Assigned: January 19, 2024
Due: January 30, 2024

Intro to Robot Programming
ECSE 4965, MANE 4961

Lab # 2

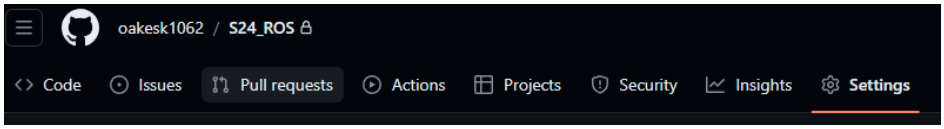
Labs are expected to be completed by each team member. Collaboration is encouraged. The deliverable will be a single report with proof of each member's completion of the tasks in the appendices.

Note that there are lot of commands being executed in the terminal. Recall some Linux tricks and commands that might be helpful: tab complete, `clear`, etc.

Task 1 - Setup a Github Repository

***Only one team member needs to do steps 1-3. They will be the repository "owner".

1. Create a team repository on Github. Name it S24_RobotProgramming_*teamName*.
2. Add all of the team members as collaborators. This is done by navigating to the new repository, selecting settings (as shown in the image). Then on the left menu select 'Collaborators' and add using Github usernames/emails. Please also add myself (username: oakesk1062) and the TA (username: eric565648).



3. Modify the README file to include all team members' names.

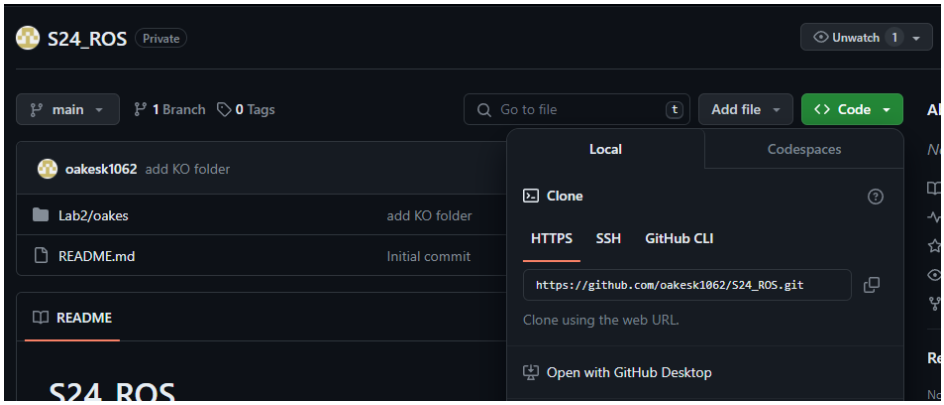
   ***The following should be completed by each team member.

4. On your virtual machine (VM), open a terminal. If you have not already done so, execute the following lines:

   - `git config --global user.name "`*`your name`*`"`
   - `git config --global user.email "`*`your github email`*`"`

   to set up your git user information.

5. Navigate to where you would like the repository folder to be (for example, the Desktop). Type: `git clone` *website* to make a local copy of the repository. The website is found on Github by clicking the green 'Code' button on the repository main page and copying from the https tab.



6. cd into this repository folder and create a folder for you within a folder called Lab2.
   - `mkdir ./Lab2/`*lastName*
   - The ./ tells the command to make this directory within the current directory.
7. Create a new text file called "name.txt" within your lab 2 folder. Add your full name and Github username to this file and save it.
8. We need to add, commit, and push this change to the main repository.
   - `git add ./Lab2/`*lastName*`/name.txt` (True if in the main repository folder.)
   - `git commit -m "Add Lab 2 folder for` *name*`"`
   - `git push origin main` (See Github Notes section for some details.)

   ***Step 8 must be completed each time you need to add files or sync the changes with the repository.
   Note: you should also execute `git pull` before starting to change files to ensure you have the latest repository version.

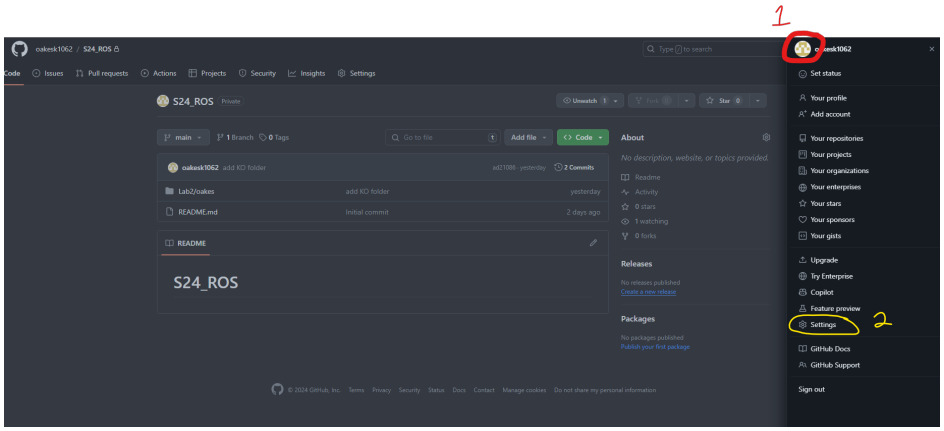
Github Notes

Note: When when pushing your changes, you will need to provide log in information.

- For the username prompt, provide your Github username.
- For the password, provide the token value from Github.

Setting up tokens.

- Select the icon in the top right of the Github site.



- Scroll down the left menu and select 'Developer Settings'.
- On the left menu, select Personal access tokens > Tokens (classic).
- Select Generate new token in the middle-right of the screen. Choose Generate new token (classic).
- Select the expiration and scopes for this token and give it a descriptive name in the Note section if desired.
- Click Generate token at the bottom. Save this somewhere or else you will need to generate a new token each time you push commits to the repository.

Task 2 - Introducing ROS2 Commands

We will get familiar with some ROS2 commands, ROS components, and node interfaces using the package turtlesim (already installed on the VM).

Starting Nodes

1. **Action:** See which nodes we can execute within this package:
   `ros2 pkg executables turtlesim`

2. **Question:** What are the executable nodes within the turtlesim package?
3. **Action:** Start a simulated turtle
   `ros2 run turtlesim turtlesim_node`
4. **Question:** What is the name and starting pose of this simulated turtle?
5. **Note:** Notice that the terminal did not return to a yahboom@VM line for us to enter commands. That is because the turtle simulation process is actively using this terminal. Therefore, we must leave this terminal open to interact with the simulation.
6. **Note:** You can right-click on windows and select 'Always on top' to make sure they stay visible.
7. **Action:** Open a new terminal. Position the terminals and simulation window such that you can see all three simultaneously. Start a node to interact with the turtle.
   `ros2 run turtlesim turtle_teleop_key`
8. **Note:** The teleop terminal must be the active terminal to interact with the turtle.
9. **Question:** What happens in the turtlesim window if you try to drive the turtle out of the window?

Exploring the Components

1. **Action:** Open another terminal and type `ros2 node list`
2. **Question:** How many nodes are running?

Topics

1. **Action:** Type `ros2 node info /turtlesim`
2. **Note:** Nodes can subscribe or publish to topics. Under the Subscribers/Publishers headings are lists of the form -> topic: message type
3. **Question:** Based on this information, for moving and tracking the simulated turtle, what information do you think this node sends and receives? What are the associated topic names?

4. **Action:** Type `ros2 node info /teleop_turtle`
5. **Question:** Based on this information, for moving and tracking the simulated turtle, what information do you think this node sends and receives? What are the associated topic names?
6. **Note:** Let's check this out some more.
7. **Action:** Type `rqt_graph`. After the window opens, in the drop-down menu on the left select 'Nodes/Topics (all)' and ensure all the check boxes are selected except 'Dead Sinks' and 'Leaf Topics'.
8. **Note:** You can hover over the check box labels for more info. Same with the blocks/lines in the image.
9. **Question:** Do the arrows match the publisher/subscriber information from the previous steps?
10. **Action:** Close the graph and back in the terminal type `ros2 topic list`
11. **Question:** What is the difference between this output and that of `ros2 topic list -t`
12. **Action:** `ros2 topic info /turtle1/cmd_vel`
13. **Question:** What does this command tell us?
14. **Note:** What does this message type actually mean? The message tells the nodes how information is sent to/received from the topic. Let's dig in some more.
15. **Action:** Type `ros2 interface show geometry_msgs/msg/Twist`
16. **Action:** Type `ros2 topic echo /turtle1/cmd_vel`
17. **Question:** What happened?
18. **Action:** Ensure the this terminal and the teleop terminal are visible. Select the teleop terminal and press an arrow key.
19. **Question:** What happened in the terminal we have recently been using? Is this what was expected based on the result from interface show?
20. **Action:** Go back to the echoing terminal and use Ctrl+C to end the process.
21. **Action:** use `topic info`, `interface show`, and `topic echo` to find similar information for the topic `/turtle1/pose`

22. **Question:** With the pose topic notice that with echo we are getting continuous data. At what rate is the simulator publishing? (Hint: `topic hz`)
23. **Action:** Instead of using the teleop window, we can directly publish to the `/turtle1/cmd_vel` topic. Type
    `ros2 topic pub --once /turtle1/cmd_vel geometry_msgs/msg/Twist "{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0, y: 0.0, z: 1.8}}"`
24. **Note:** Spacing in the previous command matters.
25. **Action:** Try changing these numbers and performing some executions.
26. **Question:** What do the x,y,z refer to?
27. **Note:** You can also send commands at a rate. Replace `--once` with `--rate #`, where # is the rate you want to publish at.

Services

1. **Action:** Type `ros2 service list`
2. **Question:** How can I see the service type for the service `/clear`? (Hint: There are two ways. Something we can add to the previous command or another command where the service name is provided to the command.)
3. **Question:** Take a look at the interface for this service type. What do you think this implies?
4. **Action:** Type `ros2 service call /clear` *`serviceType`* (where you get to fill in the service type)
5. **Question:** What happened?
6. **Challenge:** Based on what we have done so far, change the turtle line color to red.
   Some hints:
   - What is the interface for `/turtle1/set_pen`?
   - The `- - -` separates the two data exchanges associated with a service. The first part is the format used to call the service. The second is form of the response from the service.
   - When calling a service (similar to publishing to a topic), we have a command of the form
     `ros2 service call` *`serviceName serviceType data`*

- The data uses YAML format. So we provide the data for this service as "{*attributeName*: *attributeValue*, *nextName*: *nextValue*}" Note that if an attribute name is more than one character, you must surround the name with single quotes so that it is a string data type.

Drive around the turtle using the teleop terminal to see if it worked!

7. **Note:** Note that we called the server part of this service. We can confirm this by looking at the node info for turtlesim again. Note that the set pen service is under the 'Service Servers' list. That means our terminal that called the service was a service client.

Actions

1. **Question:** Generate a list of available actions.
2. **Action:** Check the info for each node again.
3. **Question:** Which node is the server and which the client for this action?
4. **Note:** There is another way to get this information.
   Type `ros2 action info` *actionName*
   It lists the clients and servers for you.
5. **Action:** Let's look at the interface for this action.
   Type `ros2 interface show` *actionType*
   - **Note:** What is the action type? Let's use the help functions to find a way to get this.
   - **Action:** Type `ros2 action --help`
   - **Note:** We can see there is no 'type' command like we had for topics and services. Maybe there is something in info?
   - **Action:** Type `ros2 action info -h`
   - **Note:** There is an optional argument that we can use with this command to show the type! Give it a try.
6. **Note:** The lines starting with # are comments. The - - - separate the sections for an action. Part 1 is the goal format. The middle is the result that is received when the action server terminates. The bottom is the format of the feedback provided by the server while executing the goal.

7. **Action:** Let's send a goal to this action.

```
ros2 action send_goal /turtle1/rotate_absolute turtlesim/action/RotateAbsolute "{theta: 1.57}"
```

8. **Action:** Rerun this command with the addition of `--feedback` at the end to see the feedback.

Task 3 - Application of ROS2 Commands

1. **Action:** Using what we have learned, draw a black square on the screen.
2. **Question:** There are multiple ways to do this. Explain your method.

Lab 2 Deliverable

Individual completion proofs should be provided in the Github repository.

- Proof should be clear by adding the instructor to the repository with all members folders containing the appropriate files. Note the commits show who made them so we should see that everyone submitted their own changes.
- For starting nodes: Provide a screenshot showing the multiple terminals.
- For topics: Provide screenshots for steps 7, 21, and 25.
- For services: Provide screenshot for the Challenge.
- For action: Execute step 8 with a unique goal position.
- For task 3: Provide a final image and a text file with a brief explanation.

Please label your files as Task#_ followed by the subsection if applicable _step#

Extended Task 1 - Application of ROS2 Commands

1. **Action:** Draw a yellow star on the screen using the turtle.