COMP 514 Assignment 2
Flying the Firefly
Sept 25

In this assignment, you will provide a trajectory for Firefly to follow. Your trajectory will be generated from a cubic polynomial (3rd order polynomial), and you will provide the positions for the robot.

Do not start this assignment before you have tried and understood the tutorials provided in the class.

1 Preparation

- Clone the rotors repository
  `> git clone git@gitlab.cs.mcgill.ca:applied-robotics/robots/rotors.git`

  You might need to install extra packages:
  `> sudo apt-get install libeigen3-dev`
  `> sudo apt-get install ros-noetic-octomap-ros`
  `> sudo apt-get install libgoogle-glog-dev`
- Add `rotors` to your catkin workspace and build the package by running
  `> catkin build rotors_gazebo`
- Once the new packages are built, you should see the robot by running
  `> roslaunch rotors_gazebo a2.launch`

2 Useful Tips

- **Make sure that you understand the tutorial before working on the assignment.**
- If you are not sure how to use a topic or a service, the following commands should help you

```
> rossrv list                       # list all available service time
> rossrv info [service name]        # check definition of [service name]
> rosservice list                   # list all services currently running
> rosservice type [service name]    # data type of [service name]
> rosservice info [service name]    # information about [service name]
> rosservice call [service name]    # request a service
```

- Some useful ROS/C++ comments for time-related operations

```
double t = ros::Time::now().toSec(); // get current time in seconds
```

3 Specifications

Your main task is to create a node that provides a service for the user to input the target position. If a user requests the service, you will plan a trajectory using cubic polynomials toward the user-defined target.

3.1 Define Your Own Service

- Create a new ROS package under your Gitlab repository `robotic-coursework-f2023`. Your new ROS package should be called `highlevel_msgs`. If you do not understand the terminology "package", go back to tutorial-01.
- Create a customized service `MoveTo.srv`. This service should have 4 inputs and 1 output. The inputs are the target position (x,y,z) and the target time (T). The output is a boolean value indicating whether the service can be executed.
- There are a few things you need to add in CMakeLists.txt. Use `hello_msgs` package as your reference. Add `genmsg` to `CMakeList.txt` and `package.xml`. This is a required package if you have customized services.
- Try "catkin build" this package to ensure that your service type is created. You should find the automatically generated header files in `devel/include/highlevel_msgs/MoveTo.h`. When you run `rossrv info highlevel_msgs/MoveTo`, you should see the service you created. (You need to `source` your workspace again at this point.) **Do not continue if it fails at this step**.

3.2 Create a Service Server

- Create a new ROS package under your Gitlab repository `robotic-coursework-f2023`. Your new ROS package should be called `cubic_polynomial_planner`.
- Write a ROS node that spins at 500hz and a **class** that performs the operations for your node.
- Your node should subscribe to the current pose of the robot `/firefly/ground_truth/pose`
- Your node should advertise a service called `move_to` which indicates that the user wants to move the robot to a specific target position. You will provide a callback function for your service. In the callback function, you need to save the following information
  - Read the user input through the service request and save the value. They should provide the target position and the target time.
  - Save the current position of the robot and the current time. (Why? You will need them soon).
  - Your service should also handle situations that the robot cannot execute. For example, if you drive the robot to position `z=0`, the robot will crash into the ground. In this case, you should set the response to false and return false as well.
- Compute the desired pose and publish it
  - Compute the translation at time t using 3rd-order polynomial. You can keep the orientation as a constant.
  - Publish the desired pose to `/firefly/command/pose`.

3.3 Launch File

- Create a launch file called `a2.launch` to launch your node. You need to name your node as `pose_planner`. If I run `rosnode list`, I should see `/pose_planner`.
- Check that you can start your node by `roslaunch cubic_polynomial_planner a2.launch`

3.4 How to submit

- It is recommended that you run `catkin clean` to clean up your workspace, rebuild your packages, and then test your code again.

- When you are ready to submit, push your change to `robotic-coursework-f2023`. Do not create a new repository. Do not add your catkin workspace. When you visit your repository, you should only see the 3 ROS packages you created:

  ```
  keyboard_controller
  highlevel_msgs
  cubic_polynomial_planner
  ```

- Please push your code after Sept 21 (3 days after the A1 deadline). In case you modify your A1 and break it by mistake.

4 Evaluation

We will build your code by running `catkin build cubic_polynomial_planner`. We will not build `highlevel_msgs` separately for you. You need to set up your packages in a way that your dependencies are automatically built. We will test your implementations as follows:

1. Open a terminal and run
   `> roslaunch rotors_gazebo a2.launch`
2. Open another terminal and run
   `> roslaunch cubic_polynomial_planner a2.launch`
3. Open another terminal and run
   `> rosservice call /pose_planner/move_to 1 2 3 5`
   The robot should move to position (1,2,3) in 5 seconds.
4. We will also test your code by trying a few different combinations of inputs

Marks (10 points total):
(1 point) T1: Correctly create a customized service
(1 point) T2: Correctly created a node name `/pose_planner` after launch
(2 points) T3: The service call returns false given bad inputs (1,1,0,3)
(2 points) T4: After calling the service with input 1 2 3 5, the robot reaches the target
(2 points) T5: After calling the service with random inputs, the robot reaches the target
(2 points) T6: After calling the service with random inputs, the robot reaches the target