BotsAndUs – Robotics Engineer

Coding Exercise

Overview

This coding exercise is part of the application process for a robotics engineering role at BotsAndUs. It involves the creation of a custom package that implements a simple local planner for use with the standard ROS navigation stack.

This is not intended to be a difficult and time-consuming task. It's a way for you to demonstrate the approach you take to writing software – how you follow specifications, how you provide documentation, the way you test functionality, and how you make decisions about what to build. It also gives us an opportunity to discuss this process with you afterwards.

This task is based on the real-world problems we have to solve for customer applications, and is broadly representative of the kind of tasks you would perform in this role. This task should be implemented in C++ using appropriate best practices**,** and should be suitable for use with the standard ROS navigation stack – but beyond that, there are no specific requirements on tools to use. We are more interested in your approach to the problem than the specific tools you have experience with already – and choosing an appropriate tool is part of the exercise.

Functional requirements

Your planner should initially implement the following features:

- Implement a ROS package which provides a new local planner for **move_base**. This planner must implement the nav_core::BaseLocalPlanner API for the nav_core package.
- The local planner should receive a path from the global planner and generate linear and angular velocities to send to the base, as defined by the **BaseLocalPlanner** API.
- The local planner should always rotate in-place towards the next point in the path, and then move towards it.
- When arriving at the last point in the path, the planner should take the supplied goal orientation by rotating in-place.
- For the purposes of this exercise, all obstacles can be ignored.

In addition to the above, you should improve the performance or feature set of your package to implement a more accurate or reliable planner, according to your own expertise. You should implement at least one of the following features:

- Improve the *computeVelocityCommands* function to increase the accuracy and reliability of the path following algorithm.
- Use the available costmap to detect obstacles, and stop the robot before a collision occurs.
- Implement a node that checks if the robot is following the calculated path and publishes a message on a ROS topic if the robot appears to be stuck and unable to follow the path.

Non-functional requirements

- Your planner must be implemented and compatible with ROS Noetic.
- The code should follow best-practices and a consistent coding style.
- You should include unit tests for the custom code you have written.
- The code should include suitable documentation and comments as required.

The local planner must be testable in Gazebo using the turtlebot 3 simulation (using the model *burger*). This launch file must be modified to bring up the navigation and localization of the platform, so that it is ready to receive goals from the standard **move_base** package interface using your new local planner.

Deliverables

You should deliver:

- Your code that implements the system described above as an independent ROS package, including its own XML configuration, launch files, code, and tests.
- Any documentation or test cases you have produced.
- Details about how the code can be built and executed.

You can submit your results in a Git repository or a ZIP file, and please don't hesitate to reach out to us with any questions.