



**Michigan
Technological
University**

Michigan Technological University
Digital Commons @ Michigan Tech

Dissertations, Master's Theses and Master's Reports

2022

DEVELOPMENT AND IMPLEMENTATION OF ROBOT OPERATING SYSTEMS FOR UNDERGRADUATES

Chelsey Spitzner

Michigan Technological University, cbspitzn@mtu.edu

Copyright 2022 Chelsey Spitzner

Recommended Citation

Spitzner, Chelsey, "DEVELOPMENT AND IMPLEMENTATION OF ROBOT OPERATING SYSTEMS FOR UNDERGRADUATES", Open Access Master's Report, Michigan Technological University, 2022.

<https://doi.org/10.37099/mtu.dc.etr/1386>

Follow this and additional works at: <https://digitalcommons.mtu.edu/etr>



Part of the [Robotics Commons](#)

DEVELOPMENT AND IMPLEMENTATION OF ROBOT OPERATING SYSTEMS
FOR UNDERGRADUATES

By

Chelsey B. Spitzner

A REPORT

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Electrical and Computer Engineering

MICHIGAN TECHNOLOGICAL UNIVERSITY

2022

© 2022 Chelsey B. Spitzner

This report has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Electrical and Computer Engineering.

Department of Electrical and Computer Engineering

Report Advisor: *Dr. Christopher Middlebrook*
Committee Member: *Dr. Glen E. Archer*
Committee Member: *Dr. Shane Oberloier*
Committee Member: *Dr. Michael C. Roggemann*
Department Chair: *Dr. Glen E. Archer*

Table of Contents

List of Figures	v
List of Tables	vii
Abstract	viii
1 Introduction and Motivation	1
1.1 Laboratory Setup	3
1.2 Pre-lab and Post-lab.....	5
2 ROS Preliminaries	6
2.1 Lab 1	6
2.2 Lab 2.....	8
3 Mobile Motion	10
3.1 Lab 3.....	10
3.2 Lab 4.....	13
4 Fixed Motion.....	19
4.1 Lab 5.....	19
4.2 Lab 6.....	24
5 Sensing.....	26
5.1 Lab 7.....	26
6 Navigation.....	28
6.1 Lab 8.....	28
6.2 Lab 9.....	31
7 Vision Systems.....	35
7.1 Lab 10.....	35
8 Future Work and Recommendations	37
9 Reference List	41
A Lab Manuals.....	43
A.1 Lab 1 ROS Fundamentals.....	43
A.2 Lab 2 Topics Services and Actions	49
A.3 Lab 3: Simulation	58
A.4 Lab 4: Mobile Motion	65
A.5 Lab 5: Fixed Motion.....	76
A.6 Lab 6: Pose Estimation.....	83
A.7 Lab 7: Sensing.....	86

A.8	Lab 8: Mapping	94
A.9	Lab 9: SLAM.....	99
A.10	Lab 10: Machine Vision	104
A.11	MATLAB Introduction	109

List of Figures

Figure 1.1 Dependency chart for the labs	2
Figure 1.2: Turtlebot3 Burger [4].	4
Figure 1.3: Niryo Ned robotic arm [5].	5
Figure 2.1: Simulation for demonstrating coordinate transformations.	7
Figure 2.2: ROS tf tree of simulation.	8
Figure 2.3: ROS node graph for the action server and client.	9
Figure 3.1: Turtlebot3 in Gazebo simulation.	11
Figure 3.2: ROS RViz visualization of the Turtlebot3 and lidar sensor.	12
Figure 3.3 ROS node graph for Turtlebot3 teleoperation.	12
Figure 3.4: ROS graph of linear and angular velocity	13
Figure 3.5: ROS node graph for automatic obstacle avoidance package.	13
Figure 3.6: tf tree of Turtlebot3.	15
Figure 3.7: RViz with Turtlebot3 and lidar scan data.	16
Figure 3.8: Automatic parking algorithm steps.	17
Figure 3.9: Turtlebot position after running the automatic parking node.	18
Figure 3.10: ROS node graph for the automatic parking.	18
Figure 4.1: Niryo Ned arm in ROS RViz simulation.	20
Figure 4.2: Application students use for mouse control of the Niryo Ned robot.	21
Figure 4.3: ROS node graph of the Niryo Ned robotic arm.	22
Figure 4.4: ROS tf tree of the Niryo Ned robotic arm.	23
Figure 4.5: Path Planning in RViz.	24
Figure 4.6: Demonstration of Niryo Ned robot planning a motion.	25
Figure 5.1: Diagram of IR sensor setup to detect negative obstacles.	27
Figure 6.1: Turtlebot3 Gazebo simulation of a house for mapping.	29
Figure 6.2: Map created after parameter optimization in the Gazebo simulation.	30
Figure 6.3 ROS node graph for Turtlebot3 mapping using rosbag to playback the data.	31
Figure 6.4 Map creating using Gmapping SLAM.	32
Figure 6.5 Map creating using Hector SLAM.	33
Figure 6.6 Map created using Karto SLAM.	33
Figure 6.7: ROS node graph for Turtlebot3 running hector SLAM node.	34
Figure 7.1 Checkerboard for intrinsic camera calibration.	36

Figure 8.1 ROS node graph for MATLAB to ROS connection for Turtlebot3	37
Figure 8.2: Laser scan data plotted using MATLAB using data from the Turtlebot3.....	38
Figure 8.3 Turtlebot3 trajectory with no controller	39
Figure 8.4 Turtlebot3 trajectory with Lyapunov controller	39

List of Tables

Table 1.1 Overview of the labs and their titles	1
Table 1.2 Lab groups.	3
Table 1.3 Materials needed for Laboratory.....	4
Table 2.1 Packages used in Lab 1	6
Table 2.2 Packages used in Lab 2.....	9
Table 3.1 Packages used in Lab 3	10
Table 3.2 Packages used in Lab 4.....	14
Table 4.1 Packages used in Lab 5.....	19
Table 4.2 Packages used in Lab 6.....	25
Table 5.1 Packages used in Lab 7.....	26
Table 6.1 Packages used in Lab 8.....	28
Table 6.2 Packages used in Lab 9.....	31
Table 6.3: SLAM methods and associated algorithms for lab 9.....	32
Table 7.1 Packages used in Lab 10.....	35

Abstract

The purpose of this project was to create an undergraduate junior lab to teach students about Robotic Operating System (ROS). The labs were designed to highlight the usefulness of ROS and the process used. Designing algorithms, how to send/receive messages, and the hierarchy of how nodes work with each other are emphasized. Taking packages that are open-source then modifying them is also emphasized. This is done so that students can transfer their knowledge from this course to other robot operating systems.

1 Introduction and Motivation

There were ten labs created for the laboratory course listed in Table 1.1. The topics were chosen for the labs to be intertwined with the course instruction. This allows students to first learn about the material in a classroom setting and then apply their knowledge in a laboratory setting.

Table 1.1 Overview of the labs and their titles

Lab Number	Lab Title
Lab 1	ROS Fundamentals
Lab 2	Topics, Services, and Actions
Lab 3	Simulation
Lab 4	Mobile Motion
Lab 5	Fixed Motion
Lab 6	Pose Estimation
Lab 7	Sensing
Lab 8	Mapping
Lab 9	SLAM
Lab 10	Machine Vision

The course associated with these laboratories is new to Michigan Technological University and could be subject to change in the future. This change is likely to come in the form of rearranging the course material to follow a different order of learning the material. If this were to happen, the labs would need to be rearranged as well to follow the course. Figure 1.1 shows a dependency chart for the labs. This will allow the instructor of the course to easily change lab order to match the course.

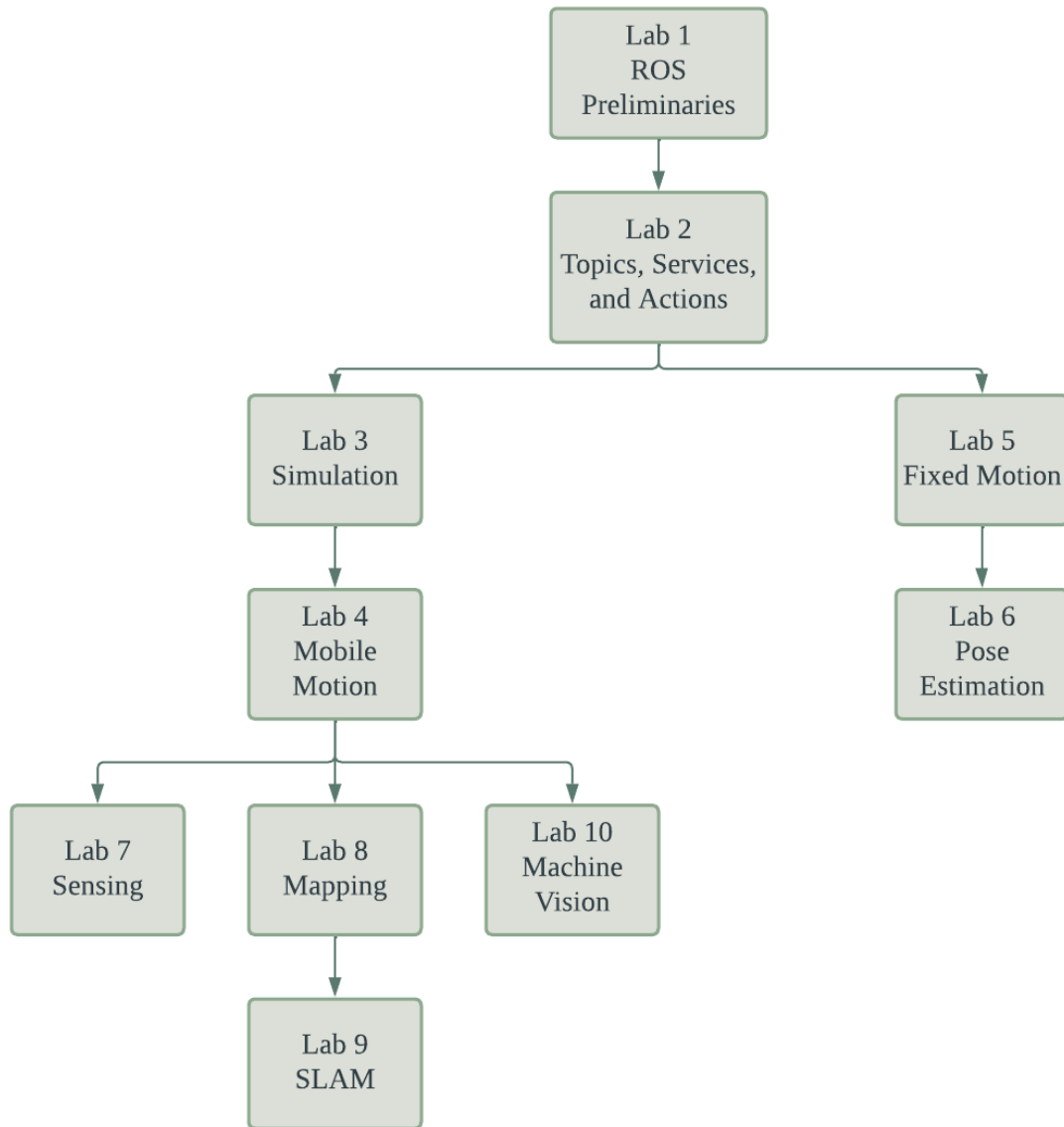


Figure 1.1 Dependency chart for the labs. It shows which labs are a prerequisite for other labs.

The labs are grouped into subject matter for the purpose of discussion in the following chapters. These groups are shown in Table 1.2. Each Group is a chapter with a discussion of the labs following this introduction.

Table 1.2 Lab groups.

Lab Group	Lab Number	Robot
ROS Preliminaries	Lab 1	Turtlebot3
	Lab 2	Turtlebot3
Mobile Motion	Lab 3	Turtlebot3
	Lab 4	Turtlebot3
Fixed Motion	Lab 5	Niryo
	Lab 6	Niryo
Sensing	Lab 7	Turtlebot3
Navigation	Lab 8	Turtlebot3
	Lab 9	Turtlebot3
Vision Systems	Lab 10	Turtlebot3

1.1 Laboratory Setup

The laboratory is going to be run with students in groups of two. This is done for a few reasons. First, it fits the space constraints of the lab. Most of the labs require physical robots moving around through mazes, maps, etc. If each student were to use their own robot, there may not be enough space in the laboratory as well as time. Since the labs are going to be two-hour labs, if there is a line waiting to use a maze or other physical setup, it could be limiting the students in what they can achieve in the two hours. The educational aspect of running laboratories in groups of two was also considered. Conducting laboratories with students in groups improves students' performances not only in the lab, but also in the overall course [1 – 3].

The laboratory space will be setup with desks along the walls of the lab. Each group of students will have their own desk with a PC and robots at each desk. Having PCs in the lab instead of students using their own laptops will minimize the time debugging ROS software which will allow students to complete the lab in a two-hour time frame. The robots that the students will be using are the Turtlebot3 Burger Figure 1.2 and the Niryo Ned Figure 1.3 robots. Both robots are built for educational purposes and interface well with ROS. Table 1.3 gives a list of the minimum number of materials needed for the laboratory.

Table 1.3 Materials needed for Laboratory

Material	Quantity
Desks	8
PC	8
Turtlebot3 Burger	8
Niryo Ned	4
Maze walls and connectors	1



Figure 1.2: Turtlebot3 Burger [4].



Figure 1.3: Niryo Ned robotic arm [5].

1.2 Pre-lab and Post-lab

Pre-labs are designed to be assigned with this laboratory. This ensures that students come to the laboratory prepared for the assignments. Most pre-labs are done on students' personal computers using a virtual machine running Linux.

Post-labs vary in format depending on the lab. The format for most of them is either submitting python code or submitting an image with comments. The images the students submit are either node/coordinate transformation graphs from ROS, or plots of the velocity or position of the robots. Post-labs ensure that the student spends time considering what they did in the lab and looking at the overall connections between ROS and the robot.

2 ROS Preliminaries

ROS preliminaries lab group consists of the first two labs. They are a prerequisite to the rest of the labs as shown in Figure 1.1. They are designed to introduce the student to ROS assuming no prior experience with the software. As these are the first labs, it is mostly step-by-step instruction that the students must follow. There are a few instances where the students must change some code which is done to force the student to look at the code. This is done so that the student doesn't copy and paste code without understanding what it does. In each lab, students are required to view either the node graph or the coordinate transform graph for the ROS nodes they are running. This ensures that students are reminded of the overall scheme of ROS without getting to enveloped in the details.

2.1 Lab 1 (A.1)

Lab 1 is titled ROS Fundamentals. It introduces main concepts that are fundamental to both ROS and robotics. This lab is mostly step-by-step instruction through the lab manual due to it being the first lab. It starts with the student setting up a workspace and creating a ROS package. This is a fundamental skill that students will use in each lab. Table 2.1 shows which packages will be used for Lab 1.

Table 2.1 Packages used in Lab 1

Packages Used
tf2
RViz
tf_tree
rospy

The students create a package with two nodes. Each node is a python script which will create a simulation with two objects where one object follows the other (Figure 2.1). This allows the laboratory to teach the concept of coordinate transformations. It also teaches students how to create packages and launch them.

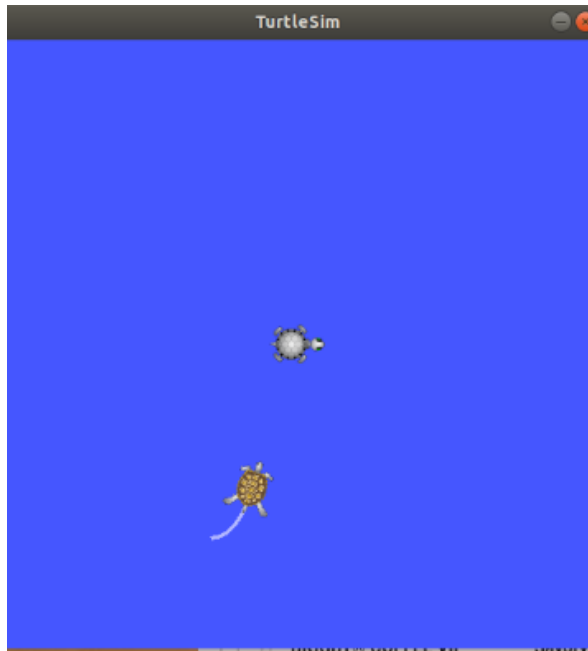


Figure 2.1: Simulation for demonstrating coordinate transformations.

Once students create this simulation, they learn how to add a static coordinate transform to one of the objects. This system can be visualized using ROS's tf tree Figure 2.2. The rest of the lab is experimenting and visualizing the coordinate transformations of the system using RViz as well as using rostopic tools to view the topics being published.

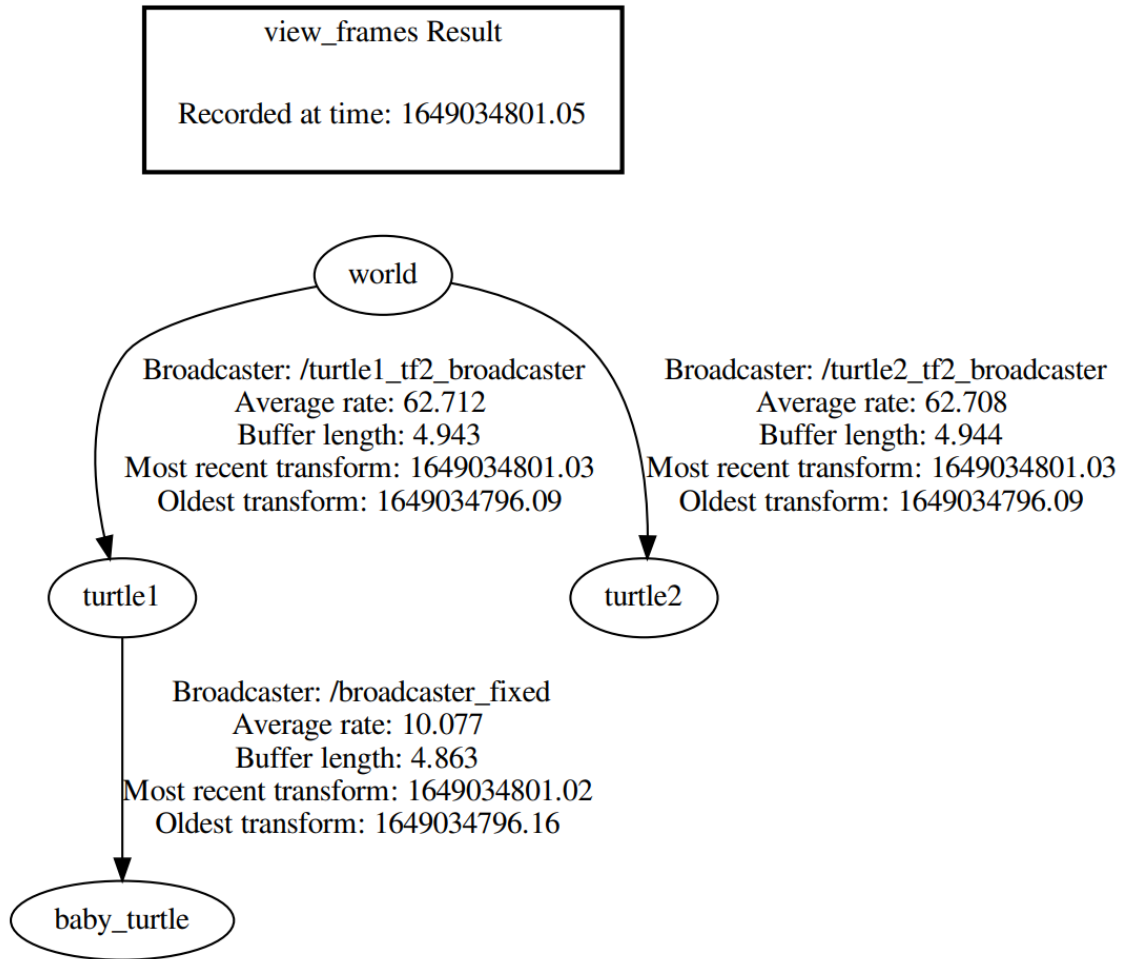


Figure 2.2: ROS tf tree of simulation. It shows how the coordinate transforms of the objects present in the ROS simulation.

2.2 Lab 2 (A.2)

Lab 2 is similar to Lab 1 because it is mostly step-by-step instruction with a few instances where the student will need to change some of the python code. It is split into three subjects: Topics, Services, and Actions. These three subjects are fundamental to how ROS operates. Although they are ROS specific, the overall concept of these subjects is present in most robot operating systems [6]. Table 2.2 shows which packages will be used for Lab 1.

Table 2.2 Packages used in Lab 2

Packages Used
Publishers
Subscribers
Services
Actions
rqt_graph

Topics consist of publishers and subscribers. This is a fundamental concept to ROS but is present in most other robotic operating systems. It allows information to be published and for nodes to subscribe to the information that it needs to operate. The students learn how to create python scripts to publish some information and then create another script to take that information and add something to it.

Services can take information and perform an operation to them. This is useful for unit conversion or math operations. The students create a service to take two user-input integers and output them to the command terminal. Topics and services are relatively straightforward and do not take much time to create.

Actions are more ROS-specific but are useful for many different applications. They are also more complicated and take most of the lab to create. In this lab, the student creates a “fake” sensor which feeds data to the action client. The action server takes the mean and standard deviation of the sensor data. This is done to show the student how actions can be used to perform some analysis on sensors. This data can then be used in an algorithm to perform a specific task.

Throughout these three tasks, students are asked to view the ROS node graph shown in Figure 2.3. This ensures that the students understand the overall connections for each subject. Rostopic echo command from ROS also allows students to see the output from the published topics on their command terminal.

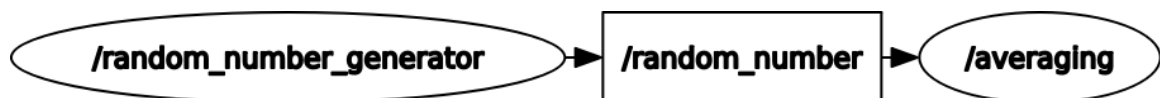


Figure 2.3: ROS node graph for the action server and client

3 Mobile Motion

This group of Labs 3 and 4 introduces mobile motion. This is done through the Turtlebot3 Burger platform. Turtlebot3 is produced by Robotis and is highly regarded as one of the leading platforms for learning robotics and for testing algorithms. It is designed to interface with ROS which allows students to focus on the process and the development of algorithms without getting caught up in the technical details. Since students are now expected to know how to use some ROS fundamentals from the previous labs, this group of labs focuses on students developing algorithms and creating their own code.

3.1 Lab 3 (A.3)

Lab 3 is a simulation lab and uses the packages in Table 3.1.

Table 3.1 Packages used in Lab 3

Packages Used
Teleoperation
Keystroke to Velocity Command
Gazebo
rqt_plot, rqt_graph
RViz
Obstacle Avoidance (created by students)

The laboratory starts with simulating the Turtlebot in Gazebo as shown in Figure 3.1. Gazebo is used to simulate the robot in accurate real-life scenarios. It does this by employing physics engines that implement the non-linearities that are present in the system as well as adding noise to sensor data. This allows students to first test their algorithms in simulation before implementing them on a physical Turtlebot. ROBOTIS offers a set of environments available to use in Gazebo. The environment showed in Figure 3.1 shows the Turtlebot3 World offered by ROBOTIS.

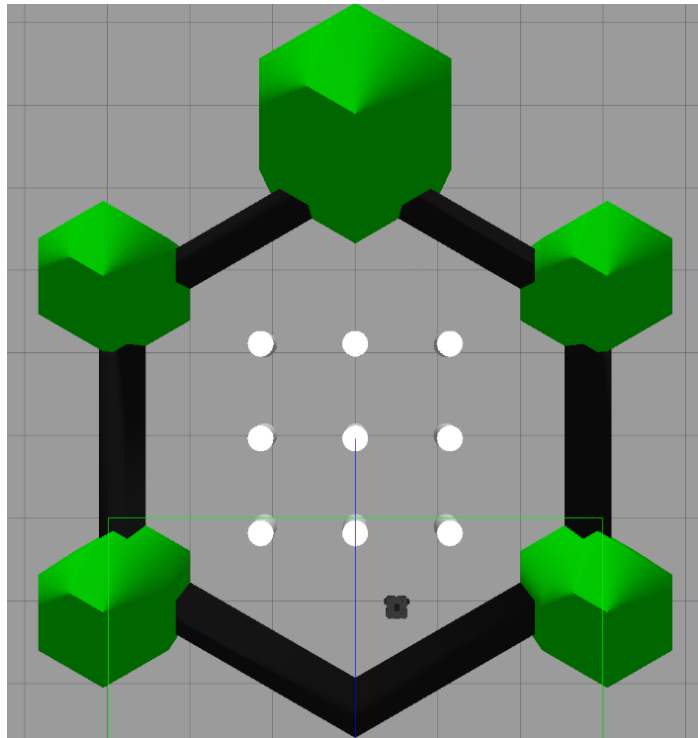


Figure 3.1: Turtlebot3 in Gazebo simulation.

Another tool the students use throughout the laboratory is RViz. RViz is a visualization tool from ROS that can show the robot model and sensor data among other things. The students use RViz in lab 3 to visualize the sensor data from the lidar sensor on the Turtlebot3 Figure 3.2: ROS RViz visualization of the Turtlebot3 and

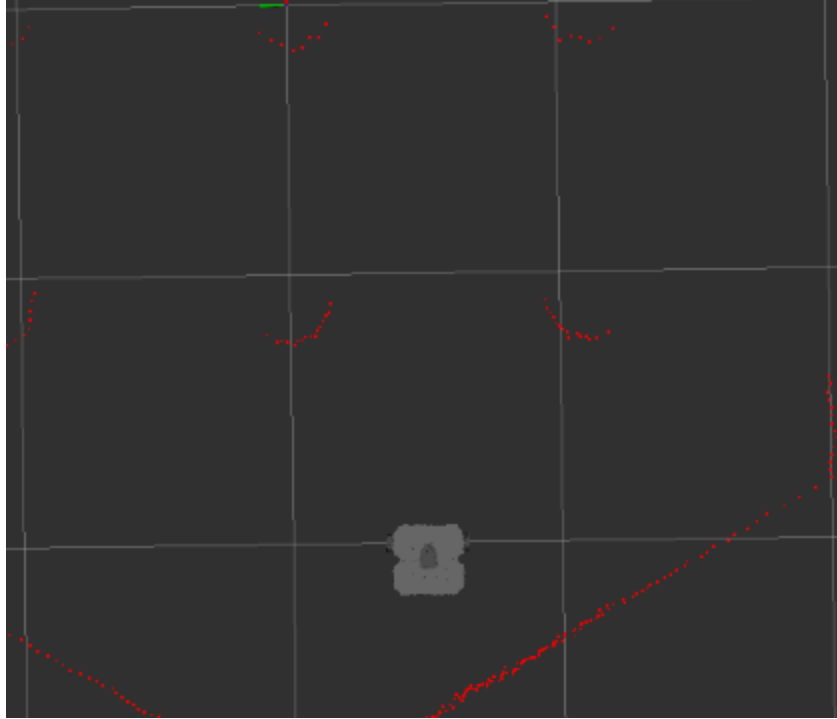


Figure 3.2: ROS RViz visualization of the Turtlebot3 and lidar sensor.

Students are tasked to create a teleoperation node for the Turtlebot. They create a script to map certain keys to velocity commands. Then they need to create a script to transform the velocity commands to more realistic commands. This involves ramping the velocity instead of making the velocity instantaneously jump. Once they execute this, they test it in the Gazebo simulation and view the ROS node graph for their package Figure 3.3.

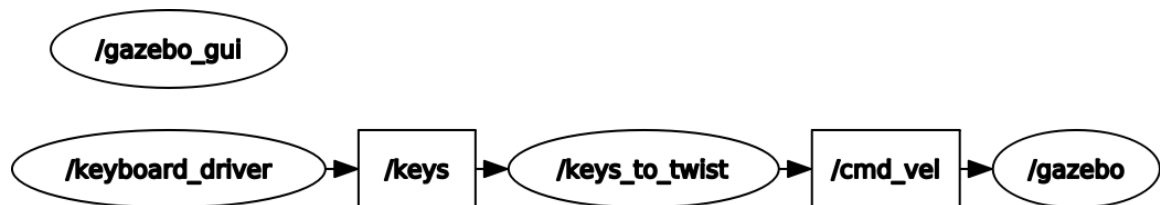


Figure 3.3 ROS node graph for Turtlebot3 teleoperation.

For the last task, they must create a simple obstacle avoidance algorithm. This requires the students to work again with creating a new package and subscribing to the Turtlebot lidar sensor data. Once they subscribe to the data, they need to write a simple algorithm to publish velocity commands if the Turtlebot is within a certain distance to an object.

Once the students are done developing their algorithm, they test it using the Gazebo simulation. While the obstacle avoidance node is running, the students plot the velocity commands that their node is sending the Turtlebot Figure 3.4.

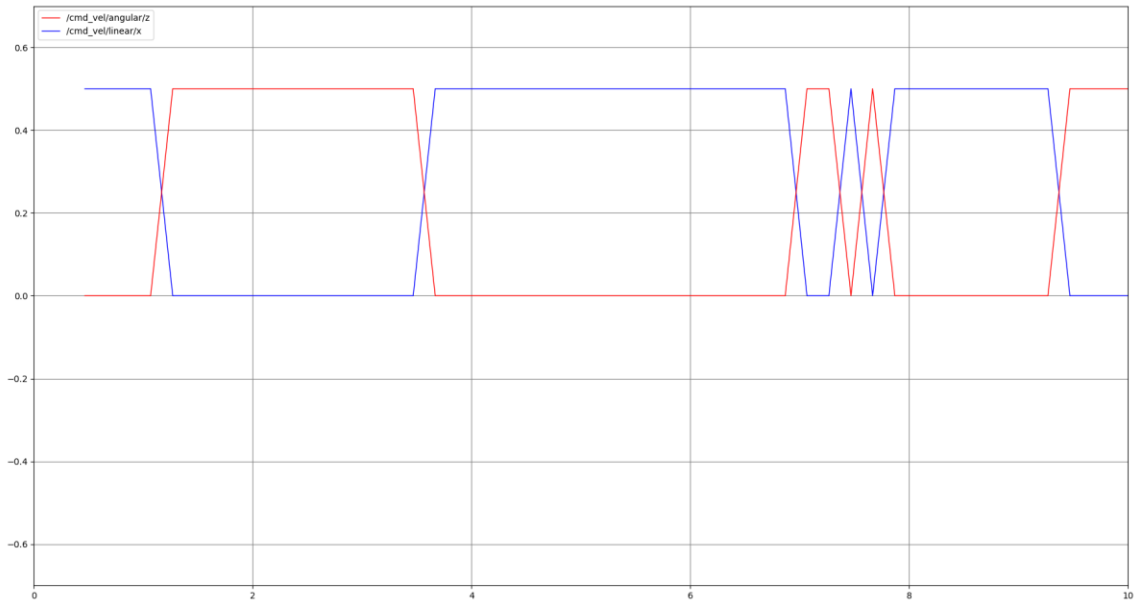


Figure 3.4: ROS graph of linear and angular velocity. These are the command velocities sent to the Turtlebot3 during the automatic obstacle avoidance algorithm.

The students also show the ROS node graph in Figure 3.5. This encourages them to see how using the obstacle avoidance algorithm creates a closed loop system with no input needed from the user as compared to Figure 3.3.

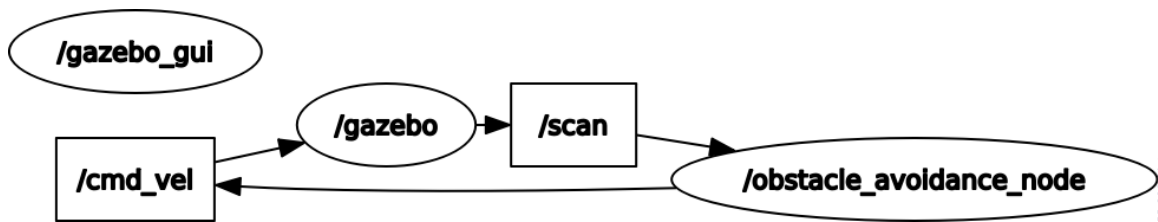


Figure 3.5: ROS node graph for automatic obstacle avoidance package.

3.2 Lab 4 (A.4)

Lab 4 is the first lab where the students are using ROS on a physical robot. This is done by using a remote PC to connect over a network to the Raspberry Pi on the Turtlebot3 Burger. The packages used in this lab are shown in Table 3.2.

Table 3.2 Packages used in Lab 4

Packages Used
Teleoperation
Gazebo
tf_tree
rqt_plot, rqt_graph
RViz
Automatic Parking (created by students)

The initial parts of this lab utilize the same teleoperation packages the students used in Lab 3. This is done to show students how the same packages in ROS can be used in both simulation and on the physical robot. The students view the tf tree of the Turtlebot 3 Figure 3.6.

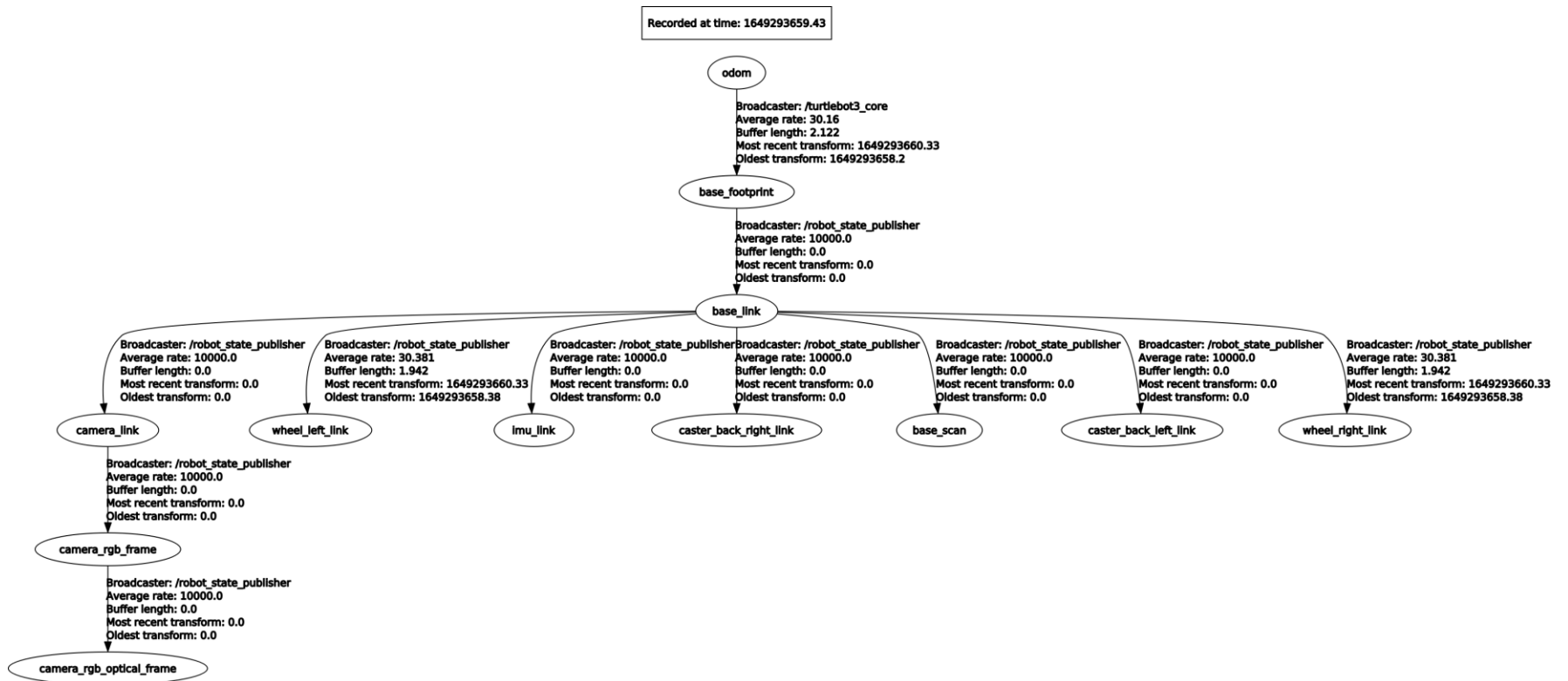


Figure 3.6: tf tree of Turtlebot3.

The final part of the lab is having the students create an automatic parking script. The automatic parking is done using the laser-distance scanner on the Turtlebot3. Reflective tape is then put on one of the walls in the laboratory. Then, the laser-distance scanner can detect the reflective tape and the students can design an algorithm to have the robot move and stop by the tape. The setup for this will be four walls to create a square/rectangle to place the Turtlebot3 into. Then, the reflective tape will be placed by the students in a random location on one of the walls. This ensures that the tape is in a different location for each student. Since lidar sensors are affected by the reflectivity of the materials around it, the reflective tape has greater intensity than anything else around the Turtlebot as shown in Figure 3.7

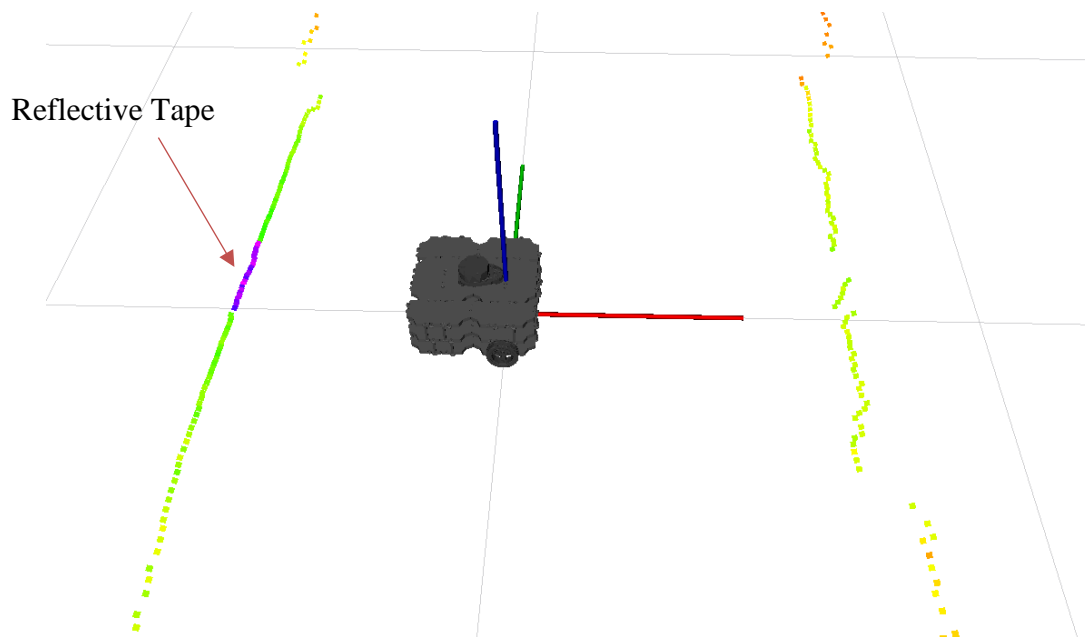


Figure 3.7: RViz with Turtlebot3 and lidar scan data. The red arrow is added to emphasize where the reflective tape is placed.

The automatic parking algorithm is split into four steps. Students are given Figure 3.8 in the laboratory manual to visualize the steps in the algorithm. An incomplete python script is given to them with four sections they need to complete with the four steps.

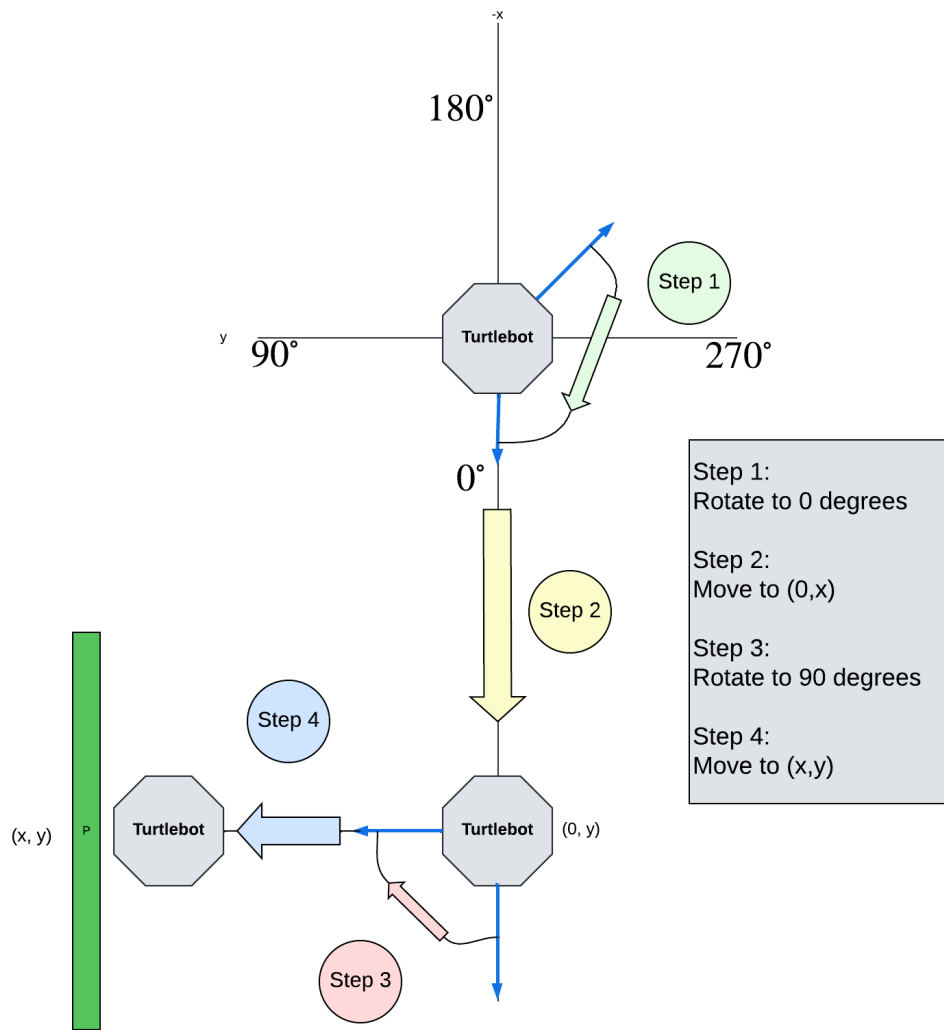


Figure 3.8: Automatic parking algorithm steps.

Once students complete the python script, they create the necessary launch files to complete the automatic parking algorithm node. They can then test their algorithm in the lab where an empty square will be setup with a strip of reflective tape. Figure 3.9 shows the RViz image of where the Turtlebot ends after the automatic parking node is run.

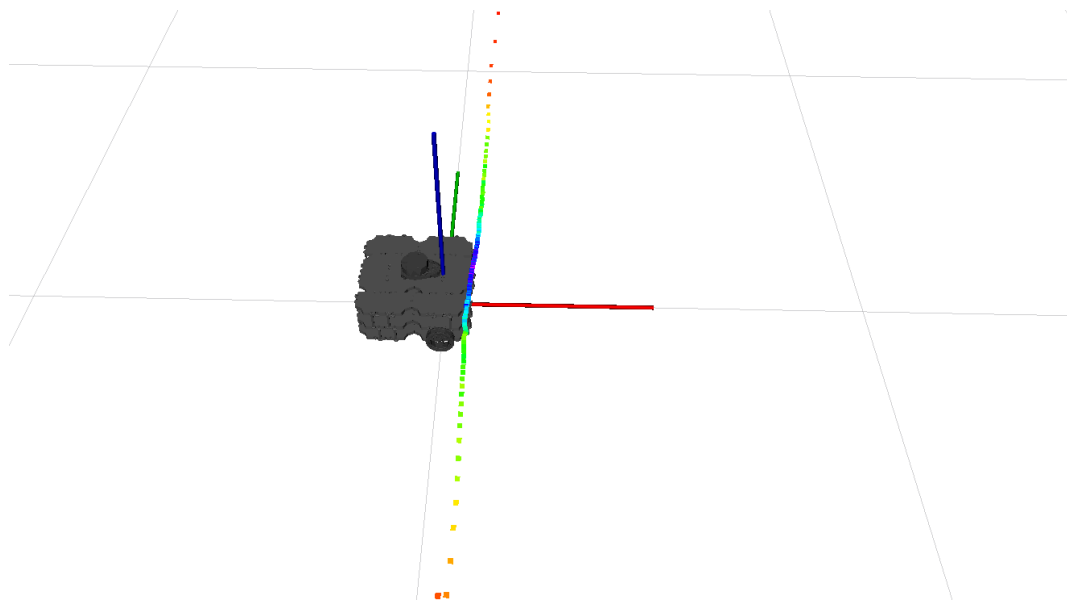


Figure 3.9: Turtlebot position after running the automatic parking node.

The node graph of the Turtlebot3 running with the automatic parking algorithm is shown in Figure 3.10.

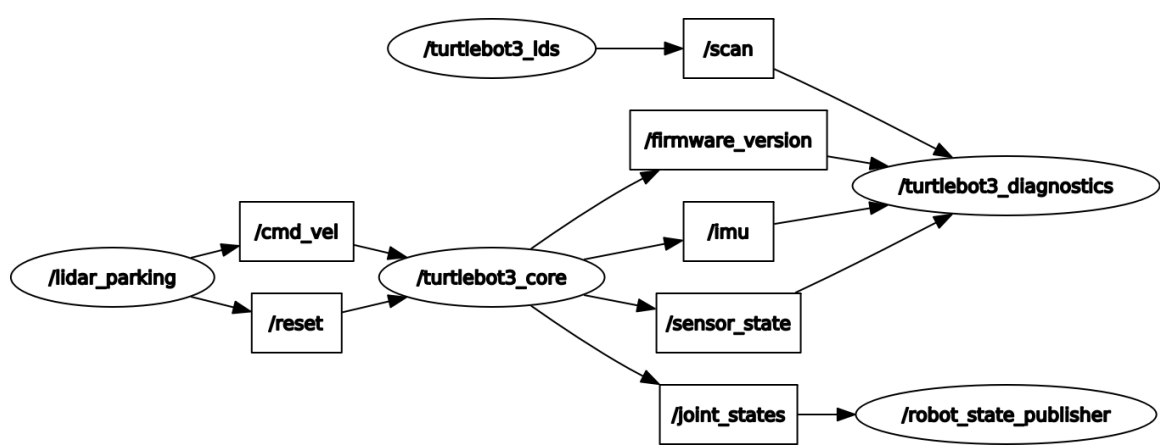


Figure 3.10: ROS node graph for the automatic parking.

4 Fixed Motion

This group of labs uses the Niryo Ned robotic arm. It is a 6-axis robot designed to be used in education and research. Niryo Ned interfaces well with ROS and offers many options to customize the robot and its applications.

4.1 Lab 5 (A.5)

Lab 5 introduces the student to the Niryo Ned robotic arm. The software packages used in this lab are shown in Table 4.1.

Table 4.1 Software packages used in Lab 5

Packages Used
RViz Simulation
Python3
MoveIt
RViz for Motion Planning

There is a graphical user interface (GUI) offered by Niryo that allows students to change the angles of each of the joints Figure 4.1. Students can observe how the joint angles move the robot in the simulation.

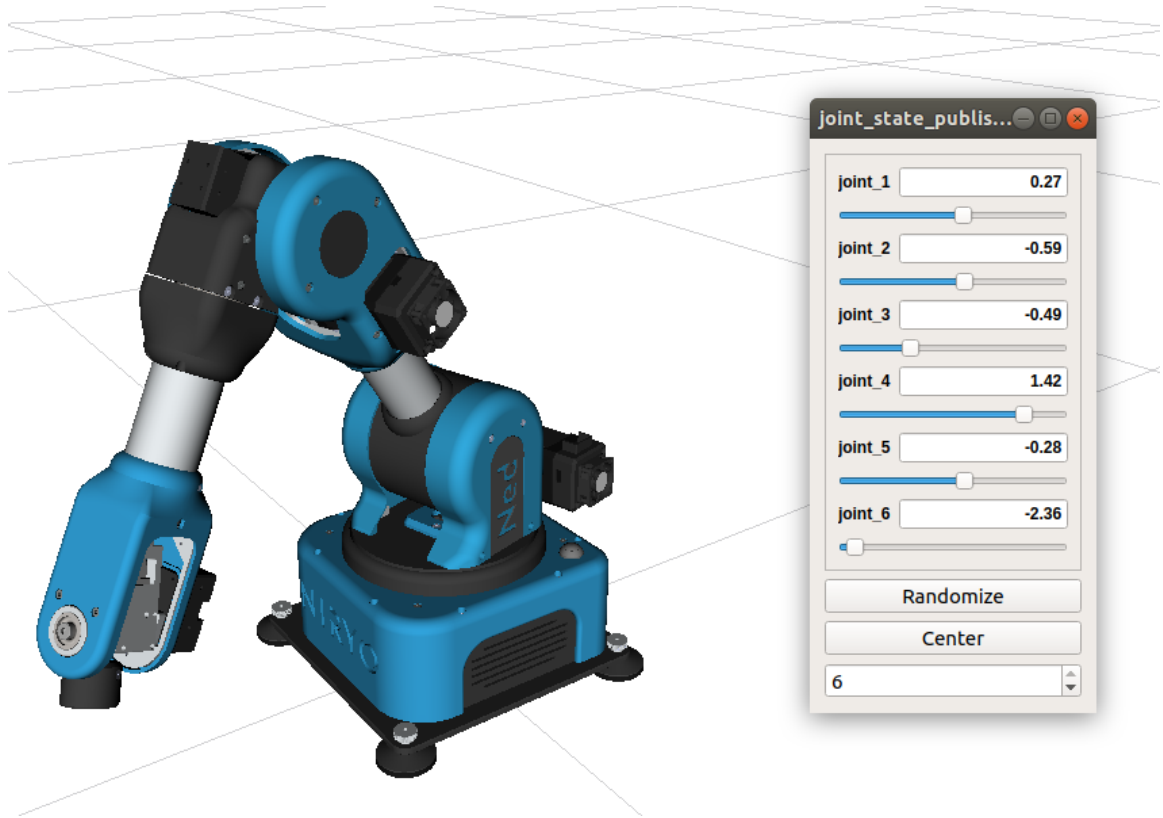


Figure 4.1: Niryo Ned arm in ROS RViz simulation.

Using this simulation, the students become familiar with where each joint is and how moving each joint angle affects the pose of the robot. Once the student is familiar with the robot and how it moves, an application is used so that students can control the simulated robot with a computer mouse Figure 4.2. This application shows the envelope for the robot in white, the current end-effector position in blue, and the desired end-effector position in green. The desired end-effector position is set by the PC mouse and is under the blue dot in Figure 4.2. The application also displays the joint angles and pose of the end-effector.

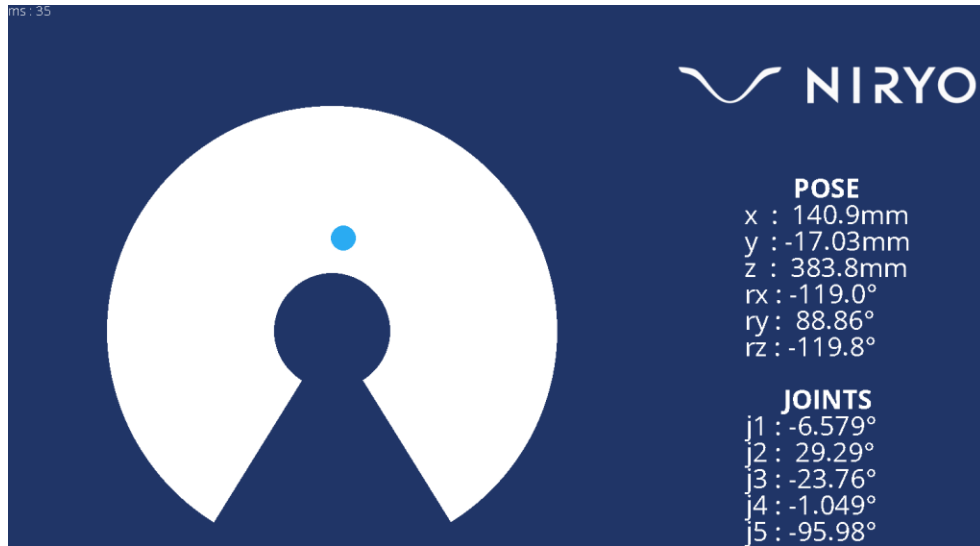


Figure 4.2: Application students use for mouse control of the Niryo Ned robot

Once the students have completed the application, they are instructed to view the node graph and the tf tree of the system. The node graph in Figure 4.3 is much larger than the Turtlebots node graph in Figure 3.5. This allows students to observe why more path planning packages are used with the Niryo Ned for most movement.

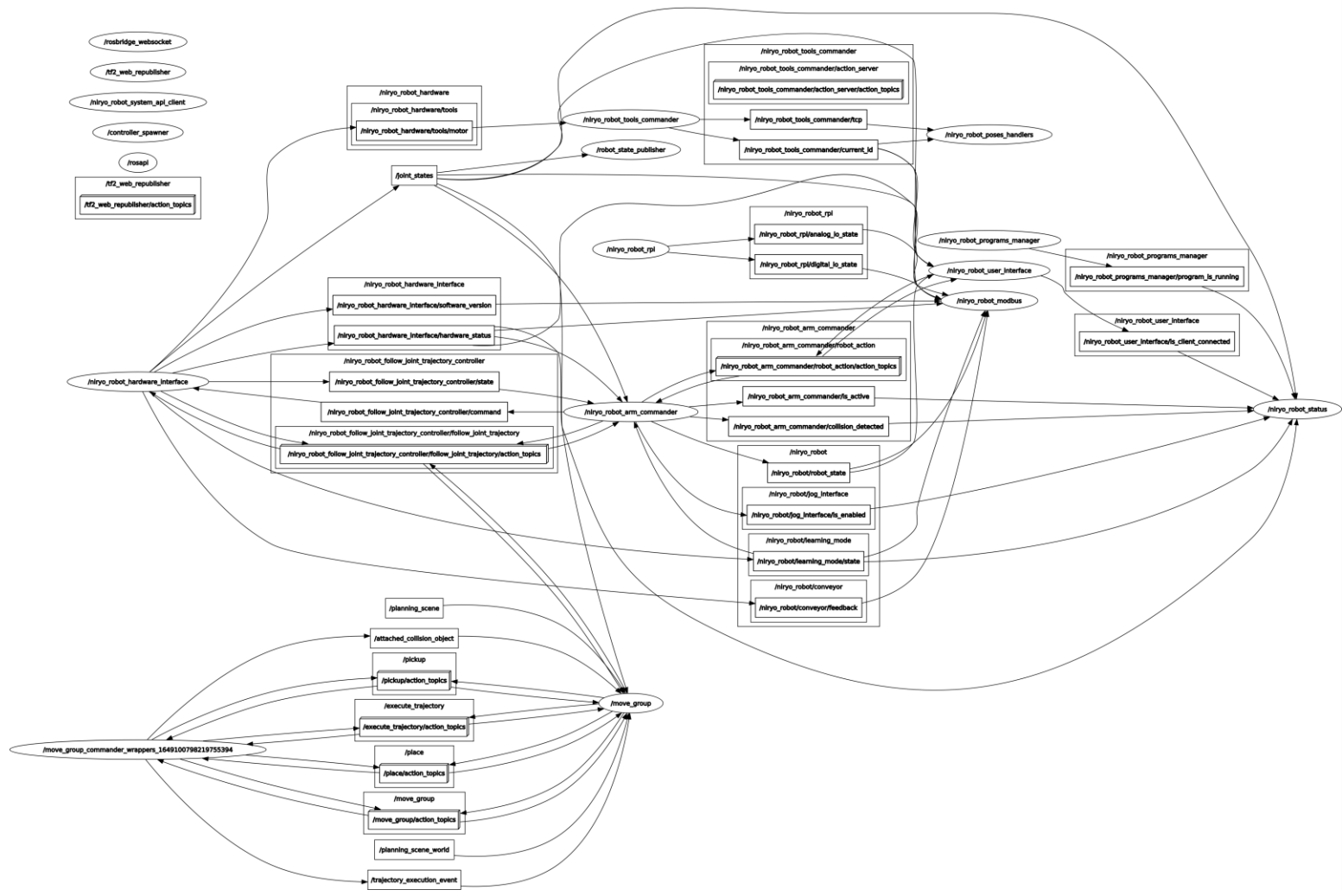


Figure 4.3: ROS node graph of the Niryo Ned robotic arm.

The *tf tree* of the robotic arm is also viewed by the students Figure 4.4.

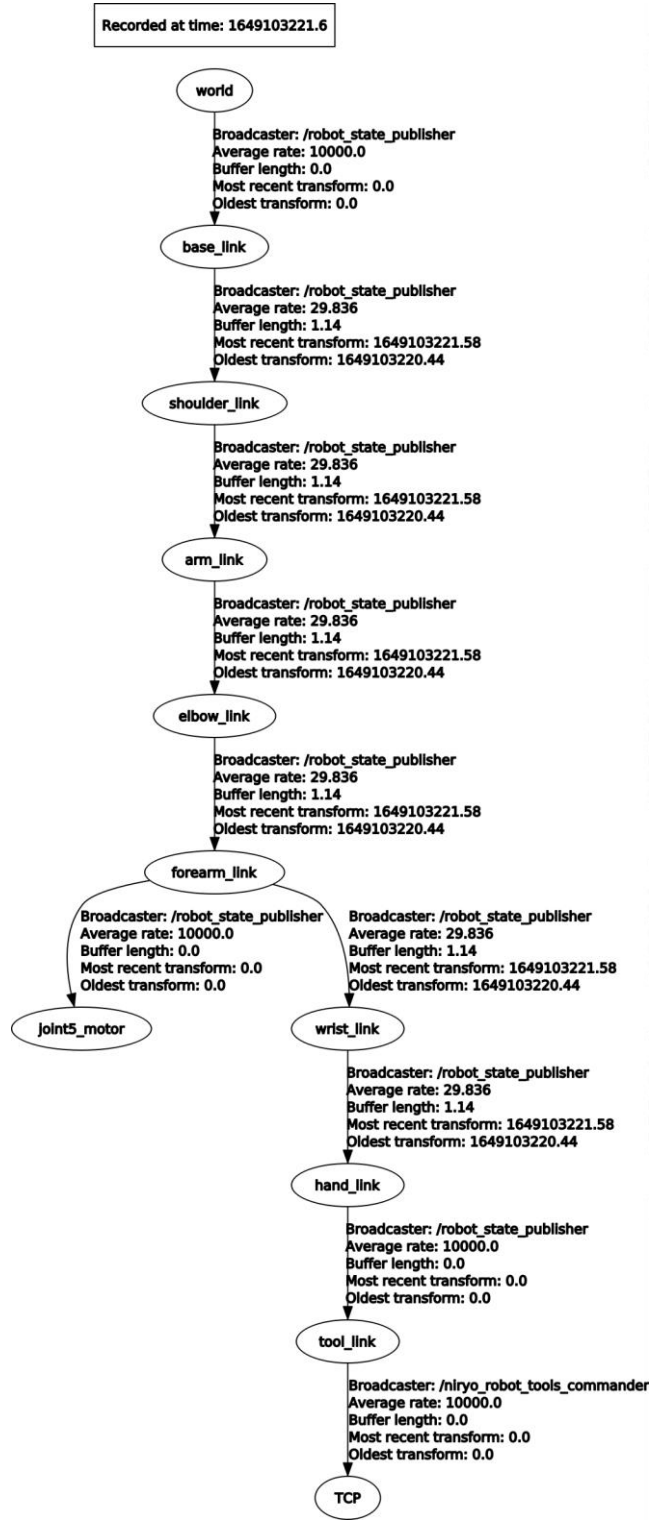


Figure 4.4: ROS *tf tree* of the Niryo Ned robotic arm.

For the last part of the lab, students use the path planning feature provided by RViz to graphically plan a motion. In Figure 4.5, the desired pose is set by clicking and dragging the blue/red arrows. The end desired pose is shown in orange and the current is shown in the solid blue. The translucent Niryo arm is the visualization of the planned path from the current to the end pose.

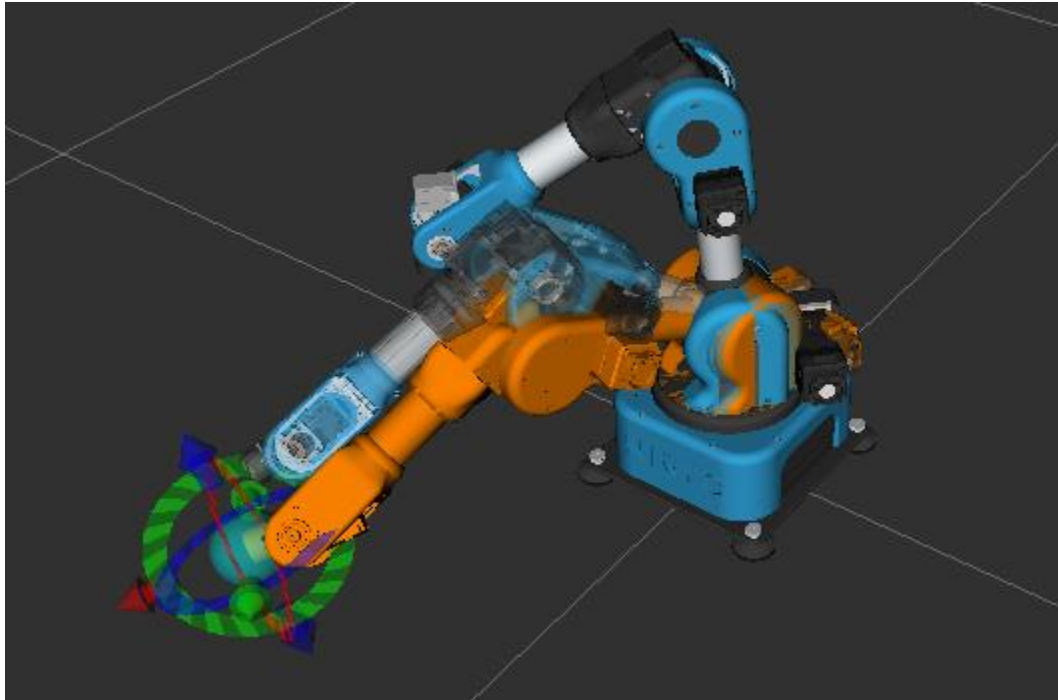


Figure 4.5: Path Planning in RViz.

4.2 Lab 6 (A.6)

Lab 6 is focused on controlling the pose of the robot arm. This is done using *MoveIt* [7]. *MoveIt* offers packages for forward/inverse kinematics, manipulation, and motion planning. This allows students to implement pose estimation in the timeframe required for the lab. First the students learn about the configuration needed to use *MoveIt*. It requires a URDF file for the robot, joint names, and joint groups. The software packages used in this lab are shown in Table 4.2.

Table 4.2 Software packages used in Lab 6

Packages Used
<i>MoveIt</i>
RViz
Path Planning

Once the MoveIt configuration files are done, the students then go through the process of motion planning. An initial python script is given to them that plans a cartesian path to a set pose of the end effector. They execute the file on the simulated Niryo arm and view the result Figure 4.6.

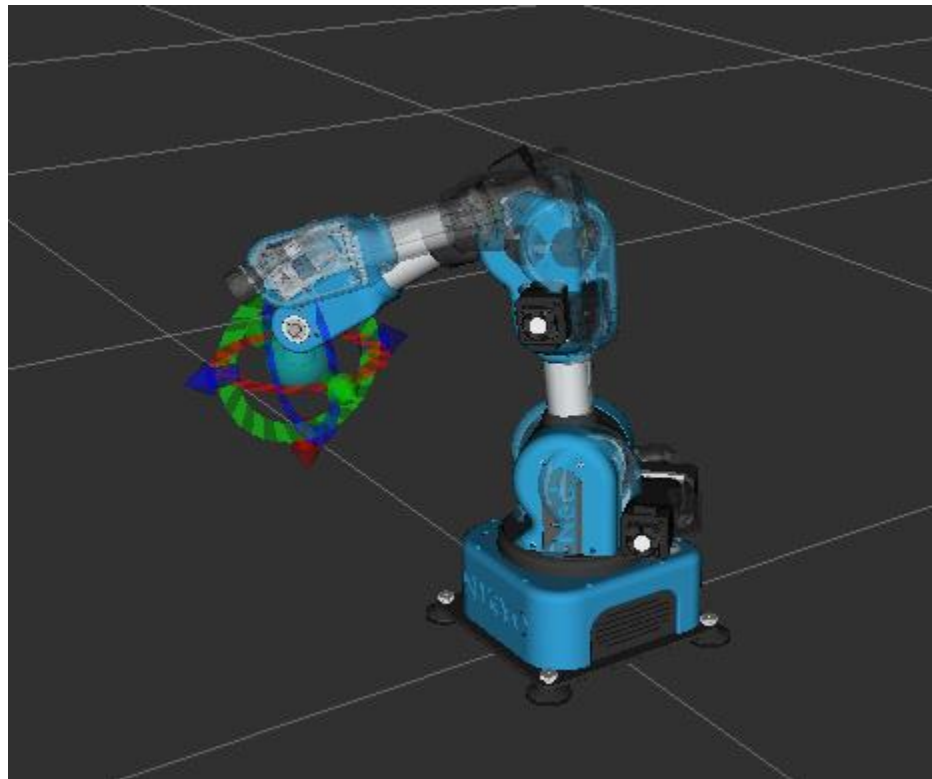


Figure 4.6: Demonstration of Niryo Ned robot planning a motion.

After students execute the initial planned movement, they must modify the script. They need to change it to have the end-effector of the robot draw a square by planning four different poses and executing it using one python script.

5 Sensing

Working with sensors is a fundamental part of robotics. This group consists of only Lab 7 and it goes back to working with the Turtlebot3. Sensors offered by Robotis are easily integrated with the Turtlebot3 and allow for students to learn how to create algorithms using sensor data without having to setup sensor interfaces.

5.1 Lab 7 (A.7)

The packages used in this lab are shown in Table 5.1.

Table 5.1 Packages used in Lab 7

Packages Used
Publisher
Subscriber
Cliff Detection (students create)

An IR sensor from Robotis is added by the students to the Turtlebot3. The sensor is going to be used by the students to allow the Turtlebot3 to detect negative obstacles. The students install the IR sensor at a 15° angle so that it can detect negative obstacles in front of the robot Figure 5.1.

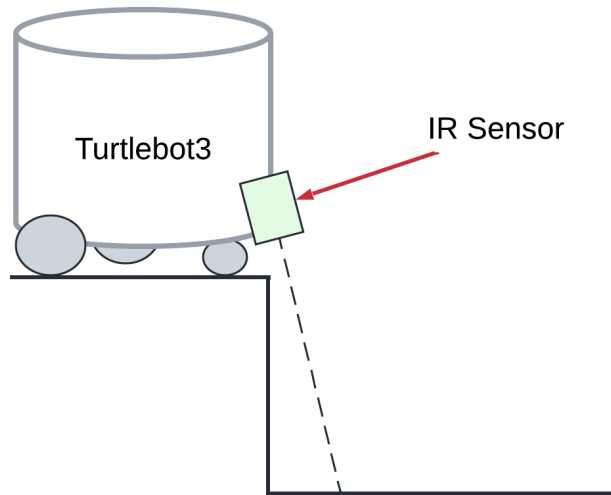


Figure 5.1: Diagram of IR sensor setup to detect negative obstacles.

At this point in the semester, more responsibility is assumed by the students to create the python scripts. The students will have to setup most of the python script with subscribers/publishers and calculating command velocities from the sensor data. Once students finish the simple algorithm, they test it on the physical Turtlebot3 in the lab.

Once they show that their simple algorithm works, they will need to improve their algorithm. This is done by accounting for angular velocity when approaching a cliff and refining the algorithm to detect a slope as well as a cliff.

6 Navigation

Robot navigation allows robots to determine their position/orientation within an environment. Labs 8 and 9 introduce the student to navigation based on the laser-distance scanner on the Turtlebot3. They do not use vision systems for navigation due to Lab 10 being the introduction to vision systems. Lab 8 is focused on the mapping of environments and how to tune the parameters in ROS to create a better map. There are some simultaneous localization and mapping (SLAM) features used in this lab, however a deep dive into the different algorithms and how SLAM works is done in lab 9.

6.1 Lab 8 (A.8)

Lab 8 introduces the students to mapping. The packages used in this lab are shown in Table 6.1.

Table 6.1 Packages used in Lab 8

Packages Used
roslaunch
Map server
Gmapping

The first part of Lab 8 is a simulation of the Turtlebot3 in a house Figure 6.1. The students use this simulation to record the data using a feature from ROS called *roslaunch*. This exposes students to recording robot data and playing back that data to use later.

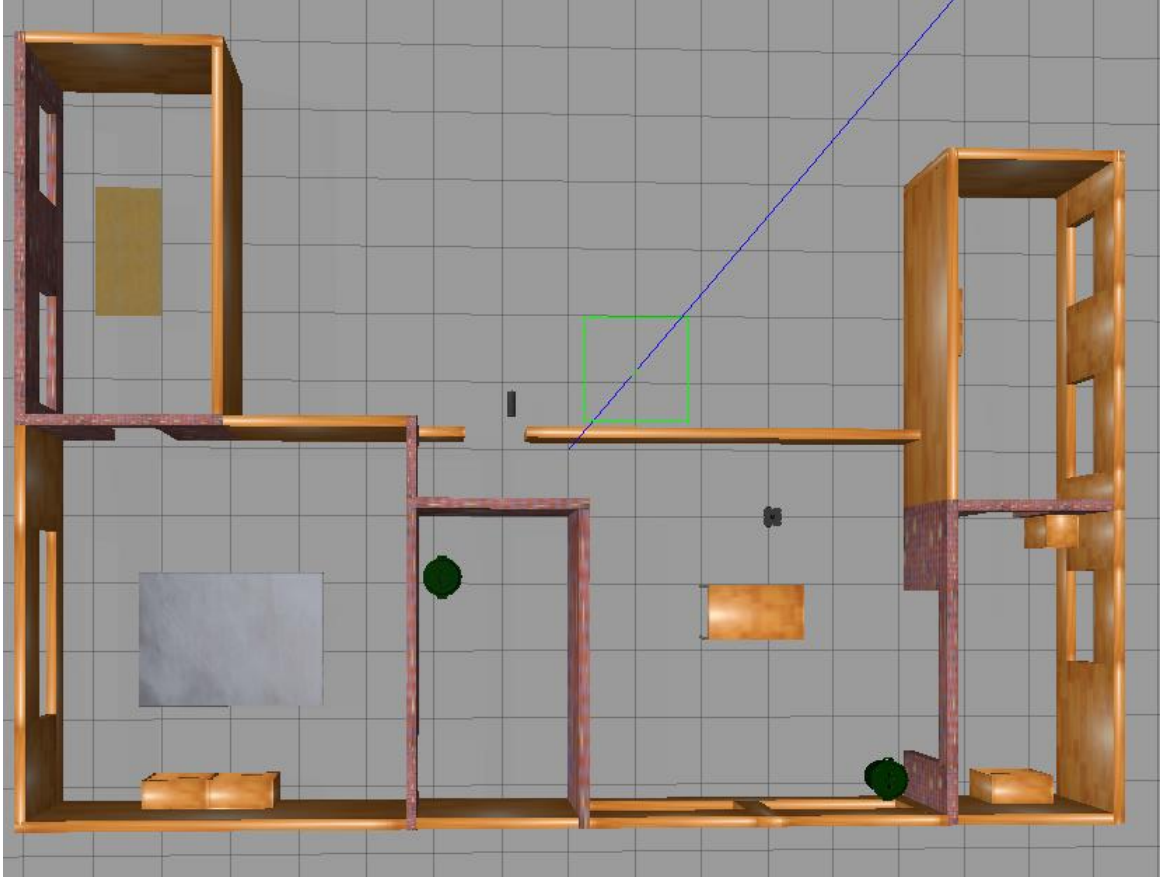


Figure 6.1: Turtlebot3 Gazebo simulation of a house for mapping.

The students record data using a teleoperation node to drive the Turtlebot around all rooms of the simulated house. Once the recording is done, they playback that data to a mapping node. Once the students have created a map, they need to tune some parameters to create a better map Figure 6.2. The parameters are angular and linear update rate and the x/y minimum and maximum values. These parameters are largely dependent on the specifications of the laser-distance scanner: its refresh rate and minimum/maximum distance.

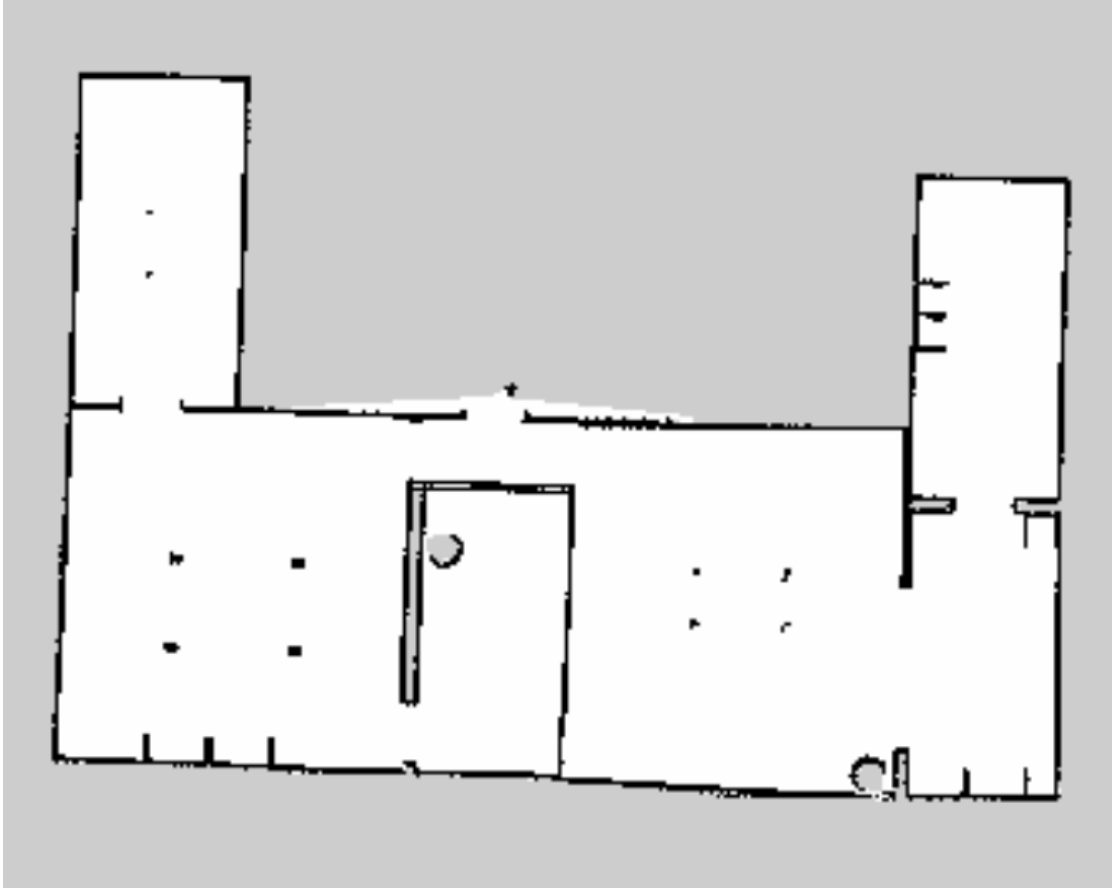


Figure 6.2: Map created after parameter optimization in the Gazebo simulation.

Once students have optimized their parameters, they repeat the same procedure for creating a map on the physical Turtlebot3. The laboratory will be setup with a maze structure for the students to map. This is done to show that there may be discrepancies between the Gazebo simulation and the physical results. These discrepancies arise out of disturbances that are not accounted for in Gazebo simulation such as light reflecting off of surfaces, sensor interference, or physical laser-distance scanner parameters being different than the simulation. The parameters that were optimized for the simulation might not be optimized for the physical mapping process. The ROS node graph for the physical Turtlebot3 using rosbag data playing back to create a map is shown in Figure 6.3.

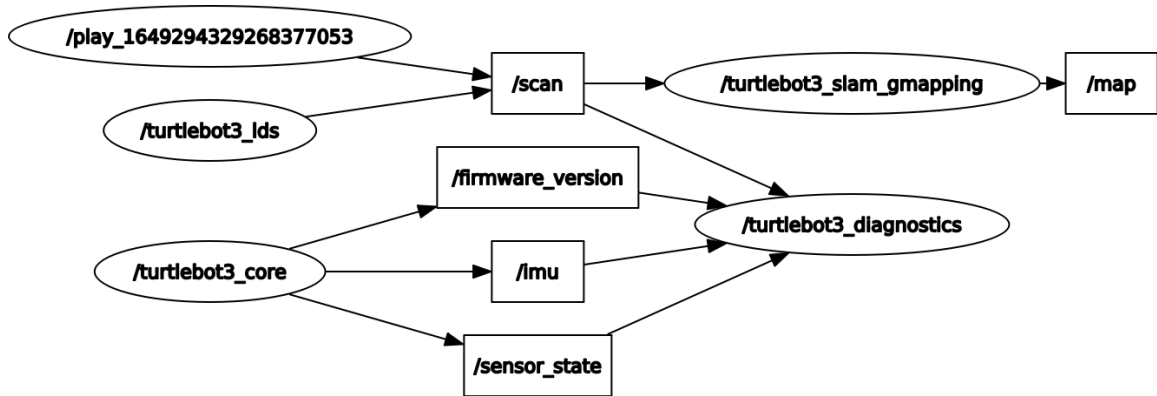


Figure 6.3 ROS node graph for Turtlebot3 mapping using rosbag to playback the data.

6.2 Lab 9 (A.9)

Lab 9 is focused on SLAM methods in ROS. The packages used in this lab are shown in Table 6.2.

Table 6.2 Packages used in Lab 9

Packages Used
Map server
Gmapping
Hector
Karto
RViz Path Planning

The students start in a simulated house to create a map Figure 6.1. There are three SLAM algorithms that the students explore in simulation. The SLAM methods and their associated algorithms are summarized in Table 6.3.

Table 6.3: SLAM methods and associated algorithms for lab 9.

SLAM Method	Algorithm
Gmapping (Figure 6.44)	Rao-Blackwellized Particle Filter (PF) with the use of odometry data, [8], [9].
Hector (Figure 6.55)	PF without the use of odometry data [9].
Karto (Figure 6.66)	Graph-based algorithm [9].

The students create a map of the same house in Gazebo simulation using all three methods. They can then select what they observe to be the best method for the Turtlebot3 in the simulation.

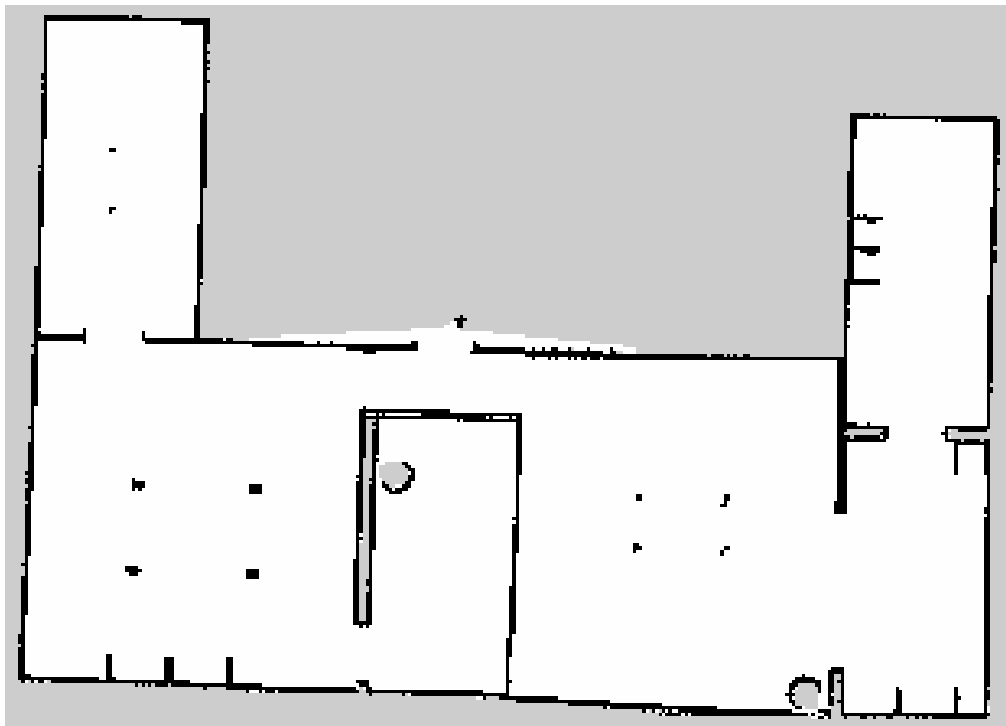


Figure 6.4 Map creating using Gmapping SLAM.

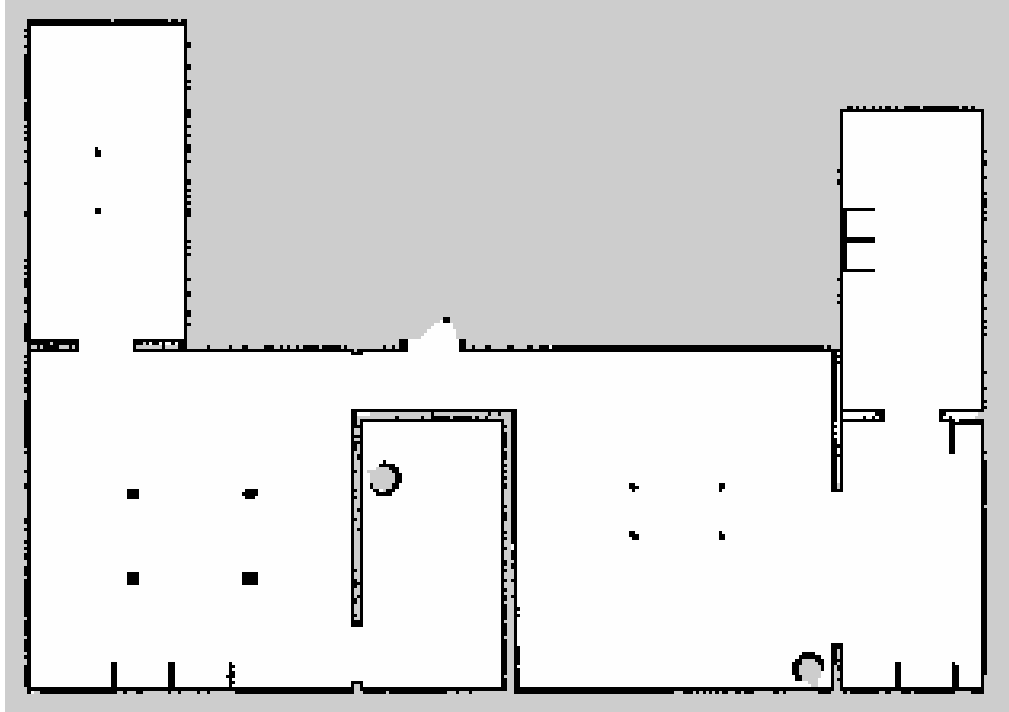


Figure 6.5 Map creating using Hector SLAM.

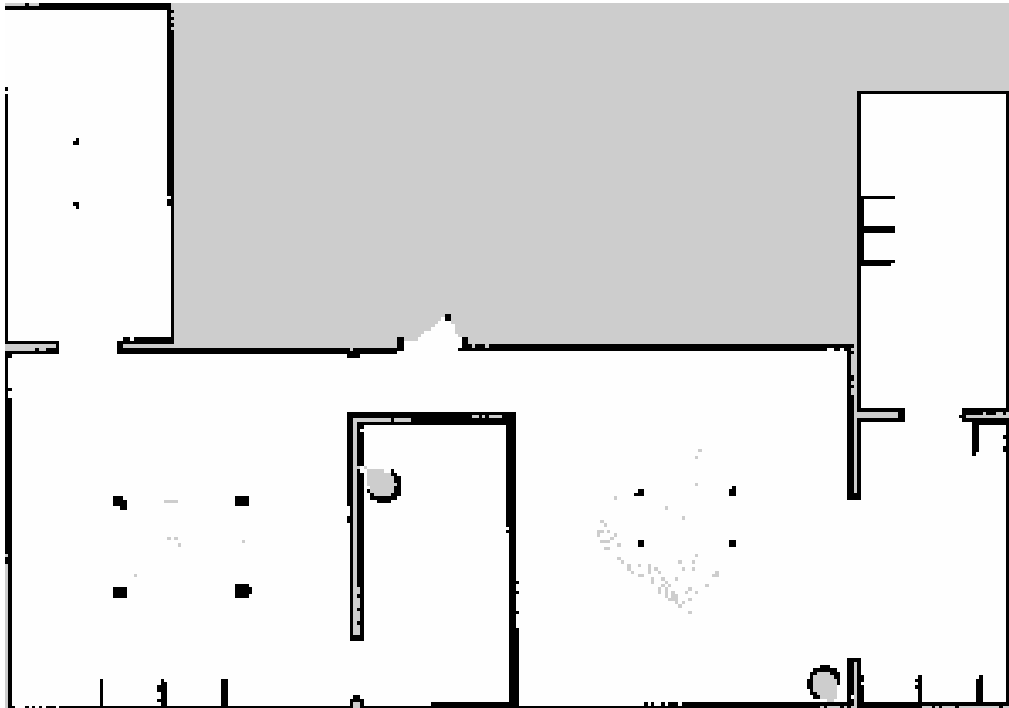


Figure 6.6 Map created using Karto SLAM.

Once they select the best SLAM method, the students complete a simulated path planning using the map they created.

After this is complete, the students use the SLAM method that they determined to be the best on the physical Turtlebot3 to map a structure in the lab. Path planning is also done on the physical robot in the map that they create. The ROS node graph for the physical Turtlebot3 using hector SLAM method to create a map is shown in Figure 6.7.

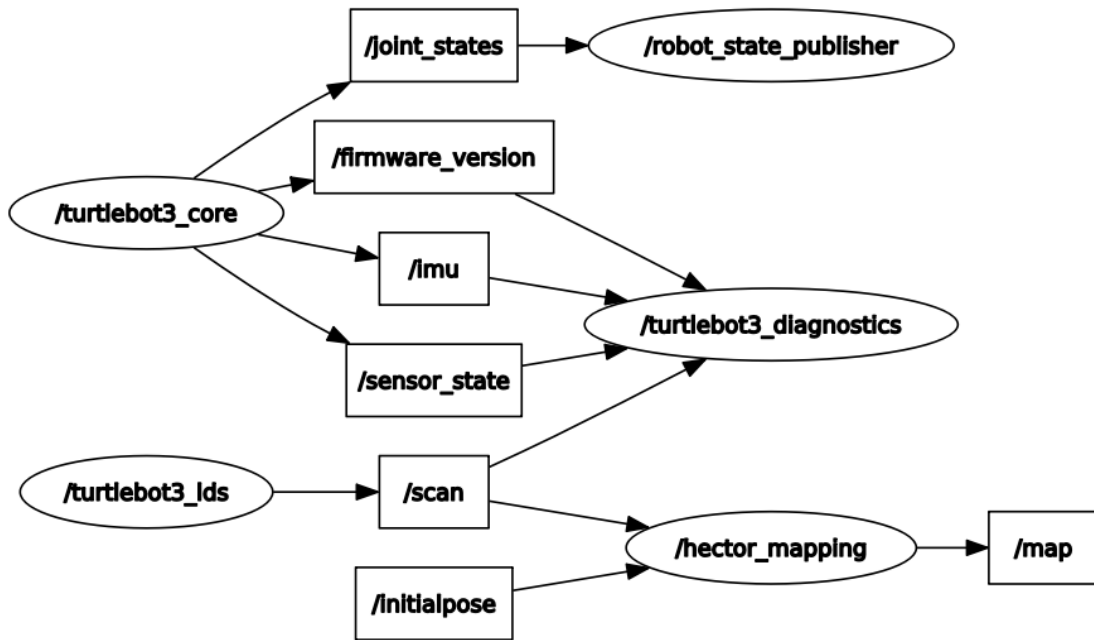


Figure 6.7: ROS node graph for Turtlebot3 running hector SLAM node.

7 Vision Systems

Before this section, the labs do not implement a vision system. Detection and navigation are achieved using a laser-distance scanner or an IR sensor. Vision systems allow students to perform a lot of different applications that weren't possible using only the laser-distance scanner. It also exposes students to the difficulties of working with cameras from distortion to calibration.

7.1 Lab 10 (A.10)

The raspberry pi camera does not work with the version of Ubuntu used for this lab. This is because the Multi-Media Abstraction Layer (MMAL) of the camera does not work with 64-bit systems. The first part of Lab 10 is calibrating the camera that is added to the Turtlebot3 burger. The packages used in this lab are shown in Table 7.1.

Table 7.1 Packages used in Lab 10

Packages Used
Turtlebot3 Autorace [10]
Pi Camera
Rqt image view
Rqt reconfigure
Camera Calibration

The intrinsic calibration uses a checkerboard pattern Figure 7.1. Intrinsic calibration accounts for parameters of the camera such as distortion, focal length, and skew. This is done using an intrinsic camera calibration GUI provided by ROBOTIS.

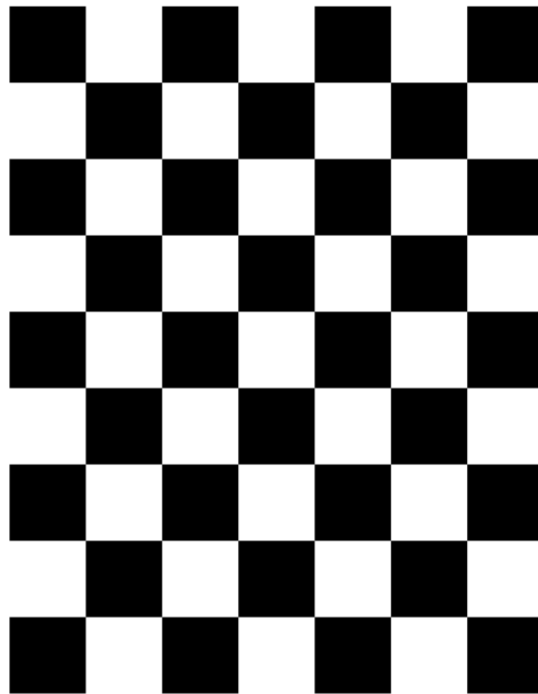


Figure 7.1 Checkerboard for intrinsic camera calibration.

Extrinsic camera calibration determines the position and imaging plane of the camera. This is done by placing the robot between two parallel lines then adjusting the max/min x and y values for the imaging plane of the camera.

The last part of the calibration is calibrating the camera to detect lanes; one white and one yellow similar to road lanes. This is done by changing the saturation, hue, and lightness values for the camera. Once students are satisfied with their calibration, they can move on to lane detection.

The students start with a lane detection package provided by Robotis that is used in their Auto Race challenges [10]. The package includes a PD controller to provide robust lane tracking. Once the students get the package working, they go to the source files of the package and change the parameters of the PD controller and comment on the change in performance.

8 Future Work and Recommendations

Ubuntu 18.04 was used to create the labs due to using [11] as a reference to learn ROS initially. In the future, the labs should be converted either to use Ubuntu 20.04 or to use ROS2. Switching to Ubuntu 20.04 will require much less work than switching to ROS2 and should be done if time is an issue. If a switch is made, all of the labs will need to be tested using the new version to ensure that the packages used are supported.

It was discussed later in this project that MATLAB may be better suited to teach students about the concepts intended for this laboratory. If it is decided that this is the direction the laboratory is to be taken, Appendix A.11 was created as an initial lab. This creates template and good starting point for the labs using MATLAB to connect to ROS. To get an overview of the connections created from MATLAB to ROS, the node graph in Figure 8.1. This node graph is with ROS running a Gazebo simulation of a Turtlebot3 and MATLAB sending command velocities to ROS.

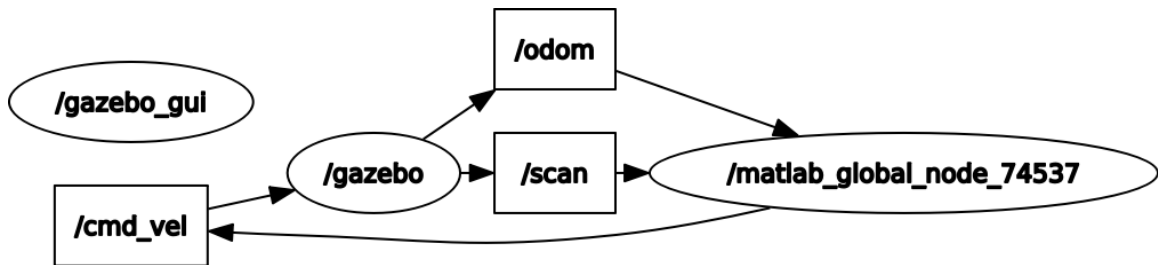


Figure 8.1 ROS node graph for MATLAB to ROS connection for Turtlebot3

MATLAB is also useful for easily plotting data that is published using ROS. An example of a plot of the laser scan data of the Turtlebot3 that is running from the node graph above is shown in Figure 8.2.

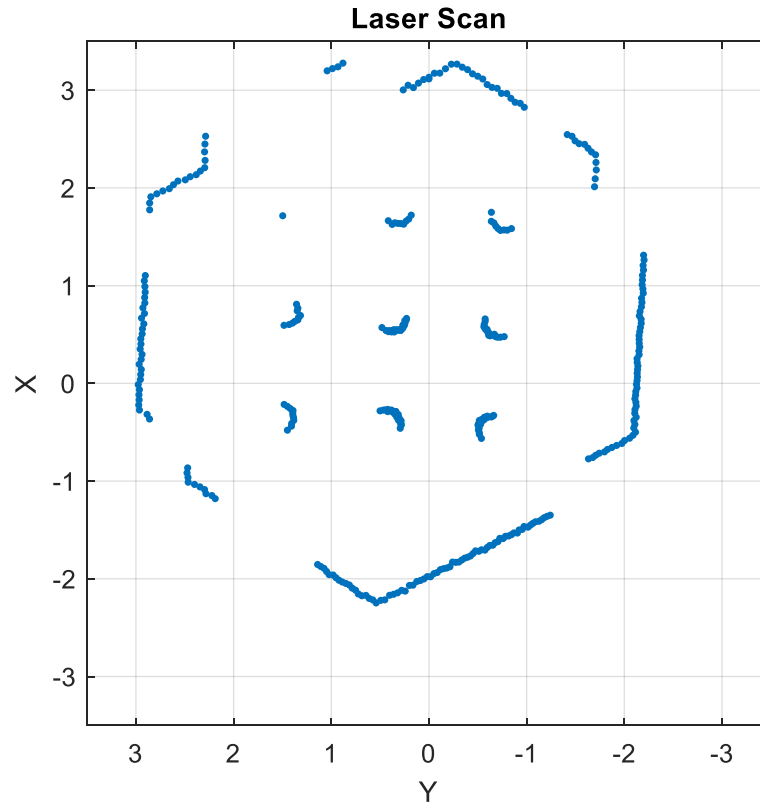


Figure 8.2: Laser scan data plotted using MATLAB using data from the Turtlebot3.

Another feature of MATLAB that could be implemented in this laboratory would be to implement controllers on the Turtlebot3 using Simulink. The results of using a trajectory controller on the Turtlebot3 is shown in Figure 8.3 and Figure 8.4.

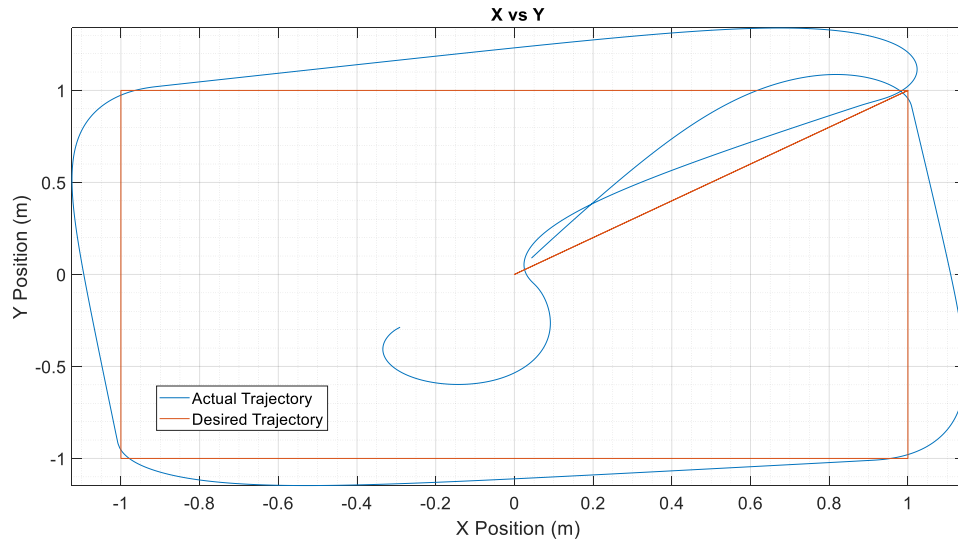


Figure 8.3 Turtlebot3 trajectory with no controller

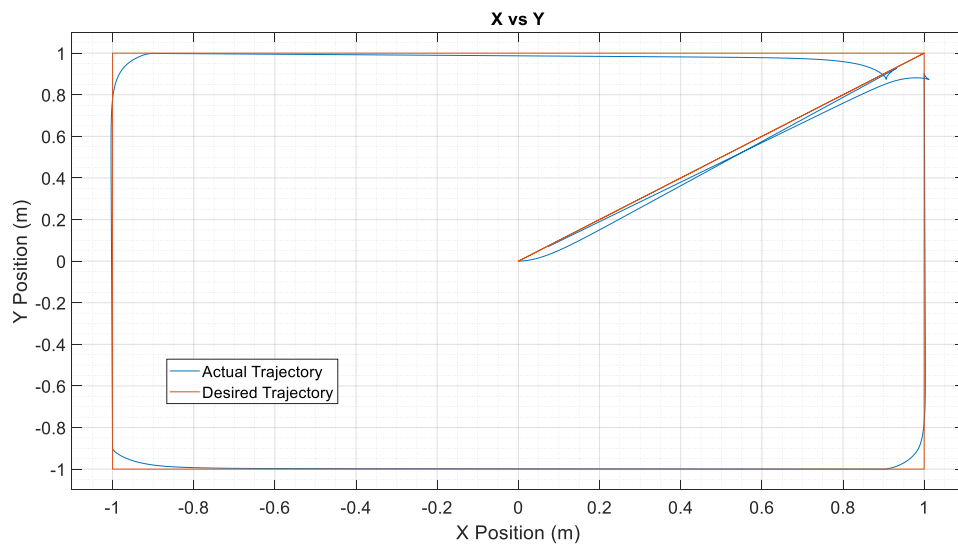


Figure 8.4 Turtlebot3 trajectory with Lyapunov controller

Some student testing with the labs was done. The students were given the instructions to complete the lab and simulations but they were not given a time-frame to complete the lab in. Therefore, some testing will need to be done with students to ensure that all of the labs are able to be completed in a two-hour time frame.

Initial feedback from students mainly involved specific commands or packages not being installed in their instance of ROS. There were also a few students who mentioned the

code being included in the lab manuals confusing so a few formatting changes were made to clearly distinguish the lines of code.

9 Reference List

- [1] T. W. Huitt, A. Killins and W. S. Brooks, "Team-Based Learning in the Gross Anatomy Laboratory Improves Academic Performance and Students' Attitudes Toward Teamwork.," *Anatomical sciences education*, vol. 8, no. 2, pp. 95-103, 2015.
- [2] S. J and R. Jobanputra, "Facilitating Group Work: To Enhance Learning in Laboratory Based Courses of Engineering Education in India," in *Association for Engineering Education - Engineering Library Division Papers*, San Antonio, 2012.
- [3] P. Weyrich et al., "Peer-Assisted Versus Faculty Staff-Led Skills Laboratory Training: a Randomised Controlled Trial.," *Medical education*, vol. 43, no. 2, pp. 113-120, 2009.
- [4] ROBOTIS, "TurtleBot 3 Burger [US]," ROBOTIS, [Online]. Available: <https://www.robotis.us/turtlebot-3-burger-us/>. [Accessed April 2022].
- [5] Niryo, "Ned Education Research Cobot," [Online]. Available: <https://niryo.com/product/ned-education-research-cobot/>. [Accessed April 2022].
- [6] G. Magyar, P. Sinčák and Z. Krizsán, "Comparison Study of Robotic Middleware for Robotic Applications," in *In: Sinčák, P., Hartono, P., Virčíková, M., Vaščák, J., Jakša, R. (eds) Emergent Trends in Robotics and Intelligent Systems. Advances in Intelligent Systems and Computing*, https://doi.org/10.1007/978-3-319-10783-7_13, 2015.
- [7] MoveIt , May 2020. [Online]. Available: <https://moveit.ros.org/>. [Accessed 4 April 2022].
- [8] G. Grisetti, C. Stachniss and W. Burgard, "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters," *IEEE transactions on robotics*, vol. 23, no. 1, pp. 34-46, 2007.
- [9] J. M. Santos, D. Portugal and R. P. Rocha, "An evaluation of 2D SLAM techniques available in Robot Operating System," in *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, 2013.
- [10] Robotis, "TurtleBot3 AutoRace 2019," 2019. [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/autonomous_driving_auto_race/. [Accessed April 2022].

- [11] M. Quigley, B. Gerkey and W. D. Smart, Programming Robots with ROS, O'Reilly Media Inc., 2015.

A Lab Manuals

A.1 Lab 1 ROS Fundamentals

1. Setup

Before any code can be written, a catkin workspace must be created for the code to live in. Then, we can create packages and nodes within our workspace. To create a workspace called lab01, run the following lines of code in a command terminal.

```
source /opt/ros/melodic/setup.bash
mkdir -p ~/lab01/src
cd ~/lab01
catkin_init_workspace src
catkin_make
source devel/setup.bash
```

To save you some work, use these lines to edit the bashrc file.

```
echo source ~/lab01/devel/setup.bash >> ~/.bashrc
source ~/.bashrc
```

A ROS package resides inside the src directory of the catkin workspace we created. ROS packages contain nodes which will contain the python code. To create a package called learning_tf2, execute the following lines of code in your open command terminal.

```
cd ~/lab01/src
catkin_create_pkg learning_tf2 tf2 tf2_ros rospy turtlesim
cd ~/lab01
catkin_make
source ./devel/setup.bash
```

Next, a node will need to be created in the new ROS package with these two lines of code.

```
cd ~/lab01/src/learning_tf2/src
mkdir nodes
```

2. TF Listener and Talker Python Script

ROS uses a package called tf2 to implement coordinate transforms. The following two python scripts create two “turtles”. One turtle is the “talker”, and the other is the “listener”. Suppose that there is a coordinate that is the “world” coordinate at the point (0,0,0) and the “talker” turtle is moving around in the world coordinate system. The ROS tf2 package will perform coordinate transformation to tell the “listener” turtle where the “talker” turtle is.

This is the “talker” turtle script which will become a script in our node. Save the file **turtle_tf2_broadcaster.py** in the following location:
/home/username/lab01/src/learning_tf2/src/nodes

Save the file **turtle_tf2_listener.py** in the same nodes folder as the talker script

Now that the python scripts are in our node, we need to tell ROS that these scripts are executable. To do this, return to your command terminal and execute these two lines.

```
chmod +x nodes/turtle_tf2_broadcaster.py
chmod +x nodes/turtle_tf2_listener.py
```

A launch file must be created for our package to be executed. The following command will create a launch folder.

```
mkdir launch
```

Save the code **start_demo.launch** in the launch folder created above. The location should be in /home/username/lab01/src/learning_tf2/src/launch

One last thing must be done before we execute our package. The following lines of code will open a file called CMakeLists.txt

```
cd ~/lab01/src/learning_tf2
gedit CMakeLists.txt
```

Scroll down until you see the “Install”

Like this:

```
#####
## Install ##
#####
```

Under “Install”, copy and paste this code, then click save.

When you are done, don’t close the text editor as we will need to modify it later.

```
install(
  PROGRAMS
  nodes/turtle_tf2_broadcaster
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})

install(
  PROGRAMS
  nodes/turtle_tf2_listener
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
```

3. Launch Package and View Simulation

These two lines of code will launch the package we created. After you execute these commands in the terminal, a simulation window will appear.

```
cd ~/lab01
roslaunch learning_tf2 start_demo.launch
```

If this is the first time you are running a ROS simulation, it may take a minute or two to appear. After it appears, if everything is working correctly, you should see two turtles. Adjust your screen to view the command terminal and the simulation. To move the “talker” turtle, use the arrow keys on your computer. Note: the command terminal must be the active window to move the turtle. You should see the “listener” turtle follow the “talker” turtle in the simulation. Don’t close your simulation and move onto the next step.

Since the tf command can be quite confusing, ROS offers several tools to visualize the command. We will explore the tf tree and RViz. First, the tf tree. To view the current tf tree of the package, first open a new command terminal tab and enter these two lines.

```
roslaunch tf2_tools view_frames.py
evince frames.pdf
```

This should open a pdf file. Take a screenshot of the tf tree to show to your TA

TA Initials: _____ Date: _____

Press ctrl+C in the current command window (the one with the tf tree) to close the pdf. Run the following command to open RViz. Once RViz is open, you should see three coordinate axes. One for the “world”, one for the “listener” turtle, and one for the “talker”

turtle. Try moving the turtles in the simulation, and watch the RViz coordinate axes move in coordination.

```
roslaunch rviz rviz -d `rospack find turtle_tf`/rviz/turtle_rviz.rviz
```

After viewing the RViz simulation, stop the simulations in both command terminal tabs by typing ctrl+C into both terminals.

4. Add Static Coordinate Frame

The process before this added two dynamic coordinate frames (the turtles). Dynamic meaning that the coordinate frames move with respect to the “world” frame. Another useful skill in ROS is adding a static coordinate frame. This could be used to add the position of a laser on a robot with respect to its center. In our case, we are going to add a baby turtle on top of the “talker” turtle.

This process is very similar to adding the dynamic coordinate frames. Save the given file **fixed_tf2_broadcaster.py** in the same nodes folder as before.

Tell ROS that the python file is executable with these commands.

```
cd ~/lab01/src/learning_tf2/src
chmod +x nodes/fixed_tf2_broadcaster.py
```

Next, we need to add this new program to the CMakeLists file..

```
cd ~/lab01/src/learning_tf2
gedit CMakeLists.txt
```

Add these lines of code below the ones you added earlier.

```
install(
  PROGRAMS
    nodes/fixed_tf2_broadcaster
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
```

Finally, the launch file must be changed.

```
cd ~/lab01/src/learning_tf2/src/launch
gedit start_demo.launch
```

Once the launch file is open, copy/paste this code into the file.

```
<node pkg="learning_tf2" type="turtle_tf2_listener.py" name="listener" />
  <node pkg="learning_tf2" type="fixed_tf2_broadcaster.py"
    name="broadcaster_fixed" output="screen"/>
```

As before, launch the package with this line in your terminal. Again, a simulation will pop up. The simulation will appear the same as before with the two turtles. The difference after adding the static coordinate frame will be seen in the tf tree and in RViz.

```
roslaunch learning_tf2 start_demo.launch
```

View tf tree and take a screenshot to show your TA. Don't forget that these lines will need to be run in the terminal window that isn't running the package simulation.

```
rosviz tf view_frames
evince frames.pdf
```

Comment on the difference between this tf tree and the one made in part 3.

TA Initials: _____ Date: _____

To view RViz, execute this code.

```
rosviz rviz rviz -d `rospack find turtle_tf`/rviz/turtle_rviz.rviz
```

To finish, another useful ROS tool is the tf_echo. This command will output the transformation between any two coordinate frames in your command window.

```
rosviz tf_echo [reference_frame] [target_frame]
```

For example, you could view the coordinate transformation between the two turtles.

```
rosviz tf_echo turtle1 turtle2
```

Or you could view the coordinate transformation between the "talker" turtle and the baby turtle.

```
rosviz tf_echo turtle1 baby_turtle
```

TA Initials: _____ Date: _____

Items to submit:

Part 3: TF tree screenshot

Part 4: TF tree screenshot and comments

A.2 Lab 2 Topics Services and Actions

1. Topics: Publisher and Subscriber

Create a Catkin workspace called “lab02”.

```
source /opt/ros/melodic/setup.bash
mkdir -p ~/lab02/src
cd ~/lab02
catkin_init_workspace src
catkin_make
source devel/setup.bash
```

Next, create a package called “learning_sub” using the following command lines. This package has dependencies on std_msgs and rospy.

```
cd ~/lab02/src
catkin_create_pkg learning_sub std_msgs rospy
cd ~/lab02
catkin_make
source ../devel/setup.bash
```

To save us some work, we are going to edit our bashrc file.

```
gedit ~/.bashrc
```

Add this line at the end then save and close.

```
source ~/lab02/devel/setup.bash
```

Now we need to create a folder in our package to hold the publisher and subscriber.

```
source ~/.bashrc
roscd learning_sub
mkdir scripts
cd scripts
```

The publisher (also called a “talker” in some cases) is defined as a Node in ROS. For this lab, our publisher is going to print a message to the command terminal, to the Node’s log file, and to roscat using a topic we are going to call “robotics”. Save the given file talker.py into the folder /lab02/src/learning_sub/scripts.

Don’t forget to tell ROS that this file is executable by using the following line in the command terminal:

```
chmod +x talker.py
```

Next, create the subscriber (or “listener”) Node. The subscriber subscribes to the “robotics” topic that was defined in the publisher. It then adds a “I heard” in front of whatever message it receives from the publisher.

Use the same method as the publisher to save the given file listener.py into the scripts folder.

Don’t forget to tell ROS that this script is executable using the `chmod +x` command.

Next we need to modify our “CMakeLists.txt” file to add our two python scripts.

```
cd ~/lab02/src/learnin_sub
gedit CMakeLists.txt
```

Add the following lines of code to the file then save and close.

```
catkin_install_python(PROGRAMS scripts/talker.py scripts/listener.py
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)
```

Now that we have the publisher and subscriber Nodes made, we need to use the “`catkin_make`” command to build them.

```
cd ~/lab02
catkin_make
source ../devel/setup.bash
```

Before we run the publisher and subscriber, we need to start roscore. After you start roscore, open a new terminal tab and run the publisher. Take note that you need to run your source files for each new command terminal.

```
roslaunch learning_sub talker.py
```

You should see the timestamp followed by “robotics is cool” scrolling on the terminal. In a new command terminal, run the “listener” Node:

```
roslaunch learning_sub listener.py
```

You should see the timestamp followed by “I heard robotics is cool” scrolling on the terminal. Type `ctrl+C` into ONLY the terminals running the publisher and subscriber nodes. Don’t `ctrl+C` roscore yet.

For the final part on subscribers and publishers, go into the talker.py script and change the rate at which the messages are published to 1Hz. Run the two nodes again and ensure that your changes are correct.

Show your TA the messages being produced on your terminals to get a signoff. Then press ctrl+C in all command terminals.

TA Initials: _____ Date: _____

2. Services

Before we can run our service and client we need to create a srv and msg. To make things easier, we are going to copy the srv from the rospy_tutorials package using the roscp command.

```
roscd learning_sub
mkdir srv
roscp rospy_tutorials AddTwoInts.srv srv/AddTwoInts.srv
mkdir msg
echo "int64 num" > msg/Num.msg
```

Next open up package.xml using the command:

```
gedit package.xml
```

Add the following two lines to the file to add message generation to our package. Don't forget to save and close the file when you are done.

```
<build_depend>message_generation</build_depend>
<exec_depend>message_runtime</exec_depend>
```

Now open the CMakeLists.txt file using the gedit command. Edit your file to add "message_generation" to the find_package command as shown below.

```
roscd learning_sub
gedit CMakeLists.txt
```

```

find_package(catkin REQUIRED COMPONENTS
  rospy
  std_msgs
  message_generation
)

```

Then scroll down until you see **## Generate services in the 'srv' folder** and add the following code then save and close.

```

add_message_files(
  FILES
  Num.msg
)
generate_messages(
  DEPENDENCIES
  std_msgs
)
add_service_files(
  FILES
  AddTwoInts.srv
)
catkin_package(
  CATKIN_DEPENDS message_runtime
)

```

Now that we have defined our service and messages, we can add our service and client nodes.

First, we are going to create a service that will add two integers that you give it. Save the given code “add_two_ints_server.py” into the folder lab02/src/learning_sub/scripts

Make the node executable using the chmod command and then add the following code to the CMakeLists.txt file.

```

roscd learning_sub
gedit CMakeLists.txt

```

```

catkin_install_python(PROGRAMS scripts/add_two_ints.py
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)

```

Next the client node needs to be created called “add_two_ints_client.py”. Save the code in the same scripts folder.

Next, as before, make the node executable using the `chmod` command and then edit your `CMakeLists.txt` as shown below.

```
catkin_install_python(PROGRAMS scripts/add_two_ints_server.py
  scripts/add_two_ints_client.py
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)
```

Now we can run our service and client. First, we need to use the `catkin_make` command in our workspace.

```
cd ~/lab02
catkin_make
source ./devel/setup.bash
```

In a new command terminal tab start `roscore`.

Run the server node using the `roslaunch` command.

```
roslaunch learning_sub add_two_ints_server.py
```

You should see the command terminal say “Ready to add two ints.”

Open a new terminal tab to run the client.

```
roslaunch learning_sub add_two_ints_client.py 1 3
```

This command runs the client and tells it to add 1 and 3. You should see the sum of these two integers output in the terminal. Try testing with other integers to ensure it is working. After showing your TA, `ctrl-C` your active terminals.

TA Initials: _____ Date: _____

3. Actions

We are going to create an action server that takes sensor data and computes the average and standard deviation of a requested amount of samples. To do this, we need to create another package called `learning_action` and a folder inside the package called `action`.

```
cd ~/lab02/src
catkin_create_pkg learning_action actionlib message_generation roscpp rospy
std_msgs actionlib_msgs
roscd learning_action
mkdir action
cd action
```

Create a file using the “gedit” command called “Averaging.action”. This file defines the goal, result, and feedback of the action. Copy and paste the following code into the file then save and close.

```
#goal definition
int32 samples
---
#result definition
float32 mean
float32 std_dev
---
#feedback
int32 sample
float32 data
float32 mean
float32 std_dev
```

Save and close the text file then use catkin_make on your workspace.

```
cd ~/lab02
catkin_make
```

Save the given code “averaging_server.cpp” into the folder /lab02/src/learning_action/src.

Type the following command to open the CMakeLists file then copy and paste the following code into the file.

```
roscd learning_action
gedit CMakeLists.txt
```

```
find_package(catkin REQUIRED COMPONENTS actionlib std_msgs
message_generation)
add_action_files(DIRECTORY action FILES Averaging.action)
generate_messages(DEPENDENCIES std_msgs actionlib_msgs)

add_executable(averaging_server src/averaging_server.cpp)
target_link_libraries(averaging_server ${catkin_LIBRARIES})
```

Now that the action server is made, we need to build our package. Note that it may take a few minutes for the package to be built. That is ok.

```
cd ~/lab02
catkin_make
```

Open a new terminal and start roscore.
Next, use rosrn to run the action server.

```
roslaunch learning_action averaging_server
```

Run rostopic and the rqt_graph to ensure that the action server is working.

```
rostopic list -v
```

```
roslaunch rqt_graph rqt_graph &
```

Ctrl-C the action server then move on to create the action client.

Save the given code “averaging_client.cpp” into the folder /lab02/src/learning_action/src.

Open learning_action/CMakeLists.txt and add the following code.

```
roscd learning_action
gedit CMakeLists.txt
```

```
add_executable(averaging_client src/averaging_client.cpp)
target_link_libraries(averaging_client ${catkin_LIBRARIES})
```

Before we can run the action client, we need to build our package.

```
cd ~/lab02
catkin_make
```

As before, in a new terminal, start roscore and then run the action client.

```
roslaunch learning_action averaging_client
```

Your terminal should output “Waiting for the action server to start”

View the rostopic list and the rqt graph to verify operation.

```
rostopic list -v
```

```
roslaunch rqt_graph rqt_graph &
```


Ctrl-C the action client and then move on.

Now that the action server and client are created, we should test that it works with some made-up sensor data. First we need to create a scripts folder for our sensor code to live in.

```
roscd learning_action
mkdir scripts
cd scripts
gedit best_sensor_ever.py
```

In the text editor, copy and paste the following code, then save and close.

```
#!/usr/bin/env python

import rospy
from std_msgs.msg import Float32
import random
def gen_number():
    pub = rospy.Publisher('random_number', Float32)
    rospy.init_node('random_number_generator', log_level=rospy.INFO)
    rospy.loginfo("Generating random numbers")

    while not rospy.is_shutdown():
        pub.publish(Float32(random.normalvariate(5, 1)))
        rospy.sleep(0.05)

if __name__ == '__main__':
    try:
        gen_number()
    except Exception, e:
        print "done"
```

Don't forget to make the node executable using the `chmod +x` command.

As always, we need to use the `catkin_make` command to compile the python script. Then we can run our sensor script using the `roslaunch` command.

```
cd ~/lab02
catkin_make
```

```
roslaunch learning_action best_sensor_ever.py
```

In a new terminal, run the action server.

```
roslaunch learning_action averaging_server
```

Open a new terminal, and view the action feedback.

```
rostopic echo /averaging/feedback
```

In a new terminal, run the action client.

```
roslaunch learning_action averaging_client
```

Open a new terminal, and view the action result.

```
rostopic echo /averaging/result
```

Run the action client several times and view how the action result changes.

In a new terminal, view the rqt graph and show your TA.

```
roslaunch rqt_graph rqt_graph &
```

Note that the “&” at the end allows the rqt_graph to run asynchronously so that we can type in commands in the terminal while leaving the graph open.

TA Initials: _____ Date: _____

Items to submit:
Part 3: rqt_graph

A.3 Lab 3: Simulation

1. Gazebo Simulation Environment

Next we will create a workspace for all of our code to live in called lab03. If you do not remember how to create a workspace, go back to Lab01 or Lab02. Don't forget to run `catkin_make` and source your `.bash` file after you create the workspace.

To save us some work, we are going to edit our `bashrc` file.

```
gedit ~/.bashrc
```

Add these lines at the end then save and close.

```
source ~/lab03/devel/setup.bash
export TURTLEBOT3_MODEL=burger
```

Now you need to install the packages needed for `turtlebot3`.

```
sudo apt-get install ros-melodic-joy ros-melodic-teleop-twist-joy \
ros-melodic-teleop-twist-keyboard ros-melodic-laser-proc \
ros-melodic-rgbd-launch ros-melodic-depthimage-to-laserscan \
ros-melodic-rosserial-arduino ros-melodic-rosserial-python \
ros-melodic-rosserial-server ros-melodic-rosserial-client \
ros-melodic-rosserial-msgs ros-melodic-amcl ros-melodic-map-server \
ros-melodic-move-base ros-melodic-urdf ros-melodic-xacro \
ros-melodic-compressed-image-transport ros-melodic-rqt* \
ros-melodic-gmapping ros-melodic-navigation ros-melodic-interactive-
markers
```

```
sudo apt-get install ros-melodic-dynamixel-sdk
```

```
sudo apt-get install ros-melodic-turtlebot3-msgs
```

```
sudo apt-get install ros-melodic-turtlebot3-*
```

```
catkin_make
```

```
cd ~/lab03/src
```

```
git clone -b melodic-devel https://github.com/ROBOTIS-
GIT/turtlebot3_simulations.git
git clone https://github.com/turtlebot/turtlebot.git
cd ~/lab03
catkin_make
```

First we are going to launch a robot called a turtlebot into an empty Gazebo simulation. You will be using a non-simulated turtlebot later in this lab so these simulations will get you used to how the robot looks and operates. To launch the simulation, run these commands.

```
roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
```

It may take several minutes to launch the simulation if this is your first time doing simulations with ROS. After the simulation launches, you should see a turtlebot in an empty environment and you should not be able to move it. To move it, we need to use the teleoperation node. Open a new terminal and execute the following command.

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Move the turtlebot around in the empty world using the keys as shown in the command terminal. The empty world is kind of boring, so ctrl-C on the terminal with the Gazebo simulation then execute the following command.

```
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

You should see a new simulation start with objects that form a turtle with the turtlebot inside. Move the turtlebot around to get used to the controls (you will need to relaunch the teleoperation node).

View the linear and angular velocity in real-time with the `rqt_plot` command.

```
rqt_plot cmd_vel/linear/x cmd_vel/angular/z
```

After you launch the `rqt_plot`, click back onto the terminal running the teleoperation node and move the robot around a bit. If you go back to the `rqt_plot` tab, you will notice that it is very hard to see much that is going on. There are some changes you can make so that you can view the plots easier. First, in the upper right hand corner, uncheck autoscroll. Then, right above the plot click on the arrow picture going up. Once you click on this, change your x and y scales to view the plot. Change your title to be your name, and the section you are on in the lab. Example: Chelsey Spitzner Lab 3 Part 1.



Save an image of your plot using the save button to show your TA and to submit later.

The turtlebot that we will use later in this lab has a LiDAR scanner on the top. We can visualize what the simulated scanner sees using the RViz command. Open a new terminal and run the command.

```
roslaunch turtlebot3_gazebo turtlebot3_gazebo_rviz.launch
```

Move the turtlebot around and watch as the laser scan changes as it detects different objects in its field of view.

View tf tree and take a screenshot to show your TA.

```
roslaunch tf view_frames  
evince frames.pdf
```

TA Initials: _____ Date: _____

2. Teleop-Bot

Now we are going to create our own teleoperation node and compare it to the one you just used.

To do this, we first need to create a node that will publish our keystrokes to be interpreted into velocity commands. Save the given code “key_publisher.py” in /lab03. Don’t forget to use the `chmod +x` command to make the code executable.

Save the given file “ramped_velocity.py” into the /lab03 folder.

The code below is mostly done, but you need to fill out the `key_mapping`. The characters are already filled in, but you will need to map them to linear or angular velocities. The mapping is as follows: ‘**character**’: [**angular velocity**, **linear velocity**]. Use only 0, 1, or -1 to map the characters to a velocity.

When you have finished the code, save and close then use the `chmod +x` command to make the code executable.

In a new terminal, start roscore. Then launch a turtlebot simulation.

```
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

Next we need to run our keyboard publisher and teleoperation program in new terminals.

```
./key_publisher.py
```

```
./ramped_velocity.py
```

Arrange your screens to be split so that you can see the Gazebo simulation and your command terminal. With the command terminal that is running the key publisher, type in some command (w,a,d,x) and watch the robot move. As you can see, the robot is moving way too much for each command. To change this, open your velocity_ramped.py and change the “g_vel_scales” and “g_vel_ramps” values.

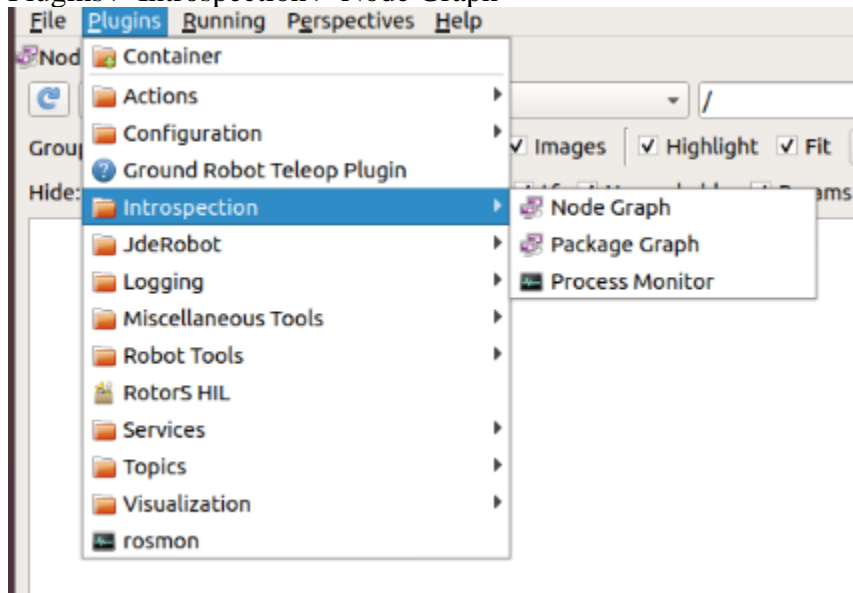
```
g_vel_scales = [0.25, 0.25] # units meters per second  
g_vel_ramps = [0.5, 0.5] # units: meters per second^2
```

After saving your script, ctrl-C the terminal running the velocity_ramped.py and rerun the script. Try moving the robot around again and it should be much easier to control without bumping into objects.

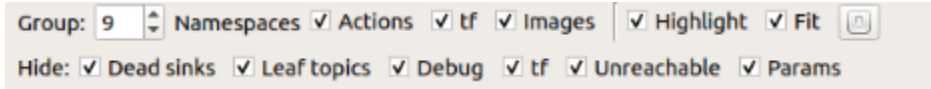
To get a feel for what all is running, run each of the following commands. Note you will have to run them one at a time and ctrl-C after each one to run the next command.

```
rostopic list  
rostopic echo cmd_vel  
rqt_graph
```

If your rqt_graph will not run, type `rqt` in the terminal. Once the window pops up, go to `Plugins > Introspection > Node Graph`



To make the node graph easier to read, under, Hide, select all as shown below



View the linear and angular velocity in real-time with the `rqt_plot` command.

```
rqt_plot cmd_vel/linear/x cmd_vel/angular/z
```

After you launch the `rqt_plot`, click back onto the terminal running the `key_publisher` node and move the robot around a bit. You may need to change your x and y scales again to see the plot clearly. Change your title to be your name, and the section you are on in the lab. Example: Chelsey Spitzner Lab 3 Part 2. After you get some good motion showing on your plot, **save an image using the save button**. Show your TA your `rqt_plot` and your Gazebo simulation.

TA Initials: _____ Date: _____

3. Obstacle Avoidance Algorithm

For our final item in this lab, you are going to develop a simple obstacle avoidance algorithm. First start off by creating a package in our workspace.

```
cd ~/lab03/src
catkin_create_pkg obstacle_avoidance rospy
cd ~/lab03
catkin_make
source ./devel/setup.bash
```

Now we need to create a launch file for our code to work.

```
roscd obstacle_avoidance
mkdir launch
roscd obstacle_avoidance/launch
gedit naive_obs_avoid.launch
```

Copy and paste the following code into the launch file then save and close.

```
<launch>
  <node name="obstacle_avoidance_node" pkg="obstacle_avoidance">
```

```
type="obs_avoid.py" />
</launch>
```

Now that our package is set up, we can go about creating our algorithm. First create the file and then copy and paste the code into the text editor.

```
cd ~/lab03/src/obstacle_avoidance/src
gedit obs_avoid.py
```

Save the given code “obs_avoid.py” into the folder /lab03/src/obstacle_avoidance/src.

To help you out, we have given you the bulk of the code for this algorithm. Take your time to understand what each part of the code is doing. The only thing you need to do is figure out the move commands `move.linear.x` and `move.linear.y` for each case. **Only use the numbers 0.0 and 0.5 for these values.** Think about what direction you want the robot to go if there are no obstacles in front of it and what it should do if it does detect obstacles in front or to the sides.

Once you have figured out the values, save and close the file and then set permissions using the `chmod +x` command.

Now we can compile our workspace and run the package.

```
cd ~/lab03
catkin_make
source ./devel/setup.bash
```

Don't forget to start a new terminal with `roscore` and then start a simulation in Gazebo.

```
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

Now we can launch our obstacle avoidance script.

```
roslaunch obstacle_avoidance naive_obs_avoid.launch
```

Once you launch the script, your turtlebot should start moving in the Gazebo simulation. View the linear and angular velocity in real-time with the `rqt_plot` command.

```
rqt_plot cmd_vel/linear/x cmd_vel/angular/z
```

You may need to change your x and y scales again to see the plot clearly. Change your title to be your name, and the section you are on in the lab. Example: Chelsey Spitzner Lab 3 Part 3. After you get some good motion showing on your plot, **save an image using the save button**. Show your TA your `rqt_plot` and your Gazebo simulation.

TA Initials: _____ Date: _____

Items to submit:

Part 1: rqt_plot image

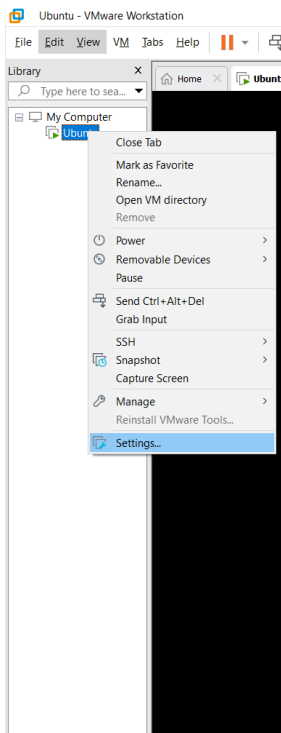
Part 2: rqt_plot image

Part 3: rqt_plot image and obs_avoid.py script

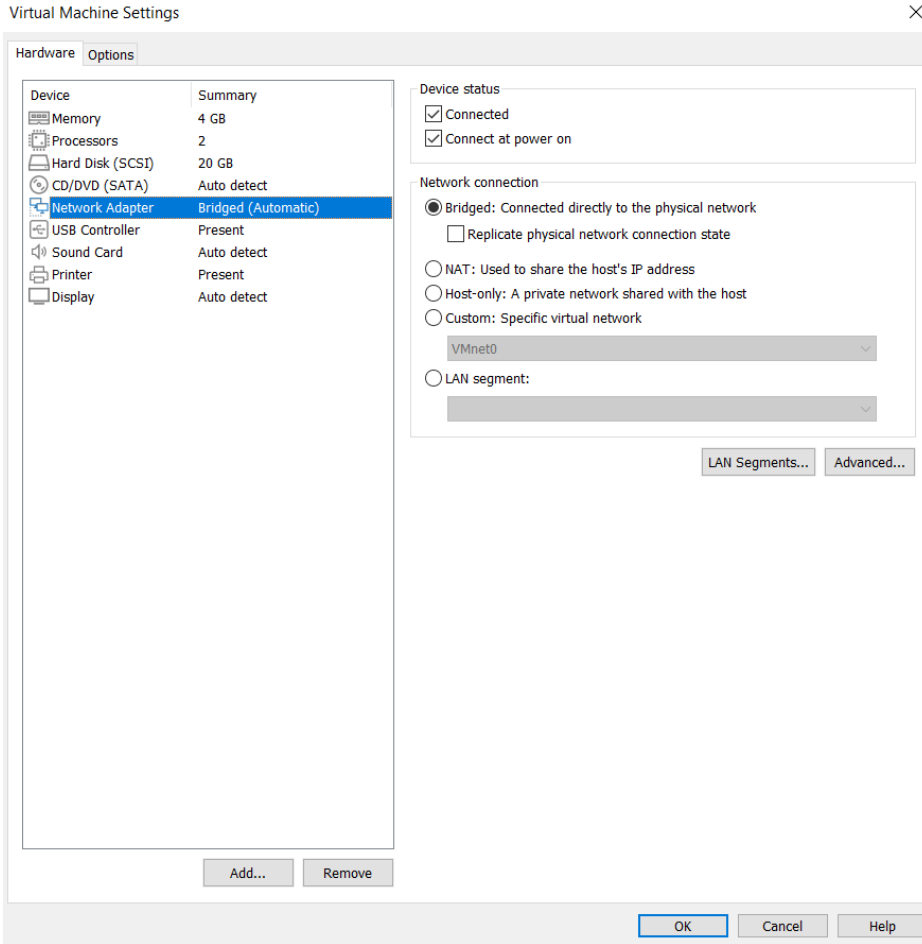
A.4 Lab 4: Mobile Motion

1. Setup Connections

If you are using VMware to set up a virtual machine to run Ubuntu, you will need to change your network connection. If you do not do this, you will be unable to connect to the turtlebot. First, right click on the virtual machine name then go to settings.



Once you go to settings, navigate to the “Network Adapter” menu as shown below. Once you are in this menu, change your network connection from “NAT” to “Bridged”.



Now that the network is configured correctly, set up a workspace called lab04 (go back to lab01 or lab02 if you do not remember how).

Next, as usual, set your bashrc file to automatically source the .bash file on each new terminal.

```
gedit ~/.bashrc
```

Add this line at the end

```
source ~/lab04/devel/setup.bash
```

Also make sure that the turtlebot3 model is set to burger not waffle then save and close the file.

```
export TURTLEBOT3_MODEL=burger
```

Now we need to connect the PC you are using to the Raspberry Pi on the turtlebot. To make sure you have some networking commands, first install net-tools.

```
sudo apt install net-tools
```

The hostname -I command will display the IP address of the computer you are using.

```
hostname -I
```

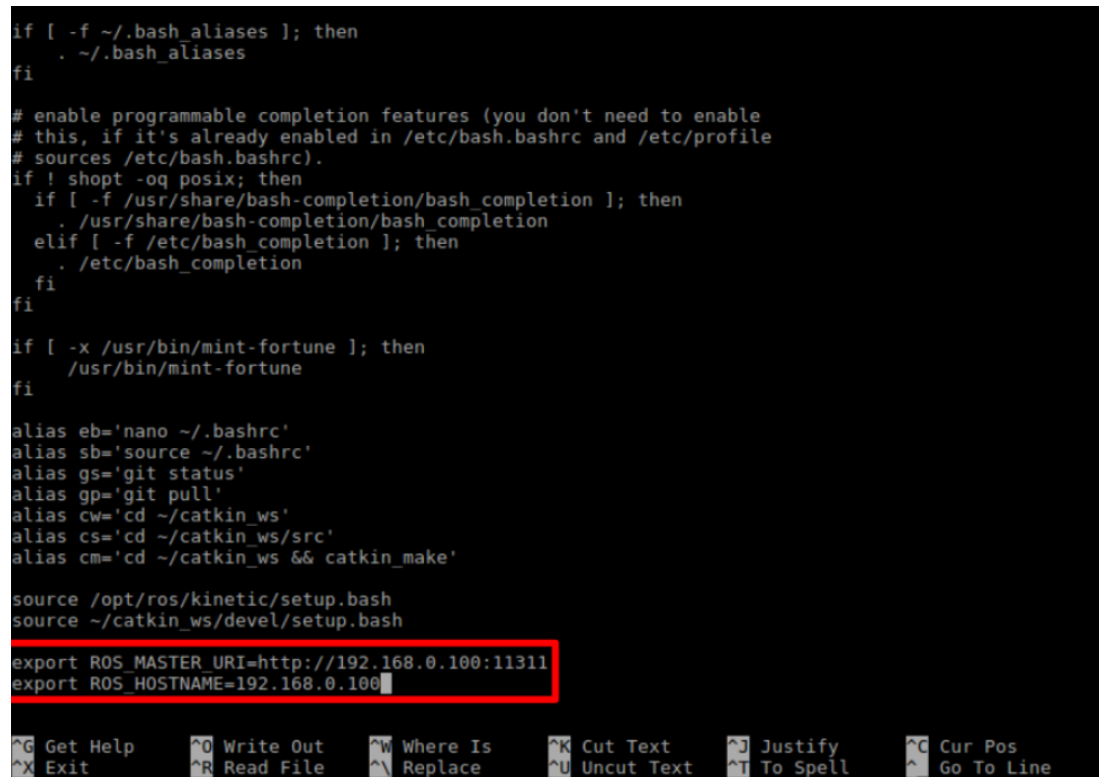
The IP address that this command displays will be used several times in the future, it would be helpful to save it somewhere. Now that we know the address, we need to update the ROS settings in the bashrc file.

```
nano ~/.bashrc
```

Now that the bashrc file is opened, you need to change the “ROS_MASTER_URI” and “ROS_HOSTNAME” to the IP address that you found above. An example of this is shown below. **Note that your IP address will be different; do not copy this ip address.**

Your bashrc file may or may not have the following lines at the end. If they are there, modify them to match the IP address you found above. If they are not there, add them as shown below.

```
export ROS_MASTER_URI=http://YOUR.IP.ADDRESS:11311
export ROS_HOSTNAME=YOUR.IP.ADDRESS
```



```
if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi

if [ -x /usr/bin/mint-fortune ]; then
    /usr/bin/mint-fortune
fi

alias eb='nano ~/.bashrc'
alias sb='source ~/.bashrc'
alias gs='git status'
alias gp='git pull'
alias cw='cd ~/catkin_ws'
alias cs='cd ~/catkin_ws/src'
alias cm='cd ~/catkin_ws && catkin_make'

source /opt/ros/kinetic/setup.bash
source ~/catkin_ws/devel/setup.bash

export ROS_MASTER_URI=http://192.168.0.100:11311
export ROS_HOSTNAME=192.168.0.100

^G Get Help      ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify     ^C Cur Pos
^X Exit          ^R Read File    ^_ Replace      ^U Uncut Text  ^T To Spell    ^_ Go To Line
```

After you change the IP address, save by pressing ctrl+S and exit using ctrl+X. Then source the bashrc file.

```
source ~/.bashrc
```

We now need to follow the same steps for the Raspberry Pi. Boot up the Raspberry Pi then connect to a monitor. Next connect the Pi to power using USB. The Pi will prompt you to login using a username and password. Use login ID `ubuntu` and Password `turtlebot`. Once you have logged in to the Pi, some system information will display on the terminal. One of the lines should be:

```
IP address for wlan0: 192.168.1.26
```

Note that your IP address will be different. Do not use this IP address.

If this line does not show upon start-up run the following command to find the IP address.

```
hostname -I
```

Similar to the previous steps, we need to open the bashrc file and tell the Pi which computer to connect to.

```
nano ~/.bashrc
```

Once you open the file, change the following items:

```
export ROS_MASTER_URI=http://{IP_ADDRESS_OF_REMOTE_PC}:11311
export ROS_HOSTNAME={IP_ADDRESS_OF_RASPBERRY_PI_3}
```

After you change the IP address, save by pressing ctrl+S and exit using ctrl+X. Then source the bashrc file and disconnect the pi from the monitor.

```
source ~/.bashrc
```

Now that your remote PC and Pi are set up, we can connect them. First start a new terminal with roscore then execute this command. Make sure to replace the brackets with the IP address of the Pi. Once you execute this command it may ask you if you are sure you want to continue connecting. Type yes and then press enter. After that, you will be prompted to enter a password. Enter `turtlebot`.

```
ssh ubuntu@{IP_ADDRESS_OF_RASPBERRY_PI}
```

After you execute this command you will see that instead of your username in the command terminal it is `ubuntu@ubuntu`. In this terminal, launch the bringup command for Turtlebot3 applications.

```
roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

After you execute this, you should see lots of information displayed in the terminal. It is showing you the parameters of the turtlebot as well as all of the processes it started to run the turtlebot. Take a moment to look over the text to get familiar with the things that are going on in the background.

Open a new terminal and launch a remote turtlebot3. You should notice that when you open a new terminal tab, the username is back to your normal username. From now on, you will need to keep track of if you are in the ssh terminal, or in your PC terminal by checking the username.

```
roslaunch turtlebot3_bringup turtlebot3_remote.launch
```

In another terminal (NOT ssh terminal, a terminal with your normal username) launch RViz. Once RViz is up and running, show your TA for a signoff.

```
roslaunch rviz rviz -d `rospack find turtlebot3_description`/rviz/model.rviz
```

TA Initials: _____ Date: _____

2. Teleoperation

Now that our robot is up and running it is time to start moving it around. First we will use our keyboard to control the turtlebot. You may recognize this command from the previous lab; that is why ros is so useful, the same nodes are used in simulation as the physical robot.

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Once you launch the node, you should be able to control the robot with the w, a, d, x keys. Move your robot around **carefully**. Once you have moved your robot around a bit, we are going to move the turtlebot around with a bluetooth remote.

Once you have moved around a bit, plot the linear and angular velocity using the rqt_plot command.

```
rqt_plot cmd_vel/linear/x cmd_vel/angular/z
```

A reminder from last week's lab, you will need to change your x and y axis limits as well as your title. An example title is: Chelsey Spitzner Lab 4 Part 2. Save your plot and show your TA for a signoff. After you get a signoff, ctrl-C all active terminals.

TA Initials: _____ Date: _____

3. Automatic Parking

This last section is going to take in data from the LiDAR scanner to create an automatic parking node. The LiDAR sensor detects light intensity as well as distance data. Therefore, if we place reflective tape somewhere, the turtlebot should be able to detect where that tape is. We are going to use this fact to create a script where the LiDAR scanner detects the tape, and then parks itself in front of the tape.

To do this, you first need to place the tape somewhere at the “eye-level” of the robot. Next we need to make sure that you have everything you need for this package installed.

```
sudo apt-get install python-pip
sudo pip install -U numpy
sudo pip install --upgrade pip
```

Next we need to create a package for the automatic parking called “lidar_parking”

```
cd ~/lab04/src
catkin_create_pkg lidar_parking rospy std_msgs sensor_msgs geometry_msgs
nav_msgs
cd ~/lab04
catkin_make
source ./devel/setup.bash
```

Next we need to edit the CMakeLists.txt file.

```
roscd lidar_parking/src
gedit CMakeLists.txt
```

Scroll down until you see:

```
## DEPENDS: system dependencies of this project that dependent projects also need
catkin_package(
#   INCLUDE_DIRS include
#   LIBRARIES lidar_parking
#   CATKIN_DEPENDS geometry_msgs nav_msgs rospy sensor_msgs std_msgs
#   DEPENDS system_lib
)
```

Then add the following code.

```
catkin_python_setup()
```

```
catkin_package(  
  CATKIN_DEPENDS  
    rospy  
    std_msgs  
    sensor_msgs  
    geometry_msgs  
    nav_msgs  
)
```

Next, scroll down to the **#INSTALL#** section of the txt file then add the following:

```
catkin_install_python(PROGRAMS  
  nodes/lidar_parking.py  
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}  
)  
  
install(DIRECTORY launch rviz  
  DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}  
)
```

Save and close the text file. Next we need to create the python setup script that we invoked in the CMakeLists.txt file.

```
roscd lidar_parking  
gedit setup.py
```

Copy and paste the following code then save and close the file.

```
from distutils.core import setup  
from catkin_pkg.python_setup import generate_distutils_setup  
  
# fetch values from package.xml  
setup_args = generate_distutils_setup(  
  packages=['lidar_parking'],  
  package_dir={'': 'src'})  
)  
  
setup(**setup_args)
```

Now we need to create a launch file for our code to work.

```
mkdir launch  
roscd lidar_parking/launch  
gedit lidar_parking.launch
```

Copy and paste the following code into the launch file then save and close the file.


```
<launch>
  <node pkg="lidar_parking" type="lidar_parking.py" name="lidar_parking"
output="screen">
  </node>
</launch>
```

To make things easier and faster for you, we are going to create a setup folder for rviz that contains a script to set up the software.

```
roscd lidar_parking
mkdir rviz
```

Save the code given to you called “lidar_parking.rviz” into the rviz folder you just created.

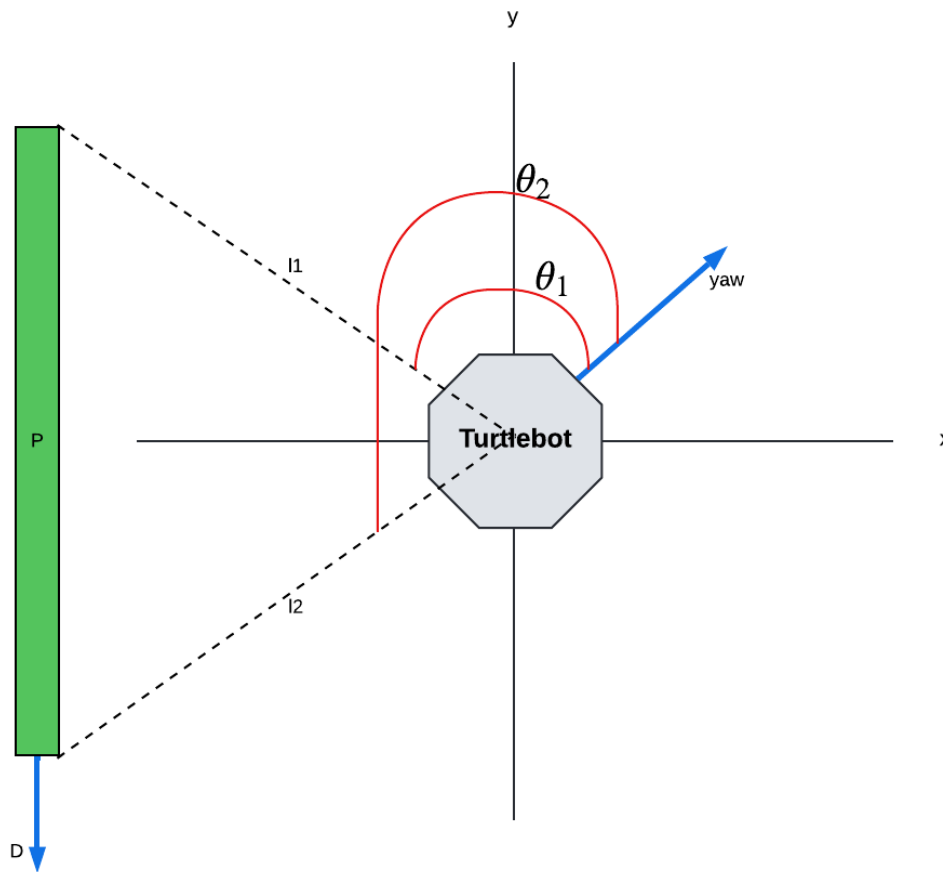
With all of this done, we can finally get to making the automatic parking algorithm. First, create a folder called “nodes” for the script to live in.

```
roscd lidar_parking
mkdir nodes
```

Save the code given to you called “lidar_parking.py” into the nodes folder you just created.

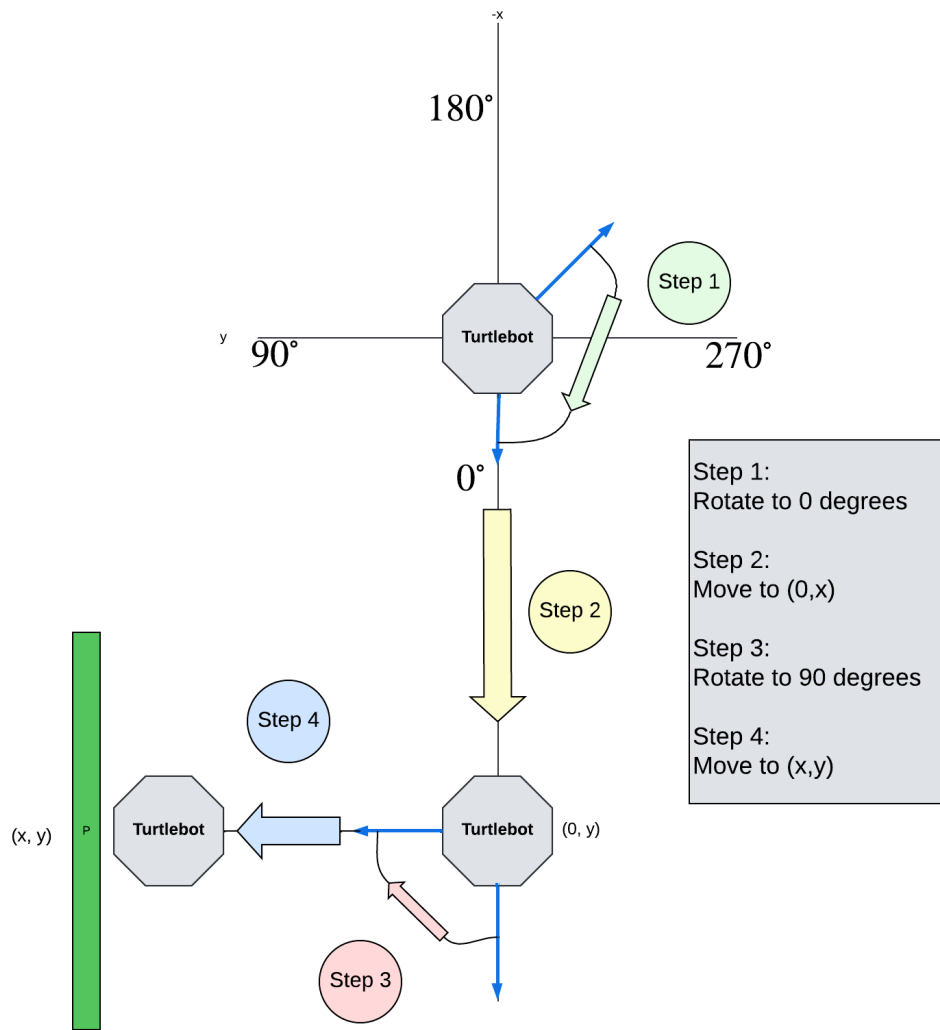
To give you a head-start, most of the code is given to you. The only part that you have to do is fill in the twist velocity commands near the end of this code. Scroll down until you find the comment saying you need to put your code here. Under this comment are several twist commands that are equal to #. Such as: `twist.linear.x = #` or `twist.angular.z = #`. You will need to put decimals instead of “#” in the code.

To help you understand this part of the code, this picture should explain how the code is interpreting the angles of the turtlebot and the position of the reflective tape.



- θ_1 = angle of start point
- θ_2 = angle of end point
- l_1 = distance of start point
- l_2 = distance of end point
- yaw = heading of Turtlebot
- D = direction of Turtlebot
- P = goal point (x, y)
- \emptyset = angle between yaw and D

Next you will see that the code is divided into four “steps”. There are four `elif step == #:` sections where the number is either 1, 2, 3, or 4. These steps are the simple algorithm used for the automatic parking. Each of these steps are shown below. It is your job to interpret these steps into twist commands and separate them into either linear x velocity or angular z velocity commands. There are comments above each velocity pair that you need to input that tell you what magnitude numbers to use.



After you complete the python script, save and close the file then set permissions using the `chmod +x` command. Now we can compile our workspace and run the package.

```
cd ~/lab04
catkin_make
source ./devel/setup.bash
```

In a new terminal, run `roscore`.

Then you need to run this command on the turtlebot's command terminal (the terminal you ssh'd into).

```
roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

Next the remote PC needs to launch the turtlebot packages.

```
roslaunch turtlebot3_bringup turtlebot3_remote.launch
```

In a new terminal, run rviz with the following command. Note that this command is locating the rviz file that we created earlier and not just launching rviz.

```
roslaunch rviz rviz -d `rospack find lidar_parking`/rviz/lidar_parking.rviz
```

After rviz is launched, we can launch the automatic parking script.

```
roslaunch lidar_parking lidar_parking.launch
```

If everything is working correctly, the turtlebot will detect where the reflective tape is, and then it will move towards the tape and stop in front of it. Try putting the tape in several places and test the limits of what the LiDAR sensor can detect. For one iteration of parking, run rqt_plot for the linear and angular velocities/

```
rqt_plot cmd_vel/linear/x cmd_vel/angular/z
```

Don't forget to change your x and y limits as well as the title. An example title is: Chelsey Spitzner Lab 4 Part 3. Save your plot and show your TA for a signoff. After you get a signoff, ctrl-C all active terminals.

TA Initials: _____ Date: _____

Items to submit:

Part 2: rqt_plot image

Part 3: rqt_plot image and automatic_parking.py script

A.5 Lab 5: Fixed Motion

1. Simulation

First, create a catkin workspace called lab05. If you do not remember how, go back to lab01 or lab02. Then export the following line to the bashrc file.

```
echo "source ~/lab05/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Next we need to install some niryo packages. **(not needed if using lab PCs)**

```
git clone https://github.com/NiryoRobotics/ned_ros src
```

```
rosdep update
```

```
rosdep install --from-paths src --ignore-src --default-yes --rosdistro
melodic --skip-keys "python-rpi.gpio"
```

```
sudo apt-get install python-pip ros-melodic-robot-state-publisher ros-
melodic-moveit ros-melodic-rosbridge-suite ros-melodic-joy ros-melodic-ros-
control ros-melodic-ros-controllers ros-melodic-tf2-web-republisher
```

To finish the installation process, run `catkin_make` and source the bashrc file.

```
catkin_make
source ~/.bashrc
```

All of these installations will work to simulate the robot arm. However, some packages beyond just simulation require a python library to work. These commands will install some python3 libraries. **(not needed if using lab PCs)**

```
pip3 install numpy
pip3 install pyniryo
pip3 install pygame
pip3 install opencv-python
```

To simulate the robot, we are going to use Rviz. This command launches rviz and trackbars for each of the 6 joints of the niryo arm. Move each of the joints to get a feel for how the robot operates.

```
roslaunch niryo_robot_description display.launch
```

Now we need to launch something that will allow us to control the robot easily. Since we installed the Ned repository, we will use their code instead of developing our own.

Ctrl+C the current Rviz terminal. Install the following repository and ubuntu packages.
(not needed if using lab PCs)

```
git clone https://github.com/NiryoRobotics/ned_applications.git
```

```
pip3 install -r src/requirements_ned2.txt
```

```
sudo apt install build-essential
```

```
sudo apt install sqlite3
```

```
sudo apt install ffmpeg
```

Then run `catkin_make` and source the `.bashrc` file.

```
rosdep install -y --from-paths . --ignore-src --rosdistro noetic  
catkin_make  
source ~/.bashrc
```

Now we can launch Rviz with the robot arm.

```
roslaunch niryo_robot_bringup desktop_rviz_simulation.launch
```

In a new terminal, open the code “`robot_ned.py`”

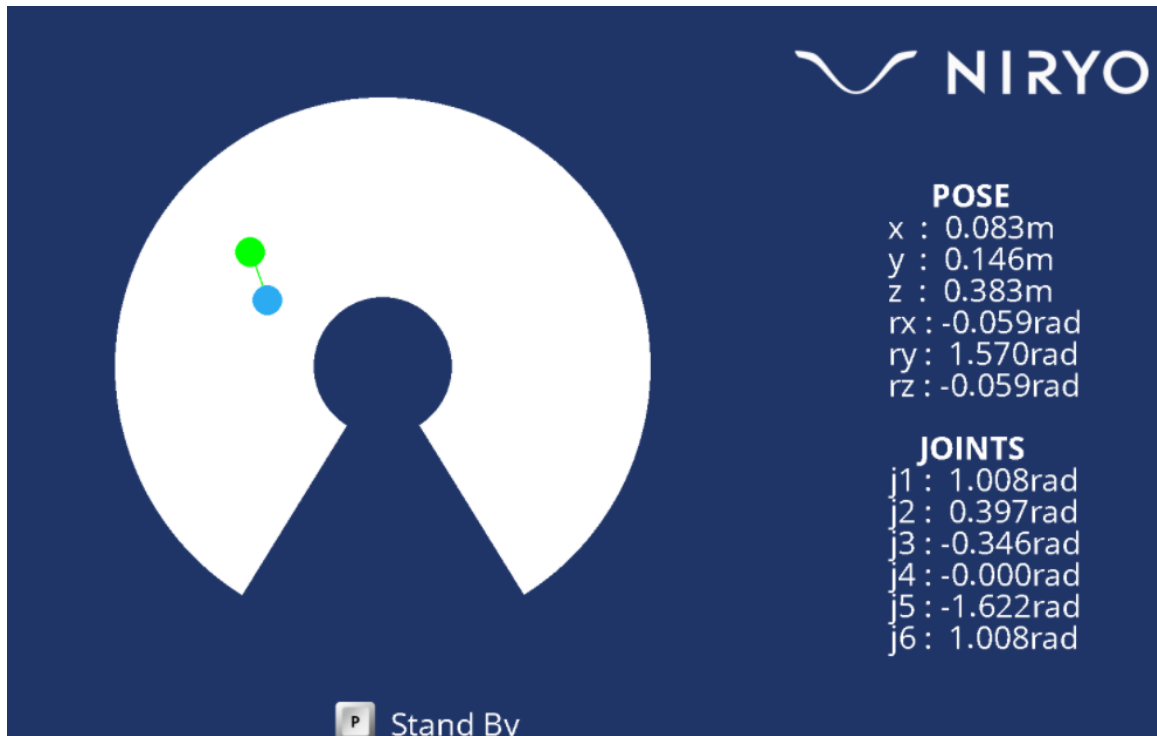
```
gedit robot_ned.py
```

Once the file is open, you need to make sure the IP address is correct. Scroll down until you see the line “`robot_ip`”. Make sure the IP is as shown below then save and close the file.

```
robot_ip = "127.0.0.1"
```

Now that the IP address is correct, we can launch the python program.

```
cd ~/lab05/ned_applications/examples/Control_Ned_Mouse  
python3 robot_ned.py
```



After you launch this, you should see a mouse control API pop up. It contains a white area with a blue dot and a green dot. The white area is the envelope of the robot arm's end effector. The green dot is the mouse position and the blue dot is the end effector's position. The API also displays the pose of the robot along with all of the joint angles.

This will allow you to control the movement of the robot with your PC mouse. Move your mouse around and watch the simulation in Rviz move as well. There are some other controls you can use as well with this program:

- Use the scroll wheel to change the end effector's height
- Use the right click to open/close the end effector
- X key to switch from rad to deg

Run the `rqt_graph` and show your TA.

```
rosrun rqt_graph rqt_graph
```

Finally, view the `tf_tree` and show your TA

```
rosrun tf view_frames
evince frames.pdf
```

TA Initials: _____ Date: _____

Ctrl-C all active terminals.

2. Physical Movement

The physical arm you are using in the lab should tell you the IP address of the arm that you need to be using. It should already be physically connected to your PC via an ethernet cable. To start, we are going to run the same program we did before. Make sure roscore is still running in a terminal. In a new terminal, open the code “robot_ned.py”

```
gedit robot_ned.py
```

Once the file is open, you need to make sure the IP address is correct. Scroll down until you see the line “robot_ip”. Change the IP address to the address that is given on the robot then save and close the file.

```
robot_ip = "IPADDRESS"
```

Now that the IP address is correct, we can connect to the physical robot. The IP address of the robot should be written on the robot, use this in the command line below.

```
ssh niryo@<ned_static_ip_address>
```

Next (in the ssh terminal), launch the hardware stack.

```
roslaunch niryo_robot_hardware_interface  
hardware_interface_standalone.launch
```

Launch the python program in a new terminal.

```
python3 robot_ned.py
```

Now you should be able to move the mouse of the PC around to control the arm. Be careful not to move the mouse too fast to prevent damage to the robot.

Reminder:

- Use the scroll wheel to change the end effector’s height
- Use the right click to open/close the end effector
- X key to switch from rad to deg

Show your TA that your robot functions through the mouse movement.

TA Initials: _____ Date: _____

Ctrl-C all terminals when you are done experimenting.

3. Using Moveit

The following command lines will install moveit on your PC. **(not needed if using lab PCs)**

```
sudo apt install python3-wstool
```

```
cd ~/lab06/src
wstool init .
wstool merge -t . https://raw.githubusercontent.com/ros-
planning/moveit/master/moveit.rosinstall
wstool remove moveit_tutorials
wstool update -t .
git clone https://github.com/ros-planning/moveit_tutorials.git -b master
rosdep install -y --from-paths . --ignore-src --rosdistro noetic
catkin_make
source ~/.bashrc
```

Next we need to ssh into the robot again.

```
ssh niryo@<ned_static_ip_address>
```

Next (in the ssh terminal), launch the hardware stack.

```
roslaunch niryo_robot_hardware_interface
hardware_interface_standalone.launch
```

Now we can launch moveit for our robot control.

```
roslaunch niryo_robot_bringup moveit_multimachines.launch
```

In a new terminal, launch rviz so that we can utilize the moveit package.

```
roslaunch rviz rviz
```

Once RViz is open, click on “MotionPlanning” to add it to our visualization. This should add a bunch of arrows around the end-effector in the visualization. Left-click to rotate, middle-click to move x/y, and right-click to zoom. Change the position of the robot using these controls. The new position of the robot should be shown in orange. Once you have decided upon a good pose for the robot, click “Plan & Execute”. The Moveit package automatically calculates the path for the robot from its current position to the planned pose.

4. Using Python to Move

Using RViz to plan a movement is nice, but it is very non-specific as it is visual instead of inputting specific coordinates. Instead of RViz we can use a python script to tell the robot where to go. This is going to be a rudimentary script as we are going to tell the robot the angle of each joint. In the next lab, we will explore more elegant ways to do this.

To start, let's create a python script called **pick_and_place.py**.

```
gedit pick_and_place.py
```

Once the file is open, copy and paste the code below into the text editor. In the third line, you will need to change **IPADDRESS** to the address that is given on the robot then save and close the file. Make sure to leave the quotes around the IP address. Examine this code to get a feel for how it works. The `robot.move_pose` command tells the robot where to put each joint (in radians). Once you are done, save and close the file.

```
from pyniryo import *

robot = NiryoRobot("IPADDRESS")

robot.calibrate_auto()
robot.update_tool()

robot.release_with_tool()
robot.move_pose(0.2, -0.1, 0.25, 0.0, 1.57, 0.0)
robot.grasp_with_tool()

robot.move_pose(0.2, 0.1, 0.25, 0.0, 1.57, 0.0)
robot.release_with_tool()

robot.close_connection()
```

Next we need to ssh into the robot again.

```
ssh niryo@<ned_static_ip_address>
```

In the ssh terminal, launch the hardware stack.

```
roslaunch niryo_robot_hardware_interface
hardware_interface_standalone.launch
```

Now we can launch our python script.

```
python3 pick_and_place.py
```

Once it is working, show your TA.

TA Initials: _____ Date: _____

Ctrl-C all terminals when you are done experimenting.

A.6 Lab 6: Pose Estimation

1. Create Path Planning Package

First, create a catkin workspace called lab06. Then export the following line to the bashrc file.

```
echo "source ~/lab06/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Create a package called path_planning.

```
cd ~/lab06/src
catkin_create_pkg path_planning rospy
cd ~/lab06
catkin_make
source ./devel/setup.bash
```

Modify the CMakeLists.txt file.

```
roscd path_planning
gedit CMakeLists.txt
```

Copy and paste the following into the CMakeLists.txt file then save and close.

```
install(PROGRAMS
  scripts/path_planning.py
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)

install(DIRECTORY launch DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION})
```

Create a launch folder and add a launch file.

```
mkdir launch
roscd path_planning/launch
gedit path_planning.launch
```

Copy and paste the following into the CMakeLists.txt file then save and close.

```
<launch>

  <node name="path_planning_node" pkg="path_planning"
  type="path_planning.py" respawn="false" output="screen">
  </node>

</launch>
```

Create a scripts folder for our python script to live in.

```
roscd path_planning
mkdir scripts
roscd path_planning/scripts
```

Download the file **path_planning.py** and save it in the scripts folder you just made. Make the code executable using the `chmod +x` command then compile the workspace.

```
chmod +x path_planning.py
cd ~/lab06
catkin_make
```

`Path_planning.py` gives you the code to set the pose (x, y, z) position for the end-effector of the robot. It plans a cartesian path, then executes it to move the robot to the goal pose.

2. Simulation

Now it is time to simulate our `path_planning` package. Start `roscore` then launch `RViz` for the Niryo arm.

```
roslaunch niryo_robot_bringup desktop_rviz_simulation.launch
```

Next launch the python script you just created.

```
roslaunch Niryo path_planning.py
```

Run the `rqt_graph` and show your TA.

```
roslaunch rqt_graph rqt_graph
```

TA Initials: _____ Date: _____

Ctrl-C all active terminals.

3. Plot a Square

Open the python script and take a good look to understand what it is doing.

```
roscd path_planning/scripts
gedit path_planning.py
```

Change the python code so that it plots a square. You already have the code setup to move the end-effector to one position. Now, change it to go to four different locations to form a square. This may require some trial and error. Test it out on the simulated Niryo Ned arm using the steps in part 2. Once you get it working, get a TA signature.

TA Initials: _____ Date: _____

4. Physical Robot

Make sure roscore is running. In a new terminal, ssh into Ned. The IP address of the robot should be written on the robot, use this in the command line below.

```
ssh niryo@<ned_static_ip_address>
```

Next (in the ssh terminal), launch the hardware stack.

```
roslaunch niryo_robot_hardware_interface  
hardware_interface_standalone.launch
```

Before we can move the robot, we need to calibrate it.

```
rosservice call /niryo_robot/joints_interface/calibrate_motors "value: 1"  
rosservice call /niryo_robot/learning_mode/activate "value: false"
```

In a new terminal, launch rviz so that we can utilize the moveit package.

```
roslaunch rviz rviz
```

Open a new terminal (NOT the ssh terminal), so that we can launch the moveit files for Ned. Niryo Robotics (the company who created Ned) has already created the moveit configuration for us. This saves us a lot of time as it can take a while to create a moveit configuration for a robot.

```
roslaunch niryo_robot_bringup moveit_multimachines.launch
```

Finally, launch the path planning python code.

```
roslaunch Niryo path_planning.py
```

If all went correctly, the robot arm end-effector should move in a square similar to your simulation.

Run the rqt_graph and show your TA the robot moving in a square.

```
roslaunch rqt_graph rqt_graph
```

TA Initials: _____ Date: _____

Items to submit:

Part 3: Python code

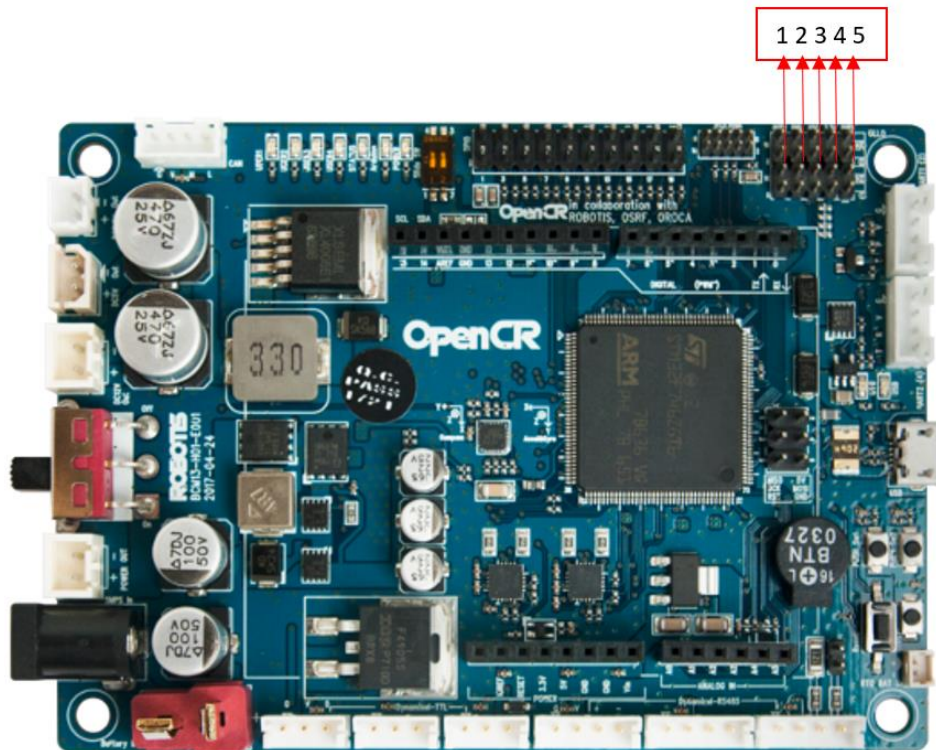
Part 4: rqt_graph

A.7 Lab 7: Sensing

1. Setup IR Sensor

In lab 3 we created a simple obstacle avoidance algorithm for a simulated turtlebot. The algorithm allowed the robot to autonomously navigate an unknown map and avoid objects. However, what would happen if there was a set of stairs in the area? The LiDAR scanner would not be able to detect this and the robot would fall. These types of obstacles are called negative obstacles. Some other scenarios could be potholes, curbs, or other sharp drop-offs. To detect negative obstacles we are going to use IR distance sensors. This process is commonly called cliff detection. The cliff detection we are implementing is relatively simple as it only uses one IR sensor. Cliff detection is commonly implemented with four to six sensors

First we need to set up our IR sensor. Plug in the IR sensor to the openCR board in the orientation shown below.



2. Cliff Detection Package

First, set up a workspace called lab07 (go back to lab01 or lab02 if you do not remember how). Next, as usual, set your bashrc file to automatically source the .bash file on each new terminal.

```
gedit ~/.bashrc
```

Add this line at the end

```
source ~/lab07/devel/setup.bash
```

Also make sure that the turtlebot3 model is set to waffle not burger then save and close the file.

```
export TURTLEBOT3_MODEL=waffle_pi
```

Next we are going to create a package called cliff_detection.

```
cd ~/lab07/src
catkin_create_pkg cliff_detection rospy std_msgs sensor_msgs geometry_msgs
turtlebot3_msgs nav_msgs visualization_msgs message_generation
cd ~/lab07
catkin_make
source ~/.bashrc
```

Open the CMakeLists.txt file in the lab07 folder.

```
roscd cliff_detection
gedit CMakeLists.txt
```

Add this code to the file then save and close.

```
catkin_python_setup()

catkin_install_python(PROGRAMS
  nodes/turtlebot3_cliff
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)

install(DIRECTORY launch rviz
  DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
)
```

Create a python setup script.

```
gedit setup.py
```

Copy and paste the following code into the python script then save and close.

```
from distutils.core import setup
from catkin_pkg.python_setup import generate_distutils_setup

# fetch values from package.xml
```



```

setup_args = generate_distutils_setup(
    packages=['turtlebot3_example'],
    package_dir={'': 'src'})
)

setup(**setup_args)

```

Now we need to create a launch file for our code to work.

```

mkdir launch
roscd cliff_detection/launch
gedit turtlebot3_cliff.launch

```

Copy and paste the following code into the launch file then save and close.

```

<launch>
  <node pkg="cliff_detection" type="turtlebot3_cliff"
name="turtlebot3_cliff" output="screen">
  </node>
</launch>

```

With all of the background work done, we can get to making the cliff detection node.

```
mkdir nodes
```

Download the file “turtlebot3_cliff” into the nodes folder you just created. Most of the code is given to you but there are a few lines of code that you need to complete. The code you need to complete is the publisher and subscriber. Publishing and subscribing is a main part of ROS and it is important to know how it works.

Let's first work on creating the publisher. The publisher has the syntax of `rospy.Publisher('topic', message type, queue size=#)`. The cliff detection algorithm should stop when the sensor detects a cliff. Therefore, we need to publish the velocity commands for the robot. The following list describes the topic and type of the velocity commands.

- 'topic' = 'cmd_vel'
- Message type = Twist
- queue_size=1

Once you have set up the publisher, the subscriber needs to be set up. The publisher has the syntax of `rospy.Subscriber('topic', message type, invoke function, queue size=#)`. We need to subscribe to the data published by the IR sensor.

- 'topic' = 'sensor_state'
- Message type = SensorState
- Function = self.get_cliff
- queue_size=1

Once you have the publisher and subscriber setup, save and close the file. Set permissions for the file using the `chmod +x` command.

```
chmod +x turtlebot3_cliff
```

Compile the workspace using `catkin_make` then source the `bashrc` file.

```
cd ~/lab07
catkin_make
source ~/.bashrc
```

3. Bringup Turtlebot

The same process from lab04 will be followed to connect your computer to the turtlebot. It has been a while since that lab, so we will walk you through it again.

The `hostname -I` command will display the IP address of the computer you are using.

```
hostname -I
```

The IP address that this command displays will be used several times in the future, it would be helpful to save it somewhere. Now that we know the address, we need to update the ROS settings in the `bashrc` file.

```
nano ~/.bashrc
```

Now that the `bashrc` file is opened, you need to change the “`ROS_MASTER_URI`” and “`ROS_HOSTNAME`” to the IP address that you found above. An example of this is shown below. **Note that your IP address will be different; do not copy this ip address.**

Your `bashrc` file may or may not have the following lines at the end. If they are there, modify them to match the IP address you found above. If they are not there, add them as shown below.

```
export ROS_MASTER_URI=http://YOUR.IP.ADDRESS:11311
export ROS_HOSTNAME=YOUR.IP.ADDRESS
```

```
if [ -f ~/.bash_aliases ]; then
  . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

if [ -x /usr/bin/mint-fortune ]; then
  /usr/bin/mint-fortune
fi

alias eb='nano ~/.bashrc'
alias sb='source ~/.bashrc'
alias gs='git status'
alias gp='git pull'
alias cw='cd ~/catkin_ws'
alias cs='cd ~/catkin_ws/src'
alias cm='cd ~/catkin_ws && catkin_make'

source /opt/ros/kinetic/setup.bash
source ~/catkin_ws/devel/setup.bash

export ROS_MASTER_URI=http://192.168.0.100:11311
export ROS_HOSTNAME=192.168.0.100
```

After you change the IP address, save by pressing ctrl+S and exit using ctrl+X. Then source the bashrc file.

```
source ~/.bashrc
```

We now need to follow the same steps for the Raspberry Pi. Boot up the Raspberry Pi then connect to a monitor. Next connect the Pi to power using USB. The Pi will prompt you to login using a username and password. Use login ID `ubuntu` and Password `turtlebot`. Once you have logged in to the Pi, some system information will display on the terminal. One of the lines should be:

```
IP address for wlan0: 192.168.1.26
```

Note that your IP address will be different. Do not use this IP address.

If this line does not show upon start-up run the following command to find the IP address.

```
hostname -I
```

Similar to the previous steps, we need to open the bashrc file and tell the Pi which computer to connect to.

```
nano ~/.bashrc
```

Once you open the file, change the following items:

```
export ROS_MASTER_URI=http://{IP_ADDRESS_OF_REMOTE_PC}:11311
export ROS_HOSTNAME={IP_ADDRESS_OF_RASPBERRY_PI_3}
```

After you change the IP address, save by pressing ctrl+S and exit using ctrl+X. Then source the bashrc file and disconnect the pi from the monitor.

```
source ~/.bashrc
```

Now that your remote PC and Pi are set up, we can connect them. First start roscore then execute the ssh command in a new terminal. Make sure to replace the brackets with the IP address of the Pi. Once you execute this command it may ask you if you are sure you want to continue connecting. Type yes and then press enter. After that, you will be prompted to enter a password. Enter turtlebot.

```
ssh ubuntu@{IP_ADDRESS_OF_RASPBERRY_PI}
```

After you execute this command you will see that instead of your username in the command terminal it is `ubuntu@ubuntu`. In this terminal, launch the bringup command for Turtlebot3 applications.

```
roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

Open a new terminal and launch a remote turtlebot3. You should notice that when you open a new terminal tab, the username is back to your normal username. From now on, you will need to keep track of if you are in the ssh terminal, or in your PC terminal by checking the username.

```
roslaunch turtlebot3_bringup turtlebot3_remote.launch
```

In a new PC terminal, launch the teleoperation node.

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

In another PC terminal, launch the cliff detection node.

```
roslaunch cliff_detection turtlebot3_cliff.launch
```

In the lab, there should be an elevated platform for you to place the turtlebot on. Using the teleoperation node, move the turtlebot around and test to see if it will stop itself when attempting to go off the edge of the platform. Note that since we only have preliminary cliff detection, it can only detect cliffs in front of the turtlebot and not to the sides or rear. Be cautious to not drive the turtlebot off of the platform in these directions.

Test the cliff detection a couple times to see how it works. To help, view the linear and angular velocity. Don't forget to change the x/y scales as well as changing the plot title.

```
rqt_plot cmd_vel/linear/x cmd_vel/angular/z
```

Show your TA the plot then ctrl+C on the terminals running the teleoperation and the cliff detection node.

TA Initials: _____ Date: _____

4. Cliff Detection Improvement

You may notice when you're experimenting with the cliff detection that there are some flaws with our algorithm. If there is some angular velocity when approaching the cliff, the code does not account for this. It also changes the linear velocity to 0.05 if there is no cliff detected instead of leaving the linear velocity to the previous value. It is your job to correct these errors in the code. To give you a head start, the variable `last_twist` is defined and can be used to get the last twist command that the robot received. Use these two commands to open the cliff detection node.

```
roscd cliff_detection/nodes
gedit turtlebot3_cliff
```

As you go through the code, you should notice that the `last_twist` variable is not used yet. The `last_twist` variable has two parts of data that we need; the linear velocity and the angular velocity. To index the linear velocity: `last_twist.linear.x`
To index the angular velocity: `last_twist.angular.z`

Your goal is to change the code of the if/else statements so that if a cliff is detected, both the linear and angular velocity are set to zero. But if there is no cliff detected. The linear and angular velocity should stay the same.

When you are done with the code, save and close the file. Compile the workspace using `catkin_make` then source the `bashrc` file.

```
cd ~/lab07
catkin_make
source ~/.bashrc
```

Your ssh and roscore terminals should still be running. Open a new terminal and re-run the teleoperation node, then start another terminal and launch the improved cliff detection node.

Test the cliff detection a couple times to see if the performance improves. View the linear and angular velocity. Don't forget to change the x/y scales as well as changing the plot title.

```
rqt_plot cmd_vel/linear/x cmd_vel/angular/z
```

Show your TA the plot then ctrl+C on all terminals and turn off the turtlebot

TA Initials: _____ Date: _____

Items to submit:

Part 3: rqt_plot image

Part 4: rqt_plot image and turtlebot3_cliff script

A.8 Lab 8: Mapping

1. Simulation

Mapping is an important part of mobile robots. ROS offers many tools and packages to help with mapping. When testing things for the first time, it is often a good idea to run a simulation first. As usual, set up a workspace called lab08 and edit the bashrc file.

```
echo source ~/lab08/devel/setup.bash >> ~/.bashrc
source ~/.bashrc
```

Launch roscore then open a new terminal to launch the simulation.

```
roslaunch turtlebot3_gazebo turtlebot3_house.launch
```

In a new terminal, launch the teleoperation node.

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

ROS has a useful tool called rosbag that allows recording of data. We are going to use this to record the lidar scanner data and the coordinate transform data. Open a new terminal and launch rosbag.

```
rosbag record -O lab08_1.bag /scan /tf /odom
```

Drive the turtlebot around in the simulation while the data is recording. Make sure to go into all of the rooms several times so that you end up with a good map of the house. Once you are satisfied that you have covered enough ground, press ctrl-C in all terminals except for roscore. To make sure that it recorded correctly, execute this line in the terminal.

```
rosbag info lab08_1.bag
```

After you execute this command, you should see some information pop up about the file. We can now build our map from the recorded data.

```
roscppparam set use_sim_time true
```

```
roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
```

After you run gmapping, it is going to wait for data to create the map. To give it the data we recorded using rosbag, we will use rosbag play. This command replays the data that is recorded.

```
rosbag play --clock lab08_1.bag
```

This will run for a minute as the map is built. When it finishes, we need to run a map server to save the map. Make sure you leave the gmapping terminal running for this step. This step may also take a few minutes to run.

```
roslaunch map_server map_saver -f map1
```

Once this is done, ctrl-C all terminals except for roscore. Open the map image created by the previous command and see how well the map was created. As you can see, the map is not perfect. No mapping will be perfect, but we can tune some parameters to try and get a more accurate map.

```
rosparam set /slam_gmapping/angularUpdate 0.1
rosparam set /slam_gmapping/linearUpdate 0.1
rosparam set /slam_gmapping/lskip 10
rosparam set /slam_gmapping/xmax 10
rosparam set /slam_gmapping/xmin -10
rosparam set /slam_gmapping/ymax 10
rosparam set /slam_gmapping/ymin -10
```

After these parameters are set, re-run through the process of running the simulation, driving the turtlebot around, then creating the map server. **Rename the bag file as lab08_02 and change the map_saver name from map1 to map2** to keep track of your map reruns. Once you do this, compare your new map with the previous one. See if it is any better and if it isn't try changing some other parameters and run through the mapping process again.

As you can see, this process can be quite tedious and time consuming as the quality of the map construction depends highly on the parameters and on how the turtlebot is moved around in the environment.

To make life a little easier for you, we can use a package from ros called explore-lite. Ctrl-C the teleoperation node, but leave the gazebo simulation and roscore running. Use sudo apt-get to install the package.

```
sudo apt-get install ros-kinetic-explore-lite
```

Once the package is installed, we need to launch something called move_base. This launch file is part of something called the navigation stack. The navigation stack combines information from a robot's sensors, position goal, and odometry then outputs velocity commands. It is used in many different types of robots on ros and can take some work to set up. Luckily the navigation stack is already made for the turtlebot and you should already have it installed back from lab03.

```
roslaunch turtlebot3_navigation move_base.launch
```

In a new terminal, launch a turtlebot slam node. Next week's lab will get into how SLAM works in ros, but for now, we are just going to use it within the explore-lite package.

```
roslaunch turtlebot3_slam turtlebot3_slam.launch
```


Now, in a new terminal, we can launch the explore-lite package. This will autonomously navigate the world the turtlebot is in and then it will create a map at the end. This takes a lot of work off of your shoulders as you can launch the node and then let the package do all of the work. However, there are some limitations; this package does not do well in very small rooms.

```
roslaunch explore_lite explore.launch
```

While the explore node is running, launch the rqt_graph to see what nodes are running.

```
rqt_graph
```

Save a picture of the graph.

Once the explore-lite package has produced a map, save it and show your TA. Ctrl-C all terminals besides roscore and close them.

TA Initials: _____ Date: _____

2. Physical World

Now that you have simulated mapping in ros, we are going to follow the same procedure with the real turtlebot in the lab. On the PC in the lab, it should give you the IP address of the turtlebot you are using, use that to ssh into the turtlebot. If it prompts you for a password, enter turtlebot.

```
ssh ubuntu@{IP_ADDRESS_OF_RASPBERRY_PI}
```

In the same terminal that you ssh'd into the turtlebot, launch the turtlebot3.

```
roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

Now open a new terminal and remote launch the turtlebot3 from the PC.

```
roslaunch turtlebot3_bringup turtlebot3_remote.launch
```

From this point, the process is going to be very similar to the simulation, but with the physical robot. In a new terminal, launch the teleoperation node.

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Open a new terminal and launch rosbag.

```
rosbag record -O lab08_2_1.bag /scan /tf
```

Drive the turtlebot around while the data is recording. Once you are satisfied that you have covered enough ground, press ctrl-C the teleoperation and rosbag terminals. To make sure that it recorded correctly, execute this line in the terminal.

```
rosviz info lab08_2_1.bag
```

After you execute this command, you should see some information pop up about the file. We can now build our map from the recorded data.

```
rosviz set use_sim_time true
```

```
rosviz gmapping slam_gmapping
```

After you run gmapping, it is going to wait for data to create the map. To give it the data we recorded using rosbag, we will use rosbag play. This command replays the data that is recorded.

```
rosviz play --clock lab08_2_1.bag
```

This will run for a minute as the map is built. When it finishes, we need to run a map server to save the map. Make sure you leave the gmapping terminal running for this step.

```
rosviz map_server map_saver
```

Once this is done, ctrl-C the gmapping terminal and the rosbag play terminal. Open the map image created by the previous command and see how well the map was created. As with the simulated turtlebot, you may have to change some of the tuning parameters for the mapping. Change some of these parameters to see how it affects the mapping.

```
rosviz set /slam_gmapping/angularUpdate 0.1  
rosviz set /slam_gmapping/linearUpdate 0.1  
rosviz set /slam_gmapping/lskip 10  
rosviz set /slam_gmapping/xmax 10  
rosviz set /slam_gmapping/xmin -10  
rosviz set /slam_gmapping/ymax 10  
rosviz set /slam_gmapping/ymin -10
```

Re-run through the process of running the simulation, driving the turtlebot around, then creating the map server. Rename the bag file as lab08_2_2 to keep track of your map reruns. Once you do this, compare your new map with the previous one. See if it is any better and if it isn't try changing some other parameters and run through the mapping process again.

As before, we are going to run the explore-lite package to see how it compares to the manual exploration of the environment. Ctrl-C the gmapping terminal and the rosbag play terminal.

Next in a new terminal, launch the move_base file.

```
roslaunch turtlebot3_navigation move_base.launch
```

In a new terminal, launch a turtlebot slam node.

```
roslaunch turtlebot3_slam turtlebot3_slam.launch
```

Now, in a new terminal, we can launch the explore-lite package.

```
roslaunch explore_lite explore.launch
```

While the explore node is running, launch the rqt_graph to see what nodes are running.

```
rqt_graph
```

Save a picture of the graph.

Once the explore-lite package has produced a map, save it and show your TA. Ctrl-C all terminals besides roscore and close them.

TA Initials: _____ Date: _____

Items to submit:

Part 2: map image and rqt_graph image

Part 3: map image and rqt_graph image

A.9 Lab 9: SLAM

1. Simulation SLAM

As before, we are going to first simulate some SLAM methods in ROS before implementing them on the turtlebot. There are many different methods available in ROS that automatically implement SLAM algorithms depending on which package you choose. Each algorithm has strengths and weaknesses depending on the environment and the robot specifications. We are going to test a couple of these algorithms and decide which is best for us to use.

First let's set up the workspace. Create a workspace called lab09 and then set up the bashrc file.

```
echo source ~/lab09/devel/setup.bash >> ~/.bashrc
source ~/.bashrc
```

Then launch roscore and open a new terminal tab (`Ctrl + Alt + T`).

The algorithms we will be using are:

- Gmapping
- Hector
- Karto

Gmapping

Starting with gmapping, we first need to launch the simulation in gazebo.

```
roslaunch turtlebot3_gazebo turtlebot3_house.launch
```

Next launch the SLAM node.

```
roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
```

Once the SLAM node is launched, the teleoperation node is used to drive the turtlebot around until a map is created.

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Drive around using the teleoperation node until the map in the SLAM RViz is sufficient. Once the map is created, open a new terminal and run the following command to save the map. Once the map is saved, open the map and leave it open.

```
roslaunch map_server map_saver -f map1
```

Ctrl-C the SLAM node and the teleoperation node but leave gazebo running. Next, the hector SLAM algorithm will be used.

Hector

First install the hector slam algorithm.

```
sudo apt-get install ros-melodic-hector-slam
```

Launch the SLAM node:

```
roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=hector
```

Once the SLAM node is launched, the teleoperation node is used to drive the turtlebot around until a map is created.

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Once the map is created, open a new terminal and run the following command to save the map. Once the map is saved, open the map and leave it open.

```
roslaunch map_server map_saver -f map2
```

Ctrl-C the SLAM node and the teleoperation node but leave gazebo running. Next, the Core SLAM algorithm will be used.

Karto

Install:

```
sudo apt-get install ros-melodic-slam-karto
```

Launch:

```
roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=karto
```

Once the SLAM node is launched, the teleoperation node is used to drive the turtlebot around until a map is created.

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Once the map is created, open a new terminal and run the following command to save the map. Once the map is saved, open the map and leave it open.

```
roslaunch map_server map_saver -f map3
```

Now you should have three maps open. Compare them and determine which method created the best map. Take note of which file it is (map1, map2, map3) as you will need to use it for the next section. Show your TA your best map. Ctrl-C all active terminals besides roscore.

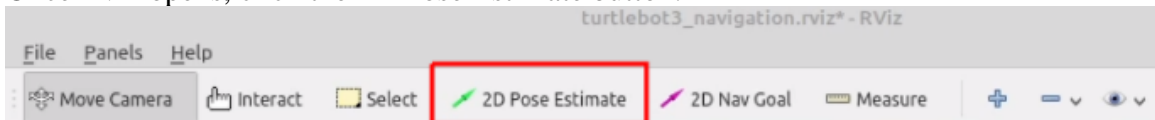
TA Initials: _____ Date: _____

2. Simulation Path Planning

One useful tool of creating maps is that the robot can move autonomously to a point with a path that is efficient and avoids obstacles in the map. To do this we need to use the navigation stack. You may remember that we used the `move_base` launch file in last week's lab from the navigation stack. However, instead of just using the `move_base` file, we need to launch the entire navigation stack and tell it what map to use. Make sure to replace `map#` with the map that you determined to be the best in the previous section.

```
roslaunch turtlebot3_navigation turtlebot3_navigation.launch  
map_file:= $\$$ HOME/map#.yaml
```

Once Rviz opens, click the 2D Pose Estimate button.

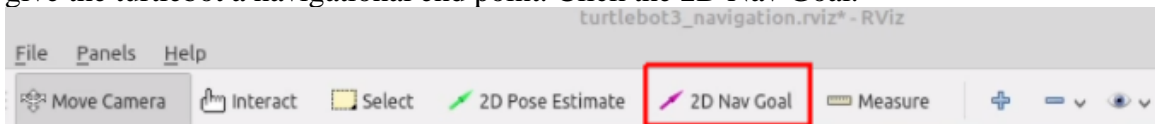


This will put a green arrow on the map with the turtlebot. Drag the arrow over to the direction the robot is facing. Repeat the steps of clicking the 2D Pose Estimate then dragging the arrow a few times.

Next we need to move the robot back and forth some to allow for a more precise initial pose estimate. Launch the teleoperation node then move the turtlebot a small amount back and forth.

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Ctrl-C the teleoperation node when you are done. Now that the initial pose is set, we can give the turtlebot a navigational end point. Click the 2D Nav Goal.



Now click on the map to tell the robot and end point. Once you click you will see a green arrow pop up. Rotate the arrow to tell the turtlebot the desired rotational position. Once you let go of the green arrow, the turtlebot should automatically move to the navigation goal.

TA Initials: _____ Date: _____

Ctrl-C all nodes besides roscore.

3. Physical Turtlebot SLAM

Now that you have simulated SLAM in ros, we are going to follow the same procedure with the real turtlebot in the lab. On the PC in the lab, it should give you the IP address of the turtlebot you are using, use that to ssh into the turtlebot. If it prompts you for a password, enter `turtlebot`.

```
ssh ubuntu@{IP_ADDRESS_OF_RASPBERRY_PI}
```

In the same terminal that you ssh'd into the turtlebot, launch the `turtlebot3`.

```
roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

Now open a new terminal and remote launch the SLAM node from the PC. Use the method that you found to be most effective from part one: Gmapping, Hector, Karto, or Frontier Exploration. Replace `BEST_METHOD` with that method.

```
roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=BEST_METHOD
```

In the lab there is a maze setup. Choose a spot to start your turtlebot and place it in the maze. On the PC, launch the teleoperation node.

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Drive the turtlebot around in the maze. Avoid changing speed or turning too quickly and make sure to get every part of the maze. After the map is complete, ctrl-C the teleoperation node then save the map.

```
roslaunch map_server map_saver -f ~/map5
```

Show your TA the map then Ctrl-C the

TA Initials: _____ Date: _____

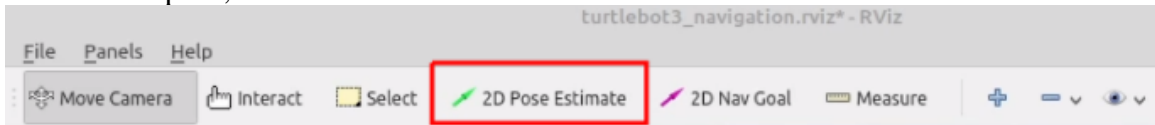
Ctrl-C the slam node.

4. Autonomous Movement Using SLAM

Similar to section 2, we are going to use the map we created to allow the turtlebot to autonomously navigate through the maze.

```
roslaunch turtlebot3_navigation turtlebot3_navigation.launch  
map_file:=$HOME/map5.yaml
```

Once Rviz opens, click the 2D Pose Estimate button.

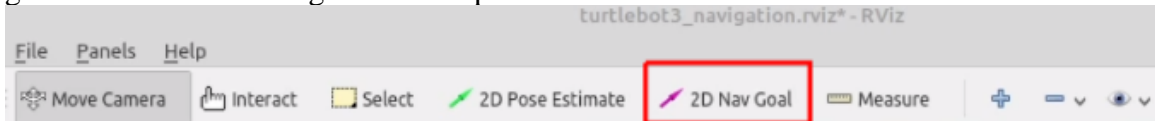


This will put a green arrow on the map with the turtlebot. Drag the arrow over to the direction the robot is facing. Repeat the steps of clicking the 2D Pose Estimate then dragging the arrow a few times.

Next we need to move the robot back and forth some to allow for a more precise initial pose estimate. Launch the teleoperation node then move the turtlebot a small amount back and forth.

```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

Ctrl-C the teleoperation node when you are done. Now that the initial pose is set, we can give the turtlebot a navigational end point. Click the 2D Nav Goal.



Now click on the map to tell the robot and end point. Once you click you will see a green arrow pop up. Rotate the arrow to tell the turtlebot the desired rotational position. Once you let go of the green arrow, the turtlebot should automatically move to the navigation goal.

TA Initials: _____ Date: _____

Ctrl-C all nodes.

A.10 Lab 10: Machine Vision

1. Camera Setup

Before we can use the camera on the turtlebot, we will need to calibrate the camera. Start by setting up a workspace called lab10 and then set up the bashrc file.

```
echo "source ~/lab10/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Next we need to install some packages.

```
cd ~/lab10/src/
git clone https://github.com/ROBOTIS-GIT/turtlebot3_autorace.git
cd ~/lab10

sudo apt-get install ros-melodic-image-transport ros-melodic-cv-bridge ros-
melodic-vision-opencv python-opencv libopencv-dev ros-melodic-image-proc

catkin_make
```

After your workspace compiles, launch roscore. In a new terminal, ssh into the robot. On the PC in the lab, it should give you the IP address of the turtlebot you are using, use that to ssh into the turtlebot. If it prompts you for a password, enter turtlebot.

```
ssh ubuntu@{IP_ADDRESS_OF_RASPBERRY_PI}
```

In the terminal that you just ssh'd into, run the following command to trigger the camera on the turtlebot.

```
roslaunch turtlebot3_autorace_camera turtlebot3_autorace_camera_pi.launch
```

In a new terminal (PC terminal, not the ssh terminal), execute the following command to view the camera image.

```
rqt_image_view
```

Once the image appears, change the first drop-down menu to **/camera/image/compressed**.

Leave the image open and execute this command in a new terminal.

```
roslaunch rqt_reconfigure rqt_reconfigure
```

This will open a gui so that you can calibrate the camera to get a clear image. In the left side of the gui, select **camera**. Change the sliders until you get a clear view from the camera image.

Once this is done, you can close the gui, but leave the camera image open. Next we need to change some values in the camera.yaml file.

```
cd ~/lab09/turtlebot3_aurorace_camera/calibration/camera_calibration
gedit camera.yaml
```

Once it is open, change the file to the following values then save and close.:

```
camera:
  ISO: 889
  awb_mode: tungsten
  brightness: 59
  contrast: 50
  exposureCompensation: 0
  exposure_mode: auto
  hFlip: false
  saturation: 0
  sharpness: 0
  shutterSpeed: 25000
  vFlip: false
  videoStabilisation: false
  zoom: 1.0
```

Now we have calibrated the camera, we need to do something called intrinsic camera calibration. This type of calibration estimates intrinsic parameters of the camera such as distortion, focal length and skew so that the image that the robot perceives is more accurate to “real-life”.

First we need to export some variables.

```
export AUTO_IN_CALIB=calibration
export GAZEBO_MODE=false
```

Next we can launch the intrinsic calibration gui.

```
roslaunch turtlebot3_aurorace_camera
turtlebot3_aurorace_intrinsic_camera_calibration.launch
```

In the lab, there should be papers with checkerboards on them. Position them in front of the turtlebot camera so that it fills the frame of the camera (check the camera view on the `rqt_image_view` terminal) but does not exclude any of the squares. Once you do this, click **calibrate** on the calibration gui.

Once it finishes calibrating, click save. This will create a compressed file in a `/tmp` folder. To extract it, run the following command.

```
cd ~/tmp
tar -xvf calibrationdata.tar.gz
gedit ost.yaml
```

This will open the ost.yaml file. Copy and paste the entire contents of this file. We need to paste this data into another file called **camerav2_320x240_30fps.yaml**.

```
cd ~/turtlebot3_aurorace_camera/calibration/intrinsic_calibration
gedit camerav2_320x240_30fps.yaml
```

When the file opens, paste the data from ost.yaml into the file then save and close. Next we need to perform extrinsic camera calibration. This will determine the position and imaging plane of the turtlebot3 camera.

First export some variables.

```
export AUTO_IN_CALIB=action
export AUTO_EX_CALIB=calibration
roslaunch turtlebot3_aurorace_${Aurorace_Mission}_camera
turtlebot3_aurorace_intrinsic_camera_calibration.launch
```

Next we can launch the extrinsic calibration file.

```
roslaunch turtlebot3_aurorace_${Aurorace_Mission}_camera
turtlebot3_aurorace_extrinsic_camera_calibration.launch
```

In a new terminal, execute rqt.

```
rqt
```

Select **plugins>visualization>Image view**. There should be two windows that open. One window, select /camera/image_extrinsic_calib/compressed and in the other window select /camera/image_projected_compensated

Keep these windows open and run the following command in a new terminal window.

```
roslaunch rqt_reconfigure rqt_reconfigure
```

In the left menu, under **Camera**, select **image_projection** and **image_compensation_projection**. Simultaneously view this window and the /camera/image_extrinsic_calib/compressed from the rqt window. Place the robot in the lanes that are setup in the lab. There should be one white and one yellow similar to an actual road. Place the robot such that the yellow lane is on the left and the white lane is on the right. Change the parameters in the reconfigure window until the red box on the camera image matches up with the lanes.

Make sure that the **image_compensation_projection** parameter is equal to 1. Ctrl-C all terminals besides roscore and the ssh terminal.

We are almost done with calibration, we just need to apply our changes to the camera. To do this, execute the following lines in the command terminal

```
roslaunch turtlebot3_aurorace_camera
turtlebot3_aurorace_intrinsic_camera_calibration.launch
```

```
export AUTO_EX_CALIB=action

roslaunch turtlebot3_atorace_camera
turtlebot3_atorace_extrinsic_camera_calibration.launch
```

Once this is done, the camera is finally calibrated. As you can see, this process can be lengthy and time consuming. However, as we will see in the next sections, having vision systems allows robots to do things that laser scanners simply cannot achieve. Leave the calibration launch terminals open and move onto the next section.

2. Lane Detection Calibration

The robot should be placed in the lanes setup in the lab. Make sure that the yellow lane is on the left and the white lane is on the right. In a new terminal, launch the lane detecting file.

```
export AUTO_DT_CALIB=calibration
roslaunch turtlebot3_atorace_detect turtlebot3_atorace_detect_lane.launch
```

In a new terminal, execute rqt.

```
rqt
```

Select **plugins>visualization>Image view**. There should be two windows that open. One one window, select `/camera/image_extrinsic_calib/compressed` and in the other window select `/camera/image_projected_compensated`. Create 3 windows and then select the following in each respective window:

```
/detect/image_yellow_lane_marker/compressed
/detect/image_lane/compressed
/detect/image_white_lane_marker/compressed
```

Open a new terminal and execute:

```
roslaunch rqt_reconfigure rqt_reconfigure
```

In the left menu, select **detect_lane**. The process of determining the parameters for lane detection can be finicky and may take some time. To get started, open `lane.yaml` and copy the values into the `rqt_reconfigure` gui.

```
cd turtlebot3_atorace_detect/param/lane
gedit lane.yaml
```

Once this is done here is a general outline for how to adjust the parameters further. Note you should be adjusting the parameters and looking at each of the three image windows that you have open to determine if your adjustments are correct.

1. Start with hue values first. These are listed in the configure gui as `hue_white_1`, `hue_yellow_1`, etc.

2. Next do saturation values; make sure to do the saturation for both white and yellow.
3. Now adjust the lightness values
 1. You do not need to adjust the `lightness_white_l` or `lightness_yellow_l` values. Just set the `_h` lightness values to 255 and leave the `_l` values alone.

Once you are done calibrating, change the new parameters you have set in the **lane.yaml** file. You should still have this file open, if you don't, just look a couple lines up to get the file open. Save and close the file then ctrl-C the `rqt_rconfigure` and the `detect_lane.launch` terminals then execute the following commands.

```
export AUTO_DT_CALIB=action
roslaunch turtlebot3_autorace_detect turtlebot3_autorace_detect_lane.launch
```

3. Lane Detection Testing

Now that we have calibrated the camera, we can launch our lane detection node.

```
roslaunch turtlebot3_autorace_control
turtlebot3_autorace_control_lane.launch
```

Next we need to bringup the turtlebot3. **Make sure you execute this command in the ssh terminal.**

```
roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

After you launch this command, the turtlebot should start navigating through the lanes. The lane detection from the auto race package we are using implements a PD controller. This enables the turtlebot to navigate through the lanes very accurately, even with disturbances. As the turtlebot is going through the lanes, try pushing it a little bit of course (not completely out of the lanes though) to see if it will “course-correct”. Find the source files for the `turtlebot3_autorace_control_lane.launch` file and change the parameters for the PD controller. Comment on the performance of the controller and how you changed it.

Run an `rqt_graph` to see what nodes are running and to see how they are connected to each other.

TA Initials: _____ Date: _____

Ctrl-C all nodes.

A.11 MATLAB Introduction

1. MATLAB Setup

Ensure that MATLAB is allowed through the firewall of the PC so that it can transmit and receive messages to/from ROS. On Windows, this is done with the “Allow an app through Windows Firewall” application.



Allow an app through Windows Firewall

Control panel

Matlab needs to have python installed. The python version depends on the MATLAB version. MATLAB 2020a uses Python 2.7.8 (<https://www.python.org/downloads/release/python-278/>). Once it is installed, execute the following command in the MATLAB command window.

```
pyenv('Version','C:\Python27/python')
```

If it downloaded correctly, these commands should output “2.7” in the command terminal of MATLAB.

```
pe = pyenv;  
pe.Version
```

To connect MATLAB to ROS, you will need to know the IP of your PC and the IP of the virtual machine running ROS (if using one). To start, run `roscore` on the command terminal that is running ROS. From now on, this terminal will be referred to as the VM. To start a ros connection from MATLAB, the `rosinit` command is used.

```
rosinit("http://IP_OF_VM:11311","NodeHost","IP_OF_PC");
```

To ensure that MATLAB is receiving information from ROS, run a `rostopic list` on MATLAB.

```
rostopic list
```

2. Control Turtlebot3 in Gazebo from MATLAB

On MATLAB, open a new script and add the following code but do not execute yet.

```
%SETUP PUBLISHER
velocity=-0.1;
robotCmd = rospublisher("/cmd_vel","geometry_msgs/Twist");
velMsg = rosmessage(robotCmd);

%SEND VELOCITY COMMAND
velMsg.Linear.X = velocity;
send(robotCmd,velMsg)
pause(4)
velMsg.Linear.X = 0;
send(robotCmd,velMsg)

%SETUP SUBSCRIBER FOR POSE
odomSub = rossubscriber("/odom","nav_msgs/Odometry");
odomMsg = receive(odomSub,3);
pose = odomMsg.Pose.Pose;
x = pose.Position.X;
y = pose.Position.Y;
z = pose.Position.Z;
[x y z]
```

On the VM, launch the Turtlebot3 in Gazebo.

```
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

Once Gazebo is launched, run the MATLAB script. Ensure that the Turtlebot3 moves forward and then stops.

3. Plot Scan Data

Update the MATLAB script and add the following code to the end.

```
%SETUP SUBSCRIBER FOR LIDAR
laser= rossubscriber("/scan","sensor_msgs/LaserScan");
scanMsg = receive(laser);
figure
plot(scanMsg)

% SOME TEST CODE
velMsg.Angular.Z = velocity;
send(robotCmd,velMsg)
tic
while toc < 20
    scanMsg = receive(laser);
    plot(scanMsg)
end
```

```
velMsg.Angular.Z = 0;  
send(robotCmd,velMsg)
```

Run the script. A plot of the data from the laser-distance scanner should show and be updated for 20 seconds

4. Obstacle Avoidance Script

At the end of the current MATLAB script, add the following code which implements a simple obstacle avoidance algorithm using lidar.

```
%TEST  
distanceThreshold=0.1;  
tic;  
while toc < 20  
    % Collect information from laser scan  
    scan = receive(laser);  
    plot(scan);  
    data = readCartesian(scan);  
    x = data(:,1);  
    y = data(:,2);  
    % Compute distance of the closest obstacle  
    dist = sqrt(x.^2 + y.^2);  
    minDist = min(dist);  
    % Command robot action  
    if minDist < distanceThreshold  
        % If close to obstacle, back up slightly and spin  
        velmsg.Angular.Z = spinVelocity;  
        velmsg.Linear.X = backwardVelocity;  
    else  
        % Continue on forward path  
        velmsg.Linear.X = forwardVelocity;  
        velmsg.Angular.Z = 0;  
    end  
    send(robotCmd,velmsg);  
end
```