CS665O

SMART SENSING FOR INTERNET OF THINGS

PATH FINDING ROBOT

Final Report

Srinivas Reddy Mareddi

EE18B057

Nithin Uppalapati

EE18B035

Pole Praneeth

CS18B036

P. Bharath Simha Reddy

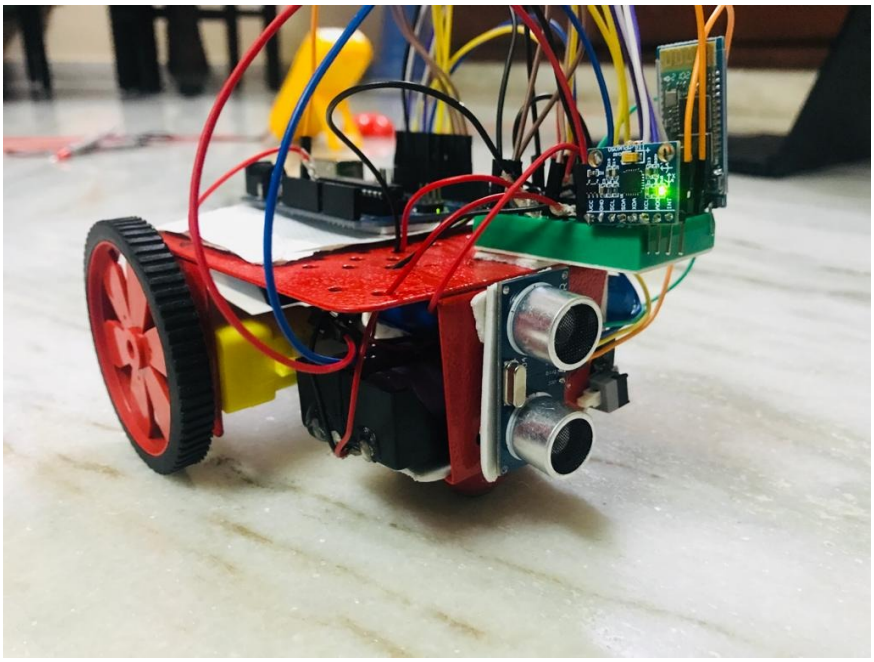
CS18B034

Abstract : We developed a remotely controlled car (Name : *PathExplorer*), in which it follows a path which the user has given to it. While doing so, it can also detect obstacles in its path and notify the user upon encountering an obstacle. The car can be controlled manually by the user at any point of time. The path transversed by the robot is discrete spline line segments in nature.

1. Introduction :

Path finding robots that can be controlled remotely or without continuous input from the user has many benefits. The most basic of these robots are relatively easy to make and implement code. The intricacy of the robot can be increased by adding new sensors and so is very robust and can be used in different applications. These robots can further be made completely autonomous using various algorithms. This is the reason we came up with the idea of building a path finding robot.

The backbone of this project is Arduino Uno, Motors, and Motor Driver. (Though from our current experience, a microcontroller with more pins and better functionality is preferred). We then introduced BlueTooth communication from the desktop app to Arduino and finally implemented obstacle detection using an Ultrasonic Sensor. These details are further discussed in the report.



2. Non Triviality of the Problem :

The problem is non-trivial because given a path the vehicle cant travel in curved paths because of the linear motion of the wheels so it is essential to approximate the path from curve to spline line segments.

3. Project Motivation :

Although there are many RC-controlled cars most of them are of manual type. Our project introduces something different by taking input from the user as the intended path and doing the rest of the work for the user i.e gives an additional functionality of autonomous control. (The robot still has manual control available.)

4. Components used :

- **Arduino Uno** - Flexible and easy to use programmable open source microcontroller board that can be integrated into a variety of electronic projects.
- **Gyroscope Sensor (MPU6050)** - Micro Electro-mechanical system (MEMS), it consists of a three-axis accelerometer and three-axis gyroscope. It helps us to measure velocity, orientation, acceleration, displacement and other motion-like features.
- **Bluetooth Module (HC-05)** - Uses serial communication to communicate with the electronics. Usually, it is used to connect devices like mobile phones using a short-range wireless connection to exchange files. It uses the 2.45GHz frequency band.
- **Ultrasonic sensor (HC SR04)** - Ultrasonic sensor is mainly used to determine the distance of the target object. It measures accurate distance using non-contact technology, which means technology that involves no physical contact between sensor and object.
- **Motor Controller (L293D) -** Half Bridge Dual Motor Control
- **Two Motors + 3 Wheels & 2 LiPo Batteries.**

5. Desktop Side Implementation :

5.1 App Design :

- App is developed using python language.
- It is made using the **Kivy** library that is available in python .
- Kivy is a library that enables us to make touch-based Apps for desktop.
- Kivy creates App UI through **GUI** window through which we can interact via a trackpad or touch screen.

We split the GUI window into two parts -

1. Controls Corridor
2. Path Drawing Corridor

5.1.1 Buttons in Corridor Control :

- **Start:** after drawing the path when we press start the car starts to move in the path drawn.
- **Approx Path Gen**: Will draw the approximated path calculated using the RDP algorithm on top of the input drawing. Once the coordinates are calculated, the kivy canvas widget is used to draw.
- **Stop:** Stop makes the car stop
- **Clear:** Clear is used to clear the drawing screen
- **Manual:** Manual is used to take the car into manual control mode. Send the **"M"** flag
- **Speed control Slider:** To adjust the speed of the car. The slider sends values between 120 and 220 (ENA and ENB voltage values). It sends the "**S**" flag and speed value
- **Manual Joystick controls:** Controls used for controlling the car manually

5.1.2 Buttons :

Buttons are implemented using the Kivy button widget. when a button is pressed a function binding it gets called and relevant action takes place and a similar process takes place when a button is released.

5.1.3 Drawing Corridor :

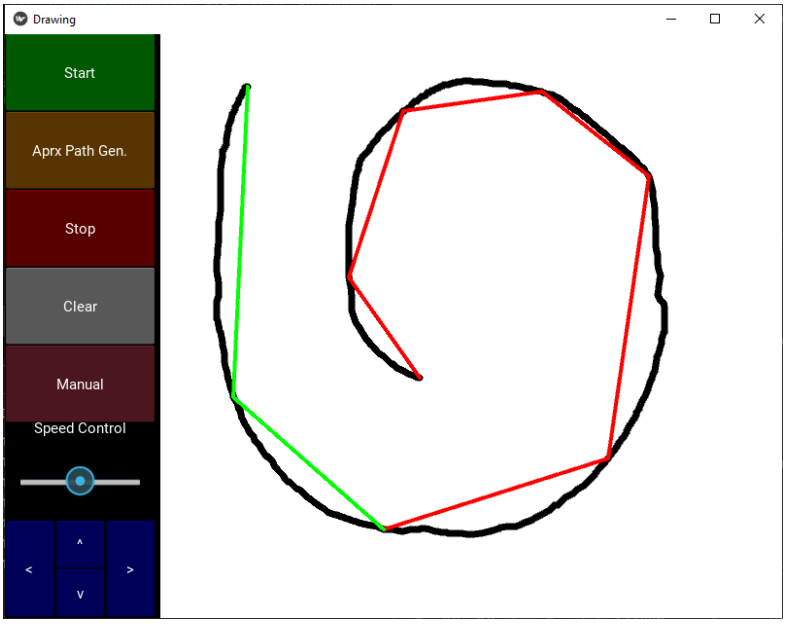
It is implemented using canvas widget that is available in Kivy.

5.1.4 Path Processing Algorithm:

Given a path, we could have found the radii of curvatures at each point in the drawn curve. Radii can vary from R=0(corresponds to rotating about itself) to R=∞(corresponds to a straight line). This could have been achieved by varying the speed on both the motors separately. But given a path, this was computationally intensive and so implemented it using approximate line segments.

We used the Ramer–Douglas–Peucker algorithm to calculate these approximate line segments. The algorithm decimates a curve composed of line segments to a similar curve with fewer points. In our case, we are expressing the final path as in the form of splines of line segments.

Our App will show the approximated path that is received from the Path Processing Algorithm and it will also give us the feedback in green color after the car has executed the path successfully or got stopped due to an obstacle.



5.2 Transmitting Data through Bluetooth :

Data transmission through bluetooth was achieved using the **pyserial** python module. We have to define a variable that contains the name of the **bluetooth port** and **baud rate**. Then we send data using the var.readString() function. We need to be aware of sending encoded data (utf-8) as we cannot send Unicode characters directly. The data was sent using a particular format. First letter denotes the flag and then the subsequent data is sent.

5.2.1 Manual Control :

For manual control, the manual control button should be pressed initially. This sends a manual object within the application. Further pressing any button on the joystick, the program transmits a string of data. The first letter of the string is a flag sent to the Arduino for manual control **(M)** and the second letter denotes the action that the Arduino should perform **(L, R, F, B, C)**.

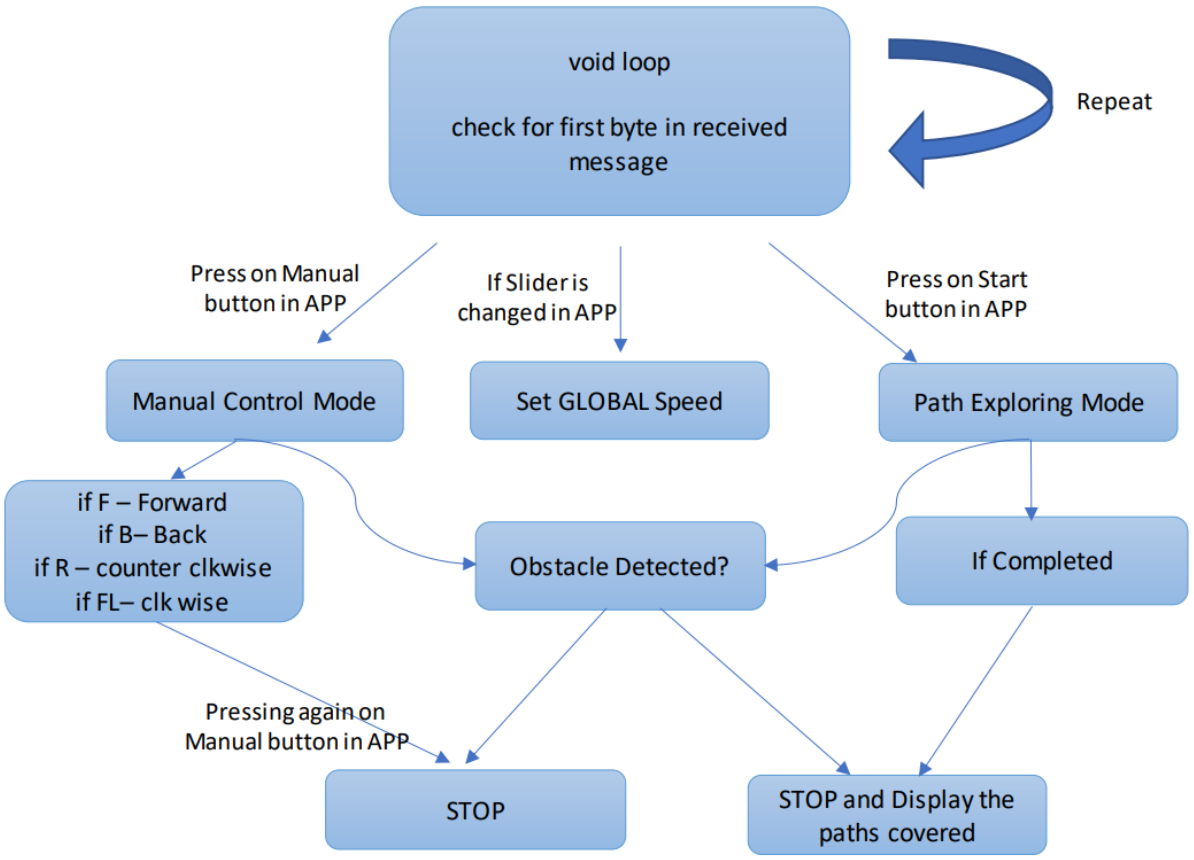
5.2.2 Pathfinding Mode :

When the robot is not in manual control mode, the path drawn by the user is sent as a list of distances and angles to be moved by the robot. We send the data corresponding to a single action at a time and store it in Arduino and then execute the motor action, and once the action is executed, we send an acknowledgment message back from Arduino to python. The (n+1)th data is sent only if the nth acknowledgment message is received.

In short, data corresponding to a path will be in the following format:

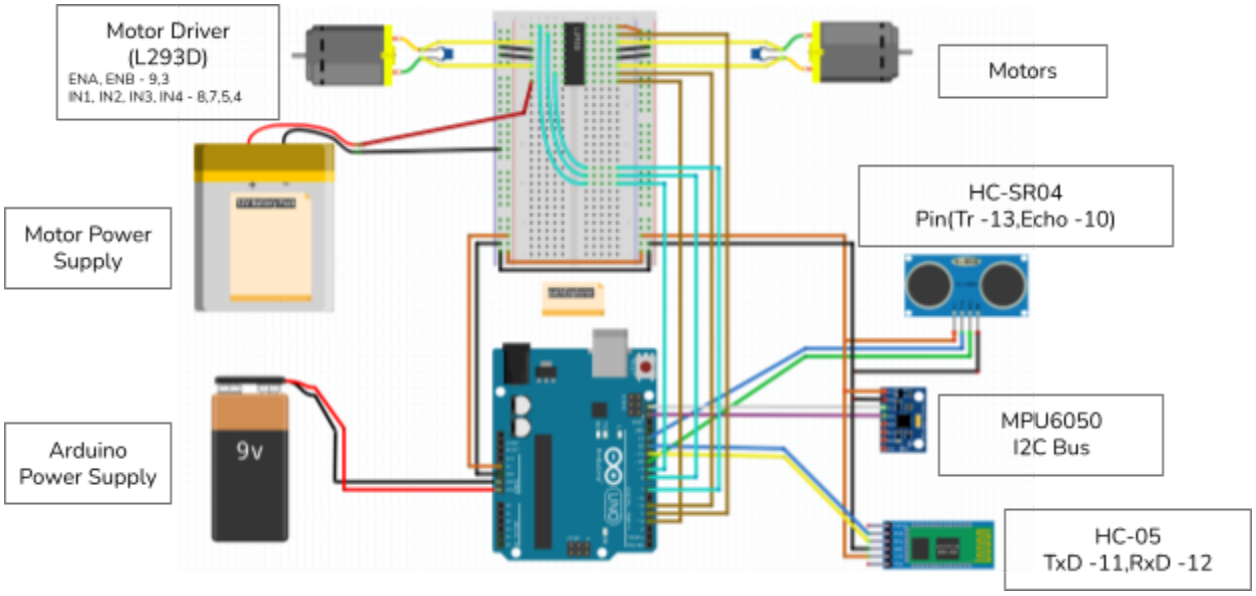
$$\mathbf{[\ length_0, angle_0, length_1, \ldots.\ , angle_n, length_n\ ]}$$

6. Arduino Side Implementation



6.1 Pin Connections :

The below figure shows all the pin connections and numbers made to the Arduino.



6.2 Programming :

We start by including **Wire.h** and **MPU6050.h** and **SoftwareSerial.h** libraries. We also define some global variables.

6.3 Setup :

We start by defining all the variables required along with assigning Input and Output pins. We also wait for gyroscope sensor initialization and then set its threshold value (Reduces Noise and Drift in the sensor data).

6.4 Receiving Bluetooth Data :

Bluetooth data is received by mimicking a serial port using software serial library and defining a virtual serial port. The bluetooth module can be received or transmitted using the pins mentioned in the SoftwareSerial variable. The variable is initialized by mentioning the baud rate similar to initialization of serial. The bluetooth module supports baud rates from 9600 upto 1382400. However Arduino Uno supports baud rate upto 115200. Here we use a baud rate of **9600**.

The messages from the desktop are received in a string fashion. There are few message identifiers as shown below :

Message Identifier	Action
1st bit in string = 'M'	Enables manual mode
1st bit in string = 'P'	Enables the path executing mode
1st bit in string = 'Q'	Stop and go to idle state
1st bit in string = 'S'	Changes the Current Speed of Arduino

Vehicle is stopped when 'Q' is received. Further when the flag is

- **M** : Data Format - **"MF", "MB", "MC" , "MR", "ML"**. Characters of the string are read one by one. "MC" is sent after any joystick button is released.
- **P** : Data Format - **"P<integer>" or "P".** Here we detect the first bit - 'P' and go into path finding mode, and then we get "P<integer>" as data in string format. The first bit is useful again in detecting whether we are sending valid data and the remaining part of the string is used for path execution.

  Ex: The following stream of messages executes the following['P', 'P120', 'P-83', 'P34']:

  First 'P' is detected and will go to path executing mode, then as 'P120' is received, it goes in a straight line for 120 units. And after this execution an acknowledgement bit is sent back to the app[1]. And then 'P-83' corresponds to rotating -83 degrees in clockwise direction, and so on...
- **S** : Data Format - **"S<integer>" .** Here the first bit is used for detecting that this data is for speed control and the remaining part of the string is used for setting up speed for motors in the arduino. The slider in python outputs a value from 0 - 100. We convert this range appropriately to use it for controlling the duty cycle of

the PWM signal sent to the enable pins in the arduino sketch. In our case we are converting the 0-100 range to 120-220 (out of 255) duty cycles' ranges

Ex: T*he following message 'S67' will set duty cycle to (120 + 67) in arduino implementation.*

6.5 Manual Control :

When the manual flag 'M' is detected, the arduino calls the manual control function. When the joystick buttons are pressed on the app, the following characters are received -

F - Forward, **B** - Backward, **R** - Right, **L** - Left

Once the user lets go of the button, character **C** is sent and the vehicle stops.

6.6 Straight Line Function :

The argument to this function is a +ve number and based on its magnitude, we move the car to that extent. The car moves forward for a specific amount of time based on the given argument. No negative feedback is given. (Requires wheel encoders)

6.7 Angle Turning Function :

We sense the current angle just after entering this function and based on the input angle and its sign, we rotate either clockwise or counterclockwise, +ve corresponds to counterclockwise and -ve corresponds to clockwise.

We implemented this entire functionality in negative feedback with the gyro sensor and terminated this function when we reached the desired state. We make use of the difference in angle as measured by Gyro to turn the required angle. We add values for a certain amount of time to reduce noise in the difference between measurements.

6.8 Obstacle Detection :

Uses Ultrasonic Sensor. Measures the time taken between the signal sent from the trigger pin and received by the echo pin.

The sensor algorithm checks if there is an obstacle within the range of 10cms. It takes 2 samples of data then averages it. When an obstacle is detected, the car will stop

moving if it is in path executing mode or in manual mode when moving forward. We can only move backwards or rotate either towards right or left, when in manual mode.

6.9 Transmitting Bluetooth Data to App:

The car sends back some data to the python program via bluetooth to acknowledge whether the data is received or if an obstacle is detected. This is achieved by printing the line in the serial communication.(softserial.println())

Message Sent Back :

Acknowledge Bit	Inference
0	0th action is executed
1	1st action is executed
n	n-th action is executed
-1	An obstacle was detected and hence the path exploring was stopped.

+ The initial connection establishment to bluetooth will take upto 1 sec, as the default setTimeout for Serial comms is 1 second in Arduino IDE. We can increase the timeout duration if needed.

7. Use Cases :

In our project we are trying to develop a desktop app for controlling an RC car which travels on a path given as input by the user. This application can be extended to drones, submarines, and different kinds of robots. Apart from the user drawn input, the path can also be generated in other ways such as AI search methods in order to find an optimal path, given the topology of the surroundings. Some of the potential use cases are :

- A pre-programmed path instructed to a remote vacuum cleaner in a house, with path denoting the path it should follow. A user can define certain paths for his vacuum cleaner to operate for better and efficient cleaning.
- It can also be useful in Rangolis / Arts as the user can draw the input (art) and thus it is possible to draw with the help of the robots which are capable of holding a drawing tool and placing inappropriate places.

- Surveillance drone operation - can be used to survey roads and overview cities for planning purposes by giving input based on geographical maps. It also can be used to survey buildings and structures for various measurements and also enter inaccessible locations which would not be possible by controlling the drone manually.
- This can be used in combination with algorithms which enable autonomous operation which can subsequently be used to explore remote areas and get a rough idea of the position and size of obstacles.
- This can be used in farming for irrigation of lands by using it as a water dripper. The path can be given as an input using a mobile application.
- The above case can be extended to harvesting, seeding, laying pipes and many more.
- It can be used as an accurate drawing machine in very large areas like marking road dividers. In this case the input can be given using satellite maps and the robot can be calibrated accordingly based on distance on map to actual distance.

8. Challenges Faced :

- Received faulty components (Gyro and Ultrasonic Sensor) and had to return and repurchase the components leading to slight delays.
- HC-05 Interfacing with MacOs is very quirky and we were having to forget the device and reconnect the bluetooth module every time we run a program to send data.
- On trying the same on Windows, we had another problem, this time while trying to install the polyprox module. This module is apparently unstable for Windows and Ubuntu. Instead we settled for RDP(Slower).
- Voltage drops occurred due to heavy current consumption of motors initially when using a single battery. So we had to connect two separate batteries to the Arduino and Motor Controller.
- Gyroscope and Motor Controller were not working together and many on the internet had the same problem. We didn't figure out why (Could be because of a proper Lithium Ion battery). Both of them were working perfectly fine individually.
- Some loose connections also interfered with the progress as unexpected results were displayed. We had to rewire them to ensure a better connection.
- On the whole, working remotely from different locations was tough as only one of us had the robot to test the code.

9. Conclusion and Future Work :

The project has been fun and difficult at the same time. It would have been much better if it were offline. Nonetheless, it was a great experience and learned a lot about all the sensors used as well as the robustness as well as limitations of the Arduino Uno.

Future work that can be done involves improving the robustness of sending data through Arduino Uno. We can include a battery level indicator using resistor dividers and measuring voltages. The APIs were easy to use but implementing code that directly communicates with the hardware and register would greatly reduce the latency and memory occupied. Also using a microcontroller with more ports would have been beneficial and could be used to interface many other sensors and increase the complexity of the robot. Finally, we could introduce more functional modes for the car to operate in and also increase the smoothness of the movement (using PID) which requires better bluetooth communication.

10. Project Roles and Contribution :

Nithin (EE18B035) - Parts procurement, RC Car Assembly and Motor Control (32%)

Bharath (CS18B034) - Path following algorithm and App UI (30%)

Praneeth (CS18B036) - Path following algorithm and Interrupt coding (8%)

Srinivas (EE18B057) - Bluetooth Communication and Interfacing (30%)

11. Reference Links

[1] ***https://pypi.org/project/rdp/***

[2] ***https://github.com/kivy/kivy***

[3] ***https://stackoverflow.com***

[4] ***https://www.arduino.cc/en/Tutorial/LibraryExamples***