

**HOUSE PROTECTION SYSTEM BASED ON** **OBJECT DETECTION**

**COMPUTER ENGINEERING**

**2021**

**Bashar KHATIB**

**2028150014**

**TensorFlow Lite Object Detection Models on the Raspberry Pi 3**

Bashar KHATIB

2028150014

**KARABÜK**

**2020**

|  |
| --- |
|  |
|  |

**TensorFlow Lite Object Detection Models on the Raspberry Pi 3**

**Karabük University**

**Faculty of Engineering**

**Department of Computer Engineering**

|  |  |
| --- | --- |
| Bashar KHATIB | 2028150014 |

# 

# ABSTRACT

Today, technology is developing in the same direction in line with rapidly increasing human needs. The work done to meet these needs makes life easier every day, and these studies are concentrated in Image Processing and Deep Learning studies.

# Table of Contents

[ABSTRACT VII](#_Toc60848191)

[Table of Contents VII](#_Toc60848192)

[LIST OF FIGURES IX](#_Toc60848193)

[LIST OF TABLES IX](#_Toc60848194)

[CHAPTER 1 10](#_Toc60848195)

[1 INTRODUCTION 10](#_Toc60848196)

[CHAPTER 2 12](#_Toc60848197)

[2 COMPONENTS OF THE PROJECT 12](#_Toc60848198)

[2.1 Raspberry Pi 3 Model B: 12](#_Toc60848199)

[2.2 Raspberry Pi Camera Module: 13](#_Toc60848200)

[2.3 Buzzer: 15](#_Toc60848201)

[2.4 Light-Emitting Diode (LED): 16](#_Toc60848202)

[2.5 BREADBOARD: 16](#_Toc60848203)

[2.6 JUMP WIRES: 17](#_Toc60848204)

[2.7 MICRO SD CARD: 17](#_Toc60848205)

[2.8 PROJECT'S FINAL STATE: 18](#_Toc60848206)

[After Collecting Parts 18](#_Toc60848207)

[18](#_Toc60848208)

[CHAPTER 3 19](#_Toc60848209)

[3 DOCUMENTATION - PREPARING RASPBERRY ENVIRONMENT 19](#_Toc60848210)

[3.1 INSTALLING OPERATING SYSTEM IMAGES: 19](#_Toc60848211)

[3.2 RASPBERRY CONFIGURATION: 21](#_Toc60848212)

[3.3 Set Up TensorFlow Lite Object Detection Models: 25](#_Toc60848213)

[3.4 The Output After running the code using Terminal: 26](#_Toc60848214)

[CHAPTER 4 29](#_Toc60848215)

[4 Conclusion suggestion 29](#_Toc60848216)

[REFERENCES 30](#_Toc60848217)

[APPENDIX 31](#_Toc60848218)

[Personal Information 36](#_Toc60848219)

# LIST OF FIGURES

[Figure ‎2.1 Raspberry Pi 3 Model B 12](file:///C:\Users\abdul\OneDrive\سطح%20المكتب\Senior%20Project\ABSTRACT%20Abdullah%20HABLAS%20-%202015710213001.docx#_Toc43413591)

[Figure ‎2.2 Raspberry Pi Camera Module 13](file:///C:\Users\abdul\OneDrive\سطح%20المكتب\Senior%20Project\ABSTRACT%20Abdullah%20HABLAS%20-%202015710213001.docx#_Toc43413592)

[Figure ‎2.3 Buzzer 15](file:///C:\Users\abdul\OneDrive\سطح%20المكتب\Senior%20Project\ABSTRACT%20Abdullah%20HABLAS%20-%202015710213001.docx#_Toc43413593)

[Figure ‎2.4 Light-emitting diode 16](file:///C:\Users\abdul\OneDrive\سطح%20المكتب\Senior%20Project\ABSTRACT%20Abdullah%20HABLAS%20-%202015710213001.docx#_Toc43413594)

[Figure ‎2.5 2.4.1 Breadboard 16](file:///C:\Users\abdul\OneDrive\سطح%20المكتب\Senior%20Project\ABSTRACT%20Abdullah%20HABLAS%20-%202015710213001.docx#_Toc43413595)

[Figure ‎2.6 Jumper Wires 17](file:///C:\Users\abdul\OneDrive\سطح%20المكتب\Senior%20Project\ABSTRACT%20Abdullah%20HABLAS%20-%202015710213001.docx#_Toc43413596)

[Figure ‎2.7 Micro SD Card 17](file:///C:\Users\abdul\OneDrive\سطح%20المكتب\Senior%20Project\ABSTRACT%20Abdullah%20HABLAS%20-%202015710213001.docx#_Toc43413597)

[Figure ‎2.8 PROJECT'S FINAL STATE 18](file:///C:\Users\abdul\OneDrive\سطح%20المكتب\Senior%20Project\ABSTRACT%20Abdullah%20HABLAS%20-%202015710213001.docx#_Toc43413598)

[Figure ‎2.9 Micro SD Card 18](file:///C:\Users\abdul\OneDrive\سطح%20المكتب\Senior%20Project\ABSTRACT%20Abdullah%20HABLAS%20-%202015710213001.docx#_Toc43413599)

[Figure ‎3.1 Raspberry Pi OS Versions 19](#_Toc43413600)

[Figure ‎3.2 Welcome Wizard 20](#_Toc43413601)

[Figure ‎3.3 Raspberry Pi Configuration 23](file:///C:\Users\abdul\OneDrive\سطح%20المكتب\Senior%20Project\ABSTRACT%20Abdullah%20HABLAS%20-%202015710213001.docx#_Toc43413602)

[Figure ‎3.4 VNC Viewer 24](file:///C:\Users\abdul\OneDrive\سطح%20المكتب\Senior%20Project\ABSTRACT%20Abdullah%20HABLAS%20-%202015710213001.docx#_Toc43413603)

[Figure ‎0.1 Output 26](file:///C:\Users\abdul\OneDrive\سطح%20المكتب\Senior%20Project\ABSTRACT%20Abdullah%20HABLAS%20-%202015710213001.docx#_Toc43413604)

[Figure ‎0.2 Output 26](file:///C:\Users\abdul\OneDrive\سطح%20المكتب\Senior%20Project\ABSTRACT%20Abdullah%20HABLAS%20-%202015710213001.docx#_Toc43413605)

[Figure ‎0.3 Output 27](file:///C:\Users\abdul\OneDrive\سطح%20المكتب\Senior%20Project\ABSTRACT%20Abdullah%20HABLAS%20-%202015710213001.docx#_Toc43413606)

[Figure ‎0.4 Output 27](file:///C:\Users\abdul\OneDrive\سطح%20المكتب\Senior%20Project\ABSTRACT%20Abdullah%20HABLAS%20-%202015710213001.docx#_Toc43413607)

[Figure ‎0.5 Output 28](file:///C:\Users\abdul\OneDrive\سطح%20المكتب\Senior%20Project\ABSTRACT%20Abdullah%20HABLAS%20-%202015710213001.docx#_Toc43413608)

[Figure ‎0.6 Output 28](file:///C:\Users\abdul\OneDrive\سطح%20المكتب\Senior%20Project\ABSTRACT%20Abdullah%20HABLAS%20-%202015710213001.docx#_Toc43413609)

# LIST OF TABLES

[Table 1 Raspberry Pi Camera Modules Specification 14](#_Toc43413610)

# CHAPTER 1

# INTRODUCTION

Object Detection is used almost everywhere these days. The use cases are endless, be it Tracking objects, Video surveillance, Pedestrian detection, Anomaly detection, People Counting, Self-driving cars or Face detection, the list goes on. Object detection system is a computer vision technique that analyzes a video or image scenes, identifies the objects in them and keeps track of the objects’ motion and position. In an image that contains two or more different elements, object detection allows us to at once classify the types of things found in this image while also locating instances of them within it. To make a long story short, object recognition segmentizes images and provides a pixel-level understanding of a specific scene’s element. Object detection can be applied in many different ways, such as crowd counting, self-driving cars, video surveillance, face detection and anomaly detection.

Object detection can be broken down into machine learning-based approaches and deep learning-based approaches. In this project deep learning-based approaches have been used to create a system capable of detecting different types of objects and determine a percentage regarding each read/analysis the system does (confidence score). This system is important to protect specific areas from unwanted elements and take actions against their existence, in this case/project, the goal is to protect domestic areas from the unwanted existence of some animals in the domestic area. These animals’ existence will result in the working of an alarm attached with the system mentioned.

When it comes to deep learning-based object detection, there are three primary object detectors encountered: R-CNN and their variants, including the original R-CNN, Fast R- CNN, and Faster R-CNN, Single Shot Detector (SSDs) and YOLO. In this project, the detector used is YOLO v2.

On the contrary of R-CNN, to help increase the speed of deep learning-based object detectors, YOLO uses a one-stage detector strategy. This algorithm treats object detection as a regression problem, taking a given input image and simultaneously learning bounding box coordinates and corresponding class label probabilities. YOLO divides up the image into a grid of 13 by 13 cells, each of these cells is responsible for predicting 5 bounding boxes. A bounding box describes the rectangle that encloses the object, then it outputs a confidence score that tells us how certain it is that the predicted bounding box actually encloses some object. For each bounding box the cell also predicts a class. This works just like a classifier: it gives a probability distribution over all the possible classes. YOLO was trained on the PASCAL VOC dataset, which can detect 20 different classes. In this project it is used to detect 3 different classes: person, dog and cat. Using YOLO is simple, it is given an input image, it goes through the conventional network in a single pass and comes out the other end as a 13\*13\*125 tensor describing the bounding boxes for the grid cells. Then the final scores for the bounding boxes must be computed, after excluding the bounding boxes scoring less than 30%. Comparing YOLO v1 to YOLO v2, YOLO v2 is faster (45 FPS) and it understands more generalized object representations.

In this project, the system was built using YOLO v2 model through TensorFlow Lite framework on Raspberry pi 3, with a pi camera module and an alarm system that works whenever an unwanted object is being detected.

# CHAPTER 2

# COMPONENTS OF THE PROJECT

## Raspberry Pi 3 Model B:

Single-board computer with wireless LAN and Bluetooth connectivity shown in Fig.1.

The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote teaching of basic computer science in schools and in developing countries. The original model became far more popular than anticipated, selling outside its target market for uses such as robotics. It now is widely used even in research projects, such as for weather monitoring because of its low cost and portability. It does not include peripherals (such as keyboards and mice) or cases. However, some accessories have been included in several official and unofficial bundles.

After the release of the second board type, the Raspberry Pi Foundation set up a new entity, named Raspberry Pi Trading, and installed Eben Upton as CEO, with the responsibility of developing technology. The Foundation was rededicated as an educational charity for promoting the teaching of basic computer science in schools and developing countries.

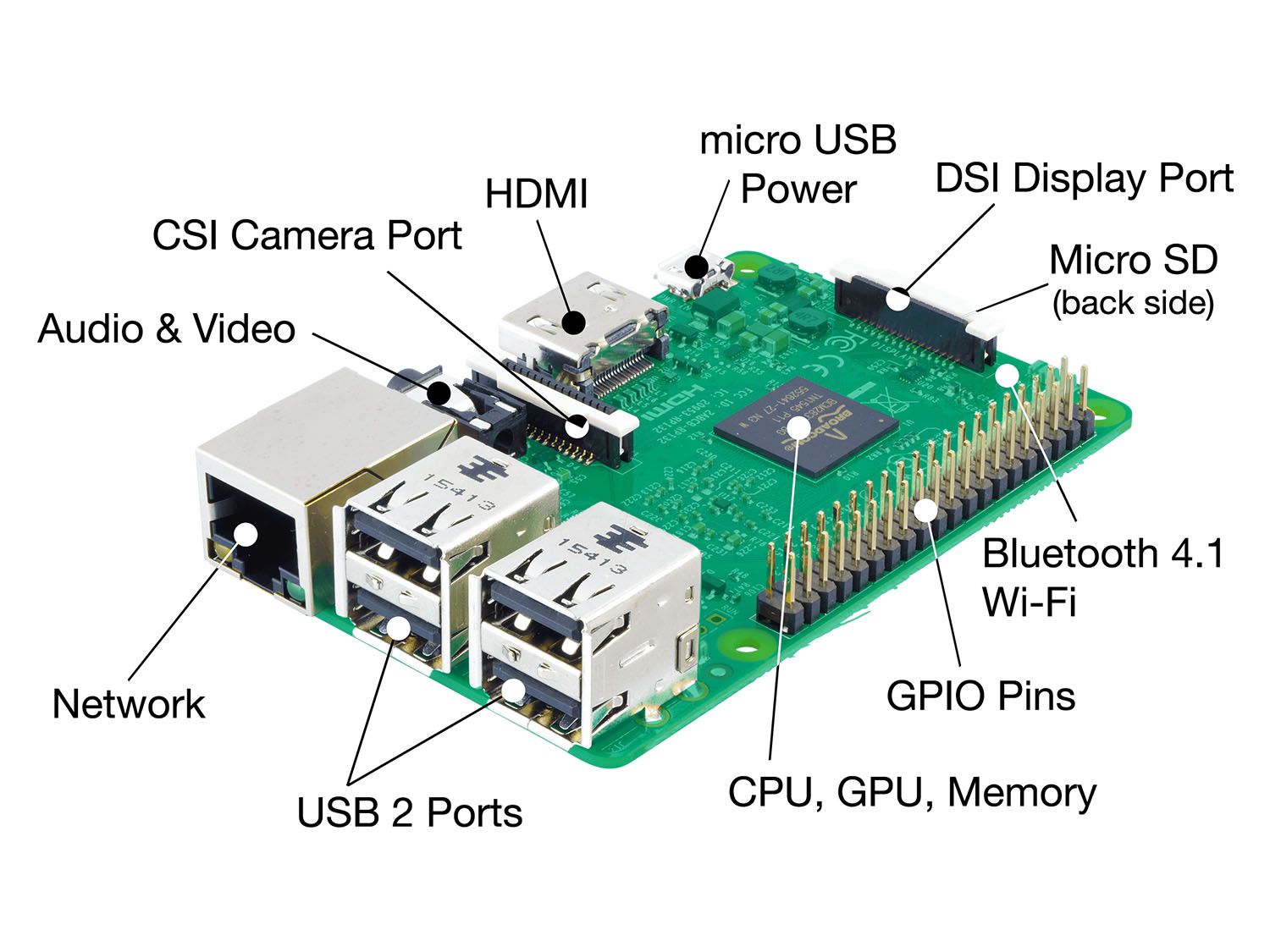
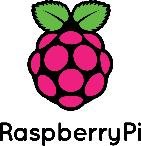
****

Figure . Raspberry Pi 3 Model B



**Specifications:**

The Raspberry Pi 3 Model B is the earliest model of the third-generation Raspberry Pi. It replaced the Raspberry Pi 2 Model B in February 2016. See also the Raspberry Pi 3 Model B+, the latest product in the Raspberry Pi 3 range.

* Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
* 1GB RAM
* BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board
* 100 Base Ethernet
* 40-pin extended GPIO
* 4 USB 2 ports
* 4 Pole stereo output and composite video port
* Full size HDMI
* CSI camera port for connecting a Raspberry Pi camera
* DSI display port for connecting a Raspberry Pi touchscreen display
* Micro SD port for loading your operating system and storing data
* Upgraded switched Micro USB power source up to 2.5A

## Raspberry Pi Camera Module:

The Raspberry Pi Camera Modules are official products from the Raspberry Pi Foundation. The original 5-megapixel model was released in 2013, and an 8-megapixel Camera Module v2 was released in 2016. For both iterations, there are visible light and infrared versions. A 12-megapixel High Quality Camera was released in 2020. There is no infrared version of the HQ Camera, however the IR Filter can be removed if required.

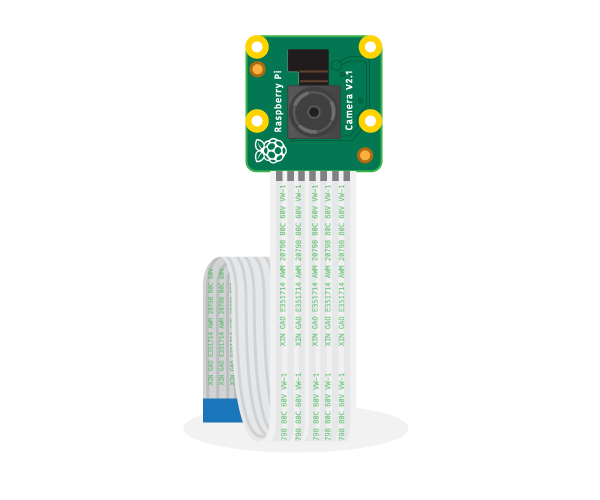


Figure . Raspberry Pi Camera Module

|  |  |  |  |
| --- | --- | --- | --- |
|  | Camera Module v1 | Camera Module v2 | HQ Camera |
| Size | Around 25 × 24 × 9 mm |  | 38 x 38 x 18.4mm (excluding lens) |
| Weight | 3g | 3g |  |
| Still resolution | 5 Megapixels | 8 Megapixels | 12.3 Megapixels |
| Video modes | 1080p30, 720p60 and 640 × 480p60/90 | 1080p30, 720p60 and 640 × 480p60/90 | 1080p30, 720p60 and 640 × 480p60/90 |
| Linux integration | V4L2 driver available | V4L2 driver available | V4L2 driver available |
| C programming API | OpenMAX IL and others available | OpenMAX IL and others available |  |
| Sensor | OmniVision OV5647 | Sony IMX219 | Sony IMX477 |
| Sensor resolution | 2592 × 1944 pixels | 3280 × 2464 pixels | 4056 x 3040 pixels |
| Sensor image area | 3.76 × 2.74 mm | 3.68 x 2.76 mm (4.6 mm diagonal) | 6.287mm x 4.712 mm (7.9mm diagonal) |
| Pixel size | 1.4 µm × 1.4 µm | 1.12 µm x 1.12 µm | 1.55 µm x 1.55 µm |
| Optical size | 1/4" | 1/4" |  |
| Full-frame SLR lens equivalent | 35 mm |  |  |
| S/N ratio | 36 dB |  |  |
| Dynamic range | 67 dB @ 8x gain |  |  |
| Sensitivity | 680 mV/lux-sec |  |  |
| Dark current | 16 mV/sec @ 60 C |  |  |
| Well capacity | 4.3 Ke- |  |  |
| Fixed focus | 1 m to infinity |  | N/A |
| Focal length | 3.60 mm +/- 0.01 | 3.04 mm | Depends on lens |
| Horizontal field of view | 53.50 +/- 0.13 degrees | 62.2 degrees | Depends on lens |
| Vertical field of view | 41.41 +/- 0.11 degrees | 48.8 degrees | Depends on lens |
| Focal ratio (F-Stop) | 2.9 | 2.0 | Depends on lens |

Specification:

Table Raspberry Pi Camera Modules Specification

## Buzzer:

A buzzer or beeper is an audio signalling device, which may be mechanical, electromechanical, or piezoelectric (piezo for short). Typical uses of buzzers and beepers include alarm devices, timers, and confirmation of user input such as a mouse click or keystroke.

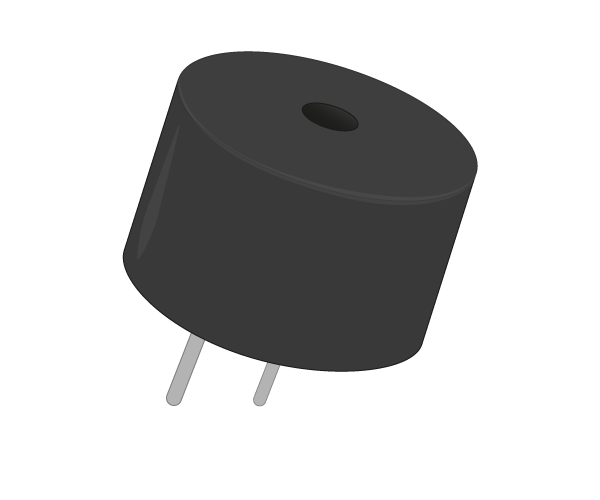
****

Figure . Buzzer

Specification:

* Rated Voltage: 6V DC
* Operating Voltage: 4-8V DC
* Rated current: <30mA
* Sound Type: Continuous Beep
* Resonant Frequency: ~2300 Hz
* Small and neat sealed package
* Breadboard and Perf board friendly

## Light-Emitting Diode (LED):

LEDs have many advantages over incandescent light sources, including lower energy consumption, longer lifetime, improved physical robustness, smaller size, and faster switching. LEDs are used in applications as diverse as aviation lighting, automotive headlamps, advertising, general lighting, traffic signals, camera flashes, lighted wallpaper, horticultural grow lights, and medical devices.

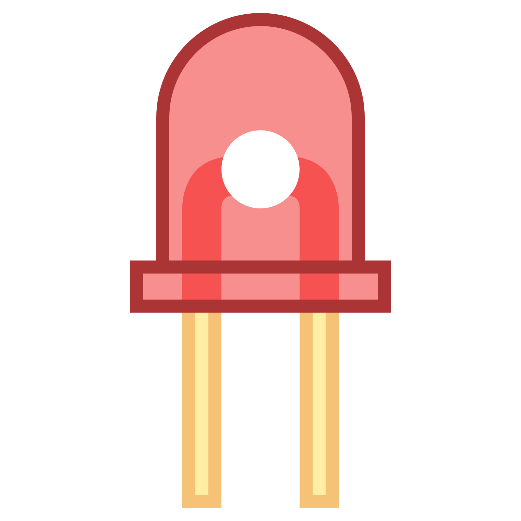


Figure . Light-emitting diode

## BREADBOARD:

A breadboard is a construction base for prototyping of electronics. Because the solderless breadboard does not require soldering, it is reusable. This makes it easy to use for creating temporary prototypes and experimenting with circuit design. For this reason, solderless breadboards are also popular with students and in technological education.

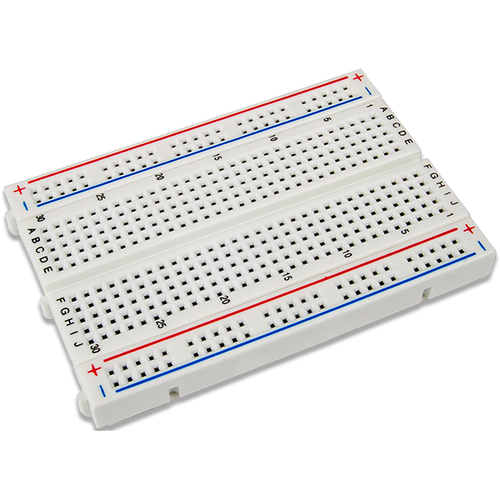


Figure . 2.4.1 Breadboard

## JUMP WIRES:

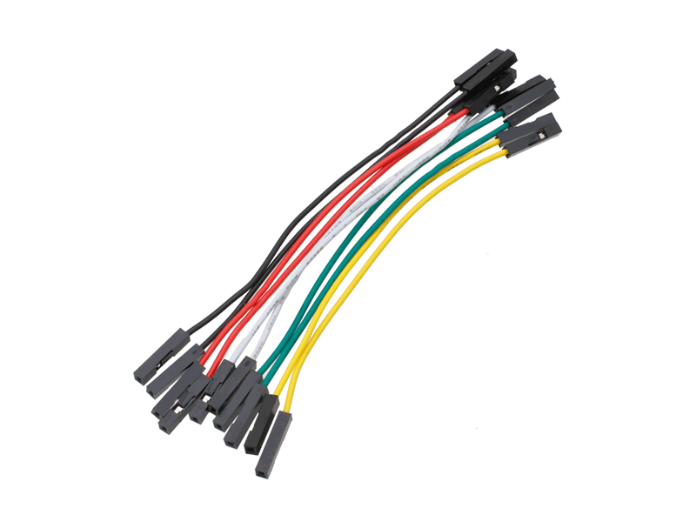
Jumper wires are simply wires that have connector pins at each end, allowing them to be used to connect two points to each other without soldering. Jumper wires are typically used with breadboards and other prototyping tools in order to make it easy to change a circuit as needed. Fairly simple. In fact, it does not get much more basic than jumper wires.

Figure . Jumper Wires

## MICRO SD CARD:

Figure . Micro SD Card

The Raspberry Pi should work with any compatible SD card,

Minimum card size is 16G.

## PROJECT'S FINAL STATE:

## After Collecting Parts

## 

Figure . PROJECT'S FINAL STATE

Figure . Micro SD Card

# CHAPTER 3

# DOCUMENTATION - PREPARING RASPBERRY ENVIRONMENT

## INSTALLING OPERATING SYSTEM IMAGES:

Prerequisites:

* Internet connection
* A computer to download and flash Raspbian or (Rufus).
* A minimum 16 GB SD card.
* Micro-SD / SD adapter (often provided with the SD card).
* A Monitor with HDMI Cable.
* Keyboard and Mouse.

Go to this page of the official website to download Raspbian Buster.

Link: https://www.raspberrypi.org/downloads/raspberry-pi-os/

Whatever your choice of version, the download is done on the same page.

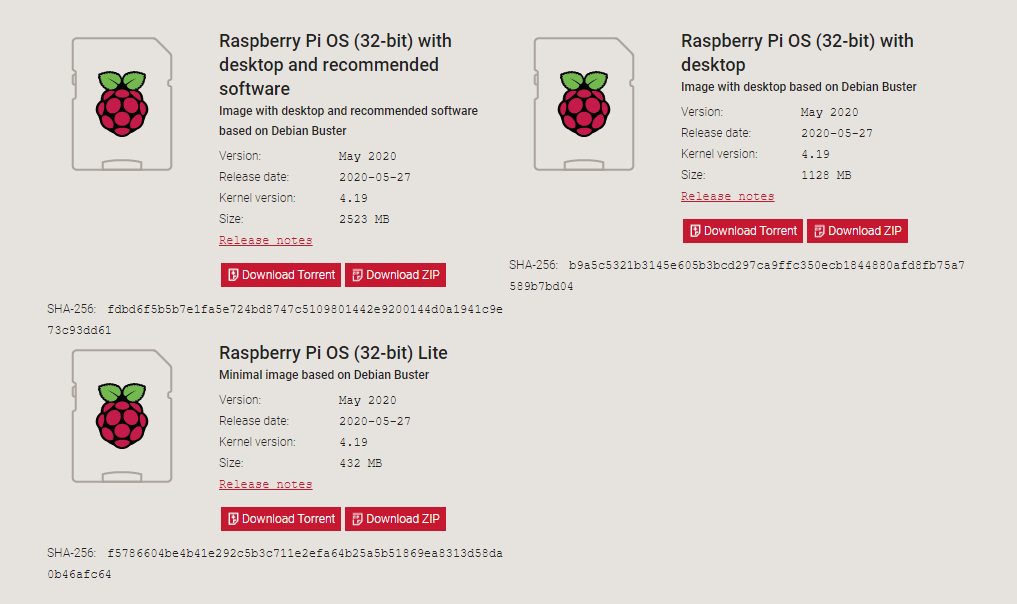


Figure . Raspberry Pi OS Versions

Click on the “Download ZIP” button to get the file, then extract the image from the ZIP file.

Download and install Etcher from the same website or use Rufus.

Flash the Disk Image onto The SD Card.

The Etcher interface is simple to use.

It is presented in 3 steps:

* 1. Image selection
  2. SD Card choice
  3. Flash!

First boot:

* Insert the SD card.
* Get your SD card and insert it into your Raspberry Pi.
* Then start the Raspberry Pi, with a screen and a keyboard.

Desktop version:

When you start on the desktop version for the first time, there is nothing to do. The system automatically logs in and introduces you to a welcome wizard.

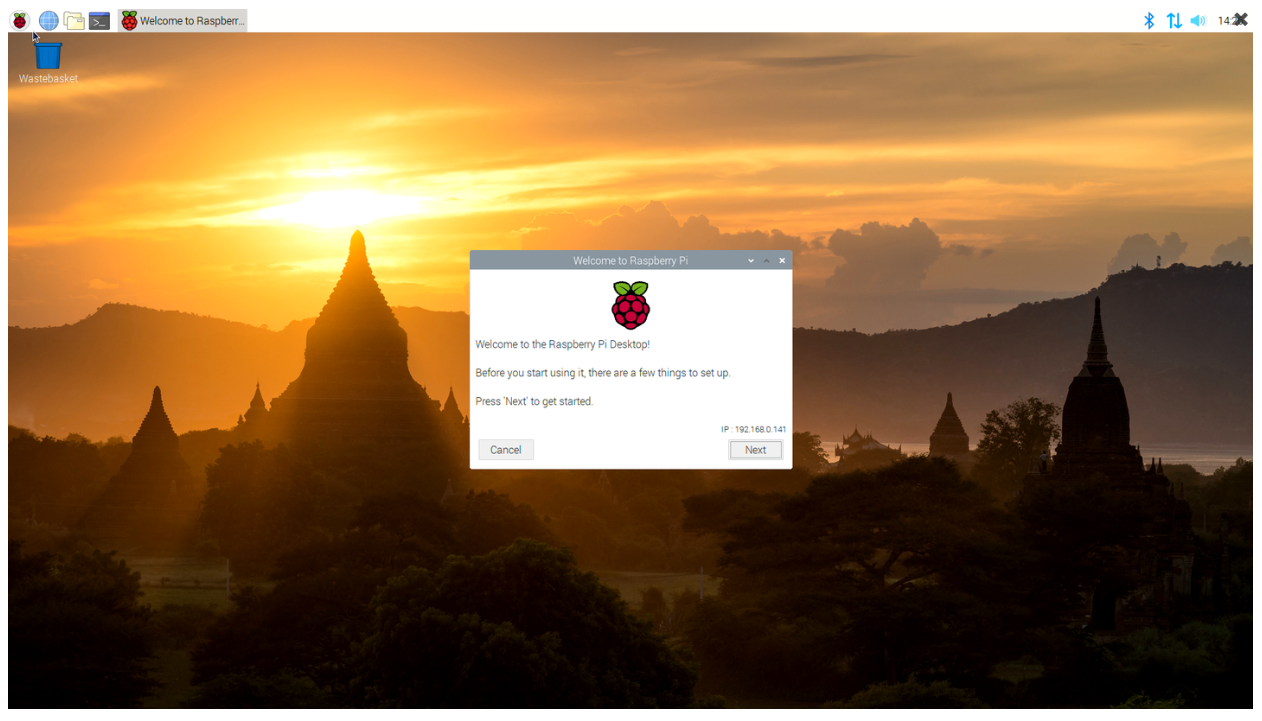


Figure . Welcome Wizard

**Lite version**

On the Lite version, you have nothing to do at first boot, the system will start alone until the login screen

Log on with the default access:

**Login: pi**

**Password: raspberry**

## RASPBERRY CONFIGURATION:

Some basic guides to configuring your Raspberry Pi.

* **raspi-config**

The Raspberry Pi configuration tool in Raspberry Pi OS, which allows you to easily enable features such as the camera, and to change your specific settings such as keyboard layout

* **config.txt**

The Raspberry Pi configuration file

* **TCP/IP networking**

Configuring the TCP/IP network stack on the Raspberry Pi

* **Connect to a wireless network**

Configuring your Pi to connect to a wireless network using the Raspberry Pi 4, Raspberry Pi 3 or Pi Zero W's inbuilt wireless connectivity, or a USB wireless dongle

* **Wireless access point**

Configuring your Pi as a wireless access point using the Raspberry Pi 4, Raspberry Pi 3, or Raspberry Pi Zero W's inbuilt wireless connectivity, or a USB wireless dongle

* **Using a proxy**

Setting up your Pi to access the internet via a proxy server

* **HDMI Config**

Setting up your HDMI device, including custom settings

* **Screen Configuration Editor**

Setting up your display device's resolution, frequency and orientation using the provided graphical editor

* **Audio config**

Switching your audio output between HDMI and the 3.5mm jack

* **Camera config**

Installing and setting up the Raspberry Pi camera board

* External storage config

Mounting and setting up external storage on a Raspberry Pi

* **Localisation**

Setting up your Pi to work in your local language and time zone

* **Default pin configuration**

Changing the default pin states.

* **Device Trees config**

Device Trees, overlays, and parameters

* **Kernel command line**

Setting options in the kernel command line

* **UART configuration**

Setting up the on-board UARTs

* **Firmware warning icons**

Description of warning icons displayed if the firmware detects issues

* **LED warning flash codes**

Description of LED warning flashes that are shown if a Pi fails to boot or must shut down

* **Securing your Raspberry Pi**

Some basic advice for making your Raspberry Pi more secure

* **Screensaver**

Configuring the screen saver and screen blanking

* **The boot folder**

What it is for and what's in it

* **Network File System (NFS)**

How to set up a NFS and connect clients to it

For this project enable the raspberry pi camera and VNC.

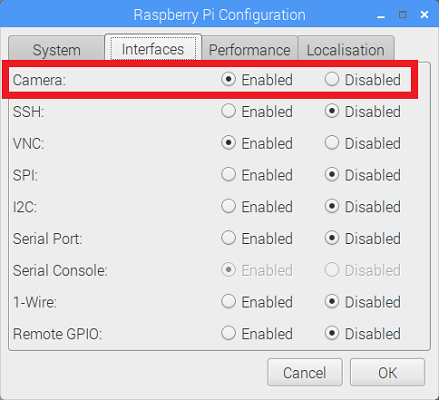


Figure . Raspberry Pi Configuration

Sometimes it is not convenient to work directly on the Raspberry Pi. Maybe you would like to work on it from another device by remote control.

VNC is a graphical desktop sharing system that allows you to remotely control the desktop interface of one computer (running VNC Server) from another computer or mobile device (running VNC Viewer). VNC Viewer transmits the keyboard and either mouse or touch events to VNC Server and receives updates to the screen in return.

You will see the desktop of the Raspberry Pi inside a window on your computer or mobile device. You will be able to control it as though you were working on the Raspberry Pi itself.

VNC Connect from RealVNC is included with Raspberry Pi OS. It consists of both VNC Server, which allows you to control your Raspberry Pi remotely, and VNC Viewer, which allows you to control desktop computers remotely from your Raspberry Pi should you want to.

You must enable VNC Server before you can use it: instructions for this are given below. By default, VNC Server gives you remote access to the graphical desktop that is running on your Raspberry Pi, as though you were sitting in front of it.

However, you can also use VNC Server to gain graphical remote access to your Raspberry Pi if it is headless or not running a graphical desktop. For more information on this, see Creating a virtual desktop, further below.

If you are not using a desktop you can install it from the command line as follows:

sudo apt update

sudo apt install realvnc-vnc-server realvnc-vnc-viewer

**Connecting to your Raspberry Pi with VNC Viewer:**

On the device you'll use to take control, download VNC Viewer. For best results, use the compatible app from RealVNC.

Enter your Raspberry Pi's private IP address into VNC Viewer:

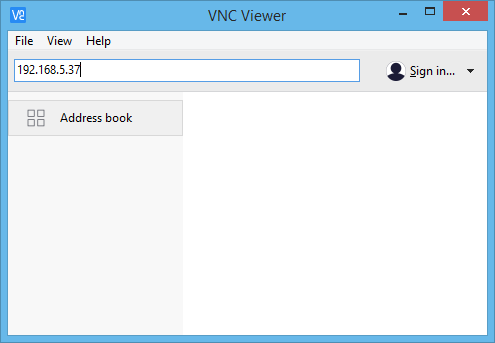


Figure . VNC Viewer

## Set Up TensorFlow Lite Object Detection Models:

Setting up TensorFlow Lite on the Raspberry Pi is much easier than regular TensorFlow!

These are the steps needed to set up TensorFlow Lite:

1a. Update the Raspberry Pi

First, the Raspberry Pi needs to be fully updated. Open a terminal and issue:

sudo apt-get update

sudo apt-get dist-upgrade

Depending on how long it has been since you have updated your Pi, the update could take anywhere between a minute and an hour.

1b. Download this repository and create virtual environment:

By following instructions provided in the GitHub link shown below you will be able to set up TensorFlow Lite and object detection system on the Raspberry Pi.

Link:

<https://github.com/AbdHablas/TensorFlow-Lite-Object-Detection-Models-on-the-Raspberry-Pi-3-Camera.git>

## The Output After running the code using Terminal:

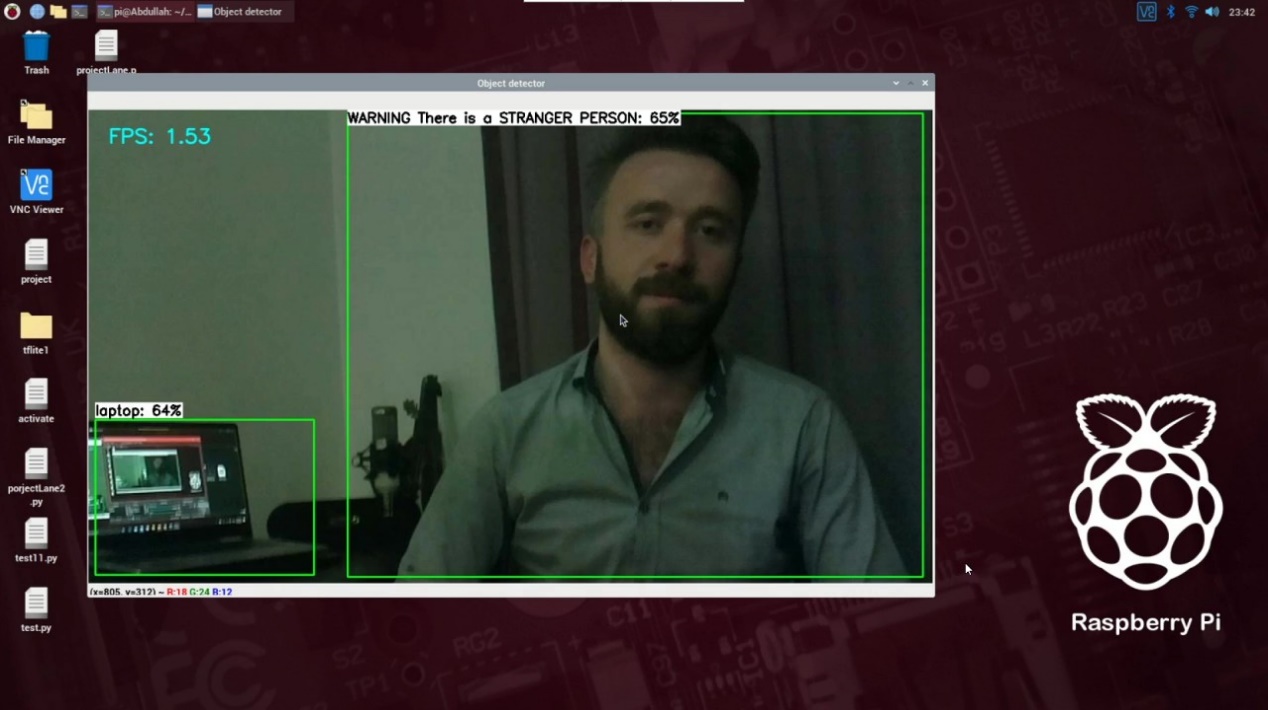


Figure . Output

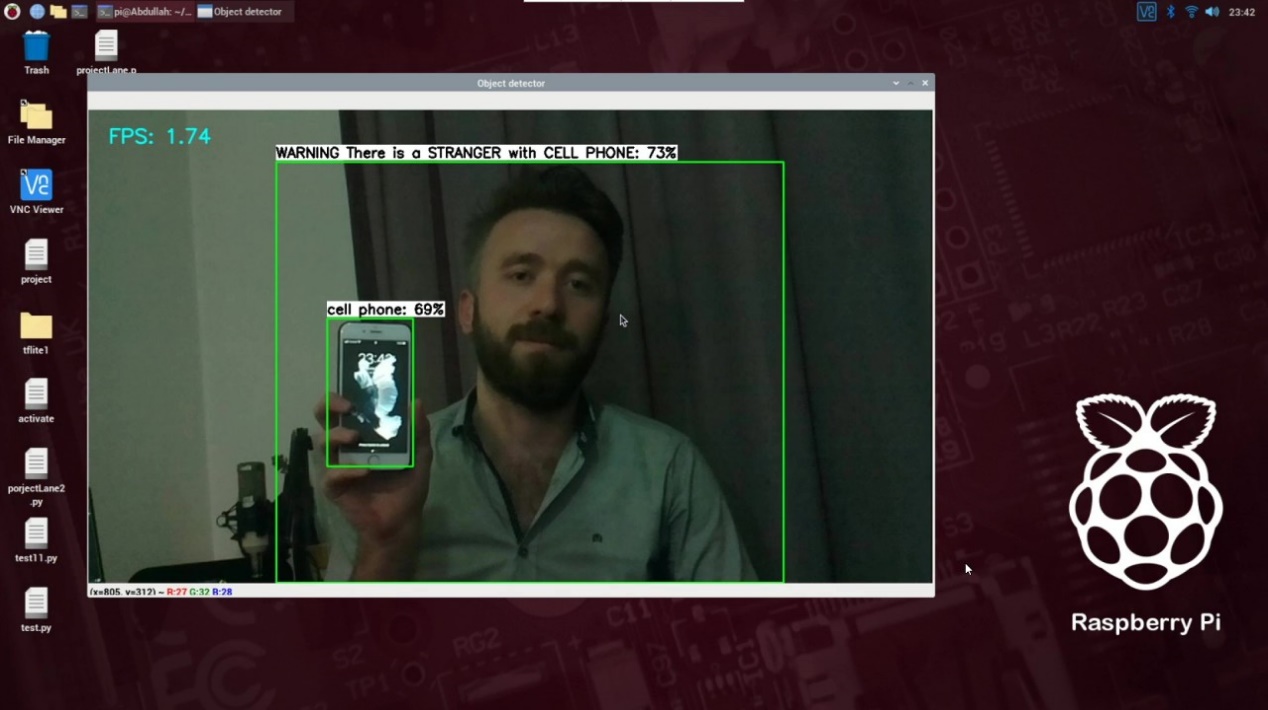


Figure . Output

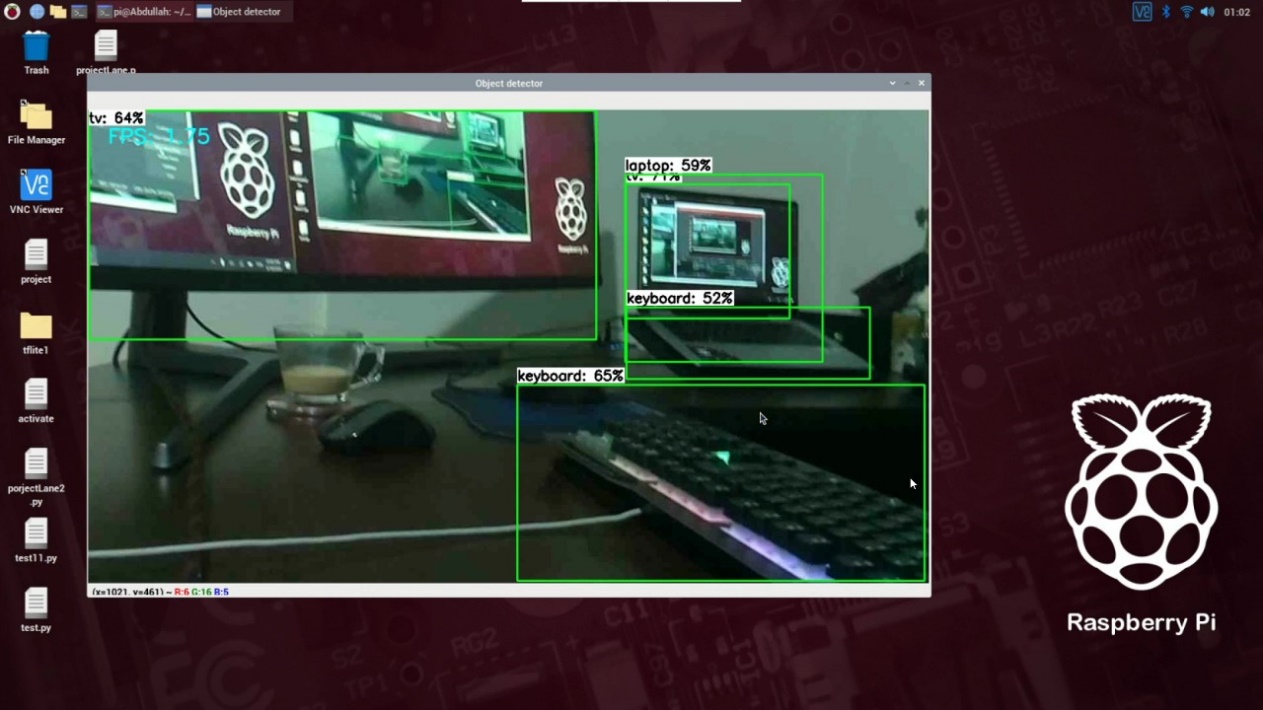


Figure . Output

Figure . Output

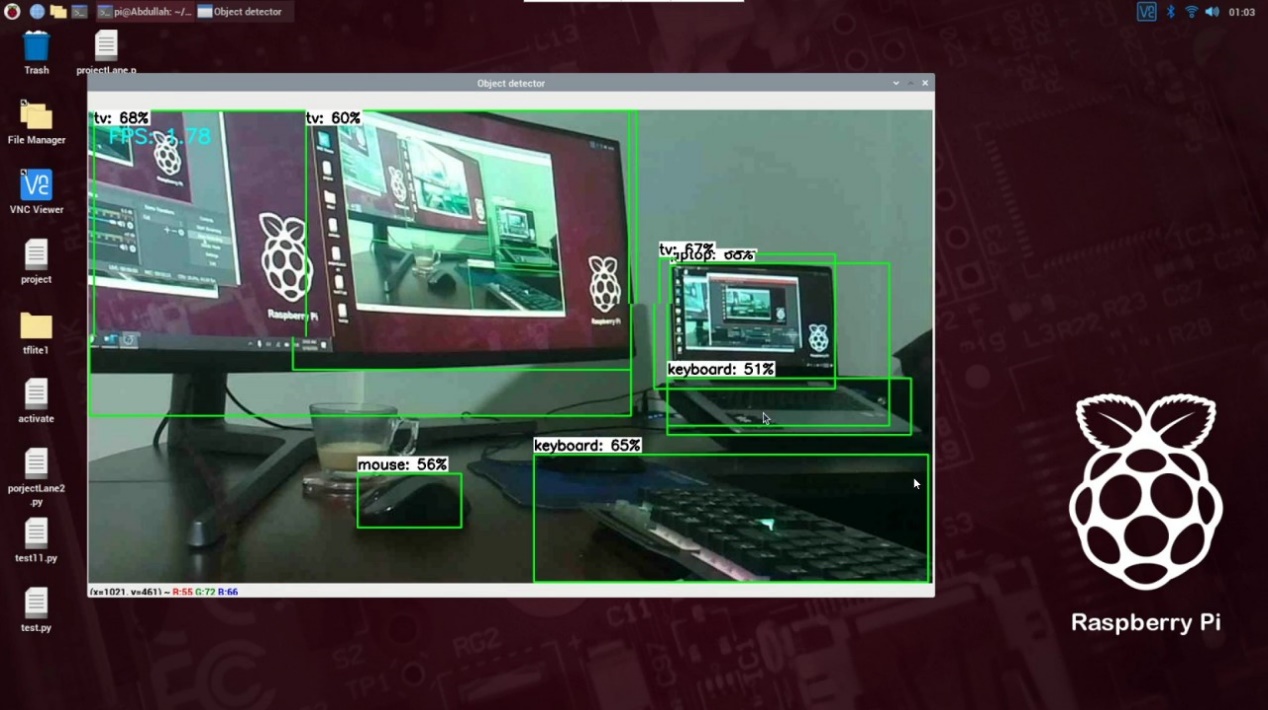
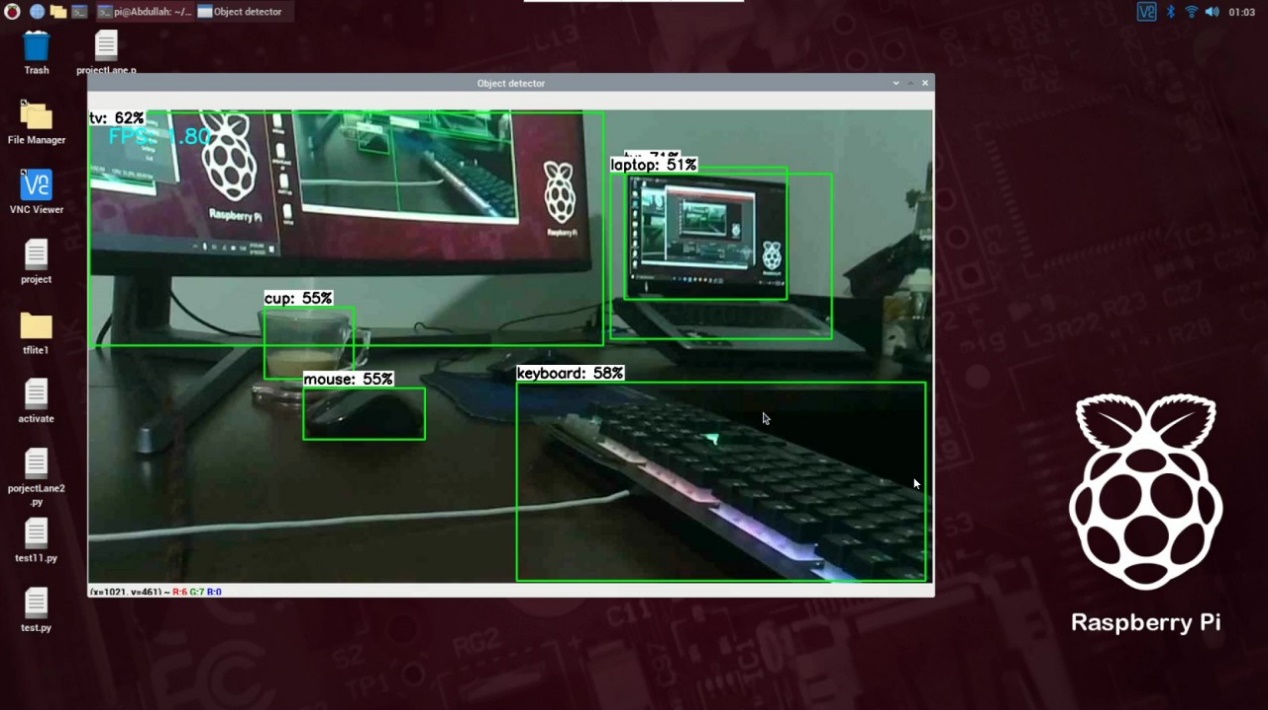


Figure . Output

Figure . Output

# CHAPTER 4

# Conclusion suggestion

Object detection is used in a wide range of industries, with use cases ranging from personal security to productivity in the workplace. It is used in several applications, as mentioned before in this report. The possibilities are endless when it comes to future use cases for object detection.

This project could be developed to guarantee a higher accuracy and better performance. For example, instead of using a normal camera module, raspberry pi3 thermal camera module could have been used along with other different types of sensors. This would give the system an extra advantage as it will be able to differentiate between real objects and pictures. For example, it could not tell the difference between a real car and a picture of a car. A thermal camera was not used as it is difficult to obtain because of its high cost.

By adding different types of sensors, this project could be taken to a next level, allowing it to be used as a real-time car detection module, which is the key success of autonomous vehicle systems. The developed system would be able to identify, locate, and track objects around it allowing it to navigate its environment safely and efficiently. And while tasks like image segmentation can be applied to autonomous vehicles, object detection remains a foundational task that underpins current work on making self-driving cars a reality.

# REFERENCES

* [https://www.raspberrypi.org](https://www.raspberrypi.org/)
* [https://www.tensorflow.org](https://www.tensorflow.org/)
* [https://www.pyimagesearch.com](https://www.pyimagesearch.com/)
* <https://en.wikipedia.org/wiki/Raspberry_Pi>
* [https://www.raspberrypi.org/products/raspberry-pi-3-model-b](https://www.raspberrypi.org/products/raspberry-pi-3-model-b/)
* [https://www.raspberrypi.org/documentation/hardware/camera](https://www.raspberrypi.org/documentation/hardware/camera/)
* <https://en.wikipedia.org/wiki/Buzzer>

# APPENDIX

**The main TFLite\_detection\_webcam Code:**

# Import packages

import os

import argparse

import cv2

import numpy as np

import sys

import time

from threading import Thread

import importlib.util

import RPi.GPIO as GPIO

from time import sleep

GPIO.setwarnings(False)

GPIO.setmode(GPIO.BCM)

GPIO.setup(17, GPIO.OUT)

GPIO.output(17, True)

sleep(4)

GPIO.output(17, False)

# Define VideoStream class to handle streaming of video from webcam in separate processing thread

# Source - Adrian Rosebrock, PyImageSearch: https://www.pyimagesearch.com/2015/12/28/increasing-raspberry-pi-fps-with-python-and-opencv/

class VideoStream:

"""Camera object that controls video streaming from the Picamera"""

def \_\_init\_\_(self,resolution=(640,480),framerate=30):

# Initialize the PiCamera and the camera image stream

self.stream = cv2.VideoCapture(0)

ret = self.stream.set(cv2.CAP\_PROP\_FOURCC, cv2.VideoWriter\_fourcc(\*'MJPG'))

ret = self.stream.set(3,resolution[0])

ret = self.stream.set(4,resolution[1])

# Read first frame from the stream

(self.grabbed, self.frame) = self.stream.read()

# Variable to control when the camera is stopped

self.stopped = False

def start(self):

# Start the thread that reads frames from the video stream

Thread(target=self.update,args=()).start()

return self

def update(self):

# Keep looping indefinitely until the thread is stopped

while True:

# If the camera is stopped, stop the thread

if self.stopped:

# Close camera resources

self.stream.release()

return

# Otherwise, grab the next frame from the stream

(self.grabbed, self.frame) = self.stream.read()

def read(self):

# Return the most recent frame

return self.frame

def stop(self):

# Indicate that the camera and thread should be stopped

self.stopped = True

# Define and parse input arguments

parser = argparse.ArgumentParser()

parser.add\_argument('--modeldir', help='Folder the .tflite file is located in',

required=True)

parser.add\_argument('--graph', help='Name of the .tflite file, if different than detect.tflite',

default='detect.tflite')

parser.add\_argument('--labels', help='Name of the labelmap file, if different than labelmap.txt',

default='labelmap.txt')

parser.add\_argument('--threshold', help='Minimum confidence threshold for displaying detected objects',

default=0.5)

parser.add\_argument('--resolution', help='Desired webcam resolution in WxH. If the webcam does not support the resolution entered, errors may occur.',

default='1280x720')

parser.add\_argument('--edgetpu', help='Use Coral Edge TPU Accelerator to speed up detection',

action='store\_true')

args = parser.parse\_args()

MODEL\_NAME = args.modeldir

GRAPH\_NAME = args.graph

LABELMAP\_NAME = args.labels

min\_conf\_threshold = float(args.threshold)

resW, resH = args.resolution.split('x')

imW, imH = int(resW), int(resH)

use\_TPU = args.edgetpu

# Import TensorFlow libraries

# If tensorflow is not installed, import interpreter from tflite\_runtime, else import from regular tensorflow

# If using Coral Edge TPU, import the load\_delegate library

pkg = importlib.util.find\_spec('tensorflow')

if pkg is None:

from tflite\_runtime.interpreter import Interpreter

if use\_TPU:

from tflite\_runtime.interpreter import load\_delegate

else:

from tensorflow.lite.python.interpreter import Interpreter

if use\_TPU:

from tensorflow.lite.python.interpreter import load\_delegate

# If using Edge TPU, assign filename for Edge TPU model

if use\_TPU:

# If user has specified the name of the .tflite file, use that name, otherwise use default 'edgetpu.tflite'

if (GRAPH\_NAME == 'detect.tflite'):

GRAPH\_NAME = 'edgetpu.tflite'

# Get path to current working directory

CWD\_PATH = os.getcwd()

# Path to .tflite file, which contains the model that is used for object detection

PATH\_TO\_CKPT = os.path.join(CWD\_PATH,MODEL\_NAME,GRAPH\_NAME)

# Path to label map file

PATH\_TO\_LABELS = os.path.join(CWD\_PATH,MODEL\_NAME,LABELMAP\_NAME)

# Load the label map

with open(PATH\_TO\_LABELS, 'r') as f:

labels = [line.strip() for line in f.readlines()]

# Have to do a weird fix for label map if using the COCO "starter model" from

# https://www.tensorflow.org/lite/models/object\_detection/overview

# First label is '???', which has to be removed.

if labels[0] == '???':

del(labels[0])

# Load the Tensorflow Lite model.

# If using Edge TPU, use special load\_delegate argument

if use\_TPU:

interpreter = Interpreter(model\_path=PATH\_TO\_CKPT,

experimental\_delegates=[load\_delegate('libedgetpu.so.1.0')])

print(PATH\_TO\_CKPT)

else:

interpreter = Interpreter(model\_path=PATH\_TO\_CKPT)

interpreter.allocate\_tensors()

# Get model details

input\_details = interpreter.get\_input\_details()

output\_details = interpreter.get\_output\_details()

height = input\_details[0]['shape'][1]

width = input\_details[0]['shape'][2]

floating\_model = (input\_details[0]['dtype'] == np.float32)

input\_mean = 127.5

input\_std = 127.5

# Initialize frame rate calculation

frame\_rate\_calc = 1

freq = cv2.getTickFrequency()

# Initialize video stream

videostream = VideoStream(resolution=(imW,imH),framerate=30).start()

time.sleep(1)

#for frame1 in camera.capture\_continuous(rawCapture, format="bgr",use\_video\_port=True):

while True:

# Start timer (for calculating frame rate)

t1 = cv2.getTickCount()

# Grab frame from video stream

frame1 = videostream.read()

# Acquire frame and resize to expected shape [1xHxWx3]

frame = frame1.copy()

frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

frame\_resized = cv2.resize(frame\_rgb, (width, height))

input\_data = np.expand\_dims(frame\_resized, axis=0)

# Normalize pixel values if using a floating model (i.e. if model is non-quantized)

if floating\_model:

input\_data = (np.float32(input\_data) - input\_mean) / input\_std

# Perform the actual detection by running the model with the image as input

interpreter.set\_tensor(input\_details[0]['index'],input\_data)

interpreter.invoke()

# Retrieve detection results

boxes = interpreter.get\_tensor(output\_details[0]['index'])[0] # Bounding box coordinates of detected objects

classes = interpreter.get\_tensor(output\_details[1]['index'])[0] # Class index of detected objects

scores = interpreter.get\_tensor(output\_details[2]['index'])[0] # Confidence of detected objects

#num = interpreter.get\_tensor(output\_details[3]['index'])[0] # Total number of detected objects (inaccurate and not needed)

# Loop over all detections and draw detection box if confidence is above minimum threshold

for i in range(len(scores)):

if ((scores[i] > min\_conf\_threshold) and (scores[i] <= 1.0)):

# Get bounding box coordinates and draw box

# Interpreter can return coordinates that are outside of image dimensions, need to force them to be within image using max() and min()

ymin = int(max(1,(boxes[i][0] \* imH)))

xmin = int(max(1,(boxes[i][1] \* imW)))

ymax = int(min(imH,(boxes[i][2] \* imH)))

xmax = int(min(imW,(boxes[i][3] \* imW)))

cv2.rectangle(frame, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)

# Draw label & conditions

state = "true"

object\_name = labels[int(classes[i])] # Look up object name from "labels" array using class index

if int(classes[i]) == 0:

for j in range(len(classes)):

if int(classes[j]) == 76:

state = "false";

if state == "true":

object\_name = 'WARNING There is a STRANGER PERSON'

GPIO.output(17, True)

sleep(0.1)

GPIO.output(17, False)

else:

object\_name = 'WARNING There is a STRANGER with CELL PHONE'

#

elif int(classes[i]) == 17:

object\_name = 'WARNING There is a Dog'

# GPIO.output(17, True)

sleep(0.1)

# GPIO.output(17, False)

# label = '%d: %s%d%%' % (int(classes[i]),object\_name, int(scores[i]\*100)) # Example: 'person: 72%'

label = '%s: %d%%' % (object\_name, int(scores[i]\*100)) # Example: 'person: 72%'

labelSize, baseLine = cv2.getTextSize(label, cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, 2) # Get font size

label\_ymin = max(ymin, labelSize[1] + 10) # Make sure not to draw label too close to top of window

cv2.rectangle(frame, (xmin, label\_ymin-labelSize[1]-10), (xmin+labelSize[0], label\_ymin+baseLine-10), (255, 255, 255,0.5), cv2.FILLED) # Draw white box to put label text in

cv2.putText(frame, label, (xmin, label\_ymin-7), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 0), 2) # Draw label text

# Draw framerate in corner of frame

cv2.putText(frame,'FPS: {0:.2f}'.format(frame\_rate\_calc),(30,50),cv2.FONT\_HERSHEY\_SIMPLEX,1,(255,255,0),2,cv2.LINE\_AA)

# All the results have been drawn on the frame, so it's time to display it.

cv2.imshow('Object detector', frame)

# Calculate framerate

t2 = cv2.getTickCount()

time1 = (t2-t1)/freq

frame\_rate\_calc= 1/time1

# Press 'q' to quit

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

break

# Clean up

cv2.destroyAllWindows()

videostream.stop()

# Personal Information

|  |  |
| --- | --- |
| Name Surname | : Bashar KHATIB |
| Nationality | : Syrian |
| Date of birth | : IDLIB 04.06.1996 |
| phone | : +905374255648 |
| e-mail | : basharkhatib51@gmail.com |
|  | |
|  |  |
|  |  |