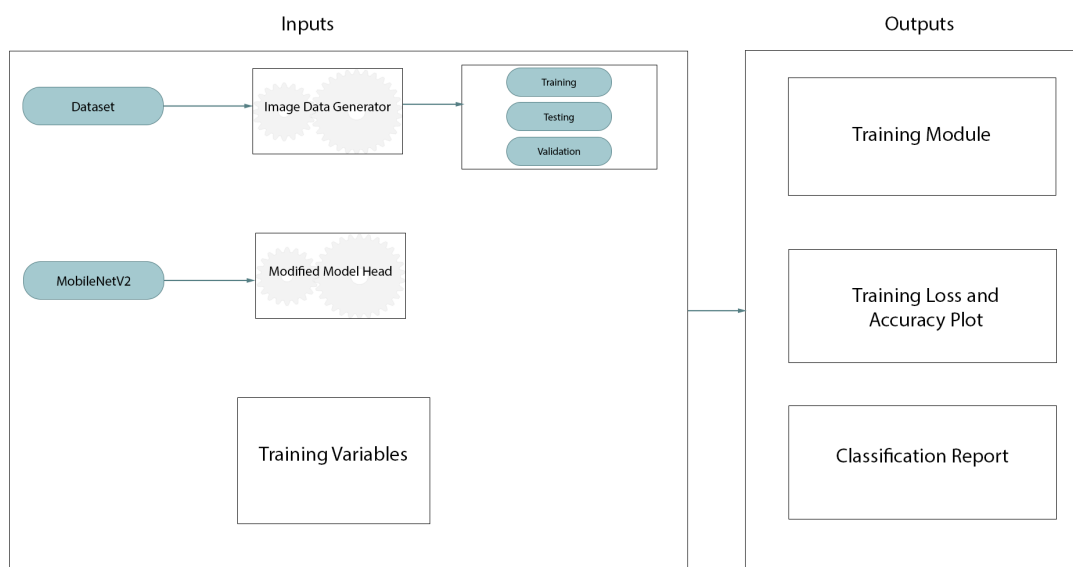


FIGURE 2.16: Design Process for face mask detection model training

Our inputs for the face mask detection model training code is the dataset, MobileNetV2 pretrained model, and the training variable. The **dataset** is first categorized into 3 labels correct, incorrect, and no mask and put into their respective folder inside the dataset folder then each image will be slightly altered using the image data generator provided by tensorflow which will change the rotation, size, zoom, and shift of the image to increase the accuracy then will divide the dataset into training, testing, and validation.

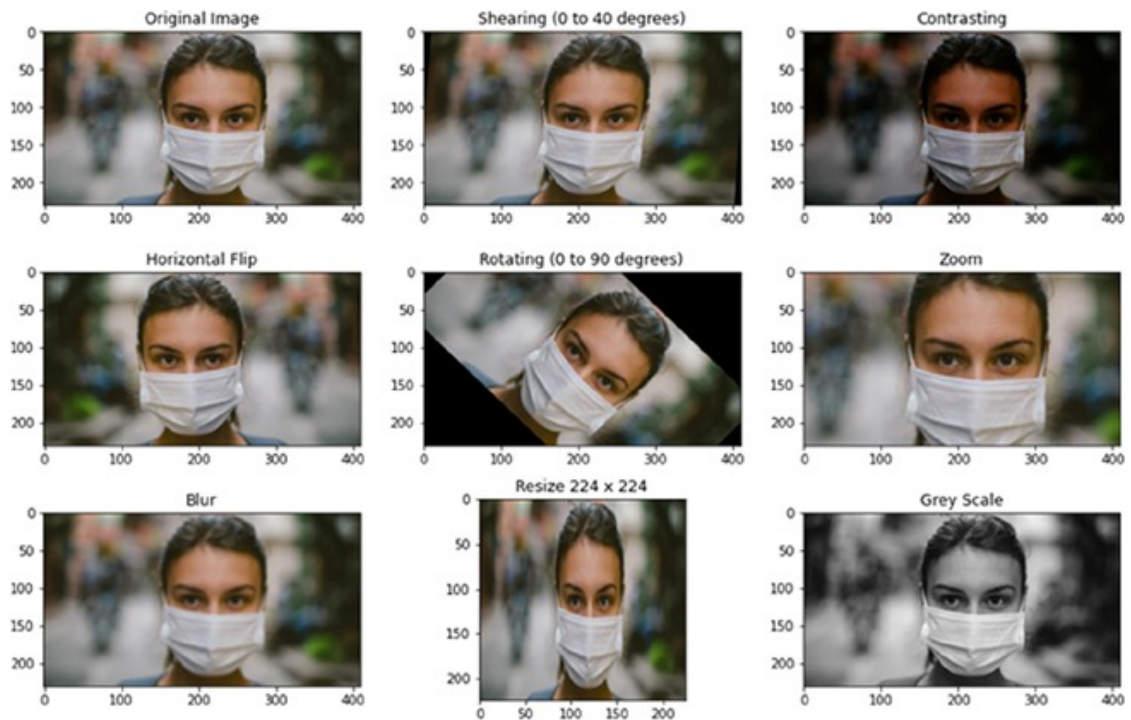


FIGURE 2.17: Creating altered images using Image Data Generator

MobileNetV2 is a pretrained module included with tensorflow that aims to help training modules, increase accuracy, and speed. Finally we picked the **training variables** like the initial training rate, batch size which will dictates how many training samples it will work through before updating the internal parameters, and finally the image size. The higher the image size is the slower it will be detecting the masks but with higher accuracy and since we are working on a real time application we selected an image size that is in middle so we get good accuracy and speed.

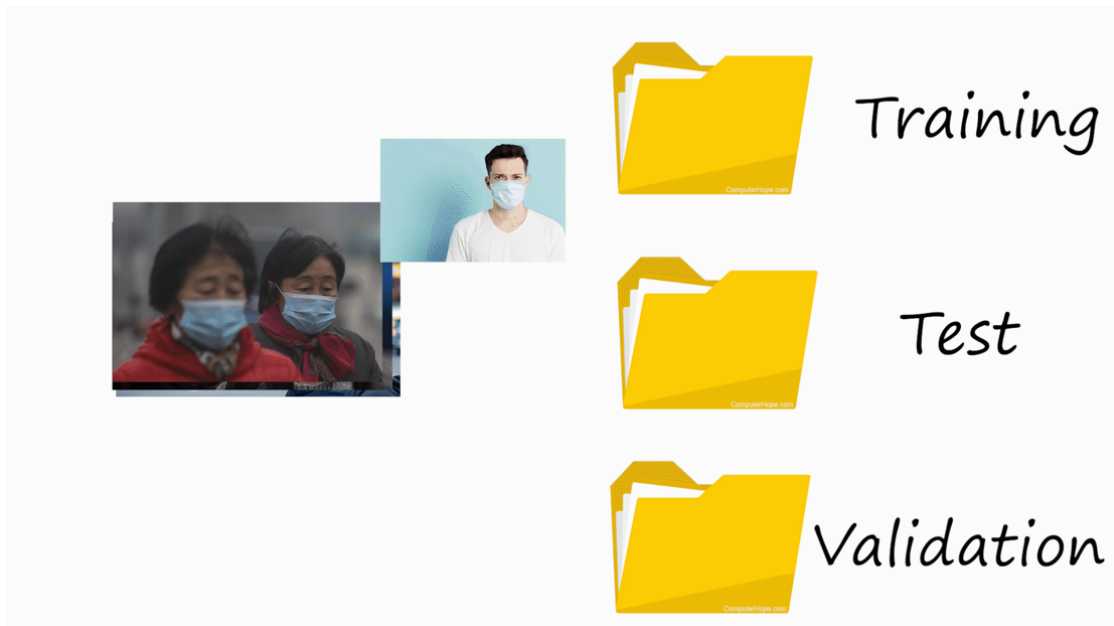


FIGURE 2.18: Splitting Dataset into 70% Training, 20% Testing, and 10% validation

Now for the outputs we will get the model that will be used in our detection code, training loss and accuracy plot, and classification report to see how well our training data set is doing against the test and validation data sets.

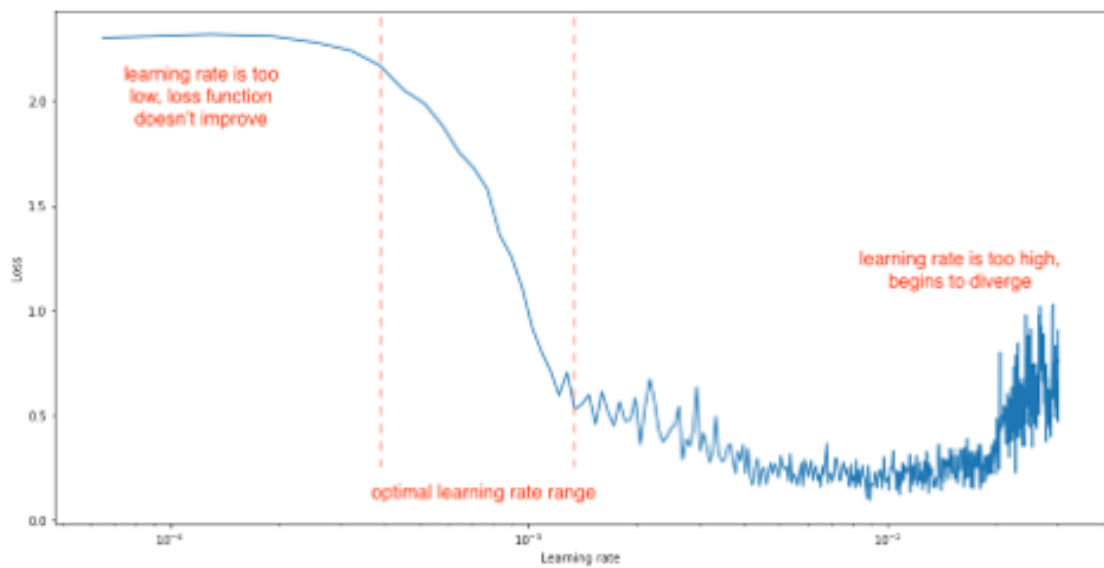


FIGURE 2.19: Different Initial Training Rates Vs Loss [13]

2.3.9 Face mask model detection process

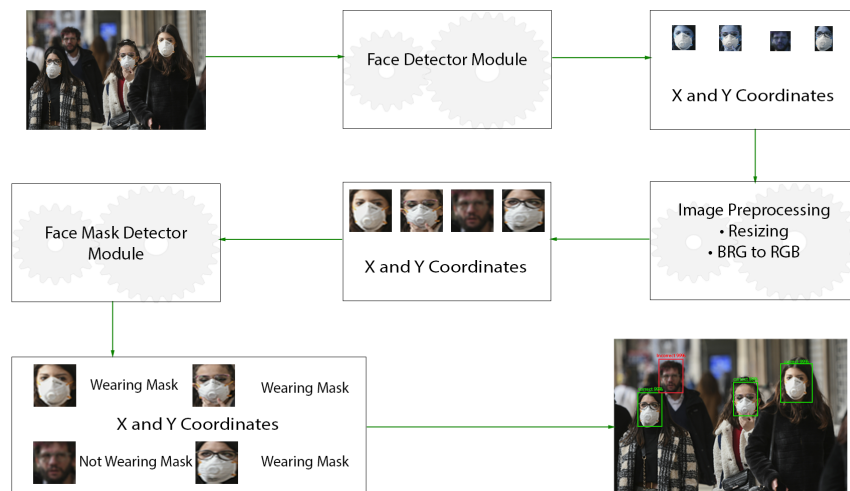


FIGURE 2.20: Design Process for face mask detection model detection

Each camera frame first goes through the pre-trained face detector module which will crop each face it detects that has confidence of 70% or higher and that face will be outputted in a BRG color along with its X and Y coordinates, so we need to change its color back to RGB then resize all this cropped face its 224 pixels wide and tall like we trained our mask detector model. Now each of these pre-processed cropped face images will go through our face detector model we made and the classification with the highest confidence will get that label. Using the coordinates of the faces and the labels it will go back to the original image and draw rectangles on the faces with color representing the label and the label with confidence percentage written above it.

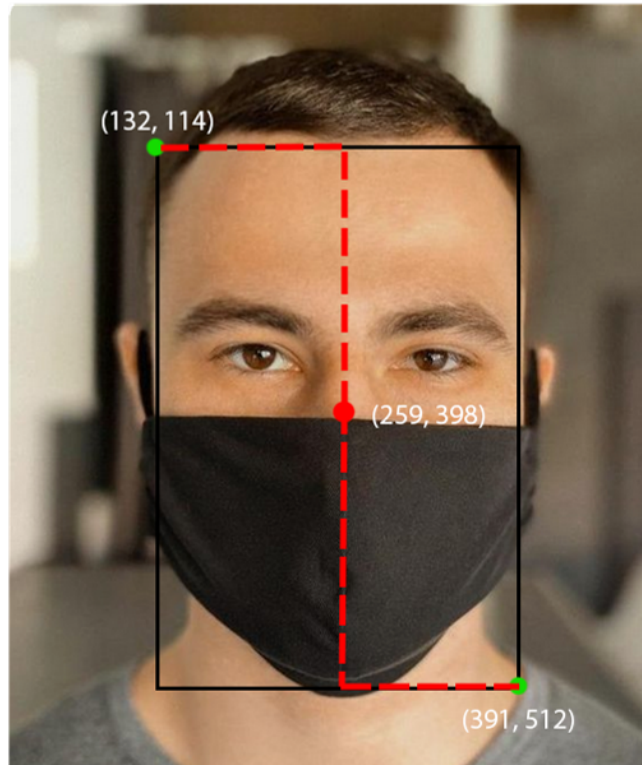


FIGURE 2.21: Getting the middle point from the face detected

We wanted to be able to take a screenshot for every new person that isn't wearing a mask or wearing incorrectly, so the default method of having it take screenshot every time there is someone in the frame not wearing mask or wearing it incorrectly isn't going to cut it since each second we loop many times and that would result in waste of space. We came up with an idea to track each person in the frame so that it wont take multiple screen shots for a single person and we did that by taking the middle point of his face as seen in Figure 2.21 Then calculate the distance 2.1 between the first frame and the next one and if that distance is above 25 and the new person is not wearing a mask or wearing it incorrectly then it will take a screenshot. This idea didn't work because someone running or is close the camera will make distance calculate really high and think it is a new person and results in taking multiple screenshots.

Our second idea was to track the number of people wearing the mask incorrectly or not wearing a mask each frame and comparing it with the next frame. If the next frame has a higher total number of people wearing mask incorrectly or not wearing a mask then it would take a screenshot and play the audio as seen in Figure 2.22. This idea was simpler to implement and worked even if the person was wearing a mask then took it off unlike the first idea.

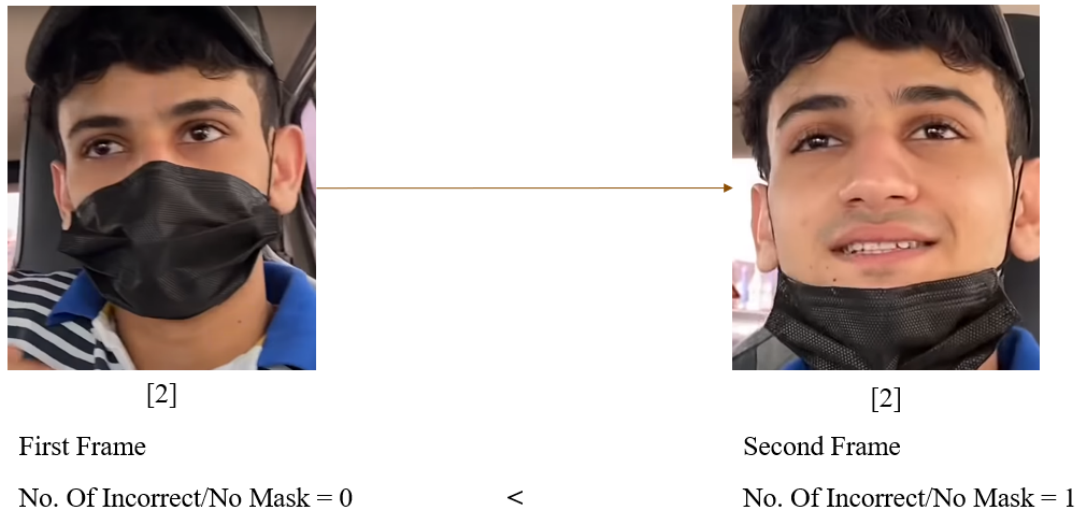


FIGURE 2.22: Tracking the number of faces wearing the mask incorrectly or no mask. [14]

Now to play the audio we used a Bluetooth speaker that is connected to the jetson and added four audio files to the jetson two for wearing it incorrectly in Arabic and English and two for not wearing a mask in Arabic and English. We used threads so that it will not hinder the face mask detection code and limited it to a single thread so that the audio will not overlap.

<p style="text-align: center;">Incorrect Mask Detected</p> <p style="text-align: center;">الرجاء وضع الكمامة بالشكل الصحيح</p> <p style="text-align: center;">Please wear your mask properly</p>	<p style="text-align: center;">No Mask Detected</p> <p style="text-align: center;">الرجاء وضع الكمامة</p> <p style="text-align: center;">Please wear a mask</p>
---	--

FIGURE 2.23: Audio Phrases for wearing mask incorrectly and not wearing a mask.

2.3.10 Autonomous navigation design process

2.3.10.1 Software requirements

In order to use ROS packages, We have to use a PC with Linux 18.04 version so we decide to use our jetson Xavier as our Remote PC. The first step we need to install ROS 1 in our Remote PC by running the following commands in the terminal.

```
sudo apt update
sudo apt upgrade
wget https://raw.githubusercontent.com/ROBOTIS-GIT/robotis_tools/master/install_ros_melodic.sh
chmod 755 ./install_ros_melodic.sh
bash ./install_ros_melodic.sh
```

After we make sure that the ROS package has been installing. We install the second package which is Dependent ROS 1 Packages by running the following command.

```
sudo apt-get install ros-melodic-joy ros-melodic-teleop-twist-joy \
ros-melodic-teleop-twist-keyboard ros-melodic-laser-proc \
ros-melodic-rgbd-launch ros-melodic-depthimage-to-laserscan \
ros-melodic-rosserial-arduino ros-melodic-rosserial-python \
ros-melodic-rosserial-server ros-melodic-rosserial-client \
ros-melodic-rosserial-msgs ros-melodic-amcl ros-melodic-map-server \
ros-melodic-move-base ros-melodic-urdf ros-melodic-xacro \
ros-melodic-compressed-image-transport ros-melodic-rqt* \
ros-melodic-gmapping ros-melodic-navigation ros-melodic-interactive-markers
```

After installing the ROS package we need also to install the ROS package designed especially for turtlebot3 by running the following three commands.

```
sudo apt-get install ros-melodic-dynamixel-sdk
sudo apt-get install ros-melodic-turtlebot3-msgs
sudo apt-get install ros-melodic-turtlebot3
```

Next, we need to choose our Turtlebot3 model so as we are using the burger model we will run the next command.

```
echo "export TURTLEBOT3_MODEL=burger" >> ~/.bashrc
```

Now we need to set up our raspberry pi 3b+. First, we need to download a ROS1 Melodic image for raspberry pi 3b+ and unzip it. After it, we launch SD Card Formater same as we used for the Nividea setup. Then we select the card drive and then select a quick format and leave the volume label as blank then click format and click yes in the warning dialog as seen in the Figure [2.24](#).

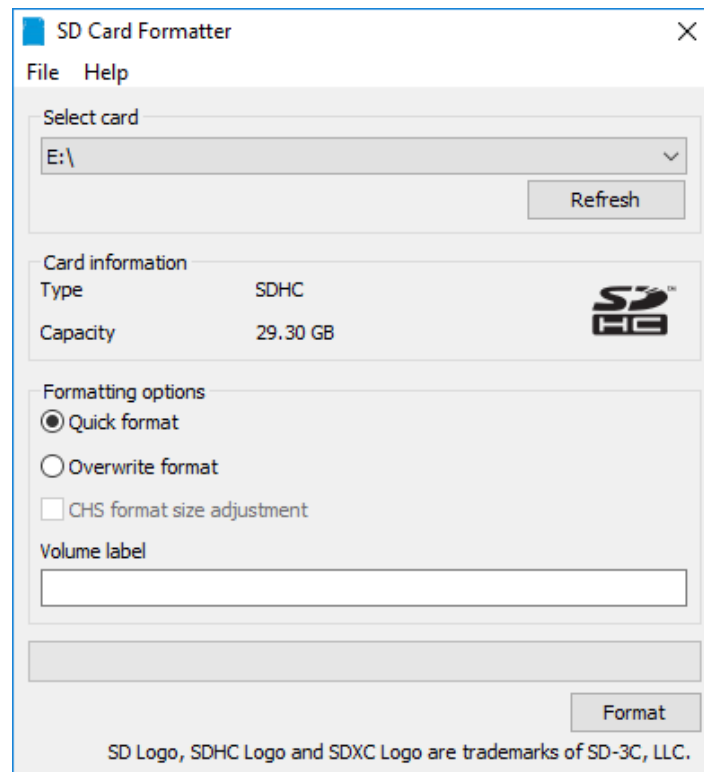


FIGURE 2.24: SD Card Formatter Software

After you are done formatting the microSD you will need to download [Etcher](#) and launch it and click "Select Image" and choose the ROS1 Melodic Image and then select drive for the microSD and click "Flash!" as seen in the Figure 2.25.

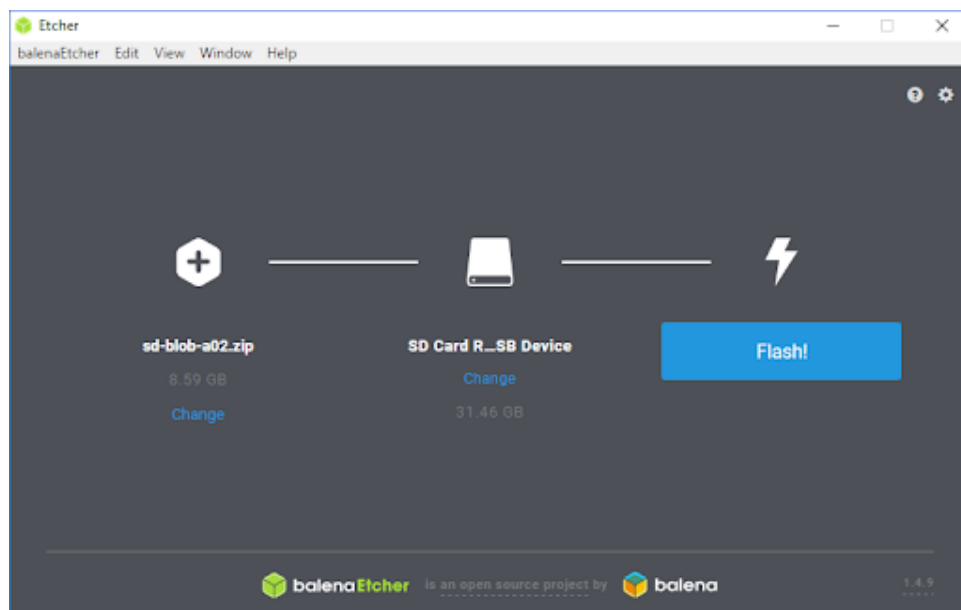
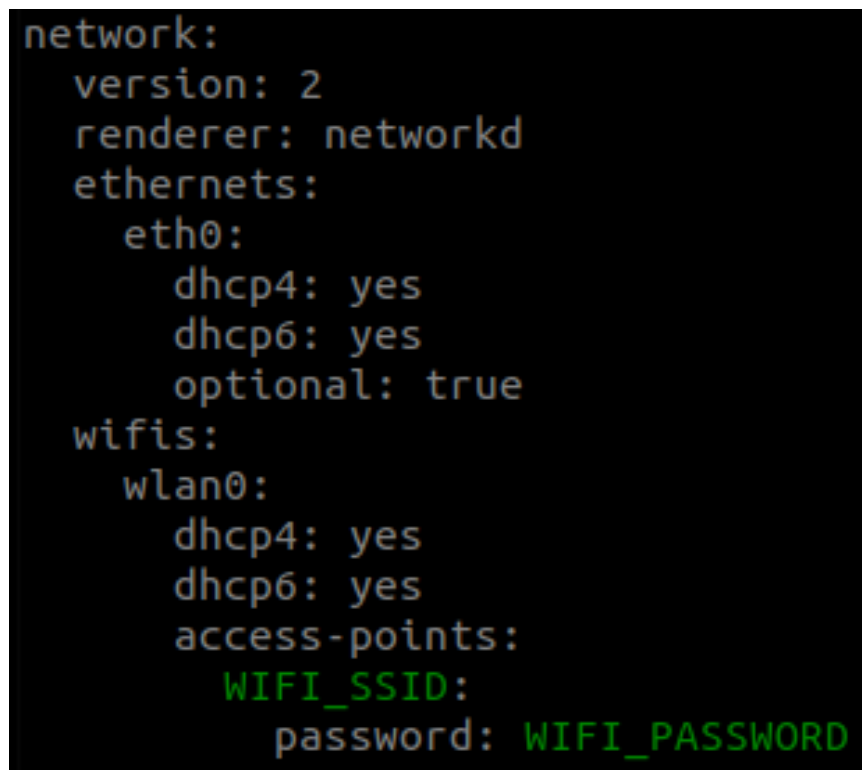


FIGURE 2.25: Etcher's Last Step for Writing into a microSD

Once it is done writing to the microSD, you will need to configure the wifi setting from your PC by running the following command.

```
cd /media/$USER/writable/etc/netplan
sudo nano 50-cloud-init.yaml
```

Then it will appear some text you need to add your network name next to the "WIFI-SSID:" and network password next to "password " as showing in Figure 2.26 then save by pressing ctrl + s



```
network:
  version: 2
  renderer: networkd
  ethernets:
    eth0:
      dhcp4: yes
      dhcp6: yes
      optional: true
  wifis:
    wlan0:
      dhcp4: yes
      dhcp6: yes
      access-points:
        WIFI_SSID:
          password: WIFI_PASSWORD
```

FIGURE 2.26: 50-cloud-init.yaml file content

After finishing the above steps you need to take the SD Card and insert it into the raspberry pi. Now you will need to connect the keyboard, mouse to the USB ports of Raspberry pi 3b+ and monitor to its HDMI Port. Now boot up the Raspberry pi 3b+ and you will need to enter the default username which is "ubuntu" and password "turtlebot" after it you need to run some command to finish ROS Configuration.

The following steps should be done for both raspberry and remote PC. You need to get IP for both Raspberry pi and Remote PC by running the following command.

```
ifconfig
```

Then edit the `.bashrc` file by running the next code.

```
nano ~/.bashrc
```

Next, we need to replace "IP-ADDRES-OF-REMOTE-PC" to our remote PC IP and replace "IP-ADDRES-OF-RASPBERRY-PI-3" to our Raspberry pi 3b+ IP as showing in Figure 2.27.

```
export ROS_MASTER_URI=http://{IP_ADDRESS_OF_REMOTE_PC}:11311
export ROS_HOSTNAME={IP_ADDRESS_OF_RASPBERRY_PI_3}
```

FIGURE 2.27: Where you need to change the IP Addresses

lastly, you need to apply the change by running the next command.

```
source ~/.bashrc
```

Now we need some steps to set up our Open CR. First, need to connect Open CR with USB to Raspberry pi then we need to install some Packages on a raspberry pi to insert it to Open CR firmware by running the following code.

```
sudo dpkg --add-architecture armhf
sudo apt-get update
sudo apt-get install libc6:armhf
export OPENCNCR_PORT=/dev/ttyACMO
export OPENCNCR_MODEL=burger
rm -rf ./opencnrcr_update.tar.bz2
wget https://github.com/ROBOTIS-GIT/OpenCR-Binaries/raw/master/turtlebot3/ROS1/latest/opencnrcr_update.tar.bz2
tar -xvf opencnrcr_update.tar.bz2
```

After Downloading the packages we need to upload them to Open CR by executing the next commands.

```
cd ./opencnrcr_update
./update.sh OPENCNCR_PORT OPENCNCR_MODEL.opencnrcr
```

2.3.10.2 TurtleBot3 Assembly

- In first layer, we attach the two motors that will make the robot move freely and use the empty space to put our main battery (Li-Po 11.1V) for autonomous driving system.

- In second layer, we attach the openCR because it needs to be charged from the main battery and the motor will be connect with it by wire .
- In third layer, We attach raspberry pi which will take power from oopencer and it will connect to opencer from usb to have communication.
- In fourth layer, We attach our NVIDIA jetson xavier nx because it will connect to the separate battery which we will install in same layer and to the next layer which will have the camera.
- In the fifth layer, We build a long layer to attach camera out of the main frame of the turtlebot to have full view of the area. Also we attach the speaker behind the camera were we have empty space and to spread the width
- In sixth layer and the last layer, we have the lidar that will scan the are to make the robot to avoid any obstacles. We choices it to be in last layer because it will scan 360 degree so in last layer there will not be anything that not be disturbed.

2.3.10.3 Map drawing and autonomous navigation using SLAM

After finishing the installation of all ROS packages and assembling our robot now we are ready to start our robot and create our map. To create Map we will use slam so first, we need to Run the Slam node by running the following command in Remote PC.

```
roscore
```

Then you need a new terminal to connect to raspberry pi with it is IP Address by executing the next two commands.

```
ssh pi@{IP_ADDRESS_OF_RASPBERRY_PI}
roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

Also, you need another terminal to lunch the slam node by running the next code in the new terminal.

```
export TURTLEBOT3_MODEL=${TB3_MODEL}
roslaunch turtlebot3_slam turtlebot3_slam.launch
```

If you succeed to run the slam node then we need something to control the robot so we will use the teleoperation node which gives users the ability to control turtlebot from

the Remote PC keyboard. We can run it by executing the following commands in a new terminal.

```
export TURTLEBOT3_MODEL=burger
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

After finishing the mapping we can test our robot by pressing the pose estimation to initialize robot location in the map and we use navigation bottom to select the point to which we want the robot to travel.

In our robot we want the robot to go in an infinite location in an infinite loop so we edit in the main python code that will give some pointers that will make the robot start moving and avoid any obstacles.

2.4 System Overview

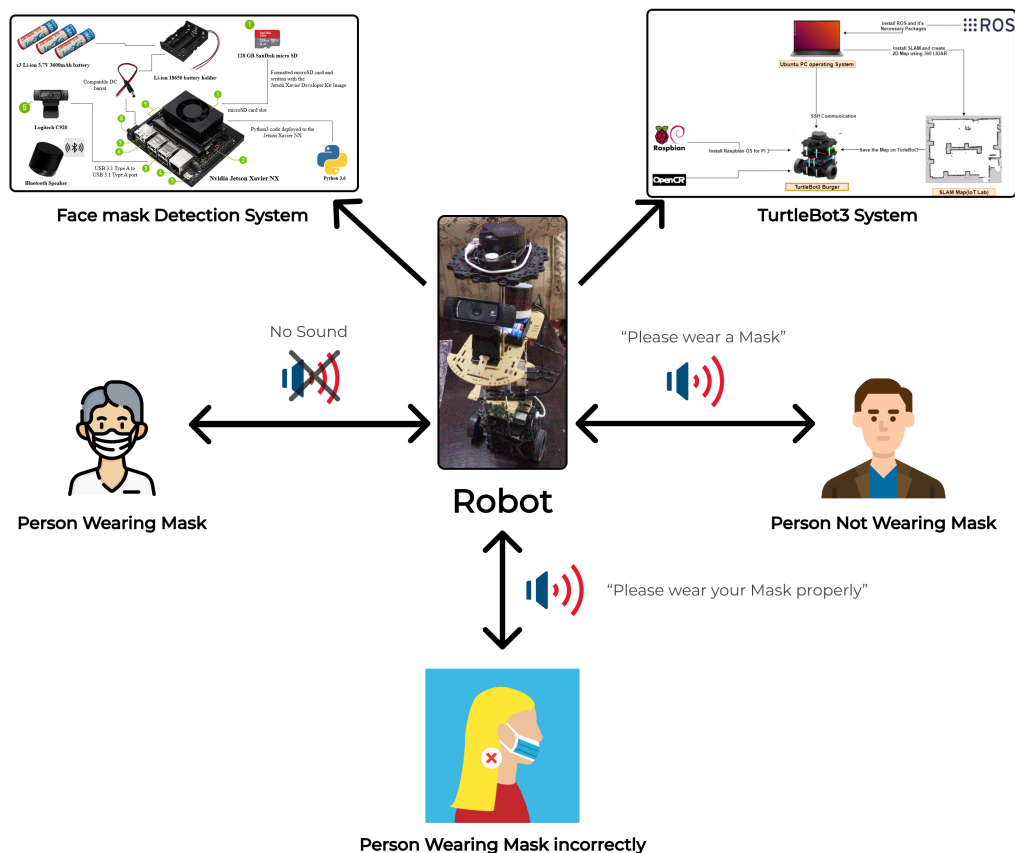


FIGURE 2.28: System Overview Diagram

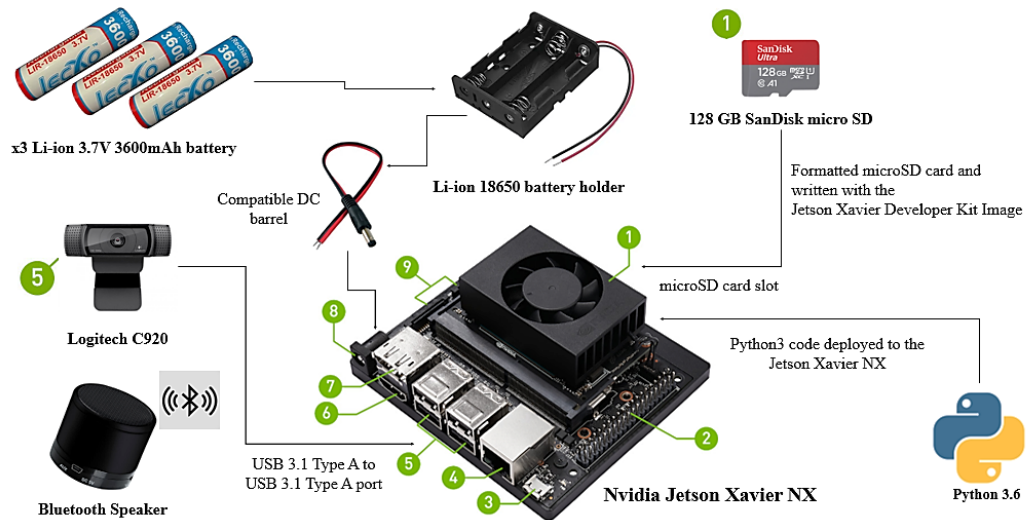


FIGURE 2.29: Face mask detection System Diagram

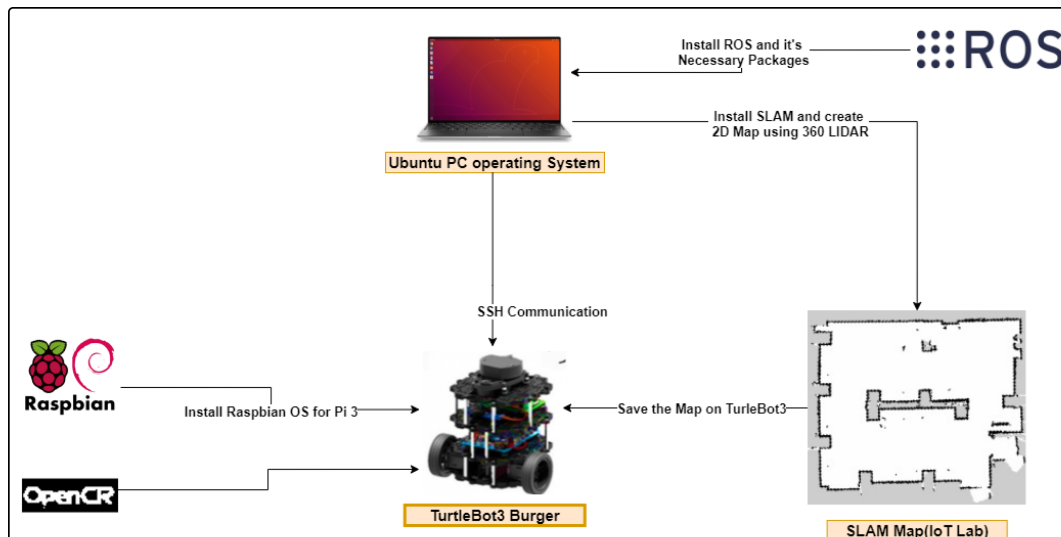


FIGURE 2.30: TurtleBot3 System Diagram

Chapter 3

Experimental Testing, Results and Analysis

3.1 Experimental Testing & Results

3.1.1 Test 1: Face mask detection using YOLOv2 object detector

- **Test Description:** In this test, we used YOLOv2 object detection to detect face masks on people's faces and label them accordingly. The labels we have used for this test are "Full Mask", "Nose Exposed" and "Chin Mask".
- **Test Steps:**
 1. We collected over 100 images of people wearing masks correctly and not (Nose exposed and Chin Mask) from online sources. See: [Face mask detection dataset](#)
 2. We resized all the images to 416x416 to match the input layer of YOLOv2 using MATLAB.

```
srcFiles = dir('F:\mask detection\images\*.jpg');  
for i = 1 : length(srcFiles)  
    filename = strcat('F:\mask detection\images\',srcFiles(i).name);  
    im = imread(filename);  
    k = imresize(im,[300,300]);  
    newfilename = strcat('F:\mask detection\images\',srcFiles(i).name);  
    imwrite(k, newfilename, 'jpg');  
end
```

3. Then, we started an image labeling session on MATLAB and imported the resized images.
4. We created 3 labels "Full Mask", "Nosed Exposed" and "Chin Mask".
5. We started labeling face masks on each image according to its right label.

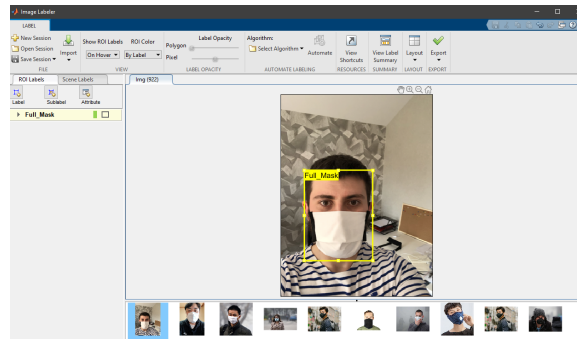


FIGURE 3.1: Image labeling session in MATLAB

6. We exported the ground truth data to our workplace which will be used as training data. The ground truth data includes the image path + the coordinates of the label on that image. The labeling shape we used is of type "box label" which has 4 points that will represent the coordinates of the label on the image.

	1 imageFilename	2 Full_Mask
1	'C:\Users\user\Desktop\Images\Img (1).jpg'	[93,37,61,50]
2	'C:\Users\user\Desktop\Images\Img (1).png'	[58,184,117,130]
3	'C:\Users\user\Desktop\Images\Img (2).jpg'	[38,53,183,144]
4	'C:\Users\user\Desktop\Images\Img (3).png'	3x4 double
5	'C:\Users\user\Desktop\Images\Img (4).jpg'	[51,59,162,143]
6	'C:\Users\user\Desktop\Images\Img (4).png'	[133,250,111,129]
7	'C:\Users\user\Desktop\Images\Img (5).png'	[89,53,73,81]
8	'C:\Users\user\Desktop\Images\Img (6).jpg'	[112,26,42,27]
9	'C:\Users\user\Desktop\Images\Img (6).png'	[182,37,42,39]
10	'C:\Users\user\Desktop\Images\Img (7).jpg'	[90,35,61,42]
11	'C:\Users\user\Desktop\Images\Img (7).png'	[204,105,176,187]
12	'C:\Users\user\Desktop\Images\Img (8).jpg'	[56,49,126,133]
13	'C:\Users\user\Desktop\Images\Img (8).png'	[22,178,140,150]
14	'C:\Users\user\Desktop\Images\Img (9).jpg'	[68,43,127,132]
15	'C:\Users\user\Desktop\Images\Img (9).png'	[76,62,58,54;276,51,62,55]
16	'C:\Users\user\Desktop\Images\Img (10).jpg'	[80,49,79,78]
17	'C:\Users\user\Desktop\Images\Img (10).png'	[]
18	'C:\Users\user\Desktop\Images\Img (11).jpg'	[93,84,49,57]
19	'C:\Users\user\Desktop\Images\Img (11).png'	[46,89,29,31;329,73,31,35]
20	'C:\Users\user\Desktop\Images\Img (12).jpg'	[78,56,118,91]
21	'C:\Users\user\Desktop\Images\Img (12).png'	[79,61,125,135]
22	'C:\Users\user\Desktop\Images\Img (13).jpg'	[91,43,86,72]
23	'C:\Users\user\Desktop\Images\Img (13).png'	[]

FIGURE 3.2: Training data exported from image labeling session

7. We calculated the Anchor boxes and Intersection over Union (IoU) using the "estimateAnchorBoxes" built-in function in MATLAB which takes the training data and number of Anchor boxes as its arguments.

```

data = load("trainingDataMask.mat");
maskDataset = data.gTruth;
summary(maskDataset);
trainingData = boxLabelDatastore(maskDataset(:,2:end));
numAnchors = 4;
[anchorBoxes,meanIoU] = estimateAnchorBoxes(trainingData,
                                             numAnchors);

anchorBoxes
meanIoU

```

8. We designed the YOLOv2 network layers which include the input layer, filter size and middle layers.

```

inputLayer = imageInputLayer([128 128 3], 'Name', 'input',
                              'Normalization', 'none');

filterSize = [3 3];
middleLayers = [ convolution2dLayer(filterSize, 16, 'Padding',
    , 1, 'Name', 'conv_1', 'WeightsInitializer', 'narrow-normal')
    batchNormalizationLayer('Name', 'BN1')
    reluLayer('Name', 'relu_1')
    maxPooling2dLayer(2, 'Stride', 2, 'Name', 'maxpool1')
    convolution2dLayer(filterSize, 32, 'Padding', 1, 'Name', 'conv_2',
        'WeightsInitializer', 'narrow-normal')
    batchNormalizationLayer('Name', 'BN2')
    reluLayer('Name', 'relu_2')
    maxPooling2dLayer(2, 'Stride', 2, 'Name', 'maxpool2')
    convolution2dLayer(filterSize, 64, 'Padding', 1, 'Name', 'conv_3',
        'WeightsInitializer', 'narrow-normal')
    batchNormalizationLayer('Name', 'BN3')
    reluLayer('Name', 'relu_3')
    maxPooling2dLayer(2, 'Stride', 2, 'Name', 'maxpool3')
    convolution2dLayer(filterSize, 128, 'Padding', 1, 'Name', 'conv_4',
        'WeightsInitializer', 'narrow-normal')
    batchNormalizationLayer('Name', 'BN4')
    reluLayer('Name', 'relu_4')
];

```

9. We assembled the YOLOv2 network which includes the front layer, filter size and middle layers.

```

lgraph = yolov2Layers([128 128 3] , numClasses ,

```

```
anchorBoxes, lgraph, 'relu_4');
```

10. We included training options and called the YOLOv2 training function which is a built-in function in MATLAB called "trainYOLOv2ObjectDetector".

```
options = trainingOptions('sgdm', ...
    'InitialLearnRate', 0.001, ...
    'Verbose', true, 'MiniBatchSize', 8, 'MaxEpochs', 16, ...
    'Shuffle', 'every-epoch', 'VerboseFrequency', 50, ...
    'DispatchInBackground', false, ...
    'ExecutionEnvironment', 'cpu');

[detectorYolo2, info] =
trainYOLOv2ObjectDetector(trainingData, lgraph, options);
```

11. We trained the model and observed the learning loss in each epoch.

5	450	00:21:06	0.27	7.2e-02	0.0010
5	500	00:23:22	0.43	0.2	0.0010
5	550	00:25:36	0.32	1.0e-01	0.0010
6	600	00:27:50	0.36	0.1	0.0010
6	650	00:30:05	0.28	7.9e-02	0.0010
7	700	00:32:19	0.42	0.2	0.0010
7	750	00:34:33	0.28	7.7e-02	0.0010
8	800	00:36:49	0.46	0.2	0.0010
8	850	00:39:24	0.39	0.1	0.0010
9	900	00:41:55	0.26	6.9e-02	0.0010
9	950	00:44:27	0.22	4.8e-02	0.0010
9	1000	00:46:57	0.49	0.2	0.0010
10	1050	00:49:27	0.18	3.3e-02	0.0010
10	1100	00:52:01	0.22	4.6e-02	0.0010
10	1150	00:53:02	0.17	2.9e-02	0.0010

Detector training complete.

FIGURE 3.3: Training the YOLOv2 detector

12. We obtained the YOLOv2 object detector as a result of training the YOLOv2 object detector.
13. We used the YOLOv2 object detector on images, videos and live video capture.
- **Expected Results:** Accurate and unbiased YOLOv2 detector that can detect face masks from reasonable distances and angles of multiple people in a frame and correctly labeling them.
 - **Observed Results:** We observed that the YOLOv2 detector is quite inaccurate when it comes to detecting multiple people in one frame, and the detector had a hard time detecting face masks from long distance and different angles.
 - **Acceptance Criteria:** The YOLOv2 object detector was able to detect face masks from distances over 3m and wide angles of people's faces when it comes

to images, but it showed low performance when it comes to live stream camera detection.

- **Test Results:** The test results succeeded when it comes to short distances, detecting individuals and while they are looking straight at the camera, but it could not meet the criteria of our test.

Although the YOLOv2 detector showed low performance on live stream camera, the results of applying the detector on images were quite accurate.



FIGURE 3.4: Full face mask detection using YOLOv2 object detection

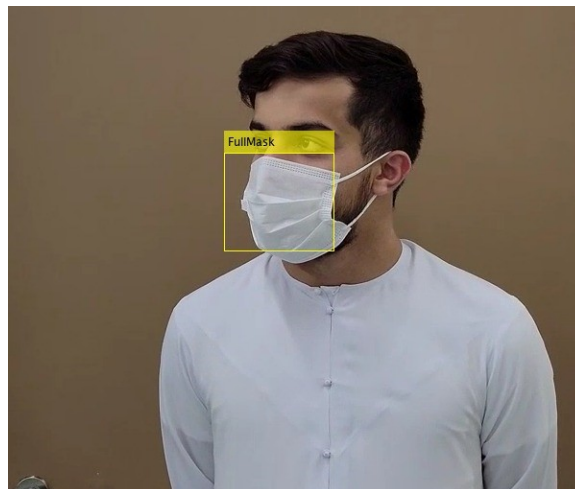


FIGURE 3.5: Full face mask detection using YOLOv2 object detection



FIGURE 3.6: Nosed exposed detection using YOLOv2 object detection



FIGURE 3.7: Mask under chin detection using YOLOv2 object detection

3.1.2 Test 2: Face mask detection using Tensorflow and OpenCV on two categories

- **Test Description:** In this test, we used Tensorflow2 to train and run deep neural networks (MobileNetV2) to obtain a detector model that is able to detect face masks and label them correctly. The labels we used for this test are "Correct" and "Incorrect" where incorrect includes wearing the mask under the chin, exposing

the nose and not wearing the mask at all. Additionally, we used a pre-trained neural network (FaceNet) to detect people's faces before applying the mask detector model. Total of labels for this test are 2 ("Correct" and "Incorrect")

- **Test Steps:**

1. The first step will be collecting our data which contains images of people wearing face masks correctly and not. The number of images we obtained are over 14,000 where 7500 are labeled as "correct" and 7500 are labeled as "incorrect". The data we have used are from a free and available online source. See: [Correct and Incorrect face mask dataset](#)
2. Install Anaconda3 on PC and launch the Anaconda prompt.
3. Install all required packages to train the model. These packages include Tensorflow2, Keras2, numpy, openCV, matplotlib and scipy from scikit-learn.
4. Create new python file for training the detection model.
5. Import all necessary packages needed for training.

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

6. Initialize variables for learning rate, number of epochs and batch size.

11. Construct training image generator for data augmentation.

```
aug = ImageDataGenerator(
    rotation_range=7,
    zoom_range=0.05,
    width_shift_range=0.15,
    height_shift_range=0.15,
    shear_range=0.1,
    horizontal_flip=True,
    fill_mode="nearest")
```

12. Load the MobileNetV2 network and ensure the head layer is not included.

```
baseModel = MobileNetV2(weights="imagenet", include_top=False,
                        input_tensor=Input(shape=(224, 224, 3)))
```

13. Then, we will construct the head of the model that will be placed on top of the base model

```
# base model
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)
```

14. Next, we will place the head model on top of the base model to be our actual model that we will train.

```
model = Model(inputs=baseModel.input, outputs=headModel)
```

15. After that, we will compile our model using Adam optimizer.

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

16. Then, we will use the fit function to train the full model.

```
Head = 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)
```

17. While training, we can make predictions on the testing set, and display a classification report of the training process.
-

```
predIdxs = model.predict(testX, batch_size=BS)
predIdxs = np.argmax(predIdxs, axis=1)

# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,
                           target_names=lb.classes_))

# save the detection model
print("[INFO] saving mask detector model...")
model.save("model-3.model", save_format="h5")
```

18. Finally, we can print the training and validation loss and accuracy of our model
-

```
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["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig("plot.png")
```

19. We can use the model to test it on the Jetson Xavier NX without training. This can be done by copying the model to the Jetson, and use the testing code developed to test the model on the Jetson Xavier NX.
20. Before testing the model on the Jetson, we need to install tensorflow and all the necessary packages for testing the model. See: [Installing Tensorflow on Jetson Xavier NX](#)
21. Connect the Logitech c920 camera to the Jetson NX and run the testing code using python3.

22. In the testing code, we import the necessary packages to test our detection model.

```
from tensorflow.keras.models import load_model
from detector import detect_mask
import cv2
import os
```

23. Next, we load the face detector model from disk.

```
print("[INFO] loading face detector model...")
prototxtPath = "deploy.prototxt"
weightsPath = "res10_300x300_ssd_iter_140000.caffemodel"
face_detector = cv2.dnn.readNet(prototxtPath, weightsPath)
```

24. Also, we load the face mask detector model from disk.

```
print("[INFO] loading face mask detector model...")
mask_detector_model_path = "ciw60.h5"
mask_detector = load_model(mask_detector_model_path)
```

25. Next, we Initialize the video stream.

```
capture = cv2.VideoCapture(0, cv2.CAP_DSHOW)
```

26. Then, we loop over the frames from the video stream, and apply the face detector and face mask detector to the frame using the imported function.

```
while True:
    # Grab the frame from the threaded video stream
    flags, frame = capture.read()

    # Detect faces in the frame and determine if they are wearing
    a face mask or not
    detect_mask(frame, face_detector, mask_detector, 0.8)

    # Show the output frame
    cv2.imshow("Frame", frame)
```

27. Now inside the `detect_mask` function we will declare the labels and colors representing them then load the input image which is the frame from the camera construct a blob from it for the face detector

```

def detect_mask(img, face_detector, mask_detector,
confidence_threshold, image_show=True):
    # Initialize the labels and colors for bounding boxes
    global currentIncorrect, currentWithout, thread1, thread2,
previousIncorrect, previousWithout

    labels = ["Correctly worn mask", "Incorrectly worn mask", "Without mask"]
    colors = [(0, 255, 0), (0, 255, 255), (0, 0, 255)]

    # Load the input image from disk, clone it, and grab the image
    spatial dimensions
    (h, w) = img.shape[:2]

    # Construct a blob from the image
    blob = cv2.dnn.blobFromImage(img, 1.0, (300, 300), (104.0, 177.0, 123.0))

```

28. The blob then gets passed to the face detector and we extract the detections from it and loop through them and saving their confidence level in a variable while later will get compared to a threshold of 0.8 (80%).

```

# Pass the blob through the network and obtain the face detections
face_detector.setInput(blob)
detections = face_detector.forward()

# Loop over the detections
for i in range(0, detections.shape[2]):
    # Extract the confidence (i.e., probability) associated
    with the detection
    confidence = detections[0, 0, i, 2]

```

29. If the confidence level is higher than the threshold then we get x,y coordinates of that face and make sure its within the frame and save that cropped face into a variable.

```

# Filter out weak detections by ensuring the confidence is
greater than the minimum confidence
if confidence > confidence_threshold:
    # Compute the (x, y)-coordinates of the bounding box for the object
    box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
    (start_x, start_y, end_x, end_y) = box.astype("int")

    # Ensure the bounding boxes fall within the dimensions of the frame
    (start_x, start_y) = (max(0, start_x), max(0, start_y))
    (end_x, end_y) = (min(w - 1, end_x), min(h - 1, end_y))

```

```

# Extract the face ROI, convert it from BGR to RGB channel
ordering, resize it to 224x224, and preprocess it
face = img[start_y:end_y, start_x:end_x]

```

30. Now we need to preprocess the face for the face mask detector model by changing its color to RGB and resize it into 224 pixels wide and tall and turn it into an array and now get the prediction and labels to determine the color to use.
-

```

face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
face = cv2.resize(face, (224, 224))
face = img_to_array(face)
face = preprocess_input(face)
face = np.expand_dims(face, axis=0)

# Pass the face through the model to determine if the face has a mask or not
prediction = mask_detector.predict(face)[0]
label_idx = np.argmax(prediction)

# Determine the class label and color we'll use to draw
the bounding box and text
label = labels[label_idx]
color = colors[label_idx]

```

31. We have to keep track the number of people wearing the mask incorrectly or without mask so every time the confidence for the label incorrect or without is given we will increase one to the variable respectively. Then check if the currentIncorrect or Without is higher than the previous incorrect or without mask and if so then we start a thread for playing audio if the thread isn't already playing and take a screenshot
-

```

if label == "Incorrectly worn mask":
    currentIncorrect = currentIncorrect + 1
elif label == "Without mask":
    currentWithout = currentWithout + 1

if currentIncorrect > previousIncorrect:
    if thread1.is_alive():
        False
    else:
        thread1 = myThread()
        thread1.start()

```



```

        i += 1
        cv2.imwrite(os.path.join(FilePath, 'Frame' + str(i) + '.jpg'), frame)

if currentWithout > previousWithout:
    if thread2.is_alive():
        False
    else:
        thread2 = myThread2()
        thread2.start()
    i += 1
    cv2.imwrite(os.path.join(FilePath, 'Frame' + str(i) + '.jpg'), frame)

```

32. Finally, The threads will start the following functions then we will add the confidence to the label and draw a rectangle over the picture and make the previous incorrect and without equal the current and reset the current for the next frame.
-

```

class myThread (threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
    def run(self):
        playsound.playsound('incArabic.mp3', True)
        playsound.playsound('incEng.mp3', True)

class myThread2 (threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
    def run(self):
        playsound.playsound('noArabic.mp3', True)
        playsound.playsound('noEng.mp3', True)

# Include the probability in the label
label = "{}: {:.2f}%".format(label, max(prediction) * 100)

# Display the label and bounding box rectangle on the output frame
cv2.putText(img, label, (start_x, start_y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
cv2.rectangle(img, (start_x, start_y), (end_x, end_y), color, 2)

previousWithout = currentWithout
previousIncorrect = currentIncorrect
currentWithout = 0
currentIncorrect = 0

```

- **Expected Results:** Accurate and unbiased mask detection model that can detect people wearing masks and not from reasonable distances and angles of multiple people in a frame and correctly labeling them.
- **Observed Results:** We observed that the mask detection model is accurate, unbiased and can detect multiple people wearing face masks and not. The model also detects from reasonable distances using live stream camera.
- **Acceptance Criteria:** The mask detection model can detect face masks with the two labels (Correct and Incorrect) and different colors (Blue, black, white and pink) with high accuracy from acceptable distances of multiple people's faces.
- **Test Results:** The face mask detection model was able to satisfy our criteria.

The following figures will show the accuracy of our face mask detector model using two categories. The figures include different variations of people wearing face masks correctly and not.

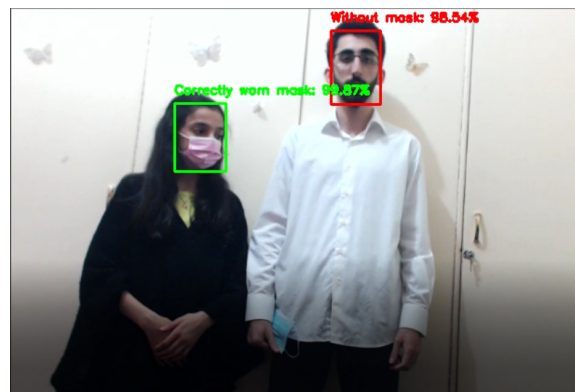


FIGURE 3.8: Face mask detection using tensorflow and openCV on one person without face mask and the other with



FIGURE 3.9: Face mask detection using tensorflow and openCV on two people without face masks

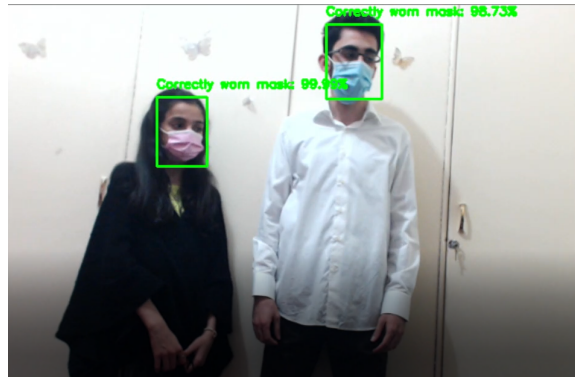


FIGURE 3.10: Face mask detection using tensorflow and openCV on two people with face masks

3.1.3 Test 3: Face mask detection using tensorflow and openCV on three categories

- **Test Description:** The main difference between this test and the previous test is that we increased the number of labels, so instead of only detecting people with masks and no masks, we will detect people who are wearing masks correctly, incorrectly and no masks at all.
- **Test Steps:**
 1. Create a dataset that includes three different files each contain images of people wearing masks correctly, incorrectly and no mask at all.
 2. Modify the training code mentioned in the previous test (Test 2) to make it able to train three categories.
 3. Train the detection model that contains three different categories and monitor the training process as shown in Figure 3.14.
 4. Save the detection model and training process plot as shown in Figure 3.15.
 5. Include the detection model in the testing python code.
 6. Modify the testing code mentioned in Test 2.
 7. Run the testing code on the Jetson Xavier NX.
- **Expected Results:** Accurate and unbiased mask detection model that can detect people wearing masks correctly, incorrectly and people without masks from reasonable distances and angles of multiple people in a frame and correctly labeling them.

- **Observed Results:** We observed that the mask detection model is quite accurate, unbiased and can detect multiple people wearing face masks correctly, incorrectly and people without face masks. The model also detects from reasonable distances using live stream camera. However, the model can get confused when it detects people wearing face masks correctly and under the nose by a bit.
- **Acceptance Criteria:** The mask detection model can detect face masks with the three labels (Correct, Incorrect and Without) and different colors (Blue, black, white and pink) with 95% accuracy from acceptable distances of multiple people's faces.
- **Test Results:** The face mask detection model was able to satisfy our criteria with acceptable accuracy.

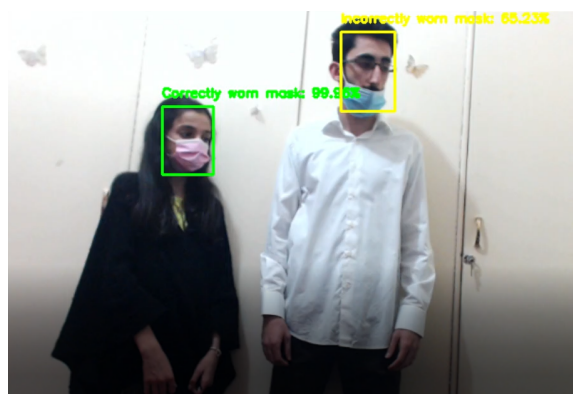


FIGURE 3.11: Face mask detection using tensorflow and openCV on two people (one wearing face mask correctly and the other is not)

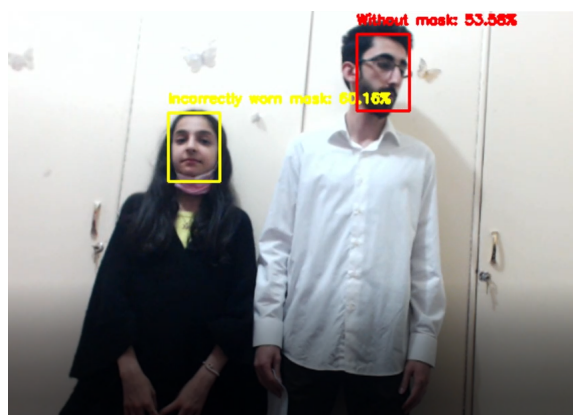


FIGURE 3.12: Face mask detection using tensorflow and openCV on two people (one wearing face mask incorrectly and the other without a mask)

people wearing face masks incorrectly, and alerting them using bluetooth speaker connected to the Nvidia Jetson Xavier NX

- **Test Steps:**

1. Edit the testing code we developed in Test 3.
2. Add code lines to make the live stream camera capture the frames of people not wearing face masks correctly, and save those images in a folder on the Jetson disk.

```

if wearingIncorrectly and runOnce == False:
    cv2.imwrite(os.path.join(FilePath, 'Frame' + str(i)
                                + '.jpg'), frame)

    i = i + 1
    runOnce = True

```

3. Make a threading class and define a run function in it that includes the alert sounds that we want people wearing masks incorrectly to hear.

```

class myThread (threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
    def run(self):
        playsound.playsound('arabic.mp3', True)
        playsound.playsound('english.mp3', True)

```

4. Start the voice alert thread when the amount of people wearing face masks incorrectly shown on the frame is at least 1.
5. Keep the voice alert thread on till all people on the frame wear their face masks correctly.

- **Expected Results:** We expect the camera to capture people who are not wearing their face masks correctly and save it on the specified file path on the Jetson, and we expect the bluetooth speaker to keep alerting people who are not wearing their face masks correctly.

- **Observed Results:** We observed that people who wore their face masks correctly then removed it got their images taken, and people who newly entered the camera frame got their images taken. Also, if someone is in the frame not wearing their face mask correctly and a new person enters the frame also wearing their face mask

incorrectly, an image will be taken. As for the bluetooth speaker, it kept alerting all people in the frame that are not wearing their face masks correctly.

- **Acceptance Criteria:** The camera should capture people who are not wearing their face masks correctly and save it, but it should not keep capturing each frame which will keep saving unnecessary images. The bluetooth model should keep alerting people not wearing their face masks correctly or not wearing them at all until all people on the frame wear their masks correctly.
- **Test Results:** The testing results matched our criteria, and we were able to capture images of people not wearing face masks, or wearing them incorrectly; and we were able to make the bluetooth speaker keep alerting people (both in English and Arabic) who are wearing face masks incorrectly (or not wearing them at all) until they wear their face masks (or wear them correctly).

3.1.5 Test 5: Jetson Xavier and Turtlebot3 operating time

- **Test Description:** In this test, we will make the Jetson Xavier NX and Turtlebot3 run at maximum performance using Li-ion 4.2V 18650 (for the Jetson) and Lithium polymer 11.1V 1800mAh (for the Turtlebot3) batteries to see how long can the overall system operate before running out of charge.
- **Test Steps:**
 1. Connect 3 Li-ion 4.2V (max charged) to the Jetson Xavier NX using an adapter.
 2. Run the code for face mask detection.
 3. Connect the Lithium polymer 11.1V (max charged) to the Turtlebot3.
 4. Run the python code using SLAM for autonomous navigation.
 5. Make the robot navigate at 70% of the maximum speed since running the robot at 100% makes the camera jitters.
 6. Set a timer as soon as the system starts to run.
 7. Wait for the Jetson Xavier NX and Turtlbot3 to run out of charge.
 8. Stop the timer when one of the systems stops responding.
 9. Calculate the period the system was running for.

- **Expected Results:** The overall system is expected to last more than an hour (hour at minimum) since each battery contains 8.15Wh per hour which means 3 batteries will have a total of 24.5Wh, and the Jetson Xavier NX needs 15W to operate at maximum performance (6 CPU cores). As for the Turtlebot3, it uses Lithium polymer 11.1V 1800mAh which supplies a total of 19.98Wh per hour which means it should make the Turtlebot3 run for 2.5 hours according to the official site of TurtleBot3. [Specifications of the TurtleBot3 Burger](#)
- **Observed Results:** The Jetson Xavier NX lasted 3 hours before running out of charge, and the Turtlebot3 lasted 2 hours before running out of charge.
- **Acceptance Criteria:** The overall system can last 2 hours which is an acceptable period of time for the robot to navigate autonomously around the IoT lab. To extend this period, we can increase the capacity of the batteries.
- **Test Results:** The mask detection system can run for 3 hours, and the autonomous navigation can run for 2 hours. These are acceptable times for Lab sessions in the IoT lab.

3.1.6 Test 6: Face mask detection model accuracy on different face mask colors

- **Test Description:** In this test, we will test our tensorflow2 and openCV face mask detection model accuracy and performance on different face mask colors to evaluate the accuracy.
- **Test Steps:**
 1. Prepare different mask colors for testing. (Blue, Black, White and Pink)
 2. Start a recording software to record this test.
 3. Run the testing code on the Jetson Xavier NX.
 4. Wear each mask in front of the live stream camera.
 5. Wear each mask correctly and incorrectly in different face angles and distances.
 6. End the recording and observe the results.

- **Expected Results:** The face mask detection model is expected to detect face masks correctly and incorrectly with high accuracy despite the differences of face mask colors since our data set included various face mask colors to avoid any biases.
- **Observed Results:** The face mask detector model was able to detect face masks with different colors with it's correct labels from different face angles and distances with high accuracy despite the color black which means our data set lacks a bit of images where people wear black face masks incorrectly and correctly.
- **Acceptance Criteria:** The model should be able to detect face masks with different colors with different face angles and distances with high accuracy.
- **Test Results:** The test results matched our acceptance criteria, and the detection model was able to detect different face mask colors from different angles and distances at high accuracy as shown in the following figures.

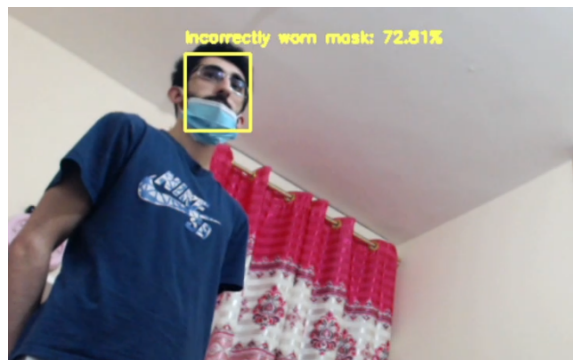


FIGURE 3.16: Wearing blue face mask incorrectly with an accuracy of 72.81%

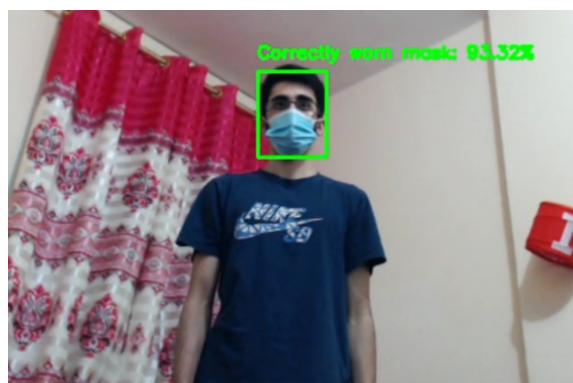


FIGURE 3.17: Wearing blue face mask correctly with an accuracy of 93.32%

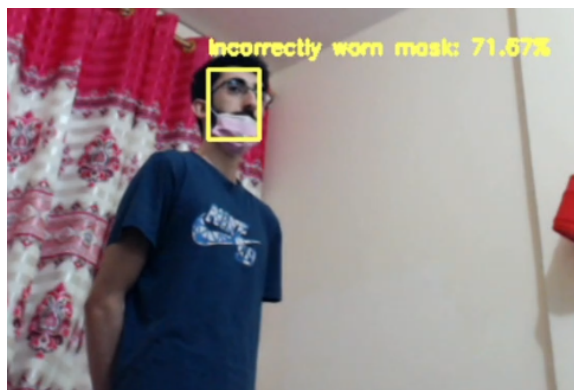


FIGURE 3.18: Wearing pink face mask incorrectly with an accuracy of 71.67%

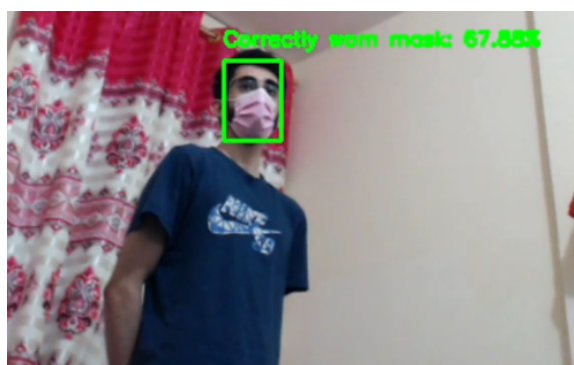


FIGURE 3.19: Wearing pink face mask correctly with an accuracy of 67.88%

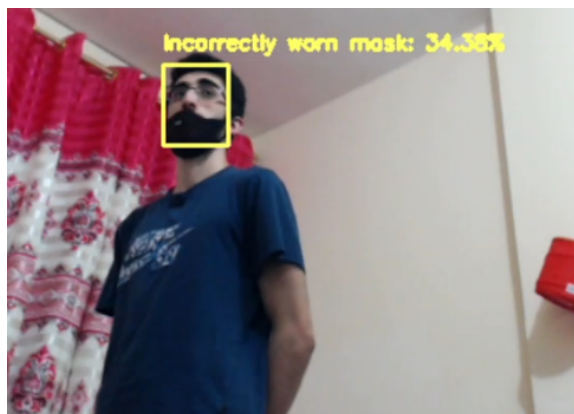


FIGURE 3.20: Wearing black face mask incorrectly with an accuracy of 34.38%

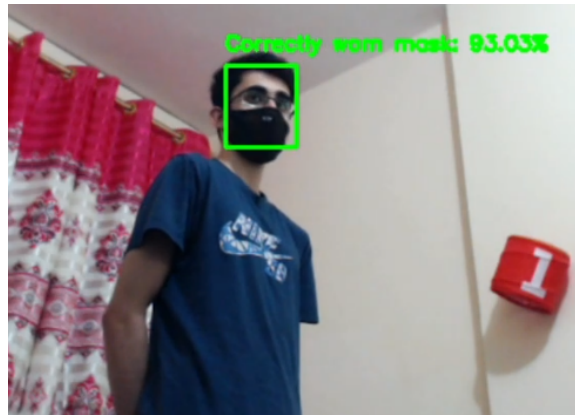


FIGURE 3.21: Wearing black face mask correctly with an accuracy of 93.03%

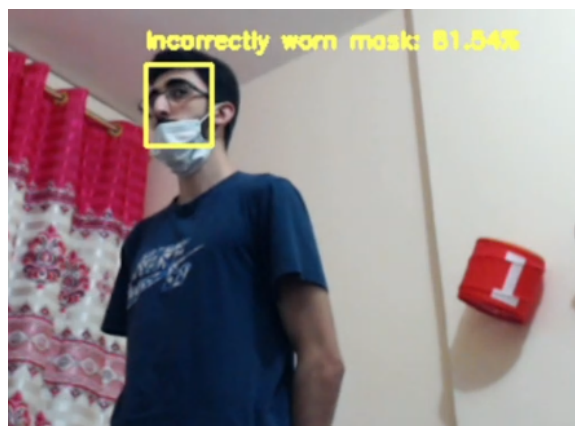


FIGURE 3.22: Wearing white face mask incorrectly with an accuracy of 81.54%

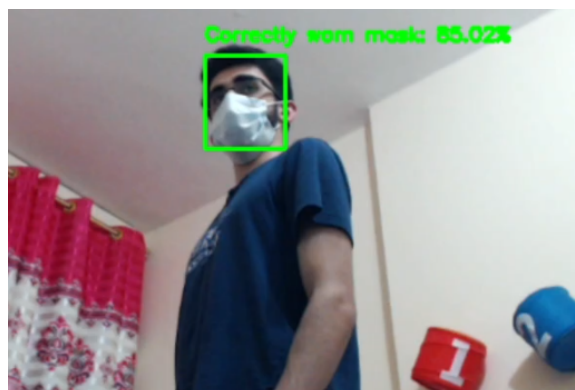


FIGURE 3.23: Wearing white face mask correctly with an accuracy of 83.02%

3.1.7 Test 7: Autonomous navigation using ROS

- **Test Description:** In this test, we used ROS, SLAM and code manipulation to make the TurtleBot3 run autonomously on an explored environment. For this

project, we used the IoT lab at the ECBE department in Abu Dhabi University as our environment.

- **Test Steps:**

1. Install ROS melodic on remote PC and all the dependant packages of ROS melodic 1 and Turtlebot3. [PC setup](#)
2. Configure IP address of the remote PC and save it on bashrc script.
3. Install TurtleBot3 melodic image, and flash it to the microSD card.
4. Boot the Rpi 3B+ using the microSD card and do the ROS1 network configuration. [ROS1 Network Configuration](#)
5. Install openCR and all the necessary packages on the Rpi 3B+. [OpenCR Setup](#)
6. Open SLAM node on remote PC as master, and run TurtleBot3 bring up on the Rpi 3B+.
7. Launch SLAM node on remote PC and control the TurtleBot3 using teleoperation to navigate through the unexplored environment (IoT Lab).
8. Start exploring and drawing the map as shown in Figure 3.24.

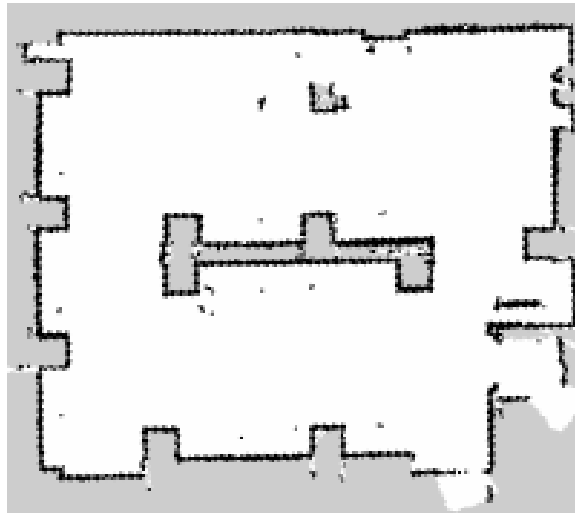


FIGURE 3.24: Drawing the IoT Lab in Abu Dhabi University using SLAM

9. Once we are done, we saved the map on remote PC.
10. Next, we can use pose estimation and goal setting to estimate the current position of the TurtleBot3 on the map and start setting goals (points on the

map) to make the robot move autonomously and avoid any obstacles on the way within the range of the LiDAR sensor.

11. Finally, we can use code manipulation using python to make the robot go to random points for as long as we want.
- **Expected Results:** The robot moves to the points we set (Total of 4 points in form of a rectangle that covers the whole IoT Lab) while avoiding obstacles on the way.
 - **Observed Results:** The robot was able to follow the goals set while avoiding any obstacles on the range of the LIDAR sensor.
 - **Acceptance Criteria:** The robot can autonomously navigate throughout the whole lab in a uniform way while avoiding any obstacles in range of the LIDAR sensor.
 - **Test Results:** The test results matched our acceptance criteria, and the robot was able to autonomously navigate through the IoT Lab while avoiding any obstacles on the way as shown in Figure 3.25.



FIGURE 3.25: Autonomous navigation and obstacle avoidance using TurtleBot3 in the IoT Lab

3.2 Results and Discussion

In this section, we will discuss all the results obtained from the previous chapter and analyze them in addition to unmentioned variables in the testing stage that could possibly affect the face detection model's accuracy and performance.

3.2.1 Data set size

Different data set sizes were used to train our face mask detection model using tensorflow and openCV, and what we want to conclude is how data set size can affect the accuracy of our detection model. So, different data sets were created that include two (correct, incorrect) and three categories (correct, incorrect, without); the data sets created are the following:

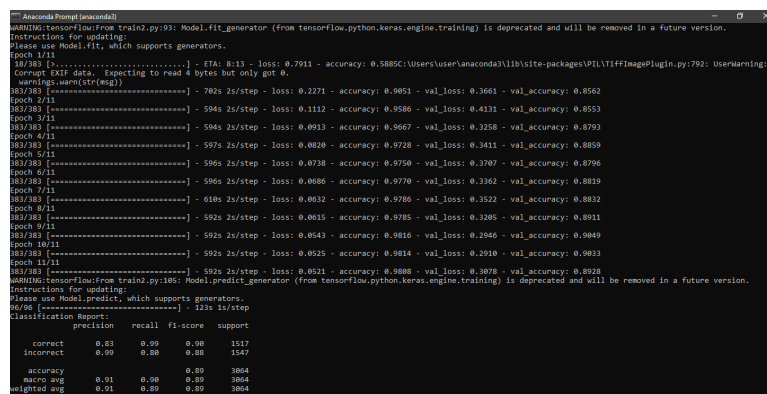
- Data set 1:
 1. Number of labels: 2 (correct, incorrect)
 2. Images per label: 7,600 images
- Data set 2:
 1. Number of labels: 2 (correct, incorrect)
 2. Images per label: 4,600 images
- Data set 3:
 1. Number of labels: 2 (correct, incorrect)
 2. Images per label: 2,200 images
- Data set 4:
 1. Number of labels: 3 (correct, incorrect, without)
 2. Images per label: 2,200 images
- Data set 5:
 1. Number of labels: 3 (correct, incorrect, without)
 2. Images per label: 1,000 images

3.2.1.1 Data set 1

After creating these data sets, we trained the model on a fixed number of epochs (11), batch size (32) and learning rate (0.0001) to make sure that the only variable in this experiment is the data set size. The results will show the training time, average precision

and recall accuracy, as well as, validation loss and accuracy at each epoch.

In Data set 1, the data set size was 7,600 images per label which means the total of data set 1 is about 15,200 images; these images will be split into training, validation and testing (70% for training, 10% for validation and 20% for testing). Each epoch, all of the training data set will be shuffled and passed through the model which will start learning from the training data and make predictions on the validation, and the model's predictions are represented as validation accuracy and loss in each epoch. Mostly, the training and validation accuracy should increase by each epoch since the model will start learning more and predict better. In cases where the data is insufficient, bias or misleading could make the model predict worse, and the user can see that the validation accuracy is decrease and the validation loss is increasing. In addition, there are other cases where the model's training accuracy is increasing while the validation accuracy is decreasing, and the validation loss is increasing; this means that the model is overfitting, and what overfitting means is that the model is memorizing the training data instead of learning it which leads to poor performance and accuracy when it comes to predicting new data that the model has not seen before. On the contrary, underfitting occurs when the model does not have enough training data to learn from it and start predicting new data (testing data) accurately. Furthermore, the results of training the Data set 1 model can be shown in Figure 3.26 and Figure 3.27.



```

WARNING:tensorflow:From train2.py:93: Model.fit_generator (from tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/11
183/183 [=====] - ETA: 81s - loss: 0.7911 - accuracy: 0.5885C:\Users\user\anaconda3\lib\site-packages\PIL\tiffimagePlugin.py:792: UserWarning:
Corrupt EXIF data. Expecting to read 4 bytes but only got 0.
WARNING:warn(STP(msg))
183/183 [=====] - 782s 2s/step - loss: 0.2271 - accuracy: 0.9851 - val_loss: 0.3661 - val_accuracy: 0.8562
Epoch 2/11
183/183 [=====] - 594s 2s/step - loss: 0.1112 - accuracy: 0.9586 - val_loss: 0.4131 - val_accuracy: 0.8553
Epoch 3/11
183/183 [=====] - 594s 2s/step - loss: 0.0913 - accuracy: 0.9667 - val_loss: 0.3258 - val_accuracy: 0.8793
Epoch 4/11
183/183 [=====] - 597s 2s/step - loss: 0.0810 - accuracy: 0.9728 - val_loss: 0.3411 - val_accuracy: 0.8859
Epoch 5/11
183/183 [=====] - 586s 2s/step - loss: 0.0738 - accuracy: 0.9758 - val_loss: 0.3707 - val_accuracy: 0.8796
Epoch 6/11
183/183 [=====] - 586s 2s/step - loss: 0.0686 - accuracy: 0.9770 - val_loss: 0.3362 - val_accuracy: 0.8819
Epoch 7/11
183/183 [=====] - 618s 2s/step - loss: 0.0632 - accuracy: 0.9786 - val_loss: 0.3522 - val_accuracy: 0.8832
Epoch 8/11
183/183 [=====] - 592s 2s/step - loss: 0.0615 - accuracy: 0.9785 - val_loss: 0.3285 - val_accuracy: 0.8911
Epoch 9/11
183/183 [=====] - 592s 2s/step - loss: 0.0543 - accuracy: 0.9816 - val_loss: 0.2946 - val_accuracy: 0.9049
Epoch 10/11
183/183 [=====] - 592s 2s/step - loss: 0.0525 - accuracy: 0.9814 - val_loss: 0.2910 - val_accuracy: 0.9033
Epoch 11/11
183/183 [=====] - 592s 2s/step - loss: 0.0521 - accuracy: 0.9808 - val_loss: 0.3078 - val_accuracy: 0.8928
WARNING:tensorflow:From train2.py:105: Model.predict_generator (from tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.predict, which supports generators.
00/00 [=====] - 123s 1s/step
Classification Report:
      precision    recall  f1-score   support
 correct      0.83      0.99      0.90      1517
 incorrect    0.99      0.88      0.93      1547
 accuracy                    0.89      3064
 macro avg      0.91      0.98      0.89      3064
 weighted avg    0.91      0.89      0.89      3064

```

FIGURE 3.26: Training process and classification report for Data set 1

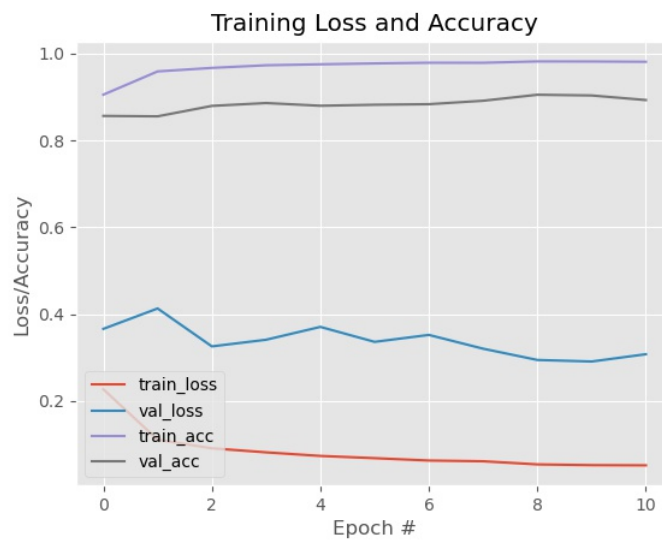


FIGURE 3.27: Training loss and accuracy plot for Data set 1

One important thing we need to consider is that numbers are numbers meaning that the best proof of accuracy is by testing the model not by only judging the training process of the model. It is common that some models show high accuracy in training but poor performance in testing. The testing results for data set 1 model can be shown in Figure 3.28 and Figure 3.29.

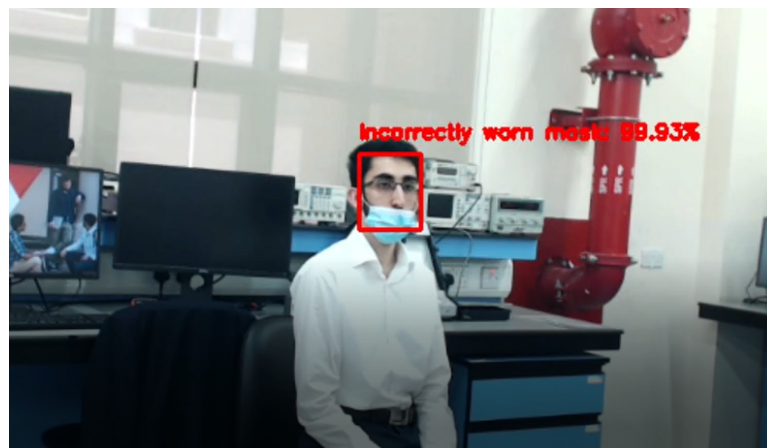


FIGURE 3.28: Face mask detection model accuracy of Wearing face mask incorrectly using Data set 1

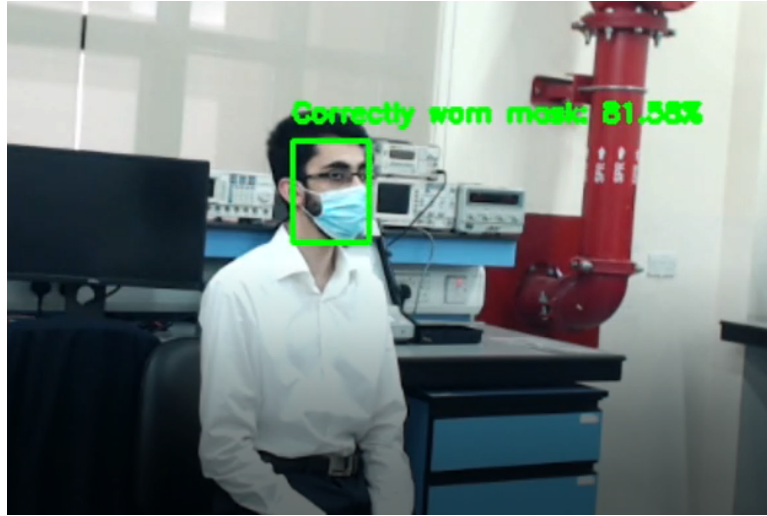


FIGURE 3.29: Face mask detection model accuracy of Wearing face mask incorrectly using Data set 1

3.2.1.2 Data set 2

In Data set 2, we have reduced the total number of training data to see if there is a relation between data set size, accuracy and performance. It can be seen in 3.30 that the model's average accuracy is more than Data set 1 model's accuracy. Why training more images does not always mean better accuracy? Although this is true for most data sets, our data set depends more on the variations of our images and how easy it can be separated linearly and differentiated by the model; this means that in Data set 1, the model faced a hard time separating face masks worn correctly and incorrectly despite the large number of images provided. In addition, does larger data set size mean overfitting? Not really; Why? because there are many cross-validation methods that can be used in order to prevent the model from overfitting despite the size of the data set used; cross-validation is a technique used in machine learning to prevent overfitting by dividing the data set into partitions; this way the model will not learn from the entire data set. Moreover, the model will divide the data set into training, testing and validation; and it can also divide the training data into folds (K-fold). How does this help? instead of making the model learn all the data set, it will learn from new data partitions each epoch which means the model will see new images each round, and this will allow the model to learn and predict more accurately on new data. Moreover, the cross-validation method we used for our face mask detection model is called the Hold-out method; this method is considered to be the simplest and most efficient method used for

cross-validation on large data sets, and what this method does is that it divides the data set into training, testing and validation (Usually 70% of the data set goes for training, 10% for validation and 20% for testing); the model will learn from the training partition and start practicing by making predictions on the validation partition, and the model will calculate the loss and accuracy for both training and validation data which will help it learn and predict more accurately during the next epochs (rounds). The training results for data set 2 can be shown in Figure 3.30 and Figure 3.31

```

Anascondi Prompt (anaconda3)
2021-08-13 21:27:59.048082: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1263]
[INFO] compiling model...
[INFO] training head...
WARNING:tensorflow:From train2.py:93: Model.fit_generator (from tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/11
227/227 [=====] - 445s 2s/step - loss: 0.2617 - accuracy: 0.8868 - val_loss: 0.2604 - val_accuracy: 0.8962
Epoch 2/11
227/227 [=====] - 336s 1s/step - loss: 0.1267 - accuracy: 0.9543 - val_loss: 0.2666 - val_accuracy: 0.8968
Epoch 3/11
227/227 [=====] - 334s 1s/step - loss: 0.0968 - accuracy: 0.9638 - val_loss: 0.2373 - val_accuracy: 0.9096
Epoch 4/11
227/227 [=====] - 334s 1s/step - loss: 0.0777 - accuracy: 0.9726 - val_loss: 0.2826 - val_accuracy: 0.8912
Epoch 5/11
227/227 [=====] - 335s 1s/step - loss: 0.0769 - accuracy: 0.9745 - val_loss: 0.2170 - val_accuracy: 0.9202
Epoch 6/11
227/227 [=====] - 334s 1s/step - loss: 0.0662 - accuracy: 0.9762 - val_loss: 0.1966 - val_accuracy: 0.9263
Epoch 7/11
227/227 [=====] - 334s 1s/step - loss: 0.0638 - accuracy: 0.9782 - val_loss: 0.1917 - val_accuracy: 0.9291
Epoch 8/11
227/227 [=====] - 333s 8s/step - loss: 0.0598 - accuracy: 0.9762 - val_loss: 0.1987 - val_accuracy: 0.9275
Epoch 9/11
227/227 [=====] - 335s 1s/step - loss: 0.0578 - accuracy: 0.9798 - val_loss: 0.1879 - val_accuracy: 0.9288
Epoch 10/11
227/227 [=====] - 334s 1s/step - loss: 0.0554 - accuracy: 0.9811 - val_loss: 0.2347 - val_accuracy: 0.9262
Epoch 11/11
227/227 [=====] - 334s 1s/step - loss: 0.0539 - accuracy: 0.9821 - val_loss: 0.2513 - val_accuracy: 0.9879
WARNING:tensorflow:From train2.py:185: Model.predict_generator (from tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.predict, which supports generators.
57/57 [=====] - 74s 1s/step
Classification Report
precision    recall  f1-score   support
 correct      0.85    0.99    0.92    895
 incorrect    0.99    0.83    0.90    927
 accuracy                    0.91    1822
 micro avg    0.92    0.91    0.91    1822
 weighted avg 0.92    0.91    0.91    1822

```

FIGURE 3.30: Training process and classification report for Data set 2

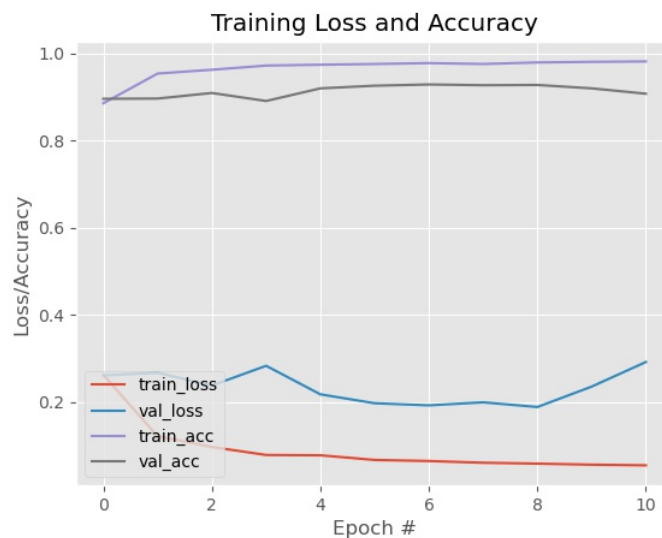


FIGURE 3.31: Training loss and accuracy plot for Data set 2

Although training and validation accuracy in Data set 2 is better than Data set 1, the testing results show otherwise; and this is due to the reduction in the data size which removed many images that could have improved the accuracy of our model, which made

the training process easier to classify due to the little variations in the data set. The testing results for data set 2 model can be shown in Figure 3.32 and Figure 3.33.



FIGURE 3.32: Face mask detection model accuracy of Wearing face mask incorrectly using Data set 2



FIGURE 3.33: Face mask detection model accuracy of Wearing face mask correctly using Data set 2

3.2.1.3 Data set 3

In data set 3, we reduced the two label data set even more; this time we made it 2,200 images per label which means about 4,400 in total. The same issue raises when we reduce the number of images in our data set, which is having higher accuracy in training and validation as shown in Figure 3.34 and Figure 3.35 while the accuracy in testing is much lower which can be seen in Figure 3.36 and Figure 3.37.

```

Anaconda Prompt (anaconda)
[INFO] compiling model...
[INFO] training head...
WARNING:tensorflow:From train2.py:93: Model.fit_generator (from tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/11
5/15/15 [====...] - ETA: 2:24 - loss: 0.7713 - accuracy: 0.5972C:\Users\user\anaconda3\lib\site-packages\IPython\terminal\plugins.py:792: UserWarning:
Corrupt EXIF data. Expecting to read 4 bytes but only got 0.
WARNING:warn(str(msg))
Epoch 2/11
15/15/15 [====...] - 273s 25/step - loss: 0.5571 - accuracy: 0.7301 - val_loss: 0.3379 - val_accuracy: 0.8672
Epoch 3/11
15/15/15 [====...] - 233s 25/step - loss: 0.3513 - accuracy: 0.8548 - val_loss: 0.2808 - val_accuracy: 0.8739
Epoch 4/11
15/15/15 [====...] - 238s 25/step - loss: 0.2932 - accuracy: 0.8723 - val_loss: 0.2328 - val_accuracy: 0.9029
Epoch 5/11
15/15/15 [====...] - 230s 25/step - loss: 0.2723 - accuracy: 0.8784 - val_loss: 0.2126 - val_accuracy: 0.9107
Epoch 6/11
15/15/15 [====...] - 231s 25/step - loss: 0.2447 - accuracy: 0.8967 - val_loss: 0.2137 - val_accuracy: 0.9141
Epoch 7/11
15/15/15 [====...] - 238s 25/step - loss: 0.2296 - accuracy: 0.9052 - val_loss: 0.1934 - val_accuracy: 0.9238
Epoch 8/11
15/15/15 [====...] - 233s 25/step - loss: 0.2257 - accuracy: 0.9044 - val_loss: 0.1989 - val_accuracy: 0.9252
Epoch 9/11
15/15/15 [====...] - 276s 25/step - loss: 0.2088 - accuracy: 0.9175 - val_loss: 0.1885 - val_accuracy: 0.9338
Epoch 10/11
15/15/15 [====...] - 296s 25/step - loss: 0.2066 - accuracy: 0.9162 - val_loss: 0.1964 - val_accuracy: 0.9219
Epoch 11/11
15/15/15 [====...] - 282s 25/step - loss: 0.1908 - accuracy: 0.9195 - val_loss: 0.2022 - val_accuracy: 0.9185
Epoch 11/11
15/15/15 [====...] - 235s 25/step - loss: 0.1882 - accuracy: 0.9258 - val_loss: 0.1761 - val_accuracy: 0.9342
WARNING:tensorflow:From train2.py:105: Model.predict_generator (from tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.predict, which supports generators.
10/20 [====...] - 38s 1s/step
Classification Report:
              precision    recall  f1-score   support
 correct      0.94      0.91      0.93      447
 incorrect    0.29      0.94      0.53      472
 accuracy          0.93
 macro avg      0.93      0.93      0.93      919
 weighted avg   0.93      0.93      0.93      919

```

FIGURE 3.34: Training process and classification report for Data set 3

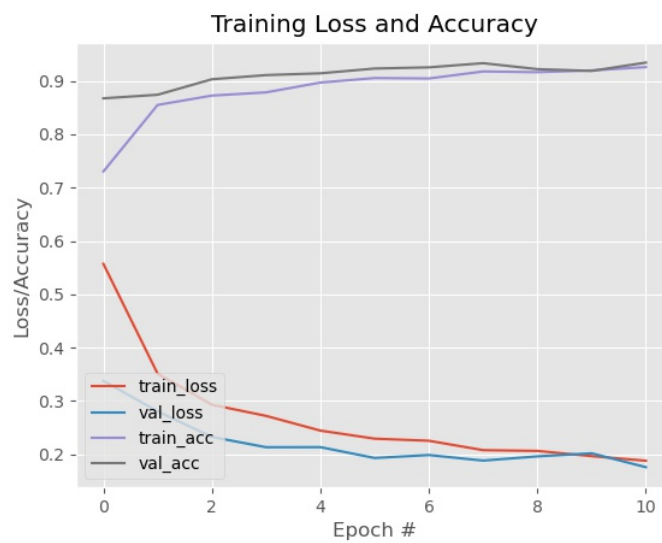


FIGURE 3.35: Training loss and accuracy plot for Data set 3

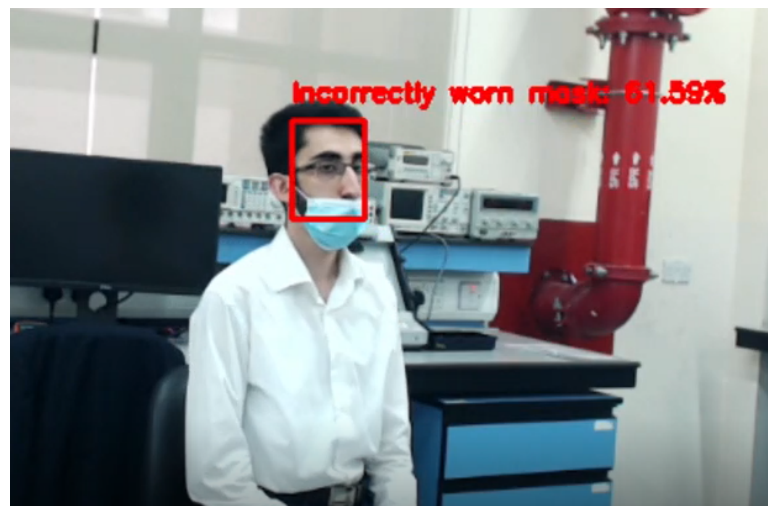


FIGURE 3.36: Face mask detection model accuracy of Wearing face mask incorrectly using Data set 3



FIGURE 3.37: Face mask detection model accuracy of Wearing face mask correctly using Data set 3

It can be noted from the previous figures that the accuracy of this model is 10% to 30% lower than the previous models (data set 1 and 2) when it comes to detecting from the same distances and face angles; this is due to the low data size used for this model.

3.2.1.4 Data set 4

In data set 4, we increased number of labels to three and separated the data set into three categories instead of two. The labels now are ("Correct", "Incorrect" and "Without"); this is an improvement to the previous data sets since now we will be able to classify people wearing face masks correctly, incorrectly and people without face masks at all. The number of images used in this data set is 2,200 per label which gives us a total of 6,600 images. Increasing the data set size comes with a cost; the more data we use the more RAM the training process consumes whether we are training on CPU or GPU. The training process for data set 4 can be shown in Figure 3.38 and Figure 3.39 while the accuracy in testing is much lower which can be seen in Figure 3.40 and Figure 3.41.

```

Anacode Prompt (anaconda)
[INFO] training head...
WARNING:tensorflow:From train.py:111: Model.fit_generator (from tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/11
100/167 [=====] - ETA: 2:01 - loss: 0.8311 - accuracy: 0.6438;Users\user\anaconda3\lib\site-packages\PIL\tiffimageplugin.py:792: UserWarning:
Consume XIF data: Expecting to read 4 bytes but only got 0.
warnings.warn(STP(msg))
Epoch 2/11
100/167 [=====] - 3099 25/step - loss: 0.6963 - accuracy: 0.7091 - val_loss: 0.2958 - val_accuracy: 0.9078
Epoch 3/11
100/167 [=====] - 3188 25/step - loss: 0.4826 - accuracy: 0.8478 - val_loss: 0.2166 - val_accuracy: 0.9314
Epoch 4/11
100/167 [=====] - 3188 25/step - loss: 0.3426 - accuracy: 0.8727 - val_loss: 0.1890 - val_accuracy: 0.9337
Epoch 5/11
100/167 [=====] - 3188 25/step - loss: 0.2936 - accuracy: 0.8920 - val_loss: 0.1962 - val_accuracy: 0.9329
Epoch 6/11
100/167 [=====] - 3174 25/step - loss: 0.2645 - accuracy: 0.9060 - val_loss: 0.1670 - val_accuracy: 0.9352
Epoch 7/11
100/167 [=====] - 3188 25/step - loss: 0.2521 - accuracy: 0.9066 - val_loss: 0.1718 - val_accuracy: 0.9421
Epoch 8/11
100/167 [=====] - 3188 25/step - loss: 0.2451 - accuracy: 0.9083 - val_loss: 0.1738 - val_accuracy: 0.9352
Epoch 9/11
100/167 [=====] - 3174 25/step - loss: 0.2363 - accuracy: 0.9118 - val_loss: 0.1579 - val_accuracy: 0.9489
Epoch 10/11
100/167 [=====] - 3548 25/step - loss: 0.2074 - accuracy: 0.9203 - val_loss: 0.1743 - val_accuracy: 0.9444
Epoch 11/11
100/167 [=====] - 3174 25/step - loss: 0.2089 - accuracy: 0.9210 - val_loss: 0.1724 - val_accuracy: 0.9428
100/167 [=====] - 3174 25/step - loss: 0.2083 - accuracy: 0.9201 - val_loss: 0.1719 - val_accuracy: 0.9413
WARNING:tensorflow:From train.py:122: Model.predict_generator (from tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.predict, which supports generators.
Classification Report
-----
precision    recall  f1-score   support
incorrect    0.95    0.91    0.93    447
without      0.90    0.95    0.90    438
correct      0.94    0.98    0.96    457
accuracy          0.95    1342
macro avg    0.95    0.95    0.95    1342
weighted avg  0.95    0.95    0.95    1342

```

FIGURE 3.38: Training loss and accuracy plot for Data set 4

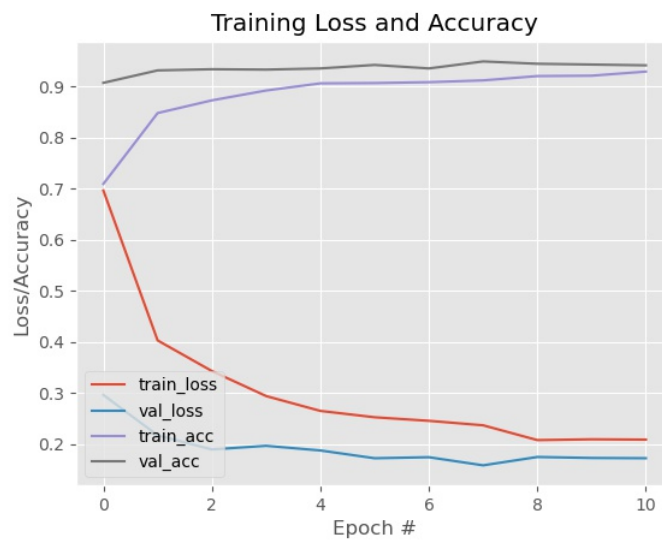


FIGURE 3.39: Training process and classification report for Data set 5

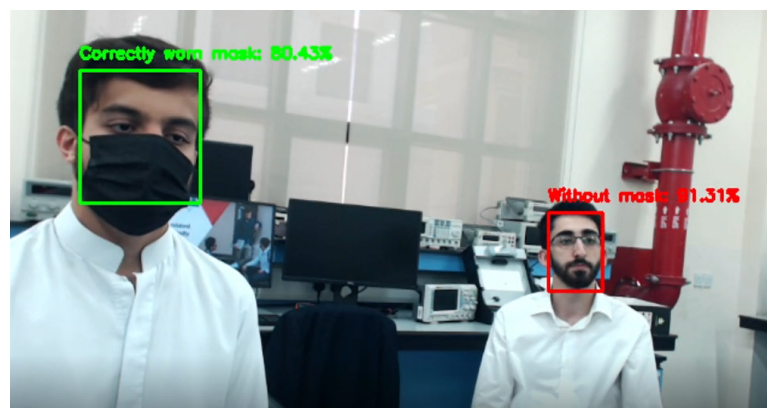


FIGURE 3.40: Face mask detection model accuracy of Wearing face mask correctly and without using Data set 4

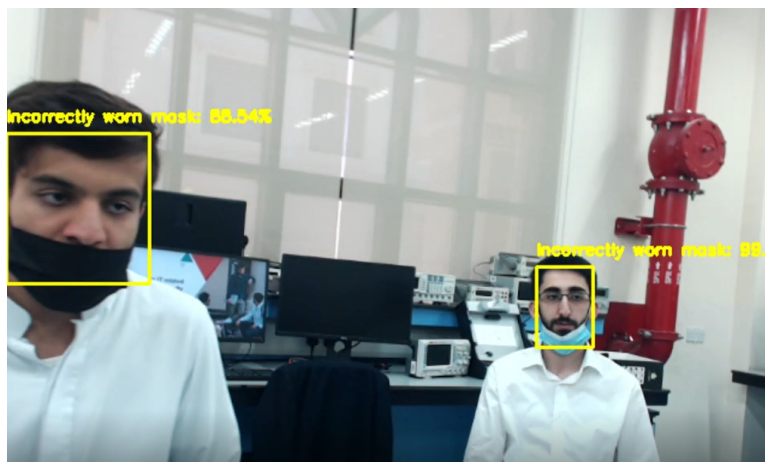


FIGURE 3.41: Face mask detection model accuracy of Wearing face mask incorrectly using Data set 4

3.2.1.5 Data set 5

In data set 5, we reduced the data set size from 2,200 to 1,000 images per label which gives us a total of 3,000 images. In the training process, the average accuracy was lower than the average accuracy in data set 4 by 1% which is not a significant difference, but this can only judge by testing the model. The training process for data set 5 can be shown in Figure 3.42 and Figure 3.43, and the results of testing our model using data set 5 can be shown in Figure 3.44, Figure 3.45 and Figure 3.46.

```

Anaconda Prompt (anaconda)
Instructions for updating:
Please use Model.fit, which supports generators.
epoch 1/11
WARNING:tensorflow:..... - ETA: 1:37 - loss: 1.1046 - accuracy: 0.4550c:\Users\user\anaconda3\lib\site-packages\PIL\tiffImagePlugin.py:792: UserWarning: C
server: DPIX data. Expecting to read 4 bytes but only got 0.
warnings.warn(str(msg))
7/77 [.....] - 178s 2s/step - loss: 0.9864 - accuracy: 0.6698 - val_loss: 0.4384 - val_accuracy: 0.8766
epoch 2/11
7/77 [.....] - 156s 2s/step - loss: 0.5472 - accuracy: 0.7852 - val_loss: 0.3386 - val_accuracy: 0.8799
epoch 3/11
7/77 [.....] - 155s 2s/step - loss: 0.4487 - accuracy: 0.8180 - val_loss: 0.2768 - val_accuracy: 0.9095
epoch 4/11
7/77 [.....] - 155s 2s/step - loss: 0.4131 - accuracy: 0.8381 - val_loss: 0.2528 - val_accuracy: 0.9112
epoch 5/11
7/77 [.....] - 155s 2s/step - loss: 0.3528 - accuracy: 0.8689 - val_loss: 0.2685 - val_accuracy: 0.9178
epoch 6/11
7/77 [.....] - 155s 2s/step - loss: 0.3362 - accuracy: 0.8758 - val_loss: 0.1907 - val_accuracy: 0.9359
epoch 7/11
7/77 [.....] - 155s 2s/step - loss: 0.2965 - accuracy: 0.8898 - val_loss: 0.2807 - val_accuracy: 0.9424
epoch 8/11
7/77 [.....] - 155s 2s/step - loss: 0.3883 - accuracy: 0.8889 - val_loss: 0.2156 - val_accuracy: 0.9391
epoch 9/11
7/77 [.....] - 156s 2s/step - loss: 0.2823 - accuracy: 0.8943 - val_loss: 0.2383 - val_accuracy: 0.9194
epoch 10/11
7/77 [.....] - 155s 2s/step - loss: 0.2603 - accuracy: 0.9025 - val_loss: 0.2851 - val_accuracy: 0.9326
epoch 11/11
7/77 [.....] - 154s 2s/step - loss: 0.2492 - accuracy: 0.9187 - val_loss: 0.2884 - val_accuracy: 0.9389
WARNING:tensorflow:from train.py:122: Model.predict_generator (from tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.predict, which supports generators.
10/28 [.....] - 29s 1s/step
Classification Report:
          precision    recall  f1-score   support
 correct      0.93      0.91      0.92      281
 incorrect    0.96      0.92      0.94      212
 without      0.91      0.98      0.94      383
 accuracy          0.94
 macro avg        0.94      0.94      0.93      616
 weighted avg     0.94      0.94      0.93      616

```

FIGURE 3.42: Training process and classification report for Data set 5

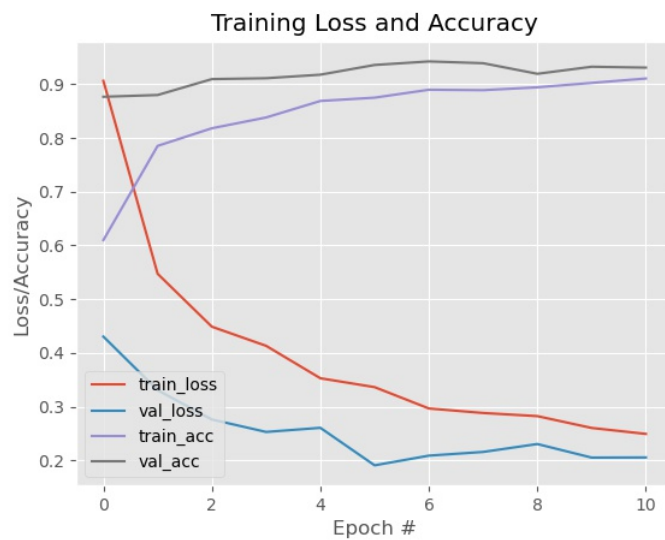


FIGURE 3.43: Training loss and accuracy plot for Data set 5



FIGURE 3.44: Face mask detection model accuracy of Wearing face mask incorrectly and correctly using Data set 5

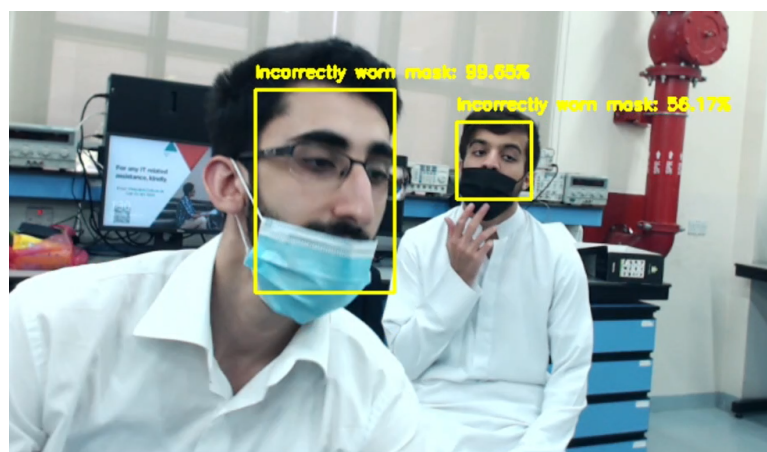


FIGURE 3.45: Face mask detection model accuracy of Wearing face mask incorrectly using Data set 5



FIGURE 3.46: Face mask detection model accuracy of Wearing face mask correctly and without using Data set 5

It can be noted from the previous figures that the accuracy of this model is a bit lower (1% – 20%) than the previous model (data set 4) when it comes to detecting from different distances and face angles.

3.2.2 Number of epochs and Batch size

The number of epochs determines the number of complete passes through the training data set. For our project, we used different numbers of epochs and found that 11 is enough to provide accurate results. Reducing the number of epochs can lead to underfitting since the model will not have enough time to learn, and increasing the number of epochs can lead to overfitting since the model will keep learning the same data for long periods. The batch size is the number of samples processed before the model is updated, and having a batch size of 32, 64 or 128 is considered to be common. If we choose a batch size of 128, the model will take the first 128 images from the data set and start training them; once the model finishes training the first 128 images, it will take the another 128 images and so on. Shuffle may apply to help countering overfitting which means the model will randomly take 128 images from the data set each time.

3.2.3 Distance and Face angle

Another important factor related to the performance of our model is how well it can detect people wearing face masks correctly or not at long distance and different face

angles. For instance, make the model detect from distances over 4 m will not be possible in our case since the face detector will not be able to detect any faces in the frames due to the small size of the face at long distance, and what we can do is either increasing the zoom of the camera, getting a camera with better megapixels and exposure which will make people's faces appear clearer on the frame, or making a new face detector that could detect faces from long distances. In addition, face angles could confuse the detection model in some cases which makes it challenging for the model to classify correctly. As an example, consider the case in Figure 3.47 where it can be clearly seen that the person is wearing his face mask incorrectly, but the model predicts that the person is wearing his face mask correctly due to the face angle of that person.

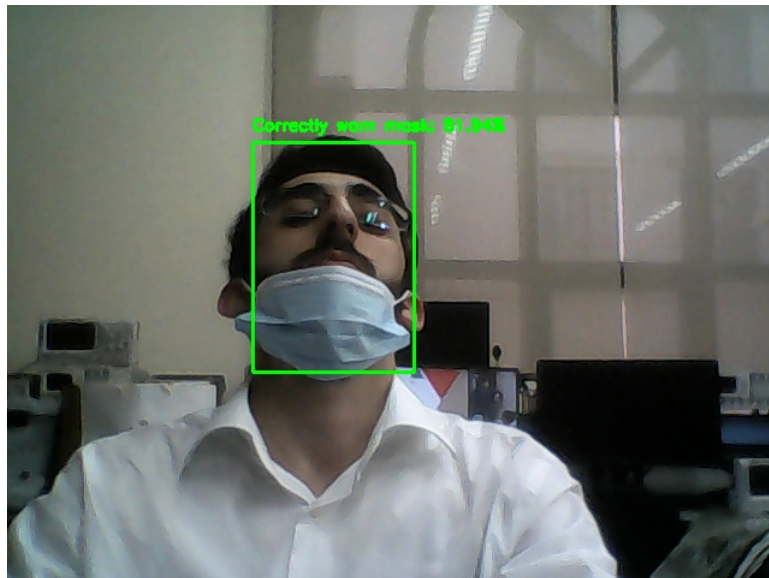


FIGURE 3.47: Incorrect classification form the model due to the angle of the face

3.3 Analysis and Interpretation of Data

3.3.1 Face mask detection model evaluation

In this section, we will evaluate our face mask detection model; and based on this evaluation, it will come clear whether our model is truly detecting face masks with high accuracy and precision.

3.3.1.1 Metrics for performance evaluation

Our main focus will be on the predictive capability of our model which means precision or recall is more important than how fast it takes to classify people in the frame.

To describe the performance of our model, we simply construct a confusion matrix. The confusion matrix can be shown in Figure 3.48

		PREDICTED CLASS	
		Class = Correct	Class = Incorrect
ACTUAL CLASS	Class = Correct	a (TP)	b (FN)
	Class = Incorrect	c (FP)	d (TN)

TP: True Positive
 FP: False Positive
 TN: True Negative
 FN: False Negative

$$\text{False positive rate} = \frac{c}{c + d}$$

$$\text{Precision (p)} = \frac{a}{a + c}$$

$$\text{True positive rate} = \frac{a}{a + b}$$

$$\text{Recall (r)} = \frac{a}{a + b}$$

FIGURE 3.48: Confusion matrix along with the most widely-used metrics [15]

Two other metrics that are often used to quantify model performance are precision and recall.

- **Precision** is defined as the number of true positives divided by the total number of positive predictions. It quantifies how correct our model's positive predictions were (in our case, positives mean correctly worn masks and negatives mean incorrectly worn masks). This means that the more the model correctly classifies the label "correctly worn masks" the more precise it will be.
- **Recall/Sensitivity** is defined as the number of true positives divided by the total number of true positives and false negatives (all actual positives). It quantifies out of all people that are wearing their face masks correctly, how many did the model (AI) miss?

In this project, precision is more important than recall because false positive is expensive; meaning if the AI predicted that the person is wearing his/her face mask correctly while he/she is not, that person will get away with wearing his/her face mask incorrectly and be the reason of spreading the disease much further.

3.3.1.2 Model evaluation and comparison

In this sub section, we will evaluate the best model we have come up with, and then try to come up with a better performance model that will achieve higher precision than the previous model. To evaluate our model, we will have to construct an ROC (Receiver Operating Characteristic) curve which plots the TPR (True Positive Rate) vs. the FPR (False Positive Rate). To construct an ROC curve, we need to:

1. Use the face mask detection model to produce a probability of correctly worn masks ($P(\text{correctly worn masks})$) in each frame captured from the live stream camera. The total number of test instances (frames) that will be used are 100.
2. Sort the instances according to the $P(\text{correctly worn masks})$ in decreasing order.
3. Apply threshold at each unique value of $P(\text{correctly worn masks})$ and count the number of TP, FP, TN, FN at each threshold.

Following the previous steps, we will be able to construct a table that will help us into constructing an ROC curve for our model. The first 29 instances of the table we constructed for the base model can be shown in Table 3.49

#	P(+)	True Class	Prediction	TP	FP	TPR (Recall)	FPR	Precision
1	100	+	+	1	0	0.018	0	1.000
2	100	+	+	2	0	0.036	0	1.000
3	100	+	+	3	0	0.054	0	1.000
4	100	+	+	4	0	0.071	0	1.000
5	100	+	+	5	0	0.089	0	1.000
6	100	+	+	6	0	0.107	0	1.000
7	99.99	-	+	6	1	0.107	0.023	0.857
8	99.99	+	+	7	1	0.125	0.023	0.875
9	99.98	+	+	8	1	0.143	0.023	0.889
10	99.97	+	+	9	1	0.161	0.023	0.900
11	99.97	+	+	10	1	0.179	0.023	0.909
12	99.95	+	+	11	1	0.196	0.023	0.917
13	99.94	+	+	12	1	0.214	0.023	0.923
14	99.93	+	+	13	1	0.232	0.023	0.929
15	99.91	-	+	13	2	0.232	0.045	0.867
16	99.89	-	+	13	3	0.232	0.068	0.813
17	99.88	+	+	14	3	0.250	0.068	0.824
18	99.87	+	+	15	3	0.268	0.068	0.833
19	99.85	+	+	16	3	0.286	0.068	0.842
20	99.83	+	+	17	3	0.304	0.068	0.850
21	99.82	+	+	18	3	0.321	0.068	0.857
22	99.76	+	+	19	3	0.339	0.068	0.864
23	99.76	-	+	19	4	0.339	0.091	0.826
24	99.75	-	+	19	5	0.339	0.114	0.792
25	99.73	+	+	20	5	0.357	0.114	0.800
26	99.65	-	+	20	6	0.357	0.136	0.769
27	99.61	+	+	21	6	0.375	0.136	0.778
28	99.57	-	+	21	7	0.375	0.159	0.750
29	99.56	+	+	22	7	0.393	0.159	0.759

FIGURE 3.49: Testing instances up to 29 frames, Probability of correctly worn masks, True class, Prediction, TP, FP, TPR, FPR, Recall and Precision for the base model

The ROC curve constructed from the table shown in 3.49 of the base model can be shown in Figure 3.50

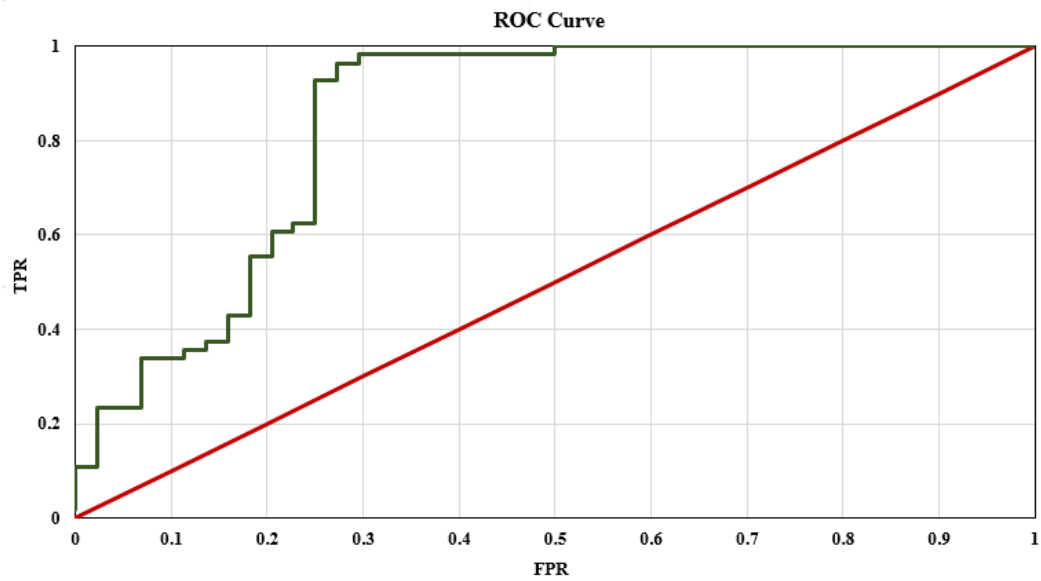


FIGURE 3.50: ROC curve of the base model shown in green line with the default classifier shown in red

To understand how we judge classifiers based on the ROC curve, we can take a look at Figure 3.51; where it shows that the more the curve is leaning towards the True Positive Rate (TPR) the more precise and accurate it is.

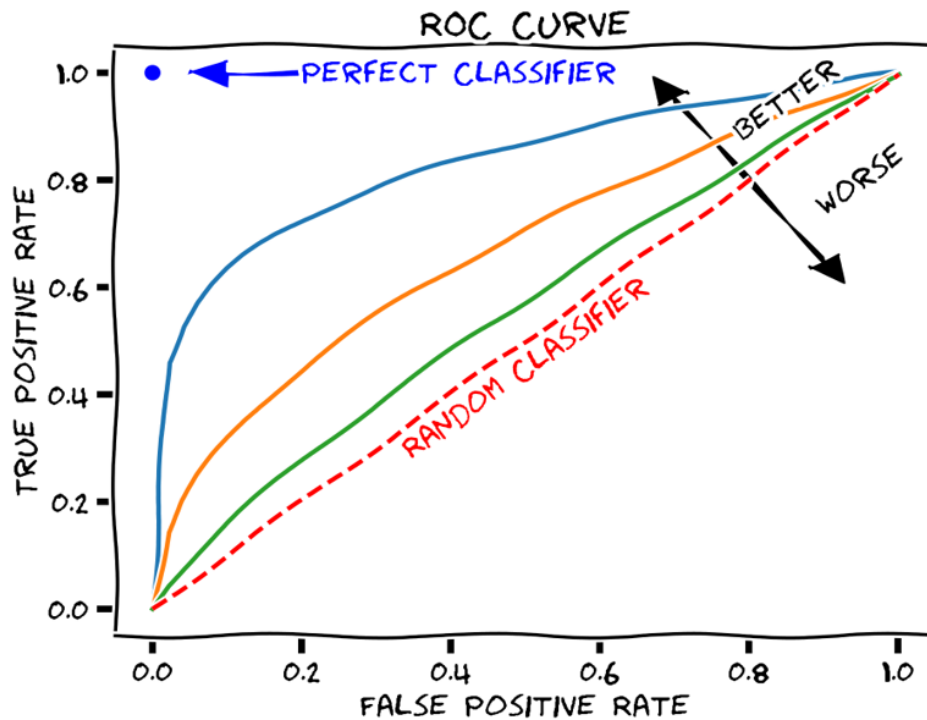


FIGURE 3.51: ROC Interpretation [15]

Next, we will try to construct a better curve by enhancing the performance of our face mask detection model. To do that, we need to:

- Increase the number of images that show people wearing face masks correctly and not in 90 degrees.
- Split the 3 label model into 3 models each contain a dominant label; meaning model 1 will contain more images of the label "Correctly worn mask" than the other labels which are "Incorrectly worn mask" and "Without face mask". This way, each model will make its own prediction on the incoming frame, and the model with the highest prediction probability will be chosen to make the prediction on that frame and label it.

The new model (better performance model) was trained and then tested on 100 different frames which will be used as testing instances for constructing an ROC curve. The table constructed for the better performance model can be shown in Figure 3.52

#	P(+)	True Class	Prediction	TP (+ +)	FP (- +)	TPR (Recall)	FPR	Precision
1	99.99	+	+	1	0	0.015	0.000	1.000
2	99.97	+	+	2	0	0.029	0.000	1.000
3	99.93	+	+	3	0	0.044	0.000	1.000
4	99.92	+	+	4	0	0.059	0.000	1.000
5	99.92	+	+	5	0	0.074	0.000	1.000
6	99.91	+	+	6	0	0.088	0.000	1.000
7	99.91	+	+	7	0	0.103	0.000	0.857
8	99.88	+	+	8	0	0.118	0.000	0.875
9	99.85	+	+	9	0	0.132	0.000	0.889
10	99.85	+	+	10	0	0.147	0.000	0.900
11	99.81	+	+	11	0	0.162	0.000	0.909
12	99.75	+	+	12	0	0.176	0.000	0.917
13	99.71	+	+	13	0	0.191	0.000	0.923
14	99.71	-	+	13	1	0.191	0.031	0.929
15	99.7	+	+	14	1	0.206	0.031	0.933
16	99.7	+	+	15	1	0.221	0.031	0.938
17	99.69	+	+	16	1	0.235	0.031	0.941
18	99.65	+	+	17	1	0.250	0.031	0.944
19	99.63	+	+	18	1	0.265	0.031	0.947
20	99.62	+	+	19	1	0.279	0.031	0.950
21	99.57	+	+	20	1	0.294	0.031	0.952
22	99.51	+	+	21	1	0.309	0.031	0.955
23	99.36	+	+	22	1	0.324	0.031	0.957
24	99.12	+	+	23	1	0.338	0.031	0.958
25	99.12	+	+	24	1	0.353	0.031	0.960
26	99.08	+	+	25	1	0.368	0.031	0.962
27	99	+	+	26	1	0.382	0.031	0.963
28	98.93	+	+	27	1	0.397	0.031	0.964
29	98.91	+	+	28	1	0.412	0.031	0.966
30	98.86	+	+	29	1	0.426	0.031	0.967

FIGURE 3.52: Testing instances up to 30 frames, Probability of correctly worn masks, True class, Prediction, TP, FP, TPR, FPR, Recall and Precision for the better performance model

The ROC curve constructed from the table shown in Figure 3.52 can be shown in Figure 3.53.

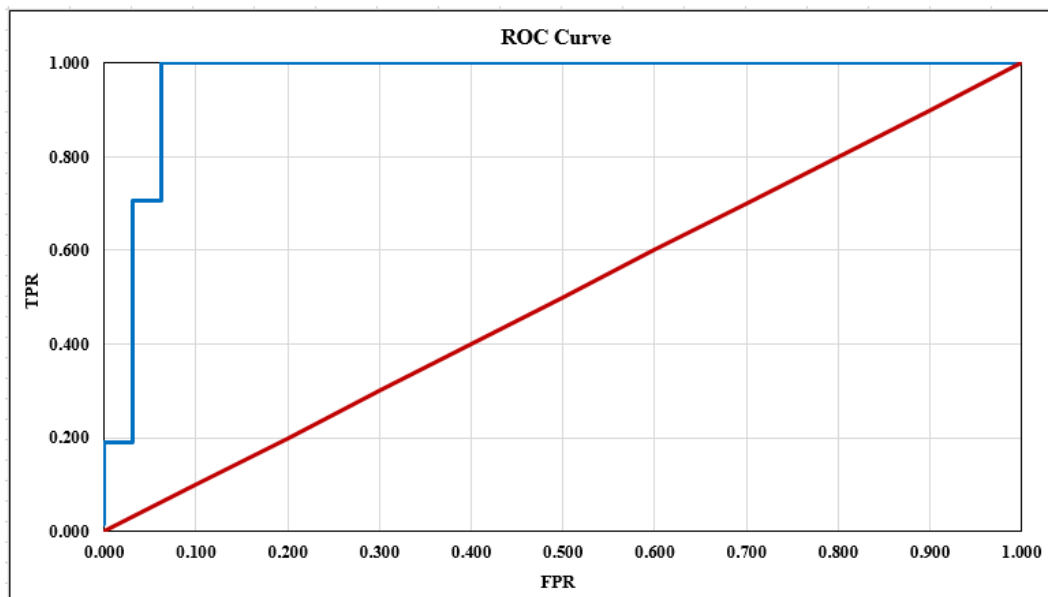


FIGURE 3.53: ROC curve of the better performance model shown in blue line with the default classifier shown in red

As can be shown in Figure 3.53, the line of the better performance model covers more

area under the graph which means it has more precision than the base model shown in Figure 3.50.

To understand how the better performance model gave a better ROC curve than the base model, we summarized the main differences in Table 3.1

The final ROC curve that shows both base model and better performance model curves can be shown in Figure 3.54 for comparison.

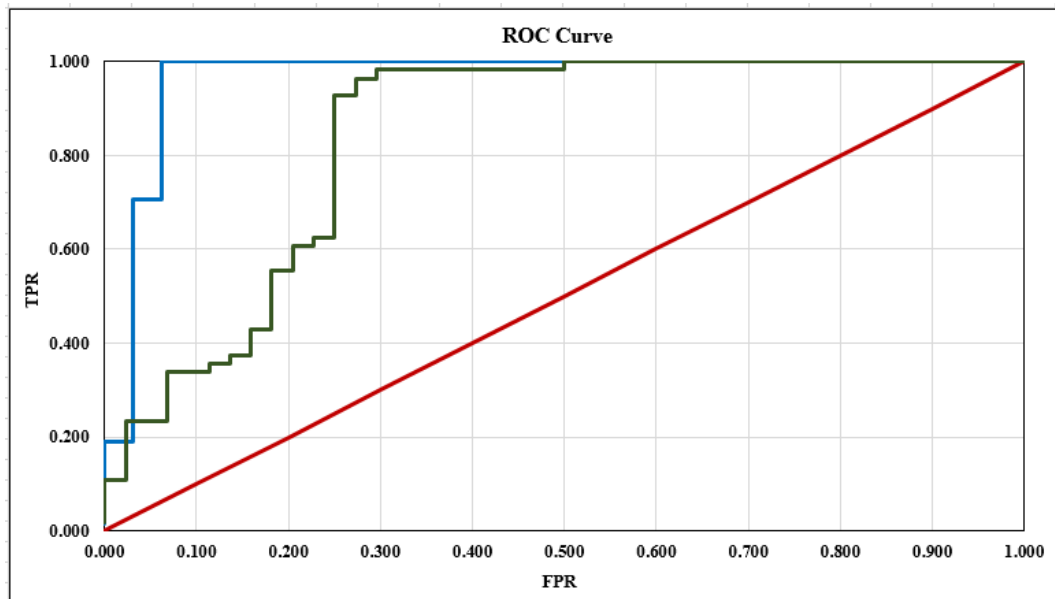


FIGURE 3.54: ROC curve of the better performance model shown in blue line, base model shown in green line and the default classifier shown in red

3.3.2 Summary of Analysis and Interpretation of Data

To summarize with, data set size, number of epochs, batch size, distance and face angle have different effects on accuracy of the detection model. Table 3.2 shows the summary of our discussion and analysis section.

Differences	Base model	Better performance model
Number of models	1 Model	3 Models
Labels per model	Correct, Incorrect and Without	Model 1: (Correct, Other) Model 2: (Incorrect, Other) Model 3: (Without, Other)
Dataset size	10,000 images	12,000 images
Image distribution	2,500 images per label (Equal distribution)	50% for label x, 25% for label y, and 25% for label z. (x, y, and z differs based on the model being trained)
Percentage of 0-degree face images in each label	~85%	~60%
Percentage of 90-degree face images in each label	~5%	~20%
No. of epochs	11 epoch	14 epoch
Batch size	32	128
Data augmentation	Yes	Yes
Face Detector Threshold	0.50	0.50
Mask Detector Threshold (between 0.50 and 1)	Picks the label with its color with the highest accuracy in the model	Picks the model that gives the highest accuracy between the 3 models then pick the label with its color with the highest accuracy in that model
Avg. accuracy per label while facing the camera with a 0-degree	Correctly worn mask: 90~100% Incorrectly worn mask: 70~100% Without mask: 95~100%	Correctly worn mask: 85~100% Incorrectly worn mask: 80~100% Without mask: 95~100%
Avg. accuracy per label while facing the camera with a 90-degree	Correctly worn mask: 65~100% Incorrectly worn mask: 50~100% Without mask: 80~100%	Correctly worn mask: 70~100% Incorrectly worn mask: 65~100% Without mask: 90~100%
Avg. Precision	0.775	0.914
Avg. Recall	0.654	0.654

TABLE 3.1: Base model Vs. Better performance model

	Two category model (Correct, Incorrect)	Three category model (Correct, Incorrect, Without)
Dataset size (per category)	7,600 images per category Avg. training accuracy: 91% Actual accuracy: 80 - 99% 4,600 images per category Avg. training accuracy: 92% Avg. testing accuracy: 70 - 99% 2,200 images per category Avg. training accuracy: 93% Avg. testing accuracy: 55 - 99%	2,200 images per category Avg. training accuracy: 95% Avg. testing accuracy: 75 - 99% 1,000 images per category Avg. training accuracy: 94% Avg. testing accuracy: 50 - 99%
No. of epochs	Number(s): 11, 12, 13 Avg. training accuracy: 93% Underfitting: <8	Number(s): 11 Avg. training accuracy: 94% Underfitting: <8
Batch Size	Value(s): 32 Avg. training accuracy: 93% Value: 128 Avg. training accuracy: 93%	Value: 32 Avg. training accuracy: 94% Value: 128 Avg. training accuracy: 94%
Distance	Distance: 1 to 4m Avg. testing accuracy: 60 - 99% Distance: above 4m Avg. testing accuracy: 0% (Faces are not detected at distances above 4m)	Distance: 1 to 4m Avg. testing accuracy: 50 - 99% Distance: above 4m Avg. testing accuracy: 0%
Face angle	Facing front Avg. testing accuracy: 75 - 99% Facing left/right Avg. testing accuracy: 60 - 97% Facing up/down Avg. testing accuracy: 50 - 94%	Facing front Avg. testing accuracy: 70 - 99% Facing left/right Avg. testing accuracy: 55 - 92% Facing up/down Avg. testing accuracy: 50 - 90%

TABLE 3.2: Dataset size, No. of epochs, Batch size, Distance and Face angle effect on accuracy

In addition, the summary of the training process time, accuracy and model performance based on different categories, data sizes and distance can be shown in Table 3.3. We can conclude from Table 3.3 that the performance is still the same despite the data set size and distance; the model performance depends on the GPU used and camera. A high quality camera with better shutter speed and exposure can increase the performance significantly, and deploying the model on a more powerful GPU with high compatibility can also increase the performance.

	Results
Two Labels (Data set 1)	Total images: 15,200 Training time: 1 hour 40 min Detection distance (Jetson): $\leq 4\text{m}$ Performance (FPS): 6~9 fps
Two Labels (Data set 2)	Total images: 9,200 Training time: 1 hour 27 min Detection distance (Jetson): $\leq 4\text{m}$ Performance (FPS): 6~9 fps
Two Labels (Data set 3)	Total images: 4,400 Training time: 42 min Detection distance (Jetson): $\leq 4\text{m}$ Performance (FPS): 6~9 fps
Three Labels (Data set 4)	Total images: 6,600 Training time: 59 min Detection distance (Jetson): $\leq 4\text{m}$ Performance (FPS): 6~9 fps
Three Labels (Data set 5)	Total images: 3,000 Training time: 29 min Detection distance (Jetson): $\leq 4\text{m}$ Performance (FPS): 6~9 fps

TABLE 3.3: Summary of data set size, training time, distance and performance

Chapter 4

Conclusion

4.1 Summary

In this project, we proposed a self-driving mask detection robot as a solution to refrain people that wear their face masks improperly or not wear them at all. The dataset provided online contained various images that show people with different mask colors; it also includes images of people wearing face masks properly and improperly. The dataset was then preprocessed then trained using the TensorFlow and OpenCV on the Jetson Xavier. After we are done with the training, The Face mask detector was then used to test different images to prove that the detector is correctly labeling the images. Different labels were used in this paper including the “Correctly Wearing a Mask”, “Without Mask” and “Wearing a Mask Improperly” labels. The TensorFlow and OpenCV detector can then be deployed to the Nvidia Jetson Xavier NX, and with the Logitech camera attached to it, it can process the real-time video frame by frame and apply the Face Mask detector to label the frames. The labeled frames can then be used to extract the images where people wore the mask incorrectly which can be used for different purposes (mainly punishment). In Addition, The labeled frame will also be used to generate a voice to warn the person who is not wearing the mask properly or not wearing it at all. The turtlebot3 has an important part in our project since it can be used to navigate autonomously and avoid obstacles in many places. The robot can create a map using the 360 Lidar sensor which scans the area around the robot and draws a 2D map in the SLAM software. The robot can then navigate autonomously from different points through the created map.

For the testing and results, we did a total of 6 tests. The first 3 tests were about testing different face mask detection models using different algorithms. First, we built a face mask detection model using YOLOv2 on MATLAB. In this test we used the image labeler tool in Matlab to label our data set images into three categories, wearing a mask, incorrectly wearing a mask, and not wearing a mask. The YOLOv2 object detector was able to detect face masks from different angles with high accuracy. In the second test we used TensorFlow and OpenCV to build a face mask detection model for 2 labels; wearing a mask and not wearing a mask. We used an open-source data set that contains 14k images divided into 2 categories 7500 images of people wearing mask correctly and the second half contain images of people not wearing a mask or wearing it incorrectly. In this test, we used a pre-trained neural network model to detect faces first called [FaceNet](#) and then we applied the face mask detector model that we have trained using MobileNetV2 to be able to detect and label the image frames. We observed that the detector was able to detect faces and classify them as either wearing mask or not wearing a mask. The final model that we have built and the test was developing a face mask detector model that contains 3 labels using the method we used in test 2. We modified the python code to add a third label which is detecting people who are wearing the mask improperly. We obtained very good results, the model was able to function from different angles and distances with an accuracy of 95%. Next, We tested the Auto image capturing and Alert sound system when detecting people without a face mask or wearing their mask incorrectly. Furthermore, We did a test on both the Jetson Xavier and the TurtleBot3 to test their operating lifetimes. We observed that the Jetson can operate for up to 3 hours on its max performance when using 3 Li-on 4.2V batteries connected in series. However, for the TurtleBot3 it was able to last up to 2 hours using an 11.1V lithium battery with a capacity of 1800mAh. For the final test, We Tested the Autonomous navigation using ROS and SLAM to make the TurtleBot3 run autonomously in a closed environment. We observed that the robot was able to follow points that we set on the virtual map using SLAM and it was avoided all the obstacles that have been detected by the LIDAR sensor.

4.2 Future Improvements and Takeaways

Many future improvements can be done to our self-driving face detection robot. From the hardware point of view, we can improve the capacity of the batteries we used to extend the overall operating time of our system. In addition, we can get a high quality camera with better shutter speed, exposure and megapixels which will make face and mask detection easier to detect and more accurate. Also, we can add ultrasonic sensors to make the robot detect any small objects on the ground and avoid them while autonomously navigating the area since the Lidar sensor can miss small objects that are not in its range of vision. Moreover, making the robot stop when it detects faces can reduce camera blur and jitter, increase prediction accuracy and save power at the same time. Finally, we can use more powerful computers that could train very large number of data set sizes to increase the accuracy of our model even further.

4.3 Learned Lessons

In this project, We had a great experience in dealing with Machine learning and computer vision. Through this project we used different types of artificial intelligence algorithms such as using YOLO v2 in Matlab then we switch our algorithm to use Python libraries since they are easier to use and implement on single-chip computers. To start with, since we were more familiar with MATLAB and its computer vision tools, we went with it to implement our face mask detector YOLOv2 Model using the ImageLabeler tool from the computer vision toolbox in MATLAB. We got a very good result from it, but since we are using the Jetson Xavier as our supercomputer it has some limitations to use MATLAB on it. We learned from this experience, so we searched for an alternative and in this case, we found that python has many good and light libraries that can be used to achieve the same goal.

Moreover, we gained a good amount of knowledge through using new technologies. Since we went with python and its open-source libraries such as OpenCV and TensorFlow which provide us with many classes and pre-trained machine learning models that can be imported and used directly. We did a lot of research about how to implement this core part of our project(Face mask detection). Our main learning source was reading about previous similar projects and searching online. Fortunately, we found a tutorial

on youtube about how to train your model using MobileNetV2 from the TensorFlow library which is the main Model used in our project.

For the Self-driving Robot, we used the turtle bot 3 original documentation to install and configure the robot. Then, we followed a youtube tutorial on how to set up SLAM and draw the 2d MAP for a closed area by controlling the robot manually. In this part, we faced an issue where the robot was not following the command we are sending to control the robot and draw the map. We followed different tutorials and did much research until we found the issue and fixed it. In general, we faced many difficulties and challenges in this project, but we always learned and gained more experience from those failers which drove us to accomplish the main goal of this project.

4.4 Team Dynamics

All team members of this project are computer engineering students.

4.4.1 Team Members

Amer Barhoush: Amer is a hard working student and thrives under pressure. He always seek improvements and perfections no matter how small it is. He is quite knowledgeable in the AI, machine learning and computer vision field which helped us a lot in this project.

Omar Farag: Omar is an engaging student that always suggest new creative plans and ideas to improve the overall system design. He has both software and hardware skills that helped us throughout this project.

Sohrab Setoodeh: Sohrab is a team worker and creative designer; he can think of new ways to enhance the feasibility of the system and reduce the power consumption.

Zayed Aslam: Zayed is a positive and quick thinker student that can overcome many difficulties in short time; his analytical and critical thinking skills makes him think deeply into problems and solve them accordingly.

4.4.2 Tasks and Work Division

To have a good result we as a team worked as one hand everyone in the group helps each other and share every single experience we have in every field. We choice Amer to be the leader because he is a very justice and restrictive person. He divides every task equally and gives a suitable deadline. We divide work into two main tasks which are the autonomous driving robot and mask detection. Each two-member has taken on the main part but that not mean we didn't work on other tasks. Each member when he has a problem he posts on whats-app and we share our knowledge to solve the problem. However, Amer and Zayed take the mask detection part because there where has experience in the Artificial intelligent part as they to the AI course. Also, Omar has a very big fact that we had very excellent mask detection result because he improves Amer and Zayed model to work better. On the other hand, Sohrab and Omar work in building an autonomous driving robot as they know hardware and communication between it. The communication part also Amer help to had communication between raspberry and Jetson Xavier which make the robot be control by the main part of the robot and not using a computer.

4.4.3 Team Communication and Gantt Chart

The main factor of success is the communication between groups. We as a team create several groups to have different types of communication for example we make a group in Microsoft teams to had video call communication Also edit the code together by sharing the screen and We create a whats-app group to have a fast chat and exchange the experience. In addition, we create another group that includes the capstone member and our Doctors to show the progress of the project and have some advice. On the other hand, we had a lot of meeting in a different place such as the university to test our robot and in one member's house to build some parts.

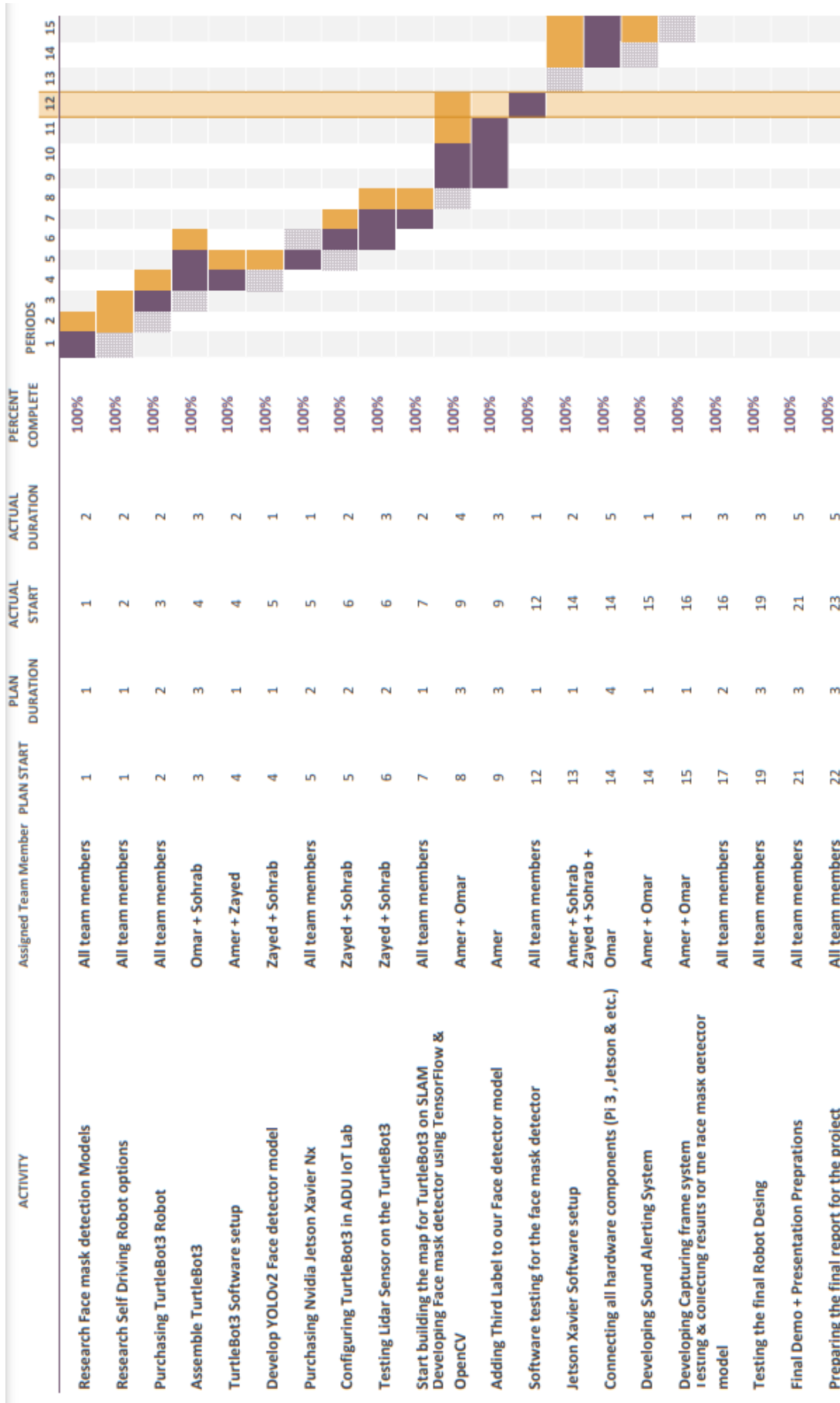


FIGURE 4.1: Caption in landscape to a figure in landscape.

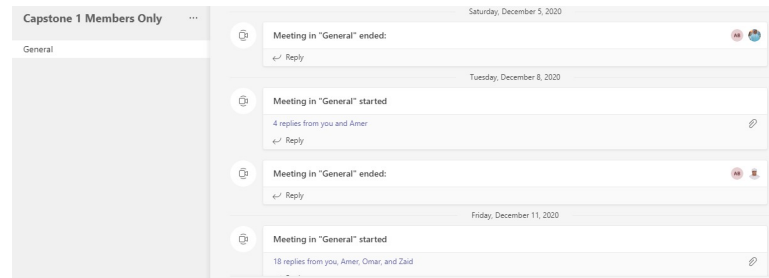


FIGURE 4.2: MS Team Group

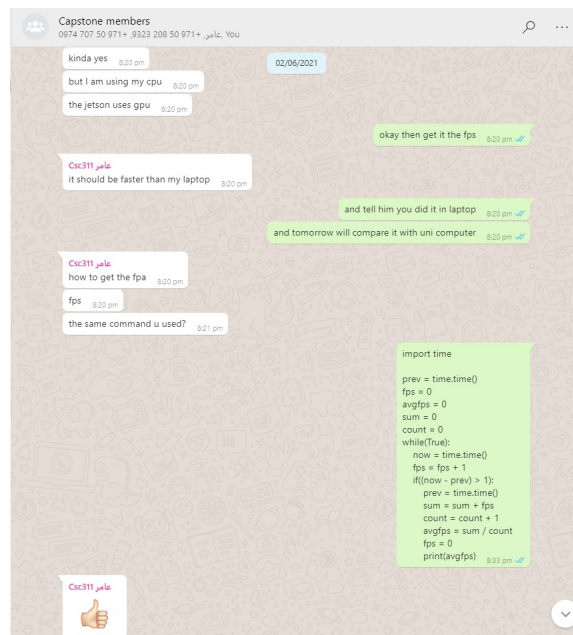


FIGURE 4.3: Whats app Group



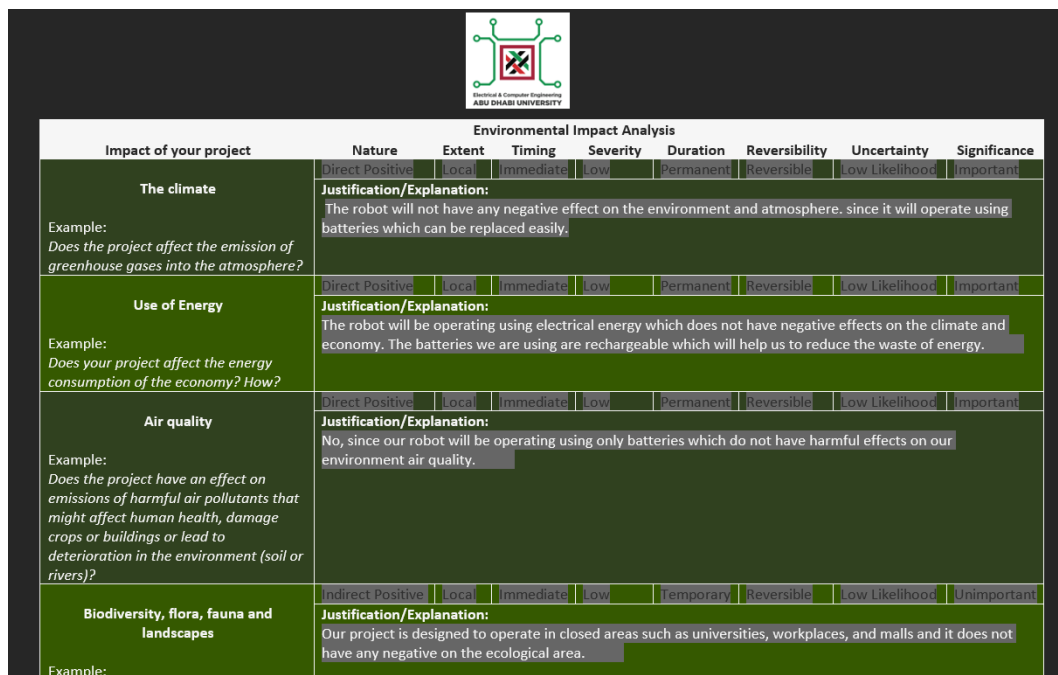
FIGURE 4.4: Home development

4.5 Impact Statement

Nowadays, the world is in a rare situation where everyone needs to follow a new rule issued by the government which is wearing face masks in public places. The robot we designed detects people wearing their face masks correctly, incorrectly, or not wearing them at all; then, it will alert them to wear their face masks properly. Our robot also helps society reduce the number of infected people by making them wear face masks all the time, which will increase awareness between people.

4.5.1 Environmental Impact

Our Self-driven robot will ensure a better and safer environment for the public with the features that we have on our robot. The robot will have cleaner air with less air pollution that may affect and damage public health. In addition, we can add a solar panel charger for the batteries as a future improvement to increase the sustainability of our robot to work longer hours depending only on solar power, which will have positive impacts on our environment.



Impact of your project	Environmental Impact Analysis							
	Nature	Extent	Timing	Severity	Duration	Reversibility	Uncertainty	Significance
The climate Example: Does the project affect the emission of greenhouse gases into the atmosphere?	Direct Positive	Local	Immediate	Low	Permanent	Reversible	Low Likelihood	Important
Justification/Explanation: The robot will not have any negative effect on the environment and atmosphere, since it will operate using batteries which can be replaced easily.								
Use of Energy Example: Does your project affect the energy consumption of the economy? How?	Direct Positive	Local	Immediate	Low	Permanent	Reversible	Low Likelihood	Important
Justification/Explanation: The robot will be operating using electrical energy which does not have negative effects on the climate and economy. The batteries we are using are rechargeable which will help us to reduce the waste of energy.								
Air quality Example: Does the project have an effect on emissions of harmful air pollutants that might affect human health, damage crops or buildings or lead to deterioration in the environment (soil or rivers)?	Direct Positive	Local	Immediate	Low	Permanent	Reversible	Low Likelihood	Important
Justification/Explanation: No, since our robot will be operating using only batteries which do not have harmful effects on our environment air quality.								
Biodiversity, flora, fauna and landscapes Example:	Indirect Positive	Local	Immediate	Low	Temporary	Reversible	Low Likelihood	Unimportant
Justification/Explanation: Our project is designed to operate in closed areas such as universities, workplaces, and malls and it does not have any negative on the ecological area.								

FIGURE 4.5: Environmental Impact Tool Screenshot

4.5.2 Economic Impact

The robot will affect the security guard jobs since the robot can monitor people with higher operating hours with less cost. This may lead some companies to choose the robot over humans in such tasks. We have already seen similar robots in business places in Dubai, where they have implemented self-driving robots to detect if people are not wearing masks. So we can say it may decrease the opportunity of some jobs such as security guards in closed places. However, it can increase the growth of the technology industry since it's a new innovation that can ensure better safety in businesses and closed places.

Impact of your project	Economic Impact Analysis							
	Nature	Extent	Timing	Severity	Duration	Reversibility	Uncertainty	Significance
Economic Prosperity Example: <i>Does the project affect the GDP/capita, employment rate, household savings?</i>	Direct Positive	Local	Immediate	Moderate	Temporary	Reversible	Low Likelihood	Important
Justification/Explanation: This project may affect some jobs, but it also can open new ideas for start-ups and companies to produce new such products.								
Investment Flows Example: <i>Does your project affect the flow of investment from outside the country? Does it encourage local investment in it?</i>	Direct Positive	Local	Immediate	Low	Temporary	Reversible	Low Likelihood	Important
Justification/Explanation: Yes, Especially Nowadays there is a huge focus on technology and AI solutions companies and start-ups which makes it a very good industry for investors.								
Public Budgets or Services Example: <i>Does the project affect the budgets of hospitals, community services, older people services, transport services, service quality, schools, policing, municipality services...etc?</i>	Indirect Positive	Local	Immediate	Moderate	Temporary	Reversible	Low Likelihood	Important
Justification/Explanation: This will robot will cost the public budget of companies in which they have to buy for the robot, but this will cost make less than for example hiring a security guard.								

FIGURE 4.6: Economic Impact Tool Screenshot 1

Innovation, Research and Development Example: <i>Does the project have commercialization potential, lead to a potential patent? Does it allow others to innovate/research through it?</i>	Direct Positive	Local	Immediate	High	Temporary	Reversible	Low Likelihood	Important
Justification/Explanation: Yes, Many future research and improvements can be further done on this project. There is always huge room for improvement.								

FIGURE 4.7: Economic Impact Tool Screenshot 2

4.5.3 Social Impact

Our Self-driving mask detection robot ensures public safety and health by detecting people who are not wearing their mask in the proper way which can affect both the public and theme-self health. Studies showed that wearing a media face mask can reduce the spread of Covid-19 by 80 percent or more [18]. Therefore, the robot can help

to reduce in making sure that everyone wears their masks especially in public places where many people come from different places. By using the Alerting sound system that we developed, the robot will keep alerting the people and taking pictures of the ones who are not wearing their masks properly.

The robot can reduce the stress level and increase the safety of the public. Since the robot will be controlled by Ai autonomously this helps us let it operate for a longer time with high accuracy. We can say that it will ensure a better quality of life in the public area. Because it will help with reducing the stress levels. When the people see such invocation this will make them more comfortable about their health's.

Impact of your project	Social Impact Analysis							
	Nature	Extent	Timing	Severity	Duration	Reversibility	Uncertainty	Significance
Health and Longevity Example: <i>Does the project impact health and longevity? Does it affect physical activity, nutrition, chronic diseases, accidental injuries, independent living, mental wellbeing?</i>	Direct Positive	Global	Immediate	High	Permanent	Reversible	Low Likelihood	Important
Justification/Explanation: The Robot will have a huge effect on the health of the well begin of the public it will ensure to detect people who are not wearing a face mask or wearing it improperly.								
Safety Example: <i>Does your project affect safety of social environment, protection of older people against abuse, protection against risks, response to emergency cases, feelings of safety, physical safety?</i>	Direct Positive	Global	Immediate	Low	Permanent	Reversible	Low Likelihood	Important
Justification/Explanation: This project can help in increasing the confidence level and feeling safer in close places such as universities and workplaces. For example, when people observe that there is a Robot that monitors people to ensure better health this will make them feel more safe and confident. Which can reduce the level of stress and anxiety.								
Productive and Valued Activities Example: <i>Does the project increase leisure time, reduce stress, lead to positive behavior, increase productivity?</i>	Indirect Positive	Global	Immediate	Moderate	Permanent	Reversible	Low Likelihood	Important
Justification/Explanation: Since our Robot's main goal is to monitor people and detect face mask detection, this can lead to positive behavior from the public. where people now will be more encouraged to wear a medical mask in public places.								

FIGURE 4.8: Social Impact Tool Screenshot 1

<p>Standard of Living</p> <p>Example:</p> <p><i>Does the project affect the quality of life? Make lives easier? Reduce poverty and deprivation? Increase life choices and opportunities?</i></p>	<p>Indirect Positive Global Immediate Low Permanent Reversible High Likelihood Important</p> <p>Justification/Explanation: Since the robot will operate autonomously, it can run longer hours with the same performance and accuracy. This is good because it can reduce the costs on companies where they no longer need to hire a security guard to do this task. In addition, the robot can work longer hours without stopping.</p>
<p>Education/Life-long Learning</p> <p>Example:</p> <p><i>Does the project affect literacy, use of ICT, chances of higher education, quality of education, life-long learning? Improve attainment of learning outcomes?</i></p>	<p>Indirect Positive Global Immediate Low Permanent Reversible Low Likelihood Important</p> <p>Justification/Explanation: For us as a team, we learned a lot of new skills and technologies that we did not introduce before and this was great. In our opinion, this robot can inspire other students to learn about robotics and machine learning which can lead to a better society that is interested in learning new things.</p>
<p>Quality of Social Interaction</p> <p>Example:</p> <p><i>Does the project affect social connectedness, social participation, volunteering?</i></p>	<p>Indirect Negative Global Immediate Moderate Temporary Reversible Low Likelihood Unimportant</p> <p>Justification/Explanation: This project may take some jobs such as security guard or volunteers jobs indirectly. but as we know technology is always developing to make people live better and yes it can remove some jobs, but then it produces new jobs in the market.</p>

FIGURE 4.9: Social Impact Tool Screenshot 2

<p>Privacy and Personal Data</p> <p>Example:</p> <p><i>Does the project reveal the user identities? Create potential private data leaks or identity theft?</i></p>	<p>Indirect Positive Local Immediate Moderate Temporary Reversible Low Likelihood Important</p> <p>Justification/Explanation: Our Robot will capture a picture for people who are either not wearing a mask or wearing it incorrectly and this data will be fully secured and used only to ensure better public health.</p>
<p>Social Reasonability</p> <p>Example:</p> <p><i>Does the project affect access to products and services for people of determination? Does it affect their integration into society? Does it affect their participation in the economy? Does it address their needs?</i></p>	<p>Indirect Positive Local Immediate High Temporary Reversible Low Likelihood Important</p> <p>Justification/Explanation: The robot's main purpose is to ensure better health for the public, it will not affect their integration with society, but it will make sure to provide a better and safer environment in closed areas.</p>

FIGURE 4.10: Social Impact Tool Screenshot 3

Appendix A

Python code for face mask detection testing

```
# import the necessary packages
import playsound
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from imutils.video import WebcamVideoStream
import numpy as np
import imutils
import time
import cv2
import os
from imutils.video import FPS
from threading import Thread
import threading
import math
import multiprocessing
import tensorflow as tf

class WebcamVideoStream:
    def __init__(self, src=0):
        self.stream = cv2.VideoCapture(src)
        (self.grabbed, self.frame) = self.stream.read()
        self.stopped = False

    def start(self):
        Thread(target=self.update, args=()).start()
        return self

    def update(self):
```

```

        while True:
            if self.stopped:
                return
            (self.grabbed, self.frame) = self.stream.read()

    def read(self):
        return self.frame

    def stop(self):
        self.stopped = True

def detect_and_predict_mask(frame, faceNet, maskNet):
    (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()
    faces = []
    locs = []
    preds = []
    for i in range(0, detections.shape[2]):
        confidence = detections[0, 0, i, 2]
        if confidence > 0.85:
            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:
        #Motion detection sensor, so the program will execute only if it detects
        motion from people
        faces = np.array(faces, dtype="float32")
        preds = maskNet.predict(faces, batch_size=32)
    return locs, preds

class myThread (threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
    def run(self):
        playsound.playsound('arabic.mp3', True)
        playsound.playsound('eng.mp3', True)

prototxtPath = r"face_detector\deploy.prototxt"

```



```

weightsPath = r"face_detector\res10_300x300_ssd_iter_140000.caffemodel"
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)
maskNet = load_model("new-mask-detector.model")

# initialize THREADED video stream
print("[INFO] sampling THREADED frames from webcam...")
vs = WebcamVideoStream(src=0).start()
i = 0
FilePath = r"busted"
new_frame_time = 0
prev_frame_time = 0
previousmiddlepointx = []
previousmiddlepointy = []
middlepointx = []
middlepointy = []
distancebetweenpoints = []
counting = 0
lowestDistance = 1000
index = 0
indextochoose = 0
incorrectAmount = 0
wearingIncorrectly = False
runOnce = False
previousIncorrectAmount = 0
thread1 = myThread()
while True:
    frame = vs.read()
    #frame = imutils.resize(frame, width=400)
    #Only if faces are detected
    (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)
    for (box, pred) in zip(locs, preds):
        counting = counting + 1
        (startX, startY, endX, endY) = box
        middlepointx.append(endX - startX)
        middlepointy.append(endY - startY)
        if len(middlepointx) >= counting and len(previousmiddlepointx) >= counting:
            for item in previousmiddlepointx:
                currentDistance = abs(middlepointx[counting-1] - item)
                + abs(middlepointy[counting-1] - previousmiddlepointy[index])
                if (currentDistance < lowestDistance):
                    lowestDistance = currentDistance
                    indextochoose = index
            index = index + 1

        distancebetweenpoints.append(math.sqrt((middlepointx[counting-1]
        - previousmiddlepointx[counting-1])**2
        + (middlepointy[counting-1]-previousmiddlepointy[counting-1])**2))
    (mask, withoutMask) = pred

```

```

label = "Correct" if mask > withoutMask else "Incorrect"
if label == "Incorrect":
    incorrectAmount = incorrectAmount + 1
    wearingIncorrectly = True
color = (0, 255, 0) if label == "Correct" else (0, 0, 255)
label = "{:} {:.2f}%".format(label, max(mask, withoutMask) * 100)
cv2.putText(frame, label, (startX, startY - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
index = 0
lowestDistance = 1000

counting = 0

if wearingIncorrectly and runOnce == False:
    cv2.imwrite(os.path.join(FilePath, 'Frame' + str(i) + '.jpg'), frame)
    if thread1.is_alive():
        False
    else:
        thread1 = myThread()
        thread1.start()
    i = i + 1
    runOnce = True
for item in distancebetweenpoints:
    if (item > 25 and wearingIncorrectly) or (len(middlepointx)
    > len(previousmiddlepointx) and wearingIncorrectly):
        cv2.imwrite(os.path.join(FilePath, 'Frame' + str(i) + '.jpg'), frame)
        i = i + 1
        break
    if incorrectAmount >= previousIncorrectAmount:
        if thread1.is_alive():
            False
        else:
            thread1 = myThread()
            thread1.start()
        break
previousIncorrectAmount = incorrectAmount
incorrectAmount = 0
previousmiddlepointx = middlepointx.copy()
previousmiddlepointy = middlepointy.copy()
middlepointx.clear()
middlepointy.clear()
font = cv2.FONT_HERSHEY_SIMPLEX
new_frame_time = time.time()
fps = 1/(new_frame_time-prev_frame_time)
fps = int(fps)
fps = str(fps)
cv2.putText(frame, fps, (7, 70), font, 3, (100, 255, 0), 3, cv2.LINE_AA)

```

```
cv2.imshow("Live Video", frame)
wearingIncorrectly = False
distancebetweenpoints.clear()
key = cv2.waitKey(1) & 0xFF
if key == ord("q"):
    print("Terminating ...")
    break

prev_frame_time = new_frame_time

cv2.destroyAllWindows()
vs.stop()
```

Bibliography

- [1] Dr. T. Uma Devi and Sakshi Gupta. Yolov2 based real time object detection. *Object detection Model*, 8:26–29, May 2020. URL <https://bit.ly/2SpojMZ>.
- [2] Hussain Mujtaba. Real-time object detection using tensorflow. *Object detection Model*, July 2020. URL <https://bit.ly/3iJh39o>.
- [3] Jetson xavier nx, 2021. URL <https://bit.ly/3iFKNnU>.
- [4] Turtlebot3, 2021. URL <https://bit.ly/3gq0Ncd>.
- [5] Opencr1.0, 2021. URL <https://bit.ly/3vj7UHy>.
- [6] Hls-lfcd2, 2021. URL <https://bit.ly/35eJzb1>.
- [7] Lipo battery 11.1v 1800mah lb-012, 2021. URL <https://bit.ly/3wpuVK0>.
- [8] C920 hd pro webcam, 2021. URL <https://bit.ly/3wm5Uzw>.
- [9] 18650 battery, 2021. URL <https://bit.ly/3cEsWd5>.
- [10] Most Products, Data Storage, and Soumya Joy. 32gb memory cards to store all the information you need - times of india, 2021. URL <https://bit.ly/2SoMU1m>.
- [11] Raspberry pi 3 model b+, 2021. URL <https://bit.ly/3zp20fZ>.
- [12] Bluetooth speaker, black, 2021. URL <https://bit.ly/3cFbIfF>.
- [13] J. jordan, "jeremy jordan," 1 march 2018. [online]. available: <https://www.jeremyjordan.me/nn-learning-rate/> [accessed 14 june 2021].
- [14] M. safder, "youtube," youtube, 14 june 2021. [online]. available: <https://www.youtube.com/watch?v=14GxXSNpAxE>. [accessed 14 june 2021].

-
- [15] Dr. Mohammed Ghazal et al. Artificial intelligence performance evaluation. May 2021.
- [16] Amro Kamal. Yolo, yolov2 and yolov3: All you want to know. *Object detection Model*, August 2019. URL <https://bit.ly/3wkLKWN>.
- [17] Lorenzo Galtarossa. Obstacle avoidance algorithms for autonomous navigation system in unstructured indoor areas. October 2018. URL <https://bit.ly/2S1a6AV>.
- [18] CYNTHIA DEMARCO. What type of mask works best for covid-19 protection. August 2020. URL <https://bit.ly/3gWiHU7>.