IoT BASED ROVER FOR EXPLORATION

A PROJECT REPORT

Submitted in partial fulfillment of the requirement for the award of the Degree of

BACHELOR OF TECHNOLOGY
IN
ELECTRONICS AND COMMUNICATION ENGINEERING

by

TIRUMALASETTY VENKATA SAI KARTHIK (15BEC1002)
KOPPAKA SHIVADEEP (15BEC1117)
CHAKKIRALA NIKHIL CHAKRAVARTHY (15BEC1184)

Under the Guidance of

PROF. MUTHULAKSHMI S

SCHOOL OF ELECTRONICS ENGINEERING
VELLORE INSTITUTE OF TECHNOLOGY
CHENNAI - 600127

April 2019

CERTIFICATE

This is to certify that the Project work titled "***IoT based rover***" that is being submitted by ***T.V.S. Karthik (15BEC1002), K. Shivadeep (15BEC1117)*** and ***Ch. Nikhil Chakravarthy (15BEC1184)*** is in partial fulfillment of the requirements for the award of **Bachelor of Technology in Electronics and Communication Engineering**, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

XXXX
Guide

The Project Report is satisfactory / unsatisfactory

Internal Examiner **External Examiner**

Approved by

PROGRAM CHAIR
B. Tech. (ECE)

DEAN
School of Electronics Engineering

ACKNOWLEDGEMENTS

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Prof. Muthulakshmi S.**, School of Electronics Engineering, for her consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to **Dr. Sreedevi.V.T**, Dean of the School of Electronics Engineering, VIT Chennai, for extending the facilities of the School towards our project and for her unstinting support.

We express our thanks to our Programme Chair **Dr. Vetrivelan. P** and Co-Chair **Prof. Reena Monica.P** for their support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

T.V.S KARTHIK	**K. SHIVADEEP**	**CH. NIKHIL**
(15BEC1002)	**(15BEC1117)**	**(15BEC1184)**

ABSTRACT

A tiny part of the world is always remained untouched since man had begun his exploration due to many unknown dangers. But now robotic exploration is allowing us to virtually venture places that we never considered going. Be it archeological, metallurgical, or any rescue operations unknown dangers to human life is always there. By using these kind of rovers the objective to explore unmanned places is achieved without posing any kind of threat to human life. Dangers generally include toxic fumes, sudden avalanches, deep pits etc.

Robots can consistently help human operators in dangerous tasks during rescue operations in several ways. Indeed, one of the main services that mobile robots can provide to rescue operators is to work as remote sensing devices reporting information from dangerous places that human operators cannot easily and/or safely reach.

We have included several sensors to give a rough estimate about the area that is to be explored. These include temperature, humidity, toxic gases, a camera to give live feed of the location, a speaker to get noticed by the people in danger.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

This rover is used for surveillance of the ruins of collapsed buildings to detect if there is any life present there, how the condition of that area is, detecting the presence of gases and providing a live video of the environment. This rover has a dedicated webpage which is used to visualize the ruins for the user to make it easier for the rescuers. It simply operates by going to the region which is out of human reach and provides a live stream of the environment of that region. There is a temperature sensor that gives the temperature of that region a gas sensor to identify any gas leakage in the environment.
A part from being used as a rescue rover a few modifications can always be made to serve the purpose of interest. Like a robotic arm can be added to collect samples from the location. These kind of rovers are already serving their purpose in mining of radioactive metals which can be dangerous if exposed to humans. Archaeological explorations can also be made where human reach is a little difficult in some places.

1.1 Objectives

The following are the objectives of the project:

- To build a prototype rover that has the capability of video and audio live streaming the surroundings.
- Able to make the rover will be able to move on rough terrain.
- To collect all the sensor values.
- Controlling the rover over a webpage or through a terminal.
- To show various parameters of the location in real time.

1.2 Background and Literature survey

A similar type of project has been done by the faculty and students of ABESIT Ghaziabad they used Arduinolilypad which is embedded with multiple sensors, they used gesture control along with accelerometer to control the rover. They used RF for transmission and reception of the signals, and motor driver L293D to drive the motors.
They proposed a low cost, low data rate design since they are using RF modules the communication range is limited to about 40 meters with good antenna design, and the

execution of the signals is sometimes misinterpreted by the decoder due to addition of noise during the transmission.

Table 1.1 shows similar kind of projects along with their research focus

Sl.No	Name of article	Name of journal	Information included
1.	A Cost Effective Way to Build a Web Controlled Search and CO Detector Rover	978-1-5090-4228-9/17/$31.00 ©2017 IEEE	• Temperature detection. • CO detection.
2.	Cost Effective Motion Based Stair Climbing Rover for Rescue Purpose	Proceedings of IEEE International Conference on Applied System Innovation 2018	• Climbing Staircases. • Object picking.
3.	Disaster Relief and Data Gathering Rover	978-1-5090-6785-5/18/$31.00 © 2018 by IEEE	• Speed-up the rescue process by gathering information.
4.	Terrestrial Land Rover with Dedicated Website	2018 2nd International Conference on Smart Sensors and Application (ICSSA)	• Developing a webpage to control the rover along with the sensors.
5.	Implementation of Reconfigurable Robot Mechanism for Earthquake Rescue Operation	International Science Press	• Developing a design to work in multiple terrains.
6.	Improving Robot Plans for Information Gathering Tasks through Execution Monitoring	International Conference on Intelligent Robots and Systems (IROS) November 3-7, 2013. Tokyo, Japan	• To use effective algorithms which make the rover work better.

1.3 Organization of the Report

The remaining chapters of the project report are described as follows:

- Chapter 2 contains the methodology, design approach, standards used and constraints of the methodology used in the project.
- Chapter 3 contains about how to setup Raspberry Pi for the project and the entire implementation of the project
- Chapter 4 Cost analysis part is dealt here
- Chapter 5 Results obtained and discussions about them
- Chapter 6 Conclusions and Future work

Chapter 2
ABOUT THE ROVER

This chapter describes the methodology, design, standards, components and the role of IoT in this project.

2.1 Methodology

The rover has been designed with ultrasonic sensors, temperature and humidity sensor, CO gas detection sensor, Camera for live video feed. All these sensors are connected to the GPIO pins of the raspberry pi 3 model B+. The data acquired from these sensors are shown in ThingSpeak platform to give a rough estimate about the location and the operator can decide his further steps in controlling of the bot. This rover is not built to operate autonomously thinking of unknown obstacles and the reactions it has to perform, which increases the complexity in coding and execution.

The rover can be controlled by either using the terminal or through the webpage. OpenCV is used to detect human presence through the camera. Speaker is also connected to the audio jack of Pi to give a signal in the location mentioning the rover’s presence.

Coming to the motors used in the rover, these are 12V rated motors with almost 10Kg torque making it easy for the rigid and the heavy rover cross the rough terrain at ease. The power source for the rover is a 12V battery. The motors are controlled using a relay board unlike motor driver ICs, as the maximum current output from the motor driver L298N is only 2Amps which is not sufficient to drive the motors which have the capacity to use current up to 4.5A.

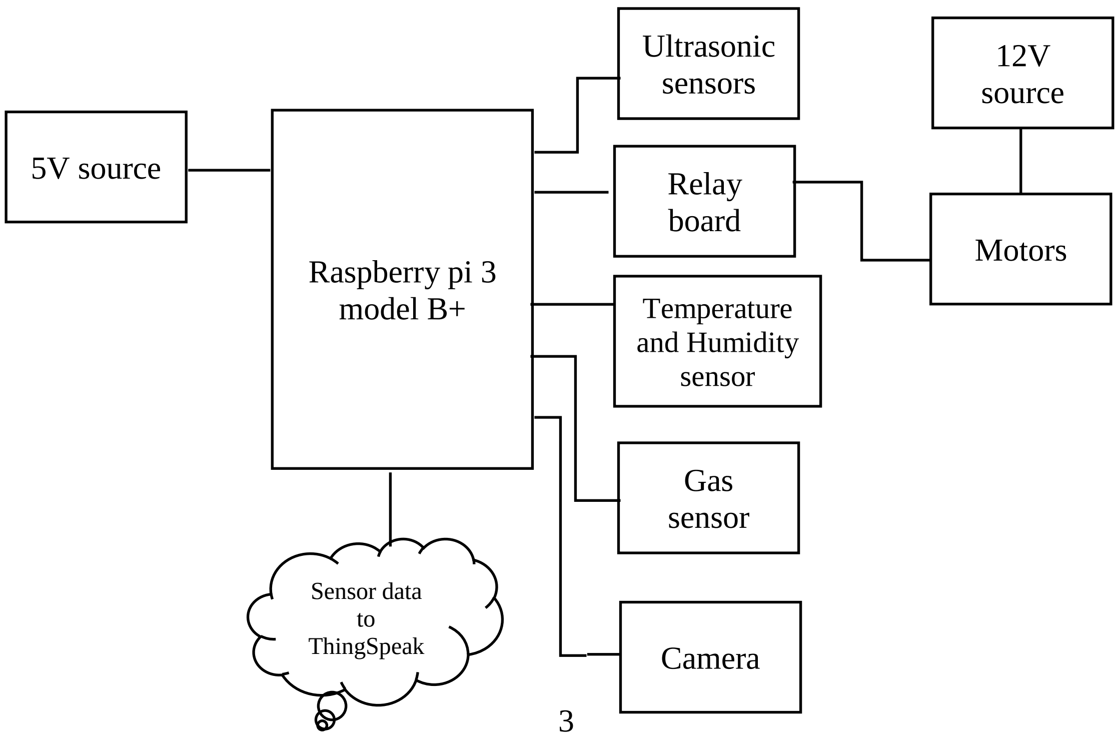


Figure 2.1 Physical Block Diagram

2.2 Design Approach

The rover is built on a custom design chassis, made of UPVC pipes and a wooden plank, to reduce the weight as much as possible but not sacrificing the strength and rigidity. It is a 4-wheel rocker design rover with the legs inclined at 45-degree angle to prevent toppling in case of rough terrain and also to efficiently use the torque of the powerful 12v DC motors.

The rover has multiple functionalities which would require multiple programs to be running simultaneously. In order to prevent any haults, a python webserver has been created to showcase the basic movement controls and the camera feed along with the orientation of camera. Any other program can be easily implemented in this webserver just by programming the webserver. This allows for an easier and efficient operation of the rover.

2.3 Standards

- WLAN IEEE 802.11ac/n (For hotspot)
- IEEE 1174-2000 (Serial interface between camera module and RaspberryPi)
- GNU Nano 2.7.4 editor to work with Raspberry pi
- HTML5 Hypertext Markup Language (HTML) and Cascading Style Sheet (CSS)-language for creating web pages.

2.4 Constrains and Trade offs

- **Constraint 1:** Frame rate and quality is not up to the mark.

  Alternative: To use RaspberryPi camera.

  Trade-off: Better picture quality with the webcam.
- **Constraint 2:** Heavy power consumption by the motors.

  Alternative: To use less powerful motors for utilizing less power.

  Trade-off: Better mobility.
- **Constraint 3:** Difficult to fit in tight places less than three feet.

  Alternative: To use a smaller chassis.

Trade-off: Highly customizable due to larger real estate.

- **Constraint 4:** Speed control is compromised because of relays.

  Alternative: To use a motor driver IC-L298N for driving the 12V DC motors.

  Trade-off: Ability to use the maximum current provided by the battery. Independent power distribution to the motors.

2.5 Technical Specifications

Temperature and Humidity sensor(Dht11):

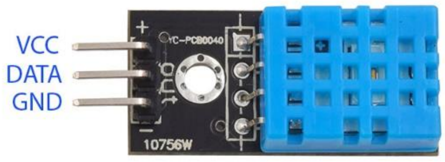


Figure 2.2 Dht11 Sensor

- Operating Voltage: 3.5V to 5.5V
- Operating current: 0.3mA (measuring) 60uA (standby)
- Output: Serial data
- Temperature Range: 0°C to 50°C
- Humidity Range: 20% to 90%
- Resolution: Temperature and Humidity both are 16-bit
- Accuracy: ±1°C and ±1%

Ultrasonic sensor:

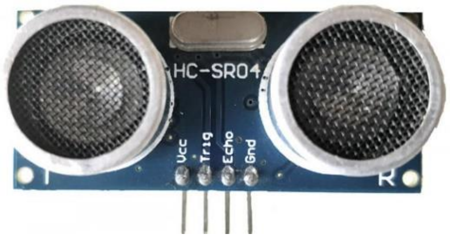


Figure 2.3 Ultrasonic Sensor

- Supply voltage: 5V (DC).
- Supply current: 15mA.

- Modulation frequency: 40Hz.
- Output: 0 – 5V (Output high when obstacle detected in range).
- Beam Angle: Max 15 degree.
- Distance: 2cm – 400cm.
- Accuracy: 0.3cm.
- Communication: Positive TTL pulse.

Gas sensor(MQ2):

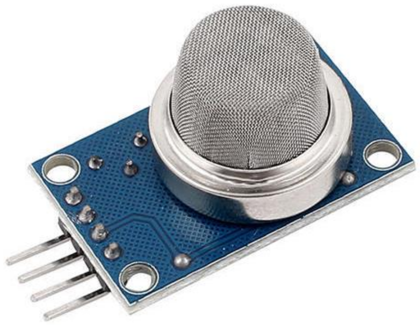


Figure 2.4 MQ2 Gas Sensor

- Operating Voltage is +5V
- Can be used to Measure or detect LPG, Alcohol, Propane, Hydrogen, CO and even methane
- Analog output voltage: 0V to 5V
- Digital Output Voltage: 0V or 5V (TTL Logic)
- Preheat duration 20 seconds
- Can be used as a Digital or analog sensor
- The Sensitivity of Digital pin can be varied using the potentiometer

Raspberry pi 3 B+:

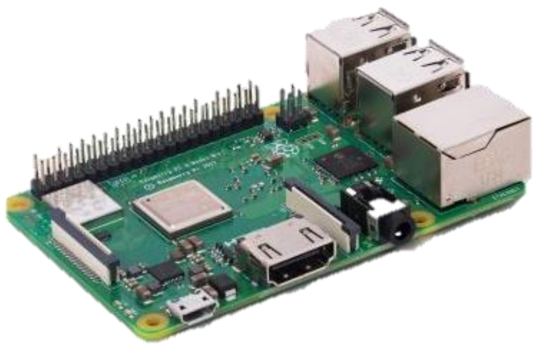


Figure 2.5 RaspberryPi 3 B+ Board

- Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- 1GB LPDDR2 SDRAM
- 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE
- Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps)
- Extended 40-pin GPIO header
- Full-size HDMI
- 4 USB 2.0 ports
- CSI camera port for connecting a Raspberry Pi camera
- DSI display port for connecting a Raspberry Pi touchscreen display
- 4-pole stereo output and composite video port
- Micro SD port for loading your operating system and storing data
- 5V/2.5A DC power input
- Power-over-Ethernet (PoE) support (requires separate PoE HAT)

8 Channel relay board:

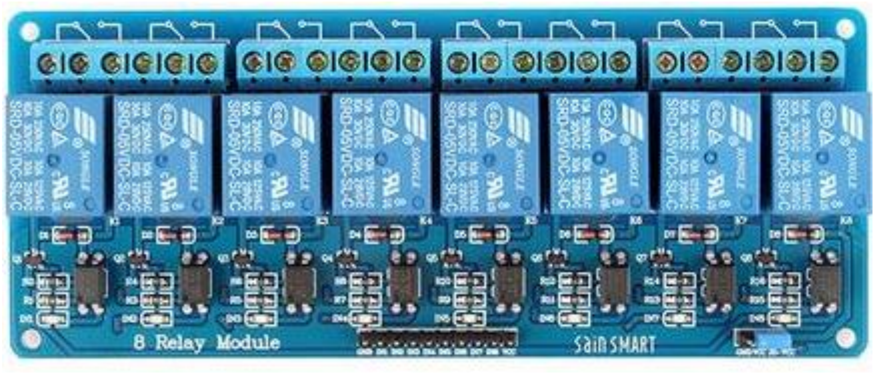


Figure 2.6 8 Channel 5V Relay Board

- Control Voltage: 5V DC.
- Max Control Capacity:10A at 250VAC or 10A at 30VDC.
- Low level triggered relay.
- Optocoupler Isolator.
- Relay switches work only when Vcc and JD-Vcc pins are shorted.

Logitech Webcam:

Figure 2.7 Logitech c270 Webcam

- Field of view :60 degrees
- Built-in mic.
- Image resolution: 720p
- Frame-rate: 30 fps
- Connection type: USB

12V DC Motor:

Figure 2.8 12V DC 10KG Torque 1000rpm Motor

- Operating voltage:12v.
- Stall Current: 4.5 A
- Maximum torque: 10Kg-Nm.
- Speed: 1000rpm

SERVO MOTOR SG90:

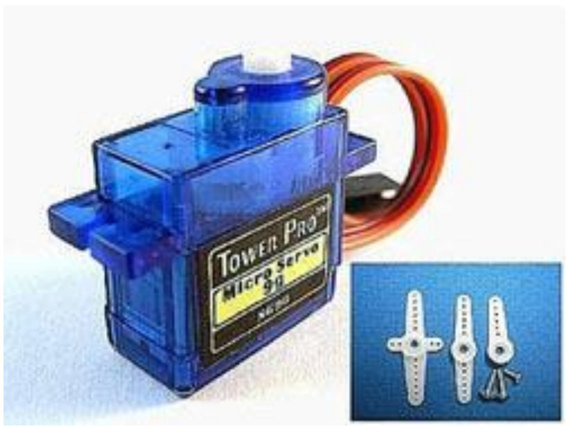


Figure 2.9 Servo motor SG90

- Analog modulation
- 1.8 Kg-cm torque
- Speed: 0.12 sec/60 degrees
- Weight: 9 grams
- Type: 3-pole
- Gear type: Plastic
- Pulse Width: 500-2400 micro seconds
- Connection type: JR

Chapter 3
IMPLEMENTATION

3.1 Initial configuration of RaspberryPi to work wirelessly:

Every software used in this project is open-sourced. The operating system used in RaspberryPi is “Raspbian”. Raspbian is an open-source software based on **Debian** optimized for Raspberry Pi hardware this operating system comes with basic utilities and programs to make our RaspberryPi working.

Debian: It is an open source software based on Linux kernel.

Raspbian comes with plenty of software for education, programming, and general use. It has Python, Scratch, Sonic Pi, Java and more.

SSH (Secure shell): It allows Raspberry Pi's **command line** to work wirelessly from another computer and any kind of operating system using SSH we can make Raspberry pi act as a remote device which we can access anywhere in the local network SSH provides strong encryption and integrity protection.

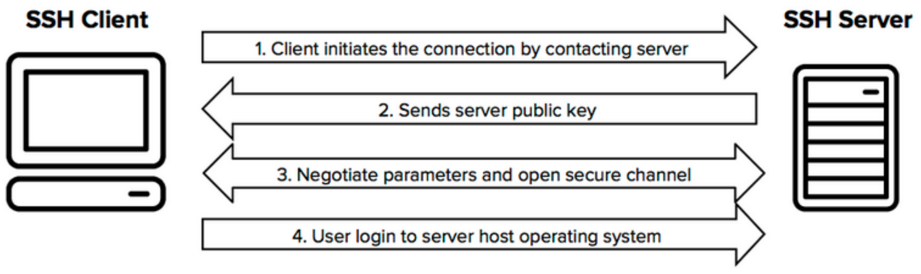
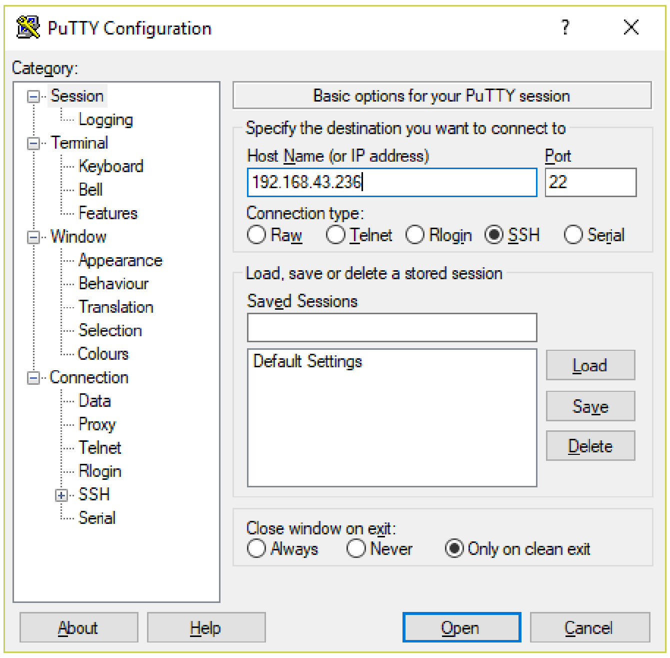

SSH Client
SSH Server
1. Client initiates the connection by contacting server
2. Sends server public key
3. Negotiate parameters and open secure channel
4. User login to server host operating system

Figure 3.1 SSH Shell Block Diagram

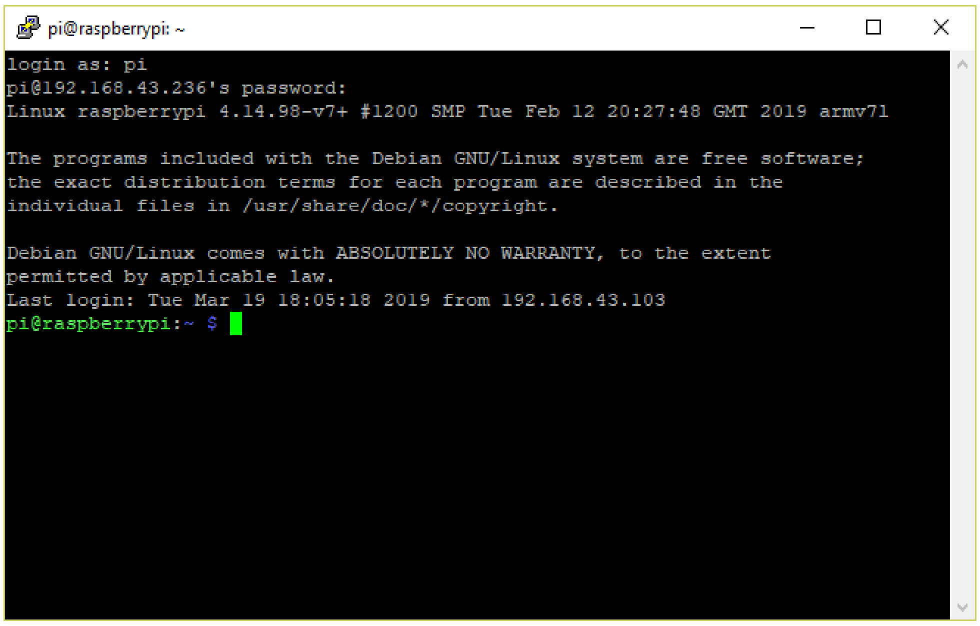
PuTTY: It is an open source terminal emulator used in our Raspberry Pi. PuTTY was originally written for windows and now we can use it in several UNIX based systems. It supports version 2 of SSH which we are using in our RaspberryPi. It comes with various encryption algorithms like RSA and DSA and it comes with several network protocols some of them are SCP, SSH, Telnet, raw socket connection and local serial port connections and it also supports **IPv6**.



Specify the destination you want to connect to
Host Name (or IP address): 192.168.43.236
Port: 22
Connection type: Raw Telnet Rlogin [✓ SSH] Serial
Saved Sessions: Default Settings
Close window on exit: Always Never [✓ Only on clean exit]

Figure 3.2 PuTTY Configuration Window

Once we type in the IP address of the RaspberryPi and select SSH option as the Connection type, we will be directed to the login page.



```
login as: pi
pi@192.168.43.236's password:
Linux raspberrypi 4.14.98-v7+ #1200 SMP Tue Feb 12 20:27:48 GMT 2019 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Mar 19 18:05:18 2019 from 192.168.43.103
pi@raspberrypi:~ $
```

Figure 3.3 RaspberryPi terminal

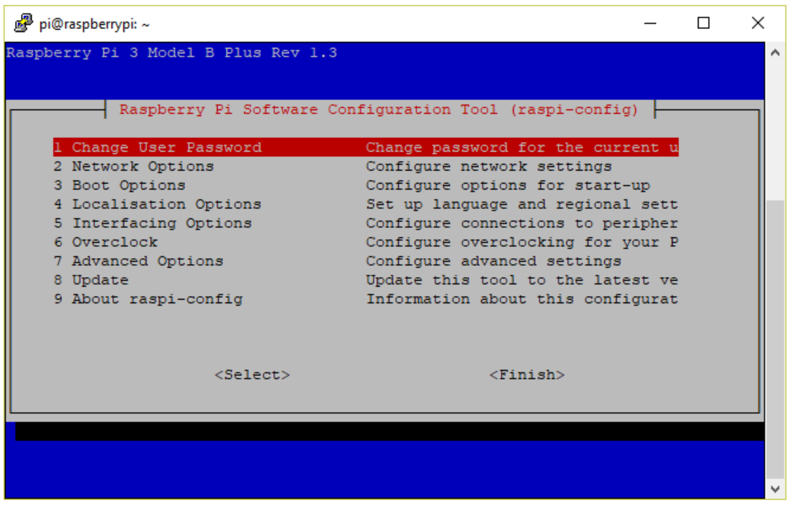
VNC server (Virtual Network Computing):

VNC is a graphical desktop sharing system which is used to remotely control the desktop interface of one computer (running VNC Server) from another computer or mobile device

(running VNC Viewer). VNC Viewer transfer the controls of keyboard and mouse to VNC Server and receives updates to the screen in return.
VNC Connect from Real VNC is included with Raspbian. It consists of both VNC Server, which allows you to control your Raspberry Pi remotely, which allows you to control desktop computers remotely from our Raspberry Pi.

Configuration process:

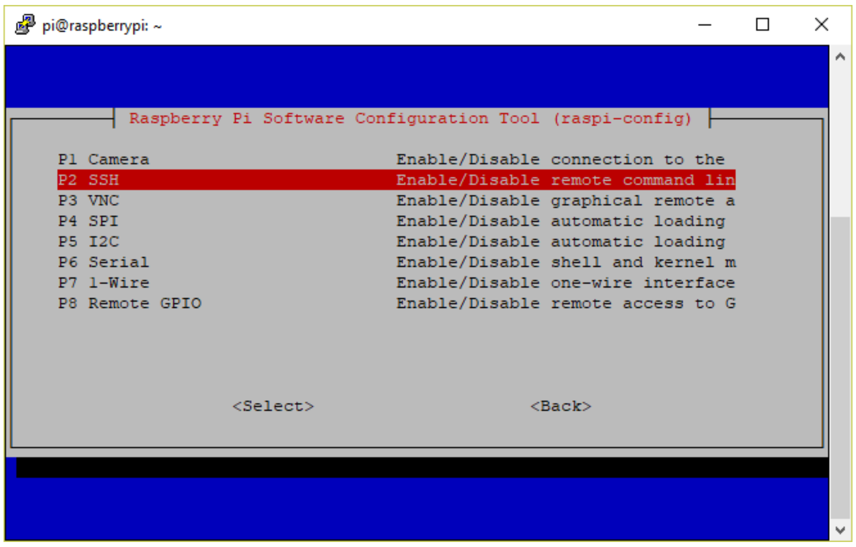
- Initially, for configuring raspberry pi we used a memory card with a storage of 32Gb. we download Raspbian stretch OS with kernel version 4.14 which will be available in Zip format and extract to an ISO file with size 4Gb or less
- The software used to write the Operating system to the memory card is called Win32DiskImager. With this software, we can read/write any kind of Operating system to the memory card/pen drive this process can be taken up to 10-15 minutes
- After successfully writing the software we insert the memory card into Raspberry Pi and power it up. initially, we have to create a blank file named SSH and copy it to the boot folder of RaspberryPi. The reason for creating this file is to work with PuTTY (SSH client).
- After performing all the above steps, we have to connect LAN cable from RASPBERRYPI to the host computer and type in the command called **"rpi.local"** in PuTTY hostname/IP address and it will connect to the Raspberry pi terminal.



```
pi@raspberrypi: ~
Raspberry Pi 3 Model B Plus Rev 1.3

        ┤ Raspberry Pi Software Configuration Tool (raspi-config) ├

     1 Change User Password          Change password for the current u
     2 Network Options               Configure network settings
     3 Boot Options                  Configure options for start-up
     4 Localisation Options          Set up language and regional sett
     5 Interfacing Options           Configure connections to peripher
     6 Overclock                     Configure overclocking for your P
     7 Advanced Options              Configure advanced settings
     8 Update                        Update this tool to the latest ve
     9 About raspi-config            Information about this configurat


                    <Select>                      <Finish>
```

Figure 3.4 RaspberryPi Configuration Window

- Next, we type in the command **"sudo raspi-config"** and it will open a pop-up in which we can configure various settings. We have to enable SSH in this window and enable all communication protocols available



```
pi@raspberrypi: ~

         Raspberry Pi Software Configuration Tool (raspi-config)

    P1 Camera                        Enable/Disable connection to the
    P2 SSH                           Enable/Disable remote command lin
    P3 VNC                           Enable/Disable graphical remote a
    P4 SPI                           Enable/Disable automatic loading
    P5 I2C                           Enable/Disable automatic loading
    P6 Serial                        Enable/Disable shell and kernel m
    P7 1-Wire                        Enable/Disable one-wire interface
    P8 Remote GPIO                   Enable/Disable remote access to G


                  <Select>                     <Back>
```

Figure 3.5 Interfacing Options

- Now, we type in command **“sudo apt-get install tightvncserver”** and **“sudo apt-get install xrdp”** to install the softwares required to share the desktop of RaspberryPi to any system in the same local network. These are very useful tools to cast the screen of RaspberryPi without using HDMI cable
- After that, we install VNC viewer for windows and type in command **“vncserver”** in the terminal of RaspberryPi then it will give a local IP address which we have to type in the VNC viewer in windows to access the desktop of RaspberryPi.
- Raspberry Pi model 3 B+ comes with wireless LAN **2.4 GHz and 5 GHz IEEE 802.11.b/g/n/**ac so we can connect to any wireless network we want and find the IP-address of pi using a command called **“ifconfig”** in the terminal.
- In Windows 10, there is a pre-installed app called **“Remote Desktop Connection”** which is used to access Desktop of RaspberryPi/any other Debian OS systems. We can now connect RaspberryPi and our computer to the same wireless network and type in the IP-address of RaspberryPi in the remote desktop app and we can access our RaspberryPi Desktop.

Connecting USB powered webcam to RaspberryPi

Using “fswebcam”:

This is the program used to generate images for a webcam. It captures a number of frames from any V4L or V4L2 compatible device, averages them to reduce noise and draws the details on it using the GD Graphics Library which also handles compressing the image to PNG, JPEG or WEBP.

Steps:

- First, we need to install fswebcam package using command **“sudo apt-get install fswebcam”** and all the packages required will be downloaded and we can take pictures using simple commands in the terminal
- We have to check whether the user has been added to the group correctly so we have to type in the command **“sudo usermod -a -G pi “**
- If we use a default pi we need not to execute above command or else, we have to add the user to the video group
- Command used to take picture from webcam is **“fswebcam sampleimg.jpg”**
- We can specify resolution of the image using command **“fswebcam -r 1280x720 sample.jpg”**
- To show the information of the image take from webcam we have to type in the command **“fswebcam -r 1280x720 --no-banner image3.jpg”**

Time lapse using CRONTAB:

There is a tool called **“crontab”** which is used to scheduling tasks like taking photos/videos/any-operation in any particular interval of time it will automatically run for like 1/ 5 minutes’ interval

Steps:

- Command for editing crontab is **“crontab -e”**
- Add a command “*** * * * * /home/pi/webcam.sh 2>&1”** to the crontab and save it which will take the photo for every 1minute
- We can add any type of command/program which will automatically run in the background

3.2 MOTION SOFTWARE:(live stream)

Motion is a software where we can take video input from several inputs and configure them as we like we can change the resolution, fps, dimensions.

It is highly configurable software which take input from various types of cameras like Pi camera, webcam, network cameras.

Using this software, we can livestream camera feed directly via local IP-address of raspberry pi.

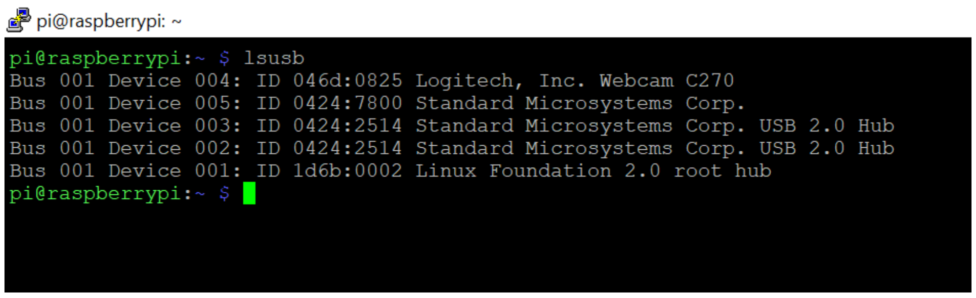
Steps for live stream:

- Install motion software using command

sudo apt-get install motion

- To check if the webcam is detected we have use command:

Lsusb



```
pi@raspberrypi: ~
pi@raspberrypi:~ $ lsusb
Bus 001 Device 004: ID 046d:0825 Logitech, Inc. Webcam C270
Bus 001 Device 005: ID 0424:7800 Standard Microsystems Corp.
Bus 001 Device 003: ID 0424:2514 Standard Microsystems Corp. USB 2.0 Hub
Bus 001 Device 002: ID 0424:2514 Standard Microsystems Corp. USB 2.0 Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
pi@raspberrypi:~ $
```

Figure 3.6 List of USB devices

- If no special drivers/software is required, then it should show the camera if we type in command called “ls /dev/video/*
- If we are using official camera modules the we have to type the following command to automatically display the camera:

sudo modprobe bcm2835-v412

- To look at the camera details we have to use command **“v4l2-ctl -V”.**
- we have to edit the motion configuration file by using command **“sudo nano /etc/motion/motion.conf”.**
- The above command will open-up a configuration file which we will be editing according to our needs.
- We need to change several variables in the configuration file we can find the variable names using “CTRL+W” and we can type in the variable we want it will navigate to the variable.
- Settings need to be changed:
- Start in daemon (background) mode and release terminal (default: off)

  daemon on

- Restrict stream connections to localhost only (default: on)

  stream localhost off
- Keep **"Stream_port"** to 8081.
- Change "**Stream_quality"** should be 100.
- Change **"Stream_localhost"** to OFF.
- Change **"webcontrol_localhost"** to OFF.
- Set **"quality"** to 100.
- Image width (pixels). Valid range: Camera dependent, default: 352

  width 640
- Image height (pixels). Valid range: Camera dependent, default: 288

  height 480 # Maximum number of frames to be captured per second.
- Valid range: 2-100. Default: 100 (almost no limit).

  framerate 100
- Set **"post_capture"** to **5.**
- We need to save the settings using "CTRL+O" and close the configuration file using "CTRL+X"
- Now, we need to type in command "**sudo nano /etc/default/motion" and** set **"start_motion_daemon"** to" **yes".**

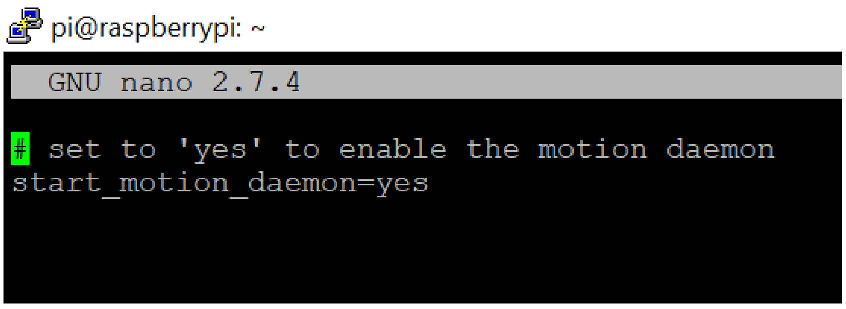

```
pi@raspberrypi: ~
  GNU nano 2.7.4

# set to 'yes' to enable the motion daemon
start_motion_daemon=yes
```

Figure 3.7 Editing the motion configuration file

- We need to restart the motion software using command:

sudo service motion restart

- To activate the webserver and to start motion we have to type in command:

sudo service motion start

- To view the livestream video, we have to go the browser of locally connected computer and type in the following URL **"http://<IP Address of RaspberryPi>:8081"**. In our case the URL is **"http://192.168.43.236:8081".**

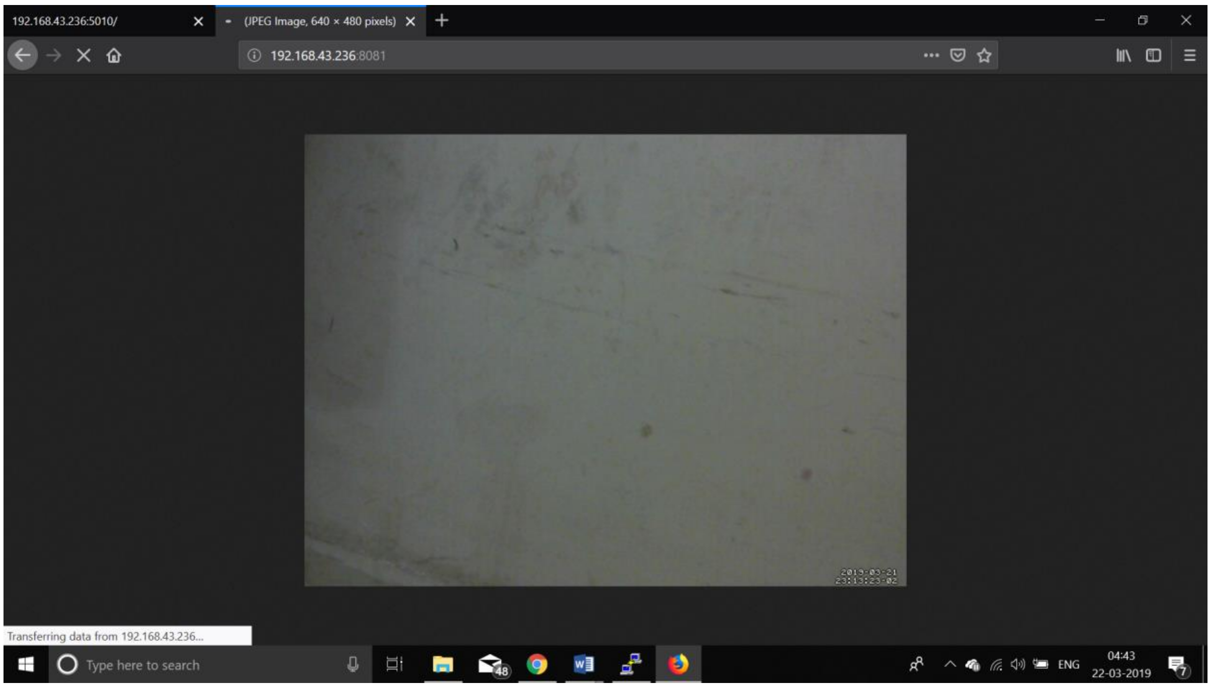


Figure 3.8 Livestream facilitated by Motion Software

- This livestream can now be used for surveillance or as an input for object detection.

3.3 Building a Python Web Server with Flask

To maneuver the rover, it would be better to design a combined environment of both vision and controls rather than running independent programs. So, we designed a web server using a software named "Flask", which has the capability to combine python programs with HTML and CSS and display on a webpage. The port created in the python program is accessible by any device connected to the same network. So, the rover can be controlled wirelessly as long as the controller and the rover are on the same network.

First, we install "Flask" software on the RaspberryPi using the command

```
sudo apt-get install flask
```

Next, we create a python file to use the flask software and create a webpage on a specified address. The HTML template to be used is also mentioned in this program. It will keep rendering the mentioned template (HTML webpage) until the program stops.

3.3.1 Creating a webpage

We can add a new webpage to our server by creating a new route. In a web application, a route is a certain path into the website, determined by the URL the user types into their web browser's address bar. It is up to the user which routes are enabled and what each of them does.

This route is made up of three parts:

1. **@app.route('/')**: This determines the entry point; the / means the root of the website, so http://192.168.43.236:5010.
2. **def_index()**: This is the name given to the route and is called **index**, because it's the index (or home page) of the website.
3. **return ''**: This is the content of the web page, which is returned when the user goes to this URL.

Now, we define a series of General Purpose Input Output (GPIO) pins to control the relay to which the motors are connected, and the servo motor to which the camera is connected.

```
pinList = [2,11,4,17,27,22,10,9,23,24,20,21]
servoPIN=12
```

Next, we initialize the servo motor to align to the center position (90 degrees) by sending an initial pulse of 7.5% duty cycle.

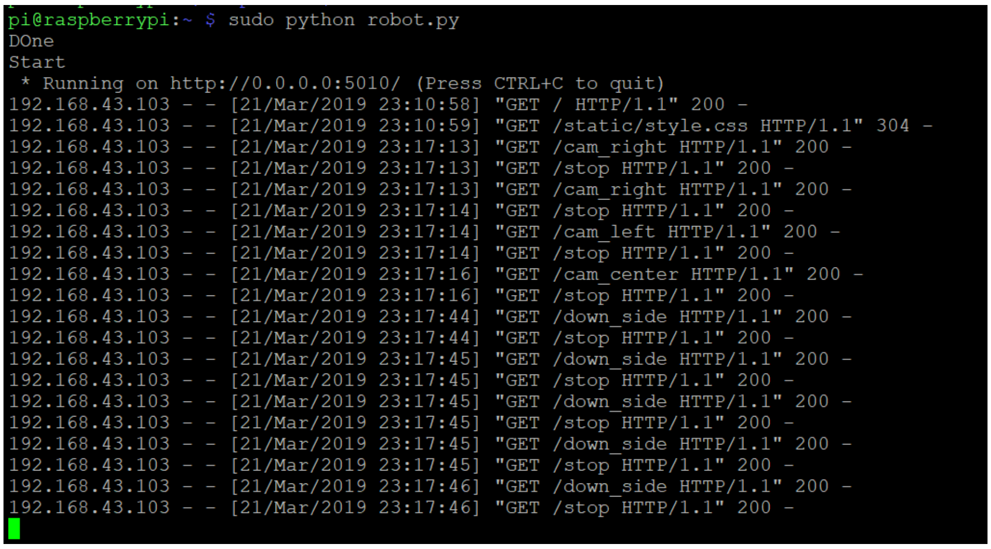
```
p=GPIO.PWM(servoPIN, 50)
p.start(7.5)
```

All the GPIO pins used for motor control are declared as output and are set to "HIGH" to reset the relay board. This will prevent the rover from uninitiated movements once the program starts running.

```
GPIO.setup(servoPIN, GPIO.OUT)
for i in pinList:
        GPIO.setup(i, GPIO.OUT)
        GPIO.output(i, GPIO.HIGH)
```

For every control, namely, forward, backward, left, right and camera movements, we create an individual route to the route to the website. These routes will contain the functions that can be called upon the request of the user on the webpage. Since we are using two relays for each motor, we program all the 8 GPIO pins for each movement.

app.run(host='0.0.0.0',port=5010) means that the webpage is accessible by any user on the same network. Every time a user tries to control the rover, it gets logged as long as the server is active.



```
pi@raspberrypi:~ $ sudo python robot.py
DOne
Start
 * Running on http://0.0.0.0:5010/ (Press CTRL+C to quit)
192.168.43.103 - - [21/Mar/2019 23:10:58] "GET / HTTP/1.1" 200 -
192.168.43.103 - - [21/Mar/2019 23:10:59] "GET /static/style.css HTTP/1.1" 304 -
192.168.43.103 - - [21/Mar/2019 23:17:13] "GET /cam_right HTTP/1.1" 200 -
192.168.43.103 - - [21/Mar/2019 23:17:13] "GET /stop HTTP/1.1" 200 -
192.168.43.103 - - [21/Mar/2019 23:17:13] "GET /cam_right HTTP/1.1" 200 -
192.168.43.103 - - [21/Mar/2019 23:17:14] "GET /stop HTTP/1.1" 200 -
192.168.43.103 - - [21/Mar/2019 23:17:14] "GET /cam_left HTTP/1.1" 200 -
192.168.43.103 - - [21/Mar/2019 23:17:14] "GET /stop HTTP/1.1" 200 -
192.168.43.103 - - [21/Mar/2019 23:17:16] "GET /cam_center HTTP/1.1" 200 -
192.168.43.103 - - [21/Mar/2019 23:17:16] "GET /stop HTTP/1.1" 200 -
192.168.43.103 - - [21/Mar/2019 23:17:44] "GET /down_side HTTP/1.1" 200 -
192.168.43.103 - - [21/Mar/2019 23:17:44] "GET /stop HTTP/1.1" 200 -
192.168.43.103 - - [21/Mar/2019 23:17:45] "GET /down_side HTTP/1.1" 200 -
192.168.43.103 - - [21/Mar/2019 23:17:45] "GET /stop HTTP/1.1" 200 -
192.168.43.103 - - [21/Mar/2019 23:17:45] "GET /down_side HTTP/1.1" 200 -
192.168.43.103 - - [21/Mar/2019 23:17:45] "GET /stop HTTP/1.1" 200 -
192.168.43.103 - - [21/Mar/2019 23:17:45] "GET /down_side HTTP/1.1" 200 -
192.168.43.103 - - [21/Mar/2019 23:17:45] "GET /stop HTTP/1.1" 200 -
192.168.43.103 - - [21/Mar/2019 23:17:46] "GET /down_side HTTP/1.1" 200 -
192.168.43.103 - - [21/Mar/2019 23:17:46] "GET /stop HTTP/1.1" 200 -
```

Figure 3.9 Log of Server requests

3.3.2 HTML Webpage:

Now, all the controls have been set in the python program and they can now be appended to the webpage. The webpage is designed in HTML and it is stored in a different directory named "templates". For this, simply create a directory named "templates". By doing this, we can use separate files with placeholders for spots where we want to insert dynamic data. In order to compensate the lack of speed control, the script of the webpage is created such that as long as the user presses and holds the button (hyperlink) on the webpage, the relays are accordingly controlled. Once the user retracts from the button, the relays are reset to HIGH. This also helps with instant stopping of the vehicle since the motors are forced to stop. In the HTML page that we created, we have included the livestream taken from the URL: port specified in the motion software "http:192.168.43.236:8081" and controls on the right side.

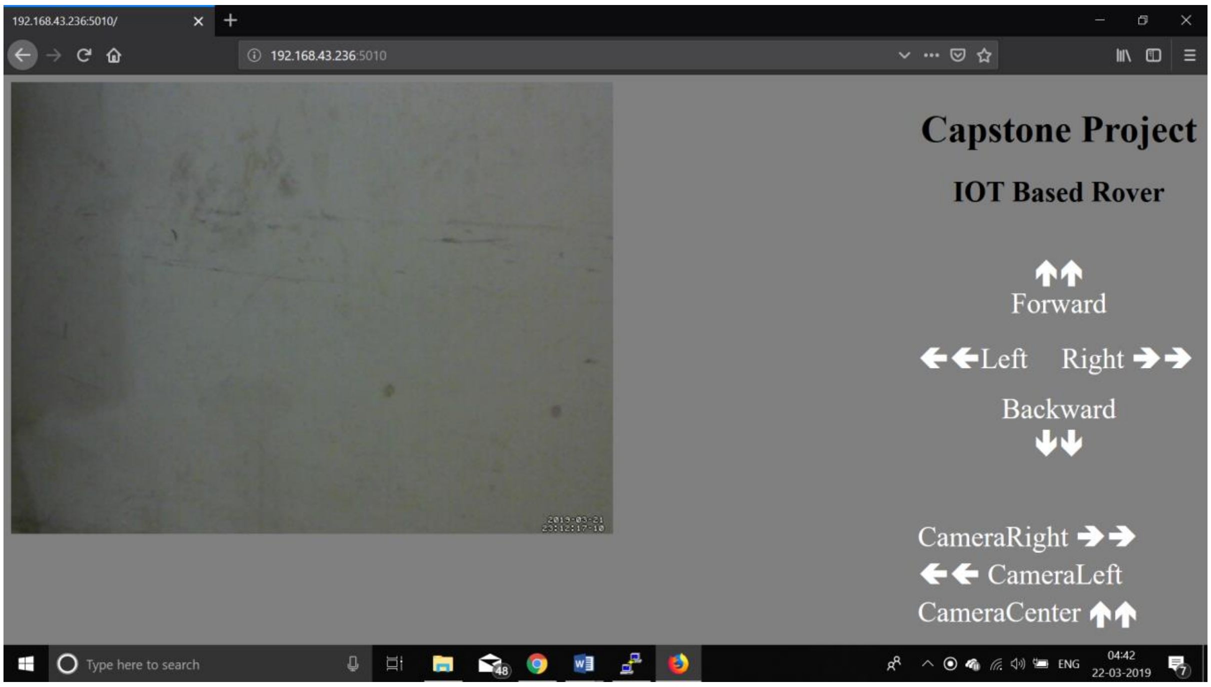


Figure 3.10 HTML Webpage

3.3.3 CSS MODEL:

The formatting of the webpage can be done by using the Cascading Style Sheet. It consists of commands pertaining to how the HTML webpage should look like. To integrate with the HTML webpage we created, we create the CSS file in a folder named "static" and point it in the HTML file itself. If the address and name of the CSS file is correctly pointed, it will render the CSS file. If there is any error in the code, no error would popup. Simply, the webpage would not look as expected. In the CSS file that we have created, we have given the background to be grey, the text font of the headings to be black and that of the controls to be white.

Now that we have created the python, HTML and CSS files, we can host the server through the RaspberryPi by using the command:

```
sudo python robot.html
```

The server will run as long as it has a known local network. The server can be terminated by simply pressing ctrl+c. The only disadvantage is that, as long as the server is running, no other program can be run.

3.4 Movement control using relays:

To provide maximum current to all the motors, it is better to use relays rather than a motor driver IC. To control the motors, a L298N IC is generally used. It's maximum current output to any motor is 2A. But, the 10Kg-cm torque motors used in this project can draw upto a maximum current of 4.5A. So, in tough terrains where torque is the main factor, it can instantly burn out the IC. In order to prevent this, relays have been used.

A relay has 3 ports:

1. Normally Open (NO) port.
2. Normally Closed (NC) port.
3. Common port.

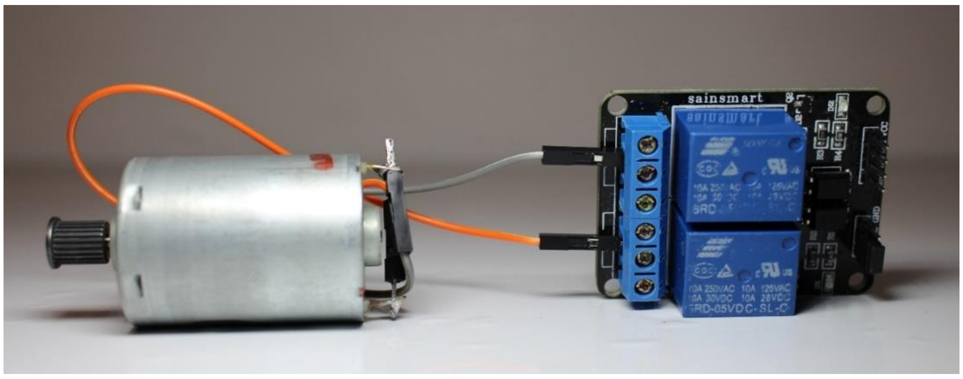


Figure 3.11 Controlling a motor with a relay

For controlling each motor, two relays are to be used. The two pins of the motor are connected to two common ports. The NC pin of each relay is connected to the negative wire of the power supply and the NO pin of each relay is connected to the positive wire of the power supply. For movement in one direction, one relay is to be ON (LOW), and the other is to be in OFF (HIGH) position. By simply reversing the relays, the motor direction can be changed. To control all the 4 motors, an 8-channel relay board has been used.

3.5 SERVO MOTOR CONTROL:

To move the camera, a 180-degree servo motor (SG90) has been used. A servo motor can be controlled by sending an electrical pulse of variable width, or pulse width modulation (PWM) through the control wire. The motor’s neutral position is defined as the position where the servo has the same amount of potential rotation in both the clockwise and anti-clockwise direction. The servo motor can rotate only upto 90-degrees in either direction, serving a total of 180-degree movement. The PWM sent to the motor determines the position of the shaft, and based on the duration of the pulse, the rotor will turn to the desired position. The servo motor expects to see a pulse every 20 milliseconds (ms) and the length of the pulse will determine how far the motor turns. For example, a 1.5ms pulse will make the motor turn to the 90-degree position. Shorter than 1.5ms moves it to 0 degrees, and any longer than 1.5ms will turn the servo to 180 degrees.

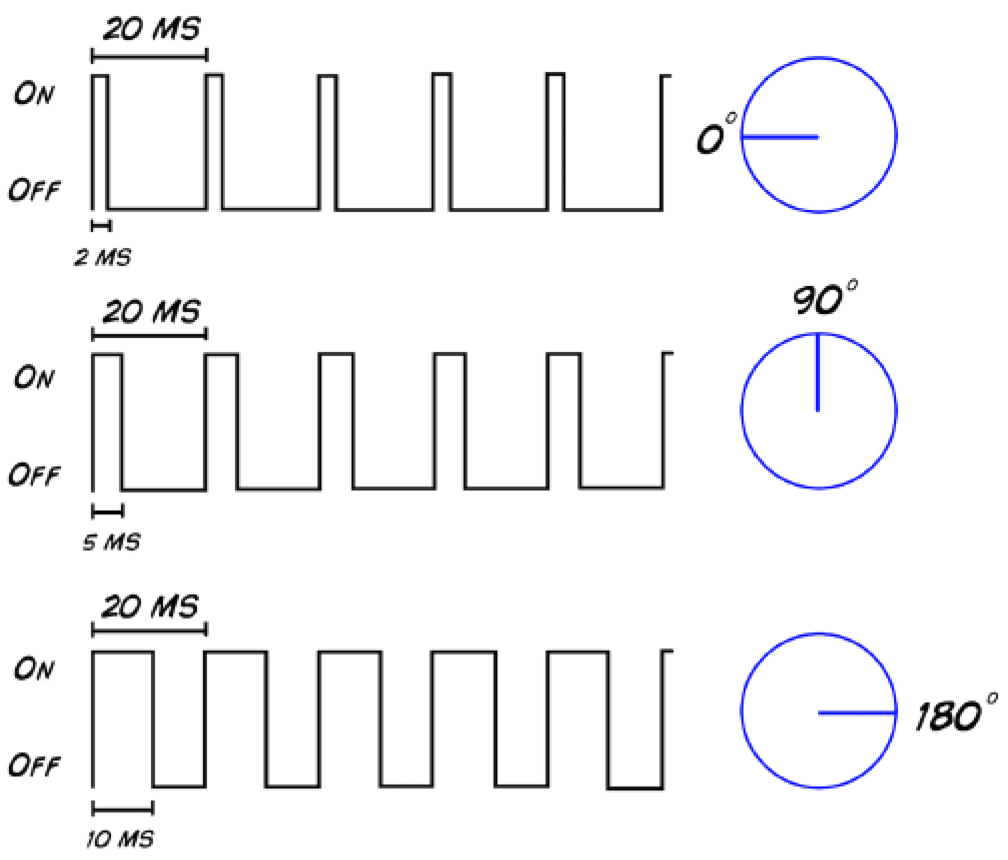


Figure 3.12 Orientations of servo motor w.r.t different PWM signals

To control the servo motor from the Raspberry Pi we are going to use the PWM module in RaspberryPi.GPIO. GPIO 12 has been used as the PWM pin using the command:

p=GPIO.PWM(12, 50)

So, GPIO 12 has been initialized with a frequency of 50Hz. That frequency was selected because the servo motor expect a pulse every 20ms (period), that means 50 pulses per second or Hertz. The duty cycle describes the proportion of *on* time to the regular interval or *period* of time. If we want a pulse with an specific length we can calculate the duty cycle as follows:

dc=length/period

Since the servo uses 20ms cycles, we can calculate the duty cycle of the 3 turns of the servo motor:

$$\mathrm{dc} = \frac{0.5}{20} \times 100 = 2.5\%$$

$$\mathrm{dc} = \frac{1.5}{20} \times 100 = 7.5\%$$

$$\mathrm{dc} = \frac{2.5}{20} \times 100 = 12.5\%$$

Figure 3.13 Duty cycles required for programming the servo motor

To change the duty cycle we can use:

p.ChangeDutyCycle(_)

To stop the pulse emission:

p.stop()

3.6 OBJECT DETECTION:

There are three methods of deep-learning-based object detection:

- Faster R-CNNs (Girshick et al., 2015)
- You Only Look Once (YOLO) (Redmon and Farhadi, 2015)
- Single Shot Detectors (SSDs) (Liu et al., 2015)

Faster R-CNNs are most commonly used method for deep learning-based object detection. But this technique is difficult to comprehend, especially for beginners in deep learning, and hard to implement and train. Also, the algorithm when implemented in real-time tends to show the output in the order around 7fps. YOLO algorithm is much faster, capable of processing 40-90 FPS on a Titan X GPU. The super-fast variant of YOLO can even get up to 155 FPS. The drawback of YOLO is that it sacrifices the much-needed accuracy.

SSDs, originally developed by Google, are a balance between the two. The algorithm is more straightforward than Faster R-CNNs. SSDs also tend to be more accurate than YOLO.

3.6.1 MobileNets: Efficient (deep) neural networks

When building object detection networks we normally use an existing network architecture, such as VGG or ResNet, and then use it inside the object detection pipeline. The problem is that these network architectures can be very large in the order of 200-500MB.

Network architectures such as these are unsuitable for resource constrained devices due to their sheer size and resulting number of computations. This is the reason why we are not implementing this algorithm in the RaspberryPi module. Instead, we will be running this model in a host system on the live feed given by the Pi. In this way, we can reduce the workload on the Pi.

MobileNets differ from traditional CNNs through the usage of depthwise separable convolution.
The general idea behind depthwise separable convolution is to split convolution into two stages:

1. A *3×3* depthwise convolution.
2. Followed by a *1×1* pointwise convolution.

This reduces the number of parameters in a network.

Combining MobileNets and Single Shot Detectors for fast, efficient deep-learning based object detection:

If we combine both the MobileNet architecture and the Single Shot Detector (SSD) framework, we arrive at a fast, efficient deep learning-based method to object detection. The model we'll be using in our project post is a Caffe version of the original TensorFlow implementation by Howard et al. and was trained by chuanqi305. The MobileNet SSD was trained on the COCO dataset (Common Objects in Context). We can detect 20 objects in images, including airplanes, bicycles, birds, boats, bottles, buses, cars, cats, chairs, cows, dining tables, dogs, horses, motorbikes, people, potted plants, sheep, sofas, trains, *and* tv monitors.

ALGORITHM:

- First, we parse the command line arguments:
  - ✓ --prototxt : The path to the Caffe prototxt file.
  - ✓ --model : The path to the pre-trained model.
  - ✓ --confidence : The minimum probability threshold to filter weak detections. The default is 20%.
- A class list and a colour set is initialized.
- Next, we load the model and set up our video stream. First we start the Video Stream, then we wait for the camera to warm up, and finally we start the frames per second counter.
- Now, each and every frame is looped over. First a frame from the stream is read, and resized. This frame is now converted into a blob using the dnn module.
- This blob is set as input to the neural network and this produces the detections.
- Next, the network looks at the confidence values and determines if it should draw a box and label the surrounding object or not.
- Now, the detections are looped over, keeping in mind that multiple objects can be detected in a single image. Also, it applies a check to the confidence (i.e., probability) associated with each detection.
- If the confidence is above minimum threshold, the class label index is extracted and the bounding box coordinates around the detected object are computed.
- The (x, y) coordinates of the box used for drawing the box and displaying text is extracted. Finally, the fps counter is updated.

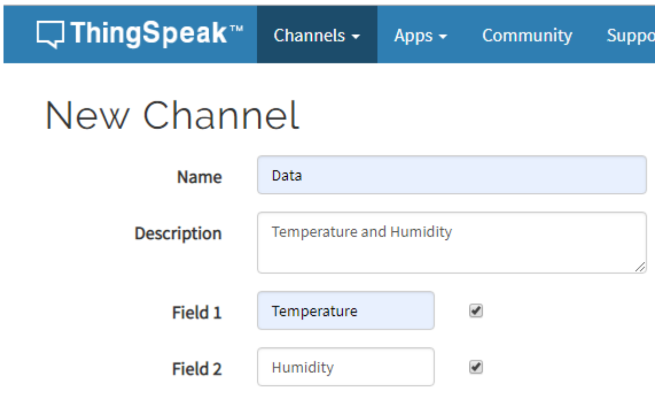
3.7 ThingSpeak:

ThingSpeak is an Open source IoT application to store and retrieve data from the internet or via local area network. ThingSpeak collaborated with Matlab to visualize and analyse uploaded data. To use this platform one must create an account in the site or use Mathworks account.

To send any data into the cloud a controller connected to the internet is required. Since we are using Raspberry pi which has a built in wifi module, its easy if there is an internet connection.

Open *thingspeak* site and login.

Step 1: Create a channel and enter the name of the data you wish to put up online.



New Channel

Name	Data
Description	Temperature and Humidity
Field 1	Temperature ✓
Field 2	Humidity ✓

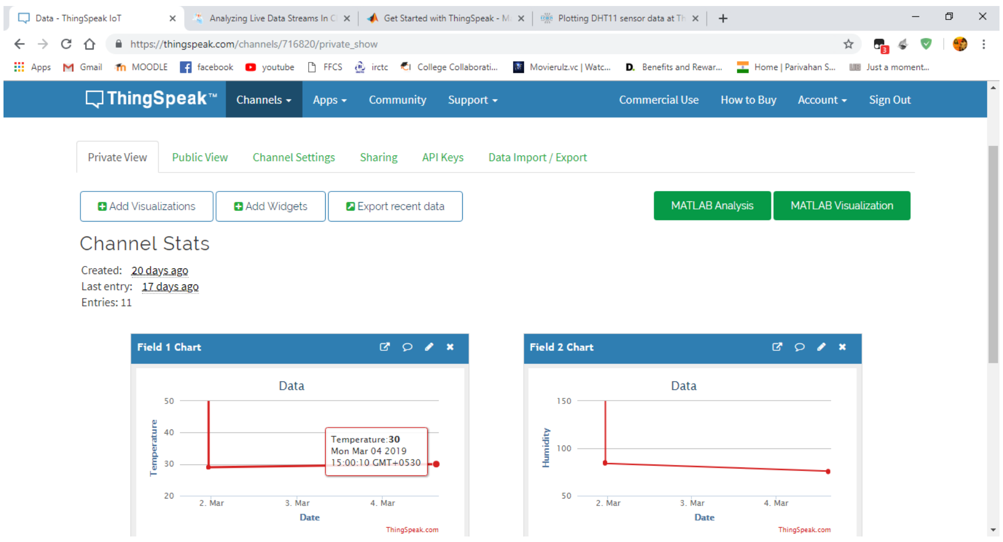
Figure 3.14 Creating new channel in ThingSpeak

Step 2: API keys

These API keys enable you to write data to a channel or read data from a private channel. In the Raspberry Pi make the connections with the sensor required, in this case we are using DHT11 temperature and humidity sensor.

https://api.thingspeak.com/update?api_key=YOUR_CHANNEL_API_KEY&field1=99

since we are sending the data into ThingSpeak we've to use the write Key



Channel Stats

Created: 20 days ago
Last entry: 17 days ago
Entries: 11

Temperature: 30
Mon Mar 04 2019
15:00:10 GMT+0530

Figure 3.15 Data in ThingSpeak platform

Chapter 4

RESULTS AND DISCUSSIONS

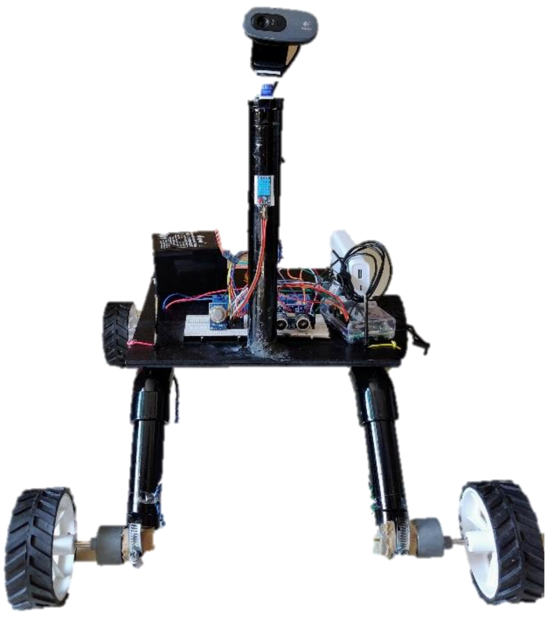


Figure 4.1 Final model (front view)

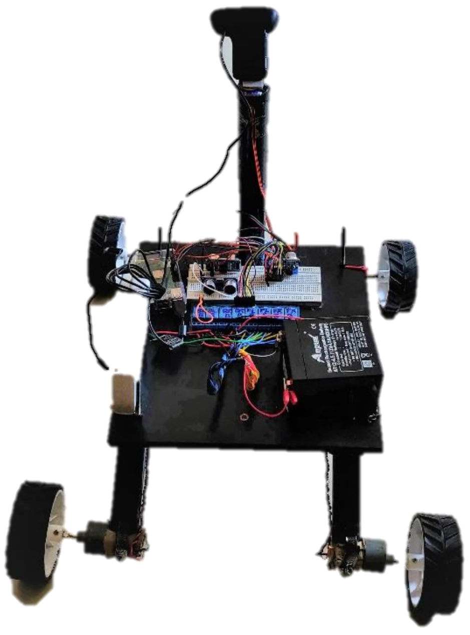
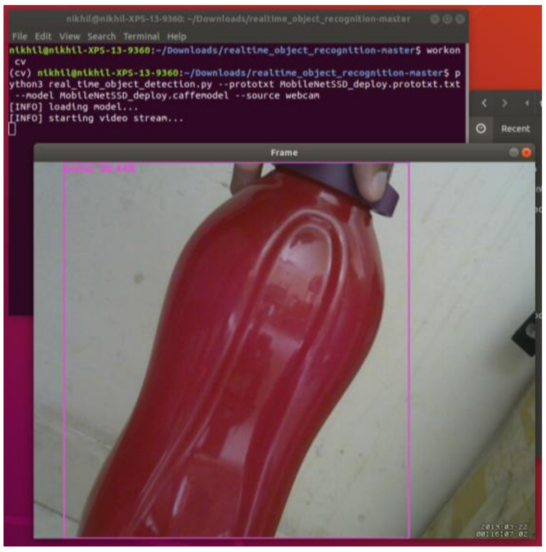
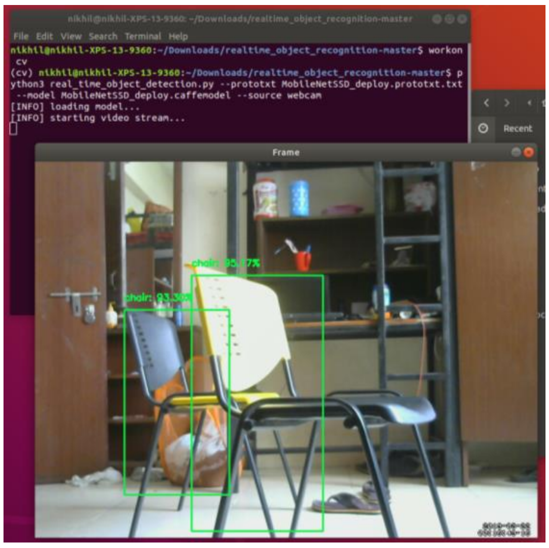


Figure 4.2 Final model (rear view)


nikhil@nikhil-XPS-13-9360: ~/Downloads/realtime_object_recognition-master
File Edit View Search Terminal Help
Recent
Frame

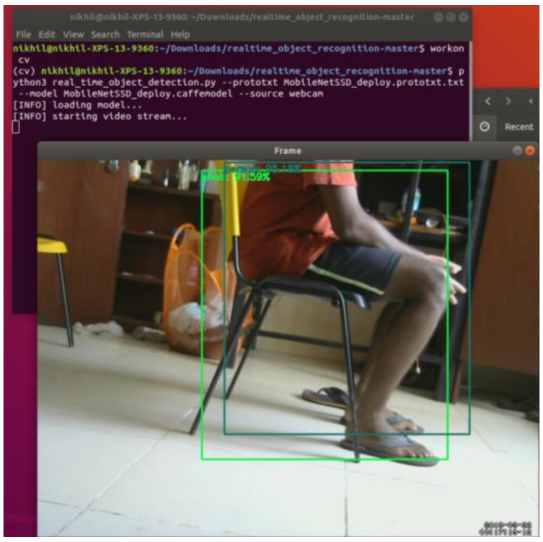
```
nikhil@nikhil-XPS-13-9360:~/Downloads/realtime_object_recognition-master$ workon
 cv
(cv) nikhil@nikhil-XPS-13-9360:~/Downloads/realtime_object_recognition-master$ p
ython3 real_time_object_detection.py --prototxt MobileNetSSD_deploy.prototxt.txt
 --model MobileNetSSD_deploy.caffemodel --source webcam
[INFO] loading model...
[INFO] starting video stream...
```


nikhil@nikhil-XPS-13-9360: ~/Downloads/realtime_object_recognition-master
File Edit View Search Terminal Help
Recent
Frame
chair: 95.17%
chair: 93.36%

```
nikhil@nikhil-XPS-13-9360:~/Downloads/realtime_object_recognition-master$ workon
 cv
(cv) nikhil@nikhil-XPS-13-9360:~/Downloads/realtime_object_recognition-master$ p
ython3 real_time_object_detection.py --prototxt MobileNetSSD_deploy.prototxt.txt
 --model MobileNetSSD_deploy.caffemodel --source webcam
[INFO] loading model...
[INFO] starting video stream...
```

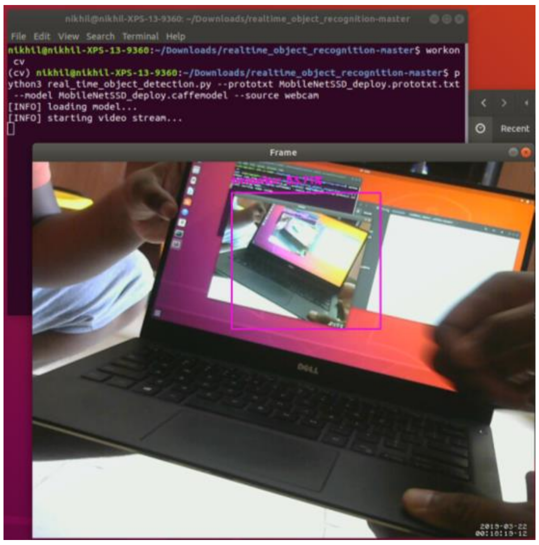
Figure 4.3 Recognizing a bottle

Figure 4.4 Recognizing multiple objects (chairs)


nikhil@nikhil-XPS-13-9360: ~/Downloads/realtime_object_recognition-master
File Edit View Search Terminal Help
Recent
Frame

```
nikhil@nikhil-XPS-13-9360:~/Downloads/realtime_object_recognition-master$ workon
 cv
(cv) nikhil@nikhil-XPS-13-9360:~/Downloads/realtime_object_recognition-master$ p
ython3 real_time_object_detection.py --prototxt MobileNetSSD_deploy.prototxt.txt
 --model MobileNetSSD_deploy.caffemodel --source webcam
[INFO] loading model...
[INFO] starting video stream...
```

Figure 4.5 A person and a chair


nikhil@nikhil-XPS-13-9360: ~/Downloads/realtime_object_recognition-master
File Edit View Search Terminal Help
Recent
Frame

```
nikhil@nikhil-XPS-13-9360:~/Downloads/realtime_object_recognition-master$ workon
 cv
(cv) nikhil@nikhil-XPS-13-9360:~/Downloads/realtime_object_recognition-master$ p
ython3 real_time_object_detection.py --prototxt MobileNetSSD_deploy.prototxt.txt
 --model MobileNetSSD_deploy.caffemodel --source webcam
[INFO] loading model...
[INFO] starting video stream...
```

Figure 4.6 Recognizing a screen

Chapter 5

COST ANALYSIS

5.1 List of components and their cost

The costs of the various components used in this project are given below in Table 5.1.

Table 5.1 List of components and their costs

COMPONENT	COST
Raspberry Pi 3 B+	Rs 2999
Logitech Webcam	Rs 799
12v 4.5A Battery	Rs 700
12V DC Motors 10KG-cm Torque, 1000rpm (4 nos.)	Rs 2000
Servo Motor Sg90 2.5Kg-cm Torque	Rs 195
Rover Wheels (4 nos.)	Rs 1400
MQ-2 Gas Sensor	Rs 140
Dht11 Sensor	Rs 115
Ultrasonic Sensors (2 nos.)	Rs 200
8 Channel Relay Board	Rs 350
Jumper Wire Set	Rs 80
UPVC Pipe (2m)	Rs 600
1x1 feet Wooden Plank (2 Pcs.)	Rs 130
1” hose clamp (4 nos.)	Rs 140
TOTAL	Rs 9848

CHAPTER 6

CONCLUSION AND FUTURE WORK

This chapter concludes the thesis on IoT based Rover for exploration and suggests additional functionalities for future implantation.

6.1 Conclusion

In this project, a rover to explore in areas with rough terrain and measure certain atmospheric parameters such as temperature, moisture and gas was demonstrated. The parameters were measured over vast areas (50m) owing to the long range of Wifi connectivity. This rover could be used for a wide variety of applications such as surveillance, home purposes, visual assistance in mines. The essential environmental parameters such as temperature, moisture and presence of gas were uploaded to ThingSpeak platform and analytics were displayed. Also, the basic movement controls of the rover were integrated into a website locally hosted by the RaspberryPi.

6.2 Future Work

With the help of the rover, we can extract any visual or environmental information and use it for more IoT purposes. Any detection algorithm can be easily integrated and run on a host system by simply utilizing the vision from the rover.

APPENDIX

CODE FOR PYTHON BASED WEBSERVER:

```
from flask import Flask
from flask import render_template, request
import RaspberryPi.GPIO as GPIO
from time import sleep
GPIO.setmode(GPIO.BCM)

app = Flask(__name__)

pinList = [2,11,4,17,27,22,10,9]
servoPIN=12
p=GPIO.PWM(servoPIN, 50)
p.start(7.5)

temp1=1

GPIO.setup(servoPIN, GPIO.OUT)
for i in pinList:
        GPIO.setup(i, GPIO.OUT)
        GPIO.output(i, GPIO.HIGH)

print "DOne"

a=1
@app.route("/")
def index():
    return render_template('robot.html')

@app.route('/left_side')
def left_side():
    data1="LEFT"
    GPIO.output(2, GPIO.LOW)
    GPIO.output(11, GPIO.HIGH)
    GPIO.output(17, GPIO.LOW)
    GPIO.output(4, GPIO.HIGH)
    GPIO.output(27, GPIO.LOW)
    GPIO.output(22, GPIO.HIGH)
    GPIO.output(9, GPIO.LOW)
    GPIO.output(10, GPIO.HIGH)

    x='z'

    return 'true'

@app.route('/right_side')
def right_side():
   data1="RIGHT"
   GPIO.output(11, GPIO.LOW)
   GPIO.output(2, GPIO.HIGH)
   GPIO.output(4, GPIO.LOW)
   GPIO.output(17, GPIO.HIGH)
   GPIO.output(22, GPIO.LOW)
   GPIO.output(27, GPIO.HIGH)
   GPIO.output(10, GPIO.LOW)
   GPIO.output(9, GPIO.HIGH)
   x='z'
   return 'true'

@app.route('/up_side')
def up_side():
   data1="FORWARD"
   GPIO.output(11, GPIO.LOW)
   GPIO.output(2, GPIO.HIGH)
   GPIO.output(17, GPIO.LOW)
   GPIO.output(4, GPIO.HIGH)
```

```
  GPIO.output(22, GPIO.LOW)
  GPIO.output(27, GPIO.HIGH)
  GPIO.output(9, GPIO.LOW)
  GPIO.output(10, GPIO.HIGH)
  x='z'
  return 'true'

@app.route('/down_side')
def down_side():
  data1="BACK"
  GPIO.output(2, GPIO.LOW)
  GPIO.output(11, GPIO.HIGH)
  GPIO.output(4, GPIO.LOW)
  GPIO.output(17, GPIO.HIGH)
  GPIO.output(27, GPIO.LOW)
  GPIO.output(22, GPIO.HIGH)
  GPIO.output(10, GPIO.LOW)
  GPIO.output(9, GPIO.HIGH)
  x='z'
  return 'true'



@app.route('cam_left')
def cam_left():
        data1="CAMLEFT"
        p.ChangeDutyCycle(12.5)
        x='z'
        return 'true



@app.route('cam_right')
def cam_right():
        data1="CAMRIGHT"
        p.ChangeDutyCycle(2.5)
        x='z'
        return 'true


@app.route('cam_center')
def cam_center():
        data1="CAMCENTER"
        p.ChangeDutyCycle(7.5)
        x='z'
        return 'true

@app.route('/stop')
def stop():
  data1="STOP"
  GPIO.output(2, GPIO.HIGH)
  GPIO.output(11, GPIO.HIGH)
  GPIO.output(4, GPIO.HIGH)
  GPIO.output(17, GPIO.HIGH)
  GPIO.output(27, GPIO.HIGH)
  GPIO.output(22, GPIO.HIGH)
  GPIO.output(10, GPIO.HIGH)
  GPIO.output(9, GPIO.HIGH)
  x='z'
  return  'true'

if __name__ == "__main__":
 print "Start"
 app.run(host='0.0.0.0',port=5010)
```

HTML WEBPAGE CODE:

```
<html>
<head>
```

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
</head>
<body>
<link rel="stylesheet" href='/static/style.css' />
<img src="http://192.168.43.236:8081" />
<div style="float:right">
</div>
<div style=" height:400px; width:300px; float:right;">
<center>
<h1>
<span style="color:#5C5C5C;">Capstone</span><span style="color:#139442">
Project</span>
</h1>
<h2>IOT Based Rover</h2>
<br>
<br>
<a href="#" id="up" style="font-size:30px;text-decoration:none;">
&#x1F881;&#x1F881;<br>Forward</a>
<br>
<br>
</center>
<a href="#" id="left" style="font-size:30px;text-decoration:none;">
&#x1F880;&#x1F880;Left</a>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
<a href="#" id="right" style="font-size:30px; text-decoration:none;"> Right
&#x1F882;&#x1F882;</a>
<br>
<br>
<center>
<a href="#" id="down" style="font-size:30px;text-decoration:none;"> Backward<br>
&#x1F883;&#x1F883;</a></center>
<br>
<br>
<br>
```

```
<a href="#" id="CAMR" style="font-size:30px;text-decoration:none;">CameraRight
&#x1F882;&#x1F882;</a>
<br>
<a href="#" id="CAML" style="font-size:30px;text-
decoration:none;">&#x1F880;&#x1F880; CameraLeft</a><br>
<a href="#" id="CAMC" style="font-size:30px;text-decoration:none;">CameraCenter
&#x1F881;&#x1F881;</a><br>
</div>
<script>
$( document ).ready(function(){
   $("#down").on("mousedown", function() {
    $.get('/down_side');
    }).on('mouseup', function() {
    $.get('/stop');
   });

   $("#up").on("mousedown", function() {
    $.get('/up_side');
    }).on('mouseup', function() {
    $.get('/stop');
   });

   $("#left").on("mousedown", function() {
    $.get('/left_side');
    }).on('mouseup', function() {
    $.get('/stop');
   });

   $("#right").on("mousedown", function() {
    $.get('/right_side');
    }).on('mouseup', function() {
    $.get('/stop');
   });
   $("#CAMR").on("mousedown", function() {
    $.get('/cam_right');
    }).on('mouseup', function() {
    $.get('/stop');
   });



   $("#CAML").on("mousedown", function() {
    $.get('/cam_left');
    }).on('mouseup', function() {
    $.get('/stop');
   });

   $("#CAMC").on("mousedown", function() {
    $.get('/cam_center');
    }).on('mouseup', function() {
    $.get('/stop');
   });



});
</script></body>
</html>
```

CSS CODE:

```
*{
   background: grey;
   color: white;
   }
h1{
   color: black;
   font-size: 80px;
   }
h2{
```

```
color: black;
font-size: 50px;
}
```

OBJECT DETECETION CODE:

```
#Trained model can be downloaded from GitHub.
# USAGE
# python real_time_object_detection.py --prototxt MobileNetSSD_deploy.prototxt.txt --
model MobileNetSSD_deploy.caffemodel --source webcam

# import the necessary packages
from imutils.video import VideoStream
import numpy as np
import sys
import argparse
import imutils
import urllib.request
import time
import cv2
from urllib.request import urlopen



host = 'http://192.168.43.236:8081/'
url = host + 'video'

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("--prototxt", required=True,
	help="path to Caffe 'deploy' prototxt file")
ap.add_argument("--model", required=True,
	help="path to Caffe pre-trained model")
ap.add_argument("--source", required=True,
	help="Source of video stream (webcam/host)")
```

```
ap.add_argument("-c", "--confidence", type=float, default=0.2,
	help="minimum probability to filter weak detections")
args = vars(ap.parse_args())

# initialize the list of class labels MobileNet SSD was trained to
# detect, then generate a set of bounding box colors for each class
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
	"bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
	"dog", "horse", "motorbike", "person", "pottedplant", "sheep",
	"sofa", "train", "tvmonitor"]
COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))

# load our serialized model from disk
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

# initialize the video stream, allow the cammera sensor to warmup,
print("[INFO] starting video stream...")

if args["source"] == "webcam":
	vs = cv2.VideoCapture(0)
elif args["source"] == "edison":
	vs = cv2.VideoCapture("http://192.168.43.236:8081")
time.sleep(2.0)

stream = urllib.request.urlopen(url)
bytes = bytes()
detected_objects = []
# loop over the frames from the video stream
while True:
	# grab the frame from the threaded video stream and resize it
	# to have a maximum width of 400 pixels
	bytes += stream.read(1024)
	a = bytes.find(b'\xff\xd8')
```

```
        b = bytes.find(b'\xff\xd9')
        if a != -1 and b != -1:
                jpg = bytes[a:b+2]
                bytes = bytes[b+2:]
                i = cv2.imdecode(np.fromstring(jpg, dtype=np.uint8),
cv2.IMREAD_COLOR)
                frame = imutils.resize(i, width=800)

                # grab the frame dimensions and convert it to a blob
                (h, w) = frame.shape[:2]
                blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)),
                        0.007843, (300, 300), 127.5)

                # pass the blob through the network and obtain the detections and
                # predictions
                net.setInput(blob)
                detections = net.forward()

                # loop over the detections
                for i in np.arange(0, detections.shape[2]):
                        # extract the confidence (i.e., probability) associated with
                        # the prediction
                        confidence = detections[0, 0, i, 2]

                        # filter out weak detections by ensuring the `confidence` is
                        # greater than the minimum confidence
                        if confidence > args["confidence"]:
                                # extract the index of the class label from the
                                # `detections`, then compute the (x, y)-coordinates of
                                # the bounding box for the object
                                idx = int(detections[0, 0, i, 1])
                                box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                                (startX, startY, endX, endY) = box.astype("int")
                # draw the prediction on the frame
                label = "{}: {:.2f}%".format(CLASSES[idx],
                    confidence * 100)
                detected_objects.append(label)
                cv2.rectangle(frame, (startX, startY), (endX, endY),
                    COLORS[idx], 2)
                y = startY - 15 if startY - 15 > 15 else startY + 15
                cv2.putText(frame, label, (startX, y),
                  cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)


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

        # if the `q` key was pressed, break from the loop
        if key == ord("q"):
            break

            cv2.imshow('i', i)
            if cv2.waitKey(1) == 27:
                exit(0)



# do a bit of cleanup
cv2.destroyAllWindows()
```

REFERENCES

- Internet of Robotic Things: concept Technologies, and Challenges, Partha pratim Ray.
- Implementation of Reconfigurable Robot Mechanism for Earthquake Rescue Operation, Swathiraj G.O.* and Madhevan B.**
- A Cost Effective Way to Build a Web Controlled Search and CO Detector Rover, Ali Adib Arnab, Sheikh Sadia Afrin, F.M. Fahad and Hasan U. Zaman.
- Cost Effective Motion Based Stair Climbing Rover for Rescue Purpose, Sazid Al Ahsan1, Amir Hamza, Md. Hasibur Rahaman, Tasmiah Tamzid Anannya2, Md. Mahboob Karim
- Disaster Relief and Data Gathering Rover S.V.V. Srinivas, Aditya Kumar Singh, Aman Raj, Abhishek Shukla, Rachit Patel, Aviral Malay
- https://tutorials-raspberrypi.com/configure-and-read-out-the-raspberry-pi-gas-sensor-mq-x/
- https://learn.adafruit.com/adafruit-amg8833-8x8-thermal-camera-sensor/raspberry-pi-thermal-camera

BIODATA

Name	: Tirumalasetty Venkata Sai Karthik
Mobile Number	: +91 7358005458
E-mail	: tirumalasettyvenkata.saikarthik2015@vit.ac.in
Permanaent Address	:

Name	: Koppaka Shivadeep
Mobile Number	: +91 9789822807
E-mail	: koppaka.shivadeep2015@vit.ac.in
Permanaent Address	:

Name	: Chakkirala Nikhil Chakravarthy
Mobile Number	: +91 9790723771
E-mail	: chakkiralanikhil.chakravarthy2015@vit.ac.in
Permanent Address	: Flat No. 408, A-Block, Fortune Heights, Sivaji Nagar, Kurmannapalem, Visakhapatnam, Andhra Pradesh-530046