IoT based Rover:

Functions:

* Moves front, back, right, left.
* Gives a live feed of the surroundings.
* Can switch to night view.
* Can detect and avoid obstacle using IR sensor
* Can detect gas🡪 <https://tutorials-raspberrypi.com/configure-and-read-out-the-raspberry-pi-gas-sensor-mq-x/>
* Can detect heat signatures using thermal camera setup (costly) <https://learn.adafruit.com/adafruit-amg8833-8x8-thermal-camera-sensor/raspberry-pi-thermal-camera>
* Live audio stream. <https://blog.mutsuda.com/raspberry-pi-into-an-audio-spying-device-7a56e7a9090e>
* All cameras can be rotated 3600 using motors.

<https://tutorials-raspberrypi.com/raspberry-pi-sensors-overview-50-important-components/>

Disadvantages of drone:

* aerial imagery isn’t much of advantage over satellite imagery.
* Difficulty in sensing gas.
* Small mistake in flight control can cause permanent damage.
* Increased weight due to the sensors🡪more power required for thrust🡪bigger battery🡪more weight again!
* Difficult to operate on a windy or a rainy day.
* Costly.

**##Audio feed:**

The Raspberry Pi has an audio output jack, but unfortunately there is no input port. We will then need a USB microphone.

Once the microphone is plugged in, you might have to load the audio module by typing the following command if it isn’t.

sudo modprobe snd\_bcm2835

**Recording and playing a test file**

Now we can try to record some audio into a file by running:

arecord -D plughw:1,0 test.wav

Just press CTRL+C once you think you’ve got enough recording. Now let’s play it to see if it works! But first, plug your earphones to the audio output of your Raspberry Pi!

If you want to record louder or adjust some parameter, you can use the alsamixer tool to play with the input/output levels of your microphone.

alsamixer

Once you have set your settings, remember to store your changes:

sudo alsactl store

#### Streaming the audio to another PC

What we want to do next, is to give the Pi the capability to send this audio through the network, maybe to a server we have somewhere. In order to do that, we will pipe the audio from the microphone into an ssh communication to the destination computer.

arecord -D plughw:1,0 -f dat | ssh -C user@remoteip aplay -f dat

Switch on your speakers in the destination PC, and hear what the Raspberry Pi is spying

**#**B**uild 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.

**Code:**

from flask import Flask

from flask import render\_template, request

import RPi.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)

**Explanation:**

We 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.

**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.

**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="camera right" style="font-size:30px;text-decoration:none;">CameraRight<br> &#x1F882;&#x1F882;</a><br><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');*

*});*

*});*

*</script>*

*</body>*

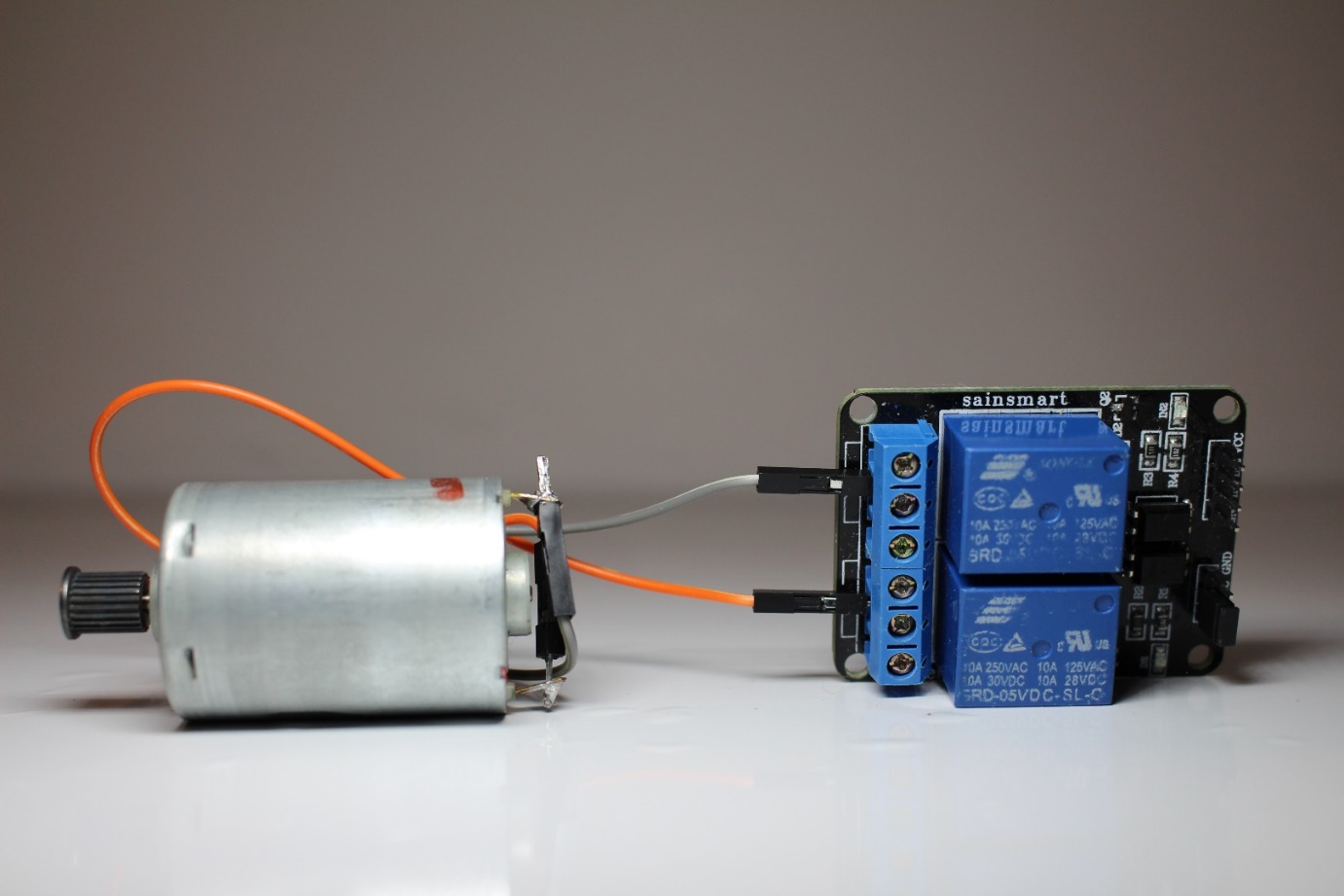
*</html>*

**##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:

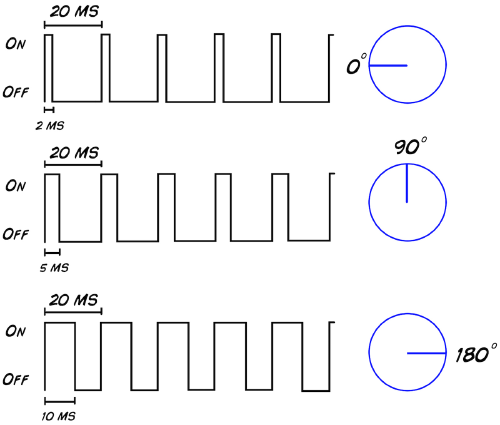
1. Normally Open (NO) port.
2. Normally Closed (NC) port.
3. Common port.



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.

**##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.



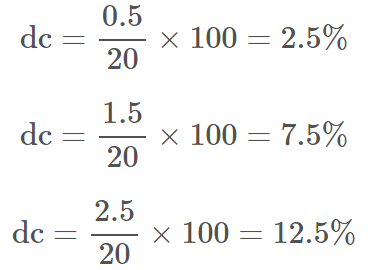
To control the servo motor from the Raspberry Pi we are going to use the PWM module in RPi.GPIO. GPIO 12 has been used as the PWM pin using the command:

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:



To change the duty cycle we can use:

p.ChangeDutyCycle(\_)

To stop the pulse emission:

p.stop()