School of Computer Science and Engineering

COMP3431/9434: Robot Software Architectures

Project 1: TurtleBot3 Mapping and Navigation

Due Date: Week 4 (roughly)

1 Overview

The aim of this project is for you to gain familiarity with working with robots, developing robot software, and using ROS and RViz. The project will require you to combine common aspects of robot software, such as mapping, navigation, simple vision processing, transforming between different coordinate frames of reference, and autonomous control.

The project, like the course itself, requires self-driven learning. It will take time to get used to ROS, so you should start early.

You will work in groups of five. Your team should work closely together, developing the code together, so that every member of the team has a good understanding of how all of the parts work. This understanding will help you in the second project.

All mobile robots have to know where they are and where they are going. The robot must build a map of its environment as it explores, and places markers in the map where specific features are found. To make the first project simpler, these features will be AR markers, that are easily identified. All of the techniques you use here are applicable to robots in domestic and work environment and are common in almost all mobile robots, even self-driving cars.

1.1 Task

The assignment is to create a map and navigate around an simple environment, using the Turtlebot3 Waffle Pi robot that is equipped with a laser scanner and a webcam.

Your task is to write software that is incorporated into the ROS infrastructure. Your software should enable the Turtlebot to:

1. Explore its world, where both you and the robot will not know the structure of the world before starting,
2. Locate any AR markers in the world,
3. Build a map of the world as the robots explores,
4. Place the location of the AR markers in the map,
5. Plot the path the robot took on the map,
6. Return the robot to its starting location once all AR markers have been located, and
7. complete the task as quickly as possible.

Your robot should be fully autonomous. Once the robot has been instructed to start (ideally using an RViz control plane) you may not interact with the robot until it completes the task and stops itself. The only exception is if you choose to terminate the program and restart from the beginning.

The performance of your robot will be evaluated as a (hopefully) live demonstration to be held during the week 4 lab. We have some flexibility with the deadline, but we should avoid crowding week 5 because other assessments will be due and the sooner you start the second project, the better.

1.2 Group Work

You will work in groups of five.

1.3 Deliverables

Your group's assignment submission consists of three deliverables:

1. ROS package written in C++ or Python containing all code, launch files, and other content required to run your group's software,
2. Live demonstration data collected using standard ROS messages, observed during testing but not collected
   - Map of the world,
   - Locations of AR markers (in the global/map frame of reference), and
   - Path travelled by the robot during exploration (in the global/map frame of reference).
3. Individual reports describing how your group solved the task, and analysing the performance of your system.

1.4 Due Dates

Deliverable	**Date**	**How**
Live Demonstration	Lab, Week 4	See Section 2
ROS Package	Friday 5pm, Week 4	git
Report	Friday 5pm, Week 4	give project1

2 Live Demonstration

The performance of your software will be assessed during a live demonstration. This process is described in Section 2.1.

2.1 Day of Assessment

Your work will assessed as follows:

1. Each group will be given 15 minutes to complete the task.
2. The robot will be placed in a maze at a pre-determined starting location.
3. The robot will be instructed to start locating AR markers, starting a "run".
4. The robot will operate until:
   - The group decides to manually terminate the run, or
   - The autonomous software decides to terminate the run, giving a clear indication.
5. Your group may decide to restart and commence a new run as many times as desired within the 15 minute window. Only the best run count towards your group's final mark.

To assist with the assessment on the day of the demonstrations:

- All robots will be placed on charge at 9am.
- For safety, one robot will be removed and placed on charge at 5pm the preceding day.
- During the assessments, only TWO robots may be in use:
  - The robot currently being operated in the maze, and
  - The robot being set-up by the next group to be assessed.
- All inactive robots will remain on charge.
- All network traffic should be minimised.

2.2 Assessment Criteria

Your group's assignment will be assessed during the live demonstration. The assessment will be made based on:

- The quality and accuracy of the map generated by your robot.
- The accuracy of the placement of the AR markers.
- The accuracy of the recorded path traversed by the robot.
- The accuracy and ability of the robot to return to it's starting location.
- The speed at which the robot completes the task.

Marks will be lost for:

- Interacting with your autonomous software, other than issuing start/stop commands.
- The robot colliding with the maze or markers.
- Interfering with the runs of other groups.

3 Report

Your group must submit a short report of no more than 5 pages describing:

- How your group solved the task,
- The ROS packages that you chose to use with and explanation of why these packages were chosen,
- An evaluation of the performance of your autonomous software, including an explanation of the performance that was observed during the live demonstration.
- The limitation of your software, specifically any shortcoming or short cuts that were taken which may prevent your software solving some variations of the task.

4 Software

It is not feasible for your group to write code from scratch to solve all of the requirements of the task. In fact part of the purpose of this assignment is to become familiar with integrating your software with other systems.

4.1 ROS packages

Your group is free to use ROS packages that are available through the standard Ubuntu ROS distribution. These are packages already installed on the lab machines and the VM. If you wish to use another ROS package not in the standard distribution, or you are unsure if you can use a ROS package, check with Claude or your tutors first.

5 Getting Started

This section makes some suggestions to help you get started on the assignment. However, you are not restricted to using only what is mentioned here.

5.1 Starter Code

We have provided you a ROS package containing a set of starter code, written in C++. If you wish to use Python, you will need to convert this code yourself. The code includes a *wallFollower* node that will track the left hand side wall of a maze. This allows the robot to autonomously explore the maze. Note that this node does not generate a map of the maze but only explores it. The *wallFollower* node is launched via a launch file located in the *launch* folder of *comp3431_starter* package. It can be run as follows:

```
roslaunch comp3431_starter wallFollow.launch
```

You will also notice some other launch files and src code in this package such as the *beacons* code. This was code that was used in previous years when the course was run slightly differently. Feel free to play around with this code but note that for the first assignment you should only need the *wallFollower* from this package.

The starter code is available in the public COMP3431 GitLab repository:

```
git clone http://robolab.cse.unsw.edu.au:4443/comp3431-rsa/comp3431-rsa.git
```

When you have successfully cloned this branch cd into it and make a new directory called src. Move the folder comp3431_starter into src and then run catkin_make. This should compile all of the code correctly.

5.2 Mapping

You need to produce a globally correct map of the maze. The standard approach to this is SLAM (Simultaneous Localisation and Mapping). We recommend using one of three SLAM implementations:

- Gmapping
- Cartographer
- Hector-SLAM

Do not attempt to implement your own SLAM, given the time requirements.

5.3 Exploration

Exploration involves the robot moving about an environment for which it does not have a map. For exploration there are three alternatives:

- Frontier Explorer (ROS Navigation Stack).

- Simple wall following exploration that we have provided in `comp3431_starter`
- Implement your own exploration.

As already explained, the starter code provides a ROS node that implements a simple wall follower that uses only the current laser scan. The node subscribes to messages of type `sensor_msgs::LaserScan` on the topic `/scan`. You will need to ensure the laser is being published on this topic, or change the topic name as required.

When the node starts, the robot does not drive. Instead, the node listens to messages of type `std_msgs::String` on the topic `/cmd`. Sending the message "`start`" will start driving the robot, while sending the message "`stop`" will stop driving the robot. An example script that does this can be found in the `comp3431_starter/scripts/cmd_controller.py`. The drive commands from the node are published on the topic `/cmd_vel` and are of type `geometry_msgs::Twist`.

The parameters for the wall follower are currently hard-coded. We recommend that if you modify this to load parameters from a `roslaunch` file.

5.4 Waypoint Navigation

Navigation is different to exploration since the robot is attempting to reach a specific location, rather than just driving around finding new areas. Thus, a different algorithm is required. For exploration there are three alternatives:

- ACML (ROS Navigation Stack).
- Implement your own A^* (not recommended given the time constraints).

5.5 Vision Processing

Vision processing requires you to identify the AR codes in the camera images, and calculating their positions in the world frame of reference.

There are many possible ways of finding the AR codes:

- Subscribing the the RGB image and using an AR library e.g. `https://docs.opencv.org/trunk/d5/dae/tutorial_aruco_detection.html`. Provided you have calibrated the camera, it should be possible to get a 3D pose of the marker relative to the camera (since the size of the marker is known).
- Using an AR library with a ROS wrapper e.g. `http://wiki.ros.org/ar_track_alvar`

Note that to get accurate positions you will need to calibrate the camera.

Vision processing is something that you need to be more aware of for assignment 2 however it is a good idea to familiarise yourself with the concepts as early as possible. For the purpose of assignment 1 we will only be asking you to identify some AR markers that will be placed at random throughout the maze. These can be identified using multiple open source packages, some of which already come with a ROS wrapper and others you may need to write the wrapper yourself.

6 Getting Help

This assignment is largely self driven and will require you to investigate how to solve these problems. The ROS wiki is well documented and has good tutorials to help you solve each of the elements of the assignment.

Claude and any of the tutors will be around to provide help, mainly during the lab times each week. We will be around the lab at other times, but may only will be able to give limited help depending on our schedules.