Electronics and Robotics Club

BITS Pilani, K.K. Birla Goa Campus
http://erc-bpgc.github.io
ROS Hackathon 2021

1. General Note

This hackathon aims to introduce you to ROS by guiding you through the process of building a robotics software stack yourself. Please go through the instructions carefully before attempting the task. All the files related to the assignment questions and necessary for solving them are present in this GitHub repo. We hope that this hackathon gives you first-hand experience with ROS, Gazebo, and how to find solutions to problems yourself. These skills will definitely be of help in any future projects you may wish to pursue. At first glance, the problem statement may be intimidating to many of you. Don't panic! They are meant to be complex and challenging and will take some time and effort to complete. We will be judging submission based on your code and on the thought that went into the attempt - so make sure to document everything! Do remember that keeping your code clean and well commented is necessary so that other people can understand it.

In the Learning Resources section of the doc, we have tried to include a comprehensive set of resources, but they may end up falling short in a few places. We encourage you to search online as much as possible for any issues or doubts you may have. In some cases, you may find resources better suited to you. We hope you will learn some valuable skills through this hackathon, and searching for yourself will be a key part of that process.

2. Logistics

2.1 Submission guidelines

The hackathon is a group activity, please form a group of a maximum of 3 people, we expect all members to actively contribute to the task.

One of the team members should create a GitHub Repository consisting of all the required documents, files and code that the team wants to submit for the attempt. The title of the repo should be ERC-ROS-Hackathon-2021. It should be properly organised with the README.md file consisting of thorough documentation of your attempt. Include a screen recording of your robot performing the navigation as well as a rosbag.
The submission deadline is 13 June EOD.

2.2 Contact Information

Feel free to ask your doubts on the ERC WhatsApp groups and the relevant channels on the Discord server. You can also contact the organising team directly-

- Archit Rungta
- Ashutosh Gupta
- Pranav Goyal
- Sushant Sreeram Swamy

3. Learning Resources

Check out the ERC handbook for an extensive compilation of information and resources for everything robotics, use the resources below to supplement them.

3.1 Python

This language is essential for robotics, being very easy to learn and prototype with. Check out this guide by Google to start. This book and the docs are good references. If you prefer video tutorials, refer to this playlist by sentdex.

3.2 Linux Terminal

This will be essential for working with the various development tools you require. Get started here.

3.3 Git

An integral part of most software projects, Git is a tool for version control and managing changes to code. Check out this course for an intro. Use this cheat sheet for reference.

3.4 ROS

The Robotics Operating System is the backbone of every complex robotics stack. You will need to first set it up on your machine. Since ROS is supported for Linux only, we recommend that you dual boot with Ubuntu 20.04. If this isn't possible, you can try either a virtual machine or Windows Subsystem for Linux for Windows users. After getting to a Linux prompt, follow the installation instructions in the wiki. Feel free to reach out to us if you run into any issues. To introduce the basic concepts, the tutorials section of the wiki will be your best friend. A good supplement that delves a little deeper is this book by Morgan Quigley.

3.5 Motion Planning

For video tutorials and articles on path planning algorithms, use resources at this link. Use these (1, 2) books for a brief introduction and reference.

3.6 Control Systems

Robotics deals with many continuously operating, dynamic systems, which are dealt with using Control Theory tools. For the basic concepts, check out the course on Control of Mobile robots by GAtech and this video series by Steve Brunton (1). The most commonly used controller everywhere is the PID controller. Check out this video series by MATLAB(1) to understand it better.

3.7 Collision Checking

For collision avoidance, You will need to use a computational geometry package along with python to check if your path intersects with the region of obstacles or not. One such commonly used python package is shapely. Check out the shapely docs to understand how to use the package. You can also refer to the Gennav stack to understand the implementation of shapely.

4. Problem Statement - Automation of Omnibase

Omnibase is a ground-based robot package developed by the ERC. Since it is a four-wheeled robot with omni-wheels, it can move in any direction given the appropriate velocity to the individual wheels. This makes the robot agile and highly manoeuvrable. For sensing the environment, Omnibase has a rplidar device attached to the top. Omnibase is primarily built to simulate the Trotbot, a robot currently being developed by the ERC, as well as being a platform for testing ground-based planning algorithms. We have also developed a navigation stack for Trotbot called GenNav.

Your task is to build a simpler software stack for Omnibase, which takes a goal point as input, plans a path through a field of static obstacles (coordinates are known) and then publishes appropriate velocity values to the **/cmd_vel** topic.

You will need to demonstrate the robot successfully navigating the given obstacle field without knocking over any obstacles. You can choose whichever path planning algorithm you want.

4.1 Required Stack Structure

1. **Obstacle detector node:** This node publishes a fixed list of obstacles. Here you will be publishing the list of the centre points we have provided.

2. **Planner node:** This node subscribes to obstacles and publishes a path. You can take user input for the goal point in this node.

3. **Controller node:** This node subscribes to the path and publishes to the topic /cmd_vel. This controller will traverse the path one point at a time using a PID controller.

This structure might seem a little redundant for our hackathon. However, it aims to introduce you to how one would set up a project with more complex components, such as an obstacle detector that works with lidar or a planner, which must keep replanning the path every time an obstacle comes in the way etc. In such cases, having a modular structure can prove helpful.

You can select whichever message types you feel appropriate. For convenience, we also recommend you write a launch file so that you won't have to run all three nodes and also launch the world separately each time.

Note

- The user will input the goal point, you need to ensure that the point lies outside obstacles before accepting the input. You can simply check if the point is within a certain distance of any of the provided obstacle centre points. Also, don't forget to include the size of the robot in this calculation.
- We have created the world file such that it can be used for both tasks. All obstacles are cylinders with 0.25m radius. All cylinders have been located in grid at locations **{ (0, 1.5), (0, 3), (0, 4.5), (1.5, 0), (1.5, 1.5), (1.5, 3), (1.5, 4.5) … (4.5, 0), (4.5, 1.5), (4.5, 3), (4.5, 4.5) }** with the bot initialized at **(0, 0)**.

5. Guidelines on how to proceed

All this might seem a little intimidating if you haven't worked on a software project before. The key thing, however, is to break it up into manageable chunks. We recommend you start off by building a small publisher which publishes to /cmd_vel to move the robot. After you are comfortable with this and can see the robot move properly in the simulator, you can work on a path planner. Initially, you can hardcode the obstacle values into the planner file until the planner starts to work properly. At this stage, you can separate everything into separate ROS Nodes.

Another essential skill one should have is “The art of googling” (not kidding). Because of the availability of platforms such as stack overflow, The task of solving errors and debugging has become a lot easier. Therefore, we highly encourage you all to google all your errors and queries first.

The steps listed below are just to give you a hint of how to proceed, you can, of course, follow any method you want.

5.1 Setting up Omnibase

We have provided the world file and the corresponding launch file required for the project in the hackathon repo. To use these with the Omnibase package, you must:

1. Clone the Omnibase repo
2. Copy **obstacles.world** into **omnibase_gazebo/worlds**
3. Copy **hackathon_omnibase.launch** into **omnibase_gazebo/launch**
4. Run **roslaunch omnibase_gazebo hackathon_omnibase.launch** to launch the world
5. Run **rosrun omnibase_control controller_node** to run the controller
6. Publishing to **/cmd_vel** should result in Omnibase moving in the world

5.2 Run Omnibase

Write a PID controller node for omnibase and test in Gazebo. You may use hardcoded paths/points for this step, do remember to make the appropriate changes for the node to work with paths/points published by the path planning node. You can also refer to the PID controller example for the omnibase given in the GenNav repo (omni_diff_pid.py).

5.3 Write the planning algorithm

Write a path planner node and print the path/plot using matplotlib to test it. Do remember to comment out any lines of code used to print/plot the path for testing once the node works as expected.

5.4 Write the obstacle detector node

Write a node to publish the list of the centre points of the obstacles provided above.

5.5 Make a Launch File

Write a launch file to launch the world with omnibase and the three ROS nodes.

5.6 Testing and Debugging

Test and debug the stack in Gazebo.