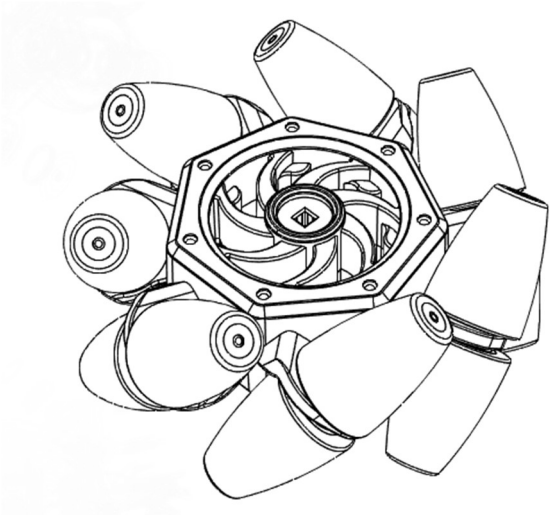
Honours Year Project

2021-22

Autonomous exploration and obstacle avoidance with the Mecanum wheel robot



DEPARTMENT OF
COMPUTER SCIENCE

University of Liverpool
Liverpool L69 3BX

Abstract

This article describes the use of the Mecanum wheel robot for autonomous exploration in an unknown environment in the Webots simulator environment. The program is based on the implementation of the Robot Operating System (ROS2). The proposed program was experimentally validated on the Summit XL Steel mobile robot in the virtual environment of Webots. It is equipped with a 3D LIDAR, a 9-axis IMU and GPS.

Taking into account the difference between outdoor and indoor environments or the presence or absence of GPS, two SLAM algorithms are provided. Gmapping is based on the PBRF particle filtering algorithm, and Cartographer is based on GraphSLAM graphical optimisation algorithm. The program also contains a navigation module and an exploration module for the autonomous exploration of the robot. The user can objectively and quickly compare the effects of different SLAM algorithms. In this article, the functional and algorithmic implementation of the different modules and the relationships between them are described in detail.

Contents

1. Introduction & Background 1
1.1 Aims & Objectives 1
1.1.1 Aims 1
1.2.1 Objectives 1
2. Design 2
2.1 The components of the system 2
2.2 The data structures and algorithms used by the system 3
2.3 The design of the user interface 6
2.4 The flow of navigation 7
3. Implementation (Realisation) 8
3.1 Webots 8
3.2 ROS2 11
4. Testing & Evaluation 15
5. Conclusion 15
6. BCS Project Criteria & Self-Reflection 17
References 18
Appendices 19

1. Introduction & Background

Robots are widely used in industry, reducing labour costs and risks and creating more value and opportunities for people. Robots will become a part of people's lives in the foreseeable future. Therefore, the study of robot motion control is an important topic. This project aims to use robots with Mecanum wheels in a simulator to explore and avoid obstacles in a virtual environment autonomously and robots can explore unknown environments on their own. Mecanum wheels, due to their structural characteristics, allow the robot to move omnidirectionally in a very flexible way [1]. It represents a challenge for robot motion control, unlike conventional vehicle structures.

The project will cover several research areas: simulators, robot motion control, environmental perception, simultaneous localisation and mapping (SLAM), and navigation planning.

1.1 Aim & Objectives

1.1.1 Aim

The aim is to design a complete robotic system. In the simulator, a robot is created with a Mecanum wheel chassis with a camera + LIDAR sensor to perform the essential Simultaneous localisation and mapping (SLAM) function. In other words, the robot can depart from an unknown location in an unknown environment, locate its position and posture and build a map using map features observed by the sensors. This enables the robot to navigate and map in a variety of scenarios. Users can build on this robot with new features such as face recognition, environmental sensor collection and more, depending on their needs.

1.1.2 Objectives

- Build a robot model and test the environment scenario. The robot is given a suitable location to place sensors such as LIDAR and cameras. Physical variables and parameters are introduced into Webots to build a virtual environment more similar to the real one.
- Write a set of motion control codes for the Mecanum wheel based on this robot to give the robot basic motion functionality.
- A variety of sensors on the robot are used to perform basic SLAM functions, including localisation and map building.
- The robot completes autonomous navigation and path planning. Ability to explore unknown environments.
- The overall software system decouples multiple modules, making it easy for users to replace different algorithms in the simulator for testing.

2. Design

2.1 The components of the system

The project is made up of two parts in terms of software. And the research will be based on two pieces of software later on.

- **Webots**: Webots is an open-source and multi-platform desktop application used to simulate robots [2].
- **ROS2**: The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications [3].

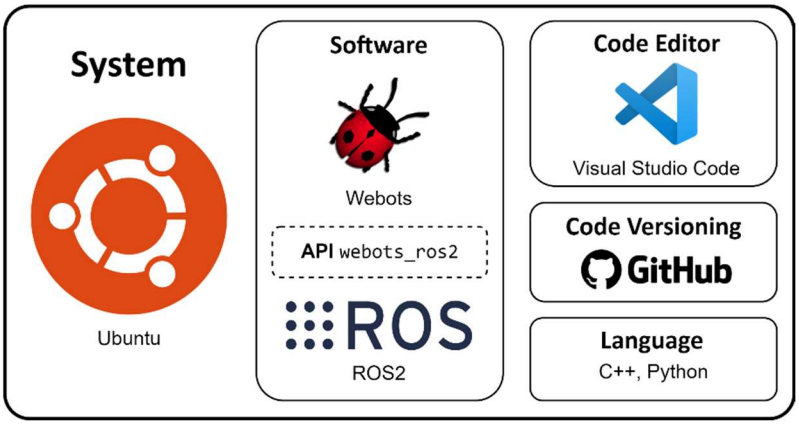
Webots core is based on a combination of modern graphical user interface (Qt), physics engine (ODE fork) and OpenGL 3.3 rendering engine (wren) to create a wide variety of simulation and interactive design environments [2]. It is perfect simulation software. Also, ROS has many valuable tools, mainly the 3D visualization tool (RVIZ), which can be used to observe the state of the robot and view sensor data. Cyberbotics Ltd. provides Webots ROS2 Interface on Github [4].

In this project, Webots are used to complete modelling and simulation of robot and scene, and ROS2 is used to perform SLAM functions, navigation and control the robot.

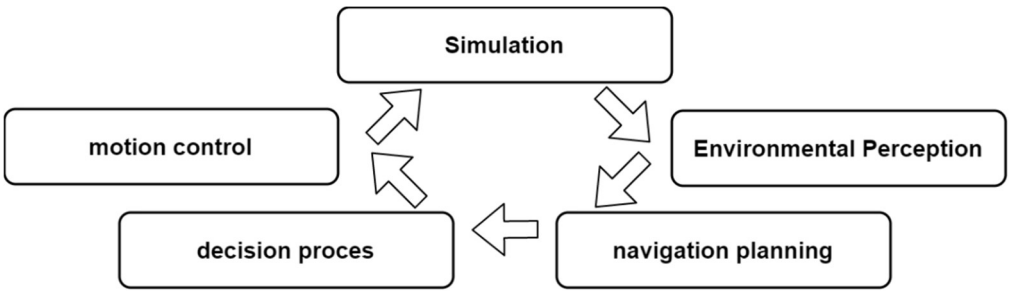
Both Webots and ROS2 can run on Windows, Linux, and macOS in terms of operating systems. However, ROS2 is not well supported on windows. So, the development platform is Ubuntu (a Linux distribution with desktop applications). Here are the version numbers of the software I am using.

Development environment:
- Ubuntu 20.04 LTS
- Webots R2022a
- ROS2 FOXY
- Visual Studio Code

Implementation language:
- C++ (Version: 9.4.0)
- Python 3.8

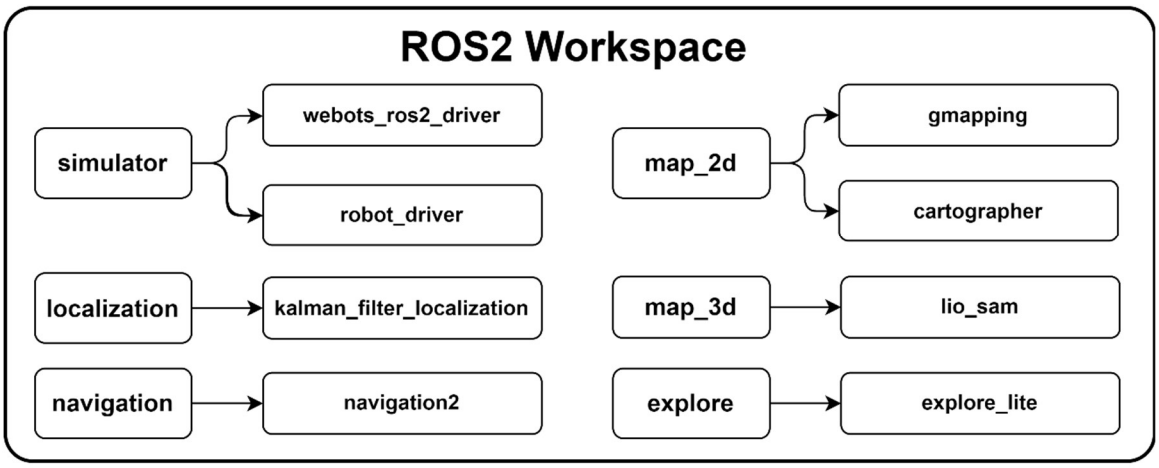


The software framework for this project is divided into five main components which relate to each other as shown in the following diagram.



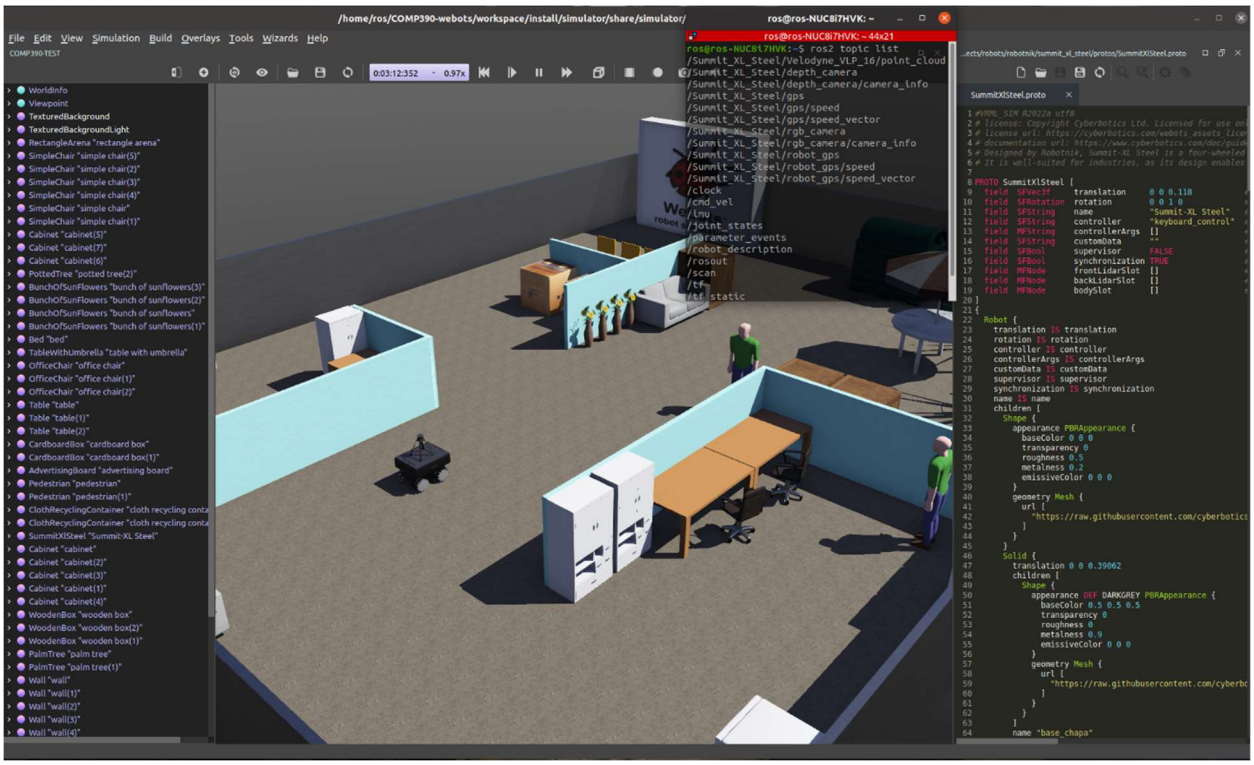
2.2 The data structures and algorithms used by the system

I have created six workspaces in the code: **simulator**, **localization**,**navigation**, **explore, map_2d and map_3d**. As shown in the picture below.



The **simulator** folder is used to open the Webots. It is divided into two main sections: `webots_ros2_driver` and the `robot_driver`.

- `webots_ros2_driver`: This subpackage automatically creates a ROS2 interface for most sensors launched from the `launch` file.



- `robot_driver`: Chassis motion code control section. The algorithm for each wheel speed is based on the "Inverse kinematic derivation [5]". Used to accept the speed command `cmd_vel` and return the speed of each wheel.

```python
WHEEL_RADIUS = 0.123
LX = 0.2045 #lateral distance from robot's COM to wheel [m].
LY =0.2225  #longitudinal distance from robot's COM to wheel [m].

# Accept speed commandSpeed command [linear.x, linear.y, angular.z]
vx = Twist.linear.x
vy = Twist.linear.y
wz = Twist.angular.z

# Return wheel speeds [front_left, front_right, back_left, back_right]
def mecanumControl(vx, vy, wz):
    fl = 1 / WHEEL_RADIUS * (vx - vy - ((LY + LY) * wz) )
    fr = 1 / WHEEL_RADIUS * (vx + vy - ((LY + LY) * wz) )
    bl = 1 / WHEEL_RADIUS * (vx + vy + ((LY + LY) * wz) )
    br = 1 / WHEEL_RADIUS * (vx - vy + ((LY + LY) * wz) )
    return fl, fr, bl, br
```

According to equations (10) and (11) for *Forward* and *Inverse* kinematics there is:

$$\begin{bmatrix}\omega_1\\ \omega_2\\ \omega_3\\ \omega_4\end{bmatrix} = \frac{1}{r}\begin{bmatrix}1 & -1 & -(l_x+l_y)\\ 1 & 1 & (l_x+l_y)\\ 1 & 1 & -(l_x+l_y)\\ 1 & -1 & (l_x+l_y)\end{bmatrix}\begin{bmatrix}v_x\\ v_y\\ \omega_z\end{bmatrix}. \qquad \text{eq. 19}$$

$$\begin{cases}\omega_1 = \frac{1}{r}(v_x - v_y - (l_x+l_y)\omega),\\ \omega_2 = \frac{1}{r}(v_x + v_y + (l_x+l_y)\omega),\\ \omega_3 = \frac{1}{r}(v_x + v_y - (l_x+l_y)\omega),\\ \omega_4 = \frac{1}{r}(v_x - v_y + (l_x+l_y)\omega).\end{cases} \qquad \text{eq. 20}$$

And

$$\begin{bmatrix}v_x\\ v_y\\ \omega_z\end{bmatrix} = \frac{r}{4}\begin{bmatrix}1 & 1 & 1 & 1\\ -1 & 1 & 1 & -1\\ -\frac{1}{(l_x+l_y)} & \frac{1}{(l_x+l_y)} & -\frac{1}{(l_x+l_y)} & \frac{1}{(l_x+l_y)}\end{bmatrix}\begin{bmatrix}\omega_1\\ \omega_2\\ \omega_3\\ \omega_4\end{bmatrix}$$

The **localization** uses `kalman_filter_localization`, which has the `ekf_localization_node` from `robot_localization`, implementing the extended Kalman filter algorithm based on quaternions. Uses an omnidirectional motion model to predict states in time and corrects predicted estimates using sensed sensor data [6].

$$\hat{x}_k = f(x_{k-1}).$$

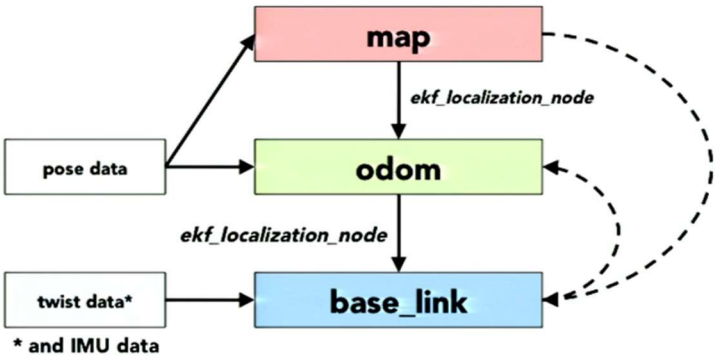
$$\hat{P}_k = FP_{k-1}F^T + Q.$$

$$K = \hat{P}_k H^T \left(H\hat{P}_k H^T + R\right)^{-1}.$$

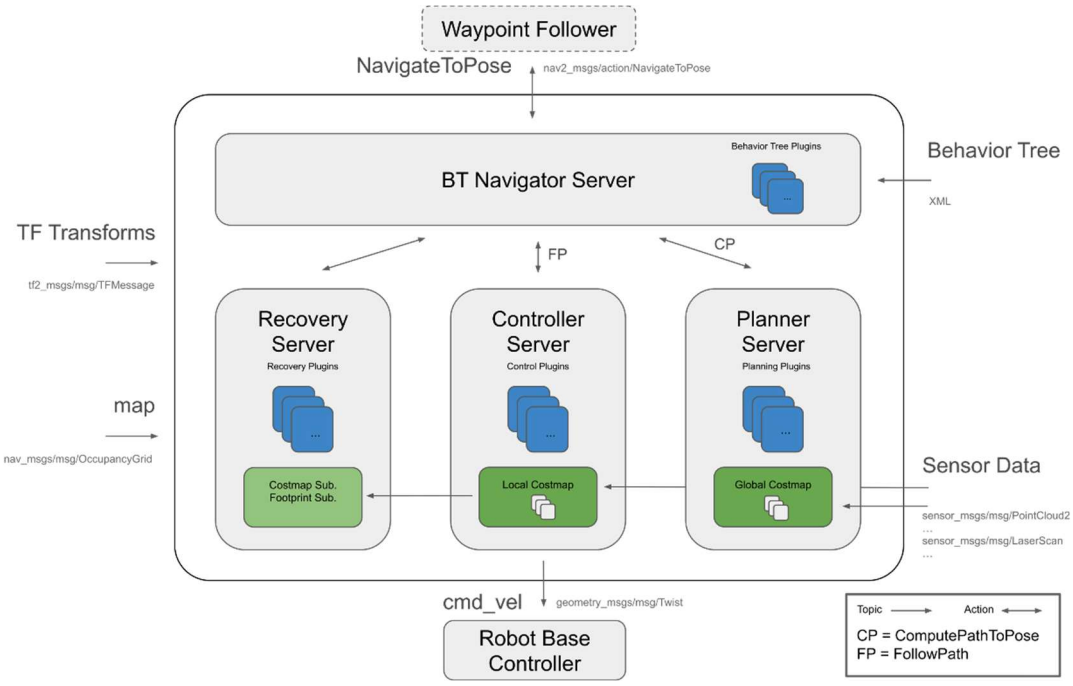
$$x_k = \hat{x}_k + K(z - H\hat{x}_k).$$

$$P_k = (I - KH)\hat{P}_k(I - KH)^T + KRK^T.$$

P: `initial_estimate_covariance`
Q: `process_noise_covariance`
R: `nav_msgs/Odometry, sensor_msg/Imu`



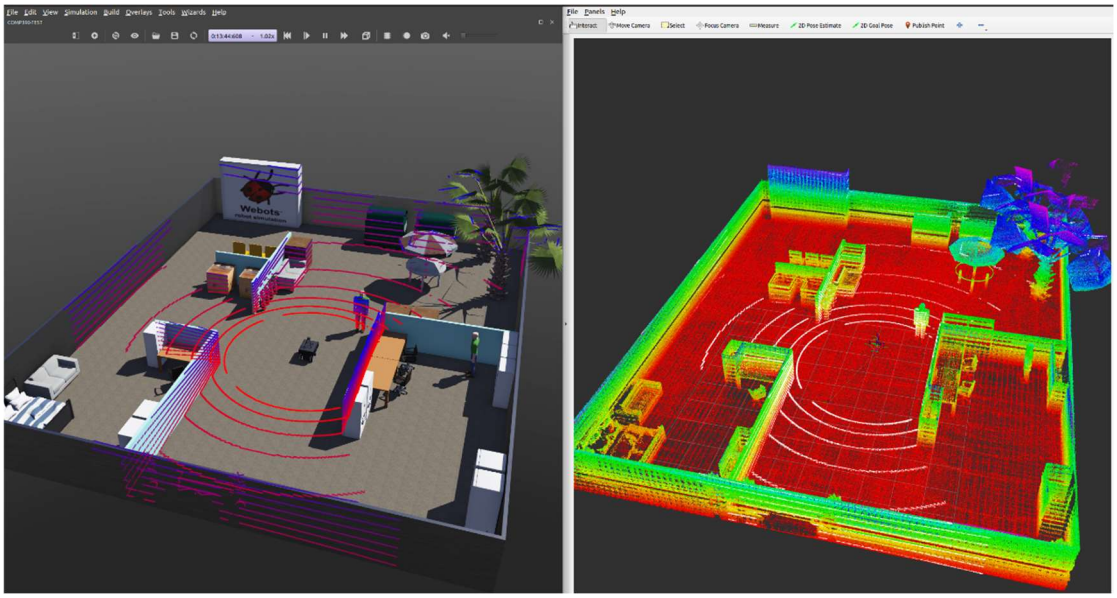
The **navigation** used is navigation2 (referred to as Nav2). This navigation node uses maps and path planning algorithms to navigate and contains a behaviour tree. It is used in the project for obstacle avoidance and navigation [7].



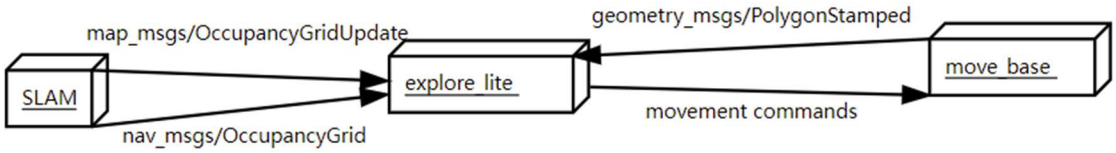
The **map_2d** contains `Gmapping` and `Cartographer`, both of which are nodes for map generation. Allowing the use of sensors such as LiDAR and IMU to generate maps. `Gmapping` is based on the idea of PBRF particle filtering algorithm and requires high machine positioning. Cartographer is a graph-based optimised slam algorithm, building maps based on the idea of GraphSLAM, with low requirements for machine positioning. In this project, `Gmapping` and `Cartographer` are used for comparison and use [8], [9].

Gmapping **Cartographer**

The **map_3d** is `lio_sam`. Factor map optimization scheme based on LiDAR, IMU and GPS multiple sensors, and the use of frame-local maps instead of frame-global maps in frame-map matching. map_3d is used in this project to generate maps for 3D point cloud maps [10].



The **explore** is `explore_lite`. This package provides greedy frontier-based exploration. The robot will greedily explore its environment when the node is running until no frontiers can be found [11]. The top level of `move_base` is replaced with the nav2.



2.3 The design of the user interface

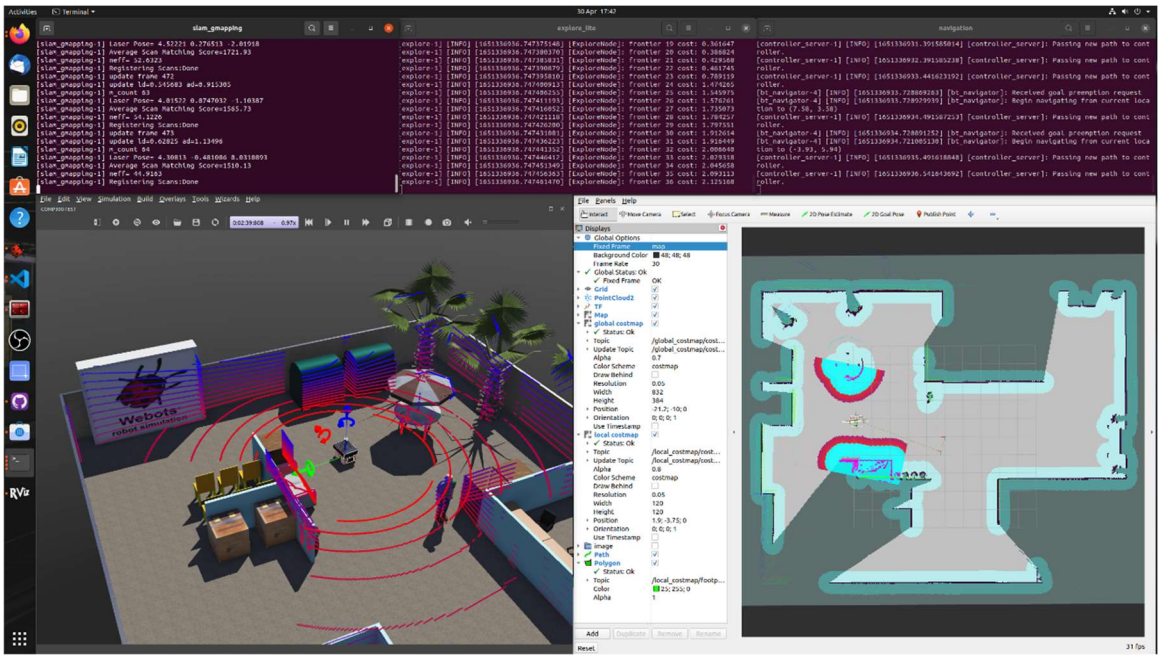
The project's main aim is to investigate the autonomous driving of robots in a simulated environment, and user interface design is not the focus of this project. The project is run on the Command-Line Interface (CLI). Webots and RVIZ graphical user interfaces (GUI) will be provided after executing the CLI. "RVIZ shows you what the robot thinks is happening, while Webots shows you what is really happening [12]."

A variety of scripts are provided in the project to quickly and easily run the project demo, or the user can open a terminal and type in the commands manually to achieve the desired effect.

After completing the compilation and environment variable configuration, the user can run the following commands separately in multiple terminals.

```
# Launching simulator
ros2 launch simulator robot.launch.py

# Launch 2D map
########## Choose one of the following ##########
# (1) Gmapping (outdoor with GPS)
ros2 launch slam_gmapping slam_gmapping.launch.py

# (2) Cartographer (indoor without GPS)
ros2 launch cartographer cartographer.launch.py
#################################################

# Launch navigation and exploration
ros2 launch navigation nav2.launch.py
ros2 launch explore_lite explore.launch.py

# Launch 3D map (Optional)
ros2 launch lio_sam run.launch.py

# Robot keyboard control (Optional)
ros2 run simulator keyboard
```

GUI and CLI demo screenshots:



2.4 The flow of navigation

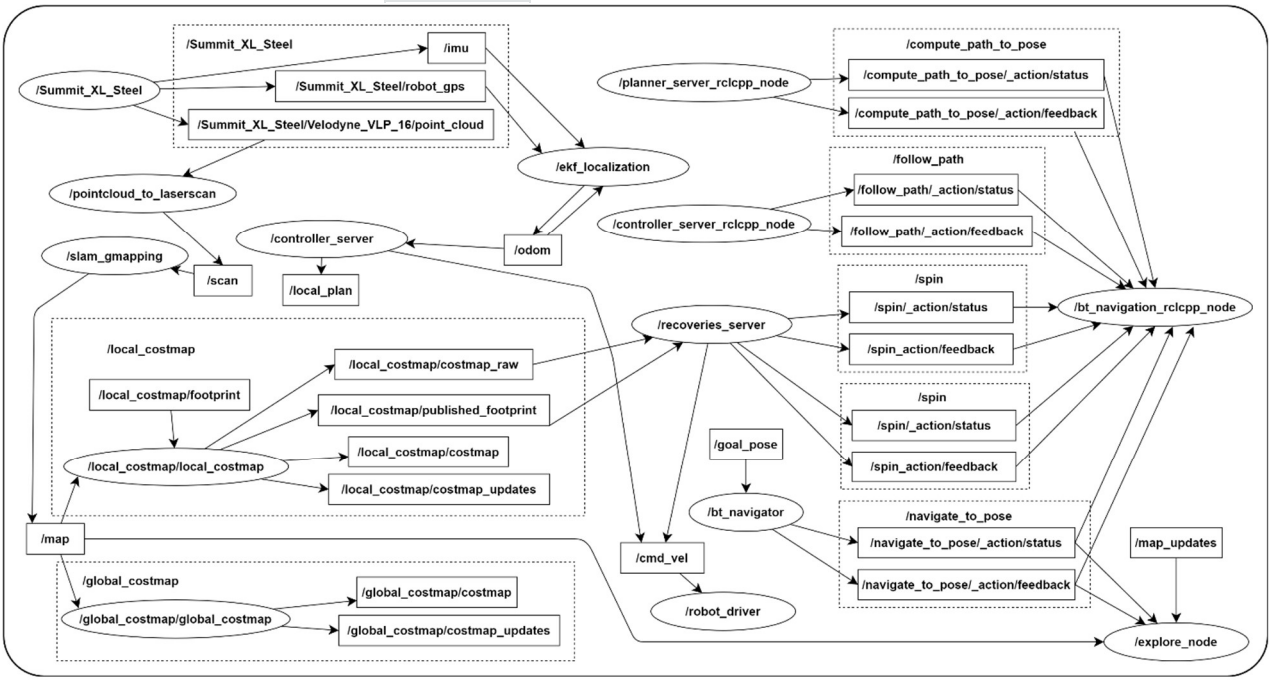
The project provides two methods to automate robot navigation, the difference being the use of 2D map generation for different implementations.

Gmapping uses GPS + IMU + LIDAR data and is suitable for outdoor GPS scenarios. Cartographer uses only IMU + LIDAR data and is suitable for indoor scenarios where GPS is unavailable.

- **Gmapping**:

First, the simulator is opened, which automatically loads the environment and the robot while providing the API interface. The IMU and GPS data given by the robot are then used to do the extended Kalman filter localization and issue the TF transform `base_link->odom`. Gmapping uses LIDAR and an odometer to create a probability-based 2D gridded map (`map_server`). The navigation part uses navigation2, which loads the existing map via `map_server` and provides the `map->odom` transformation. nav2 has a BT navigator that provides planning, behaviours, and controller servers. It can map the extent of obstacles on the map and provide navigation paths. The exploration package is then used to explore the environment based on boundaries greedily. The robot will greedily explore the environment until no boundaries are found. Move commands are sent to the `move_base` nav2 server.

The graph generated by `rqt_graph` to view the relationships between nodes and topics.



- **Cartographer**

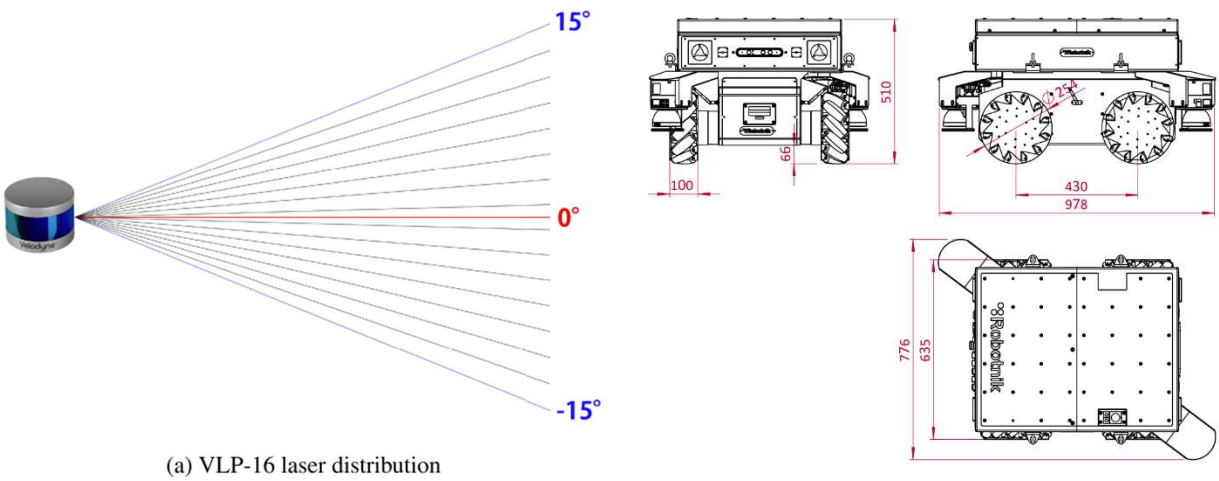
It is roughly the same as the framework used in Gmapping above, except that the extended Kalmar filter positioning using GPS is removed, and only IMU + LIDAR data is used. Cartographer itself will publish the `base_link->odom` and `map->odom` transformations and construct a 2D map.

3. Implementation (Realisation)

3.1 Webots

The Webots software contains the Summit XL Steel and Velodyne VLP-16 in the PROTO nodes. Import (Robot) and VelodyneVLP16 (Lidar) into the world by `Add a node` in Webots to avoid unnecessary troubles caused by using PROTO alone later.

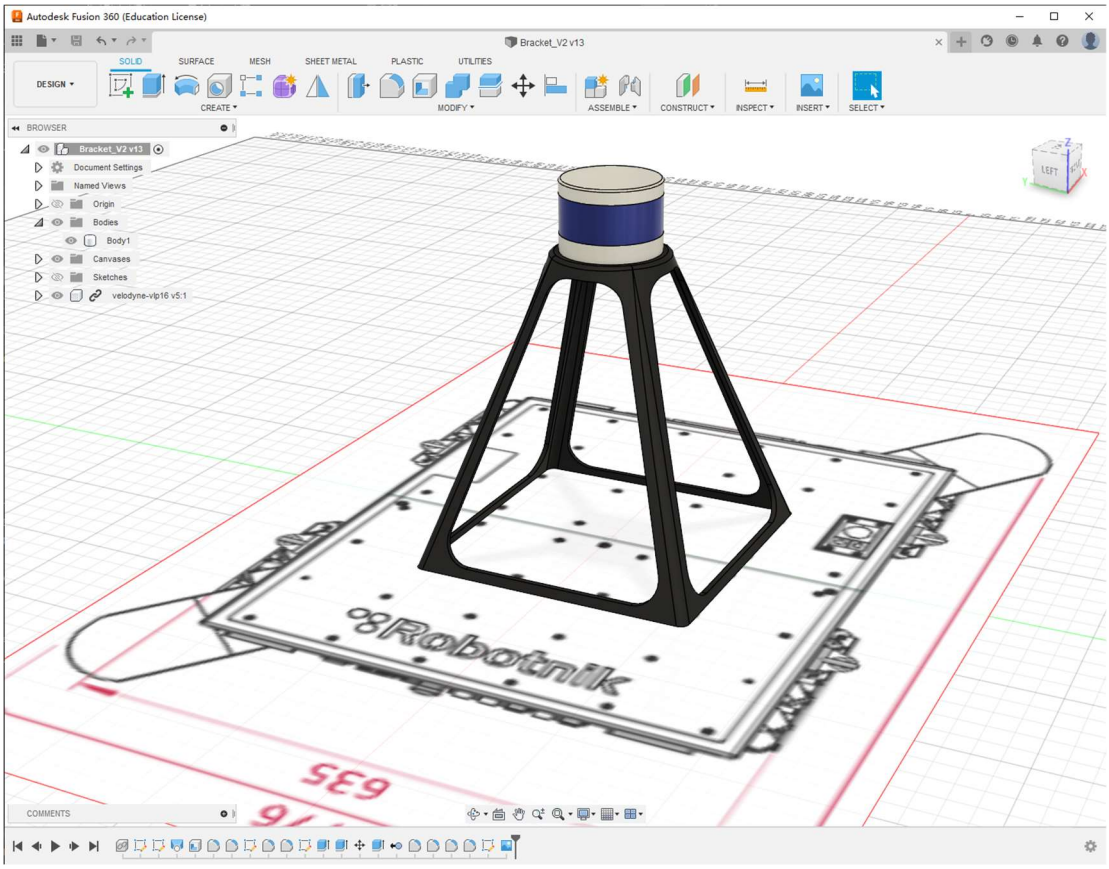
Regarding the Velodyne VLP-16 lidar, the documentation shows that the Field of view (vertical) is 30° (+15° to -15°) [13]. Due to the Field of view of the LIDAR pitch angle, if it is placed directly in the middle of the body, it is easy for the chassis to block the LIDAR's view of the ground. So the lidar needs to be elevated to avoid this problem.



(a) VLP-16 laser distribution

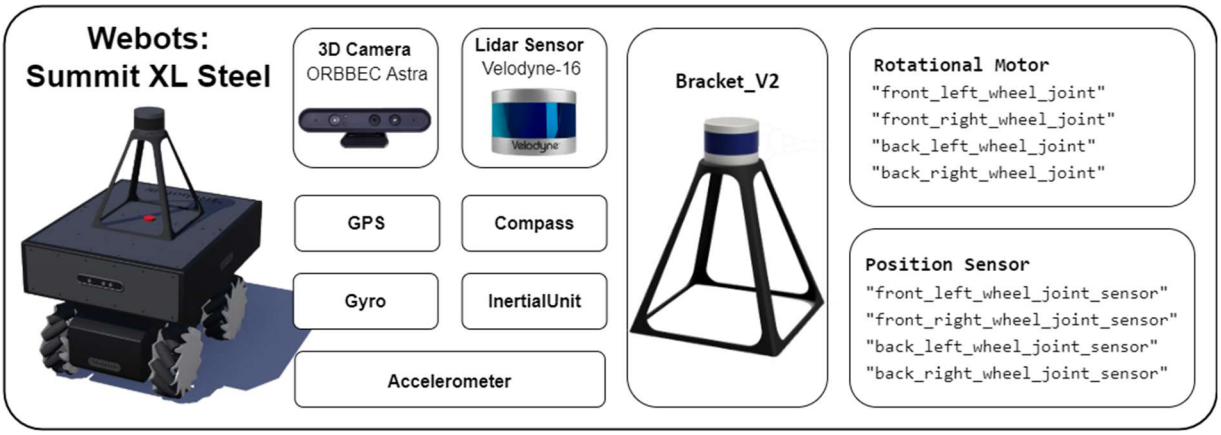
According to the drawing, the size of the robot platform is 580 * 720 mm. Therefore, I built a bracket in Fusion 360 (3D Design & Modeling Software), shaped like a pyramid and cut off the top for the lidar, with a height of 315 mm and a square bottom with a side length of 300 mm. The middle is hollowed out. This allows for a connection of the robot to the LIDAR and raises the LIDAR.

Here is a screenshot of the Fusion 360. The bracket model is imported into the `.wbt` file of Webots to complete the stitching of the robot and LIDAR.



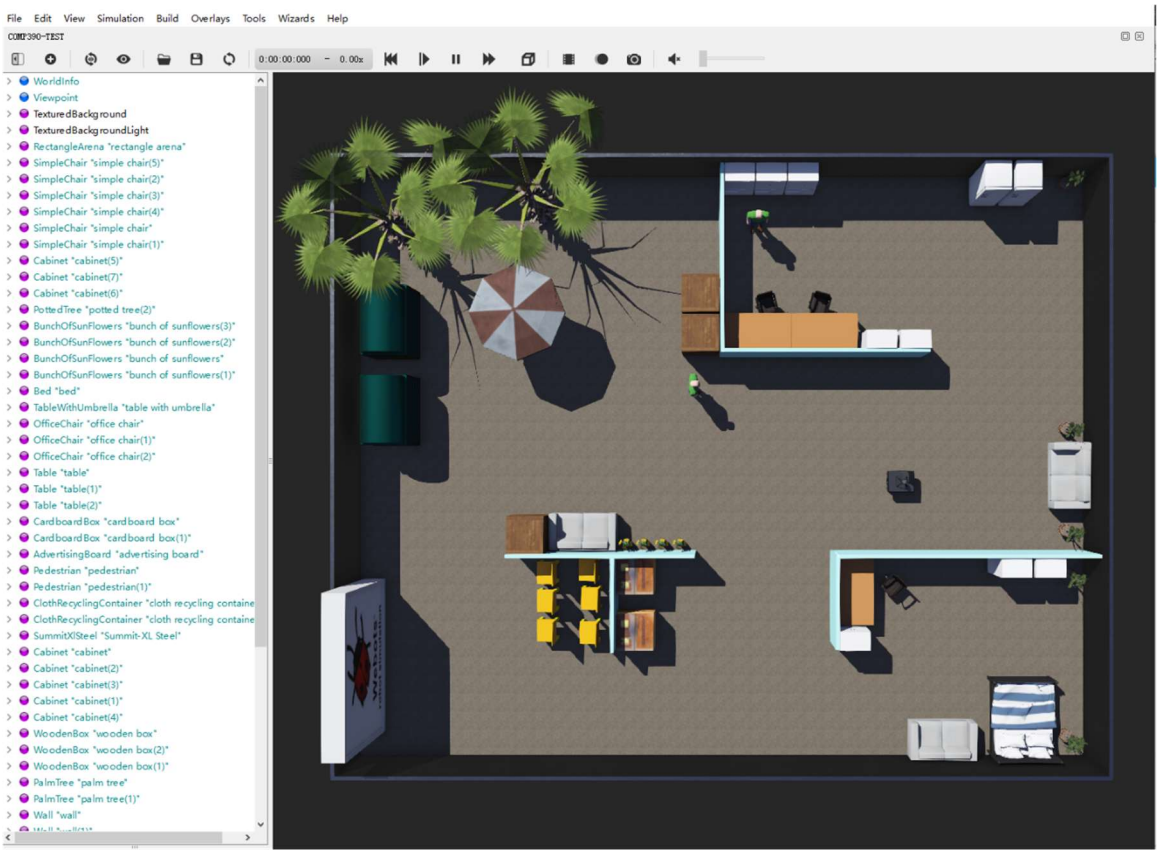
The Summit XL Steel PROTO file shows that the modification includes the ORBBEC Astra 3D camera, which can output depth images and colour images. Four rotating motors and four position sensors. It can be seen as four wheels, each consisting of a separate rotating motor and position encoder. I also added GPS, Compass, Gyro, InertiaUnit and Accelerometer sensors to the robot body.

In the diagram below, I have organized the components used by Summit XL Steel in Webots.



Add walls around the world. Multiple obstacles such as walls, pedestrians, boxes, and trees are placed in the middle of the open space. Also, the `basicTimeStep` parameter is adjusted in `WorldInfo` to accelerate the simulation. Because SLAM requires a high rate IMU data, the controller cannot be faster than the simulator by design.

Finally, save the world. Furthermore, get the file with the suffix `.wbt` in the world folder. At this point, the robot and environment modelling in the Webots simulator are complete.

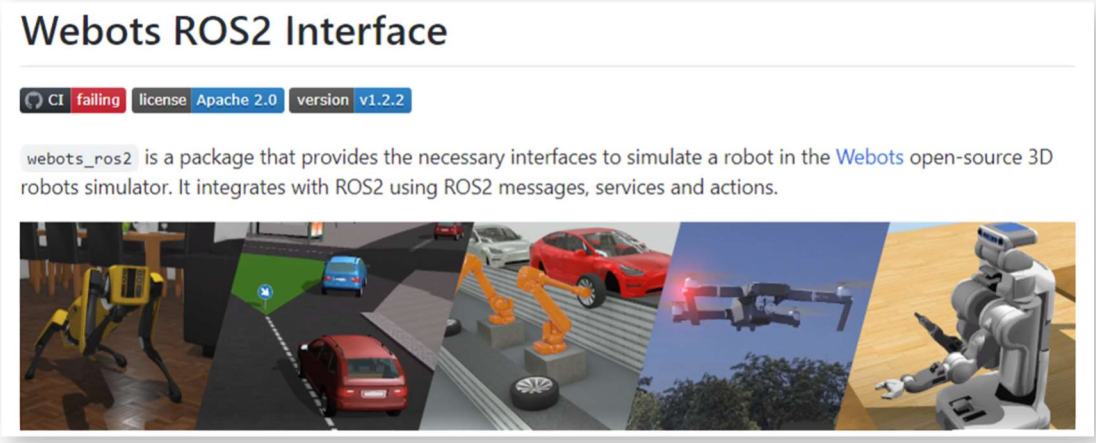


- It is important to note that the Webots software that uses wbt files should ideally be the same version. During my development, I found that Webots 2021b and Webots 2022a importing the same PROTO file would result in a coordinate system error. Therefore, I unified the software version number to Webots 2022a. (The latest version of the current)
- If the computer is not well equipped (without GPU), can adjust Tools -> Preferences
  General:
    Number of threads: select the maximum number
  OpenGL:
    Ambient Occlusion: Disabled
    Texture Quality: Low
    Disable shadows: Yes
    Disable anti-aliasing: Yes
  The above settings will make Webots simulator run faster.

3.2 ROS2

- **simulator:**

Cyberbotics provides the Webots ROS2 interface on GitHub [4].



There is also a detailed tutorial on setting up a robot simulation (Webots) in the ROS2 documentation. When creating the package with the following code, add the dependencies `rclpy`, `geometry_msgs` and `webots_ros2_driver`.

```
ros2 pkg create robot_driver simulator -dependencies rclpy geometry_msgs
webots_ros2_driver
```

This code will automatically generate a series of files under `src` and manually create the launch and world folders in the `src` folder. Use code editor (Visual Studio Code) to open this folder.

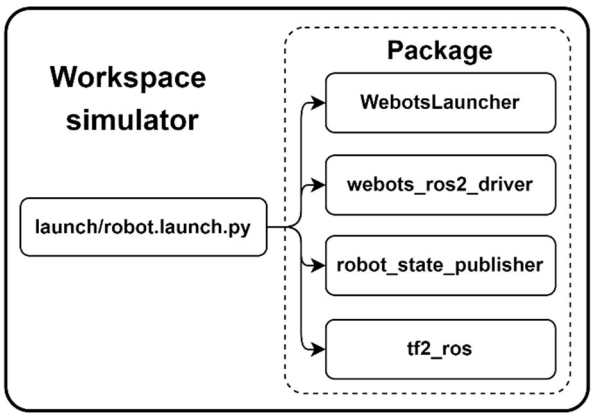
```
workspace/
└── simulator/
    ├── launch/
    │   └── robot.launch.py
    ├── resource/
    │   └── my_robot.urdf
    ├── simulator/
    │   ├── __init__.py
    │   ├── keyboard.py
    │   └── robot_driver.py
    ├── worlds/
    │   └── my_world.wbt
    ├── package.xml
    ├── setup.cfg
    └── setup.py
```

Put the created robot + environment model into the world folder with the name `my_world.wbt` and use it to start the simulation.

In the `robot_driver.py` file, the chassis interface section contains a subscriber for receiving `cmd_vel` messages of type `geometry_msgs/Twist Message`. Based on the interface provided by Summit XL Steel PROTO and the Mecanum mechanics model, an algorithm is created to control the movement of the chassis.

The `keyboard.py` file contains the publisher topic `cmd_vel` for testing the robot chassis by the keyboard.

Create the launch file `robot.launch.py`, add these packages in the launch file. Put the above files into the launch file and launch the simulator on your system.




```xml
<webots>
    <plugin type="webots_ros2_driver::Ros2IMU">
        <enabled>True</enabled>
        <updateRate>200</updateRate>
        <topicName>/imu</topicName>
        <alwaysOn>True</alwaysOn>
        <frameName>imu</frameName>
        <inertialUnitName>imu</inertialUnitName>
        <gyroName>gyr</gyroName>
        <accelerometerName>acc</accelerometerName>
    </plugin>
    <plugin type="simulator.robot_driver.main" />
</webots>
```

URDF file `my_robot.urdf` is created to declare the plugin, and a 9-axis IMU is added using `webots_ros2_driver::Ros2IMU` with a refresh rate of 200Hz. Unfortunately, the simulator cannot reach this speed.

At first, I thought it was a computer performance issue regarding this aspect of the problem. After testing on three different computers in succession, when `BasicTimeStep` is adjusted to 6 and the simulator is opened alone, `webots_ros2_driver::Ros2IMU` provides a `/imu` topic of 160-170Hz. But as soon as I open the LIDAR topic, the `/imu` rate drops to below 50Hz or even lower. This suggests that the problem is not related to the computer's performance but to the limitations of the simulator itself. By checking the information and asking the official Webots (Discord), I suspect the problem is with the middleware and event tasks. This was a rather frustrating discovery, and it was not something I could fix.

Before

```
ros@ros-NUC8i7HVK:~$ ros2 topic hz /imu
average rate: 153.942
        min: 0.002s max: 0.015s std dev: 0.00236s window: 156
average rate: 156.264
        min: 0.002s max: 0.021s std dev: 0.00261s window: 315
average rate: 155.510
        min: 0.002s max: 0.021s std dev: 0.00252s window: 470
average rate: 153.851
        min: 0.002s max: 0.021s std dev: 0.00256s window: 619
average rate: 154.849
        min: 0.002s max: 0.021s std dev: 0.00256s window: 778
average rate: 154.931
        min: 0.002s max: 0.021s std dev: 0.00256s window: 934
average rate: 155.207
        min: 0.002s max: 0.021s std dev: 0.00254s window: 1091
average rate: 154.706
        min: 0.002s max: 0.021s std dev: 0.00250s window: 1243
average rate: 155.451
        min: 0.001s max: 0.021s std dev: 0.00255s window: 1405
average rate: 155.067
        min: 0.001s max: 0.021s std dev: 0.00252s window: 1557
average rate: 155.143
        min: 0.001s max: 0.021s std dev: 0.00248s window: 1714
average rate: 154.839
        min: 0.001s max: 0.021s std dev: 0.00247s window: 1866
average rate: 155.212
        min: 0.001s max: 0.021s std dev: 0.00246s window: 2026
```

After

```
ros@ros-NUC8i7HVK:~$ ros2 topic hz /imu
average rate: 49.504
        min: 0.008s max: 0.046s std dev: 0.01195s window: 51
average rate: 47.179
        min: 0.007s max: 0.060s std dev: 0.01270s window: 97
average rate: 44.481
        min: 0.007s max: 0.060s std dev: 0.01305s window: 137
average rate: 42.367
        min: 0.007s max: 0.060s std dev: 0.01329s window: 174
average rate: 42.365
        min: 0.007s max: 0.060s std dev: 0.01342s window: 217
average rate: 42.409
        min: 0.007s max: 0.060s std dev: 0.01342s window: 260
average rate: 41.766
        min: 0.007s max: 0.060s std dev: 0.01318s window: 299
average rate: 42.767
        min: 0.007s max: 0.060s std dev: 0.01320s window: 350
average rate: 42.807
        min: 0.007s max: 0.060s std dev: 0.01322s window: 394
average rate: 42.326
        min: 0.007s max: 0.060s std dev: 0.01341s window: 432
average rate: 42.069
        min: 0.007s max: 0.060s std dev: 0.01354s window: 472
average rate: 41.846
        min: 0.007s max: 0.060s std dev: 0.01365s window: 512
average rate: 42.536
        min: 0.007s max: 0.060s std dev: 0.01365s window: 563
```

Modify the `setup.py` file by adding the files `my_world.wbt`, `my_robot.urdf` and `robot.launch.py` to the `data_files` variable. At this point, a complete package has been created.

- **localization:**

`kalman_filter_localization`, ROS2 package of Kalman Filter Based Localization in 3D using GNSS/IMU/Odometry [6].

```
// odom Publisher
std::string output_pose_name = "/odom";
current_pose_pub_ = create_publisher<nav_msgs::msg::Odometry>(output_pose_name, 10);
current_pose_pub_->publish(pose_);
geometry_msgs::msg::TransformStamped t;
// odom -> base_link
t.header.stamp = current_pose_.header.stamp;
t.header.frame_id = "odom";
t.child_frame_id = "base_link";
tf_broadcaster_->sendTransform(t);
```

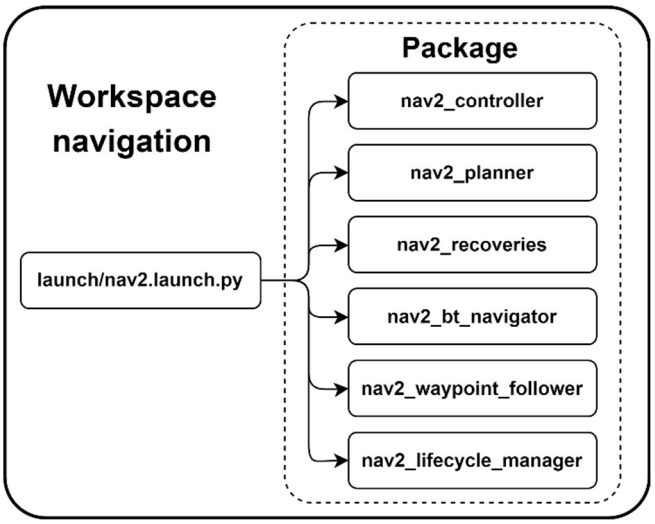
Add `/odom` publisher to ekf_localization_component.cpp in the src folder to implement the `odom` -> `base_link` conversion and publish `/tf`
Launch file is `ekf.launch.py`, under `ekf_localization_node` change to topics from `/Summit_XL_Steel` like gps and imu data. Add display of rviz. Use `odom` for reference frames and `base_link` for robot frames. This allows us to get the position of the robot relative to the world.

- **navigation:**

Create a new package named `navigation` and go to the folder.

```
ros2 pkg create --build-type ament_python navigation
```

Create files `nav2.launch.py` and `nav2.yaml` in the launch file and parameters file. Add packages and plugins respectively. The approximate format is shown below, with the launch file on the left and the param file on the right.




```
bt_navigator:
    plugin_lib_names: nav2*

controller_server:
    goal_checker: "nav2_controller::SimpleGoalChecker"
    FollowPath: "dwb_core::DWBLocalPlanner"

local_costmap:
    inflation_layer: "nav2_costmap_2d::InflationLayer"
    obstacle_layer: "nav2_costmap_2d::ObstacleLayer"

global_costmap:
    static_layer: "nav2_costmap_2d::StaticLayer"
    inflation_layer: "nav2_costmap_2d::InflationLayer"
    obstacle_layer: "nav2_costmap_2d::ObstacleLayer"

planner_server:
    GridBased: "nav2_navfn_planner/NavfnPlanner"

recoveries_server:
    spin: "nav2_recoveries/Spin"
    backup: "nav2_recoveries/BackUp"
    wait: "nav2_recoveries/Wait"
```

- **map_2d:**

  Gmapping:

  Using the ROS2 package called `SLAM_gmapping` from GitHub, the robot odometry and LIDAR data are used to create a 2D map.

  Fine-tuning of the `GridSlamProcessor` parameters in the `slam_gmapping.cpp` file.

  Cartographer:

  Create a new package named `cartographer`.

```
ros2 pkg create --build-type ament_python cartographer
```

  Some files and parameters were borrowed from turtlebot3_cartographer's ROS2 package. Create the launch file `cartographer.launch.py` and the parameters file `carto_2d.lua`. The package in the launch file is `cartographer_ros`. [carto-github]

  Due to Cartographer's graph-based optimization algorithm, it takes a long time to scan and determine the modelling map when the robot is stationary. It appears as the map changes from grey to white in RVIZ. I added map format conversion to reduce waiting time.

- **map_3d:**

Using the ROS2 package called `lio_sam` from GitHub

In `params.yaml` change the topic name and the parameters of the LIDAR, calibrate the parameters of the IMU and the gravity accelerometer.

```
# Topics
pointCloudTopic: "/Summit_XL_Steel/Velodyne_VLP_16/point_cloud" # Pointcloud data
imuTopic: "/imu"                                                # IMU data
odomTopic: "odometry/imu"      # IMU pre-preintegration odometry, samefrequency as IMU
gpsTopic: "/gps/odom"          # GPS odometry topic from navsat, seemodule_navsat.launch file
# Frames
lidarFrame: "Velodyne_VLP_16"
baselinkFrame: "base_link"
odometryFrame: "odom"
mapFrame: "map"
# Sensor Settings
sensor: velodyne          # lidar sensor type, either'velodyne' or 'ouster'
N_SCAN: 16                # number of lidar channel (ie., 16, 32, 64, 128)
Horizon_SCAN: 1800        # lidar horizontalresolution (Velodyne:1800, Ouster:512,1024,2048)
```

- **explore:**

`eplore_lite` ROS2 package from m-explore ROS2 port. Changes have been made to some of the documents to adapt them to the project's environment.

4. Testing & Evaluation

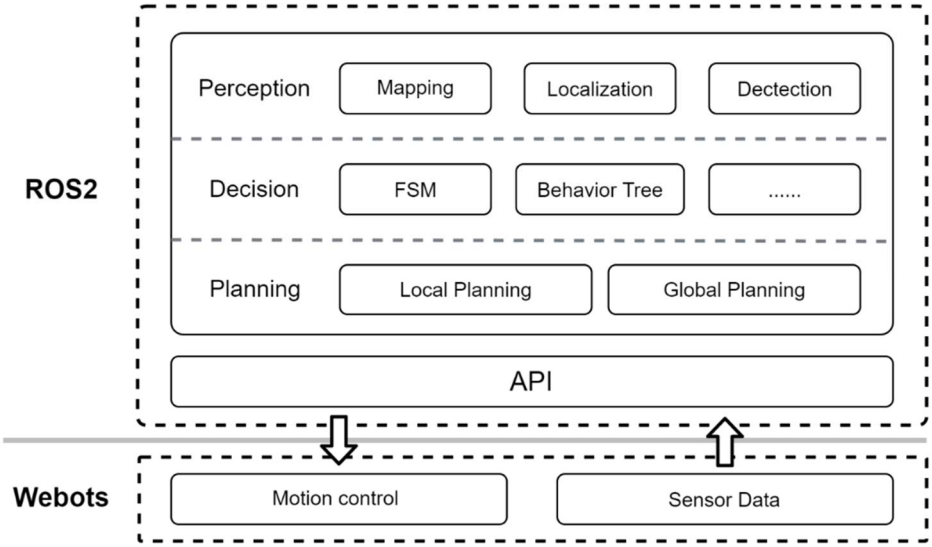
The test is divided into five parts: scenes and robots in Webots, motion control, API interface, SLAM algorithm, navigation and exploration.

- The **scenes and robots in Webots** are launched in the `webots_ws` work package, which shows the environment and robots fully. The physics engine and the rendering engine are both working fine. The desired objectives are met.
- **Motion control**: a software component of the `webots_ws` work package, completes the motion control. Webots robot receives motion commands and executes them correctly. The desired goal is met.
- **API interfaces**: the `webots_ros2_driver` component automatically provides interfaces to sensors and publishes topics. For example, include LIDAR, position sensors, IMU and vision sensors. Of these, IMU cannot provide data at high rates due to the limitations of the Webots simulator itself. Due to the middleware and incoming event methods involved, this is not something I can accomplish and is not the project's primary task. The default is 30Hz, and the data update rate was increased by adjusting `basicTimeStep` in Webots' worldInfo to be optimised to 160Hz. There is some impact on the accuracy of subsequent map modelling.
- **SLAM algorithm:** containing localisation and map modelling. There are two approaches, one GPS+IMU+LIDAR, where GPS and IMU data are used for localisation and LIDAR for map building. The other is IMU+LIDAR, based on GraphSLAM for localisation and map building. Both implementations are working fine. They meet the desired objectives.
- **Navigation**: the nav2 component was used to map the obstacle ranges and generate local paths for the navigation part. Due to time constraints, global navigation was not used. Only local navigation was used. They meet the desired objectives.
- **Exploration**: the greedy algorithm searches an unknown environment until no boundary unknown is found. They meet the desired objectives.

5. Conclusion

Robotics is a relatively broad subject involving several areas such as environmental perception, intelligent decision-making, navigation planning and motion control. Thus, the study of robotics is an important topic. In this project, the automatic navigation of the robot was successfully completed using the Webots simulator and ROS2.

The web and books provide a wealth of tutorials and documentation, and Webots provides a partial listing of the robot on Github. I have some basic programming skills in C++ and Python and some simple mathematics. I have worked independently on creating the robot, motion control, map modelling and positioning, navigation planning and exploration. The software framework for the project is roughly shown in the diagram below.



The entire software system of this project is decoupled from multiple modules and is very portable. Users from different fields can study and test an algorithm in this robotics project. This is one of the highlights of this project.

The learning and research process was not always smooth. At first, I aimed to use 3D maps for modelling and automatic navigation, and this was the real purpose of my choice of multi-line LIDAR. It used lio_sam to make a global point cloud map and use Octomap to make a raster map or use Crtographer 3D SLAM. An octree-based 3D map creation tool that shows the presence or absence of obstacle areas [14]. I am also interested in this algorithm myself. Unfortunately, the Webots simulator has its limitations and cannot output high rate IMU data. I also find that adjusting the simulation speed in Webots reduces the accuracy. The low rate and low accuracy of data results in impoverished global point cloud generation in real-time. Furthermore, the performance of the 3D mapping system is heavily dependent on the quality of the IMU measurements. I found that the difference in IMU rate alone between the dataset and the simulator was more than ten times. It was also the only problem that I could not solve in this project. So instead, I used 2D maps to complete the navigation, sacrificing the advantages of multi-line LiDAR.

If there are conditions later, I will directly use the robot in the real world to do the test and complete the automatic navigation, because this limitation does not exist in the real world. For now, simulators are no match for the real world.

However, the disadvantage mentioned above do not overshadow the excellence of the Webots simulator itself. The Webots simulator is undoubtedly a good choice for beginners who want to learn about robotics, especially if you do not have the hardware. Webots has a wide range of robots, including UGV, UAV, AUV, robotic arms and human avatars, and supports a wide range of sensors. The Webots are in a very active state of development. More importantly, the Webots Discord server where people can communicate and official technician responses are very responsive. This alone is very rare in simulator software. I have significantly benefited from the learning process.

6. BCS Project Criteria & Self-Reflection

Practical and analytical skills:
The project uses the knowledge and skills I have acquired in software engineering and algorithms at university and the knowledge of linear algebra. It also makes use of my strengths in computer science in coding.

Creativity:
It is not an easy process to build a robot from scratch and run it successfully with autonomous navigation. The decoupling of several modules for easy portability in software frameworks has excellent potential. It can be used directly in real robots (using the standard ROS2 API interface). What is more, each module uses almost the latest and most popular technology, which will ensure that the project will not be outdated for some time.

Evaluation:
The Summit XL Steel robotics platform was chosen for this project to complete the project requirements. It covers various modules for environment awareness, intelligent decision making, navigation planning and motion control. Each module is self-contained and can be programmed independently. It is a relatively exemplary system in terms of software architecture alone. There is a strong shaping power.

Real need:
A complete robotics architecture has been designed so that anyone who needs it can use it. Especially for beginners and those without the hardware, this project will give them a quick way to learn robotics systematically. People in related fields can also use it to optimise the algorithms and test the effectiveness of specific modules.

Self-manage:
I have good self-management skills, a clear head and an understanding of how the robot works. This has ensured the successful completion of the project.

Self-evaluation:
In my self-evaluation, I have learned a lot during this project. I have gained a deeper understanding of how robots work and have figured out my research direction. I hope to keep up the enthusiasm and get involved in more significant projects.

References

[1] "The advantages and disadvantages of the Mecanum wheel," cmtendency. [Online], Available: https://omni-robots.com/the-advantages-and-disadvantages-of-the-mecanum-wheel/ [Accessed Dec. 2, 2021].

[2] "Webots OPEN SOURCE ROBOT SIMULATOR," *Cyberbotics Ltd.* [Online], Available: https://cyberbotics.com/ [Accessed Dec. 18, 2021].

[3] "ROS 2 Documentation," [Online], Available: https://docs.ros.org/en/foxy/index.html [Accessed Jan. 22, 2022].

[4] "Webots ROS2 Interface," *Cyberbotics Ltd.* [Online], Available: https://github.com/cyberbotics/webots_ros2 [Accessed Jan. 23, 2022].

[5] H.Taheri, B.Qiao, and N.Ghaeminezhad, "Kinematic Model of a Four Mecanum Wheeled Mobile Robot," *International Journal of Computer Applications (0975 – 8887),* Volume 113 No. 3, March 2015.

[6] T. Moore, D. Stouch, "A Generalized Extended Kalman Filter Implementation for the Robot Operating System," *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)* July 2014.

[7] S.Macenski, and T.Moore, "Nav2," [Online], Available: https://navigation.ros.org/index.html [Accessed Dec. 18, 2021].

[8] G.Grisetti, C.Stachniss, and W.Burgard, "Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters," *IEEE Transactions on Robotics,* Volume 23, pages 34-46, 2007.

[9] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-Time Loop Closure in 2D LIDAR SLAM," *in Robotics and Automation (ICRA),* 2016 IEEE International Conference on. IEEE, 2016. pp. 1271–1278.

[10] T.Shan, B.Englot, D.Meyers, W.Wang, C.Ratti, and D.Rus, "LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping," arXiv:2007.00258 [cs.RO], Jul. 2020.

[11] J.Z.Kolter, A.Coates, A.Y.Ng, Y.Gu, and C.DuHadway. 2008. "Space-indexed dynamic programming: learning to follow trajectories. In Proceedings of the 25th international conference on Machine learning," (ICML '08). *Association for Computing Machinery*, NY, USA, 488–495. doi: /10.1145/1390156.1390218.

[12] B.Gerkey, L. Quigley, and D. Smart, " Programming Robots with ROS: A Practical Introduction to the Robot Operating System," *O'Reilly Media, Inc.* Dec. 2015.

[13] J. Lambert et al., "Performance Analysis of 10 Models of 3D LiDARs for Automated Driving," *in IEEE Access*, vol. 8, pp. 131699-131722, 2020, doi: 10.1109/ACCESS.2020.3009680.

[14] A.Hornung, K.M.Wurm, M.Bennewitz, et al. "OctoMap: an efficient probabilistic 3D mapping framework based on octrees," *Auton Robot 34,* 189–206 (2013). doi: 10.1007/s10514-012-9321-0.

Appendices

List of Abbreviations

LIDAR: Light Detection and Ranging
2D: two-dimensional
3D: three-dimensional
ROS2: Robot Operating System 2
GPS: Global Positioning System
SLAM: simultaneous localization and mapping
IMU: inertial measurement unit
NAV2: navigation 2
Odom: Odometry
GNSS: Global navigation satellite system
TF: transform
PBRF: Rao-Blackwellized Particle Filter
UGV: Ground mobile robots
UAV: aerial robots
AUV: underwater robots

List of Attached Files

This is a list of the companion files for this project. The source code is also available on GitHub (The code for this project may be updated later) Or ZIP file.
URL: https://github.com/MistyMoonR/Webots-SummitXL

```
├── README.md
├── scripts
│   ├── cartographer-explore.sh
│   ├── gmapping-explore.sh
│   ├── environmental-install.sh
│   └── ros2+webots-install.sh
└── workspace
    ├── explore
    │   └── explore_lite
    ├── localization
    │   ├── kalman_filter_localization
    │   └── gmapping
    ├── map_2D
    │   ├── cartographer
    │   ├── map_2D_master
    │   ├── pointcloud_to_laserscan
    ├── map_3D
    │   ├── lio_sam
    │   └── octomap
    ├── navigation
    └── simulator
```

Workspace file description:

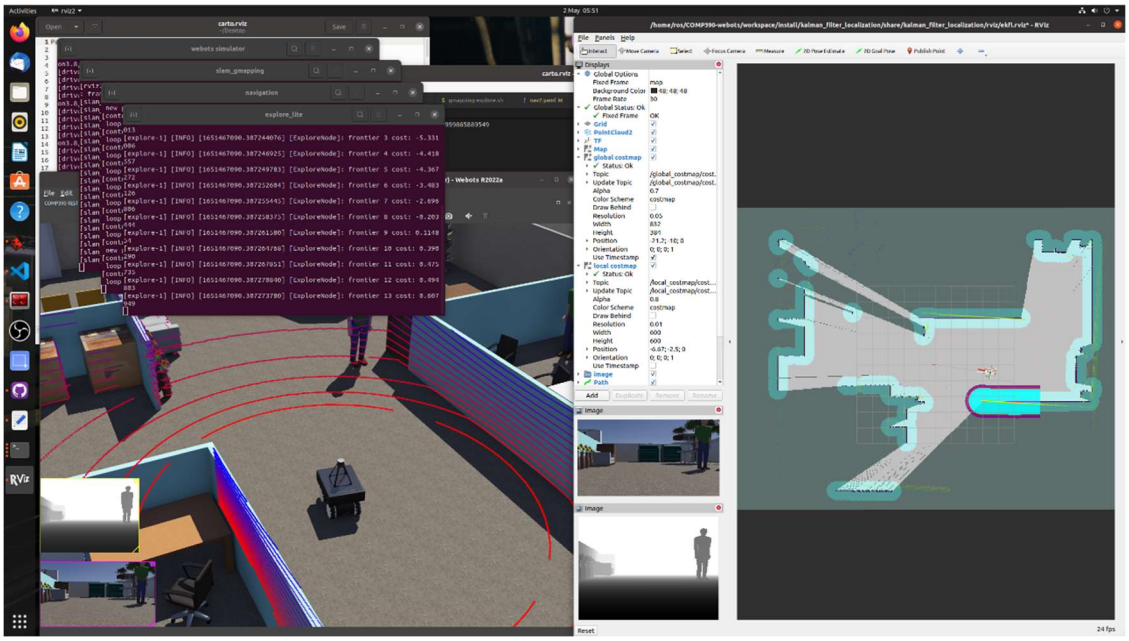
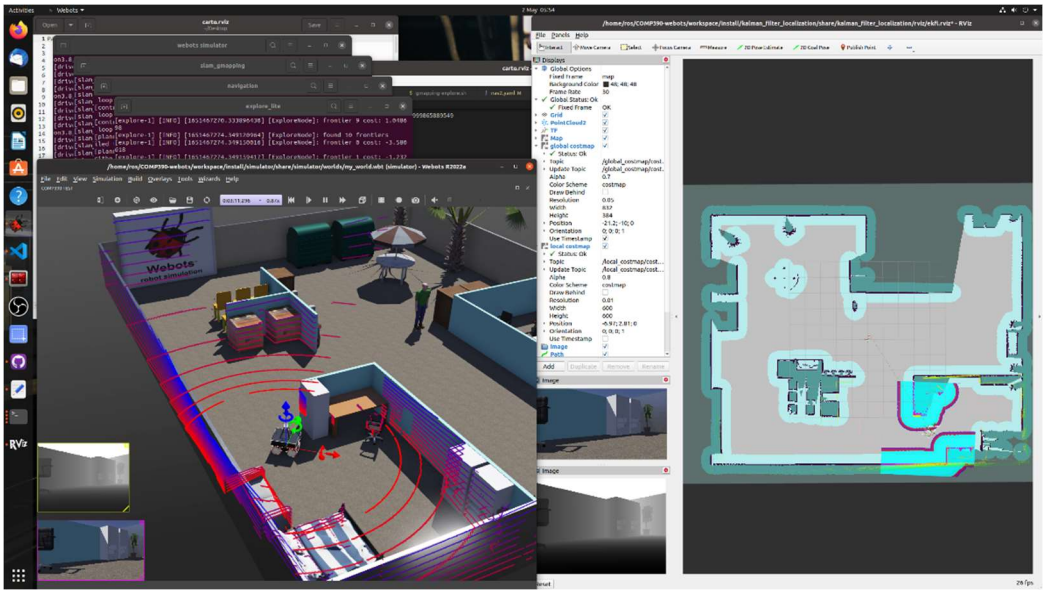
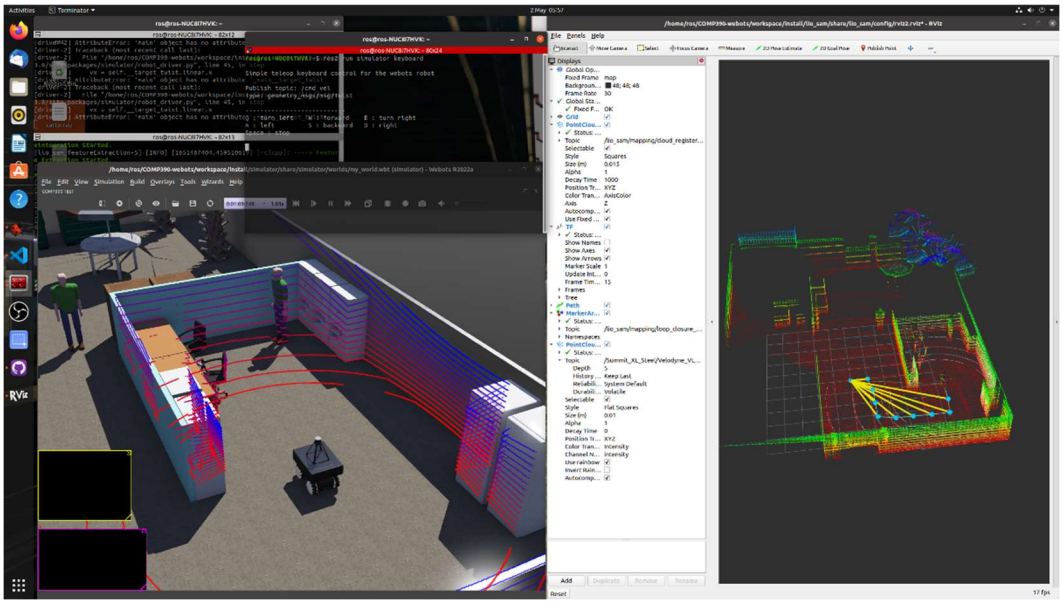
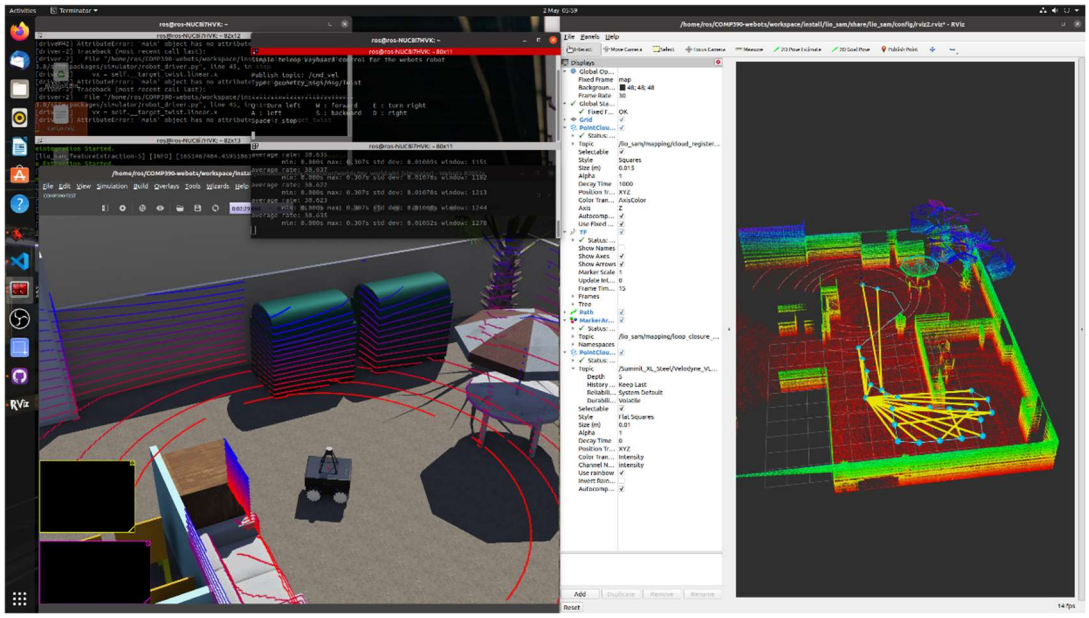
- `explore` Exploration algorithm
- `localization` Kalman filter localization
- `map_2D` Two-dimensional mapping
- `map_3D` Three-dimensional mapping
- `navigation` navigation
- `simulator` webots simulator

Scripts file description:

`ros2+webots-install.sh` Install ros2 + webots and some tools
`environmental-install.sh` Install the environment
`gmapping-explore.sh` Autonomous exploration software using gmapping's map
`cartographer-explore.sh` Autonomous exploration software using cartographer's map

How to run

There are details of how to compile and run the program for this project in README.md
The following figure shows the compiled output of a complete package

```
Finished <<< openslam_gmapping [0.86s]
Finished <<< explore_lite [0.95s]
Finished <<< kalman_filter_localization [0.97s]
Finished <<< mobile_robot_ekf [1.02s]
Finished <<< lio_sam [1.32s]
Finished <<< octomap_msgs [1.39s]
Finished <<< pointcloud_to_laserscan [0.69s]
Finished <<< slam_gmapping [0.55s]
Finished <<< octomap_server2 [0.30s]
Finished <<< navigation [1.71s]
Finished <<< cartographer [1.76s]
Finished <<< simulator [1.13s]

Summary: 12 packages finished [2.32s]
```

Some screenshots