Aalto University
Year 2024

ELEC-E8111 – Autonomous Mobile Robots

Practical Work Report

Date: 21.5.2024

Sai Kiran Pullabhatla – 101475576
Tuomas Kivistö – 770107
Tapani Pärssinen – 715803
Zhibin Yan – 779564
Hsiu-Fang Chien – 101418232

First steps

At first, we configured our terminals by creating a setup.bash file to a location /etc/turtlebotr4 where we added the following lines “/opt/ros/humble/setup.bash”, “export ROS_DOMAIN_ID=0” and “export RMW_IMPLEMENTATION=rmw_fastrtps_cpp”. Then we added the following command “source /etc/turtlebot4/setup.bash” to our “~/.bashrc”. This meant that every time we opened a new terminal it was configured the same way. The first command “lines /opt/ros/humble/setup.bash” configures the ros2 environment we are using and defines that we are using the humble release of ros2. The second command “export ROS_DOMAIN_ID=0” is perhaps the most important as it makes sure that only the ros2 devices with the same id are communicating with each other. This is important because if we have multiple systems in the same network, we could not differentiate which commands are meant for which systems because ros2 does not have a master node. This is also one of the main differences between Ros and Ros2 because Ros has a master node that handles the communication between different nodes. This also means that since there is no master node any message that is not subscribed to by any other node will simply disappear as there is no one to receive it, unless the publisher node decides to store the message itself.

Once we had configured our terminals settings it was time to connect our laptop and the raspberry pi of the turtlebot. This was done by finding the ip of the raspberry pi with the command “ros2 topic echo /ip”. Then we used the Ip address received to establish a ssh connection with the command “ssh ubuntu@<the ip we received>. After this was done, we went to a create3 web page with our raspberry pi ip and the 8080 port where we connected our turtlebot4 to the Wi-Fi network. After this was done, we applied the command “ros2 run teleop_twist_keyboard teleop_twist_keyboard” which allowed us to control the turtlebot4 with the keyboard of our laptop that was connected to the same Wi-Fi.

1) RViz2 application

In this project we used Rviz2 for visualization of the robot, the robot’s environment and as a tool for navigating as you are able to point where the robot should go and then Rviz2 will visualize the path that the robot will take. Rviz2 is also able to show the point clouds generated by the 2D-lidar sensor that is on turtlebot4. Rviz2 could also show the images taken from the turtlebot4’s, OAK-D stereo camera. You can also access the data from the camera and the lidar from the nodes /scan this is for the 2D-lidar data and /color/preview/image for the camera data.

2) SLAM FROM REAL DATA

To create a mapping from real data we first used the command “ros2 launch turtlebot4_navigation slam.launch.py” to start the SLAM algorithm which stands for simultaneous localization and mapping, and this is exactly what the algorithm is used for. Then we ran the command “ros2 launch turtlebot4_viz view_robot.launch.py” to visualize the results from SLAM.

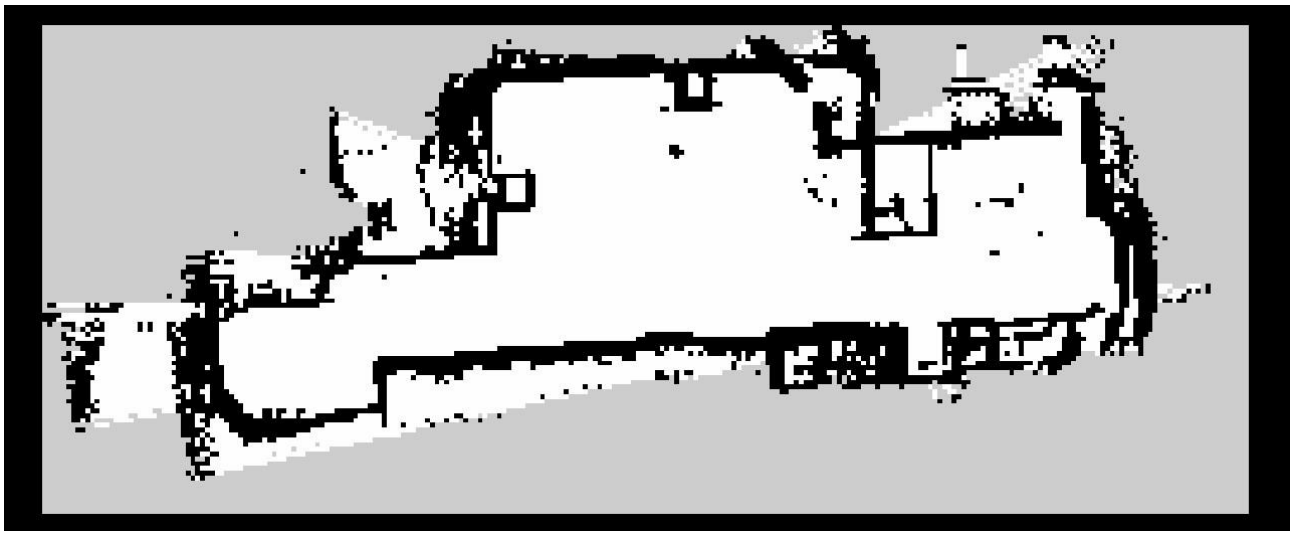
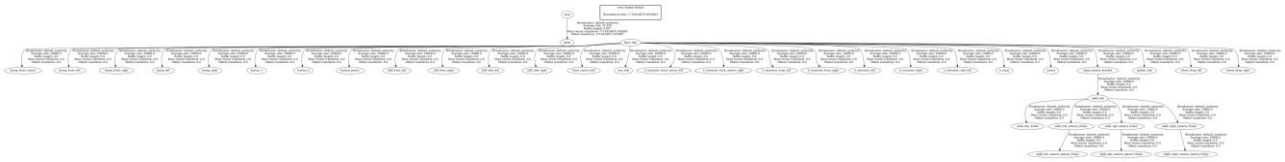


Figure 1: Here is the mapping we got from running the slam algorithm. This is a map of a room in an apartment building. The mapping is a 2d map of the layout of the room, it is basically a cross section of the room at the height of the lidar sensor of the turtlebot4.

Figure 2: Explanation of what is in the mapping.

1) coffee table, 2), small night table, 3) bed (the robot can go under it so it is not really visible) 4) desk (again not really visible as the robot can go under it) 5) sofa 6) tv table

Figure 3: the tf tree, it will be submitted as a separate file as it might be difficult to read from this picture.



3) Ros 2 bag

In this part we first collected data to a Ros bag by driving the robot around in a room, and then ran the slam algorithm with the data that we managed to collect. The data we collected were:

/map / pose /tf_static /tf /scan /odom /color/preview/image /imu. These were chosen because map is the map that we are interested in pose is important for localization of the robot. tf and tf_static were chosen as these include the transformations from the base of the robot to its sensors and the mapping seemed to work better when these topics were included, scan includes the lidar data, odom includes the odometry data, color/preview/image includes the camera data which is important for mapping and imu was chosen because it is important for the localization. The main drawbacks of using ros bag would be the high data and computing requirements if we record many different topics at the same time to gain the best possible results.

Figure 4: The rosbag information.

```
course-coordinator@AMR:~$ ros2 bag info bag

Files:             bag_0.db3
Bag size:          25.8 MiB
Storage id:        sqlite3
Duration:          184.597s
Start:             May  9 2024 21:06:34.364 (1715277994.364)
End:               May  9 2024 21:09:38.962 (1715278178.962)
Messages:          33705
Topic information: Topic: /odom | Type: nav_msgs/msg/Odometry | Count: 3327 | Serialization Format: cdr
                   Topic: /imu | Type: sensor_msgs/msg/Imu | Count: 16451 | Serialization Format: cdr
                   Topic: /tf_static | Type: tf2_msgs/msg/TFMessage | Count: 4 | Serialization Format: cdr
                   Topic: /scan | Type: sensor_msgs/msg/LaserScan | Count: 1370 | Serialization Format: cdr
                   Topic: /tf | Type: tf2_msgs/msg/TFMessage | Count: 11612 | Serialization Format: cdr
                   Topic: /pose | Type: geometry_msgs/msg/PoseWithCovarianceStamped | Count: 611 | Serialization Format: cdr
                   Topic: /map | Type: nav_msgs/msg/OccupancyGrid | Count: 330 | Serialization Format: cdr
```

4) SLAM FROM DATASET

Here is the map that we got from running the slam algorithm based on the topics that we described in the above section. The area is the same as the one in the real SLAM algorithm. As we can see it is not nearly as accurate as the one that we got from the live slam. But this is to be expected as we did not record all the possible features. Though we also have to consider that we did not take as long of a recording as we did when we ran the live slam, so that might also affect the result.

Figure 5: the mapping from rosbag.

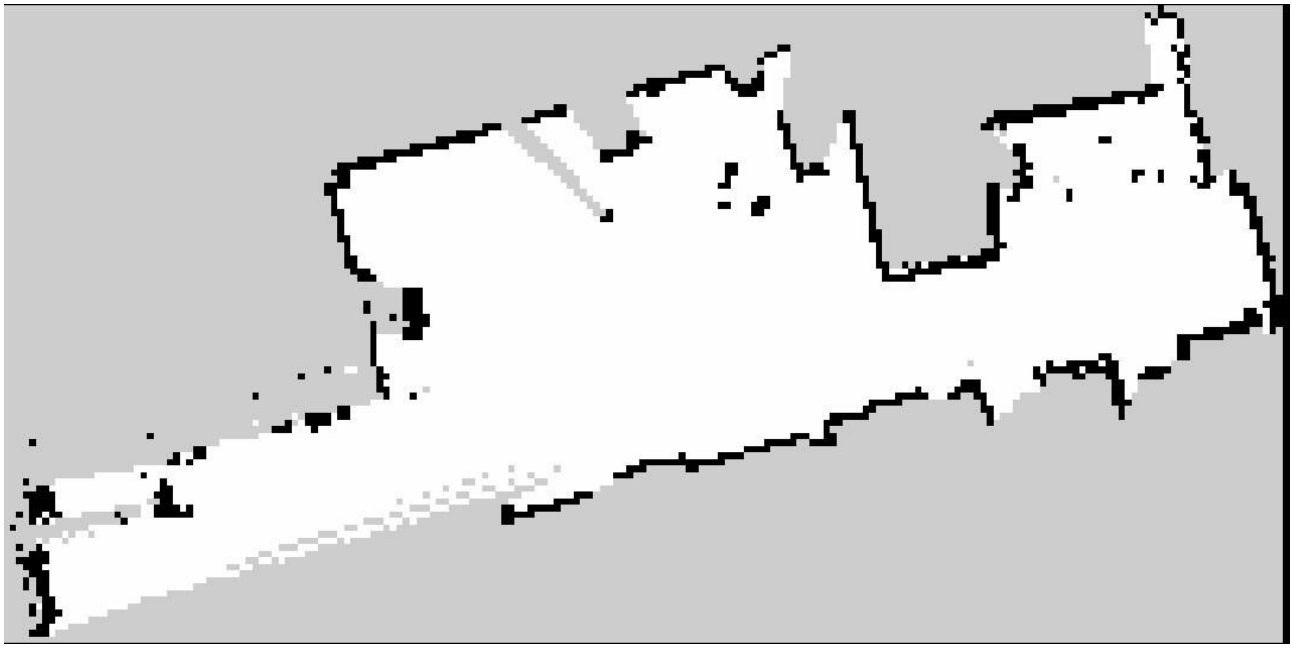
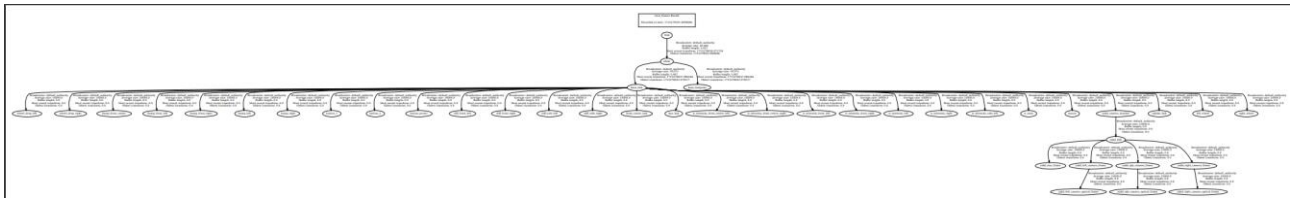


Figure 6: Tf tree obtained from the rosbag

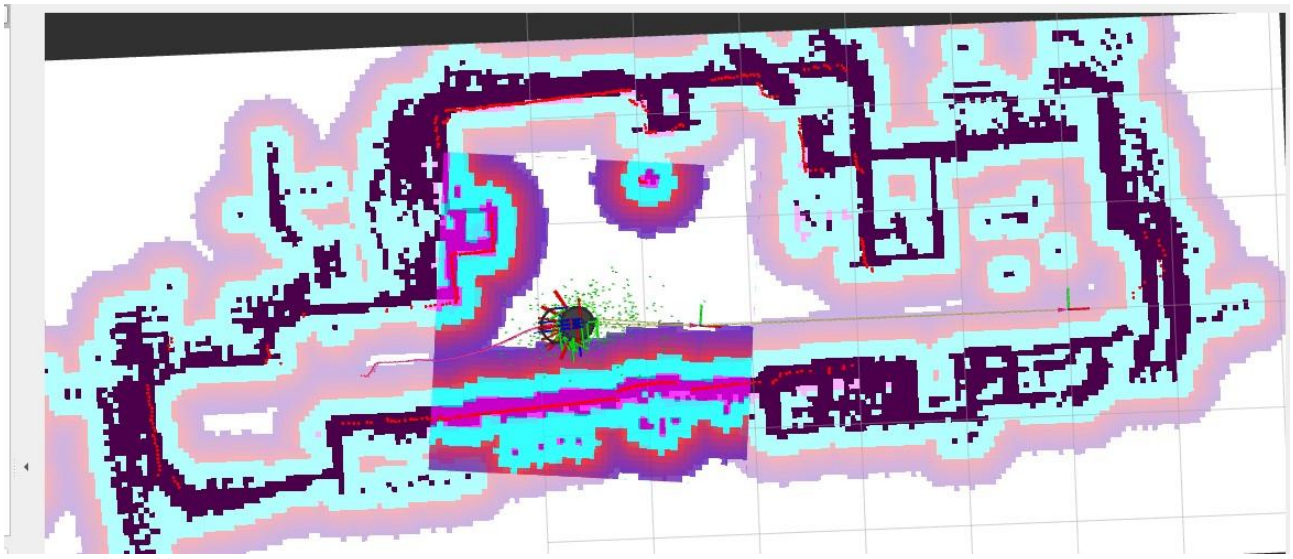


Surprisingly it seems that the Rosbag's tf-tree is more detailed than the tf-tree obtained from the real slam. I would have assumed that the real slam would have generated more detailed tf-tree because it has access to more data. But as there are more links between the nodes and more nodes 38 vs 41 in the rosbag tf-tree it is more detailed. This might be because the data that is being recorded is processed more, to get the most out of it.

5) Navigation

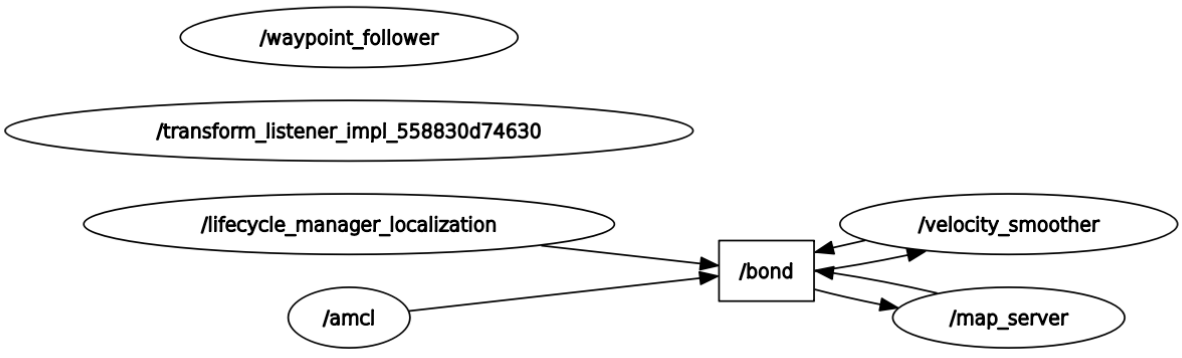
To navigate the robot we first ran the command "$ ros2 launch turtlebot4_navigation localization.launch.py map:=map_name.yaml". This command starts the localization of our robot within the specified map. The launch file specifically specifies what map we are using and some other parameters that are needed for the localization. Then we ran the command ros2 launch turtlebot4_navigation nav2.launch.py, is the software that runs the navigation, and then finally we ran the "$ ros2 launch turtlebot4_viz view_robot.launch.py" command to visualize what is happening.

figure 7: Here we can see that the robot is navigating towards the left.



Our robot is controlled by varying the rotational speed between the two wheels of the TurtleBot thus it is a differential drive system which is non-holonomic. The transformation frames are important because they keep track of all the transformations between the robot's base and its sensors, these are essential for the robot's localization and mapping. The configuration space of the turtlebot4 includes all the parameters of the robot that define its pose and position so for our differential drive robot this would include its x and y coordinates and the angle in which it is heading θ. TurtleBot4 is also capable of dodging objectives when it is navigating. Turtlebot4 dos this by changing its navigation path to go around the object when possible. On the other hand, when the goal is put inside an object the turtlebot4 will move around the object trying to find a way to the goal point but finally will abort this goal as it proves to be impossible.

Figure 8: the rqt graph of our navigating system.



The figure consists of bond, map server, velocity smoother, lifecycle manager localization, transform listener impl, amcl and waypoint follower. Let's explain what the purpose of each of these is. Bond is a higher-level planner that handles the connections between the nodes and their interactions. Map server offers access to the map that the robot has created to other nodes. Velocity smoother makes sure that the movements of our robot are smooth. Lifecycle manager localization makes sure that the nodes relating to localization are handled properly (being killed when no longer need or activating new nodes when the need arises). Transform listener impl provides the different transformation information that other nodes might need. Amcl is a 2D probabilistic localization system, which uses a particle filter to test many different hypotheses for the robot's location based on its history and the current sensor readings. Waypoint follower handles the navigation through waypoints.