SPEED CONTROL USING ULTRASONIC SENSOR

FINAL REPORT

GROUP 14
SAGAR KANANI : 34091
PRATIBHA RACHEL JOHN : 33912

SUBMISSION : 28/06/2021

Contents

1. STATE OF THE ART

The goal of this topic is to measure the distance of obstacles by using an Ultrasonic sensor automatically control the speed by detecting the range of obstacles or accidental prone zones. There are many techniques used for speed control like image processing using camera along with raspberry pi 3, MAV (micro air vehicle), cruise control system and ultrasonic sensor alone based systems. These methods are particularly suitable for environments where optical sensors are unusable such as smoke, dust and similar.

Image processing with raspberry pi 3 technique is helpful to detect sign boards of speed control and the control the speed of motor with specific speed limit using image processing limit and also used to detect objects with in range and reduce speed of motor when object is sensed. For this it needs a 3D Camera, Signal conditioning unit, raspberry pi, BLDC Motor (Brush-less),display screen, detection sensor.

MAV is developed for indoor and outdoor environmental conditions. It operates with many sensors involved like infrared sensor (IR), laser sensor, sound sensor and a camera. Here different types of sensors are used to detect obstacles and measure the distance. For adaptive speed control, the hardware used is ATmega 328 processor, APM autopilot board, ESC (electrical speed control), GPS, MAV Quadcopter.

Cruise control system is also known as Adaptive cruise control system, Conventional cruise control system, Automatic Cruise control system, speed control system,etc. It is useful during traffic when speed is to be reduced before approaching another vehicle and accelerate again after the traffic is cleared. It is an efficient technology to avoid high collision risk. The hardware used includes Microcontroller, LCD display, SONAR, DC Motor.

There are many more alternative options to control a robot by using a tether (wired), wireless (remotely controlled by Bluetooth) or autonomously (predefined strategy or software coding).

2. STANDARD SOLUTIONS

The solution regarding the speed control varies vividly depending on the application and the level of complexity with respect to the technology incorporated. It can vary with a small project with Arduino to the application in a commercial vehicle involving neural networks. A combination of different components together can possibly provide some standard solutions in the field of speed control which have already been part of experimentation and implementation for real time application by many. Some of those standard solutions as part of the research are explained below.

a. Automatic Vehicle Speed Control: to control the speed of a vehicle by obstacle detection and also in accident-prone/restricted zones(schools, pedestrian crossing etc), a combination of different components like RF module, Ultrasonic sensor, GSM module and analyzing the data accordingly in python can be a possible solution. The overall working includes the communication between the RF module between the speed post and the vehicle controller system which provides a signal according to which the vehicle speed is controlled (if RF gives a signal of restricted zone and the vehicle is speeding, the speed is lowered automatically with adhering control strategy). The Ultrasonic sensor would detect any obstacle (principle: Doppler Effect) and the information is passed to a Microcontroller which controls the speed using PWM pulses. A GSM modem could aid in transmitting the data detected in the speed control system to the user mobile phone, which also includes the range of ultrasonic sensor distance. In addition to it, an Optocoupler is used along with the motors to measure the speed in real time. Python software could then analyse the data and plot graphs between speed and time before and after obstacle detection.

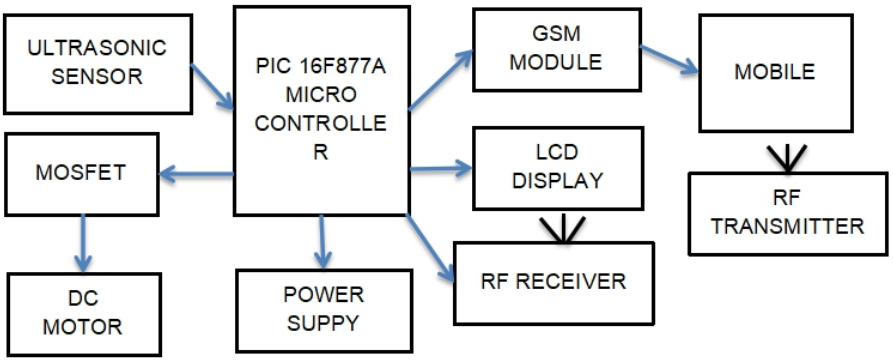


Figure 2.1: Working of the setup

[Referred from: MAT JOURNALS; Embedded System for Automatic Vehicle Speed Control and Data Analysis Using Python Software; Volume 5 Issue 2; Pages 14-20; 2019]

b. Cruise control system using Ultrasonic Sensor: Many commercial high end vehicles have this feature which can control the speed, reducing the probability of causing an accident by driver's mistake/carelessness. Measuring the distance between the vehicle in front, the speed could be controlled accordingly to avoid any possible collision and Ultrasonic sensors prove it's efficiency here compared to the conventional radioactive sensors which have a lower frequency and range compared to the Ultrasonic sensors.

Mostly, a Real time operating system(RTOS) is used along with the system basically for multi-tasking and to timely update with the current state of the vehicle system. The Ultrasonic sensor module HC-SR04 provides a range of 2cm-400 cm with the ranging accuracy of about 3mm. The IO trigger for at least 10microseconds high level signal.The module automatically sends eight 40kHz and detects whether a pulse signal gets back. If the signal is back through a high level time of high output IO duration is the time sending ultrasonic waves to return.

Test distance = (high level time * velocity of sound (340 m/s) /2)

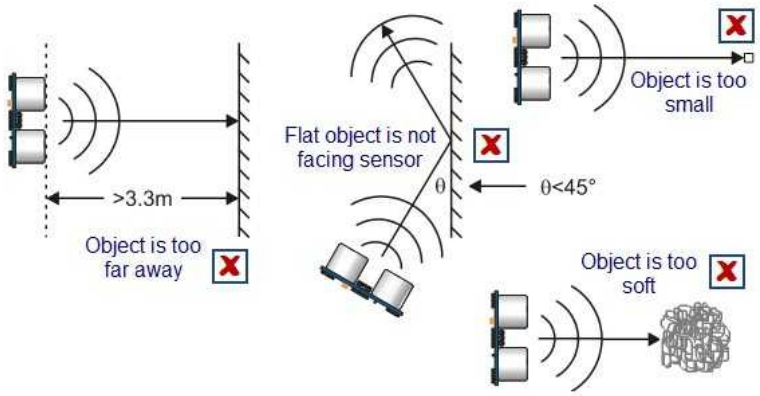


Figure 2.2: Ultrasonic Sensor; sensing conditions

[Source:https://www.geekstips.com/arduino-snow-depth-remote-sensing-with-ultrasonic-sensor/]

c. Ultrasonic Sensor and Image Processing Technique with Raspberry Pi 3 to control speed: A camera that is dedicated for speed control by image processing is used along with the Ultrasonic sensor to reduce the speed in-case of obstacle detection. This is specifically needed for real application as mentioned in other examples previously, in a vehicle. The sign boards are an integral part and need to be recognized to control the vehicle speed (automated vehicle) which requires a camera or other means of recognition. The logical signals are provided from the Raspberry pi to the Ultrasonic sensor, camera (through Camera Serial Interface (CSI)), motors and hence controlled.The on board 802.11n Wifi module ,Bluetooth is 4.1 Classic low energy and Display Serial Interface (DSI) could be used for headless connectivity and display with LCD.

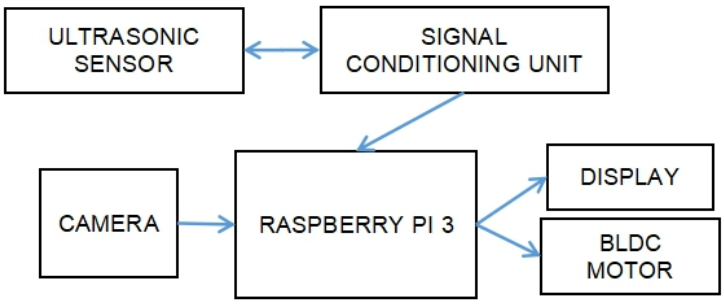


Figure 2.3: Working of the setup

[Referred from: IRJET;Speed Control of Electric Motor using Ultrasonic Sensor and Image Processing Technique with Raspberry Pi 3; Volume 06 Issue 01; Jan 2019]

d. Line Follower Robot with Obstacle Avoidance Capability : A conventional line follower robot will follow a specified path (eg:-transporting goods in industries) without avoiding obstacles. This can be enhanced by including an Ultrasonic sensor which can control/reduce the speed or eventually stop if it encounters an obstacle. Adding the features of obstacle avoiding robot to a traditional line follower robot prevents any damage to the robot. This can be implemented with Arduino UNO on which the Ultrasonic sensor and IR sensor(2) is connected. The ultrasonic sensor library has to be installed in Arduino IDE. In the program both the IR sensors have to be initialized. Four output pins of the motor have to be initialized. Three variables have to be declared, two for both the IR sensors and one for the ultrasonic sensor.The two variables which are declared for the IR sensor will read the value of IR sensor1 and IR sensor2. The variable which is declared for the ultrasonic sensor checks for any obstacle till a mentioned distance. If the ultrasonic sensor detects any obstacle in its path all the motors should stop, the four output pins of the motor drive should be programmed as LOW, which means they should stop working. So when an obstacle is detected by the ultrasonic sensor then the motors will stop and the robot will stop till the obstacle is removed from its path. When no obstacle and no black line is detected then the robot should move forward.

e. Maneuver Emergency braking with Ultrasonic Sensors: The assistant constantly monitors the vehicle's surroundings by means of ultrasonic sensors. Using the steering angle and driving speed, the system calculates the car's intended path and identifies any obstacles that could get in the way when maneuvering into the parking space. In the event of an imminent collision, the assistant applies the brakes in good time until the car stops, thereby preventing expensive and annoying dents or scratches. This is an advanced incorporation of Ultrasonic sensors for speed control that requires precision and faster feedback operation of the system.

f. Multiple Ultrasonic Range Sensors: Two Ultrasonic Sensors on either side of the vehicle can be used to control the rotational direction of both the motors to avoid both front and side collisions effectively which is sometimes not achieved when sensors are only at the front of a bigger vehicle and hence some blind spots on side of vehicle is formed where the range of sensor does not cover. These sensors could then be connected to an Arduino or Raspberry pi and motor controller modules with the motors to be controlled.

3. IMPLEMENTATION STRATEGY IN HARDWARE

The available hardware (Raspi based Rover) works headless via the VNC viewer and has the following components included :

Table 3.1: Components

SHIELD	HEARTBEAT LED PUSHBUTTON H-BRIDGE (L2938D) VOLTAGE-REGULATOR (LM1084)
POWER SUPPLY UNIT	USB 9V RECHARGABLE BATTERY
DRIVE	2 MOTORS, ROLLER BALL(front)
ULTRASONIC DISTANCE SENSOR	HC-SR04
REFLEX LED/PHOTODIODE	

We do not have cameras or RF module to detect the obstacle. Hence, localizing the task depends solely on the internal code/ system with the Ultrasonic sensor. HC-SR04 is said to provide an amiable sensing range and appreciable accuracy which can aid us in the task of controlling speed according to the maneuvering situations. PWM signals from our Microcontroller can control the speed of the DC Motors after receiving signal from the Ultrasonic sensor in a programmed fashion. Autonomous control is useful nowadays which can be introduced by obstacle avoidance. But manual control is better in terms of accuracy (when you have to control speed).

Ultrasonic sensor transmits waves and receives the echo from any objects in its proximity which then detects for the presence of any obstacle. If any object happens to be in the path of the robot, the sensor detects it as mentioned before and sends a signal to the Microcontroller. The Microcontroller then produces required PWM pulses to the electric motor to slower down the speed/stop the motor as such. All these will be done automatically without the intervention of the user, subsequently reducing the chances of any collisions. The Photodiode can be of further advantage for detecting any change of path from the marking area and hence give a signal to reduce speed , throughout the maneuvering of the rover.

For the effective implementation, we need to have a prior idea of the different maneuvering situations of the robot, for example;

- ✓ *If the robot needs to turn after 100 cm to avoid a boundary wall and the rover is currently at a higher speed, it should be able to detect the curved boundary and slower down the speed to avoid getting off the track or hitting the wall.*
- ✓ *If an object is in the way of the rover (trivial case), it should detect beforehand and slow down the motor to eventually stopping it.*
- ✓ *An additional case could be thought upon, when an obstacle is detected and is removed before the rover reaches it, in which the rover could reduce the speed at first and then increase it without stopping the motor.*
- ✓ *Another situation could be to reduce the speed and stop the motor to then make the rover move in reverse direction - in case of an obstacle moving coming towards the rover.*

4. WORK LAYOUT

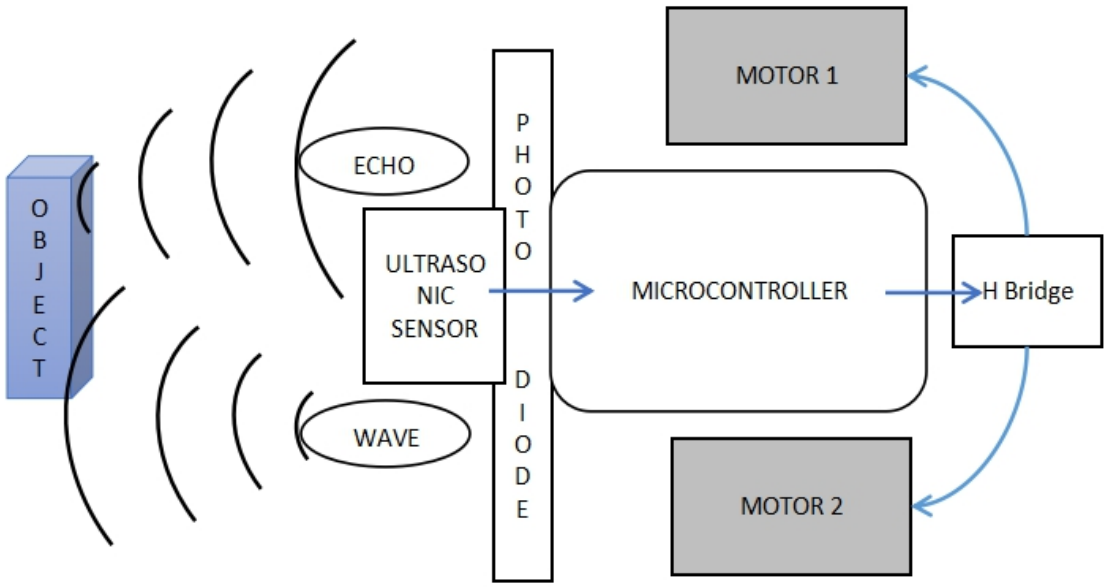
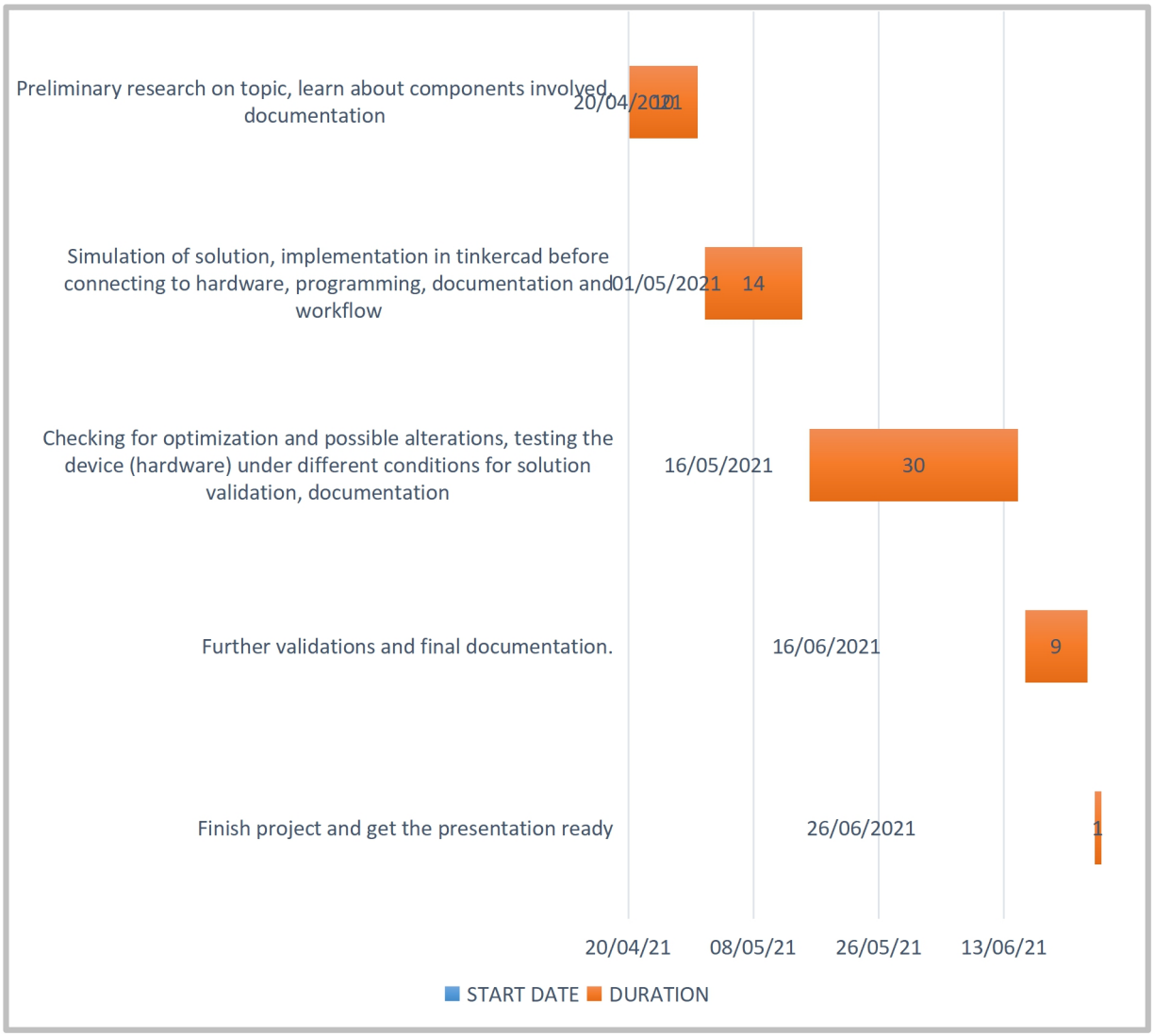


Figure 4.1 : Block diagram for the working

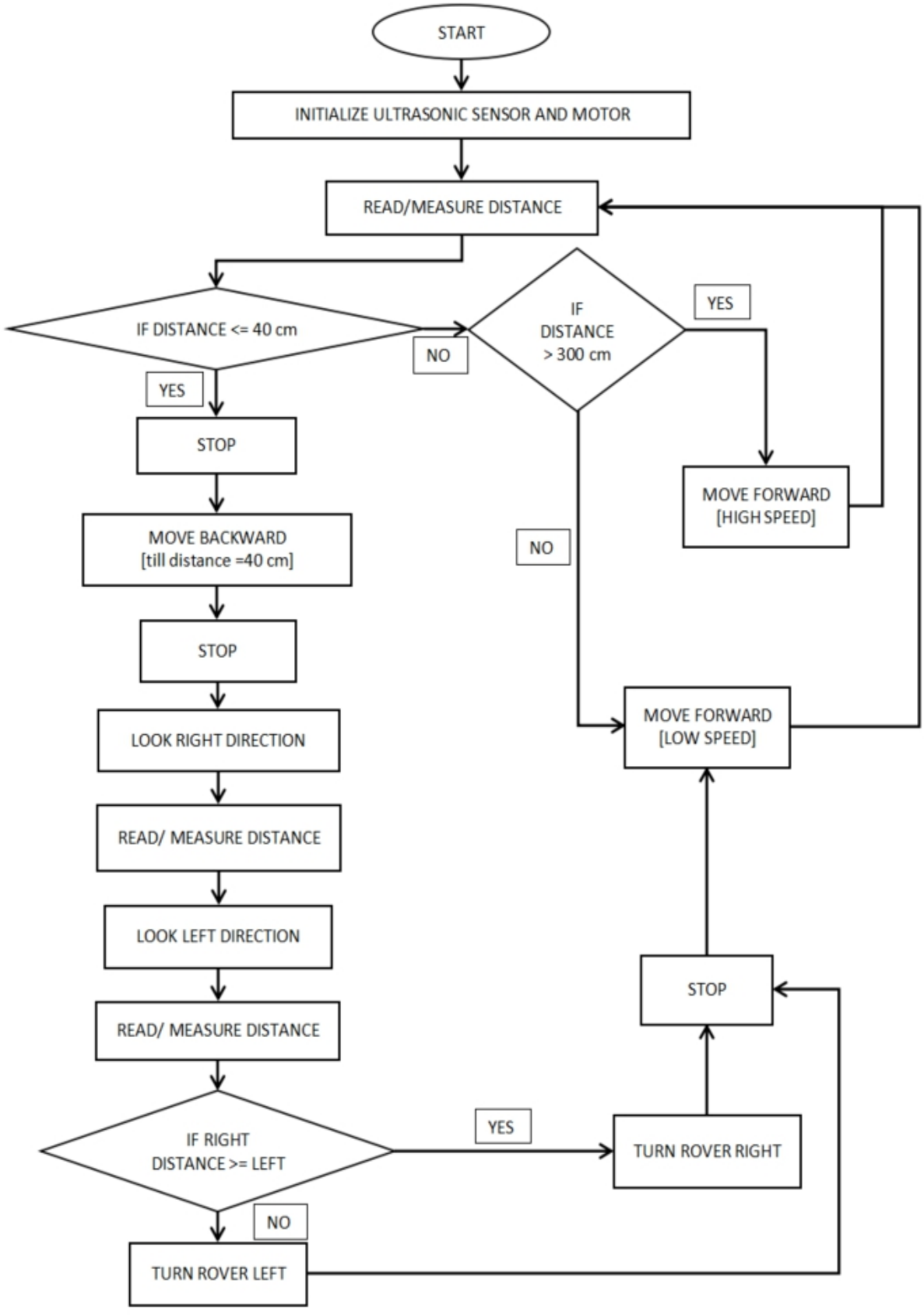
Outline of the Work

1. Implement the ideas into program to provide input signals and control strategy which can slow down/stop/speed up the motors depending on different maneuvering situations.
2. Detection of any object within the range by the Ultrasonic sensor
3. The distance measured by ultrasonic sensor is required to decide the speed to which the motors need to be changed..
4. Signal is sent to the Microcontroller, based on the distance measured by sensor.
5. Microcontroller provides PWM signal to H-Bridge according to the speed-changes required, based on the distance between rover and the obstacle.
6. The H-Bridge controls (turns) /stops both the motors based on PWM signal and the input signals for each motor.
7. Different scenarios need to be worked upon to ensure the controllability of the program and if any further alterations need to be incorporated.
8. Addition of the usage of Photodiode to aid the speed control is a viable option to be look upon.
9. The accuracy of Ultrasonic sensor measurements need to be calibrated if possible.
10. Testing the rover to marginalize the limitations and ensuring the rover has the capability as intended is vital.

5. EXPECTED PROJECT TIMELINE



6. FLOW CHART


START
INITIALIZE ULTRASONIC SENSOR AND MOTOR
READ/MEASURE DISTANCE
IF DISTANCE <= 40 cm
NO
IF DISTANCE > 300 cm
YES
YES
STOP
MOVE FORWARD [HIGH SPEED]
MOVE BACKWARD [till distance =40 cm]
NO
STOP
MOVE FORWARD [LOW SPEED]
LOOK RIGHT DIRECTION
READ/ MEASURE DISTANCE
LOOK LEFT DIRECTION
STOP
READ/ MEASURE DISTANCE
IF RIGHT DISTANCE >= LEFT
YES
TURN ROVER RIGHT
NO
TURN ROVER LEFT

6.1 STRATEGY OUTLINE

- The rover is kept at a place where it need to be traversed and the subsequent functional operation is started by powering it up.
- The ultrasonic sensor and the motors are initialized to start their functions through the program and the sensor starts measuring the distance
- Now, the measured distance is checked if it meets the different conditions stated.
- If the distance comes up to be less than or equal to 40 cm (a safe distance for turning if needed), the rover should stay (if already at standstill) or stop (if it was in motion) and move backwards till the safe distance is reached.
- The sensor should then look for right and left directions and measure the respective distances for further decision making and action.
- If the distance towards rightwards direction is larger than towards left, the rover should turn at spot towards right or vice-versa.
- The rover then moves forwards at lower speeds(for initial safety) and the distance is measured again by sensor.
- The above steps continues throughout the program.
- In case, the distance measured initially is greater than 300cm, the rover could travel faster and if the distance is less than 300 cm but greater than 40cm, the rover could travel at a slower pace.
- The distance is measured throughout by the ultrasonic sensor and speed is controlled accordingly

Note: *Additional Control strategy changes may be included during program implementation or inclusion of Photodiode in a later time.*

7. IMPLEMENTATION OF THE PREVIOUSLY DEVELOPED LOGIC

The logic developed earlier is explained here for a layout to the code which is included within this section. The code is a first draft which is constantly updated as tests are carried out with possible changes and alterations for a better performance. So, this section is formulated in such a way so that any one who reads it can understand the issues developed from the first logic and the subsequent alterations that was included and the reason for the same which will provide with better understanding.

The code is drafted in Visual Studio and then tested within Thonny Python by connecting the Raspberry pi with VNC Viewer and subsequent changes are made.
Firstly, the pins which we require for the functionality is defined which includes the Led, Motors(PWM and I/O pins) , Ultrasonic (trigger and echo pins) etc. Then different functions are written and declared that does certain tasks when called under different situations. This includes functions for forward & reverse motion of each motors and functions for the specific motion of the whole rover that actuates and controls the motor based on different distance inputs from the Ultrasonic sensor, controlling the speed basically to traverse without hitting any obstacle.

There are three exclusively defined functions - def straight( ), def back( ), def turn( ) that does the task of moving the rover across a path by measuring the distance in a timely fashion and avoiding obstacles by taking a turn or by stopping the rover completely.

- def straight( ) :- This function is called at the beginning if the distance measured is greater than 40 cm at the starting position of the robot else the back( ) function is called. Here the task is to move the rover forward in a straight direction at a specific speed and stop when the distance becomes less than 40 cm [back( ) function is then called after the condition is checked].
- def back( ) :- This function is called at the beginning if the distance measured is less than 40 cm at the starting position of the robot. The task is to move the rover backwards for some distance and then stop to turn to a direction where there is no obstacle. To turn the rover, a function turn( ) is called within this function declaration.
- def turn( ) :- This function turns the rover to right or left depending on the distance measured, by 90° or 180° from its current position at that moment, at a lower speed [duty cycle 25 or 30]. The turn is realized by making the robot to spin at a point instead of turning at a radius while we assumed first the room has 90° corners. After turning the straight( ) function is called to move the rover further in the path without obstacles.

[Further alterations and changes are later made to optimize the motion of the rover, which is explained in later sections]

A distance of 40 cm was taken as a safety distance to set a limit or boundary for the robot to have its movements, considering the measurement inaccuracies at times by the Ultrasonic

sensor due to variations in object texture or geometry. This distance was chosen after testing the robot motion with different lesser distances to which it possibly reacts to stop without any ambiguity.

The figure below shows the logic that makes the rover move by measuring distance (D) continuously and avoiding obstacles by controlling the speed of the rover.

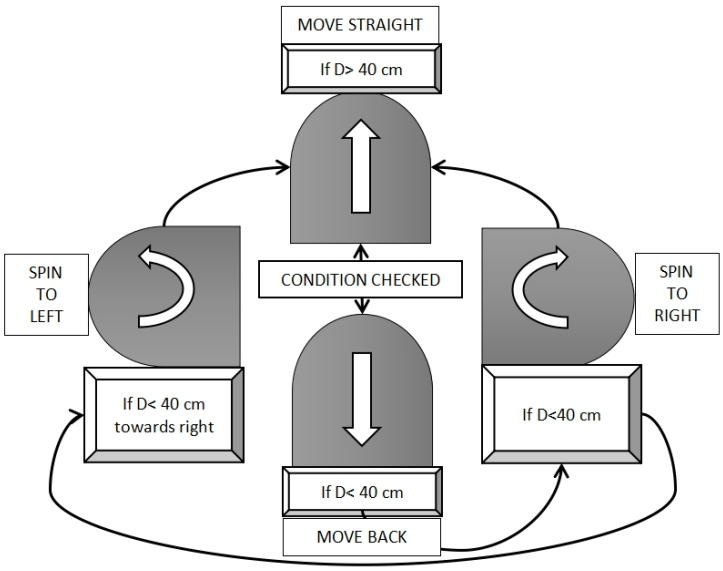


Figure 7.1: Logic of the Rover motion by reacting to the measured distance

PROGRAM TO RUN THE ROBOT

```
import RPi.GPIO as GPIO
import time
from gpiozero import DistanceSensor
from time import sleep

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

duty_cycle = 0
LED = 7
Taster = 5

Motor1_PWM = 18
Motor1_IN1 = 17
Motor1_IN2 = 22

Motor2_PWM = 19
Motor2_IN1 = 24
Motor2_IN2 = 4

#SR04
trigger=25
```

```
echo=27

sensor = DistanceSensor(echo=27, trigger=25)

def M1_forward():
    GPIO.output(Motor1_IN2,GPIO.LOW)
    GPIO.output(Motor1_IN1,GPIO.HIGH)

def M1_backward():
    GPIO.output(Motor1_IN1,GPIO.LOW)
    GPIO.output(Motor1_IN2,GPIO.HIGH)

def M2_forward():
    GPIO.output(Motor2_IN2,GPIO.LOW)
    GPIO.output(Motor2_IN1,GPIO.HIGH)

def M2_backward():
    GPIO.output(Motor2_IN1,GPIO.LOW)
    GPIO.output(Motor2_IN2,GPIO.HIGH)

 #function to move forward straight
def straight():
    while( sensor.distance * 100 > 40): #condition check for safe distance
        print("Distance: ",sensor.distance*100, "cm")
        GPIO.output(LED, GPIO.HIGH)
        PWM_1.ChangeDutyCycle(80)
        M1_forward()
        PWM_2.ChangeDutyCycle(65)#since rotor doesn't move straight at same duty cycle
        M2_forward()
        time.sleep(0.2)
        GPIO.output(LED, GPIO.LOW)
        if (sensor.distance * 100 < 40):
            back() #function call to make a reverse motion

#function to take a small reverse to turn later
def back():
   if (sensor.distance*100 < 40):
        print("Distance: ",sensor.distance*100, "cm")
        PWM_1.ChangeDutyCycle(30)
        M1_backward()
        PWM_2.ChangeDutyCycle(30)
        M2_backward()
        time.sleep(0.5)
        PWM_1.ChangeDutyCycle(0)
        PWM_2.ChangeDutyCycle(0)
        time.sleep(0.175)
        turn() #function call to take a turn to either left or right
```

```
#function to turn either left or right
def turn():
    while(sensor.distance *100 <40):
     print("Distance: ",sensor.distance*100, "cm")
     GPIO.output(LED, GPIO.HIGH)

     PWM_1.ChangeDutyCycle(25) #turning right with smaller speed
     M1_forward()
     PWM_2.ChangeDutyCycle(0)
     M2_forward()
     time.sleep(0.7)   #time needed for a 90° rotation
     PWM_1.ChangeDutyCycle(0)
     PWM_2.ChangeDutyCycle(0)
     time.sleep(0.175)
     if (sensor.distance *100 < 40):  #if the right direction has obstacle, turn left
        PWM_1.ChangeDutyCycle(0)  #turning left
        M1_forward()
        PWM_2.ChangeDutyCycle(25)
        M2_forward()
        time.sleep(1.31) #time needed for a 180° rotation
        PWM_1.ChangeDutyCycle(0)
        PWM_2.ChangeDutyCycle(0)
        time.sleep(0.175)
    straight() #function call to make a straight motion after taking the correct turn

GPIO.setup(Motor1_IN1,GPIO.OUT)
GPIO.setup(Motor1_IN2,GPIO.OUT)
GPIO.setup(Motor1_PWM,GPIO.OUT)
PWM_1 = GPIO.PWM(Motor1_PWM, 90) #GPIO as PWM with 90Hz
PWM_1.start(0) #Duty Cycle = 0

GPIO.setup(Motor2_IN1,GPIO.OUT)
GPIO.setup(Motor2_IN2,GPIO.OUT)
GPIO.setup(Motor2_PWM,GPIO.OUT)
PWM_2 = GPIO.PWM(Motor2_PWM, 90) #GPIO as PWM with 90Hz
PWM_2.start(0) #Duty Cycle = 0

GPIO.setup(LED, GPIO.OUT)

d = sensor.distance * 100

#Starting of Condition checks for rover motion
if (d< 40):
    back()
else:
    straight()
```

7.1 PRECAUTIONS TO BE TAKEN WHILE TESTING

- Keep lighter or softer objects along the path as obstacles to be tested to lower the impact on the rover in case of any hitting.
- For the preliminary testing, make sure the duty cycle is kept at a minimal value for the same reason stated above. Speed could be subsequently varied for testing as you are confident about the program
- Make sure the battery is charged fully before testing since a lower battery can create difference in the performance characteristics of the motor and hence the rover motion in effect.

7.2 TESTING PHASE AND SUBSEQUENT ALTERATIONS MADE IN PROGRAM

- ✓ During the testing phase, while running def straight() function, the rover was not going in a straight direction but instead moving left a little bit. To control this and make the rover move straight, PWM duty cycle variation was required. PWM 1(left) being 80, the PWM 2 (right) was changed to a duty cycle of 65.
- ✓ Another problem faced was when distance is less than 40 and robot had to take right or left turn, during that condition robot didn't spin exactly 90° or 180° from one side to another side. Resolving that issue, time sleep changes were required for spin of the rover at 90 and 180 degrees.

Non-rectifiable situations:

- ✓ The obstacles that were too low in height like a small pen or other objects on the floor were not detected and hence went over it which created a jerky effect.
- ✓ Sensor could measure a range maximum of 100 cm and not more as described in catalogue. Hence the idea to have an increased speed with distance above 300 cm as mentioned in flow diagram was not implemented

8. FINAL OPERATION OF THE ROBOT

To get a clear idea of the rover motion with respect to the above mentioned code, a video link is given below that depicts the movements.

- Video Link: *https://youtube.com/video/VFjQohlNCls/edit*

The measurements from a different situation where the rover is able to detect some surfaces and not some small obstacles that leads to a change in the path it travels is explained below with a graph from Python shell interface.

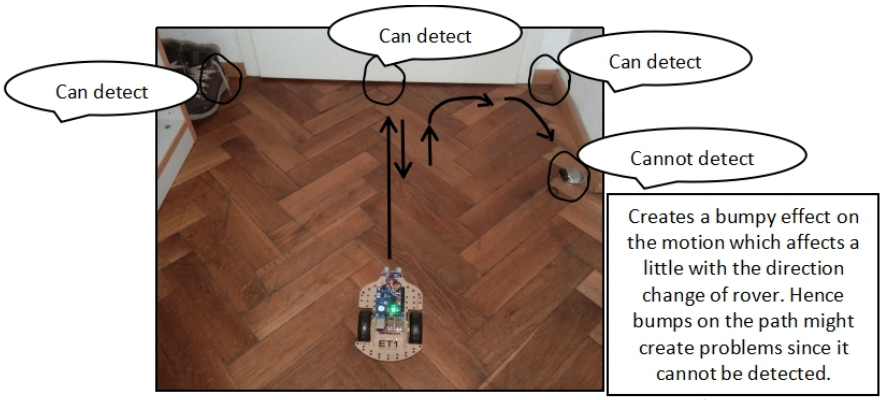
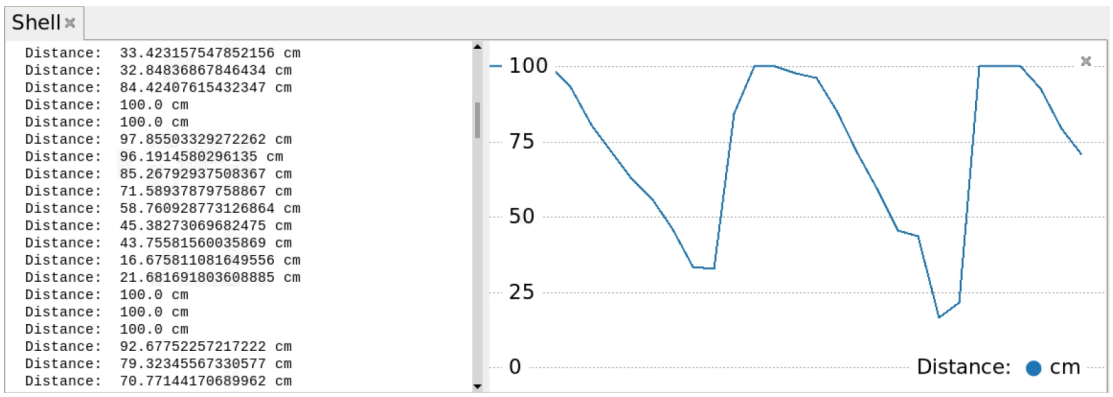


Figure 8.1: Moving in a closed space with a bumpy obstacle that is not detected

The graphs shown below are from the above situation motion where the rover traverses inside a closed space and turns when a wall is reached. The small bump like structure makes the wheels to turn and travel in different direction since it is not detected by the sensor due to minimum height requirements for the sensing. This bump could create serious change in directions or even a jerk if the rover is at higher speeds which could be lowered if the obstacles so low in height could be detected to take a different path.



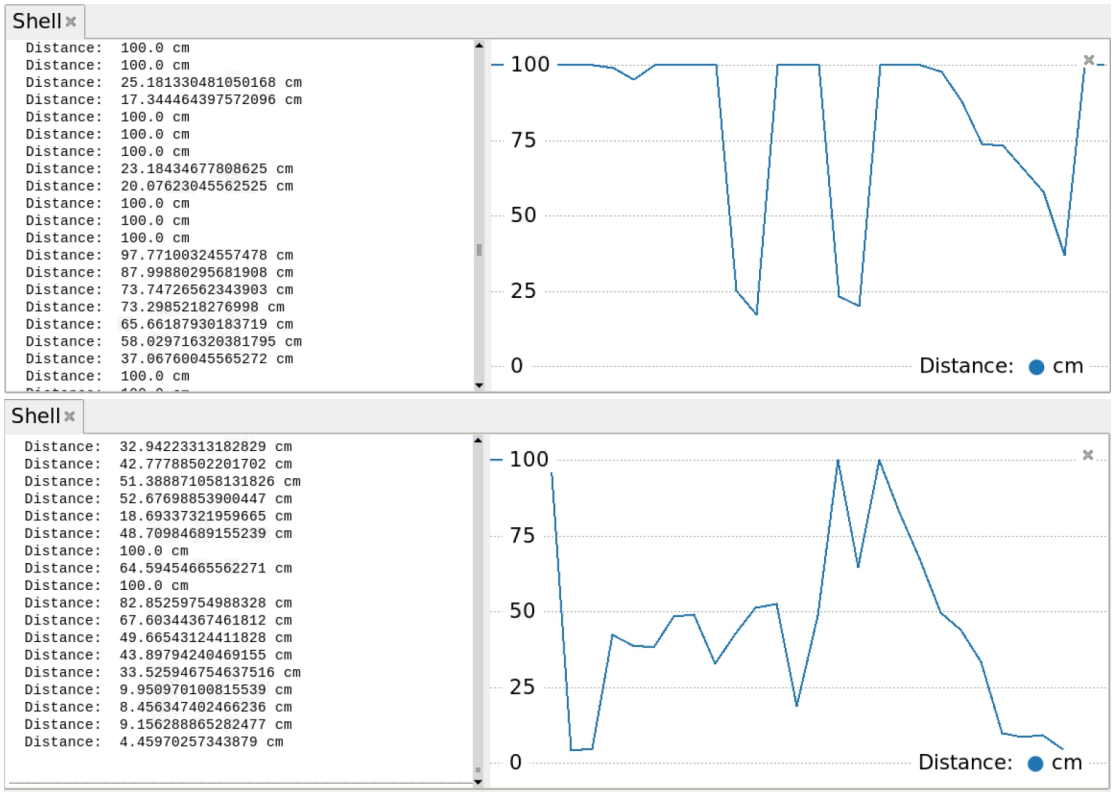


Figure 8.2: Graphs from Python shell which displays the distance with time.

9. CONCLUSION

The task of controlling the speed and subsequently have a motion without any collision was tried to implement within the limitations of our knowledge and available device. An introductory study about the components on the Rover and the given task of speed control was done and information about the existing sophisticated solutions were collected for reference. A suitable solution logic was implemented with work layout that could be realized with the Rover. An expected date of completion was also made to have a time layout to keep within the scheduled submissions.

The logic was realized into program in Python ans tested with the rover multiple times to have suitable alterations and changes for a better performance. Some of the situations were not rectifiable which was within the limitations of the whole set up of the rover and available devices. The use of Photo-diode was also not included since it didn't add up to any possible advantage to control speed unless a specific path was defined which in this case we haven't introduced.

A further scope of the project is possible in an accurate way if an Optocoupler is available to give the specific speed information of each wheel by the motor rotation. Also additional usage of different speed conditions for each wheel could be introduced into the logic for taking turns that are not particularly for a 90° bend but instead with some radius of curve. This too needs an accurate input of wheel speed information to make it perfectly realizable.