Final Project

The final project will cover everything you have learnt in the last month, demonstrating how to bring everything together into a complete robotics stack. There are two tasks that you can choose from.

1. Tasks

1.1 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 maneuverable. For sensing the environment, Omnibase has an 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.

Your task is to build a controller for Omnibase which takes a goal point as input, plans a path through a field of static obstacles and then publishes appropriate values to cmd vel.

We have provided the world file as well the corresponding launch file required for the project in the QSTP 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 `qstp_omnibase.launch` into `omnibase_gazebo/launch`
4. Run `roslaunch omnibase_gazebo obstacles.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 the world

2.2 Automation of RotorS

RotorS is a MAV (Micro Aerial Vehicle) Gazebo simulator. It provides some multirotor models such as the AscTec Hummingbird, the AscTec Pelican, or the AscTec Firefly, but the simulator is not limited for the use with these multicopters. This package also contains some example controllers, basic worlds, a joystick interface, and example launch files.

Your task is to make a 3D path planner which will generate a set of waypoints consisting of an obstacle free path. You then have to write a node which publishes these points on the topic `/firefly/command/pose.`

1. Follow the guidelines to download all the packages related to RotorS simulator and its dependencies.
2. Copy `obstacles.world` into `rotors_gazebo/worlds`
3. Copy `qstp_firefly.launch` into `rotors_gazebo/launch`
4. Run `roslaunch rotors_gazebo qstp_firefly.launch` to launch the world
5. Publishing waypoints to `/firefly/command/pose` should result in the firefly drone moving to the published waypoint

2. Project Guidelines

2.1 A Note on Obstacles

We have created the world file such that it can be used for both tasks. All obstacles are 10m tall 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)`**. The goal point is **`(6, 6)`** In your path planning algorithm, you can take this list as input. When you want to check if a point is inside an obstacle (something commonly required in sampling based algorithms), you can simply check if the point is within a certain distance of any of the provided centre points. Also don't forget to include the size of the robot in this calculation.

The obstacles have also been designed to fall over if hit. So the target is to leave the field intact as well get the robot to the other side.

2.2 Required Structure

1. **Obstacle detector** node which publishes a fixed list of obstacles. Here you will be publishing the list of centre point we have provided.
2. **Planner** node which subscribes to obstacles and publishes a path. You can take any two appropriate points in the world for start and end
3. **Controller** node which subscribes to the path and publishes cmd_vel. This controller will traverse the path one point at a time using PID.

This structure might seem a little redundant for our project. However it aims to introduce you to how one would set up a project with more complex components such as an obstacle detector which 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 each time.

2.3 Final Product

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

2.4 A Note of 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` or `/firefly/command/pose` 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.

This process is just to give you a hint of how to proceed. You can ofcourse follow any method you want. Also please feel free to approach one of the mentors if you have any difficulties.

Submission

The submission needs to be in the form of a ROS package which you will upload to your github repository. Be sure to include a recording of your robot performing the navigation as well as a rosbag. The deadline is **Wednesday 24th June.**