School of Computer Science and Engineering

COMP3431: Robot Software Architecture

Assignment 1: Turtlebot Rescue with ROS

Due Date: Week 5

Version 1.0: Initial Spec
Version 1.1: Corrected file names in Section 5.1
Version 2.0: Initial Release

1 Overview

The purpose of this assignment is to allow you to gain familiarity with working with robots, developing robotic software, and using ROS/RViz. This assignment will require you to combine common aspects of robotic software, such as mapping, navigation, vision processing, transforming between different coordinate frames of reference, and autonomous control.

This assignment, much like the course, requires self driven learning. It will take time to wrap your head around the ROS system. Therefore, it is advised that you start early.

You will work in groups of four. Your team should work closely together on the assignment, 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 assignment/project.

The assignment is inspired by the the challenge of autonomous robotics for Urban Search and Rescue (USAR). In USAR after a disaster occurs, a robot is used to explore an unknown environment, such as a partially collapsed building, and locates trapped or unconscious human victims of the disaster. The robot builds a map of the collapsed building as it explores, and places markers in the map of where the victims are located. This map is given to trained first responders who use it to go into the building and rescue the victims. All of the techniques you use here are also 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 solve a simplified version of the USAR problem using the Turtlebot3 robots which come equipped with a laser scanner and a RealSense RGB-D camera. For the simplified problem, the "building" is represented by a maze and "victims" by four unique coloured beacons.

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

1. Explore the maze, where both your and the robot will not know the structure of the maze before starting,
2. Locate any beacons contained in the maze,
3. Build a map of the maze as the robots explores the maze,
4. Place the location of the beacons in the map,

5. Plot the path the robot took on the map,
6. Return the robot to its starting location once all beacons have been located, and
7. Complete the simplified USAR task as fast 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 simplified USAR 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 live demonstration to be held during the week 5 Wednesday lab.

1.2 Group Work

You will work in groups of four.

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:
   - Map of the maze,
   - Locations of beacons (in the global/map frame of reference), and
   - Path travelled by the robot during exploration (in the global/map frame of reference).
3. A report describing how your group solved the simplified USAR task, and analysing the performance of your system.

1.4 Due Dates

Deliverable	**Date**	**How**
Live Demonstration	Wednesday Lab, Week 5	See Section 2
ROS Package	Friday 5pm, Week 5	git
Report	Friday 5pm, Week 5	give assignment1

2 Live Demonstration

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

The deliverables for the live demonstrations will be collected on the assessor computer, which will be connected to your `roscore`. To assist with this collection, you are required to publish messages on common topics, and use common frames of reference, described in Section **??**.

2.1 Day of Assessment

Your work will assessed as follows:

1. Each group will be given 15 minutes to complete the simplified USAR task.
2. The robot will be placed in the maze at a pre-determined starting location.

3. The robot will be instructed to start locating beacons, 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 live 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 beacons.
- The accuracy of the recorded path traversed by the robot.
- Using the correct topics and coordinate frames of reference to publish all deliverables.
- The accuracy and ability of the robot to return to it's starting location.
- The speed at which the robot completes the simplified USAR task.

Marks will be lost for:

- Interacting with your autonomous software, other than issuing start/stop commands.
- The robot colliding with the maze or beacons.
- 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 simplified USAR task.

4 Software

It is not feasible for your group to write code from scratch to solve all of the requirements of the simplified USAR task. In fact part of the purpose of this assigned 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 are 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.

4.2 Crosbot

Crosbot is a robot software suite developed by the CSE AI Lab, largely for use in RoboCup Rescue competitions. It includes software for mapping and navigation. Your group is free to use any of the Crosbot ROS packages. Crosbot is available through the Robolab GitLab as a publicly readable repository. Crosbot can be installed by running this command inside of your `catkin_ws/src` folder:

```
git clone http://robolab.cse.unsw.edu.au:4443/rescue/crosbot.git crosbot
```

There is a branch set up for RSA that you can get by running `git checkout rsa-18s2`. This ignores some of the SLAM mapping packages, allowing you to use a different package described in section 5.2.

You will not be able to push changes to the GitLab repository. However, you are free to modify any Crosbot code. Again, your report should document the changes.

Documentation for Crosbot packages is available on the GitLab wiki:

`http://robolab.cse.unsw.edu.au:4443/rescue/crosbot/wikis/home`

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 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 node that reads the beacons configuration launch file, including the unique id for each beacon, creating a list of `comp3431::Beacon` objects. Run node via:

```
roslaunch comp3431_starter beacons.launch
```

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
```

An updated version for this semester can be retrieved by doing a `git checkout 18s2`. The starter code depends on two Crosbot packages (`crosbot` and `crosbot_msgs`). Therefore to use the starter code you will also need to install and compile Crosbot (see Section 4.2).

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`
- Crosbot explore package (Wall Following Mode).
- Implement your own exploration.

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).
- Crosbot A* planners + Crosbot explore package (Follow Path/Waypoint Mode).
- Implement your own exploration.

Implementing your own waypoint navigation (typically with some sort of A* planning) is quite feasible. You will need to use the Occupancy Grid produced by the common SLAM packages. To do so you will need to take note of the following topics:

- `/cmd_vel` (Type: `geometry_msgs/Twist`) - The topic you should publish to, in order to get the robot move. Only the linear.x and angular.z components can be used by the turtlebots drive system.

- `/map` (Type: `nav_msgs/OccupancyGrid`) - This is the latest version of the map. It will be published every few seconds as the robot moves. (If you use Crosbot, the topic will be `/slam`)

5.5 Vision Processing

Vision processing requires you to identify the beacons in the camera images, and calculating their positions in the world frame of reference. There are two main ways of finding the beacons:

- Subscribing to the RGB and Depth images from the RealSense cam. This requires mapping RGB pixels to Depth pixels, and can directly be used with OpenCV.
- Subscribing to the 3D Point Cloud produced by the RealSense. This already gives pixels in the camera frame of reference but either requires implementing your own colour filtering, or manually generating OpenCV images from the point cloud.

There are two suggested methods for beacon detection:

- Writing your own colour filtering and image processing.
- Using OpenCV image processing, which requires the ROS opencv_bridge package.

5.6 Visualisation Markers

Reporting the location of the beacons is achieved by ROS visualisation markers. The documentation for the visualisation markers ROS packages contains useful tutorials for sending markers. However, you will need to ensure that you set the correct parameters, especially:

- id
- header: frame id
- header: time stamp
- type
- lifetime
- action

5.7 RViz Control Panel (WARNING: UNTESTED)

The starter code provides a RViz plugin that acts as a simple control panel for starting and stopping the robot. These buttons only send messages of type `crosbot_msgs::CrosbotControl`. You will need to write code to subscribe to these messages and respond accordingly. The panel also listens to status messages of type `crosbot_msgs::CrosbotStatus` and displays them in the panel. Again, you will need to write code to send status messages. The starter code gives an example of listening for control messages and sending status messages.

You can also adapt the `cmd_controller.py` from the starter code to create a simple control panel to control from a terminal.

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, Addo, Colm and Peter will be around to provide help, mainly during the lab times each week. We will be around the lab at other times, but many only will be able to give limited help depending on our schedules.