Lab 1 – Setup and Introduction to ROS
100 points

Overview:
The purpose of this lab is to get the ROS system we will be using installed and working on your computer. You will be working through some ROS tutorials and getting to know the ROS environment. To complete the lab you must submit via Canvas a zip file containing your source code and a video recording of your program execution.

Installation Instructions:
In this class we will use ROS 2 Humble. ROS 2 Humble is supported on Ubuntu 22.04. [Other versions of ROS will require other versions of Ubuntu, make sure that you check the version compatibility if you decide to try a different ROS version, or if you have a more recent Ubuntu version]. The labs may work with other versions of ROS and Ubuntu as well but this is not guaranteed.

Part I: Learning ROS

We will be working with ROS Humble this semester. To get it installed, go to

https://docs.ros.org/en/humble/Installation.html#

Once you have completed your ROS installation, please work through the Beginner Level Core ROS tutorials. The tutorials will involve configuring your ROS environment, creating a workspace, and creating a package. We will also be primarily using Python this semester, so you may focus on the Python versions of the tutorials. Make sure you complete the following tutorials:

https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools.html
https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries.html
https://docs.ros.org/en/humble/Tutorials/Intermediate/Launch/Launch-Main.html

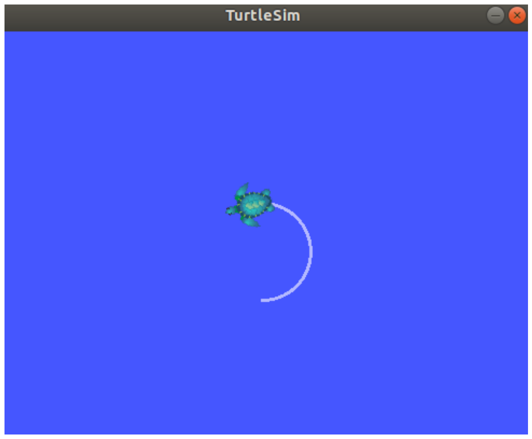
(Important) Make sure that you have gone through the steps of installing `colcon` and creating a workspace before moving on to Part II.

Part II: Using ROS

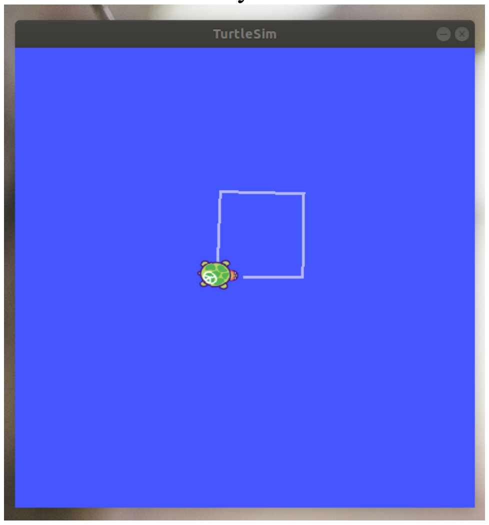
1. First, we will create a new package to house the code we will work on in class.
   - Open a new terminal and change the active directory (cd) to ~/ros2_ws/src
   - In ~/ros2_ws/src type the following (all on one line) and then press enter
     - ros2 pkg create --build-type ament_python ai_labs
   - Unzip Lab 1 files from Canvas into the new ai_labs folder
     - unzip lab1.zip -d ~/ros2_ws/src
   - From the root directory of your ROS workspace, run `colcon build` and then run `source ~/ros2_ws/install/setup.bash` to finish building the package
   - After you have completed these steps, you should have the following directory structure:
     - ros2_ws
       - build
       - install
       - src
         - ai_labs
           - package.xml
           - setup.cfg

- setup.py
- ai_labs
  - <python scripts for labs>

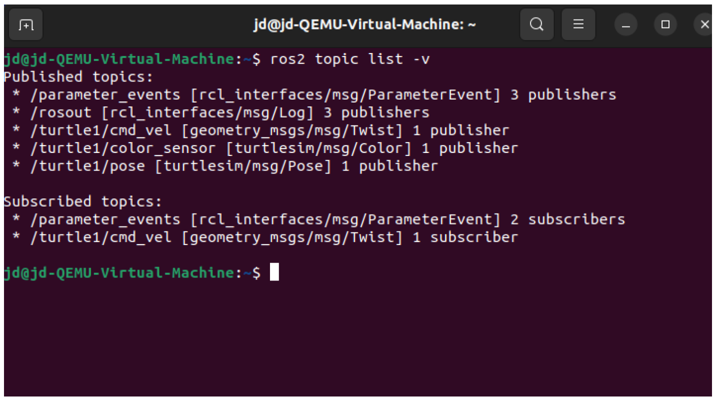
2. Run the following command from the terminal. You should see the simulator window pop up and a robot driving in a circle.
   - `ros2 launch ai_labs  lab1_launch.py`
   - Instead of using a launch file, you may also run the following commands from two separate terminals:
     i. `ros2 run turtlesim turtlesim_node`
     ii. `ros2 run ai_labs square`
   - If it doesn't work, try to run `colcon build` from the ros2_ws folder or run `source ~/ros2_ws/install/setup.bash`



3. Modify the code in square.py so that the robot moves in a square. (70 pts)
   - Note that the code sends a Twist command to the robot, then sleeps, awaking 10 times per second (this rate can be changed) to send a new command). To understand Twist commands, please see the appendix below.
   - Remember to run `colcon build` every time after the code is modified



4. Use the *ros2 topic list -v* command to print out the list of ROS topics. (15 pts)

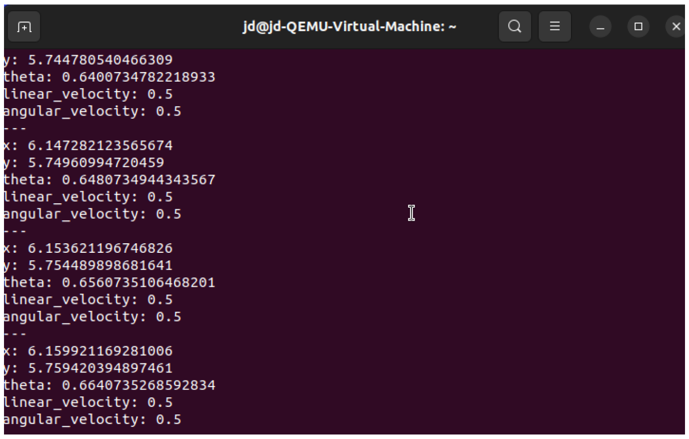


```
jd@jd-QEMU-Virtual-Machine: ~
jd@jd-QEMU-Virtual-Machine:~$ ros2 topic list -v
Published topics:
 * /parameter_events [rcl_interfaces/msg/ParameterEvent] 3 publishers
 * /rosout [rcl_interfaces/msg/Log] 3 publishers
 * /turtle1/cmd_vel [geometry_msgs/msg/Twist] 1 publisher
 * /turtle1/color_sensor [turtlesim/msg/Color] 1 publisher
 * /turtle1/pose [turtlesim/msg/Pose] 1 publisher

Subscribed topics:
 * /parameter_events [rcl_interfaces/msg/ParameterEvent] 2 subscribers
 * /turtle1/cmd_vel [geometry_msgs/msg/Twist] 1 subscriber

jd@jd-QEMU-Virtual-Machine:~$ 
```

5. Use the *ros2 topic echo <topic_name>* command to print out the robot pose at each step. (15 pts)

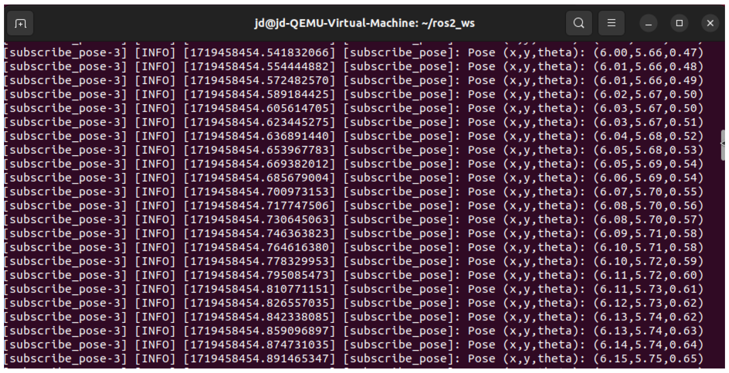


```
jd@jd-QEMU-Virtual-Machine: ~
y: 5.744780540466309
theta: 0.6400734782218933
linear_velocity: 0.5
angular_velocity: 0.5
---
x: 6.147282123565674
y: 5.74960994720459
theta: 0.6480734944343567
linear_velocity: 0.5
angular_velocity: 0.5
---
x: 6.153621196746826
y: 5.754489898681641
theta: 0.6560735106468201
linear_velocity: 0.5
angular_velocity: 0.5
---
x: 6.159921169281006
y: 5.759420394897461
theta: 0.6640735268592834
linear_velocity: 0.5
angular_velocity: 0.5
```

Part III: Extra Credit

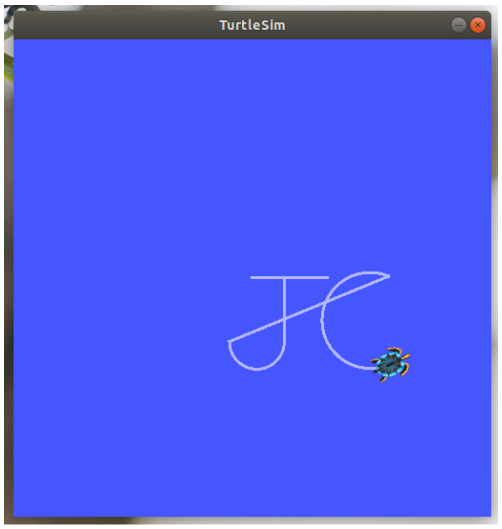
1. Create a new Python script named "*subscribe_pose.py*" that implements a simple subscriber to print out the robot pose (10pts). Refer to http://docs.ros.org/en/melodic/api/turtlesim/html/msg/Pose.html to find out the components of a Pose message. The robot pose is published on the topic "*turtle1/pose*". Use the following format:

   Pose (x,y,theta): (8.82,3.59,0.48)



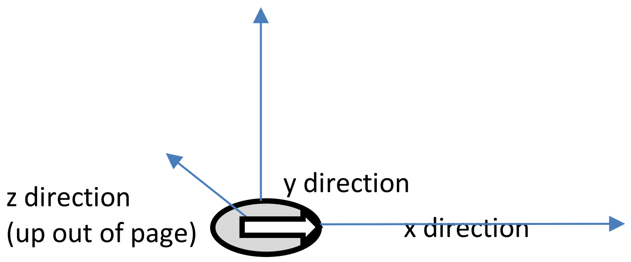
```
jd@jd-QEMU-Virtual-Machine: ~/ros2_ws
[subscribe_pose-3] [INFO] [1719458454.541832066] [subscribe_pose]: Pose (x,y,theta): (6.00,5.66,0.47)
[subscribe_pose-3] [INFO] [1719458454.554444882] [subscribe_pose]: Pose (x,y,theta): (6.01,5.66,0.48)
[subscribe_pose-3] [INFO] [1719458454.572482570] [subscribe_pose]: Pose (x,y,theta): (6.01,5.66,0.49)
[subscribe_pose-3] [INFO] [1719458454.589184425] [subscribe_pose]: Pose (x,y,theta): (6.02,5.67,0.50)
[subscribe_pose-3] [INFO] [1719458454.605614705] [subscribe_pose]: Pose (x,y,theta): (6.03,5.67,0.50)
[subscribe_pose-3] [INFO] [1719458454.623445275] [subscribe_pose]: Pose (x,y,theta): (6.03,5.67,0.51)
[subscribe_pose-3] [INFO] [1719458454.636891440] [subscribe_pose]: Pose (x,y,theta): (6.04,5.68,0.52)
[subscribe_pose-3] [INFO] [1719458454.653967783] [subscribe_pose]: Pose (x,y,theta): (6.05,5.68,0.53)
[subscribe_pose-3] [INFO] [1719458454.669382012] [subscribe_pose]: Pose (x,y,theta): (6.05,5.69,0.54)
[subscribe_pose-3] [INFO] [1719458454.685679004] [subscribe_pose]: Pose (x,y,theta): (6.06,5.69,0.54)
[subscribe_pose-3] [INFO] [1719458454.700973153] [subscribe_pose]: Pose (x,y,theta): (6.07,5.70,0.55)
[subscribe_pose-3] [INFO] [1719458454.717747506] [subscribe_pose]: Pose (x,y,theta): (6.08,5.70,0.56)
[subscribe_pose-3] [INFO] [1719458454.730645063] [subscribe_pose]: Pose (x,y,theta): (6.08,5.70,0.57)
[subscribe_pose-3] [INFO] [1719458454.746363823] [subscribe_pose]: Pose (x,y,theta): (6.09,5.71,0.58)
[subscribe_pose-3] [INFO] [1719458454.764616380] [subscribe_pose]: Pose (x,y,theta): (6.10,5.71,0.58)
[subscribe_pose-3] [INFO] [1719458454.778329953] [subscribe_pose]: Pose (x,y,theta): (6.10,5.72,0.59)
[subscribe_pose-3] [INFO] [1719458454.795085473] [subscribe_pose]: Pose (x,y,theta): (6.11,5.72,0.60)
[subscribe_pose-3] [INFO] [1719458454.810771151] [subscribe_pose]: Pose (x,y,theta): (6.11,5.73,0.61)
[subscribe_pose-3] [INFO] [1719458454.826557035] [subscribe_pose]: Pose (x,y,theta): (6.12,5.73,0.62)
[subscribe_pose-3] [INFO] [1719458454.842338085] [subscribe_pose]: Pose (x,y,theta): (6.13,5.74,0.62)
[subscribe_pose-3] [INFO] [1719458454.859096897] [subscribe_pose]: Pose (x,y,theta): (6.13,5.74,0.63)
[subscribe_pose-3] [INFO] [1719458454.874731035] [subscribe_pose]: Pose (x,y,theta): (6.14,5.74,0.64)
[subscribe_pose-3] [INFO] [1719458454.891465347] [subscribe_pose]: Pose (x,y,theta): (6.15,5.75,0.65)
```

2. Further modify the code in square.py so that the robot moves in a trajectory that spells out your initials. See example below. (20pts)



Tips and Tricks:

Commands are sent via a Twist message:
The following Coordinate system is used, where the grey circle is the robot, facing in the direction of the arrow:



- Twist – A Twist object has two main fields, which allow for linear and angular velocities to be set. This is super general so that it could work for any robot. We will only use two of the fields (**bolded** below), but it helps to remember which two if you understand what they all mean.
    - Twist.linear
        - **Twist.linear.x** – this sets the linear velocity in the x direction, which is forward/backwards (for +/- values). This gives the forward speed of the robot
        - Twist.linear.y – this sets the linear velocity in the y direction, which is sideways. Our robot can't move like this, so we will not use this field.
        - Twist.linear.z – this sets the linear velocity in the z direction, which is up. Our robot can't fly, so we will not use this field either.
    - Twist.angular
        - Twist.angular.x – This sets the angular velocity/ rotation around the x axis, or the roll of the robot. Since we are on the ground, we cannot rotate around this axis, so we won't use this.

        - Twist.angular.y – This sets the angular velocity /rotation around the y axis, or the pitch of the robot (moving front up/down). <u>Since we are on the ground, again, we won't use this.</u>
        - **<u>Twist.angular.z</u>** – This sets the angular velocity/rotation around the z axis, which rotates the robot left/right. We will use this command to turn the robot. Units are in radians/sec - positive is a left turn (increasing the angle/heading of robot), while negative is right turn.

- To drive the robot forward/backward, set twist.linear.x to +/- values. (0.5 seems to work ok)
- To spin the robot, set twist.angular.z to +/- values (again, 0.5 seems a good starting point)
- If you have trouble getting ROS to roscd or errors launching (launch file not known or does not exist) to your directory, try running the following command from the catkin_ws directory. This will make sure that ROS knows about our packages
    - catkin_make
    - source devel/setup.bash
    - Add the line "source <path to your catkin>/catkin_ws/devel/setup.bash" to your .bashrc file to have this happen automatically when you log into a terminal. (make sure that <path to your catkin> is replaced by the path on your system to the catkin_ws folder. You can get this by typing pwd from the terminal when you are in the directory.
    - You can add this to your bashrc file using the following commands (each all on one line) from the terminal -
        - echo "source <path to catkin>/catkin_ws/devel/setup.bash" >> ~/.bashrc
        - source ~/.bashrc

Frequently-Asked Questions
Q: How can I run ROS code if VirtualBox does not work on my computer?
A: You may use other virtual machine software such as https://mac.getutm.app/ or https://www.vmware.com/products/fusion.html

Q: How to fix the error "`ros2: command not found`"?
A: Refer to the instructions here to enable ROS 2 commands on the terminal https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Configuring-ROS2-Environment.html. The command `source /opt/ros/humble/setup.bash` needs to be either executed on every new shell or sourced on the shell startup script.

Q: How to fix the error "`Package 'ai_labs' not found`"?
A: Make sure that the lab files are downloaded into the correct directory structure (refer to Part II, Step 1). Run the command `source ~/ros2_ws/install/setup.bash` to update the ROS search path for packages in your ROS workspace. If ROS cannot detect the ai_labs package, it may be that it is missing the package.xml file. Make sure that you use `ros2 pkg create` command to create the ai_labs package instead of creating it manually.

Q: How to fix the error "`No executable found`":
A: Make sure that you have added the Python script as an executable entry point (https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Py-Publisher-And-Subscriber.html#id2) and have used the commands `colcon build` and `source ~/ros2_ws/install/setup.bash` to build and setup the workspace.

Q: Why is my robot not moving in a square even after I have updated the code in square.py?

A: Make sure that you save the file square.py after any code changes are made. In addition, the command `colcon build` has to be executed in the top-level of the workspace every time any updates are made to the launch files or Python scripts in order for the changes to be registered in the ROS environment.

Q: Is it okay if the robot is not moving in a perfect square?
A: Yes. Due to numerical rounding or message timing issues, the robot may not move in a perfect square. Any code that can move the robot in the general shape of a square will be considered valid.

Deliverables
To complete the lab you must submit via canvas a zip file containing your source code and a screen recording of your program execution. The screen recording should involve (i) the robot moving in a square, (ii) output of the *rostopic list* command and (iii) output of the *rostopic echo* command.