**Abstract**

The project is about creating a robot which will avoid obstacles and detect human being faces and stores them. The robot consists of a gopigo3 base, raspberry pi 3, camera module and distance sensor. Face detection is done by using concept of cascading by using open cv library. The robot can be made autonomous or can be controlled from a remote desktop.

**Introduction**

In chapter one we discuss about the previous works done on obstacle avoidance and face detection.

In chapter two we discuss about the tools which acts as a user interface to drive the robot and helps in things we intended to do. some of the them like putty, WinSCP, vncserver and remote desktop to connect to raspberry pi via network using ip address of a raspberry pi3. We also discuss about the python and also talk about libraries that need to be installed.

In chapter three we discuss about the functional design of the project which includes block diagram and description of hardware modules used in the project.

In chapter four we discuss about obstacle avoidance and learn how to program it by using python. We also learn about how to control a robot using a keyboard and we also make it autonomous using python script. Face detection is done by using concept of cascading by using open cv library.

Finally, we can use this robot to monitor the old people or children in a house by connecting remotely using a PC. We can also monitor them by live feed by using web interface which requires the ip address of the raspberry pi. We can extend its capabilities using other hardware.

**1.Literature search**

Autonomous ground vehicles and mobile robots have been developed with diﬀerent conﬁgurations and sensing capabilities since approximately the 1970s. One of the earliest wheeled robots was the SHAKEY. The SHAKEY was based on a combination of on-board computers and a radio link to larger computers elsewhere. SHAKEY’s primary sensor was a scanning camera. Around the early 1980’s, the CMU Rover represented the state of the art in robotic mobility. The CMU Rover also had multiple general-purpose processors on-board and had provisions for connecting to a large remote computer.

Many diﬀerent sensors and instruments have been used on mobile robots. These include ultrasonic sensors, infrared sensors, vision sensors, tactile sensors, and encoders. Ultrasonic sensors have been one of the simplest and widely used sensors for measuring distance to the nearest obstacle from a mobile robot. Design and development of circuits for distance measurement between a transmitter and receiver ultrasonic transducer have been extensively researched. Approaches to correct various types of errors during distance measurement have also been developed.

Ultrasonic sensors have a range of approximately 3 meters and constitute the primary sensor on the mobile robot. While sensors based on the ultrasonic, pulse-echo technique described in the previous paragraph have been the mainstay for collision avoidance systems in mobile robots, other alternatives that are not based on sound have also been examined by researchers. In particular, where it is diﬃcult to provide a medium for propagation of sound waves, light-based sensors have been explored. Some of these sensors are based on time-of-ﬂight and triangulation techniques.

Infrared sensors are used for detection of objects within a very small distance (less than 20 cm) from the robot and to cut oﬀ power (via a relay) to the motors driving the robot wheels. It is worth noting that light-based sensors are typically used to determine the presence of a “target” (or an object) rather than to measure distance to them.

There are two underlying motivations to present this literature search: the first is to provide an up-to-date review of the existing literature, and the second is to offer some insights into the studies of machine recognition of faces. To provide a comprehensive survey, existing recognition techniques of face recognition are categorized and detailed descriptions of representative methods within each category are presented. In addition, relevant topics such as psychophysical studies, system evaluation, issues of illumination and pose variation are covered. Automated face recognition was developed in 1960s. The first semi-automated system for face recognition required the administrator to locate features such as eyes, ears, nose, and mouth on the photographs before calculating the distances and the ratios to a common reference point, which were then compared to the reference data. In 1970s, the problem with both of these early solutions was that the measurements and locations were manually computed. In 1990, Kirby and Sirovich applied Principal Component Analysis, a standard linear algebraic technique, to the face recognition problem. This was considered as a milestone. It is shown that less than one hundred values were required to accurately code a suitably aligned and normalized face image. As a result of many studies, scientists come up with the decision that face recognition is not like other object recognition. Face recognition is one of the few biometric methods that possess the merits of both high accuracy and low intrusiveness. It has the accuracy of a physiological approach without being intrusive. For this reason, since the early seventies, face recognition has drawn the attention of researchers in fields from security, psychology and image processing. In this project we are going to use Cascade classifiers which is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001.

**2.Software Requirements**

User interface plays an important role in connecting user to a robot. Some of the software used to connect to gopigo3 are

1. Raspbian OS
2. Etcher
3. Win32Diskimager
4. Putty
5. WinSCP
6. Vnc viewer
7. Python

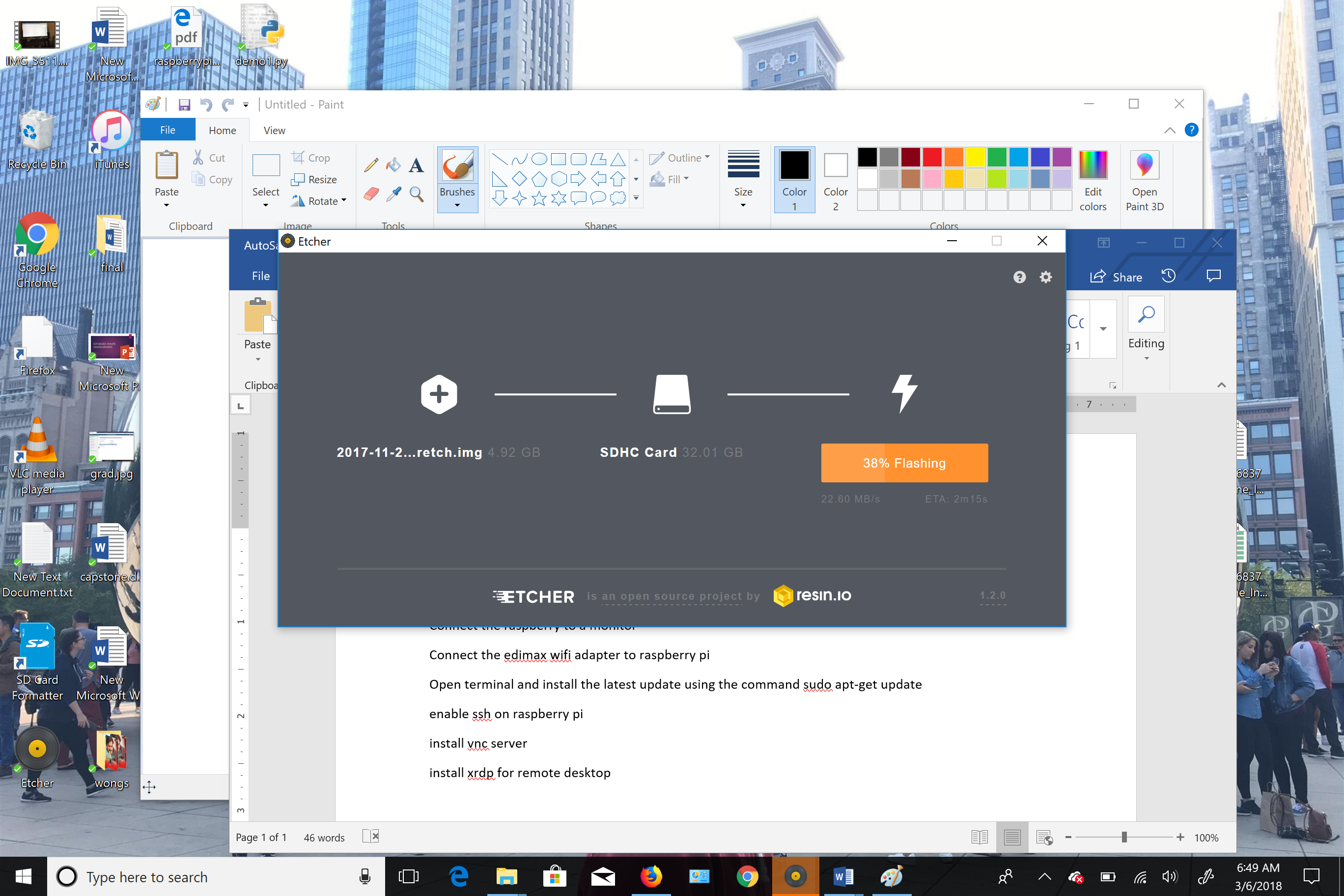
**2.1 Raspbian OS:**

Raspbian is a [Debian](https://en.wikipedia.org/wiki/Debian)-based [computer operating system](https://en.wikipedia.org/wiki/Operating_system) for [Raspberry Pi](https://en.wikipedia.org/wiki/Raspberry_Pi). There are several versions of Raspbian including Raspbian Stretch and Raspbian Jessie. Since 2015 it has been officially provided by the [Raspberry Pi Foundation](https://en.wikipedia.org/wiki/Raspberry_Pi_Foundation) as the primary operating system for the family of Raspberry Pi [single-board computers](https://en.wikipedia.org/wiki/Single-board_computers). Raspbian was created by Mike Thompson and Peter Green as an independent project. The initial build was completed in June 2012. The operating system is still under active development. Raspbian is highly optimized for the Raspberry Pi line's low-performance [ARM](https://en.wikipedia.org/wiki/ARM_architecture) CPUs.

It is the core and central part of a robot on which the robot runs. This os is downloaded from the site <https://www.raspberrypi.org/downloads/raspbian/> . It is a Linux base operating system.

**2.2 Etcher:**

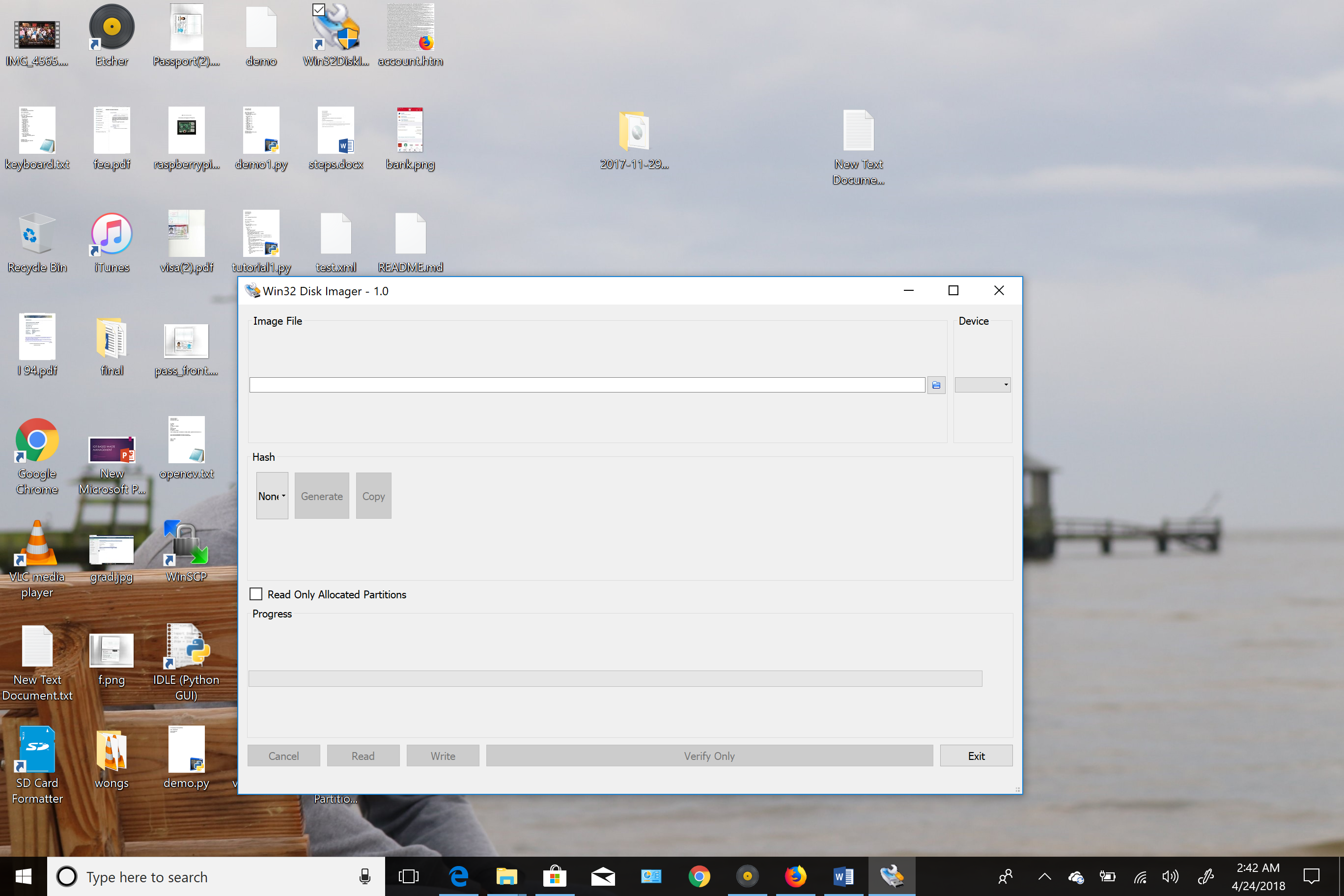
Etcher is an open source project by [resin.io](https://resin.io) which is used to mount the os image on to a sd card.



Etcher

**2.3 Win32Diskimager:**

A Windows tool for writing images to USB sticks or SD/CF cards. This program is designed to write a raw disk image to a removable device or backup a removable device to a raw image file. It is very useful for embedded development, namely Arm development projects. We can also read the data from the sd card and save it as a image which can be used as a backup when the data gets corrupted on the sd card.

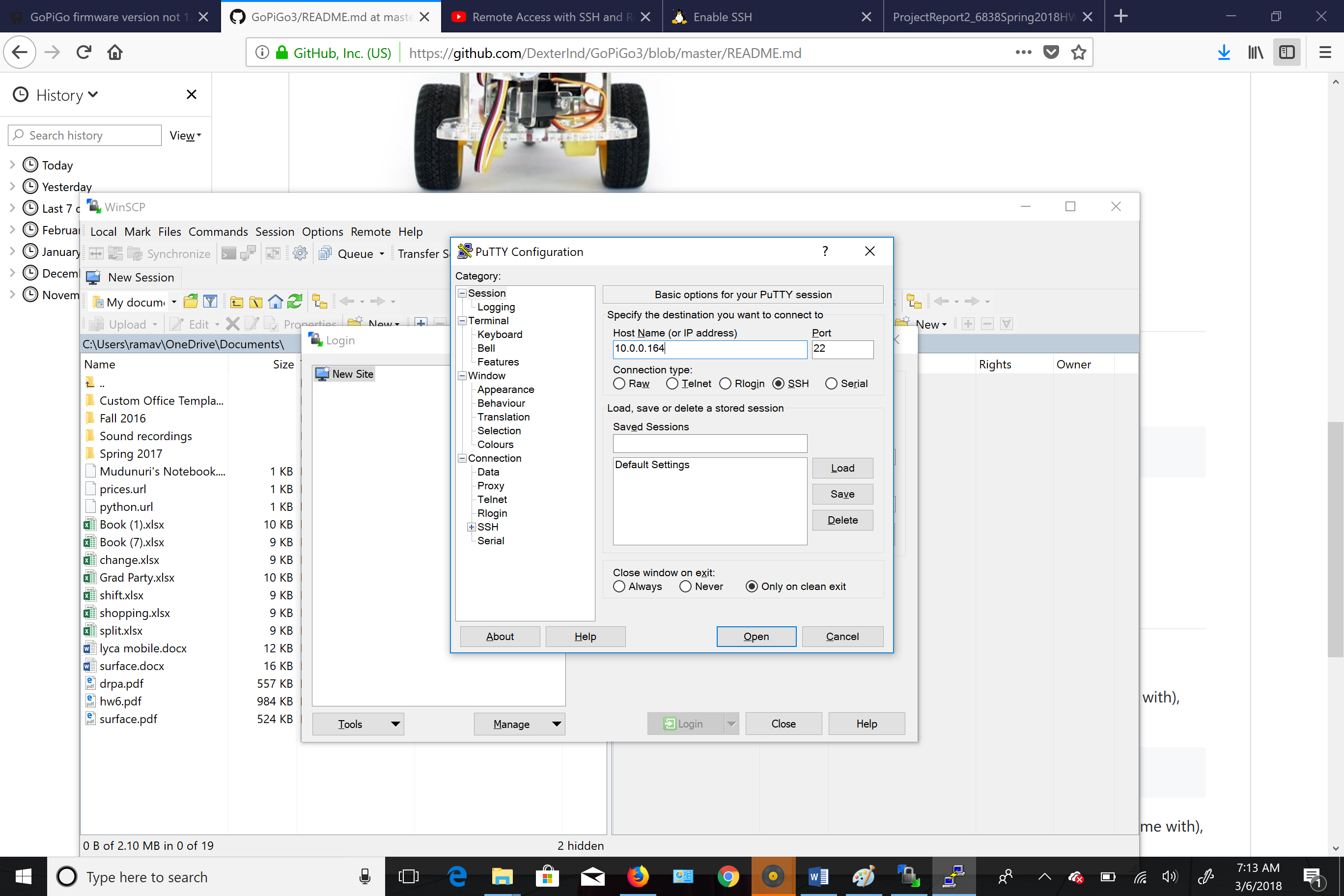


Win32diskimager

**2.4 Putty:**

Putty is a [free and open-source](https://en.wikipedia.org/wiki/Free_and_open-source) [terminal emulator](https://en.wikipedia.org/wiki/Terminal_emulator), [serial console](https://en.wikipedia.org/wiki/Serial_console) and network file transfer application. It supports several [network protocols](https://en.wikipedia.org/wiki/Network_protocol), including [SCP](https://en.wikipedia.org/wiki/Secure_copy), [SSH](https://en.wikipedia.org/wiki/Secure_Shell), [Telnet](https://en.wikipedia.org/wiki/Telnet), [rlogin](https://en.wikipedia.org/wiki/Rlogin), and raw socket connection. It can also connect to a [serial port](https://en.wikipedia.org/wiki/Serial_port). The name "PuTTY" has no official meaning.

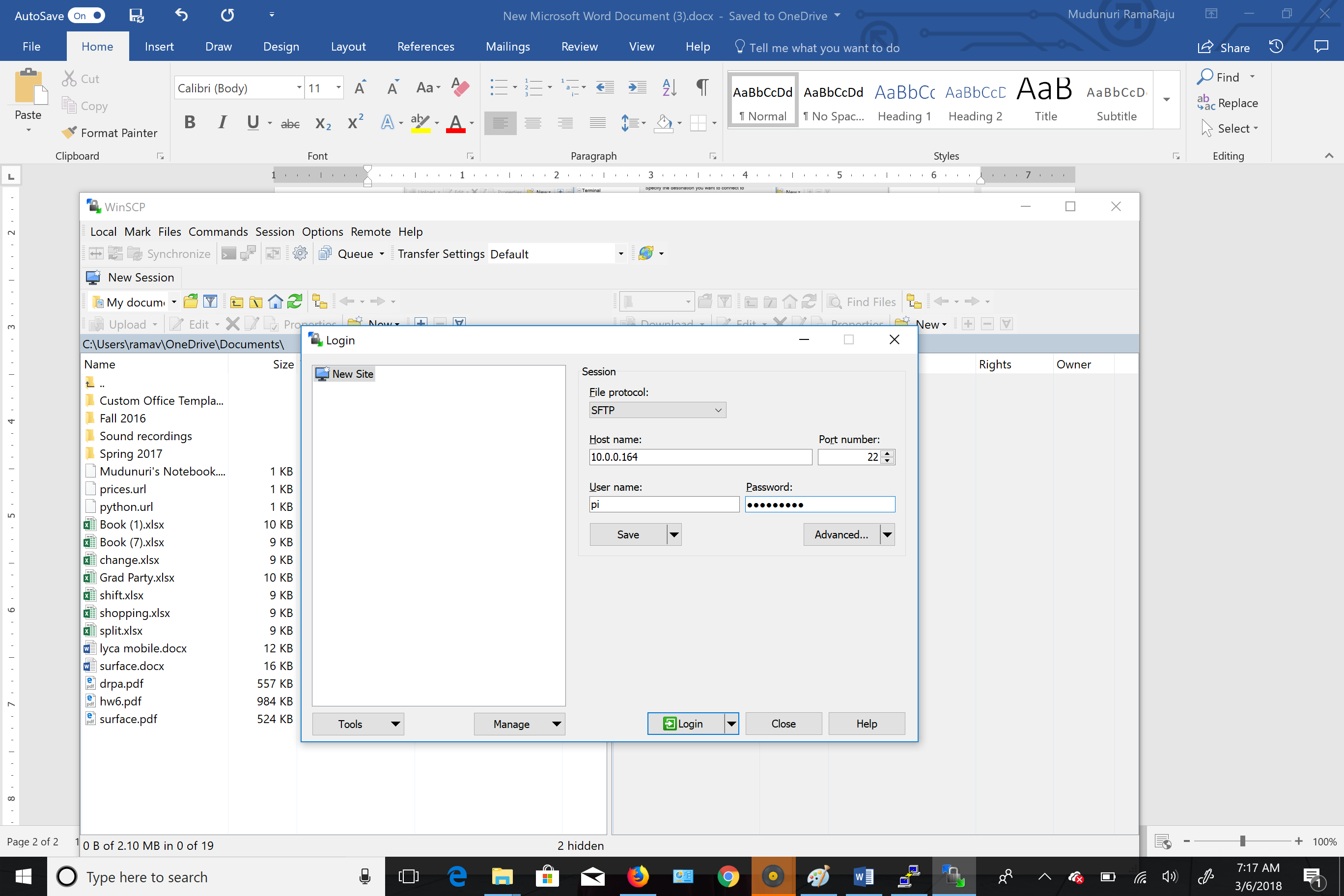
PuTTY was originally written for [Microsoft Windows](https://en.wikipedia.org/wiki/Microsoft_Windows), but it has been [ported](https://en.wikipedia.org/wiki/Porting) to various other [operating systems](https://en.wikipedia.org/wiki/Operating_system). Official ports are available for some [Unix-like](https://en.wikipedia.org/wiki/Unix-like) platforms, with work-in-progress ports to [Classic Mac OS](https://en.wikipedia.org/wiki/Classic_Mac_OS) and [macOS](https://en.wikipedia.org/wiki/MacOS), and unofficial ports have been contributed to platforms such as [Symbian](https://en.wikipedia.org/wiki/Symbian), [Windows Mobile](https://en.wikipedia.org/wiki/Windows_Mobile) and [Windows Phone](https://en.wikipedia.org/wiki/Windows_Phone).



Putty

**2.5 WinSCP:**

WinSCP (*Windows Secure Copy*) is a [free and open-source](https://en.wikipedia.org/wiki/Free_and_open-source) [SFTP](https://en.wikipedia.org/wiki/SSH_file_transfer_protocol), [FTP](https://en.wikipedia.org/wiki/File_Transfer_Protocol), [WebDAV](https://en.wikipedia.org/wiki/WebDAV), [Amazon S3](https://en.wikipedia.org/wiki/Amazon_S3) and [SCP](https://en.wikipedia.org/wiki/Secure_copy) client for [Microsoft Windows](https://en.wikipedia.org/wiki/Microsoft_Windows). Its main function is secure file transfer between a local and a [remote computer](https://en.wikipedia.org/wiki/Remote_computer). Beyond this, WinSCP offers basic [file manager](https://en.wikipedia.org/wiki/File_manager) and [file synchronization](https://en.wikipedia.org/wiki/File_synchronization) functionality. For secure transfers, it uses Secure Shell ([SSH](https://en.wikipedia.org/wiki/Secure_shell)) and supports the SCP protocol in addition to SFTP.

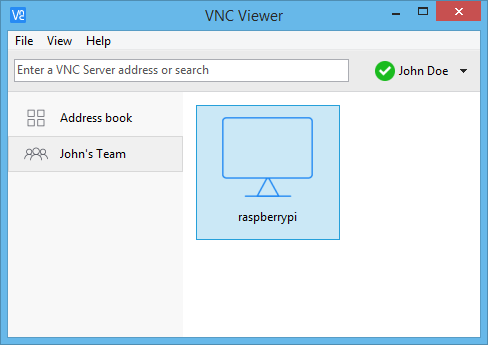


Winscp application

**2.6 Vnc Viewer:**

Virtual Network Computing (VNC) is a graphical [desktop sharing](https://en.wikipedia.org/wiki/Desktop_sharing) system that uses the [Remote Frame Buffer protocol (RFB)](https://en.wikipedia.org/wiki/RFB_protocol) to remotely control another [computer](https://en.wikipedia.org/wiki/Computer). It transmits the [keyboard](https://en.wikipedia.org/wiki/Computer_keyboard) and [mouse](https://en.wikipedia.org/wiki/Computer_mouse) events from one computer to another, relaying the graphical [screen](https://en.wikipedia.org/wiki/Computer_screen) updates back in the other direction, over a [network](https://en.wikipedia.org/wiki/Computer_network).

VNC is platform-independent – there are clients and servers for many GUI-based operating systems and for [Java](https://en.wikipedia.org/wiki/Java_Virtual_Machine). Multiple clients may connect to a VNC server at the same time. Popular uses for this technology include remote technical support and accessing files on one's work computer from one's home computer, or vice versa.



Vnc viewer

**2.7 Python:**

Python is an [interpreted](https://en.wikipedia.org/wiki/Interpreted_language) [high-level programming language](https://en.wikipedia.org/wiki/High-level_programming_language) for [general-purpose programming](https://en.wikipedia.org/wiki/General-purpose_programming_language). Created by [Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum) and first released in 1991, Python has a design philosophy that emphasizes [code readability](https://en.wikipedia.org/wiki/Code_readability), and a [syntax](https://en.wikipedia.org/wiki/Syntax_(programming_languages)) that allows programmers to express concepts in fewer [lines of code](https://en.wikipedia.org/wiki/Source_lines_of_code), notably using [significant whitespace](https://en.wikipedia.org/wiki/Significant_whitespace). It provides constructs that enable clear programming on both small and large scales.

Python features a [dynamic type](https://en.wikipedia.org/wiki/Dynamic_type) system and automatic [memory management](https://en.wikipedia.org/wiki/Memory_management). It supports multiple [programming paradigms](https://en.wikipedia.org/wiki/Programming_paradigm), including [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming), [imperative](https://en.wikipedia.org/wiki/Imperative_programming), [functional](https://en.wikipedia.org/wiki/Functional_programming) and [procedural](https://en.wikipedia.org/wiki/Procedural_programming), and has a large and comprehensive [standard library](https://en.wikipedia.org/wiki/Standard_library).

We install these following frameworks which we make use during python programming:

* GoPiGo3
* Opencv

**2.7.1 Gopigo 3:**

Gopigo 3 has the functionalities related to motors, encoders, gopigo3 servo, balanced bot with gopigo3 and gopigo3 LED’s. By using these functionalities in this project, we make the robot autonomous and can also operate with the help of a pc keyboard via remote desktop connection to pi 3

To install GoPiGo3 library using following commands

1. Clone this repository onto the Raspberry Pi:
2. **sudo git clone http://www.github.com/DexterInd/GoPiGo3.git /home/pi/Dexter/GoPiGo3**
3. Run the install script: **sudo bash /home/pi/Dexter/GoPiGo3/Install/install.sh**
4. Reboot the Raspberry Pi to make the settings take effect: **sudo reboot**

To update the gopigo3 library use the following command:

1. **sudo curl -kL dexterindustries.com/update\_gopigo3 | bash**

To have the additional functionalities related to sensor install easygopigo3 library which includes Button sensor, Buzzer sensor, Light sensor, Distance sensor and Line follower.

To install easygopigo3 sensor library using following command:

1. **sudo sh -c "curl -kL dexterindustries.com/update\_sensors | bash"**

**2.7.2 OpenCV:**

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding [14 million](https://sourceforge.net/projects/opencvlibrary/files/stats/timeline?dates=2001-09-20+to+2018-04-18). The library is used extensively in companies, research groups and by governmental bodies.

Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many startups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV. OpenCV’s deployed uses span the range from stitching street view images together, detecting intrusions in surveillance video in Israel, monitoring mine equipment in China, helping robots navigate and pick up objects at Willow Garage, detection of swimming pool drowning accidents in Europe, running interactive art in Spain and New York, checking runways for debris in Turkey, inspecting labels on products in factories around the world on to rapid face detection in Japan.

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, [Android](https://opencv.org/platforms/android/) and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured [CUDA](https://opencv.org/platforms/cuda.html) and [OpenCL](https://opencv.org/platforms/opencl.html) interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

In this project we are using opencv 3.4.1 version for detecting faces in a picture using cascading algorithm.

**3.Functional Design**

In this chapter we are going to discuss the hardware specifications like assembling a GoPiGo3, Block Diagram and about detailed specs of the individual modules used in constructing a robot.

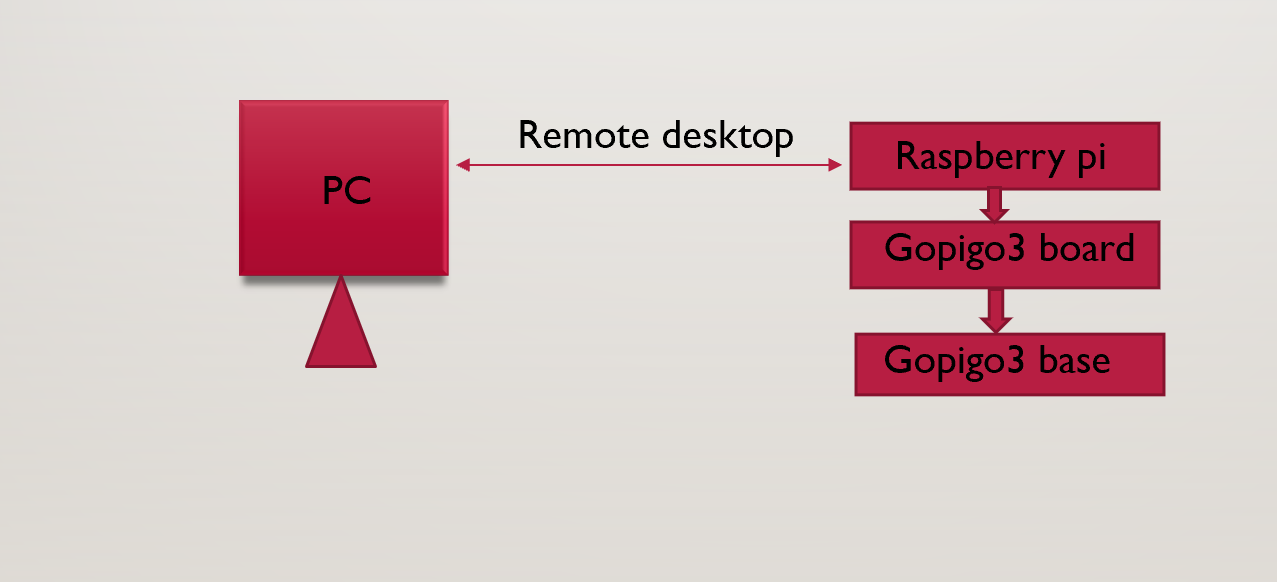
The steps for assembling gopigo3 are discussed below.

**3.1 Assembling Gopigo 3:**

Steps to assemble gopigo3 base kit

**3.2 Block Diagram:**

In this project to have communication between pi and pc we need connect them to a common network. Once, we connect we need to enable remote desktop connection on pi. So, that



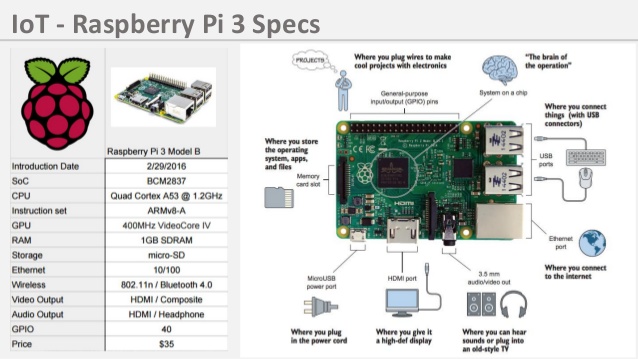
Block diagram

we can take full control of pi using PC. We also enable ssh o pi to have a secure connection between PC and pi. We write a code on PC and we can transfer that file using a WinScp application onto pi. We can also control the robot by using Keyboard of the PC by installing specific libraries.

**3.3 Modules Description:**

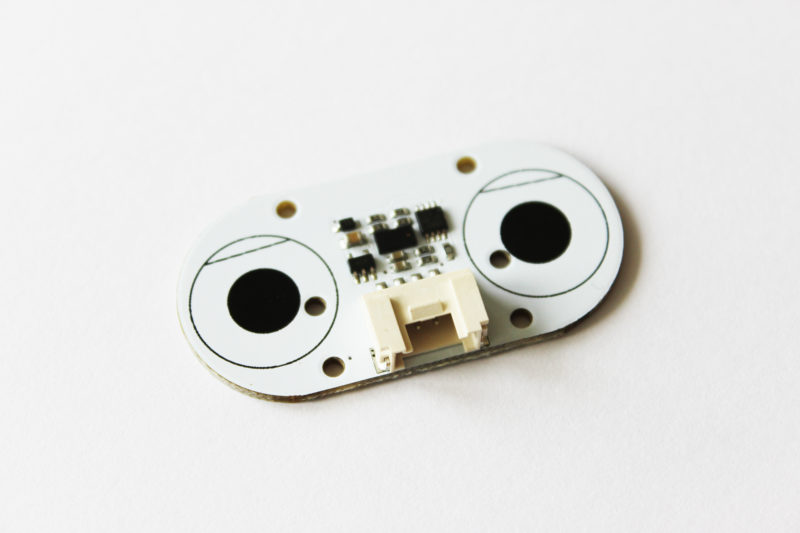
**3.3.1 Raspberry Pi 3 Specifications:**

Following picture describes the detail specifications of raspberry pi 3



Pi 3 Specifications

**3.3.2 Distance Sensor:**



Distance sensor

* It runs at 5V
* The distance sensor uses a small laser to determine the distance to an object. The sensor uses the Time of Flight method for a very fast and accurate distance reading
* The distance sensor has 1mm distance precision
* The laser beam cone is 35 degrees wide, and the sensor has 25 degrees of sensitivity
* The sensor range is from < 2cm up to just a little more than 2m

**3.3.3 Camera Module:**



Pi camera

* High-Definition video camera for Raspberry Pi Model A or B, B+, model 2, Raspberry Pi 3
* 5MPixel sensor with Omni vision OV5647 sensor in a fixed-focus lens
* Integral IR filter
* Still picture resolution: 2592 x 1944
* Max video resolution: 1080p

**3.3.4 Raspberry Pi 3 TFT LCD Display:**

It is a 3.5-inch resistive touch screen designed for Raspberry Pi 3B 2B/B+/A+ /A/B, 320\*480 resolution, better display. No external power supply needed



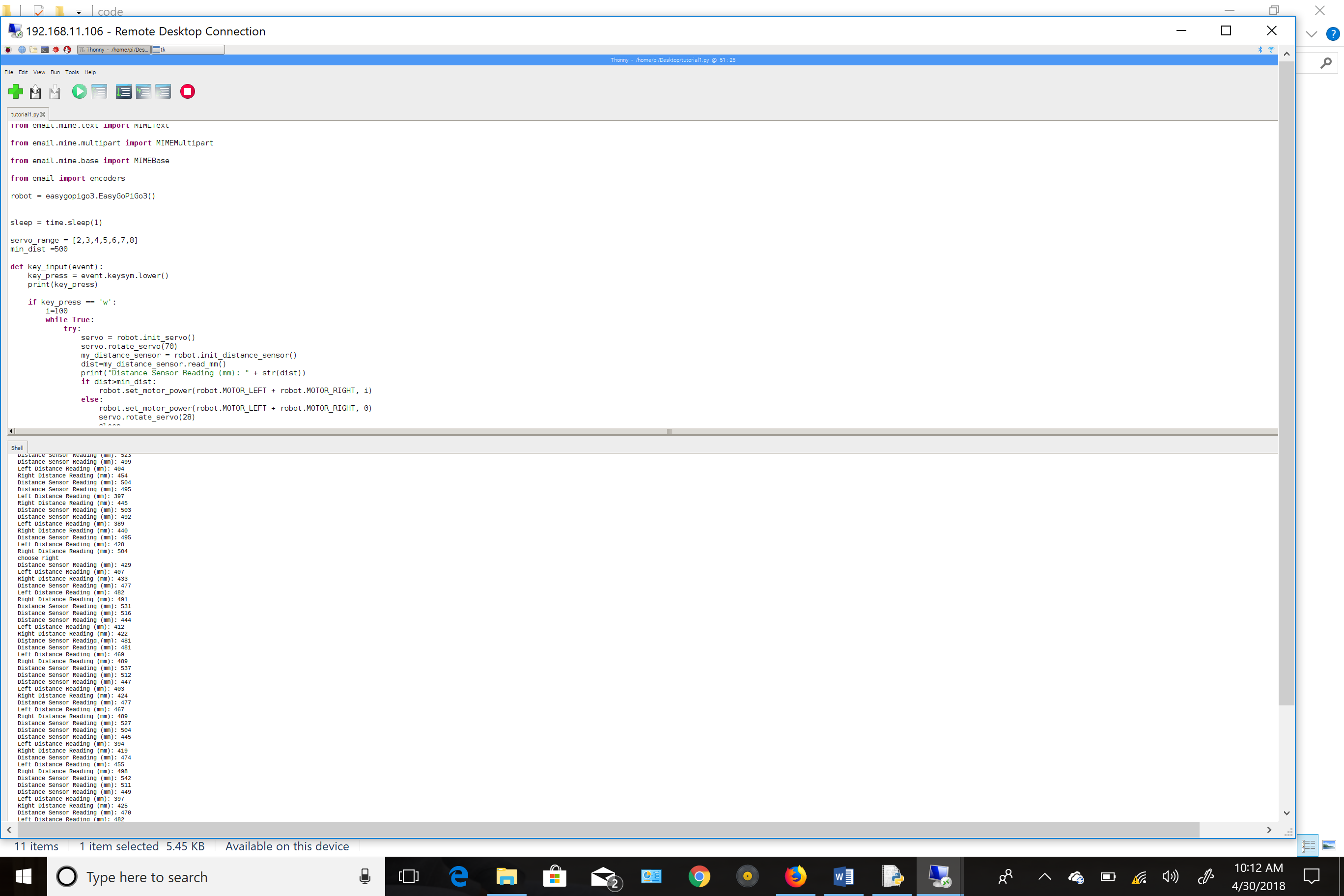
Pi Display

We mount the header of the display on to GopiGo board. To activate the display, we install the drivers using commands on the terminal

**4.Obstacle avoidance and face detection**

**4.1 Obstacle Avoidance:**

Obstacle avoidance is one of the most important aspects of mobile robotics. Without it robot movement would be very restrictive and fragile. To accomplish the task of obstacle avoidance within the home environment we are going to use a distance sensor which can detect object before 3 meters. In this project, we are setting threshold of 50 cm. if robot is encountered by an object less than 50cm we are going to measure the left and right distance which is at an angle 28 degrees (left) and 112 degrees (right) respectively with the help of a servo motor. After measuring left and right distances, if the left distance greater than right distance then we are going to rotate the robot towards left and if the right distance is greater than left distance we are going to rotate the robot towards right. If both left and right left distances are less than the threshold we are going to move the robot either backwards or rotate. This entire thing is written in the form of a code which is kept in an infinite loop to make the robot autonomous. We can also control the robot with the help of keyboard of a pc by assigning a specific functionality to each key. As we are using a single distance sensor we are not able avoid the obstacles under certain situations, but this can be resolved by adding more distance sensor to the robot

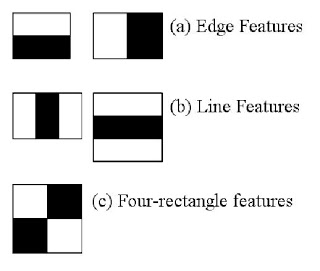


Real time readings of distance sensor

**4.2 Face detection:**

Face detection is a computer technology being used in a variety of applications that identifies human faces in digital images. Face detection also refers to the psychological process by which humans locate and attend to faces in a visual scene. In this project we are using OpenCV library for face detection. We are using a cascading algorithm to detect faces in an image. Cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Here we will work with face detection. Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. Then we need to extract features from it. For this, Haar features shown in the below image are used. They are just like our convolutional kernel. Each feature is a single value obtained by subtracting sum of pixels under the white rectangle from sum of pixels under the black rectangle.



Some features of cascade

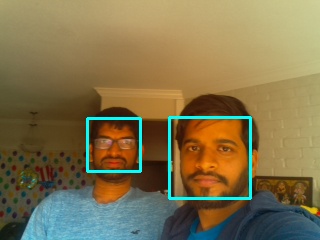
Now, all possible sizes and locations of each kernel are used to calculate lots of features. For each feature calculation, we need to find the sum of the pixels under white and black rectangles. To solve this, they introduced the integral image. However large your image, it reduces the calculations for a given pixel to an operation involving just four pixels.

In an image, most of the image is non-face region. So, it is a better idea to have a simple method to check if a window is not a face region. If it is not, discard it in a single shot, and don't process it again. Instead, focus on regions where there can be a face. This way, we spend more time checking possible face regions.

For this they introduced the concept of Cascade of Classifiers. Instead of applying all 6000 features on a window, the features are grouped into different stages of classifiers and applied one-by-one. (Normally the first few stages will contain very many fewer features). If a window fails the first stage, discard it. We don't consider the remaining features on it. If it passes, apply the second stage of features and continue the process. The window which passes all stages is a face region.

Thedetector had 6000+ features with 38 stages with 1, 10, 25, 25 and 50 features in the first five stages. on average 10 features out of 6000+ are evaluated per sub-window.

In this project we are converting the raw image into 320x240 resolution and we are applying cascade on that image. cascade is xml file which the has features to detect the multiple faces in an image. If a face is detected a square box is drawn on the face and the image is saved on the desktop. This image can be sent to a person through an email using python script.



Real time face detection using robot

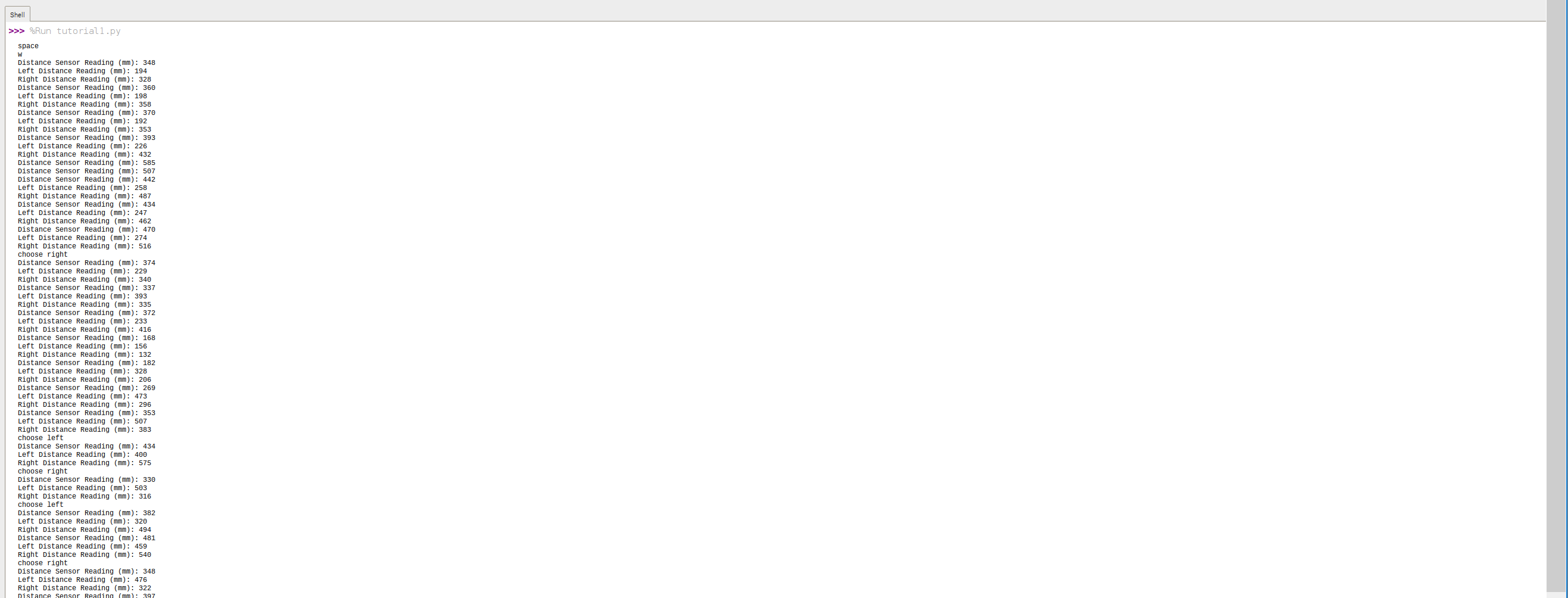
**5.Results**

We have attained some decent results with obstacle avoidance and face detection using a GoPiGo3 robot.

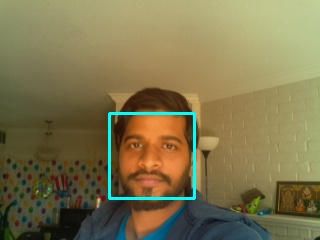
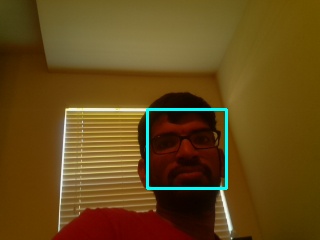
We have encountered problems with obstacle avoidance as we are using a single distance sensor we are not able avoid the obstacles under certain situations, but this can be resolved by adding more distance sensor to the robot.

As we are using a pi camera the face detection doesn’t work under low light situation and also when the subject is far from the camera. This can also be resolved by using a camera with high resolution.

The following images show the readings of the distance sensor and face detection in an image



Distance sensor readings



Results of Face detection

**6.Conclusion**

Finally, we have made a robot which can avoid obstacles using a distance sensor and also detect multiple faces in an image using cascading algorithm. The robot can be used monitor the old people or children in a house by connecting remotely using a PC. We can also monitor them by live video feed. By adding different hardware modules, we can extend the capabilities of the robot.

**7.Coding**

import tkinter as tk

import easygopigo3

import picamera

import io

import time

import smtplib

import cv2

import numpy

from email.mime.text import MIMEText

from email.mime.multipart import MIMEMultipart

from email.mime.base import MIMEBase

from email import encoders

robot = easygopigo3.EasyGoPiGo3()

sleep = time.sleep(1)

servo\_range = [2,3,4,5,6,7,8]

min\_dist =500

def key\_input(event): **#controlling robot using keys**

key\_press = event.keysym.lower()

print(key\_press)

if key\_press == 'w': **#autonomy**

i=100

while True: **#infinite loop**

try:

servo = robot.init\_servo()

servo.rotate\_servo(70)

my\_distance\_sensor = robot.init\_distance\_sensor()

dist=my\_distance\_sensor.read\_mm()

print("Distance Sensor Reading (mm): " + str(dist)) **# display distance sensor reading**

if dist>min\_dist: **#If measured distance is greater than min distance move forward**

robot.set\_motor\_power(robot.MOTOR\_LEFT + robot.MOTOR\_RIGHT, i)

else:

robot.set\_motor\_power(robot.MOTOR\_LEFT + robot.MOTOR\_RIGHT, 0)**#stop robot**

servo.rotate\_servo(28) **#rotate using servo to 28 degrees left**

sleep

left\_dir = my\_distance\_sensor.read\_mm() **# measure left distance**

print("Left Distance Reading (mm): " + str(left\_dir))

servo.rotate\_servo(112) **# rotate using servo to 112 degrees**

sleep

right\_dir = my\_distance\_sensor.read\_mm() **# measure right distance**

print("Right Distance Reading (mm): " + str(right\_dir))

if left\_dir < min\_dist and right\_dir < min\_dist: **# if left and right distance less than min distance move backward and rotate right**

robot.set\_motor\_power(robot.MOTOR\_LEFT + robot.MOTOR\_RIGHT, -i)

robot.right()

sleep

elif left\_dir > right\_dir and left\_dir > min\_dist: **# if left distance greater than right and min distance rotate left**

print('choose left')

robot.left()

sleep

elif left\_dir < right\_dir and right\_dir > min\_dist: **# if right distance greater than left and min distance rotate right**

print('choose right')

robot.right()

sleep

except KeyboardInterrupt: **# press any key to stop the robot**

robot.set\_motor\_power(robot.MOTOR\_LEFT + robot.MOTOR\_RIGHT, 0)

elif key\_press == 's': **# to move backward**

i=100

robot.set\_motor\_power(robot.MOTOR\_LEFT + robot.MOTOR\_RIGHT, -i)

elif key\_press == 'a': **# to move left**

robot.left()

elif key\_press == 'e': **# to email the image**

email\_user = 'XXXXX@gmail.com'

email\_password = 'XXXX'

email\_send = 'XXXX@gmail.com'

subject = 'sending mail from gopigo'

msg = MIMEMultipart()

msg['From'] = email\_user

msg['To'] = email\_send

msg['Subject'] = subject

body = 'Hi there, sending this email from Python!'

msg.attach(MIMEText(body,'plain'))

filename='result.jpg'

attachment =open(filename,'rb')

part = MIMEBase('application','octet-stream')

part.set\_payload((attachment).read())

encoders.encode\_base64(part)

part.add\_header('Content-Disposition',"attachment; filename= "+filename)

msg.attach(part)

text = msg.as\_string()

server = smtplib.SMTP('smtp.gmail.com',587)

server.starttls()

server.login(email\_user,email\_password)

server.sendmail(email\_user,email\_send,text)

server.quit()

elif key\_press == 'i': **# face detection**

**#Create a memory stream so photos doesn't need to be saved in a file**

stream = io.BytesIO()

**#Get the picture (low resolution, so it should be quite fast)**

**#Here you can also specify other parameters (e.g.:rotate the image)**

with picamera.PiCamera() as camera:

camera.resolution = (320, 240)

camera.capture(stream, format='jpeg')

**#Convert the picture into a numpy array**

buff = numpy.fromstring(stream.getvalue(), dtype=numpy.uint8)

**#Now creates an OpenCV image**

image = cv2.imdecode(buff, 1)

**#Load a cascade file for detecting faces**

face\_cascade = cv2.CascadeClassifier('/home/pi/Desktop/test.xml')

**#Convert to grayscale**

gray = cv2.cvtColor(image,cv2.COLOR\_BGR2GRAY)

**#Look for faces in the image using the loaded cascade file**

faces = face\_cascade.detectMultiScale(gray, 1.1, 5)

print ("Found "+str(len(faces))+" face(s)")

**#Draw a rectangle around every found face**

for (x,y,w,h) in faces:

cv2.rectangle(image,(x,y),(x+w,y+h),(255,255,0),2)

**#Save the result image**

cv2.imwrite('result.jpg',image)

elif key\_press =='d': **# to move right**

robot.right()

elif key\_press == 'space': **# to stop**

robot.set\_motor\_power(robot.MOTOR\_LEFT + robot.MOTOR\_RIGHT, 0)

elif key\_press == 'c': **# to capture image**

camera = picamera.PiCamera()

camera.capture('image.jpg')

elif key\_press == 'v': **# to measure the voltage of battery**

v=robot.volt()

print(v)

elif key\_press == 'r': **# to record a video for 5 seconds**

camera = picamera.PiCamera()

camera.start\_recording('video.h264')

time.sleep(5)

camera.stop\_recording()

elif key\_press == 'u': **# to mesaure the distance**

my\_distance\_sensor = robot.init\_distance\_sensor()

print("Distance Sensor Reading (mm): " + str(my\_distance\_sensor.read\_mm()))

elif key\_press.isdigit(): **# to control the robot using numbers**

servo = robot.init\_servo()

servo.rotate\_servo(int(key\_press)\*14)

time.sleep(1)

command = tk.Tk()

command.bind\_all('<Key>', key\_input)

command.mainloop()

**8.Installation Procedure**

**Step 1: Assemble the gopigo3 kit**

**Step 2: Write the the Raspbian os on to sd card using etcher application**

**Step 3: Mount the sd card on to raspberry pi and connect power supply to raspberry pi**

**Step 4: Using the hdmi cable connect the raspberry pi to a monitor to connect pi to the internet for the first time**

**Step 5: Connect the pi to wifi using your wifi credentials.**

**Step 6: Type the following commands in the terminal**

Sudo apt-get update && upgrade (will update and upgrade the os to newer version)

Sudo apt-get install tightvncserver (will install vnc server)

Sudo apt-get install xrdp (will enable remote desktop connection)

Sudo apt-get install python python-tk (GUI interface for python2)

Sudo apt-get install python python3-tk(GUI interface for python3)

Sudo reboot ( will reboot pi)

**Step 7: To enable the interfaces and ssh type the following command**

Sudo raspi-config

Enable the following under interfaces:

* Camera
* Spi
* I2c
* Ssh

Reboot the pi

**Step 8: To install the gopigo3 framework following commands :**

You can install it in two ways:

* Clone this repository onto the Raspberry Pi:
* sudo git clone http://www.github.com/DexterInd/GoPiGo3.git /home/pi/Dexter/GoPiGo3
* Run the install script: sudo bash /home/pi/Dexter/GoPiGo3/Install/install.sh
* Reboot the Raspberry Pi to make the settings take effect: sudo reboot

Or else type the following command in terminal

* sudo curl -kL dexterindustries.com/update\_gopigo3 | bash

**Step 9: To install the framework related to functionalities of sensors use the following command:**

sudo sh -c "curl -kL dexterindustries.com/update\_sensors | bash"

**Step 10: To send a email using python script we use ssmtp library. To install**

**the library use the following command:**

sudo apt-get install ssmtp mailutils

**Step 11: To get the video feed install vlc player using the following command:**

sudo apt-get install vlc

**Step 12: To install Opencv use the following commands**

**Step i: Install and update os and install cmake**

* sudo apt-get update && sudo apt-get upgrade
* sudo apt-get install cmake(CMake is an open-source, cross-platform family of tools designed to build, test and package software)

**Step ii: Install python3**

* sudo apt install python3 python3-setuptools python3-dev python3-venv -y
* pip3 install numpy

**Step iii: Install Dependencies**

* sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng12-dev -y
* sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev -y
* sudo apt-get install libxvidcore-dev libx264-dev -y
* sudo apt-get install libgtk2.0-dev -y
* sudo apt-get install libatlas-base-dev gfortran -y
* sudo apt-get install python3-dev python3-venv -y

**Step iv: Download the OpenCV source code**

* cd ~
* mkdir src
* cd src
* wget -O opencv.zip https://github.com/opencv/opencv/archive/3.4.1.zip
* unzip opencv.zip

**It's recommended to include contrib package.**

* wget -O opencv\_contrib.zip https://github.com/opencv/opencv\_contrib/archive/3.4.1.zip
* unzip opencv\_contrib.zip

**Step v: Compile and Install OpenCV**

* cd opencv-3.4.1
* mkdir build
* cd build
* cmake -D CMAKE\_BUILD\_TYPE=RELEASE \

-D CMAKE\_INSTALL\_PREFIX=/usr/local \

-D BUILD\_opencv\_java=OFF \

-D BUILD\_opencv\_python2=OFF \

-D BUILD\_opencv\_python3=ON \

-D PYTHON\_DEFAULT\_EXECUTABLE=$(which python3) \

-D INSTALL\_C\_EXAMPLES=OFF \

-D INSTALL\_PYTHON\_EXAMPLES=OFF \

-D BUILD\_EXAMPLES=OFF \

-D BUILD\_TESTS=OFF \

-D BUILD\_PERF\_TESTS=OFF \

-D OPENCV\_EXTRA\_MODULES\_PATH=../../opencv\_contrib-

3.4.1/modules \

-D WITH\_CUDA=OFF \

..

**In the Python section of the log, you should see something like this:**

-- Python 3:

-- Interpreter: /usr/bin/python3 (ver 3.5.3)

-- Libraries: /usr/lib/arm-linux-gnueabihf/libpython3.5m.so

(ver 3.5.3)

-- numpy: /home/yyoo/.local/lib/python3.5/site-

packages/numpy/core/include (ver 1.13.1)

-- packages path: lib/python3.5/dist-packages

--

-- Python (for build): /usr/bin/python3

**Now, let's build & install it.**

* make -j4 (During this command it make use of 4 cores)
* sudo make install

**Due to some bugs, the name of the Python binding library is not correct.**

**During the installation, you will see log like this.**

**-**- Installing: /usr/local/lib/python3.5/dist-packages/cv2.cpython-35m-

arm-linux-gnueabihf.so

**Let's fix it with a symbolic link**

* ln -s /usr/local/lib/python3.5/dist-packages/cv2.cpython-35m-arm-linux-gnueabihf.so /usr/local/lib/python3.5/dist-packages/cv2.so

**Now, you're ready to use OpenCV from Python system-wide. Testing with**

**Python 3.5**

$ python3

Python 3.5.3 (default, Jan 19 2017, 14:11:04)

[GCC 6.3.0 20170124] on linux

Type "help", "copyright", "credits" or "license" for more information.

>>> import cv2

>>> cv2.\_\_version\_\_'3.4.0'

**Step 13: To use 3.5 ‘pi display. Mount the header of display on to gopigo board. To activate display, use the following commands:**

clone the repository onto pi

* sudo rm -rf LCD-show
* git clone https://github.com/goodtft/LCD-show.git
* chmod -R 755 LCD-show
* cd LCD-show/

**In case of 3.5" LCD**

* sudo ./LCD35-show

**If you need to switch back to the traditional HDMI display**

* sudo ./LCD-hdmi

**9.Cost Analysis**

The following table and pie chart depicts the cost factors involved in this project

|  |  |  |
| --- | --- | --- |
| Items | Price | Remarks |
| GoPiGo3 Base Kit | $94.01 |  |
| Raspberry Pi 3 | $37.99 |  |
| Rechargeable Batteries | $20.99 |  |
| Distance sensor | $21.99 |  |
| Servo Motor | $14.99 |  |
| Pi Camera | $13.49 |  |
| Sd card (32gb) | $12.99 |  |
| Pi Display | $19.39 |  |
|  |  |  |
| Total | $235.84 |  |

Hardware cost analysis

Cost analysis using pie chart

**10.Troubleshooting**

In this project we have faced some problems and resolved them successfully. We are going to address these problems in this section.

First one is while we are installing OpenCV library on Raspbian OS. We can install the library, but we cannot import it in python. We faced an issue regarding OpenCV binding with python. After trying different installation procedures finally, we found a way to import OpenCV to python. This installation process is addressed in the installation procedure section.

Second one is while installing OpenCV we use a command called ‘make -j4’ which uses the four cores of the processor to make the installation process fast. During this process the processor gets hot and it may damage. To cool down the processor arrange a fan or use a command ‘make’ but it will take a lot of time for installation.

Third one is when you install the drivers of the Pi display and activate it. We cannot import some of the libraries like easygopigo3 which has the functionalities of the robot. We can use these functionalities by disabling LCD screen and by activating HDMI. These steps are addressed in installation procedure.

**11.Glossary**

**SSMTP - Secure simple mail transfer protocol**

**SSH - Secure Shell**

**WinScp - Windows Secure Copy**

**OpenCV - Open Source Computer Vision**

**References**

1. Raspberry pi 3

<https://www.raspberrypi.org/>

1. Proximity Sensor

<https://en.wikipedia.org/wiki/Proximity_sensor>

1. Camera Module

<https://en.wikipedia.org/wiki/Camera_module>

1. GoPiGo3 Documentaion

<https://www.dexterindustries.com/gopigo3-tutorials-documentation/>

1. <https://realpython.com/face-recognition-with-python/>
2. Opencv installation

<https://github.com/ys7yoo/PiOpenCV/blob/master/README.md>

1. Face Detection

<https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html>

1. k. sridharan and p. rajesh kumar: robotic exploration and landmark determination, studies in computational intelligence (sci) 81, 13–23 (2008)
2. General survey of face recognition

<http://shodhganga.inflibnet.ac.in/bitstream/10603/29248/7/07_chapter2.pdf>