ANKARA UNIVERSITY

FACULTY OF ENGINEERING

ELECTRICAL AND ELECTRONICS ENGINEERING

BACHELOR THESIS

LOCALIZATION AND PATH PLANNING OF AN AUTONOMOUS OBSTACLE-AVOIDING VEHICLE

Arif Nasirov

ANKARA
2020

14.06.2020

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

In the field of intelligent vehicles, the problems of localization and path planning have attracted considerable attention. The process of robot localization and trajectory planning in vehicle operating scenes, that are partially or completely known, represents one of the main issues that are essential for providing the desired vehicle functionality. This paper introduces the basic elements of localization for an autonomous vehicle. For the autonomous motion, the vehicle should be able to localize its current position, perform path planning for its future movements, and move to the next position as expected avoiding all obstacles on its path. The vehicle should use the given localization information to plan the shortest path to visit some particular locations that are specified by user.

1. INTRODUCTION

The goal of this project was to create a system that will be capable to understand its location by precise use of an Inertial Measurement Unit (IMU) and path planning. Moreover, the vehicle should be capable of planning its path, that is performed by using already created and known map. Furthermore, sonar sensors attached to all physical corners of the vehicle should track placement of potential obstacles that can be intentionally created, and avoid them by using basic principle of obstacle avoidance.

For the case of the obstacle avoidance, the most difficult problem is environment recognition. The environment can be static or dynamic. Dynamic environment means that the objects it contains are in motion and their location can be changed over time, which creates a lot of uncertainties and information limitations. The static environment, however, have very useful mathematical data, which can be used in order to create an effective probabilistic model that can carry out the initial movement path for the vehicle.

1.1 Localization of the Autonomous Vehicle

The assigned tasks can be performed safely if a vehicle is able to move on accurate localization estimation.

Localization is the process of determining the position and orientation of the autonomous vehicle.

In order to solve Localization problem 2 primary steps are used in this thesis:

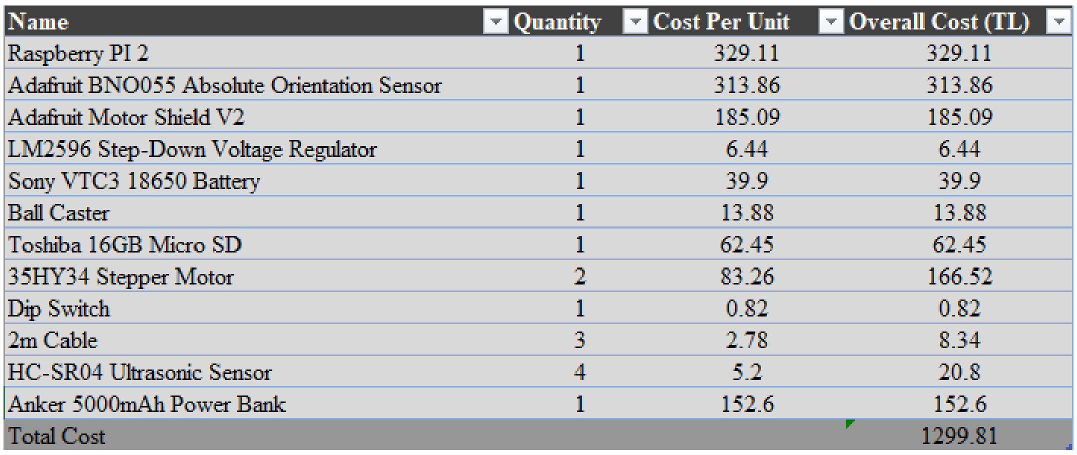
1) The vehicle must have a representation of its belief regarding its pose in the given environment.
2) A number of steps conducted by motors should be recorded and the coordinates should be calculated with respect to the known maps (environment).

2. HARDWARE AND CONNECTIONS

In this paper, the vehicle's construction and coding is completed. The vehicle can move in the uncertain distance and can rotate by using basic commands. The model of the vehicle was created by using "3D-Printer", "CURA" (software for slicing the model), "TinkerCAD" and "Solidworks". 4 sonar sensors are attached at all corners of the vehicle, this should give vision to the vehicle. This vision is required in order to avoid the obstacles in the dynamic environment. At this stage we have 2 stepper motors connected to motor driver that if required for giving the vehicle the basic ability of motion and rotation. Motors are chosen as stepper motors because the preciseness of the steps is one of our primary goals as it gives us the data required for the calculation of the localization of the vehicle.

2.1 List of Equipment and Vehicle Structure

For the beginning, list of the equipment was created. The list has to include all hardware equipment required in order to construct the vehicle that can reach a goal that was set. The Table 2.1 depicts the sum of all expenses and gives a list of equipment used by the end of the project. As it can be seen, total expense of completed project from the beginning is calculated as 1299.81 Turkish Lira.



Name	Quantity	Cost Per Unit	Overall Cost (TL)
Raspberry PI 2	1	329.11	329.11
Adafruit BNO055 Absolute Orientation Sensor	1	313.86	313.86
Adafruit Motor Shield V2	1	185.09	185.09
LM2596 Step-Down Voltage Regulator	1	6.44	6.44
Sony VTC3 18650 Battery	1	39.9	39.9
Ball Caster	1	13.88	13.88
Toshiba 16GB Micro SD	1	62.45	62.45
35HY34 Stepper Motor	2	83.26	166.52
Dip Switch	1	0.82	0.82
2m Cable	3	2.78	8.34
HC-SR04 Ultrasonic Sensor	4	5.2	20.8
Anker 5000mAh Power Bank	1	152.6	152.6
Total Cost			1299.81

Table 2.1 List of Equipment and expenses

2.1.1 Ultrasonic sensors

In order to give vision to the vehicle. 4 ultrasonic sensors were assigned to all corners of the vehicle. All sensors will have to constantly monitor all sides of the vehicle.

In order to give a vehicle chance to avoid obstacles. The ultrasonic sensors assigned was HC-SR04 (Figure 2.1). This sensor is capable of finding the obstacle from 4-meter distance within 15 ° angle and has comparatively low power consumption (15 mA / 5 V).

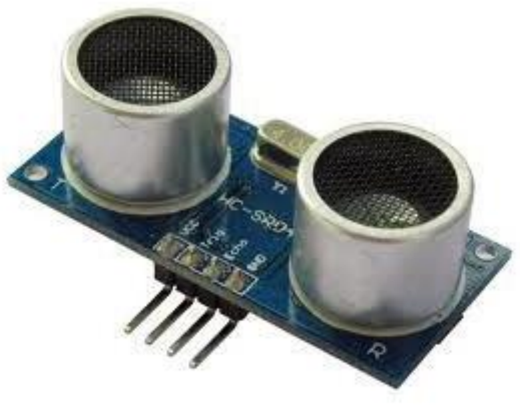


Figure 2.1 Ultrasonic distance sensor - HC-SR04

2.1.2 Motors

As it has been already told in introduction, 2 stepper motors are used. Stepper motors are used because they are comparatively precise in the response to the commands. By using this precision, the number of steps made by the motors can be monitored precisely, which means that the first part of the localization can be continued as the amount of the data transmitted will be conducted precise enough for localization. Both of motors used are Nema 14 35HY34 (Figure 2.2).

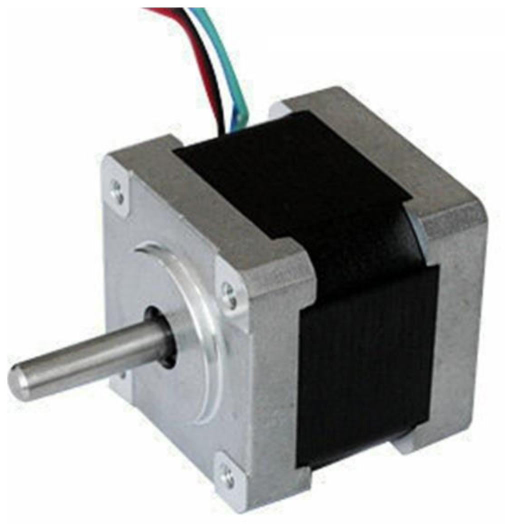


Figure 2.2 Step Motor 35HY34

High reliability and torque of this stepper motor increases the precision and reliability of the whole vehicle, which is very important. This is 4 leads 2-phase hybrid bipolar motor with power consumption of 1 A at 2.7 V per phase, allowing for a holding torque of 566 g. Totally this motor is conducting 200 steps per revolution, which means its step angle is 1.8 °. In order to control 2 coil motor (Figure 2.3), motor driver was used.

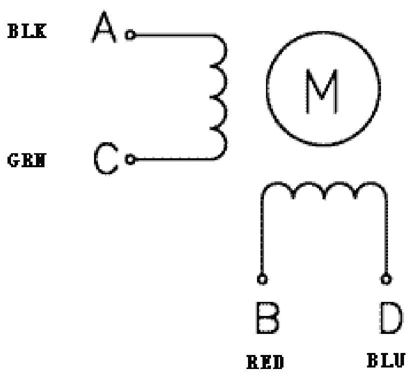
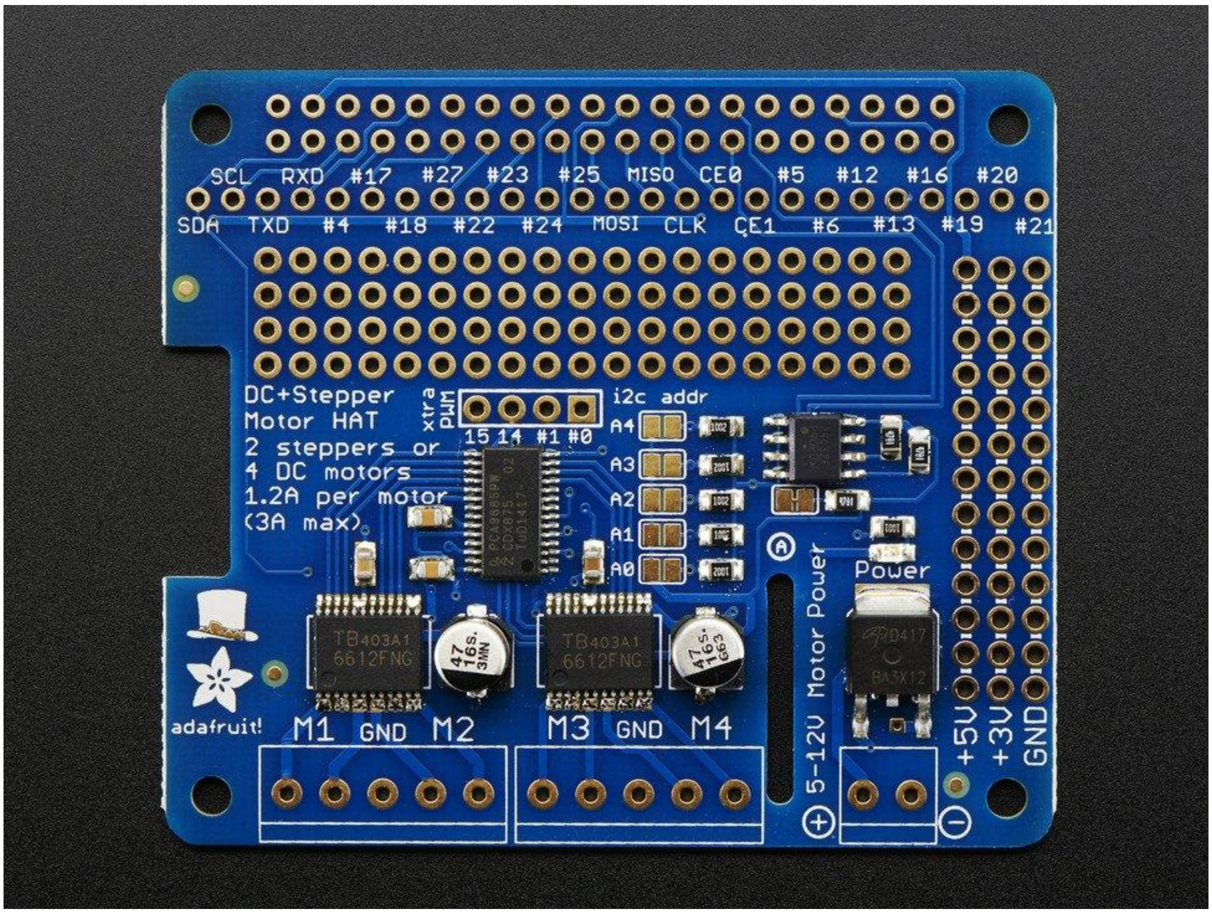


Figure 2.3 Bipolar Stepper motor wiring diagram

2.1.3 Motor driver

The motor described in the previous chapter requires bipolar motor driver in order to drive it. The motor driver chosen for this purpose was Adafruit DC & Stepper Motor Hat (Figure 2.4).



SCL RXD #17 #27 #23 #25 MISO CE0 #5 #12 #16 #20
SDA TXD #4 #18 #22 #24 MOSI CLK CE1 #6 #13 #19 #21
DC+Stepper Motor HAT 2 steppers or 4 DC motors 1.2A per motor (3A max)
xtra PWM
15 14 #1 #0
i2c addr
A4 A3 A2 A1 A0
Power
5-12V Motor Power
adafruit!
M1 GND M2
M3 GND M4
+5V +3V GND

Figure 2.4 Adafruit DC & Stepper Motor Hat

Before choosing this motor driver it was opted for L298D and A4988, however, both of the had similar problems. Software that was designed for them was not suitable for this particular project. The library of L298D didn’t required data of steps count. A4988 on the other side gave that value, however, in order to function both of them at the same time “threading” or “multiprocessing” had to be used. It was just unacceptable to use threading or multiprocessing libraries only for motion at this early stage.

Adafruit Motor Driver has an intuitive library that helps the development of the project. In more details, Adafruit Motor Driver can be used with 2 stepper motors at once and has 4 available options for the style of motion. This is a list of them:

- stepper.SINGLE (default) for a full step rotation to a position where one single coil is powered [6]
- stepper.DOUBLE for a full step rotation to position where two coils are powered providing more torque [6]
- stepper.INTERLEAVED for a half step rotation interleaving single and double coil positions and torque [6]
- stepper.MICROSTEP for a microstep rotation to a position where two coils are partially active. [6]

2.1.4 Inertial Measurement Unit

As discussed previously in the introduction, there were 2 important steps for accurate and precise localization and path planning. One of them was to understand vehicles belief regarding its pose in the given environment. This function can be acquired by using Inertial Measurement Unit (IMU). For this purpose, BNO055 was chosen (Figure 2.8). At first, it was tried to use MPU-6500, however, it was fairly unsuccessful. MPU6500 doesn’t have onboard processing unit. So, all calculations had to be conducted on raspberry (will be discussed later) by using threads. That was actually done, and yaw angle rotation’s measurement return with roughly flawless output with precise angle while motors were stall and rotation was conducted by hand. But, after stepper motors turned on and rotate the car on their own, tremendous amount of error occurred in the measurements of the MPU-6500. Investigation showed that, the magnetic field and the noise created by stepper motor can be powerful enough so that the MPU-6500 will create an error and even Kalman filter used in this experiment couldn’t help. As a result, for the further development decision was made for BNO055 which had its own processing unit onboard, mainly ARM Cortex-M0. BNO055 is 9-DOF sensor that supports sensor fusion. Sensor Fusion is used for combination of the raw data collected from accelerometer, magnetometer and gyroscope in order to output stable three-axis orientation.

This unit had support library that had a specific function for our needs, namely Euler Vector. By using this function yaw, roll and pitch angles were outputted. Moreover, this unit could work on its own and doesn't require raspberry to always loop measurement function. As a result, Euler vectors from the BNO055 unit were called only when they were required.

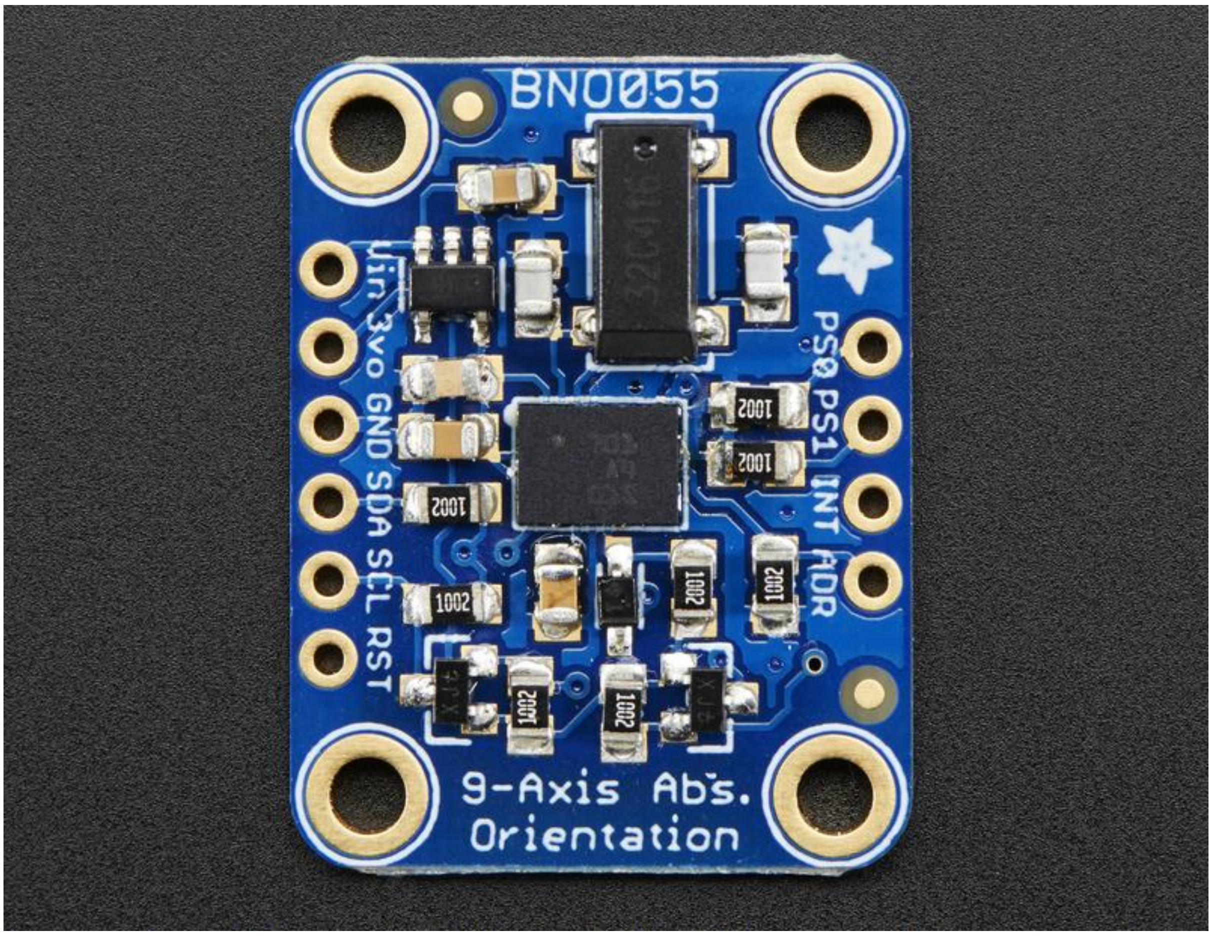


Figure 2.5 BNO055 Inertial Measurement Unit

2.1.5 Raspberry PI

To drive this variety of equipment, microcontroller is required. Raspberry PI (Figure 2.9) was chosen as the best option because it is using python as an Operating System (OS). That means that Raspberry PI is far more intuitive than other alternatives in terms of calculation the probability of path created.

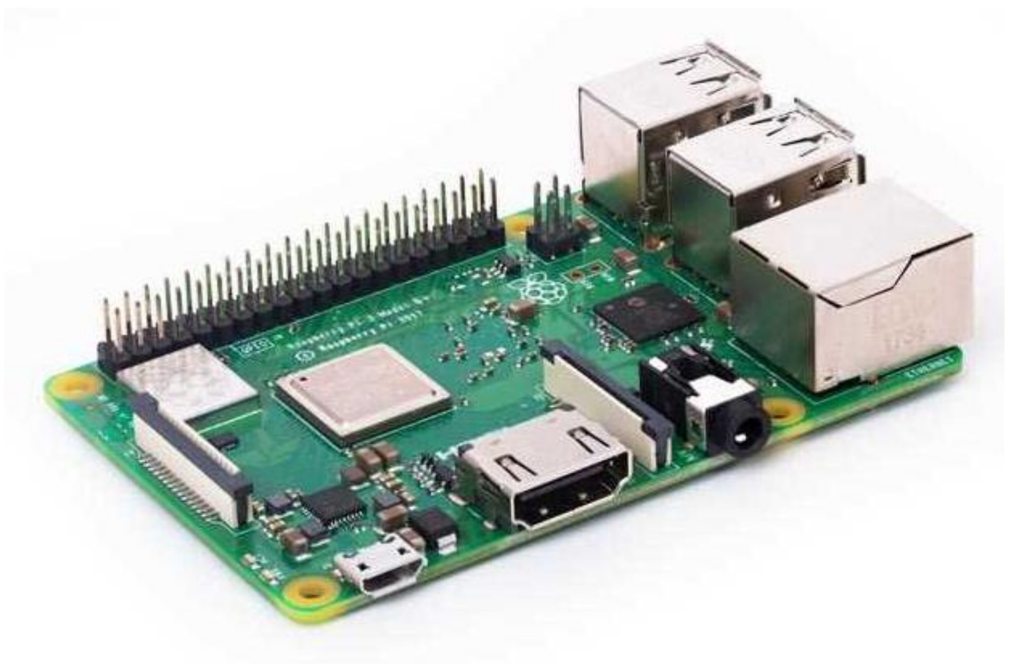
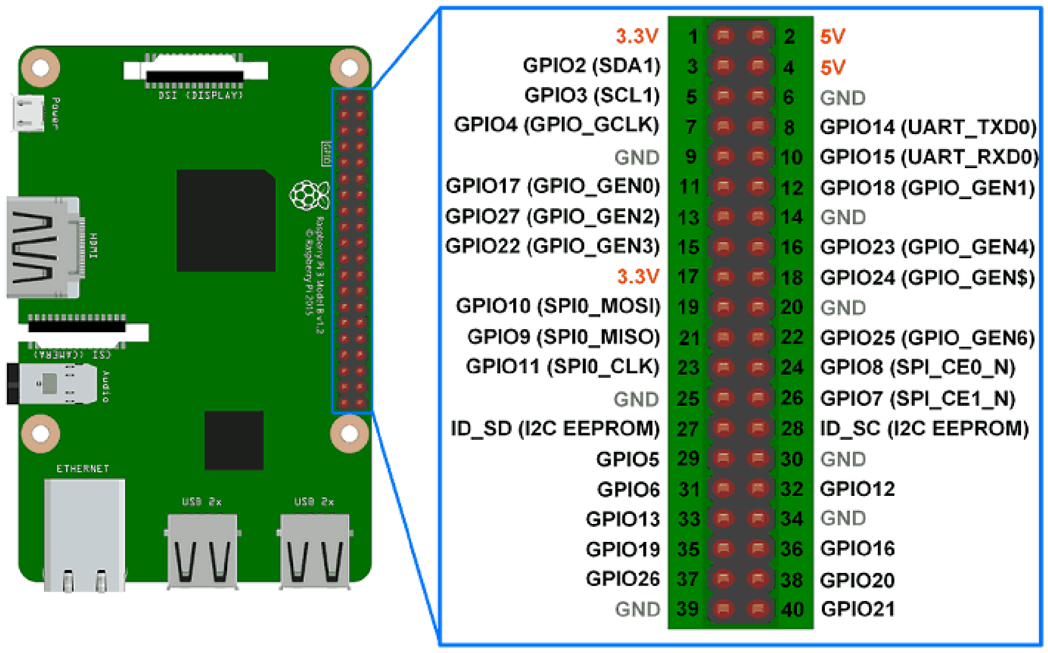


Figure 2.6 Raspberry PI

Raspberry PI has 40 available pins on it beside 4 USB ports, ethernet connection, 5V power supply port, HDMI and more. Pin layout of raspberry pi is given in the Figure 2.10.



DSI (DISPLAY)
Power
HDMI
CSI (CAMERA)
Audio
ETHERNET
USB 2x
USB 2x
GPIO
Raspberry Pi 3 Model B v1.2
© Raspberry Pi 2015
3.3V 1 2 5V
GPIO2 (SDA1) 3 4 5V
GPIO3 (SCL1) 5 6 GND
GPIO4 (GPIO_GCLK) 7 8 GPIO14 (UART_TXD0)
GND 9 10 GPIO15 (UART_RXD0)
GPIO17 (GPIO_GEN0) 11 12 GPIO18 (GPIO_GEN1)
GPIO27 (GPIO_GEN2) 13 14 GND
GPIO22 (GPIO_GEN3) 15 16 GPIO23 (GPIO_GEN4)
3.3V 17 18 GPIO24 (GPIO_GEN$)
GPIO10 (SPI0_MOSI) 19 20 GND
GPIO9 (SPI0_MISO) 21 22 GPIO25 (GPIO_GEN6)
GPIO11 (SPI0_CLK) 23 24 GPIO8 (SPI_CE0_N)
GND 25 26 GPIO7 (SPI_CE1_N)
ID_SD (I2C EEPROM) 27 28 ID_SC (I2C EEPROM)
GPIO5 29 30 GND
GPIO6 31 32 GPIO12
GPIO13 33 34 GND
GPIO19 35 36 GPIO16
GPIO26 37 38 GPIO20
GND 39 40 GPIO21

Figure 2.7 Raspberry PI pin layout

As it can be seen 2 5 Volt pins, 2 3.3 Volt and 8 GND pins are available on the computer. Pins that are left can be used in the project for transmitting and receiving DC or PWM signals. So that, by using this device 2 motor drivers, IMU and Ultrasonic Sensors at the same instance.

2.1.6 Power supply

In this project power was very important, as the lack of power given to the stepper motors may result in vibrations that will mislead the vehicle to the wrong path. In order to neglect this issue, DC-DC step-down power module was required. For this need, it was opted for LM2596 DC-DC Buck Converter Step-Down Power Module. This module (LM2596) was able to take DC 3V-40V input and convert it into 1.5V-35V. As it was already declared that stepper motor requires 2.7 V to operate, power module was set to this value. As an input for this power module, it was opted for Sony 18650 battery (Figure 2.11). Relatively small and powerful battery was more than enough for the power due of these motors. This battery is rated at 3.7V and 3000mAh. So as a result, 3.7 input was given to the power module and after the setting of the power module was made it resulted in 2.7V output.

Figure 2.8 18650 battery cell

In order to power the Raspberry it was opted for Anker powerbank as it was relatively small and was able to leave raspberry online for prolonged period. As it was determined this has to be more than enough for vehicle created.

18650 battery was connected to a single motor driver through dip switch. Also power bank was connected to Raspberry through dip switch (Figure 2.12).

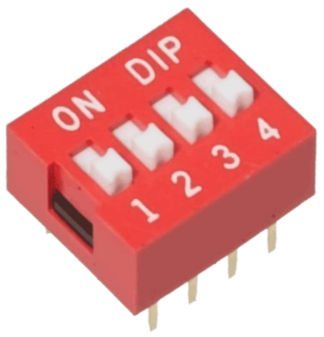


Figure 2.9 Dip switch with 4 pins

Discussed dip switch is very small, compact and have 4 switches on a single line. This type of switch helps to compact the whole wiring and even leaves space for more insurance if needed. One power bank is capable of delivering 5 V to Raspberry for more than 4 hours. Totally the power bank used is rated for 5000 mAh storage. Estimated working time of power bank depends strictly on the load given by Raspberry which is not always constant.

2.1.7 Modelling the vehicles structure

The vehicles basis was modeled (Figure 2.13) using “TinkerCAD” and “Solidworks”. “TinkerCAD” is very basic and intuitive software for creating and modifying 3D-models. The model itself was created by using measurements of the equipment used. Specific stands and holders are set with an exact measurement in order to make a ready product that can suit required equipment. Namely sockets for raspberry pi, ultrasonic sensors, motors and motor drivers were created.

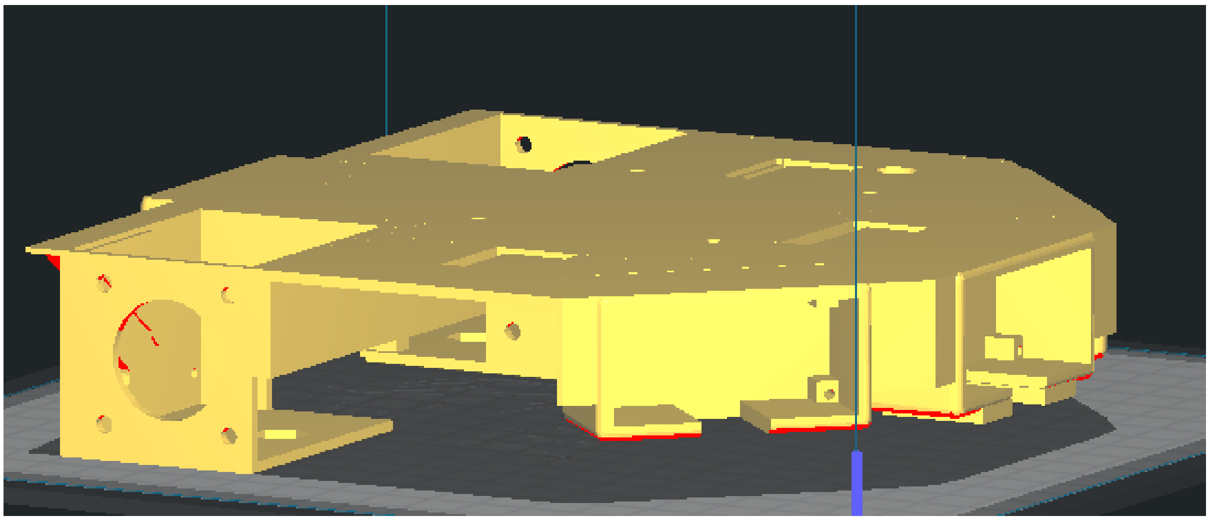


Figure 2.10 Side view of the vehicle model

The side view of the model is depicted in the Figure 2.14 and top view in Figure 2.15.

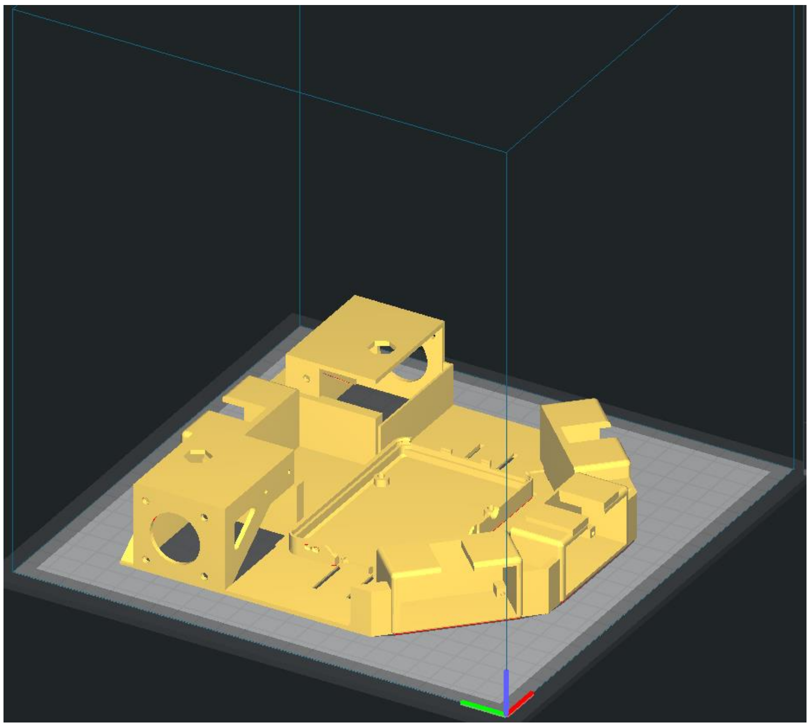


Figure 2.11 Car model from side view (model was 180 ° reversed)

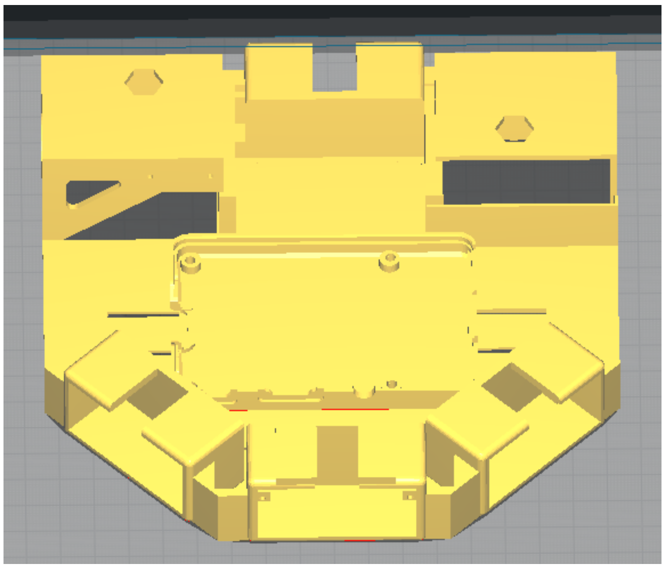


Figure 2.12 Car model from top view (model was 180 ° reversed)

As it can be seen in the Figure 2.15 and Figure 2.14 special sockets are placed for sensors, raspberry and stepper motors. According to these model ultrasonic sensors should be mounted by using 1.6 mm screws and M4 screws should be used for Raspberry. These models should create reliable structure that will protect the equipment from being unintentionally detached. Front panel of the model is designed so that ball caster will be mounted on top of the ultrasonic sensor socket by 2 M4 screws. These models were sliced by using CURA 4.4. This software helped to create a support and gcode material for 3D-Printer to print it. The model was reversed by 180 degrees in order to minimize the number of supports. As a result, estimated printing time was reduced to 6 hours from 24h.

2.2 Connections

2.2.1 Ultrasonic sensors cable connections

In order to test the equipment few tasks were performed. Firstly, the ultrasonic sensors were tested by determining the location of random obstacles. All connections were made by using wiring from Figure 2.16.

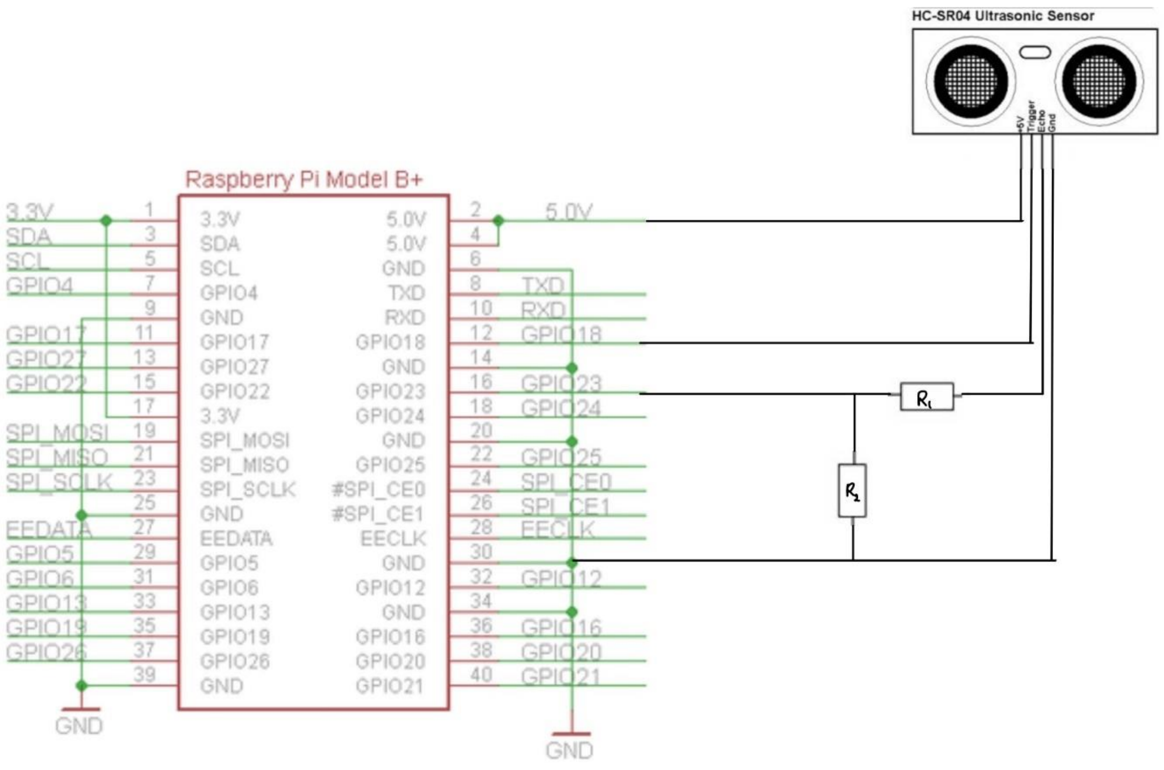


Figure 2.13 Connection between Raspberry PI and Ultrasonic Sensor

As it can be seen from the Figure 2.16 Ultrasonic 5V and GND pins are connected to the respected pins on the Raspberry. However, Trigger and Echo pins should be connected to GPIO (general-purpose input/output) pins instead. From the Figure 2.16 it is shown that trigger pin is connected to GPIO18 pin and Echo pin is connected to GPIO23 pin through 330 ohm resistor (R1) and with ground output from that wire through 470 ohm resistor which should result in noise reduction. Moreover, after this test a new task was given, to make 4 ultrasonic sensors work at the same instance (Figure 2.17).

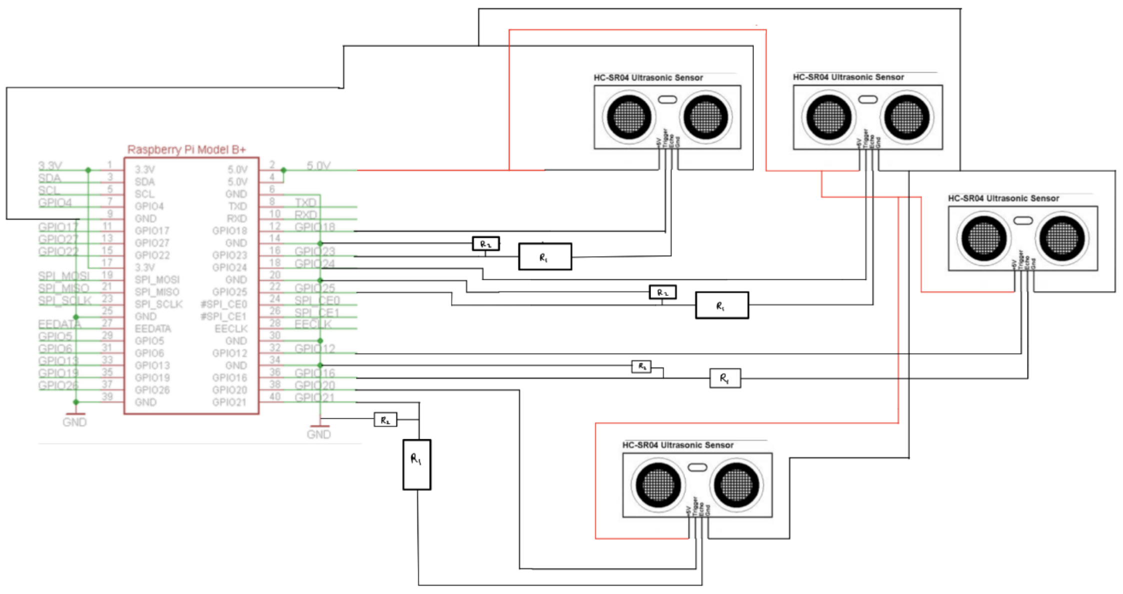


Figure 2.14 Four ultrasonic sensors connected to raspberry PI

2.2.2 Motors connections

After that the motor driver was tested with connected stepper motor (Figure 2.18). These tests were conducted in order to understand how the motors are working and how to command them. This motor hat is connected to the raspberry by using only SDA and SCL pins. So that, the chip handles the motor and speed controls over I2C and leaves space for any other devices to connect into I2C ports.

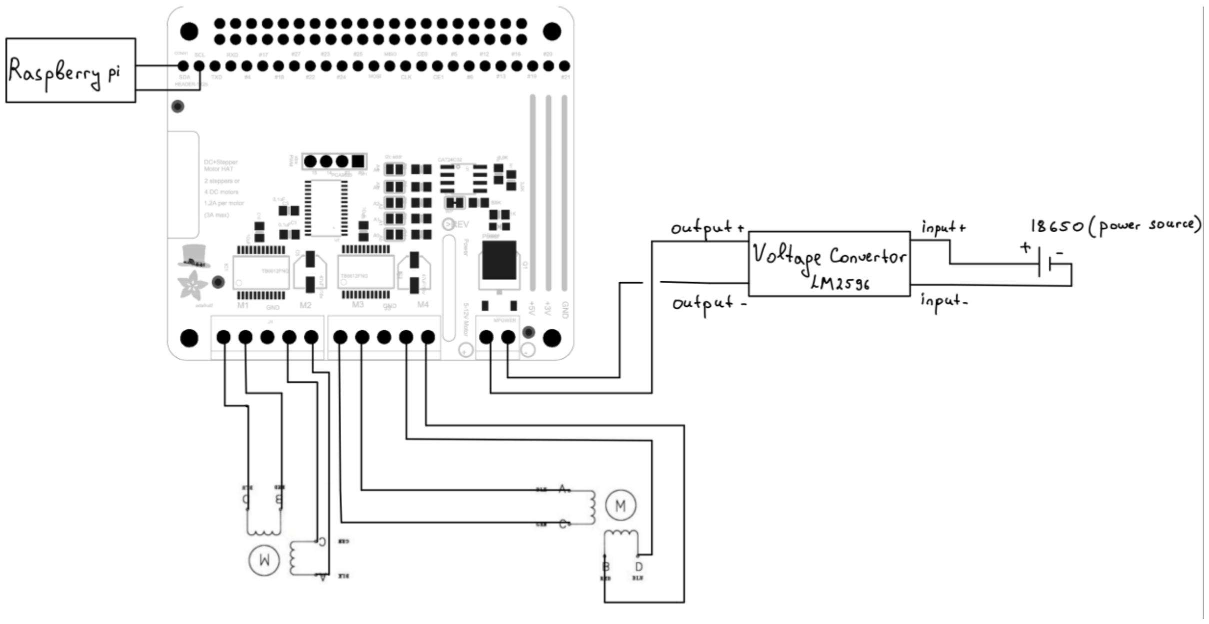


Figure 2.15 Adafruit motor hat connected to stepper motor and raspberry pi

As it can be seen in the Figure 2.18 the motor is divided into 2 coils and each pair of wires are connected to respective M1/M2 and M3/M4 ports of motor driver. After, 2.7V power is given into the power sockets of adafruit motor hat. Raspberry is connected to motor hat through SDA and SCL interface as it is shown in the Figure 2.18.

2.2.3 BNO055 Connections

For this project in order to neglect all issues that may appear because of motors, it was decided to communicate with BNO055 via UART interface. The connections made are given in the Figure 2.19 below.

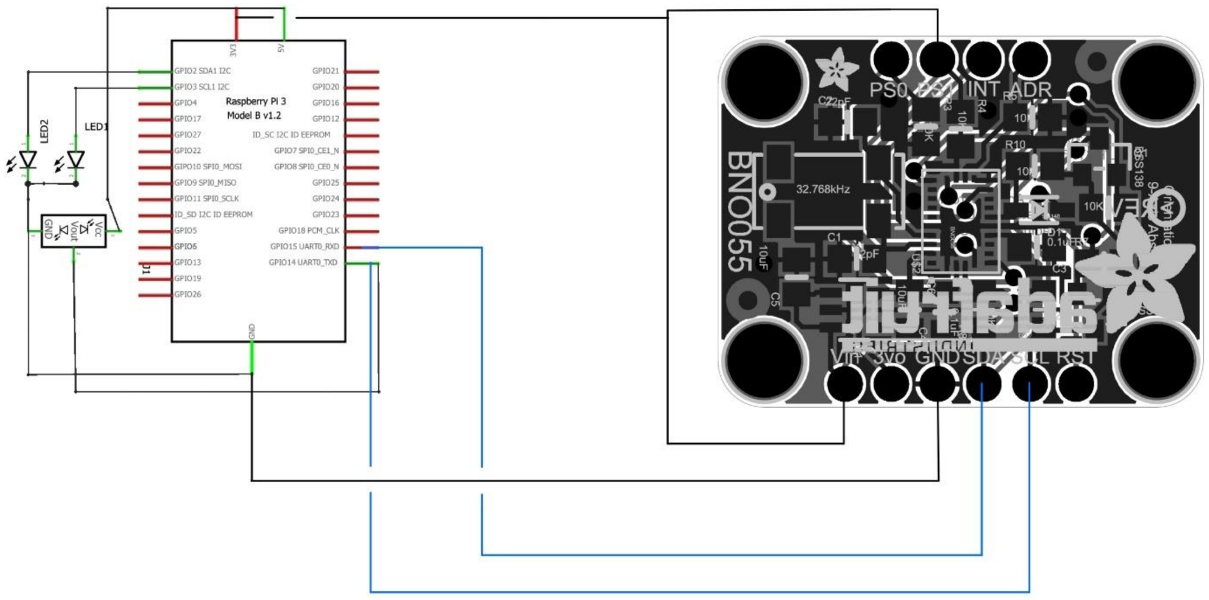


Figure 2.16 Circuit Diagram of BNO055 connected to raspberry pi

As it can be seen from the Figure 2.19, BNO055 is connected to 3.3V power output of raspberry. As it was decided to use UART mode, PS1 port is connected to Vin. This will result in activating UART mode in BNO055, so there will be no need in I2C interface.

3. SOFTWARE CODING

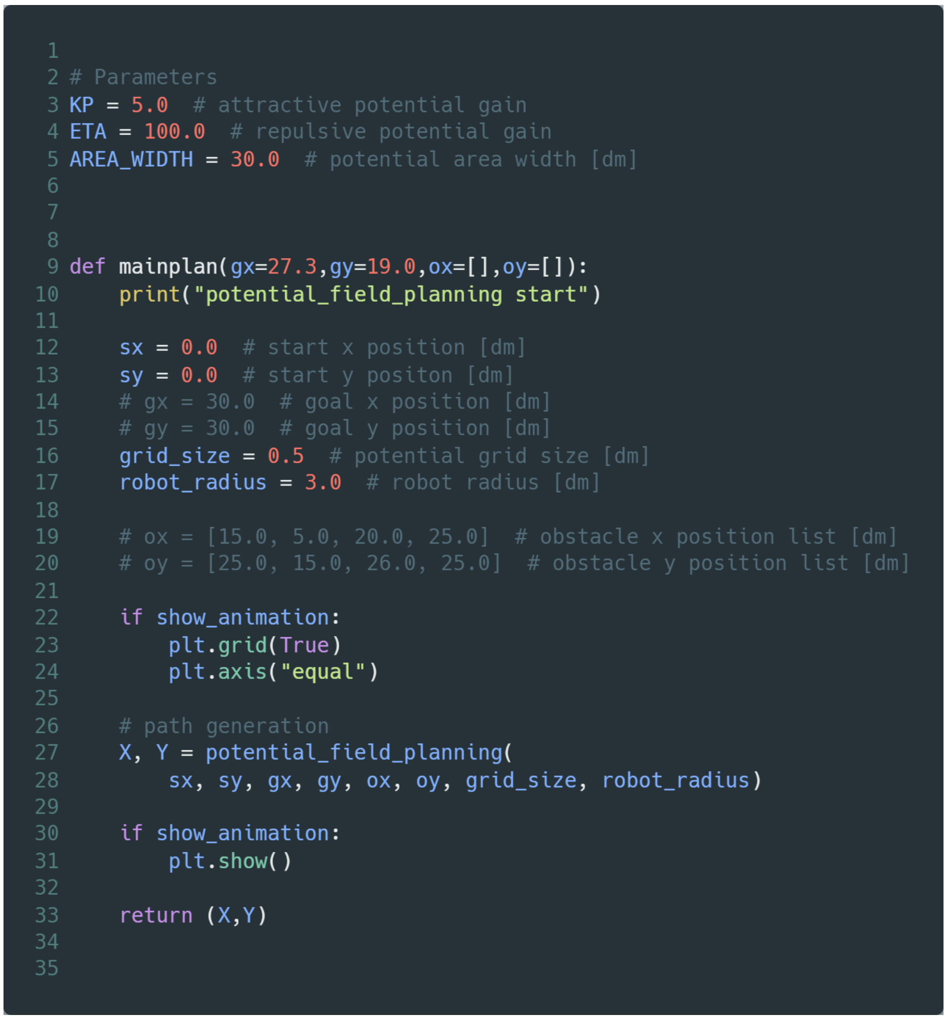
3.1 Path Planning

The aim of this part of code is to create a route to the goal coordinates while avoiding the obstacle, coordinates of which are already intentionally appended to the array. Logic behind this code is that we have 2 potential fields:

1) “Attractive” potential – move the goal.[7]

2) “Repulsive” potential – avoid obstacles.[7]

So, the vehicle should move closer to the attractive potential field and at the same instance it should repulse from the repulsive potential field. Following, if we have very high repulsive potential gain, vehicle will increase a radius by which it will repulse from the repulsive field. In our case repulsive field is an obstacle from which vehicle have to repulse. On the other side, the goal creates an attractive field to which vehicle have to move. The code of the path planning is given below in the Figure 3.0.



```

# Parameters
KP = 5.0  # attractive potential gain
ETA = 100.0  # repulsive potential gain
AREA_WIDTH = 30.0  # potential area width [dm]



def mainplan(gx=27.3,gy=19.0,ox=[],oy=[]):
    print("potential_field_planning start")

    sx = 0.0  # start x position [dm]
    sy = 0.0  # start y positon [dm]
    # gx = 30.0  # goal x position [dm]
    # gy = 30.0  # goal y position [dm]
    grid_size = 0.5  # potential grid size [dm]
    robot_radius = 3.0  # robot radius [dm]

    # ox = [15.0, 5.0, 20.0, 25.0]  # obstacle x position list [dm]
    # oy = [25.0, 15.0, 26.0, 25.0]  # obstacle y position list [dm]

    if show_animation:
        plt.grid(True)
        plt.axis("equal")

    # path generation
    X, Y = potential_field_planning(
        sx, sy, gx, gy, ox, oy, grid_size, robot_radius)

    if show_animation:
        plt.show()

    return (X,Y)


```

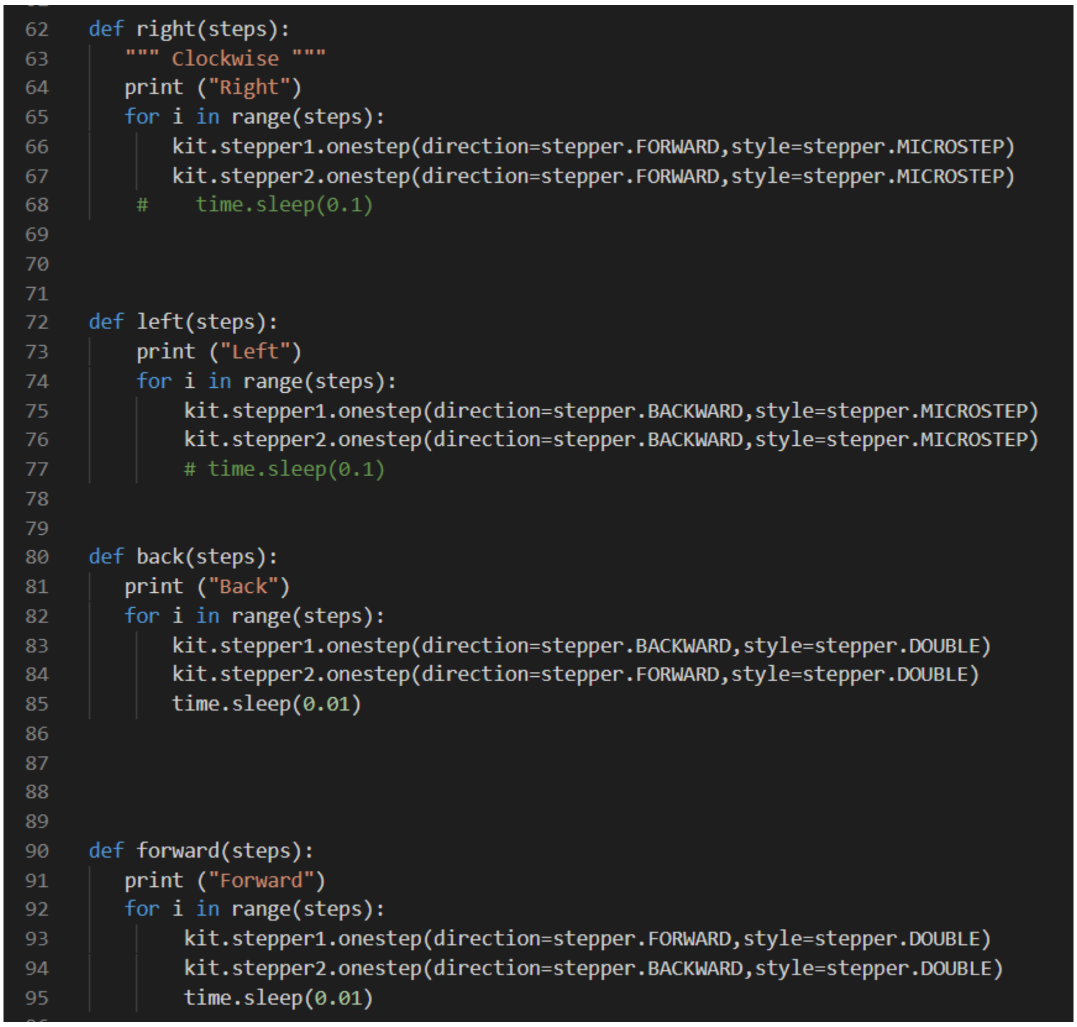
Figure 3.0 Path Planning main code

As you can see from the Figure 3.0, Attractive and Repulsive potential gains are declared at the beginning with AREA_WIDTH (Will be explained in chapter 4). Here mainplan function is defined, this function takes as arguments the goal points x and y coordinates, namely "gx" and "gy". Moreover, it has 2 arrays as arguments, these arrays are positions of an obstacle in both x and y coordinates, namely ox and oy. Lines 12 and 13 are declaring the starting position in x and y coordinates ("sx" and "sy"). Furthermore, in the line 16 and 17 the "grid_size" and "robot_radius" is set to 0.5 and 3.0. "grid_size" is the size of the grid to which the whole route will be divided into, very high value will result in the very big steps for the vehicle. "robot_radius" is the radius of the vehicle, from the measurements it was set to 3.0 dm (30 cm). In the lines 27 and 28 the "potential_field_planning" will proceed and return X and Y arrays at the end, which will be used soon to reach the goal. X and Y arrays show x and y coordinates of every step until the goal.

3.2 Main Functions

In this part the main part of code will be discussed.

Motor codes are depicted in the Figure 3.1:

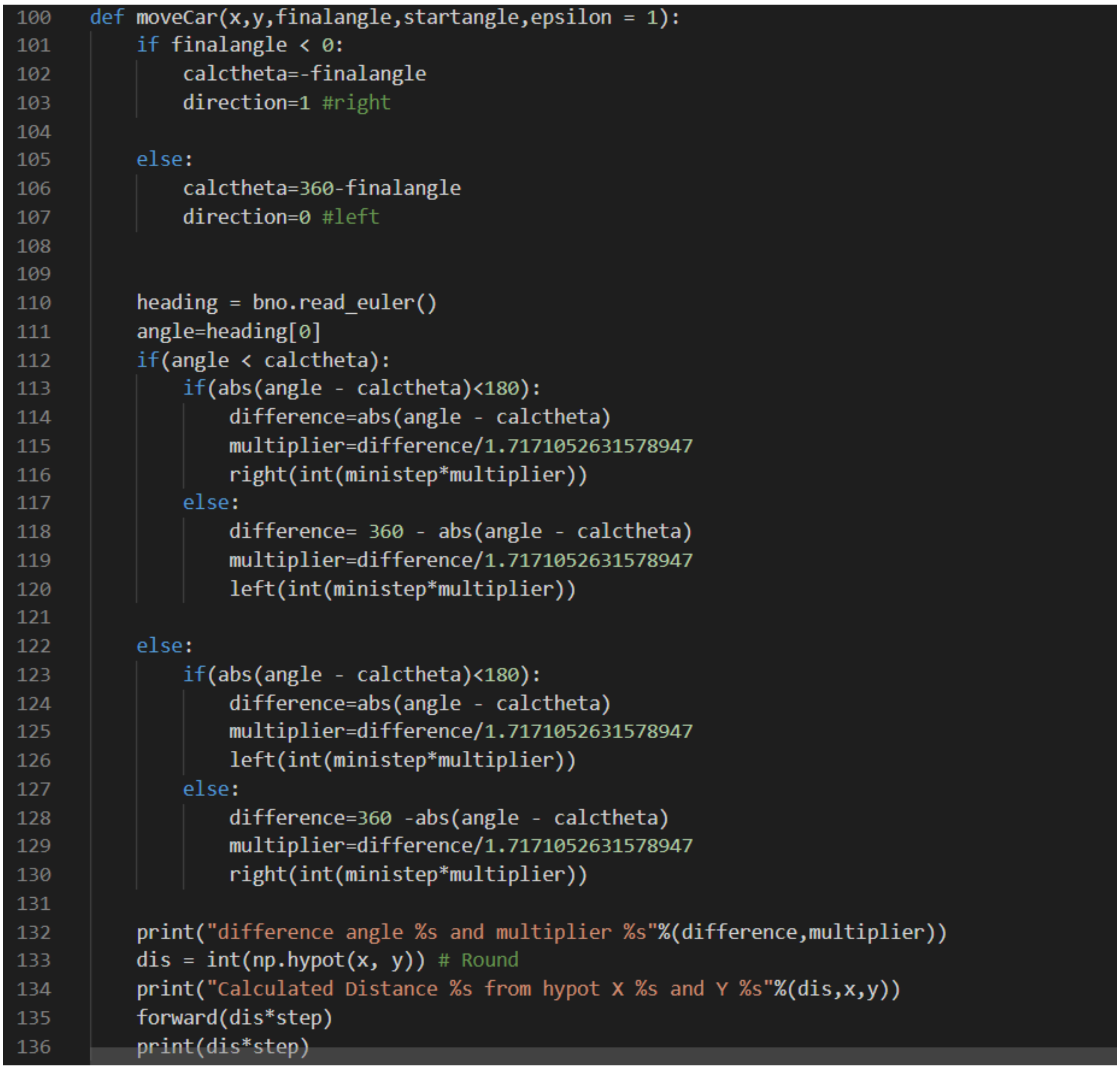


```python
def right(steps):
    """ Clockwise """
    print ("Right")
    for i in range(steps):
        kit.stepper1.onestep(direction=stepper.FORWARD,style=stepper.MICROSTEP)
        kit.stepper2.onestep(direction=stepper.FORWARD,style=stepper.MICROSTEP)
     #    time.sleep(0.1)



def left(steps):
     print ("Left")
     for i in range(steps):
         kit.stepper1.onestep(direction=stepper.BACKWARD,style=stepper.MICROSTEP)
         kit.stepper2.onestep(direction=stepper.BACKWARD,style=stepper.MICROSTEP)
         # time.sleep(0.1)


def back(steps):
    print ("Back")
    for i in range(steps):
        kit.stepper1.onestep(direction=stepper.BACKWARD,style=stepper.DOUBLE)
        kit.stepper2.onestep(direction=stepper.FORWARD,style=stepper.DOUBLE)
        time.sleep(0.01)




def forward(steps):
    print ("Forward")
    for i in range(steps):
         kit.stepper1.onestep(direction=stepper.FORWARD,style=stepper.DOUBLE)
         kit.stepper2.onestep(direction=stepper.BACKWARD,style=stepper.DOUBLE)
         time.sleep(0.01)
```

Figure 3.1 Motor Codes

In this Figure 3.1 functions are defined: right, left, forward and back. These functions print the direction of the movement and move the car in the name direction by creating a “for” loop, so that both motors are driven in the direction needed. “onestep” function written in the code takes 2 parameters: direction and style. Direction can be “forward” and “backward”. The style is assigned to one of 4 available styles (“Single Steps”, “Double Steps”, “Interleaved Steps”, “Microstepping”) provided from the manufacturer of the motor driver. Style “double” uses more power than “single” but is stronger. The functions “right”, “left”, “forward” and “back” can take parameter “steps”. That will loop the motors exactly the number “steps” is giving to the function. In the “left” and “right” functions “MICROSTEP” style is used, as it will maximize the preciseness of the rotational movement.

The next part of code is moving the vehicle to a certain distance and find the most suitable angle based on the calculations and measurements:

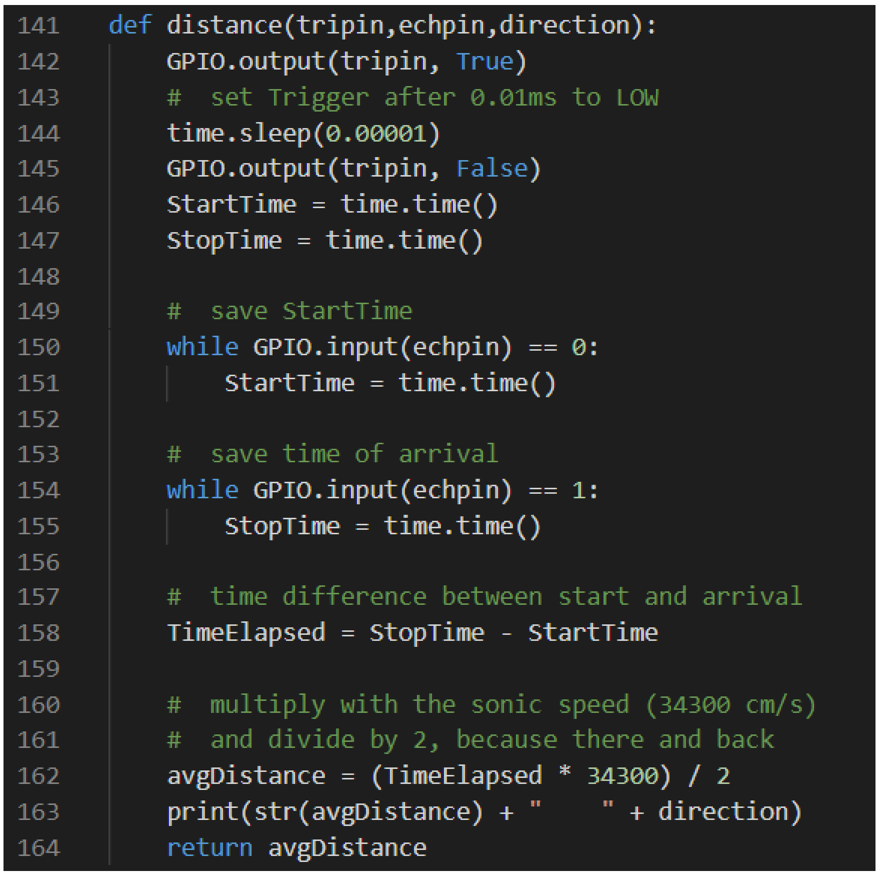


```python
def moveCar(x,y,finalangle,startangle,epsilon = 1):
    if finalangle < 0:
        calctheta=-finalangle
        direction=1 #right

    else:
        calctheta=360-finalangle
        direction=0 #left


    heading = bno.read_euler()
    angle=heading[0]
    if(angle < calctheta):
        if(abs(angle - calctheta)<180):
            difference=abs(angle - calctheta)
            multiplier=difference/1.7171052631578947
            right(int(ministep*multiplier))
        else:
            difference= 360 - abs(angle - calctheta)
            multiplier=difference/1.7171052631578947
            left(int(ministep*multiplier))

    else:
        if(abs(angle - calctheta)<180):
            difference=abs(angle - calctheta)
            multiplier=difference/1.7171052631578947
            left(int(ministep*multiplier))
        else:
            difference=360 -abs(angle - calctheta)
            multiplier=difference/1.7171052631578947
            right(int(ministep*multiplier))

    print("difference angle %s and multiplier %s"%(difference,multiplier))
    dis = int(np.hypot(x, y)) # Round
    print("Calculated Distance %s from hypot X %s and Y %s"%(dis,x,y))
    forward(dis*step)
    print(dis*step)
```

Figure 3.2 Vehicle moving function

The code written in the Figure 3.2 will move a vehicle to a certain distance and try to adjust the angle to the calculated one based on the measured angle from BNO005. In details, the arguments that the “moveCar” function have are x / y coordinates and “finalangle” / “startangle”. Lines from 101 until 107 will change the range of the required angle (“finalangle”) from the [-pi,pi] to [0,360]. Then, lines 110 and 111 will check the BNO005 measurements and then assign “angle” to the yaw angle measured at that instance. Afterwards, the car will turn left or right to a certain degree based on the if statements declared in the lines 112 and 130. As it was measured by BNO005 “1.7171052631578947” is the degree angle by which vehicle is rotating on the 48 steps (“ministep”) in “MICROSTEP” style. So, by dividing the difference of angles to this value the certain multiplier will be calculated, this multiplier can achieve better preciseness while finding the calculated angle by using BNO005 (Measured) angle. In the lines 133 and 135 the variable “dis” is attached to the hypothenuses of x and y in order to find the distance to these coordinates. After finding the distance to them, the “function” is called by taking the arguments distance multiplied by step. (1 step = 1 cm)

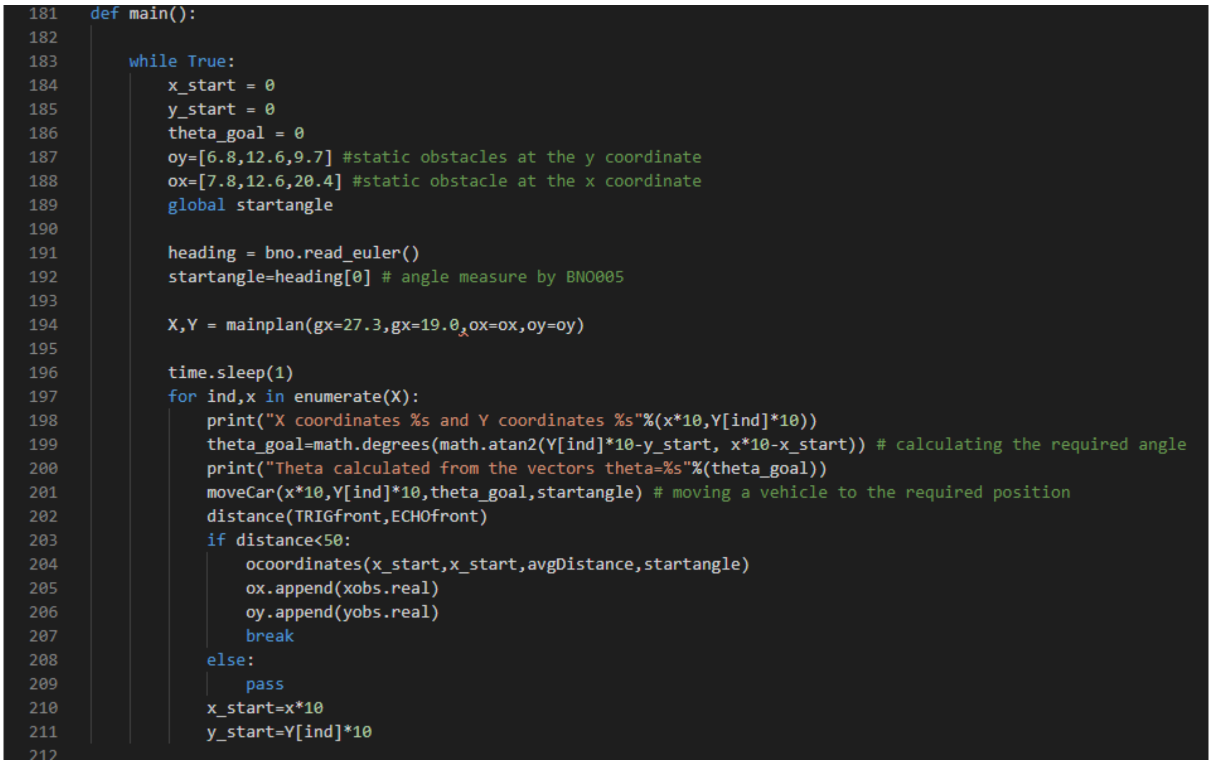
In the Figure 3.3 the “distance” function is depicted, the purpose of this function is to take as an arguments trigger and echo pins of sonar sensors and return the average distance to them.



```python
def distance(tripin,echpin,direction):
    GPIO.output(tripin, True)
    #  set Trigger after 0.01ms to LOW
    time.sleep(0.00001)
    GPIO.output(tripin, False)
    StartTime = time.time()
    StopTime = time.time()

    #  save StartTime
    while GPIO.input(echpin) == 0:
        StartTime = time.time()

    #  save time of arrival
    while GPIO.input(echpin) == 1:
        StopTime = time.time()

    #  time difference between start and arrival
    TimeElapsed = StopTime - StartTime

    #  multiply with the sonic speed (34300 cm/s)
    #  and divide by 2, because there and back
    avgDistance = (TimeElapsed * 34300) / 2
    print(str(avgDistance) + "     " + direction)
    return avgDistance
```

Figure 3.3 Distance function

The main function is given below in the Figure 3.4:



```
181  def main():
182  
183      while True:
184          x_start = 0
185          y_start = 0
186          theta_goal = 0
187          oy=[6.8,12.6,9.7] #static obstacles at the y coordinate
188          ox=[7.8,12.6,20.4] #static obstacle at the x coordinate
189          global startangle
190  
191          heading = bno.read_euler()
192          startangle=heading[0] # angle measure by BNO005
193  
194          X,Y = mainplan(gx=27.3,gx=19.0,ox=ox,oy=oy)
195  
196          time.sleep(1)
197          for ind,x in enumerate(X):
198              print("X coordinates %s and Y coordinates %s"%(x*10,Y[ind]*10))
199              theta_goal=math.degrees(math.atan2(Y[ind]*10-y_start, x*10-x_start)) # calculating the required angle
200              print("Theta calculated from the vectors theta=%s"%(theta_goal))
201              moveCar(x*10,Y[ind]*10,theta_goal,startangle) # moving a vehicle to the required position
202              distance(TRIGfront,ECHOfront)
203              if distance<50:
204                  ocoordinates(x_start,x_start,avgDistance,startangle)
205                  ox.append(xobs.real)
206                  oy.append(yobs.real)
207                  break
208              else:
209                  pass
210              x_start=x*10
211              y_start=Y[ind]*10
```

Figure 3.4 Main function

The purpose of the "main" function is to repeatedly restart "mainplan" function of potential field planning and append dynamic obstacles to the list of the static obstacles in the "While" loop. Moreover, in the line 201 "MoveCar" function is called to move the vehicle. The arguments of "MoveCar" functions are coming from the angle at the goal calculation, namely line 199 "theta_goal". Line 199 calculates the angle at to the goal in degrees by using arctan of the difference between the coordinates. At the lines 202 to 209 the distance function is called and the information about the dynamic obstacles is calculated then appended to the list of the static obstacles by using method "append". Appended coordinates are calculated in the "ocoordinate" function that gives the real values of the obstacle coordinates. This is done by the following equations:

$$ox = \sqrt{d^2 - \frac{d^2}{1 + \frac{1}{tan^2 \alpha}}} + x$$

Equation (3.0)

Where,

ox = x coordinate of the detected obstacle

d = distance to the obstacle

α = angle of hypothenuses between starting x and obstacle x coordinate

x = x coordinate at start

$$o\mathrm{y} = \sqrt{\frac{d^2}{1 + \frac{1}{\tan^2 \alpha}}} + y$$

Equation (3.1)

Where,

oy = y coordinate of the detected obstacle

d = distance to the obstacle

α = angle of hypothenuses between starting y and obstacle y coordinate

y = y coordinate at start

4. TESTING AND RESULTS

For the testing part a small area was prepared. The area was 273 cm width and 190 cm height. In the code it was written as x =273 and y = 190. So the main task of the vehicle was to go from the starting point x=0,y=0 to the point x=273,y=190. Moreover, in order to do so the vehicle was supposed to avoid all static and dynamic obstacles on the path created by the path planning function. For localization the map had to be created in order to show the localization of the car at the moment. Figure 4.0 depicts the area created for this experiment:

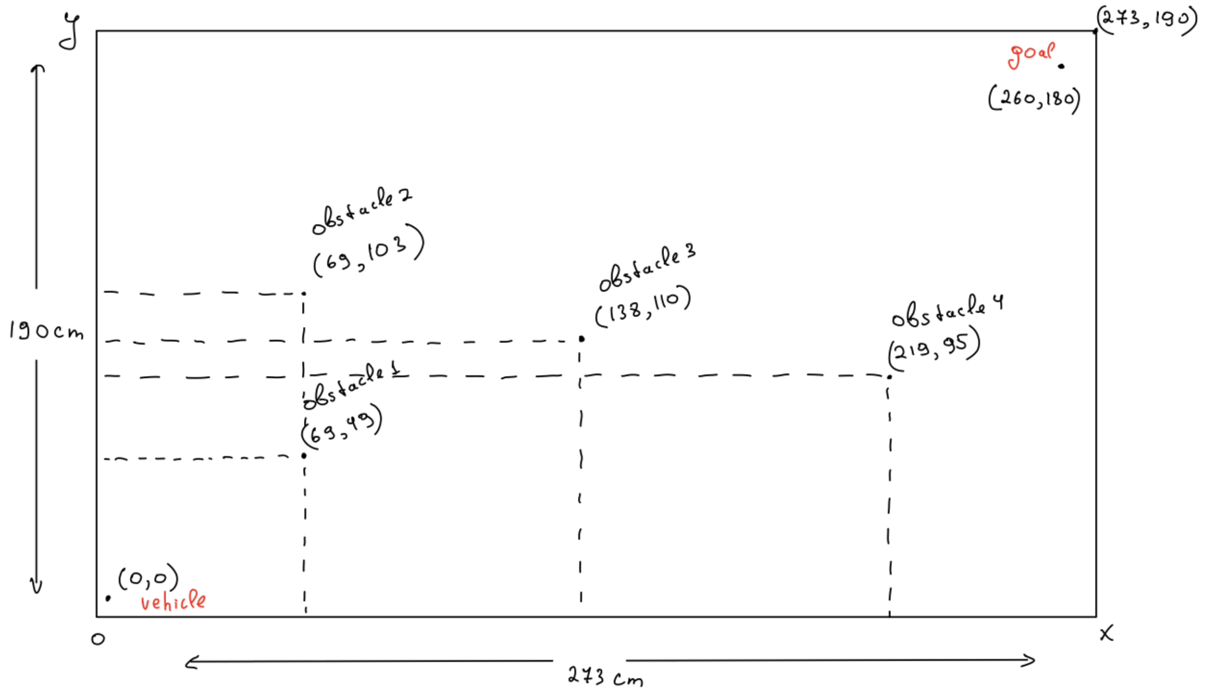


Figure 4.0 Raw draft of the experiment area

Figure above depicts the draft of the area where the experiment was conducted. In this experiment there were 4 obstacles. 3 of them were static and were attached to the "ox" and "oy" arrays. However, the obstacle 4 was dynamic and was not assigned to the array intentionally. To sum up the coordinates of the obstacles were:

1. Static obstacle at coordinates (69,49)
2. Static obstacle at coordinates (69,103)
3. Static obstacle at coordinates (138,110)
4. Dynamic obstacle at coordinates (219,95)

After, conducting the experiment it was understood that the vehicle has certain difficulties with dynamic obstacles eventhough the vehicle passed the test and avoided all obstacles. Few attempts revealed that the vehicle is still not as precise as it should be. Unfortunately, it is not possible for now to the vehicle more detailly.

In the Figure 4.1 the coordinate system created by the vehicle is demonstrated.

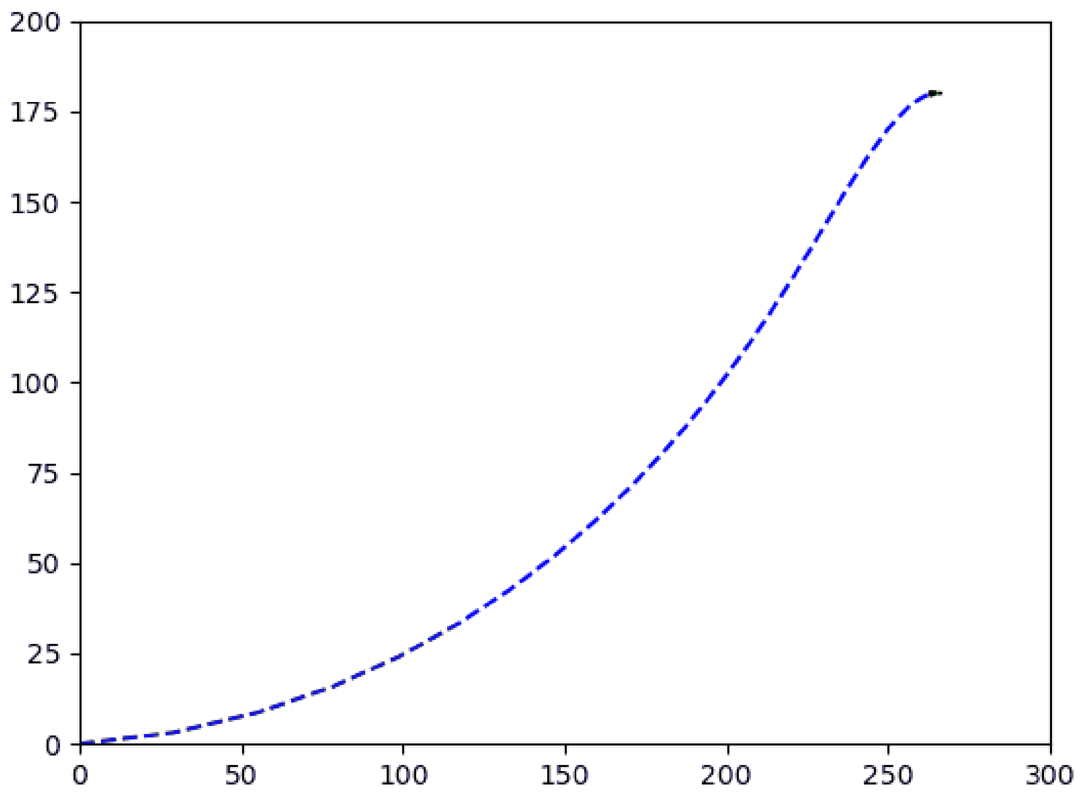
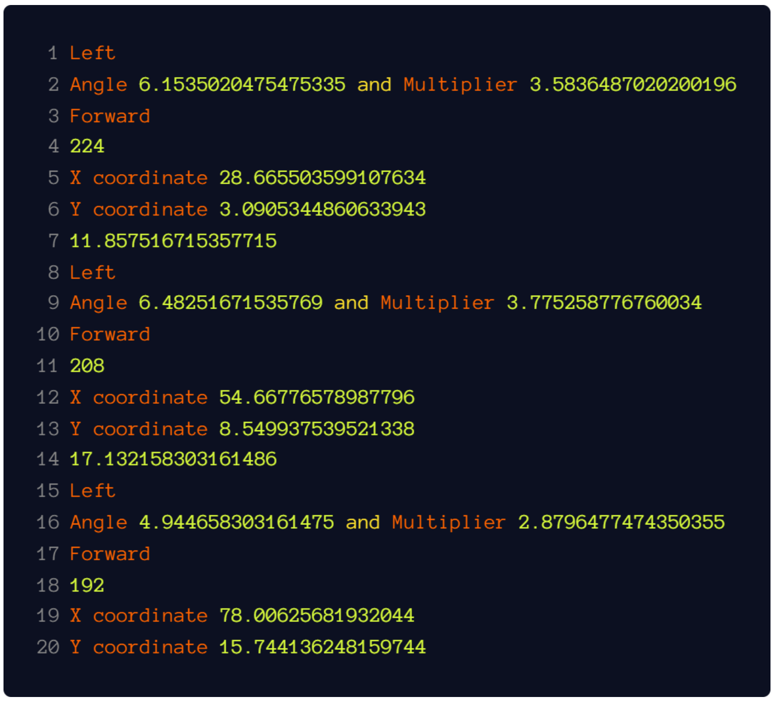


Figure 4.1 Coordinate system map created by the vehicle

The figure above shows the map created by the vehicle while movement. This is the basis of localisation done in this project. Moreover, the code prints the X and Y coordinates between each manoeuvre. Figure below illustrates the part of the outputs taken during the experiments.



```
Left
Angle 6.1535020475475335 and Multiplier 3.5836487020200196
Forward
224
X coordinate 28.665503599107634
Y coordinate 3.0905344860633943
11.857516715357715
Left
Angle 6.48251671535769 and Multiplier 3.775258776760034
Forward
208
X coordinate 54.66776578987796
Y coordinate 8.549937539521338
17.132158303161486
Left
Angle 4.944658303161475 and Multiplier 2.8796477474350355
Forward
192
X coordinate 78.00625681932044
Y coordinate 15.744136248159744
```

Figure 4.2 Output during experiment

The Figure 4.2 briefly shows a small part of all outputs taken during the experiments. As it can be seen, at first, the vehicle turns left by 6 angles and then moves forward, until the movement is over. When the movement is over the vehicle starts to check for further goal coordinates and the prints them, as it can be seen in the lines 5 and 6. After vehicle checks angle, rectifiers own angle and move to the next coordinates. These loop continues until the vehicle reaches the goal coordinates. Static obstacles are already avoided by the path created by the vehicle.

5. CONCLUSION

The aim of the project was to:

1. Create shortest path by probabalistic model that avoids static obstacles,
2. Localise the car at any instance,
3. Avoid dynamic obstacles.

The localization and path planning was done, however, the avoidance of the dynamic obstacles is not as precise as it should be. The testing of the vehicle could be done in more precise way, the localization's perfection is still obscure, more detailed tests are required.

As it was already concluded in the previous paper, the hardware changes was required. Required stages were conducted in order to rectify that drawbacks motor driver changed, power of the circuit reorganized and IMU replaced. That helped for the hardware part. As a summary, the project is not perfect and still has its own drawbacks, there is still work to be done with this project.

REFERENCES

[1] Wang, M. 2009. Mobile Robot Localization and Path Planning using an Omnidirectional Camera and Infrared Sensors, IEEE International Conference on Systems, Man, and Cybernetics, October, National Chung Cheng University, Chiayi 621, Taiwan.

[2] Pop, C.M. and Mogan, G.L. and Neagu, M. 2015. Localization and Path Planning for an Autonomous Mobile Robot Equipped with Sonar Sensor. Applied Mechanics and Materials, Vol. 772, pp 494-499.

[3] Anonymous. 2019. Web Site: https://www.invensense.com, Accessed On:12.01.2020.

[4] Anonymous. 2019. Web Site: https://www.pololu.com, Accessed On: 05.01.2020.

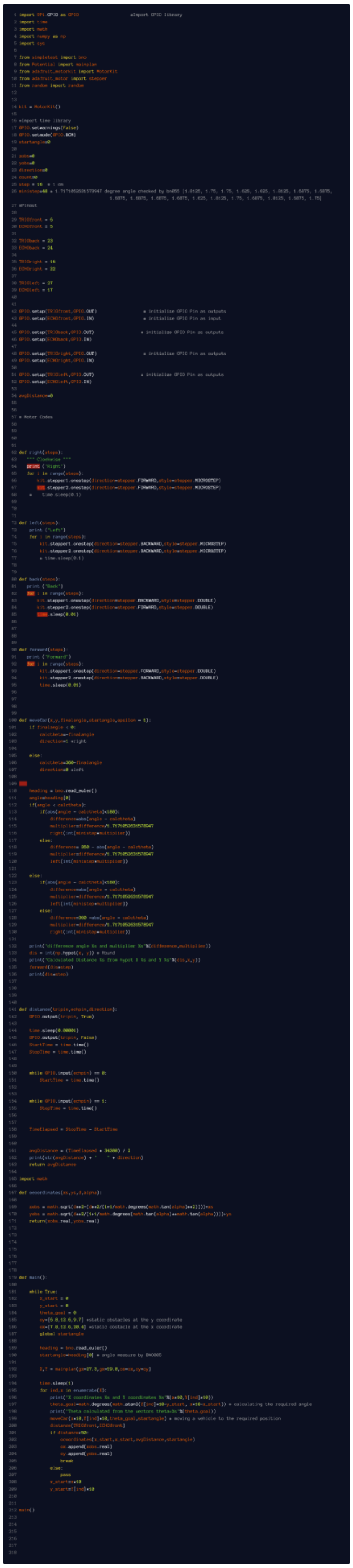
[5] Anonymous. 2019. Web Site: http://www.senith.lk, Accessed On: 12.01.2020.

[6] Anonymous. 2020. Web Site: https://learn.adafruit.com/, Accessed On: 14.06.2020.

[7] Howie Choset,Ji Yeong Lee,G.D. Hager and Z. Dodds () Robotic Motion Planning: Potential Functions, Available at: https://www.cs.cmu.edu/~motionplanning/lecture/Chap4-Potential-Field_howie.pdf (Accessed On: 14.06.2020).

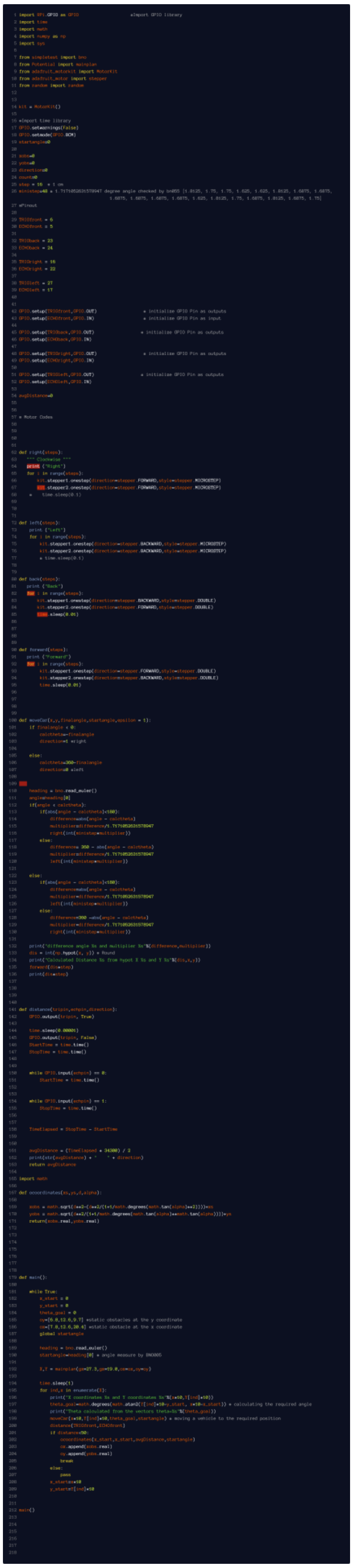
APPENDIX

Mainfunction.py



```python
import RPi.GPIO as GPIO                    #import GPIO library
import time
import math
import numpy as np
import sys

from simpletest import bno
from Potential import mainplan
from adafruit_motorkit import MotorKit
from adafruit_motor import stepper
from random import random


kit = MotorKit()

#import time library
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
startangle=0

xobs=0
yobs=0
direction=0
count=0
step = 16  # 1 cm
ministep=40 # 1.7171052631578947 degree angle checked by bn055 [1.8125, 1.75, 1.75, 1.625, 1.625, 1.8125, 1.6875, 1.6875,
                                    1.6875, 1.6875, 1.6875, 1.6875, 1.625, 1.8125, 1.75, 1.6875, 1.8125, 1.6875, 1.75]
#Pinout

TRIGfront = 6
ECHOfront = 5

TRIGback = 23
ECHOback = 24

TRIGright = 16
ECHOright = 22

TRIGleft = 27
ECHOleft = 17


GPIO.setup(TRIGfront,GPIO.OUT)                   # initialize GPIO Pin as outputs
GPIO.setup(ECHOfront,GPIO.IN)                    # initialize GPIO Pin as input

GPIO.setup(TRIGback,GPIO.OUT)                   # initialize GPIO Pin as outputs
GPIO.setup(ECHOback,GPIO.IN)

GPIO.setup(TRIGright,GPIO.OUT)                   # initialize GPIO Pin as outputs
GPIO.setup(ECHOright,GPIO.IN)

GPIO.setup(TRIGleft,GPIO.OUT)                   # initialize GPIO Pin as outputs
GPIO.setup(ECHOleft,GPIO.IN)

avgDistance=0


# Motor Codes




def right(steps):
    """ Clockwise """
    print ("Right")
    for i in range(steps):
        kit.stepper1.onestep(direction=stepper.FORWARD,style=stepper.MICROSTEP)
        kit.stepper2.onestep(direction=stepper.FORWARD,style=stepper.MICROSTEP)
     #    time.sleep(0.1)



def left(steps):
     print ("Left")
     for i in range(steps):
         kit.stepper1.onestep(direction=stepper.BACKWARD,style=stepper.MICROSTEP)
         kit.stepper2.onestep(direction=stepper.BACKWARD,style=stepper.MICROSTEP)
         # time.sleep(0.1)


def back(steps):
    print ("Back")
    for i in range(steps):
        kit.stepper1.onestep(direction=stepper.BACKWARD,style=stepper.DOUBLE)
        kit.stepper2.onestep(direction=stepper.FORWARD,style=stepper.DOUBLE)
        time.sleep(0.01)




def forward(steps):
    print ("Forward")
    for i in range(steps):
         kit.stepper1.onestep(direction=stepper.FORWARD,style=stepper.DOUBLE)
         kit.stepper2.onestep(direction=stepper.BACKWARD,style=stepper.DOUBLE)
         time.sleep(0.01)




def moveCar(x,y,finalangle,startangle,epsilon = 1):
    if finalangle < 0:
        calctheta=-finalangle
        direction=1 #right

    else:
        calctheta=360-finalangle
        direction=0 #left


    heading = bno.read_euler()
    angle=heading[0]
    if(angle < calctheta):
        if(abs(angle - calctheta)<180):
            difference=abs(angle - calctheta)
            multiplier=difference/1.7171052631578947
            right(int(ministep*multiplier))
        else:
            difference= 360 - abs(angle - calctheta)
            multiplier=difference/1.7171052631578947
            left(int(ministep*multiplier))

    else:
        if(abs(angle - calctheta)<180):
            difference=abs(angle - calctheta)
            multiplier=difference/1.7171052631578947
            left(int(ministep*multiplier))
        else:
            difference=360 -abs(angle - calctheta)
            multiplier=difference/1.7171052631578947
            right(int(ministep*multiplier))

    print("difference angle %s and multiplier %s"%(difference,multiplier))
    dis = int(np.hypot(x, y)) # Round
    print("Calculated Distance %s from hypot X %s and Y %s"%(dis,x,y))
    forward(dis*step)
    print(dis*step)




def distance(tripin,echpin,direction):
    GPIO.output(tripin, True)

    time.sleep(0.00001)
    GPIO.output(tripin, False)
    StartTime = time.time()
    StopTime = time.time()


    while GPIO.input(echpin) == 0:
        StartTime = time.time()


    while GPIO.input(echpin) == 1:
        StopTime = time.time()


    TimeElapsed = StopTime - StartTime


    avgDistance = (TimeElapsed * 34300) / 2
    print(str(avgDistance) + "    " + direction)
    return avgDistance

import math

def ocoordinates(xs,ys,d,alpha):

    xobs = math.sqrt(d**2-(d**2/(1+1/math.degrees(math.tan(alpha)**2))))+xs
    yobs = math.sqrt(d**2/(1+1/math.degrees(math.tan(alpha)*math.tan(alpha))))+ys
    return(xobs.real,yobs.real)







def main():

    while True:
        x_start = 0
        y_start = 0
        theta_goal = 0
        oy=[6.8,12.6,9.7] #static obstacles at the y coordinate
        ox=[7.8,12.6,20.4] #static obstacle at the x coordinate
        global startangle

        heading = bno.read_euler()
        startangle=heading[0] # angle measure by BNO005

        X,Y = mainplan(gx=27.3,gy=19.0,ox=ox,oy=oy)

        time.sleep(1)
        for ind,x in enumerate(X):
            print("X coordinates %s and Y coordinates %s"%(x*10,Y[ind]*10))
            theta_goal=math.degrees(math.atan2(Y[ind]*10-y_start, x*10-x_start)) # calculating the required angle
            print("Theta calculated from the vectors theta=%s"%(theta_goal))
            moveCar(x*10,Y[ind]*10,theta_goal,startangle) # moving a vehicle to the required position
            distance(TRIGfront,ECHOfront)
            if distance<50:
                ocoordinates(x_start,x_start,avgDistance,startangle)
                ox.append(xobs.real)
                oy.append(yobs.real)
                break
            else:
                pass
            x_start=x*10
            y_start=Y[ind]*10


main()
```

Pathplanning,py



```python
import RPi.GPIO as GPIO                    #import GPIO library
import time
import math
import numpy as np
import sys

from simpletest import bno
from Potential import mainplan
from adafruit_motorkit import MotorKit
from adafruit_motor import stepper
from random import random


kit = MotorKit()

#import time library
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
startangle=0

xobs=0
yobs=0
direction=0
count=0
step = 16  # 1 cm
ministep=40 # 1.7171052631578947 degree angle checked by bn055 [1.8125, 1.75, 1.75, 1.625, 1.625, 1.8125, 1.6875, 1.6875, 1.6875, 1.6875, 1.6875, 1.6875, 1.625, 1.8125, 1.75, 1.6875, 1.8125, 1.6875, 1.75]
#Pinout

TRIGfront = 6
ECHOfront = 5

TRIGback = 23
ECHOback = 24

TRIGright = 16
ECHOright = 22

TRIGleft = 27
ECHOleft = 17


GPIO.setup(TRIGfront,GPIO.OUT)                  # initialize GPIO Pin as outputs
GPIO.setup(ECHOfront,GPIO.IN)                   # initialize GPIO Pin as input

GPIO.setup(TRIGback,GPIO.OUT)                  # initialize GPIO Pin as outputs
GPIO.setup(ECHOback,GPIO.IN)

GPIO.setup(TRIGright,GPIO.OUT)                  # initialize GPIO Pin as outputs
GPIO.setup(ECHOright,GPIO.IN)

GPIO.setup(TRIGleft,GPIO.OUT)                  # initialize GPIO Pin as outputs
GPIO.setup(ECHOleft,GPIO.IN)

avgDistance=0


# Motor Codes




def right(steps):
   """ Clockwise """
   print ("Right")
   for i in range(steps):
       kit.stepper1.onestep(direction=stepper.FORWARD,style=stepper.MICROSTEP)
       kit.stepper2.onestep(direction=stepper.FORWARD,style=stepper.MICROSTEP)
    #    time.sleep(0.1)



def left(steps):
    print ("Left")
    for i in range(steps):
        kit.stepper1.onestep(direction=stepper.BACKWARD,style=stepper.MICROSTEP)
        kit.stepper2.onestep(direction=stepper.BACKWARD,style=stepper.MICROSTEP)
        # time.sleep(0.1)


def back(steps):
   print ("Back")
   for i in range(steps):
       kit.stepper1.onestep(direction=stepper.BACKWARD,style=stepper.DOUBLE)
       kit.stepper2.onestep(direction=stepper.FORWARD,style=stepper.DOUBLE)
       time.sleep(0.01)




def forward(steps):
   print ("Forward")
   for i in range(steps):
        kit.stepper1.onestep(direction=stepper.FORWARD,style=stepper.DOUBLE)
        kit.stepper2.onestep(direction=stepper.BACKWARD,style=stepper.DOUBLE)
        time.sleep(0.01)




def moveCar(x,y,finalangle,startangle,epsilon = 1):
    if finalangle < 0:
        calctheta=-finalangle
        direction=1 #right

    else:
        calctheta=360-finalangle
        direction=0 #left


    heading = bno.read_euler()
    angle=heading[0]
    if(angle < calctheta):
        if(abs(angle - calctheta)<180):
            difference=abs(angle - calctheta)
            multiplier=difference/1.7171052631578947
            right(int(ministep*multiplier))
        else:
            difference= 360 - abs(angle - calctheta)
            multiplier=difference/1.7171052631578947
            left(int(ministep*multiplier))

    else:
        if(abs(angle - calctheta)<180):
            difference=abs(angle - calctheta)
            multiplier=difference/1.7171052631578947
            left(int(ministep*multiplier))
        else:
            difference=360 -abs(angle - calctheta)
            multiplier=difference/1.7171052631578947
            right(int(ministep*multiplier))

    print("difference angle %s and multiplier %s"%(difference,multiplier))
    dis = int(np.hypot(x, y)) # Round
    print("Calculated Distance %s from hypot X %s and Y %s"%(dis,x,y))
    forward(dis*step)
    print(dis*step)




def distance(tripin,echpin,direction):
    GPIO.output(tripin, True)

    time.sleep(0.00001)
    GPIO.output(tripin, False)
    StartTime = time.time()
    StopTime = time.time()


    while GPIO.input(echpin) == 0:
        StartTime = time.time()


    while GPIO.input(echpin) == 1:
        StopTime = time.time()


    TimeElapsed = StopTime - StartTime


    avgDistance = (TimeElapsed * 34300) / 2
    print(str(avgDistance) + "    " + direction)
    return avgDistance

import math

def ocoordinates(xs,ys,d,alpha):

    xobs = math.sqrt(d**2-(d**2/(1+1/math.degrees(math.tan(alpha)**2))))+xs
    yobs = math.sqrt(d**2/(1+1/math.degrees(math.tan(alpha)*math.tan(alpha))))+ys
    return(xobs.real,yobs.real)







def main():

    while True:
        x_start = 0
        y_start = 0
        theta_goal = 0
        oy=[6.8,12.6,9.7] #static obstacles at the y coordinate
        ox=[7.8,12.6,20.4] #static obstacle at the x coordinate
        global startangle

        heading = bno.read_euler()
        startangle=heading[0] # angle measure by BNO005

        X,Y = mainplan(gx=27.3,gy=19.0,ox=ox,oy=oy)

        time.sleep(1)
        for ind,x in enumerate(X):
            print("X coordinates %s and Y coordinates %s"%(x*10,Y[ind]*10))
            theta_goal=math.degrees(math.atan2(Y[ind]*10-y_start, x*10-x_start)) # calculating the required angle
            print("Theta calculated from the vectors theta=%s"%(theta_goal))
            moveCar(x*10,Y[ind]*10,theta_goal,startangle) # moving a vehicle to the required position
            distance(TRIGfront,ECHOfront)
            if distance<50:
                ocoordinates(x_start,x_start,avgDistance,startangle)
                ox.append(xobs.real)
                oy.append(yobs.real)
                break
            else:
                pass
            x_start=x*10
            y_start=Y[ind]*10


main()
```

bno.py

```python

import logging
import sys
import time
import RPi.GPIO as GPIO                    #Import GPIO library

import math

import numpy as np
import sys

from Adafruit_BNO055 import BNO055


bno = BNO055.BNO055(serial_port='/dev/ttyAMA0', rst=18)

if len(sys.argv) == 2 and sys.argv[1].lower() == '-v':
    logging.basicConfig(level=logging.DEBUG)

if not bno.begin():
    raise RuntimeError('Failed to initialize BNO055! Is the sensor connected?')

status, self_test, error = bno.get_system_status()

if status == 0x01:
    print('System error: {0}'.format(error))
    print('See datasheet section 4.3.59 for the meaning.')


sw, bl, accel, mag, gyro = bno.get_revision()

```