CS/ME/ECE/AE/BME 7785

Lab 4

Planned ROS Computation Diagram Drawing Due: March 07, 2025

Lab Demo Due: March 14, 2025 at 4 pm

1 Overview

The objective of this lab is to design a set of controllers to make a robot drive through a set of way points, given to you in a text file, in the presence of unknown obstacles. Figure 1 shows a cartoon of the path that the robot will follow. The blue box is in a known stationary position within the environment, while the purple object will be added by an instructor during the demo. The robot will use onboard odometry and dead reckoning to determine its global position during the navigation. It will be assumed that the robot starts at global position (0m, 0m) with orientation aligned with the x-axis.

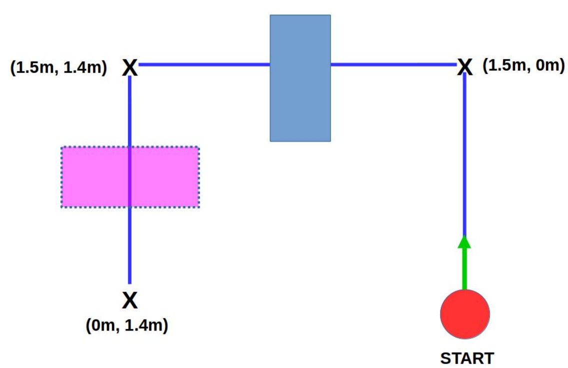


Figure 1: Cartoon of the experiment setup. The blue box is fixed in place while the purple faded box will be placed by an instructor during the demo. Note, the purple faded box will be put in your robots path and you should not assume its orientation.

You are not required to run all files on-board the robot. If you wish to run some computational ROS nodes on your computer, you can choose to. Be aware that WiFi communication can be unreliable. If you choose to constantly communicate information between the robot and your computer you are accepting the risk that your demo may fail due to WiFi communication.

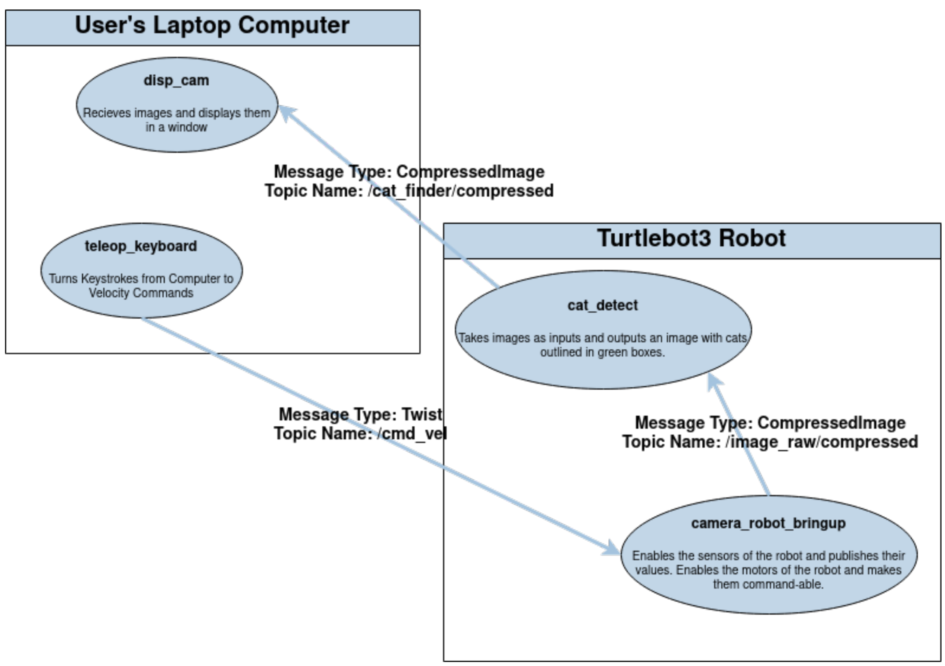
We strongly encourage you to use all available resources to complete this assignment. This includes looking at the sample code provided with the robot, borrowing pieces of code from online tutorials, and talking to classmates.

You may discuss solutions and problem solve with others in the class, but this remains a team assignment and each team must submit their own solution. Multiple teams can not jointly write the same program and each submit a copy, this will not be considered a valid submission.

Planned ROS Computation Diagram Drawing

Before you start coding, it is often a good idea to sketch out how and where you will do computation and send data to solve the problem.

The first step of this lab is to draw a ROS computation graph of how you will solve the problem. This should look similar to an rqt-graph but with a few more details. You should include the nodes and what they are computing, the topics and messages they subscribe and publish to, and where the nodes are physically going to run (i.e. the Turtlebot or your computer). For example, if I were to make ROS computation graph for tele-operating the robot while receiving a camera feed that is finding cats, it may look like the following:



You can draw this diagram on the whiteboard or in your notebook and take a picture for submission or use an online tool like draw.io. The purpose of this step is to have you plan your software solution before coding furiously!

2 Lab Instructions

Create a package called `TeamName_navigate_to_goal` (refer back to the ROS tutorials from Lab 1 if needed). Useful dependencies include `rospy`, `roscpp`, `sensor_msgs`, `std_msgs`, `nav_msgs`, and `geometry_msgs`. You can add as many nodes as you like. An example structure would be:

getObjectRange: This node should detect the ranges and orientation of obstacles. It should subscribe to the `scan` node and publish the vector pointing from the robot to the nearest point on the object.

> Note: You will have to do some filtering of the LIDAR data to determine what measurements of the 360 are useful. You may also want to segment your readings to be able to discern two obstacles apart from one another. You will only encounter one obstacle at a time, but if your LIDAR sees the wall or a stray chair/other robot you will want it's object estimate to be robust. It is also important to remember that this data is with respect to the robot's local coordinate frame.

goToGoal: This node should subscribe to the `odom` node which determines the robots global position from onboad sensors for you (using dead reckoning). It should also subscribe to the getObjectRange node to determine if there are any obstacles that need to be avoided.

This node should first read in the given goal locations from the `wayPoints.txt` file, or you can include them in your code some other way. You should then create several controllers that drive the robot through the sequence of given goal points without colliding with unknown obstacles. To receive full credit the robot must stop for 10 seconds within a 10cm radius of the first goal point, 15cm radius of the second goal point, and 20cm radius of the third goal point, the robot must not hit any obstacles, and the robot must reach the destination in under 2 minutes 30 seconds.

3 Possible Issues

1. Remember the onboard odometry and goal points are given in the same global frame while the measurements are in the robot's local frame. The package tf2 in ROS (`https://docs.ros.org/en/humble/Tutorials/Intermediate/Tf2/Introduction-To-Tf2.html`) may be useful to transform coordinate frames if you want but is not necessary for this lab.

2. The Turtlebot3 has built in odometry which you are free to use. You can access it by subscribing to the `/odom` topic. It relies on proper calibration beforehand which can mess up if you move the robot during its bringup. It is highly recommended to print out the robot's estimated pose to make sure the odometry is correct and not drifting while the robot is stationary. If you find it is messed up it can be fixed by placing the robot on the floor and restarting the bringup.

3. The odometry node saves the current position of the robot and starts where it left off. If you pick up the robot and restart your program to run the course, the odometry given to the robot will be the position and orientation the robot was last at before you picked it up. We have given you a python script (RotationScript.py) which records the initial odometry readings and subtracts them as an offset so your assumed starting position is the origin with heading aligned with the x-axis. You may integrate this into your project however you want. The script provided does not run on its own but assumes it is within a class with a variable self.Init that should be set to true in the beginning of the class you use it in (this allows the provided function to save the initial odometry measurements to correct your starting position every run). As always, reach out to the instructors if you have any questions about this script.
4. The angular component of the odometry is represented by a quaternion which should be used appropriately.
5. If you wish to create dead reckoning position updates yourself, or augment the ones produced in the /odom topic, you can access the IMU and encoders through published topics /imu and /sensor_state. More details can be found at http://wiki.ros.org/turtlebot3_bringup#Published_Topics.
6. There are some problems with the motor commands sent to the robot. Make sure your commands are below 0.2m/s linear and 1.5 rad/s angular.

4 Grading

You are allowed **5 attempts** to demo this to an instructor and will receive the best score of your attempts.

Planned ROS2 Computational Diagram	25%
Drive within 10cm of the first goal point	$25\%(e^{-\frac{stopped\_distance\_outside\_of\_goal\_in\_cm}{25}})$
Drive within 15cm of the second goal point	$25\%(e^{-\frac{stopped\_distance\_outside\_of\_goal\_in\_cm}{25}})$
Drive within 20cm of the third goal point	$25\%(e^{-\frac{stopped\_distance\_outside\_of\_goal\_in\_cm}{25}})$
Each collision with an obstacle	-5%
Take more than 2 minutes, 30 seconds to reach the final goal point	-15%

Example grade:

You run all your code on the robot. Your robot reaches the first goal point within 10cm, hits the obstacles once but makes it within 20 cm of the second goal point, and then reaches the final goal point within 20cm. This is all done within 2 minutes, 30 seconds. Your grade would be...

$$\begin{aligned} grade &= 25 - 5 + 25e^{-\frac{5}{25}} + 25 + 25 \\ &= 25 - 5 + 20.5 + 25 + 25 \\ &= 90.5 \end{aligned}$$

5 Submission

1. Perform a live demonstration of you running the robot to one of the course staff by the listed deadline.

2. Put the names of both lab partners into the header of the python script. Put your python script and any supplementary files, in a single zip file called Lab4_LastName1_LastName2.zip and upload on Canvas under Assignments–Lab 4.

3. Only one of the partners needs to upload a submission.

We will set aside class time and office hours on the due date for these demos, but if your code is ready ahead of time we encourage you to demo earlier in any of the office hours (you can then skip class on the due date). Office hour times are listed on the Canvas homepage.