ME 4451 Lab 6 Handout

Anirban Mazumdar, Bryan Blaise, Nikhil Nandikanti

(3 Session Lab when combined with Lab 7)

Spring 2024

1 Objectives

1. Get familiar with the Turtlebot and ROS
2. Be exposed to the motion control of mobile robots
3. Learn how to read odometry and lidar data from the Turtlebot

2 Prelab Assignment

- Read Lab 6 Handout
- Write a closed-loop version of the function `turtlebotGoDistance`
- Write a script executing the square path and odometry data collection throughout the path

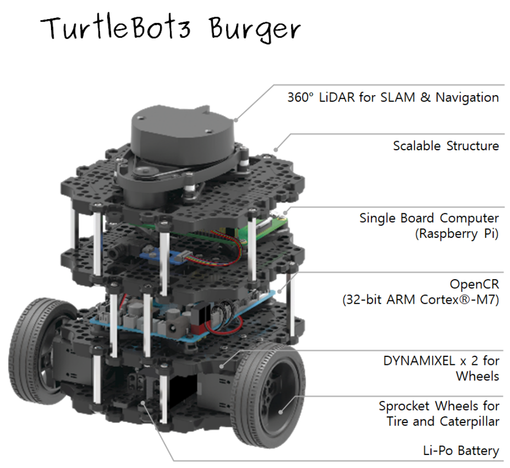
3 Connecting and Controlling

The TA will do a short ROS conceptual overview and a get-to-know-your-turtlebot. They will explain the various sensors on the robot you can read data from and the methods for issuing commands to the robot's wheels. NOTE: `lab6_skeleton.m` contains code sections for connecting to the robots and learning how to implement ROS through MATLAB. Take your time to look through and understand each part.

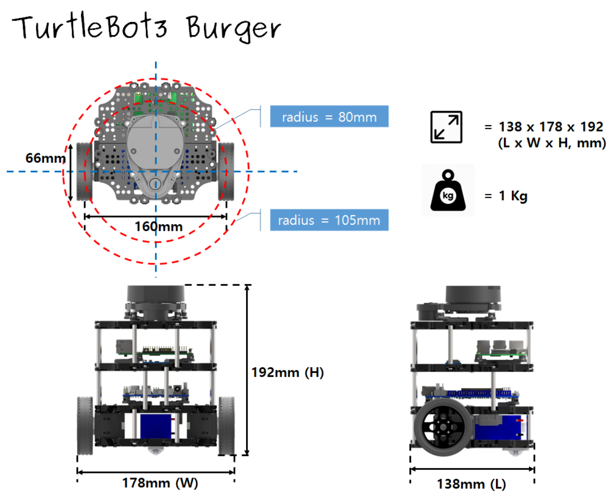
DO NOT RUN THIS AS A SCRIPT. Run individual sections as needed

3.1 Turtlebot3 Burger

For this lab, you will be using the Turtlebot3 Burger. Turtlebot3 is a new generation mobile robot that's modular, compact and customizable.


TurtleBot3 Burger
360° LiDAR for SLAM & Navigation
Scalable Structure
Single Board Computer (Raspberry Pi)
OpenCR (32-bit ARM Cortex®-M7)
DYNAMIXEL x 2 for Wheels
Sprocket Wheels for Tire and Caterpillar
Li-Po Battery

(a) Turtlebot3 Components.



(b) Turtlebot3 Dimensions.

Figure 1: Turtlebot3 Burger will be used for this lab to explore ROS and mobile robotics.

3.2 Getting ROS Up and Running on TurtleBot

With your laptop, make sure you are connected to the local WiFi ('TP-Link_05E4') rather than GT's WiFi (eduroam). If on Windows 10, it will likely say "Connected, No Internet". That is fine.

If the robot is not running, first check if the LiPo battery is connected and the switch is turned on. If the issue continues, ask the TA for help. Often, even after the robot has been working for an extended period, it develops an issue communicating with MATLAB or ROS. Luckily, many issues can be resolved by stopping ROS, shutting down the robot, and restarting.

3.3 Initialize Communication with the Robot

Use `lab6_skeleton.m` to connect to your TurtleBot:

- We recommend using your laptop since there are often issues with the desktop computers in lab's networking permissions preventing proper communication with the robots.
- Make sure your computer is connected to the lab wi-fi (SSID: TP-Link_05E4, PASSWORD: 52779077).
- Find the IP address of the TurtleBot you want to connect to. The IP address should be 192.168.0.xxx (where xxx is unique for each robot).
- Open your terminal/command prompt. SSH into the TurtleBot by typing the follow command `ssh su@192.168.0.xxx`. The password is me4451. You may get a yes/no prompt before you can input the password. Input `yes`

- Once you've SSH'ed into the robot, press the up arrow to cycle through previous commands executed on the robot. You'll see a command beginning with roslaunch. Execute that command. The robot should initiate its functions
- Enter your turtlebot's IP address in the first code block in MATLAB and run the block

3.4 Explore ROS

Use `lab6_skeleton.m` to test some functions to control the turtlebot. Typing `rostopic list` in the MATLAB command window will list all the available "topics" you can either publish or subscribe to on the robot. A ROS publisher publishes a specific type of message along a topic. A topic is the communication stream along which the messages are sent.

Each message type has its own data structure. Most have several subfields with values representing the different components of the message. For example, take a look at the 'Twist' documentation on the ROS Wiki. It has two subfields, 'linear' and 'angular', representing the robot's linear and angular velocity. Both these fields have have their own subfields.

To send velocity commands to the robot, you would publish messages of the type 'Twist' on the topic '\cmd_vel'. The robot is subscribing to the same topic, waiting for new velocity commands. Once a new command is published on your end, it will be received by the robot and the turtlebot will update the velocity commands it sends to its wheels.

The robot is also publishing data from its sensors at regular intervals. If you are interested in receiving data from the robot, you do so by subscribing to a particular topic using a Subscriber node. For example, if you wanted to read acceleration data from the robot, you would create a subscriber that reads messages of the type Imu from the topic '\imu'.

In `lab6_skeleton.m`, each code block builds on the previous and adds additional functions. These functions will help you perform some low-level tasks. Open the custom functions to understand what's going on under the hood and be able to explain them when asked. Later, if you'd like to optimize your code, you can improve upon these functions.

4 Reading Sensor Data

Before attempting to control the robot, familiarize yourself with the method for reading the data the robot is sending out. For our purposes, we are concerned with reading the data generated from the lidar and odometry sensors. You will need these in lab 7 so we'd like to make sure you are comfortable reading and interpreting the data.

4.1 Reading Odometry

Odometry is the use of motion sensors to determine the robot's change in position relative to some known position. In `lab6_skeleton.m`, you have sample code to create a subscriber for the odometry data using the `rossubscriber` command, but we left out the topic name. Refer to the list of topics generated by the command `rostopic list` to figure out what topic to subscribe to.

The command `rosmessage(odometrySubscriber)` creates an empty data field in the correct structure for the message type. Print its output to the console to see its contents. It is useful to visualize how the data is structured so you can see how to craft your own data to publish on the subscriber.

To actually read the odometry data, use the `receive(odometrySubscriber)` function. You can do this as a one time command or in a loop (with pauses). Inspect the `getTurtlebotOdometry` function and the ROS Wiki page for the topic to get a better understanding of how to extract the useful data from the field.

Note: Every time the robot is rebooted, it's odometry data resets to zero. This means, the robot believes it is at the origin and facing the positive x direction. Thus, the robot considers the position and orientation upon reboot to be the global reference frame.

4.2 Reading Lidar

Lidar is used to detect the distances of objects surrounding the robot. It does so by emitting lasers that bounce off objects and measuring the time it takes for them to return. Follow the same sequence of steps you used to read the odometry data:

- Identify the appropriate topic to read using `rostopic list`
- Create a subscriber to read the Lidar
- Look up the corresponding message's documentation on the ROS wiki to get a better understanding of the data structure and each field's contents
- Generate a blank sample message using the `rosmessage` command to inspect its format
- Read data from the laser's topic and inspect the data. Determine the units for the distances reported by the sensor

In a loop, continuously read the laser data while moving an object towards and away from the robot from different directions. Notice which elements of the laser data are changing. Use this to identify which elements of the array correspond to the front, right, left, and rear of the robot? Be sure you're no closer than the laser's minimum readable distance and no further than its maximum readable distance.

Once you've identified the range of indices corresponding to each direction, demonstrate this to the TA.

5 Commanding the Robot

Now that you're comfortable reading data from the robot's sensors, you can try sending commands to the actuators. In this case, you are not sending voltage or velocity commands to the individual wheels directly. Instead, you are sending velocity commands for the entire robot, and the turtlebot, under the hood, generates the appropriate voltage commands for the wheels.

Try editing the velocity message's contents and publishing this command to the robot. The only two fields that should be modified are the `Linear.X` and `Angular.Z` fields, since those are the only degrees of freedom the robot has. Try to make the robot move forward, backwards, and turn in place in both directions. How can you make the robot execute a moving turn (i.e. turn while traversing around a corner)? Make a script that reads in the initial odometry data, have the TurtleBot move for a few seconds, stop the robot, and read the odometry data again.

Try using the `turtlebotGoDistance` and `turtlebotTurnAngle` functions to execute motions. Are these functions accurate? Can you improve upon these functions? What could be the sources of their error?

`turtlebotGoDistance` is written as an open-loop function. This means it does not use sensor readings to inform the motion command sent to the robot. Observe how `turtlebotTurnAngle` uses the odometry reading. As part of your pre-lab, improve upon the `turtlebotGoDistance` function by converting it to a closed-loop function.

6 Path Following

6.1 Square Path

To earn credit for this lab, you will need to program the robot to follow a square path, where the robot will end up at its starting point. Write code using the provided functions, improve upon them, or build your own. For the length of the sides, use $3ft$. This is equal to the length of 3 of the floor tiles in lab. The TAs have outlined the target path in tape on the floor of the lab. Your task is to execute the square path while remaining within the lines. If your robot touches the lines, you'll be asked to restart until you can complete the path. Make sure you can complete the path in both the clockwise and counterclockwise directions.

While executing the path, you are to collect odometry data of your robot. At the start of the path, end of the path, and each of the 4 corners (before and after the turns), save the pose of the robot (x, y, and θ). Do this for both the forward and reverse directions. Plot this sequence of poses.

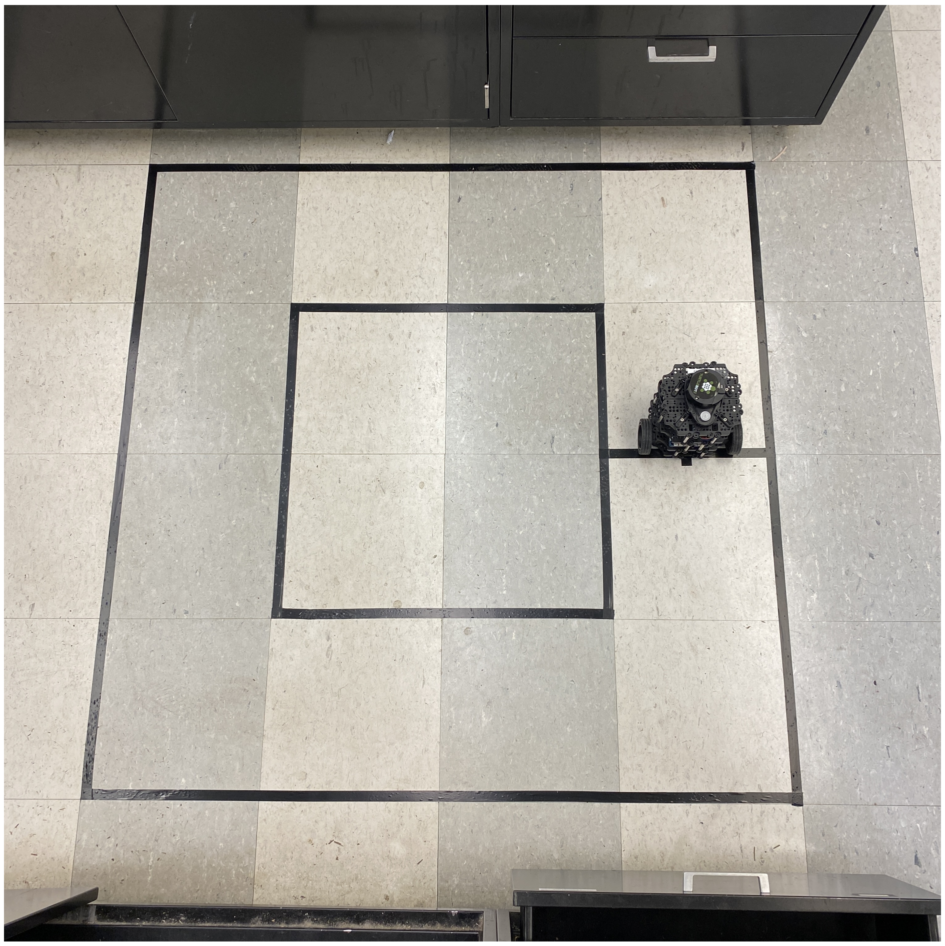


Figure 2: Square path for robot to follow. Make sure the robot stays between the lines.