MARYLAND APPLIED GRADUATE ENGINEERING

Z. Kootbally
Fall 2023
UMD
College Park, MD

RWA3 (v1.0)

ENPM809Y: Introductory Robot Programming

Due date: **Wednesday, December 13, 2023, 11:59 pm**

Contents

1 Changelog

- **v1.0** (12/03/2023): Original release of the document.

2 Disclaimer

Certain commercial products or company names are identified in this document to describe examples of robots, batteries, and sensors. Such identification is not intended to imply recommendation or endorsement by the University of Maryland, nor is it intended to imply that the products or names identified are necessarily the best available for the purpose.

3 Conventions

In this document, you will find a number of text styles that distinguish between different kinds of information.

- link *folder* *file*
- Important note.
- List of tasks to do.
- T topic N node M message F frame

4 Prerequisites

- *params.yaml* (Canvas)
- *rwa3_setup.sh* (Canvas)
- You may need code from lecture 11 and lecture 12

5 Objectives

- This is a group assignment.
- This assignment consists of writing a ROS package to move a turtlebot in a maze. To move through the maze, the turtlebot relies on Aruco markers. While the turtlebot navigates the maze, it will need to find and report objects found in the environment.
- Learning outcomes and skills to be developed:
  - Creating a ROS package.

- Writing ROS node(s), publishers, and subscribers.
- Broadcasting and listening of frames.

6 Setup and Installation

- Aruco marker detection requires the latest version of OpenCV.
  - Install `pip3`
    - `sudo apt update`
    - `sudo apt install python3-pip`
    - Check `pip3` is installed with `pip3 --version`
  - Some of you may already have a specific version of OpenCV installed for other projects. However, for this assignment we need the latest version. Check if a version is already installed
    - `python3` (press Enter)
    - `import cv2` (press Enter)
    - `print(cv2.__version__)` (press Enter)
      - Write down this version so you can reinstall this specific version once you are done with this assignment.
  - Upgrade or install OpenCV
    - `pip3 install --upgrade opencv-contrib-python`
    - `pip3 install --upgrade opencv-python`
- To install ROS packages for this assignment, it is recommended to create a new workspace. Use one of the two following solutions.
  1. You can execute the following commands one by one.
     - `cd`
     - `mkdir -p ~/rwa3_ws/src`
     - `cd rwa3_ws`
     - `git clone https://github.com/zeidk/enpm809Y_fall2023.git -b rwa3 src`
     - `rosdep install --from-paths src -y --ignore-src`
     - `colcon build`
  2. You can execute the provided script *rwa3_setup.sh*
     - Download the script.
     - `cd <path to the downloaded script>`
     - `chmod +x rwa3_setup.sh`
     - `./rwa3_setup.sh`
- Create your package for this assignment using your group name as the package name, e.g., `ros2 pkg create group1 ...`
  - This package should have at least the following dependencies (add more later if needed):

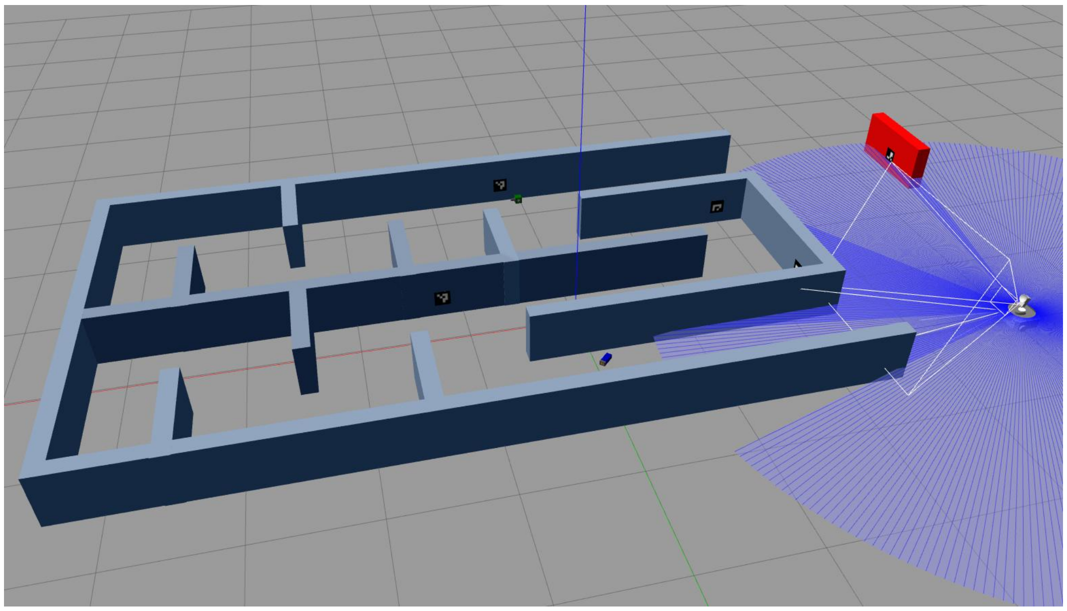

Figure 1: Maze environment for RWA3.

    - ◇ rclcpp, mage_msgs, geometry_msgs, std_msgs, tf2_ros, tf2, tf2_geometry_msgs
  - Create a *config* folder in your package and place *params.yaml* in this folder. This file should be loaded with your node.
- Since we are now using a new workspace, make adjustments in your *.bashrc*|*.zshrc* file.

7 Task Description

This section describes all the tasks you need to complete.

7.1 Start the Simulation

First, start the simulation with `ros2 launch turtlebot3_gazebo maze.launch.py`, which should start a Gazebo environment as seen in Figure 1. This environment consists of a turtlebot with an RGB camera and a logical camera.

- The RGB camera can be used to to detect Aruco markers, which are placed on some of the maze's walls. When an Aruco marker is detected by the camera, its information is published on the topic T aruco_markers. An example of a message published on this topic is provided below. In the example, the marker with id 0 has been detected and the pose of the marker in the RGB camera frame

F camera_rgb_optical_frame is reported. marker_ids is an array.

```
---
header:
  stamp:
    sec: 4
    nanosec: 194000000
  frame_id: camera_rgb_optical_frame
marker_ids:
- 0
poses:
- position:
    x: -0.020704749009379495
    y: -0.05986234372388051
    z: 2.2576233779460475
  orientation:
    x: 0.9952107818269418
    y: 0.0013430607889916505
    z: -0.09693999401516935
    w: -0.012503338892598635
---
```

- The logical camera can be used to detect parts located in the environment. It publishes messages M mage_msgs/msg/AdvancedLogicalCameraImage on T /mage/advanced_logical_camera/image. These parts will be floating in the thin air in the environment. The maze environment has two parts, a blue battery and a green battery. The topic output below shows a detected blue battery in the logical camera frame F logical_camera_link
  - *part_poses* is an array of parts detected at the same time by the logical camera. When a part is detected, this array is filled out, otherwise it is empty. In the snippet, the camera detected a part with color 2 and type 10, which is actually a blue battery if you inspect M mage_msgs/msg/Part The pose of the detected part is in the camera frame F logical_camera_link
  - *sensor_pose* is the pose of the camera in the F odom frame.

```
---
part_poses:
- part:
    color: 2
    type: 10
  pose:
    position:
      x: 1.3969394269660098
      y: -0.07944962174362648
      z: -0.5281643909552133
    orientation:
      x: 0.07398405794903733
      y: 0.17914659008268224
      z: 0.3745241155018947
      w: 0.9067328963584945
sensor_pose:
  position:
    x: -1.492994574620601
    y: 1.0032183441319673
    z: 0.2072354163257374
  orientation:
    x: 0.00020405928306005668
    y: -0.19382234114948693
    z: 0.0009764121233133386
    w: 0.9810361385038302
---
```

7.2 Write the Node(s)

Write a node or multiple nodes (use a name of your choice for your nodes) which can accomplish the following tasks:

1. Start by moving the robot in a straight line and stop the robot until it is at a distance ≤0.4 m from the Aruco marker.
   - Use the topic T cmd_vel and only provide a value for ***linear.x*** (a linear speed of 0.1 should be fine).
   - You can get the pose of the robot with the topic T odom
   - You need to transform the detected Aruco marker in F odom (broadcaster) and then compute the distance between the robot and the marker.
2. Once the robot is at the required distance from the marker, use the appropriate parameter to know what to do next. If the detected marker has the id 0, then check the value of the parameter P aruco_marker_0, if the detected marker has the id 1, then check the value of the parameter P aruco_marker_1, etc.
   - If the value of the parameter is ***right_90***, then rotate the robot 90 degrees

right.
    - If the value of the parameter is ***left_90***, then rotate the robot 90 degrees left.
    - If the value of the parameter is ***end***, then you have reached the last step, you can terminate your program.
    - You can store parameter information as soon as your node starts and use this stored information when needed.
3. Repeat steps 1 and 2 until the robot detects a marker with the parameter value ***end***.
4. Once the robot has detected the marker with the parameter value ***end***, print in the terminal the pose of the detected objects (batteries) in the frame F odom.
    - This requires a subscriber to T mage/advanced_logical_camera/image
    - It also requires a broadcaster and a listener.
    - For the provided environment, we should see in the terminal:
```
Blue battery detected at xyz=[0, 0.926746, 0.25] rpy=[0, 0, 0.785398163397]
Green battery detected at xyz=[0.116440, -1.881520, 0.25] rpy=[0, 0, 1.57]
```
    - You will get numbers which are not exactly the same ones I provided but they should be very close.
    - Each part type and part color combination is unique. That is, if the logical detects a blue or a green battery a second time, it should be ignored.
    - If you attended the last two lectures, you should know that in the code for lecture 12, I provided utility functions to convert quaternions to euler angles.

8 Documentation

All classes, methods, and attributes must be documented using Doxygen. For this assignment, there is no need to generate the HTML documentation. You only need to document your code.

9 Submission

- Place *Readme.txt* in the root folder of your package. In this file, include instructions on how to run your code. You will lose 5% of your grade if the instructions are not correct. With the large number of groups, our TA and grader cannot chase students who provided wrong instructions.
- Zip your package only and submit it. You will lose 5% of your grade if you submit anything else than your package. If you include the packages I have provided in

your submission, `colcon build` will not work because of duplicate packages.

- Make sure you submit the correct package as we will not accept anything else passed the deadline.
- Any submission passed the deadline will incur a penalty, even if it is 1 minute after the deadline. Rules which pertain to penalty can be found in the syllabus. 📝 We were lenient in the past about late submissions but as we are reaching the end of the semester you are required to submit the assignment by the due date.
- There is no extension for this assignment unless you provide documented justifications. If you have other submissions due around the same time, you are responsible for organizing your work to submit the assignment on time.
- A grade of 0 (zero) will be assigned for any plagiarized submission.
- A grade of 0 (zero) will be assigned for hardcoded information. For example, parameters should be retrieved using ROS C++ methods and not hardcoded. If we change the value for some parameters, your code will not work if you hardcode parameter information. In the same way, the color and the type of detected parts have to be retrieved from M mage_msgs/msg/Part (we saw something similar in class with constants in msg files). If you are not sure about what is allowed/not allowed to hardcode, ask us.

10 Grading Rubric

The grading of this assignment consists of 4 sections

- 40% – Your robot can read Aruco markers and can take appropriate actions using parameters.
- 40% – Your robot can find all the objects in the environment and your program displays their poses in the terminal.
- 10% – Your code is properly documented AND commented. Make sure you comment long methods.
- 10% – C++ and ROS best practices are followed. Include guards, naming conventions, etc.