ECED 3901

Laboratory 2

Part A: Git version control

Part B: Gazebo simulation

Date: January 14, 2026

Version: 1.2

V. Sieben

Optional reading after completion of lab:

https://git-scm.com/book/en/v2

https://guides.github.com/activities/hello-world/

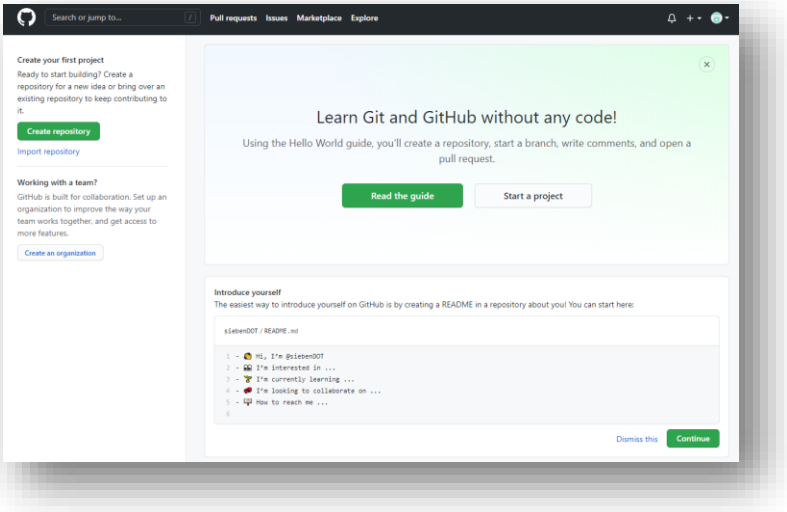
https://thenewstack.io/dont-mess-with-the-master-working-with-branches-in-git-and-github/

https://docs.ros.org/en/humble/Tutorials.html

Part A – Git version control

A.1. Create a GitHub USER accounts (All Team Members)

1. Visit www.github.com and click “sign up” if you need an account.
2. Enter your username, email, password.
3. Complete the setup, and verify your account with the email sent.
4. Click “Skip this for now” until the site resembles the following.



A.2. Creating the REMOTE Repository (Only on 1 Team Member's account)

1. Now that everyone in the Team has GitHub accounts, ONLY ONE team member will make the Team's repository as described below. We will add the others after.

2. Create your Team's Remote "NEW" repository

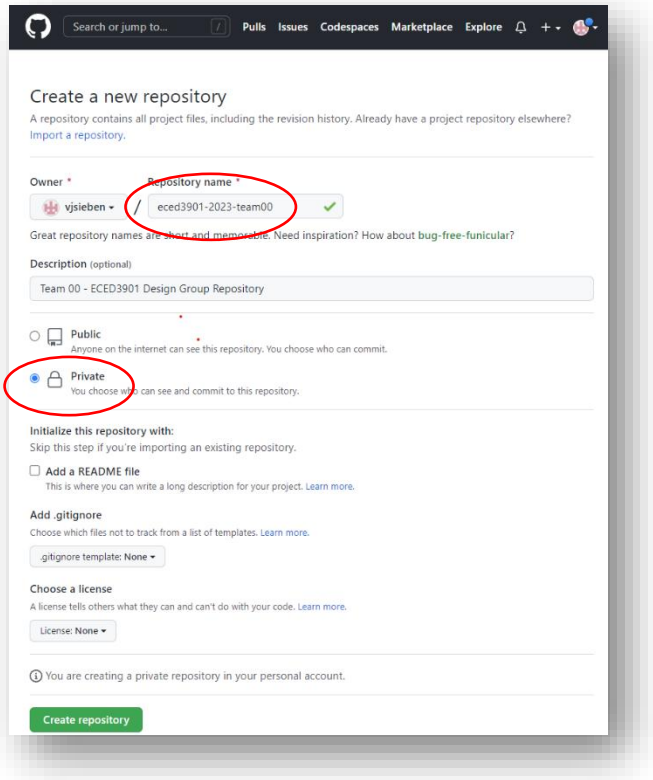
    a. Type in the name, please make it follow this syntax **EXACTLY**:

    eced3901-2026-team## //where ## is your team number (01, 02, … 20)

    b. Enter a description.

    c. **Make it private**.

    d. Click "Create repository"



3. Add the **other team members** to your repository.
    a. Make sure you are logged into GitHub and navigate to your team's repository, created above.

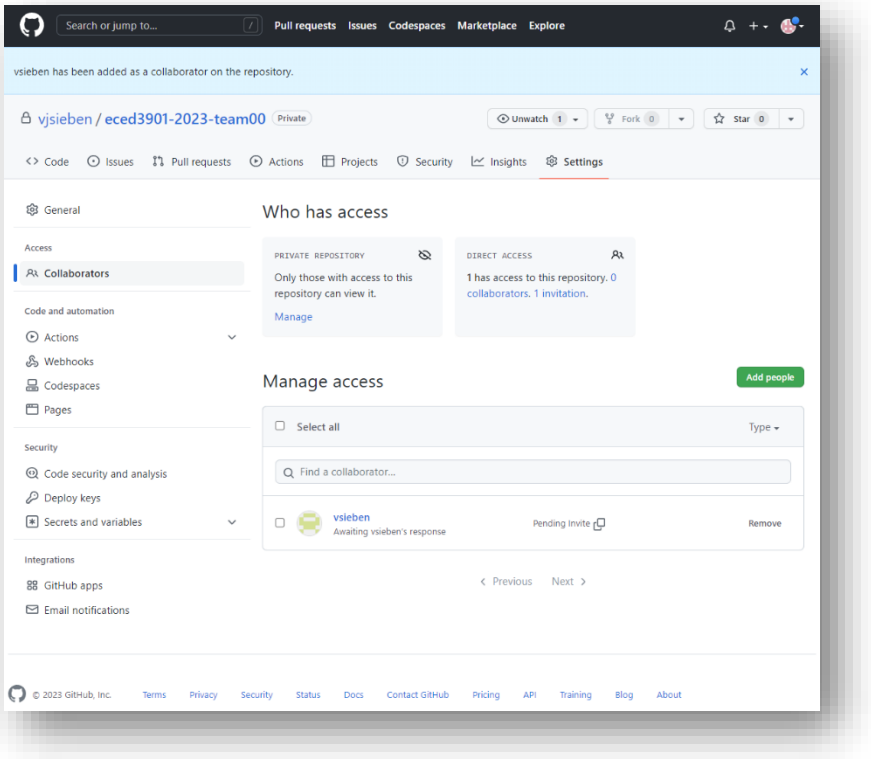
    b. Click on the "Settings" tab.

c. Click on the "Collaborators" left menu item.

d. Click on the "Add people" button, in the manage access box.

e. Type in your team member usernames from the steps above, select, and finally click "Add XXXXX to this repository."

After you have added a person, they should show up on the access list.



You are now finished with your GitHub portion of the laboratory, which sets up the REMOTE repository. From here on, we will be minimally interacting with web-portal. **We will be using command lines to access** this repository.

A.3. Creating the LOCAL Repository (All Team Members on 1 Robot)

1. Start your Robot Ubuntu installation.

2. Open a new terminal; press and hold “ctrl + alt + t”

3. Configure Git global settings by typing the following commands (ignore lines starting with #, they are comments). Replace red text with details (e.g. Robot #)

```
# Default text editor
$ git config --global core.editor nano

# Name
$ git config --global user.name "ECED3901 RobotNumber"

# Email
$ git config --global user.email "email@dal.ca"
```

4. Check Git configuration is set correctly.

   ```
   git config --list
   ```

   Note: If you install the VM on your own laptop to design and do code commits, please use your own name as it will add the correct contribution information to the repo logs and records (marked later).

5. Move to the directory of the package.

```
$ cd ~/ros2_ws/src/eced3901
```

6. **Initialize Git folder.** Should be in ~/ros2_ws/src/eced3901

```
$ git init
```

   Typing “**git status**” should show you that there are ‘red’ untracked files that could be included in the version control.

7. **Stage files for first commit**. Here we add all files in the repository folder.

```
$ git add .
```

Typing "**git status**" should show you that there are 'green' uncommitted files that could be committed version (with a unique hash id).

8. **Commit the files**. The first entry point of the master branch. Note that we are still only making changes to the local repository (i.e. on the hard drive of the desktop PC).

```
$ git commit -m "Initial commit after package creation with files provided"
```

Typing "git status" should show you that there is "nothing to commit", you have now made a discrete "version."

You can use the following command to check your "repo" history.

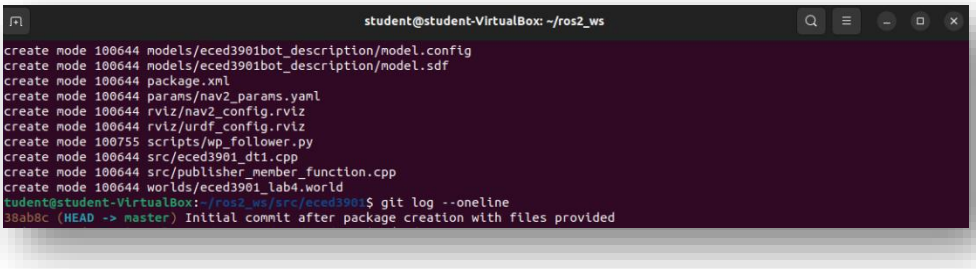
git log --oneline

git log --pretty=format:"%h%x09%an%x09%ad%x09%s"

or

git log --pretty=format:"%h%x09%an%x09%ad%x09%s" --date=short

In case you were curious what the different options were:
%h = abbreviated commit hash
%x09 = tab (character for code 9)
%an = author name
%ad = author date (format respects --date= option)
%s = subject



A.4. Running and Changing the Publisher Node

1. We **compile** the code.

```
$ cd ~/ros2_ws
$ colcon build
```

2. We **run** the code.

```
$ ros2 run eced3901 talker
```

   After you have enjoyed reading "hello world" enough, press and hold "ctrl + c" to kill the talker program, or ROS2 node.

3. Change the message to something else you want in the code, recompile, and run. (this step requires your own effort to complete and will take time, so don't worry! i.e. self-directed)

   Get Source File Location:
   $ cd ~/ros2_ws/src/eced3901/src

   Open file:
   $ nano publisher_member_function.cpp

   Save, exit, rebuild in the "~/ros2_ws" directory, rerun.

4. The code with new message is ready to share with our team members. Please **stage and commit the changes** (git add and git commit) to the LOCAL repository.

A.5. Pushing LOCAL repository to REMOTE repository (ROBOT to Web)

1. We will be pushing the updated local repository to the remote repository that all team members or collaborators can access/download.

   On the Robot, type the following using **your team's GitHub MAIN USER ACCOUNT and destination repository (From A.2 above).** This creates a symbolic link between "origin" and the specified URL so that we do not have to type it again and again.

   In the repo folder, **add** the origin link:

   ```
   $ cd ~/ros2_ws/src/eced3901
   $ git remote add origin git@github.com:useraccount/eced3901-2026-team##.git
   ```

   IF you have to **remove** the origin link:

   ```
   $ git remote remove origin
   ```

   IF you have to **replace** the origin link:

   ```
   $ git remote set-url origin git@github.com:useraccount/eced3901-2026-team##.git
   ```

   "git config --list" will show you the configuration.

2. Before we upload anything, we will be prompted for a username and password. However, instead of a password, you will have to use a **Deploy Key** for GitHub.

   First, we need to make our key pair on the **robot**. To do so, open a terminal and type the following:

   ```
   $ ssh-keygen -t ed25519 -C "student@dal.ca"
   ```

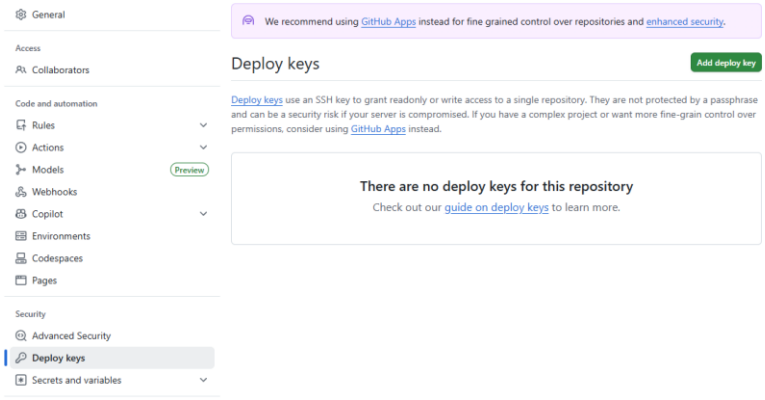
   This will prompt you for the File in which to save

   i. Just hit "enter" to accept the defaults
   ii. A passphrase (or hit "enter" for no passphrase). I suggest using a simple passphrase the entire team knows to ensure someone else with access to the robot cannot access your repository.

HINT: If you ever forget your passphrase, simply rerun the key generation. It will prompt you to overwrite the key, and say “yes”. You will have to follow the steps below to add your new key & you can delete the old one.

Second, we need to deploy our key, go to the **github website**:

a. Go to your git repo. (https://github.com/**useraccount**/eced3901-2026-team**##**.git) and hit the settings link.
b. On the left menu at the bottom, click “Deploy keys”
c. Click the “Add deploy key” button.



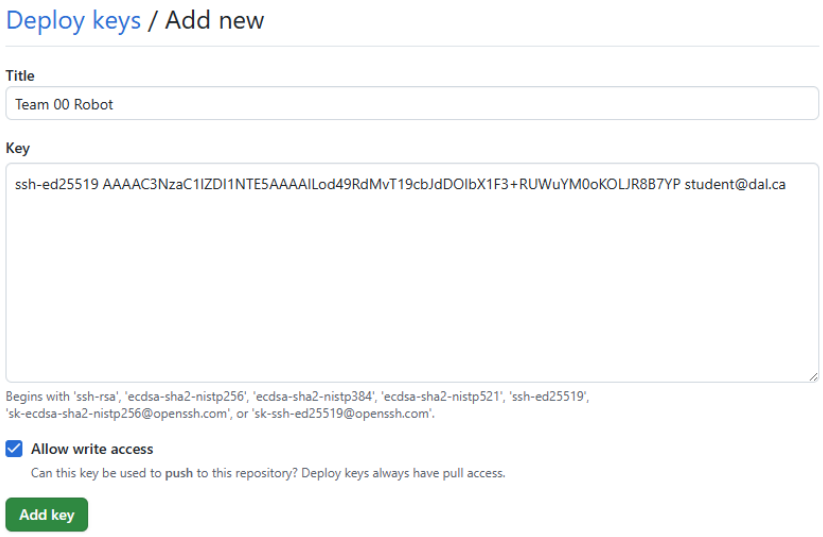
d. You will need to print your “public” key to copy into the website. Open a terminal on the **robot** and type the following:

```
$ cat /home/student/.ssh/id_ed25519.pub
```

This should return a string that resembles:

ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAILod49RdMvT19cbJdDOIbX1F3+RU
WuYM0oKOLJR8B7YP student@dal.ca

e. Enter the key along with a title into the **github website** form. Check the “Allow Write Access” button.



f. Press the “Add Key” button.

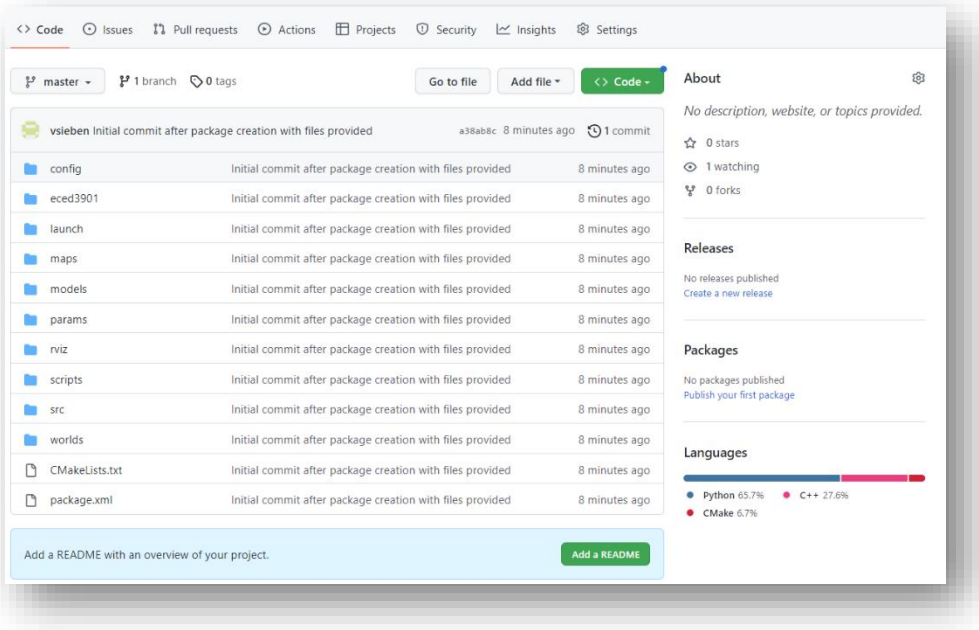
3. PUSH the **local** repo to the **remote** repo location. Here we upload the code to the remote repository in GitHub. If you set a password for your key, it is your passphrase for your key set above. If you get prompted for a GITHub username/password, check your origin is set to ssh and NOT https!

```
$ git push -u origin master
```

You might be prompted regarding the authenticity of host, say “yes” to connect. Everything should have worked and a message like below should appear.

```
student@student-VirtualBox:~/ros2_ws/src/eced3901$ git push -u origin master
The authenticity of host 'github.com (140.82.112.4)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wvvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvCOqU.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
Enumerating objects: 44, done.
Counting objects: 100% (44/44), done.
Delta compression using up to 4 threads
Compressing objects: 100% (38/38), done.
Writing objects: 100% (44/44), 24.85 KiB | 3.11 MiB/s, done.
Total 44 (delta 9), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (9/9), done.
To github.com:siebenECED3901/eced3901-2026-team00.git
 * [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

When viewed through the GitHub web portal, the repository should resemble this:



After the first time, you can use

```
$ git push
```

A.6. Pulling the Team's Code, Robot Computer or VMs

IMPORTANT: You DO NOT need to run the following commands by default. The following commands are only needed if you wanted to set up your workspace on another computer.

1. The other team members can now download the repository into their workspace source folder, compile the code, and run it.

2. To demo, remove the old package that comes with the default course image

```
# Move to your workspace
$ cd ~/ros2_ws

# Delete the build and install files
$ sudo rm -r build install log

# Move to your workspace source folder
$ cd src

# Delete the source files
$ sudo rm -r eced3901
```

3. Now that all files are gone, we need to clone the team's ROS2 package back into your src folder.

```
# Clone package
$ git clone git@github.com:username/eced3901-2026-team##.git eced3901
```

4. With the code cloned from the repository, we need to compile/make, including generation of the many ROS2 symbolic links to the package.

```
$ cd ~/ros2_ws
$ colcon build
$ colcon build
```

5. Run the talker that Team member 1 created!

```
$ ros2 run eced3901 talker
```

Congratulations, you can pull code from a shared repository. Later, please explore “branch”, “checkout”, and “merge” commands, if you want to utilize the full potential of GIT. And remember: **Always PULL before you start to change code, which will ensure you are using the latest version of team code!**

Messed something up?

If you want to wipe away changes to a local file, you can specifically check out that file. This can be helpful when you modified a file locally and just want to go back to how things were for this file:

```
$ git checkout dir/somefile.py
```

This can be helpful when a teammate has committed a fixed version of a file, and you no longer need your local changes.

Part B: Gazebo simulator

We start by exploring a pre-canned world.

1. Start your Robot or VM.

2. Open 3 terminals, press "**ctrl + alt + t**" for a window, or **"ctrl + shft + t"** for a tab.

3. Start Gazebo: use the launch file below. This will create the simulation world, launch several nodes, and it will place a virtual DALIBOT robot in the environment.

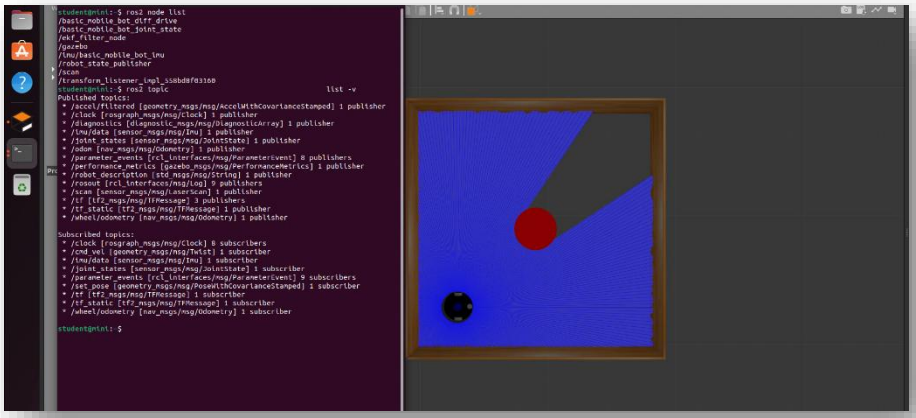
```
# Terminal 1:
$ ros2 launch eced3901 lab5.launch.py
```

4. Use the "**ros2 node list**" and the "**ros2 topic list**" commands to show the current ROS2 nodes and ROS2 topics available.

```
# Terminal 2:
$ ros2 node list
$ ros2 topic list -v
```

You should see something that resembles the image below. There are three very important topics for your design course efforts:

1) **/cmd_vel** -> the topic which directs robot motor control; the robot will listen to commands for controlling the motor velocities.
2) **/scan** -> the topic which returns laser sensor feedback from the spinning 360° range finder, or LiDAR.
3) **/odom** -> the topic which estimates the robot's position using dead reckoning, or odometry, based on encoder ticks + IMU data, coupled with the Differential Drive package for calculating position estimates.



These topics are “virtual”, in that the robot does not actually exist. Therefore, the Gazebo simulation is creating them to emulate the robot sensors, actuators and physical interaction within the world loaded.

For your Design Task(s), the simulated topics will be replaced by versions generated from the real robot. During your team’s personal video runs and marked trial runs, these topics are your interfaces to take in sensor data from the robot and control it to achieve the course goals. Therefore, the Gazebo simulator allows you to test your code before experimental runs. This saves battery life and trial & error iterations. However, like all simulators, it is not perfect. You may need to iterate your design code a few times in experimental settings. This is normal and expected, but if you do not simulate beforehand, it will be slow and difficult to succeed in the design tasks.

5. Let’s start the teleoperation package in terminal 3. Execute the following command, but do not use the keyboard to move the robot just yet.

   Tip: use “Tab” on your keyboard to autocomplete typing of commands. If you type “ros2 run tel” and then push “Tab” on your keyboard, it will attempt to autocomplete based on symbolic links.

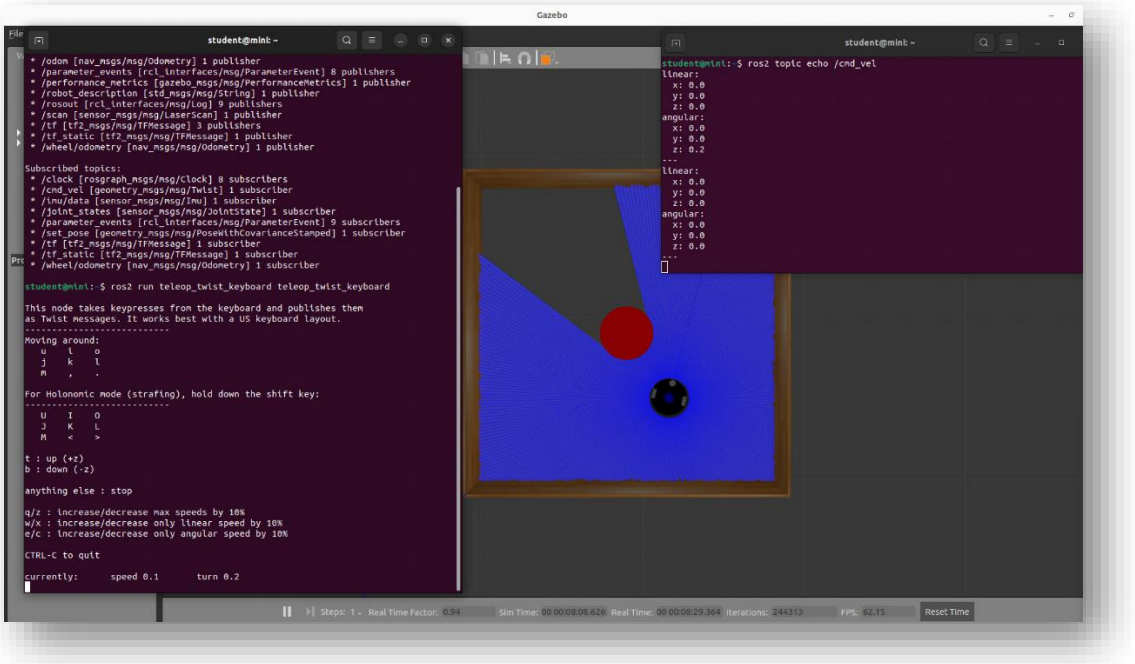
```
# Terminal 3:
$ ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

   This executes a python program, which establishes a ROS2 node that publishes to the ROS2 topic /cmd_vel. After the lab, it might be worth your time to investigate how this is accomplished. The “ros2 run” command is used to directly create a node without configuration options in *.launch files. If you go back to Terminal 2 and retype “ros2 node list” you should see the /teleop_twist_keyboard appear.

6. Use the “ros2 topic echo” command to observe one of the topics. In this lab, we chose /cmd_vel, but you can echo most topics. Be aware that sensors that produce vast datasets (LiDARs, cameras, etc.) are not as easy to troubleshoot with this command.

```
# Terminal 2:
$ ros2 topic echo /cmd_vel
```

7. If you push “j” in terminal 3, it will instruct the robot to rotate on the spot counterclockwise at 0.2 radian per second. You should see the topic in terminal 2 update with the velocity command now issued. The robot should also be spinning in Gazebo, your screen might look like below. The mouse scroll wheel can be used to zoom in to inspect your robot in Gazebo.



The angular velocity should read “z: 0.2” in the terminal 2 echo of the ros2 topic /cmd_vel.

Try and change the rotational speed of the robot in terminal 3 and confirm that the robot rotation in Gazebo changes and that you can see it in terminal 2’s feedback. Similarly, alter the translational velocity and observe the behavior. More information on /cmd_vel can be found using the ROS API descriptions:

https://docs.ros.org/en/api/geometry_msgs/html/msg/Twist.html

Tip: for smooth non-jerky motion, please keep the real robot under 1.0 rad/s for rotational or angular velocity (“z”), and, under 0.2 m/s for translational or forward/reverse velocity (“x”).

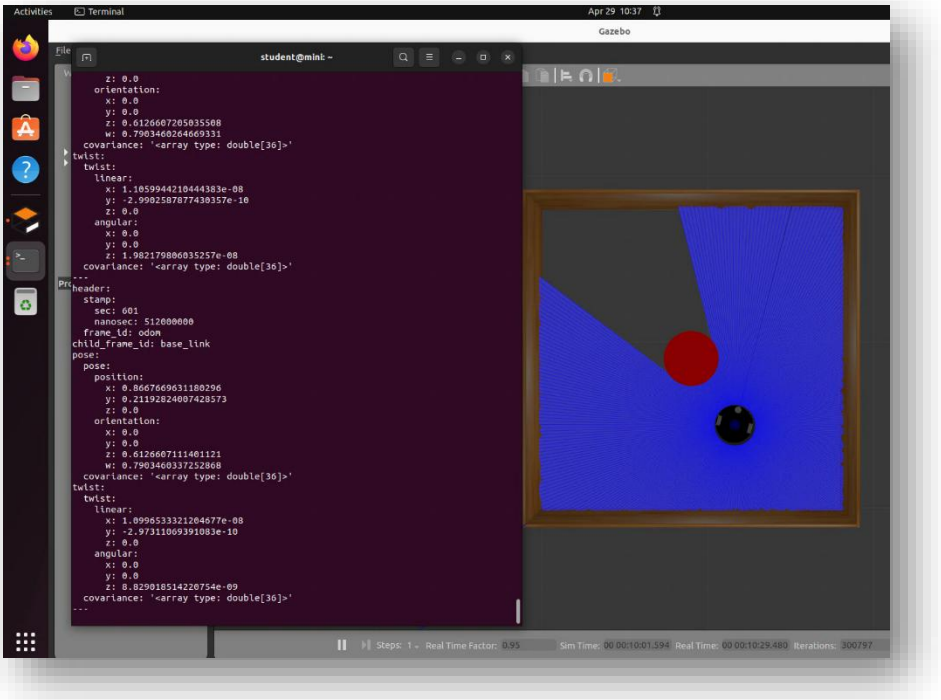
Push “k” in terminal 3 to stop the robot motion.

8. Press “ctrl + c” in terminal 2 to stop the ros2 topic echo command.

9. Use the “ros2 topic echo” command to observe the /odom topic. This should generate lots of data and your terminal will need to be made larger. There are a number of ways to enlarge the terminal, but dragging the window using the title bar to the left/right edge of the screen will automatically snap the window to half the screen size.

```
# Terminal 2, to display odom with no arrays:
$ ros2 topic echo /odom --no-arr

# OR, to display a specific field:
$ ros2 topic echo /odom --field pose.pose.position
```



The /odom topic is comprised of a header (time, frame names, etc.) and two messages, “pose” and “twist”, which estimate the position and velocity of the robot in free space, respectively. Each message also includes a covariance matrix.

The pose message includes the pose position in cartesian x, y and z values (meters). For ECED3901, only the x and y values are required. The pose

orientation is listed with 4 variable **Quaternion notation** (x, y, z, w), instead of the more intuitive 3 variable **Euler angle notation** (roll, pitch, yaw). You will need to use transforms to extract the relevant angle for ECED3901, or yaw (radians). As mentioned in the lectures, the position state-space for the robot can be represented as (x, y, theta - or yaw), extracted from the /odom topic.

The twist message includes the x, y and z translational velocities (m/s). As the ECED3901 DALiBOT robot is a differential drive robot, it will only require an "x" translational velocity to indicate forward/reverse. The "y" component is typically used for holonomic or strafing robots with more degrees of freedom, and the "z" component is typically used for drones that can actuate in 3D space. Lastly, the angular rotational velocities are shown about the x, y and z axis. Since we are in the x-y plane with a differential drive robot, only the rotational velocity about the z-axis (yaw) is of relevance. Otherwise, our robot would be doing barrel rolls, or taking off the ground, or burying itself into the ground! Summarily, the velocity of our robot is captured by linear velocity x and rotational velocity z.

10. Set the linear velocity to ~0.2 by using w/x on the keyboard in terminal 3.

11. Set the angular velocity to ~1.0 by using e/c on the keyboard in terminal 3.

12. Then, with parameters set and with knowledge of the /odom message descriptions, take some time and navigate to another area of the map. Take note of the change in position and velocity on terminal 2's listing of the /odom topic.

    Tip: It is easier to first adjust the robot to the desired heading by pushing "j/l" for CCW/CW rotation, then stopping the robot by pushing "k". With a correct heading, you can then push "i or <" to move forward/backward, pushing "k "to stop again.

13. Press "**ctrl + c**" in all terminals.

Congratulations, you now are using the simulator and teleoperation manual control!