ME495: Embedded Systems in Robotics

Homework: ROS command line tools, and publisher/subscriber challenge

In this homework, you will be exploring many of the ROS concepts that we have covered in class. As online documentation is a big part of this class (and open-source software in general) you will be expected to present your solutions to this homework in a way that is thoroughly documented, nicely formatted, and easy to follow. You will accomplish these goals by writing a README for the repository that you'll turn in with this assignment. Your document does not to be extremely long, but it should include clear indications of what questions you're answering, links to relevant files and code snippets, and any resources that you used to help you solve the problems. Failing to meet these requirements will impact your grade.

GitHub Classroom and README Formatting

In Canvas, you'll be provided a link for automatically creating a private repository that you'll use to submit your answers to questions and relevant code. GitHub has the ability to automatically render several different markup languages as HTML, and we'll be using this feature to produce a nicely formatted README. **Don't forget to provide the URL for your repository in Canvas.** Here are a few helpful resources:

- Likely you'll want to use Markdown for your README. To do this, all you need to do is name your file README.md, and it will automatically be shown on your repo's homepage, and the Markdown will automatically be converted into HTML.
  - Check out this document from the `trep_puppet_demo` for an example Markdown file we've seen this year. You can either clone the repo, or click *Raw* to view the file before it was converted to HTML.
  - Also note that the README for the `trep_puppet_demo` is automatically displayed and nicely rendered when you visit the homepage. This is how you want your submission to look.
- Rules for GitHub Flavored Markdown
- General Markdown syntax and information
- There are great Chrome and Firefox extensions for doing Markdown processing and previews. In Chrome I use MarkView and Markdown Preview Plus, and in Firefox I've heard good things about Markdown Viewer. Also note that many editors have live markdown previewer plugins (such as Sublime's `sublimetext-markdown-preview` plugin). There are also online Markdown editors such as Dillinger.
- You can link to specific lines or chunks of a file
- You don't have to use Markdown, you could also use any other markup languages supported by GitHub. I personally use `org-mode` for all of my markup.

1 ROS Command Line Tools and Package Analysis

For this section you will clone a package that I have written in C++, and then it will be up to you to use a combination of ROS command line tools, code you write, and analysis of the (intentionally) poorly documented code to answer the questions below. Note that the command line tools should be sufficient to answer all questions and they provide the easiest path to solutions. The package contains only a single node, `ros_cl_demo`, that can be started with `rosrun`. Note that the package will have to be compiled with `catkin_make` to build the executable, and the custom messages. The source repository is located here: https://github.com/NU-MSR/me495_hw1.git To set yourself up properly for the rest of the assignment, I'd likely create a README.md file use it to answer these questions. Then in Sec. 3 you will create a ROS package and Git repo that you can add this README.md file to.

1. The single node contains one publisher and one subscriber. Answer the following questions and describe how you found the answers.
   (a) What are the names of the corresponding topics?
   (b) Describe the message definition for each of the topics.
   (c) What packages define the messages?
   (d) What frequency (approximately) does the publisher publish data at?
   (e) Use `rqt_plot` to plot the data being published. This data should be familiar, what is it?
   (f) Publish a message on the subscriber's topic to trigger a callback in the node. What happens in the terminal where the node is running?
2. The node also contains a single service provider. Answer the following questions and describe how you found the answers.
   (a) What is the name of the service?
   (b) Describe the definition of the service request and response.
   (c) This service performs some sort of mathematical computation. Describe this computation.

2 SSH Authentication and GitHub

In this section, you will basically just follow some GitHub tutorials. There is no need to write anything about this section, and you may have completed some of this already.

3. In general, I find it far more convenient to work with GitHub over SSH instead of HTTPS. Not only is it more secure, but it also provides a very easy way to avoid typing your GitHub password over and over. GitHub claims you can use a credential helper to cache your password over HTTPS, but I've never had much luck getting this to work. For this task, all you need to do is read and follow the directions for all of the pages linked on the Connecting to GitHub with SSH page. Pay specific attention to the following pages:
   - Checking for existing SSH keys
   - Generating a new SSH key and adding it to the ssh-agent[1]
   - Adding a new SSH key to your GitHub account
4. Non-MSR students can learn more about SSH by looking at the networking exercise that we did during the MSR orientation this year.

[1]Note that on Ubuntu, you probably don't need to manually add your SSH key to the `ssh-agent`. By default the `gnome-keyring-daemon` will allow you to unlock an SSH key and keep that key unlocked. You could use the `ssh-agent` to give you more fine-grained control over locking/unlocking keys, but you don't need to.

3 Working with the `turtlesim` package

In this section, you will work with the `turtlesim` package, a simple package used for teaching basic ROS concepts. Your goal is to create a ROS package containing a node that controls the `turtlesim` to follow a reference trajectory. **All code in this problem should be written in Python.**

5. Create a ROS package using `catkin_create_pkg`. As you answer the rest of the questions be sure to keep your `package.xml` and `CMakeLists.txt` files up-to-date with the packages you depend on.

6. Initialize a Git repository that tracks all *relevant* files for your package, and doesn't track any unnecessary files (likely you'll want to use a `.gitignore` file). As you complete the rest of this section, try to commit as you go. Once your local repo is initialized, you'll want to add the private GitHub repo created by GitHub Classroom as a *remote*. Your command will look something like

   ```
   git remote add origin git@github.com:ME495-EmbeddedSystems/homework-1-f2017-USERNAME.git
   ```

7. Add your README.md from Sec. 1 to the repo and push it to GitHub. You should add any answers for this section to the file as well.

8. Write a node that makes the turtle follow a reference trajectory given by

   $$x_d(t) = 3\sin\left(\frac{4\pi t}{T}\right) \qquad y_d(t) = 3\sin\left(\frac{2\pi t}{T}\right) \qquad t \in [0, T].$$

   This is a vertically-oriented "figure 8" centered at the origin. The turtle should complete one traversal of the reference in T seconds. Don't worry if your figure 8 isn't oriented perfectly. Here are a few hints:

   - You will want to call the `turtle1/teleport_absolute` service to get the turtle located and oriented correctly for the $t = 0$ case.
   - The turtle essentially follows the standard "kinematic-car" model (sometimes called the "unicycle model"). Which means only two of the values in the `cmd_vel` topic are actually used when integrating the turtle's kinematics.
   - The turtle is what's called a "differentially flat" system. For our purposes, this means that given the reference above, we can analytically calculate the translational and rotational velocities to command the turtle with to exactly follow the reference.
   - There are well-known algorithms for generating the rotational and translational velocity signals required to get this kinematic car model to perfectly follow the given reference trajectory. Feel free to use algorithms that you find in research papers, but be sure to cite your source.

9. Make your node accept a **private parameter** (not a command-line argument) that can be used to override the default value for T (the user can then specify at runtime how fast the turtle should go).

10. Use `rosbag` to record enough messages on the `/turtle1/cmd_vel` topic to have the turtle complete one circuit of the reference. Note that the `-l` option to `rosbag` can be used to control how many messages are recorded. You can then use `rosbag` to play back your recorded messages to control the turtle without ever starting up your node. **Be sure to include a copy your bag file in whatever you turn in.**

11. Add a high-level description of the code that you wrote to your README. You should be sure to describe the which Python script contains the node (with links), the ROS topics/services that this node provides/uses, and point out any critical lines of code.

12. Push all changes to your GitHub private repo created by GitHub Classroom, and be sure to submit the URL of your repo before the due date.