Project 2 - Collaborative Robots

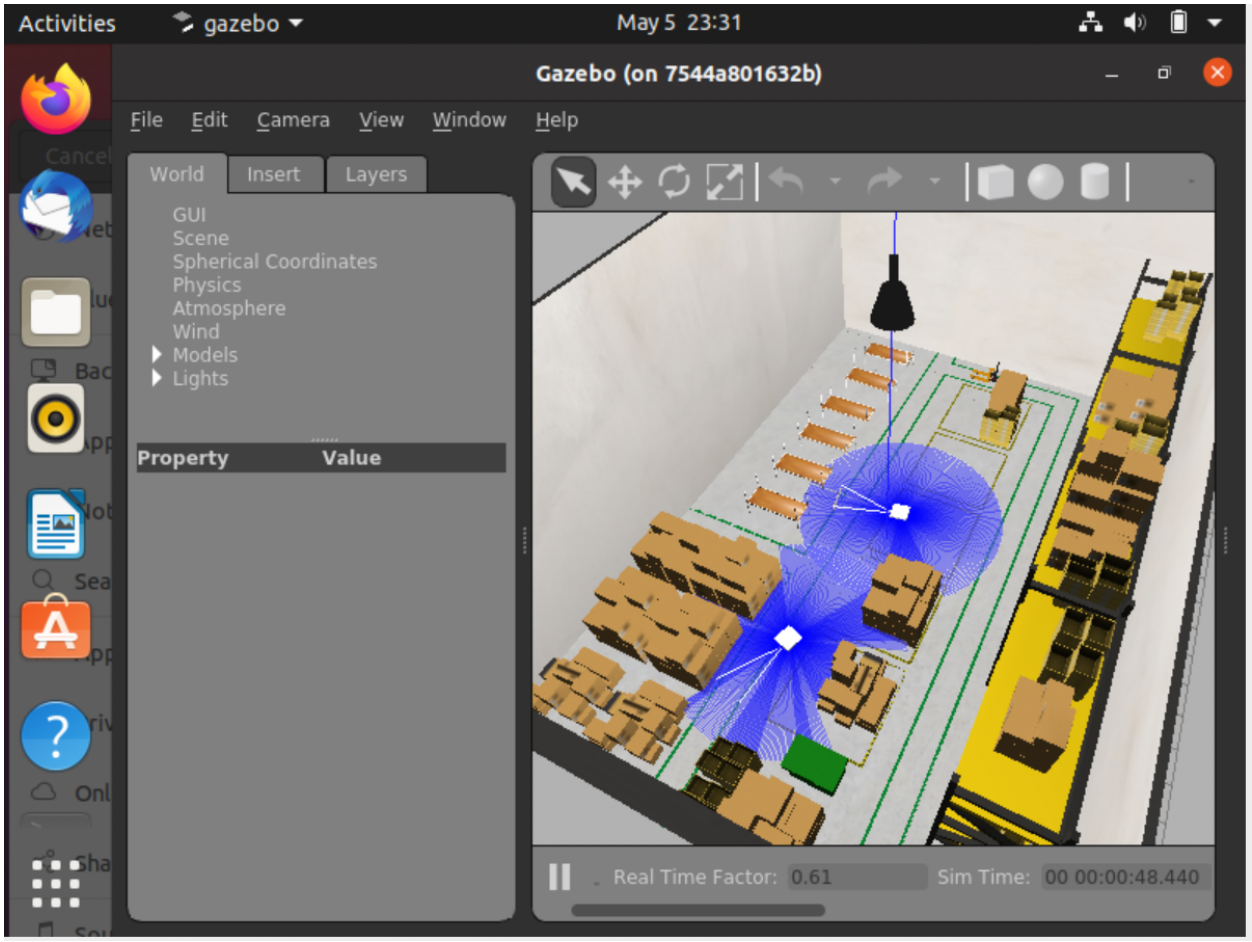
Sean Hornbuckle
Satyen Singh
Riyaz Ahmed:

Github link: https://github.com/ssingh1997/Collab_Robotics

Task 1:

Setup Environment

After pulling the docker container from the tutorial and sourcing the workspace, we could successfully pull up the two Rviz windows as well as the Gazebo window that shows the robots in the warehouse. One could reach here either by simply pulling the docker container or building one from the source. After attempting to build one from source, we ran into many different issues such as incorrect sourcing of files, storage issues, incorrect drivers, etc. Therefore, we



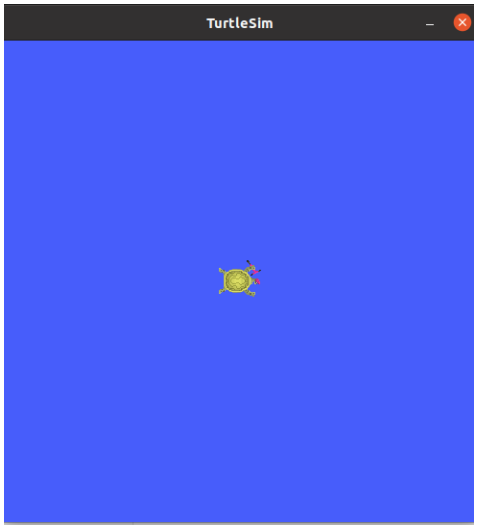
used the docker container that provided.

Understanding ROS 2 Foxy

Before actually getting around to using ROS 2, we decided to read the documentation and go through the tutorials in the ROS 2 documentation to get a better understanding of what it actually is.

To understand ROS 2 Foxy, we decided to run through the documentation and tutorials to get a feel for the software. After going through the installation and configuration of our ROS 2 workspace, we took a stab at the Turtlesim tutorial.

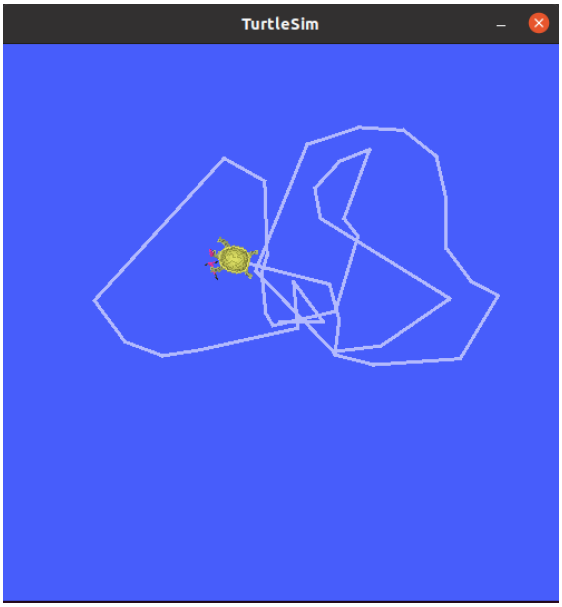
Turtlesim Tutorial part 1: Spawning and Moving a Single Turtle Robot



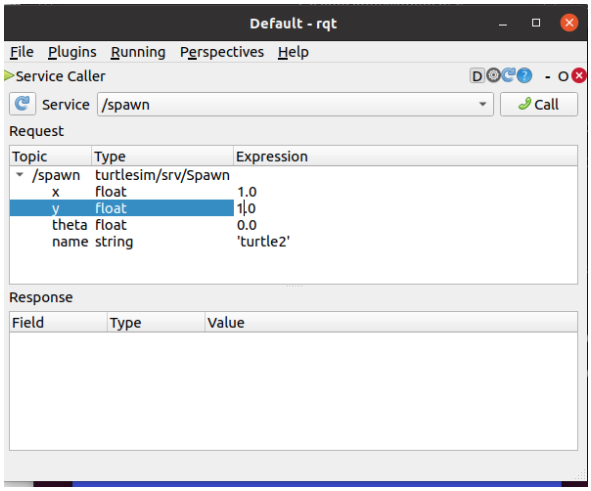
We initialize our turtlesim window as you can see in this window. A turtle is spawned in the center of its world. Turtle spawns at the given position below.

```
Spawning turtle [turtle1] at x=[5.544445], y=[5.544445],
```

In a new terminal, you can create a new node which you can use to control the movement of the turtle by using the up and down arrow keys to move and left and right to change the direction of the turtle.

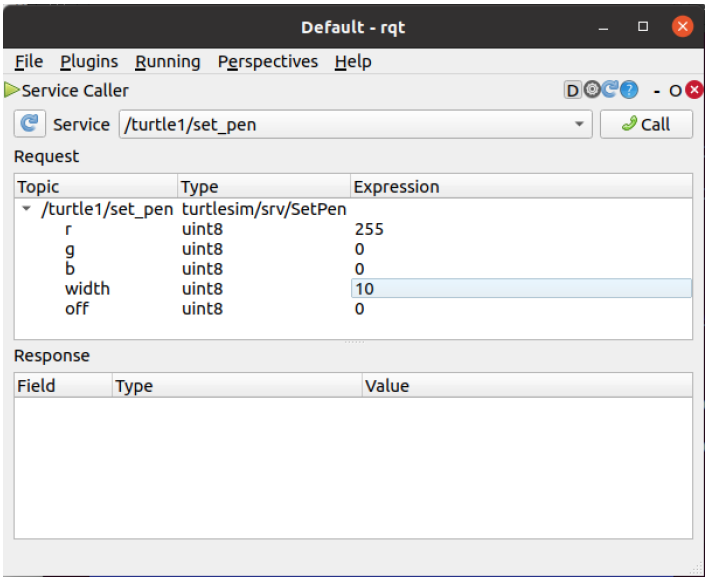


Turtlesim Tutorial Part 2: Spawning and Moving Two Turtles



Using rqt, which is the Graphical User Interface ROS 2, we can spawn another by setting its position, which we set as (1.0, 1.0) and name as turtle2

As you can see from this image, we have spawned both turtles successfully. Now we need a way to differentiate the movement of turtle1 and turtle2 in order to track both of their movements.

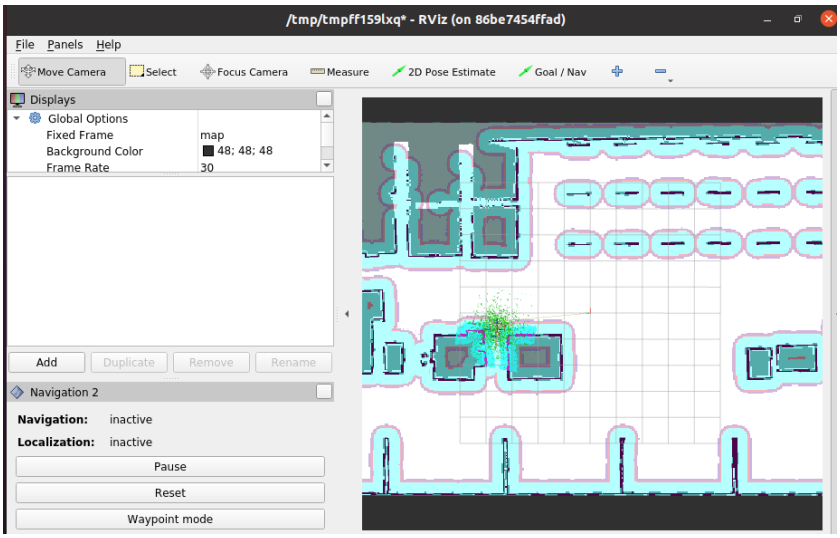


Using rqt, we can set the color and thickness of the pen that tracks the movements of each the turtle. Here we are giving red a value of 255 which will make it a red color and a thickness of 10 to clearly see the movement.

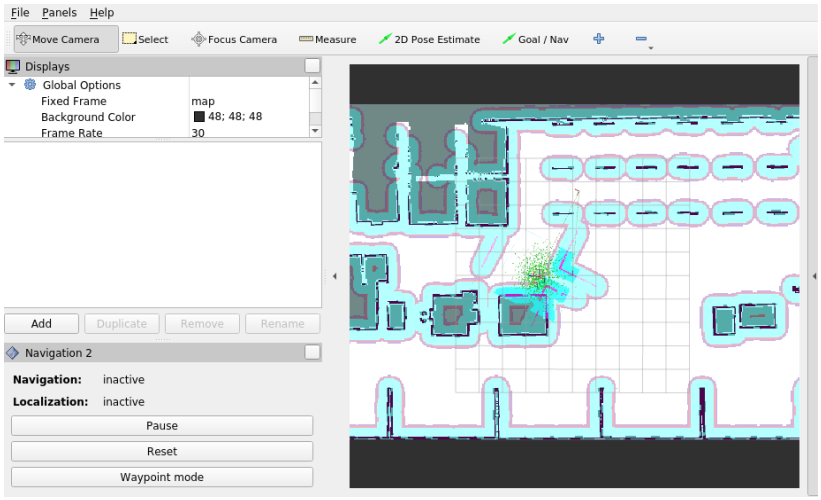
Problems faced in Setup and Configuration of Environment

As mentioned earlier, we faced numerous problems with the entire overall setup and configuration of our environment. To begin with, we needed quite some space and after much experimentation, we realized that we needed to dedicate at least 40 GB of space on our hard drive and 4 processors to be able to run ROS 2 on our machines. Another problem we faced was the inability to run ROS 2 natively. Because of this, the speed and performance were severely compromised as there were 3 layers of abstraction. ROS 2 is sort of an OS in itself which was running in the Docker container, which was running on our virtual machine, which ultimately was running on our native operating system. We would like to have been able to figure out how to run ROS without a VM as we believe it would've helped speed up the process and make the experience much smoother.

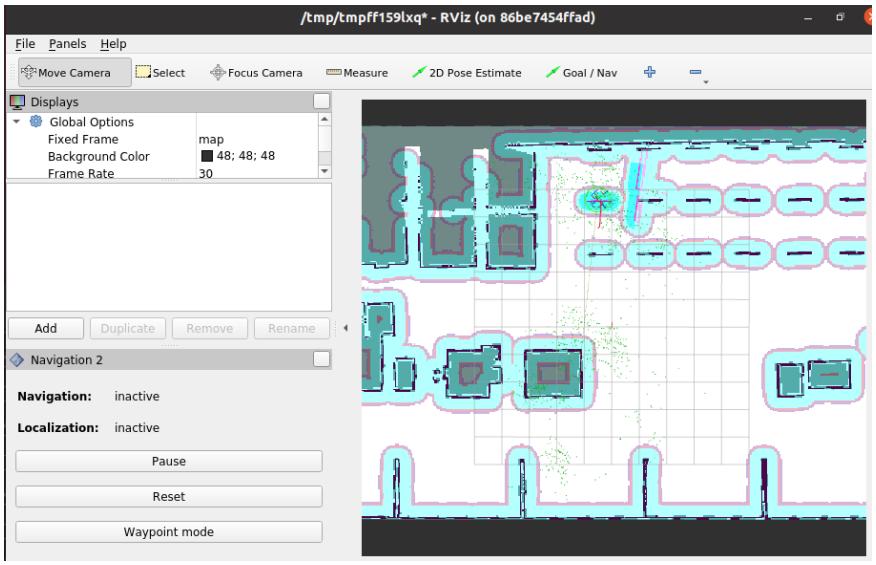
Task 2: Before writing our own planner, we decided to go through the multi-robot simulation that was shown in the tutorial



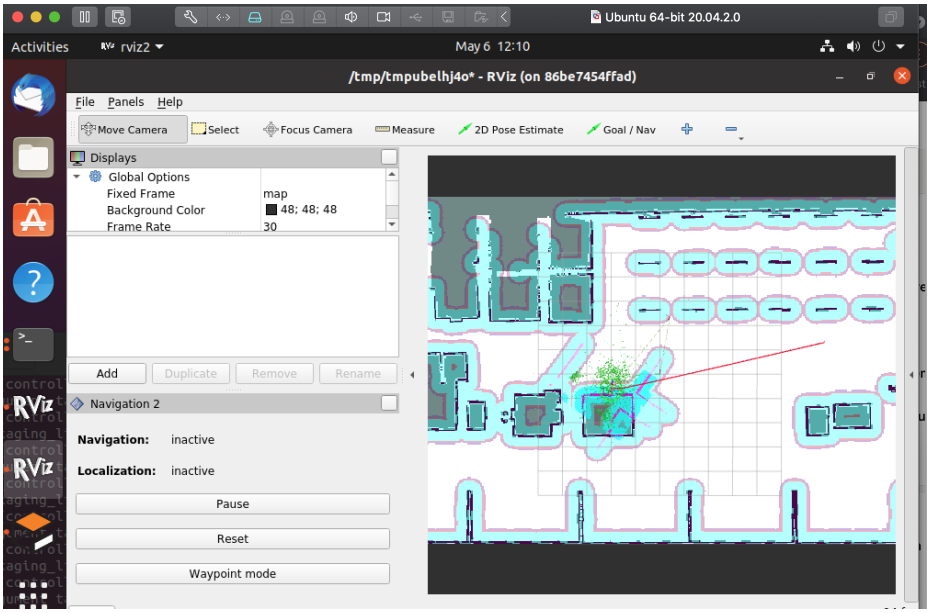
As you can see from this concentration of green, we initialized the position of robot1 between the two squares and gave it an upwards trajectory. We played around with a bunch of different configurations to see which starting position may be the most optimal.



Similarly to Robot1, we also initialized and played around with the starting point of Robot2.



Robot1 was able to move around quite a bit around the warehouse. It was able to carry the palletes successfully around the warehouse.



Unfortunately, Robot2 had some trouble carrying out the palettes. However, we tried different starting points in order for it to work. It was more of a trial-and-error experiment to see which starting point would be the best.

Problems faced with Task 2:

Unfortunately, the setup for this project took so much of our time that we did not really have a working environment until very late into the project. Because of this, we did not get the chance to get an in-depth understanding of the planner as well as some other tools like Plansys2 or Nav2. However, we intend to keep looking into the details more so as we found this project to be very interesting and it was a satisfying feeling to just be able to test out the default planner for the multi-robots and see them both in action in the Gazebo window. We also reached out to the author of the multi-robot project, Shreyas Gokhale, to gain a deeper insight. He suggested us to

attempt the single-robot exercise before attempting the multi-robot one as it is not as complex. He also pointed out that in order to implement Plansys2, we would need to directly re-structure some of the source code so that is a challenge that we intend to tackle beyond the scope of this class.

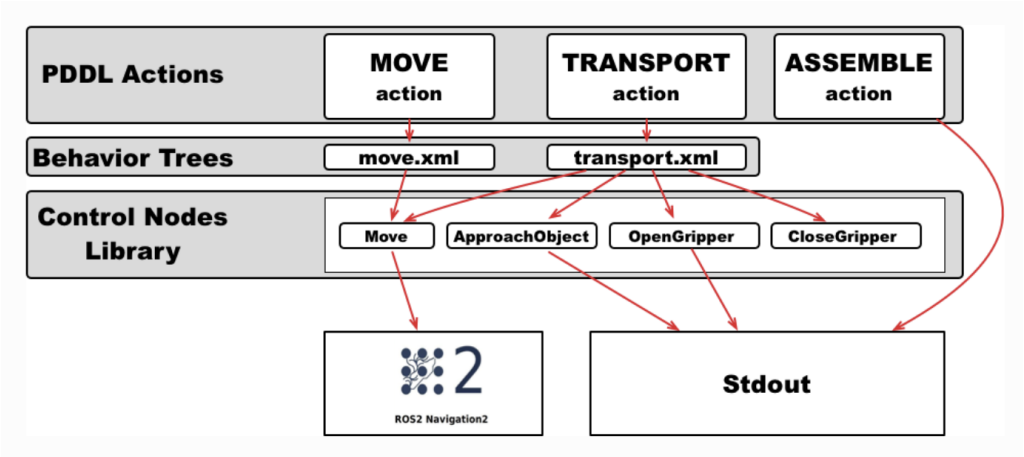
Task 3:

Use the symbolic planning language PDDL to write a planning controller that integrates Plansys2 and Nav2. See the Dock-Worker Robots Planning Domain in this post as well as the PlanSys2 tutorials.

The controller may be triggered by an event i.e. customer arriving in the pickup hatch location.

This section of the lab required us to use PDDL (Planning Domain Definition Language) to write a planning controller. This language is typically used for planning tasks, which is why it is useful for this exercise. There are (5) main components of a PPDL task that need to be defined. They are the Objects (items in the world that we are concerned with), Predicates (properties of the objects), Initial State (the state we start in), Goal Specification (things we want to achieve and be true), Actions/Operators (how we can change the state of the world). We then also have domain files (for predicates and actions) and problem files (for objects, initial state, and goals).

Plansys2 (ROS2 Planning System) is a project that was created to give robotics developers a PDDL based planning system - it is implemented in ROS2. Plansys2 implements the use of behavioural trees. What this does is split the whole operation of the robots into smaller behaviours in a specific structure that will follow rules that we establish. This is the same case for Navigation2, which uses behavioural trees. Each action that is called up for our robots to do is a separate node, that communicates with the behavioural tree over a ROS action server.



Behavior trees are an effective way of creating complex, modular systems. Behavior trees are a hierarchical structure. This means that larger trees can be composed of smaller trees as sub-branches. This allows the programmer to effectively piece together tree modules that have already been created to form new larger trees. This also allows for the code to be more readable. Overall, the modularity and the reusability of tree components make behavioral trees great for the application of robotics where the overall motion that we need to control is very complex.

Issues faced with task 3: While we were able to set up and run the robot simulation, none of the group members had powerful machines with graphics cards that could handle the simulations well through a hypervisor such as virtualBox or VMWare. This made it very difficult to test and run the simulations fully.

Task 4:

Write a markdown report where you explain your design of the planner in Task 2 and 3 as well as providing positive and negative results from the robotic delivery of the pallets. Explain the negative results and suggest ways that can be fixed if you have no time fixing these issues.

Between task 2 and task 3, the biggest difference is the type of planner that was used. In task 2, we used a planner that was written in python and in task 3 we used (or attempted to us) PDDL, while integrating Plansys2 and Nav2. In the 3rd task, the ability of the planning controller to use Plansys2 and Nav2 is advantageous since it allowed us to leverage behavioural trees.

In the future, one of the biggest improvements that we can look to make is the use of a computer with a dedicated graphics card. When running the hypervisor on a Macbook Pro, there was a lot of lag on our machines due to the lack of computer resources, which made testing difficult.

As mentioned earlier, we needed to look a bit deeper into how exactly the planner is structured in order to even make any changes to it so that the robots do not collide with each other. Given more time, we would have liked to go through the single robot simulation as that would allow us to understand how just a single robot interacts with its environment before attempting two robots operating simultaneously. Last, we could keep optimizing our initial state for each robot to the point that they are operating smoothly within the environment and can carry the palettes without getting stuck like we had in a few test runs.

Although this project was difficult with our current background knowledge and machine limitations, it was a great introduction to programming robotics. Going forward, we are going to continue to look into other projects that can be done in this space to build our portfolios and learn more about the software. Ideally we will be able to have access or build a more powerful machine at that time, which can also be used for other deep learning projects as well.

Web Sources:

https://jderobot.github.io/RoboticsAcademy/exercises/MobileRobots/single_robot_amazon_warehouse/

https://intelligentroboticslab.gsyc.urjc.es/ros2_planning_system.github.io/tutorials/docs/bt_actions.html

https://navigation.ros.org/