Turtlebot4

Catalina Murray

December 17, 2022

1 Background

The Turtlebot4 Standard [1] and Turtlebot4 Lite [1] are mobile robots with a few main components. First their base is the create 3, built inside of it is the UI (user interface board) which houses the Raspberry Pi [1].The Raspberry Pi and create 3 are the two main computers of the robot. The two main sensors of the robot are the LiDar [1](RPLIDAR A1M8 is a 360 degree Laser Range Scanner) and the Oak D camera [1], the standard has the pro camera and the lite has the Oak-D-lite camera. For more information about the Turtlebot4 see Turtlebot4 User Guide

2 Getting Started

The Turtlebot4 includes two main components the Raspberry Pi and the Create3. Both of these components need to be connected to the same WiFi as your PC that is running ROS2 and Ubuntu 20.04. The Create 3 is responsible for sending topics, actions, and services over WiFi, for example it can send information like battery state, sensor data, and docking actions. The Raspberry Pi is responsible for running Turtlebot4 ROS nodes and sensor ROS nodes. The PC requires Ubuntu 20.04 and ROS2 Galactic. Figure [2] shows how all of these computers interact with one another.

2.1 Connecting to WiFi

Software: Make sure you have installed ROS2 galactic on your PC with Ubuntu 20.04. The following steps are only required when first connecting the robot to WiFi or when wanting to change the robots WiFi connection.

Raspberry Pi: The first step is to connect the Raspberry Pi to WiFi.

Step 1: Power on the robot by placing it on the charging dock and waiting of the robot to make a chime.

Step 2: ROS2 Galactic's default middleware is CycloneDDs, to configure this so the user can see the

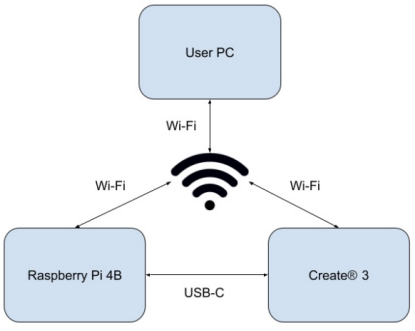

Figure 2: Diagram of WiFi connections between the User PC, Raspberry Pi and Create 3.

Figure 1: Turtlebot4 Standard (left) and Turtlebot4 Lite (right) Diagram, including camera, LiDAR, Raspberry Pi, and UI (user interface board). The Lidar and camera are present on both the standard and lite models. The UI board is only on the standard.

robot topic properly first download the xml file by calling this function in your terminal:

```
wget https://raw.githubusercontent.com/turtlebot
/turtlebot4_setup/galactic/conf/cyclonedds_pc.xml
```

Step 2.1: find the available network interface using "ip link" command.

Step 2.2: open the xml file and add the following line below:

```
<DontRoute>true</DontRoute>,
<NetworkInterfaceAddress>"wifi interface name"</NetworkInterfaceAddress>
```

Step 2.3: We want to make sure configure the CycloneDDs each time a new terminal is opened, in order to do this we need to add it to the /.bashrc file. Add this line of code to the bashrc file.

```
export CYCLONEDDS_URI=/etc/cyclonedds_pc.xml
```

Step 3: With the Raspberry Pi in AP mode connect to the Turtlebot4 WiFi, once connected ssh into the Raspberry pi.

Step 4: configure the Raspberry pi's WiFi settings with

```
sudo wifi.sh -s <WIFI_SSID> -p <WIFI_PASSWORD> -r <REGULATORY\_DOMAIN> && sudo reboot
```

2.2 Connect 3

Now that the Raspberry Pi is connected to WiFi we must also connect the connect 3.

Step 1: Press the two buttons surrounding the home button at the same time, the light ring will flash, wait until it turns blue. This puts the connect 3 in AP mode so you can configure its WiFi.

Step 2: connect to the Connect 3 WiFi and once connected in a browser go to 192.168.10.1

Step 3: on the page select the connect tab enter WiFi information, once connected the Turtlebot will play a chime.

Step 4: check to make sure the create 3 is publishing topics by running "ros2 topic list"

3 Driving the Turtlebot

Once the Turtlebot is powered on and connected to WiFi you can begin to drive it.

Step 1: install teleop twist so you can teleop the robot.

```
"sudo apt install ros-galactic-teleop-twist-keyboard"
```

Step 2: you need to source the setup file for Ros2 galactic in order to use Ros2. You need to do this each time you open a new terminal or you can add this command to the bashrc file so that it is automatically sourced each time you open a new terminal.

```
"source /opt/ros/galactic/setup.bash"
```

Step 3: run the teleop twist node.

```
"ros2 run teleop_twist_keyboard teleop_twist_keyboard"
```

Alternatively, you can use some of Create 3's ROS2 Actions which include "DriveDistance", "DriveArc", and "RotateAngle" . All of these actions make it so you can command the robot to go a specified distance at a specific speed.

4 Navigation

4.1 Mapping: SLAM

In order to create a 2D map of the turtle bots surroundings you can utilize SLAM which stands for Simultaneous Localization and Mapping. This is a method of algorithms developed to help an autonomous vehicle map out unknown environments. There are two main components involved in SLAM, firstly the front-end processing or sensor dependent processing. This first step utilizes cameras or Li-DAR to acquire information about the environment. The second component is the back-end processing which is the pose-graph optimization and is independent of sensors. To run SLAM first launch the Turtlebot4 Naigation package and the slam sync node . For example, the command line would look like:

```
ros2 launch turtlebot4_navigation slam_sync.launch.py
```

Alternatively you could also run the asynchronous SLAM as well however the results will be lower resolution.

4.2 Mapping: Rviz

Next we can use Rviz, which is a 3D visualization tool for ROS. This will help us see the map we are creating. To use Rviz, launch the Turtlebot4 visualization package and the corresponding node the command line will look like this,

```
ros2 launch turtlebot4_viz view_robot.launch.py
```

4.3 Mapping: viewing the Map

You can then use one of the previous driving methods to drive the robot around the area you want to map. You can visualise the area the robot is mapping by simultaneously looking at the RViz. To Save the map you can use the following command, and replace "map name" with your own name for the map.

```
ros2 service call /slam_toolbox/save_map slam_toolbox/srv/SaveMap "name:
  data: 'map_name'"
```

This will create a ".pgm" file so you can view the saved map as well as a ".yaml" file. The ".yaml" file can be used later for other navigation purposes such as creating a path for the robot to follow.

4.4 Create a Path

4.4.1 Localization

Localization is an alternative method to SLAM to figure out where the robot is on a map. Localization requires that a map already exists.

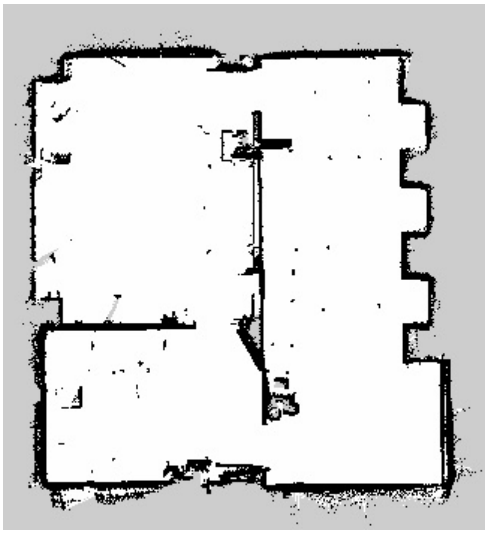

Figure 3: Example of a map created using SLAM and RViz, this map shows the 3 rooms in the dry lab.

4.4.2 Nav2

Nav2 is a navigation stack for ROS2. This allows us to see where the robot is with the generated map and allows us to interact with it, for example we can find various poses of points in the map or we can set a goal for the robot to drive to. More information on Nav2 can be found here:

Step 1: To use Nav2 first launch the turtlebot4 navigation package and the nav bringup node.We also want to make sure we turn on localization and provide our .yaml file of our map. For example:

```
"ros2 launch turtlebot4_navigation

nav_bringup.launch.py

slam:=off localization:=true map:='map_name.yaml'"
```

Step 2: Next you want to launch Rviz so you can view the map.

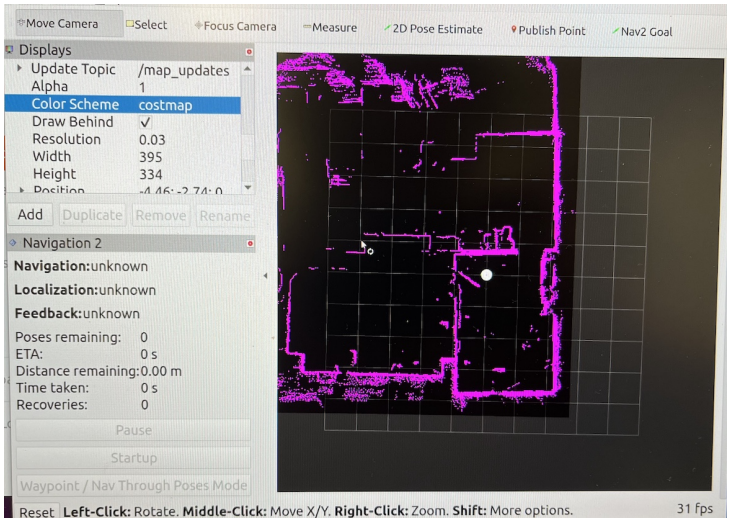

Figure 4: Example of Map with Turtlebot location using localization. This shows what RViz will look like when you launch it. It includes some of the commands you will be able to use like publish point as well as nav to goal.

4.4.3 Nav2: publish a point

To publish a point and find the specific pose of a point on a map you can publish point tool.

Step 1: call the topic echo and node clicked point for example call:

```
ros2 topic echo /clicked_point
```

Step 2: To see the point you then have to click on the publish point icon in Rviz and then click on a point in the map, you will see the x,y,and z coordinated of that point pop up on the terminal.

4.4.4 Nav2: create a path

You can you some of the other functions of Rviz to create a path or a goal for the robot, for example you can use the waypoints function to have to robot follow through a number of points, or you can you Nav2 Goal to have the robot reach a certain point.

Figure 5: This shows the turtlebot4's location in the room using localization. As you can see the turtlebot for is highlighted inside the map that we provided to localization.

5 ROS2

ROS stands for Robot Operating System and is a set of software libraries and tools that is highly utilized in robotics. It is important to note that you need to source ROS each time you open a new terminal or add the source line to the bashrc file to it is automatically sourced each time you open a new file.

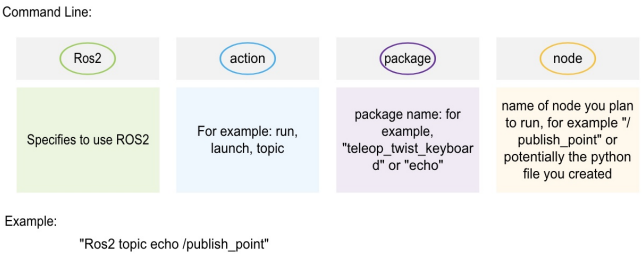

Figure 6: Diagram of ROS2 command line

5.1 colcon

colcon is utilized to build packages in ROS2, to install:

```
sudo apt install python3-colcon-common-extensions
```

you can add the line "colcon-argcomplete.bash" to the bashrc file if you want the colcon command to be able to auto complete when you don't know the full name of a backage or node.

5.2 ROS2: creating a package

Step 1: Create a workspace using "mkdir"

Step 1.2: Create an src folder inside the workspace

Step 1.3: use colcon build to build the folders "install", "log" "build" inside the workspace

Step 1.4: source the setup.bash folder

Step 2: Create a package to organize the nodes, inside the src folder use "ros2 pkg create 'package name'"

Step 3: open the package in something like VS code.

5.3 Ros2 create nodes

Once VS code is launched there will be a few main files, first the package.xml file which will have information like the description, name, etc, this is important if you plan to publish this code on github. The second is the setup.cfg file which helps ROS to know where to set up the nodes, next is the file with your package name, this is where you create all the nodes.

6 Autonomous Mapping

6.1 overview

There are 4 main components to create a map using the Nav2 stack. The first one is localization which as mentioned previously using a previous map to determine the robots coordinates within that map. The next is SLAM which will be utilized to create the new map. The third is a python script that allows the robot to move autonomously throughout the room/rooms that are being mapped. The fourth is the launch file that allows you to save the ".yaml" and ".pgm"

```python
#main_launch

from math import ldexp
from launch import LaunchDescription
from launch_ros.actions import Node
from ament_index_python.packages import get_package_share_directory
from launch.actions import DeclareLaunchArgument
from launch.substitutions import LaunchConfiguration, PathJoinSubstitution
from launch.actions import (DeclareLaunchArgument, GroupAction,
                            IncludeLaunchDescription, SetEnvironmentVariable)
from launch.launch_description_sources import PythonLaunchDescriptionSource


def generate_launch_description():

    create_path = GroupAction([

        IncludeLaunchDescription(
            PythonLaunchDescriptionSource('/opt/ros/galactic/share/turtlebot4_navigation/launch/nav_bringup.launch.py'),
            launch_arguments={'map': '/home/catalina/Turtlebot4_Maps/dry_lab3.yaml',}.items(),),

         IncludeLaunchDescription(
            PythonLaunchDescriptionSource('/opt/ros/galactic/share/turtlebot4_navigation/launch/slam_sync.launch.py'),
            launch_arguments={}.items(),),

         Node(
            package='turtlebot4_python_tutorials',
            executable='follow_waypoints',
            name='path',
            output='screen',
            parameters=[],)

    ])

    ld = LaunchDescription()

    ld.add_action(create_path)

    return ld
```

Figure 7: Code for the current launch file which successfully calls all three files and runs the first two. However it does not successfully run the third.

file to your machine. Three of these files are launch files and one of them is a python script. You can run all of these files simultaneously using a "launch.py" file. In this launch file you would first call the launch file to start the localization, you would then call the launch file to start slam and finally you would start the python file that has your robot move throughout the rooms. Once the robot moves through all the rooms you can call the launch file that saves the map to your machine. You can do this using a conditional statement like the "IfConition" command. My current launch file calls the first two launch files and the python script "follow waypoints." However only the localization and slam run, it seems the system is overloaded trying to run all three at one time and I believe I would need some conditional statement or timer to ensure the python script runs alongside the two launch files.

6.2 launch files

As you can see from the figure above I created a launch.py file, you can also create launch files as .xml files. The first function you have to call in a lunch file is to generate the launch description.

```
def generate_launch_description():
```

This is where you can call launch files or python packages. To call other launch files you can use the command "IncludeLaunchDescription" and to call python files you can create a Node. From the figure you can see I called two launch files and then I created a Node for the python script.

6.3 current status

As previously mentioned, I was not able to successfully call all three files in the one launch file. The problem may stem from the fact that you can not run the python script and SLAM simultaneously. SLAM updates the location and seems to override the previous input locations.

One of the main parts of the overall function is having the robot move through the three rooms using the python script and localization. This was done successfully using one of the launch files and a python script.

Step 1: The first thing you need to do is to start the localization. To do this you will call the launch file "nav bringup" located in the Turtlebot4 navigation folder. It should be on your computer in the location "/opt/ros/galactic/share/turtlebot4navigation". To do this you can use the command.

```
ros2 launch turtlebot4_navigation nav_bringup.launch.py
```

This launch file has a few different parameters, one of them is 'map' which requires the location of the .yaml file for the map you want to use. The next is slam which has options sync, async, and off, which lets you turn on asynchronous SLAM, synchronous SLAM or to turn SLAM off. In this case we want to turn slam off or else the python script wont run correctly. The next parameter is localization which has option true and false, and since we ant to use localization in this instance we should set this to true. The full command would be as shown below:

```
"ros2 launch turtlebot4_navigation nav_bringup.launch.py

slam:=off localization:=true map:='your map location .yaml' "
```

Step 2: call the python script "follow waypoints." To do this you will need to clone the file from the Turtlebot github page. You should also use the colcon build function to build the python files as well as source to ensure ROS knows where the file is. Once you have built the python files you can open them with visual studio code to visualize what is going on. First the script makes sure localization is running, this is why it is important to call navbringup before this step. Next the script looks to see if it is docked, if it is not it will try to dock itself. That way the coordinates [0,0] will be at the docking station. Once it is docked it will undock itself and it will look for its first goal pose which is the first coordinate you input. It will follow the following goal poses until it reaches the end and tries to re dock itself. This is why it is important to make the last goal pose near the docking station.

step 2.1 To create the goal poses the script will use you can utilize RViz the visualization tool to see the map that localization is running. Once RViz is pulled up, you can also subscribe to the topic "/clicked point" to see coordinates on the map. You can use this code to launch the RViz file.

```
ros2 launch turtlebot4_viz view_robot.launch.py
```

You can use this code to subscribe to the topic clicked point to see the coordinates you want the robot to move through.

```
ros2 topic echo /clicked_point
```

You will the click "Publish point" on RViz and dray the red circle to a place on the map you want to see the coordinates. The coordinates will then pop up in the terminal. Once you have all your coordinates you should input them into the follow waypoints code for each follow waypoints command. For example in the script the code will look something like this:

```
"goal_pose.append(navigator.getPoseStamped([0.0,0.0],

Turtlebot4Directions.WEST))"
```

You'll notice there are also directions at the end of each line of code, this direction corresponds to the direction you want your robot to be facing once it reaches that goal pose. North

```python

import rclpy
from turtlebot4_navigation.turtlebot4_navigator import TurtleBot4Directions, TurtleBot4Navigator

def main():
    rclpy.init()
    navigator = TurtleBot4Navigator()
    # Start on dock
    if not navigator.getDockedStatus():
        navigator.info('Docking before intialising pose')
        navigator.dock()

    # Set initial pose
    initial_pose = navigator.getPoseStamped([0.0, 0.0], TurtleBot4Directions.NORTH)
    navigator.setInitialPose(initial_pose)

    # Wait for Nav2
    navigator.waitUntilNav2Active()

    # Set goal poses
    goal_pose = []
    goal_pose.append(navigator.getPoseStamped([-1.05, -0.706], TurtleBot4Directions.EAST))
    goal_pose.append(navigator.getPoseStamped([-0.02, -1.19], TurtleBot4Directions.NORTH))
    goal_pose.append(navigator.getPoseStamped([2.21, -1.44], TurtleBot4Directions.WEST))
    goal_pose.append(navigator.getPoseStamped([-1.05,-0.706],TurtleBot4Directions.NORTH))
    # Undock
    navigator.undock()

    # Follow Waypoints
    navigator.startFollowWaypoints(goal_pose)

    # Finished navigating, dock
    navigator.dock()

    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Figure 8: Diagram of Follow Waypoints code showing where you input coordinates, and the docking and undocking routine. This code shows an example of 4 goal poses

corresponds to the direction the robot is facing at the docking station. The code for follow waypoints can be seen below.

6.4 Next Steps

The next steps for this project would be to find a way to have the SLAM routine run while the follow waypoints and localization scripts are running. This may require rewriting the follow waypoints script or find a different way for the turtlebot to navigate through the three rooms. Following this routine It would also be beneficial to start 3D mapping. This includes generating a point cloud. It may also require attaching a camera to the top of the turtlebot4 so that you can get visualization of more objects that are above the turtlebots line of sight currently.