



Singapore University of Technology and Design

Intelligent Robotics Term 7
ROS2 Challenge


Authors:	Student ID:
Aum Bansal	1005617
Joel Luke Tan Yi	1006321
Nuryn Insyirah Bte Mohd Adib	1006091
Laura Zanon Barney	1009527

Table of Contents

1. Introduction	3
2. Experimental Setup	3
2.1 Hardware Setup	3
2.1.1 Webcam	3
2.1.2 TurtleBot3	3
2.2 Software Setup.....	4
2.2.1 Ultralytics YOLOv8n	4
1.2.2 Frontier-based Exploration	5
2.3 Setting and Configuration	5
3) Algorithm for Counting Images Using Python	11
3.1 Location-based counting using LIDAR	11
3.2 Stop sign detection.....	12
3.3 Location-based counting using Occupancy Grid	12
4) Tuned Frontier_Exploration parameters	13
5) Results with images.....	16
6) A video (separate files consisting of screen recording & camera) illustrating the Performance.....	17
7) References.....	18

1. Introduction

Our task for this project is to navigate a TurtleBot3 robot through a maze while achieving the following 3 objectives:

1. Mapping the maze and followed by performing waypoint navigation (with SLAM using manual waypoints).
2. Mapping autonomously with Frontier Exploration;
3. Include image recognition (i.e. count number of items seen plus stop autonomously using the STOP sign. 

Hence, this report will take you through how we managed to achieve the above objectives.

2. Experimental Setup

In this section, we will discuss both the hardware and software integration respectively in our Turtlebot3 project. The hardware setup part describes the different important hardware components used. Hence it will cover (1) the webcam and the webcam driver used, as well as the (2) characteristics of the TurtleBot. Meanwhile, the software setup part will cover the Open-source software libraries used, which are (1) Nadja4 Frontier-based Exploration for autonomous navigation and (2) Ultralytics YOLO library for image recognition. Lastly in this section, we will include the Setting and Configuration, where we will cover how we connected the remote PC to the physical TurtleBot3 itself, accompanied by screenshots of the Command Line Interface (CLI). Given that for the image recognition part, we needed to count the number of items seen plus stop the TurtleBot3 autonomously using the STOP sign, we developed an algorithm for this using Python, which will be elaborated in Section 3) Algorithm for Counting Images Using Python.

2.1 Hardware Setup

2.1.1 Webcam

The camera model used was the ASUS Webcam C3, which is a USB camera with 1080p 30 fps recording. For the camera to communicate with the Latte Panda's (LP) Linux kernel and ROS2, an external driver needs to be installed. Upon evaluation, the `usb_cam` driver was found to be suitable and hence was installed on the LP.

2.1.2 TurtleBot3

The robot that we will combine the camera to is a TurtleBot3 by Open robotics and Robotis. The TurtleBot3 robot has 2 models: "burger" and "waffle"- both of which differ in terms of their physical properties. The model used needs to be specified, and thus for this project we specified our TurtleBot3 as the "burger" model.

2.2 Software Setup

2.2.1 Ultralytics YOLOv8n

YOLOv8n is an object detection model that operates by initially dividing the input image into a grid of cells, each cell in the grid is responsible for localizing and predicting the class of the object that it covers, along with the probability/confidence value [1]. Figure 1 shows how the algorithm divides the images into a grid and then detects the object.

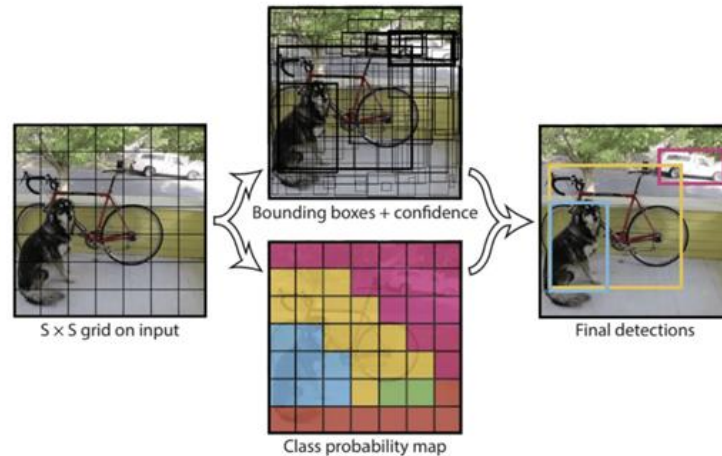


Figure 1: YOLO. Image from [Medium]. URL: [Faster R-CNN vs YOLO vs SSD — Object Detection Algorithms | by Abonia Sojasingarayar | IBM Data Science in Practice | Medium](#)

The YOLO algorithm is one of the best object detection algorithms because of the following reasons [2]:

Speed: This algorithm improves the speed of detection because it can predict objects in real-time.

High accuracy: YOLO is a predictive technique that provides accurate results with minimal background errors.

The yolobot_recognition package subscribes to a topic, which usb_cam publishes images to. These images are run through the object detection, allowing for class names to be identified, as well as bounding boxes to be drawn around the detected object. The class names and bounding box locations are published as an array to the /Yolov8_Inference topic. This topic will be read by other packages to perform stop sign detection and object counting.

In order to reduce false detections, a confidence threshold is set in the prediction function. Furthermore, the class list to detect is restricted to those in the maze. This prevents unnecessary messages from being sent out of detected objects that are not in the maze, and decreases the number of classes the model will have to search the image for.

1.2.2 Frontier-based Exploration

We decided to use Nadja4 Frontier-based Exploration, which is a ROS package that maps an unknown indoor environment using an autonomous mobile ground robot. It combines exploratory behavior with simultaneous localization and mapping (SLAM) to autonomously map any unknown indoor environment. [3]

2.3 Setting and Configuration

The next crucial step is to establish a connection between the remote PC and the physical TurtleBot3 so that they can share the same ROS topics. To be more precise, the connection is to enable the remote PC (Subscriber) to subscribe to topics published by TurtleBot3's LP (Publisher) by both being in same network environment. This connection was established using SSH connection. After the connection was established, we ran the TurtleBot3 bringup launch file directly on the LP, via SSH on the remote PC. The launch file initialises all the relevant ROS2 nodes and topics to control the robot. After that, we also ran all the other relevant launch files that we used in this project in separate terminals simultaneously. The screenshot of the terminals can be found below:

TurtleBot3 Bringup (terminal 1, via SSH)

```
arms@arms-LP: ~/ros2_ws 112x27
Last login: Mon Dec 9 18:53:17 2024 from 192.168.0.122
arms@arms-LP:~$ cd ros2_ws/
arms@arms-LP:~/ros2_ws$ source install/setup.bash
arms@arms-LP:~/ros2_ws$
ros2 launch turtlebot3_bringup robot.launch.py
[INFO] [launch]: All log files can be found below /home/arms/.ros/log/2024-12-09-18-59-05-151703-arms-LP-2992
[INFO] [launch]: Default logging verbosity is set to INFO
urdf_file_name : turtlebot3_burger.urdf
[INFO] [robot_state_publisher-1]: process started with pid [2993]
[INFO] [ld08_driver-2]: process started with pid [2995]
[INFO] [turtlebot3_ros-3]: process started with pid [2997]
[ld08_driver-2] LD5-02 started successfully
[robot_state_publisher-1] [INFO] [1733745545.526270081] [robot_state_publisher]: got segment base_footprint
[robot_state_publisher-1] [INFO] [1733745545.526410058] [robot_state_publisher]: got segment base_link
[robot_state_publisher-1] [INFO] [1733745545.526434860] [robot_state_publisher]: got segment base_scan
[robot_state_publisher-1] [INFO] [1733745545.526448720] [robot_state_publisher]: got segment caster_back_link
[robot_state_publisher-1] [INFO] [1733745545.526460045] [robot_state_publisher]: got segment imu_link
[robot_state_publisher-1] [INFO] [1733745545.526472482] [robot_state_publisher]: got segment wheel_left_link
[robot_state_publisher-1] [INFO] [1733745545.526487671] [robot_state_publisher]: got segment wheel_right_link
[turtlebot3_ros-3] [INFO] [1733745545.532216506] [turtlebot3_node]: Init TurtleBot3 Node Main
[turtlebot3_ros-3] [INFO] [1733745545.533420892] [turtlebot3_node]: Init DynamixelSDKWrapper
[turtlebot3_ros-3] [INFO] [1733745545.535613939] [DynamixelSDKWrapper]: Succeeded to open the port(/dev/ttyACM0)
!
[turtlebot3_ros-3] [INFO] [1733745545.537527567] [DynamixelSDKWrapper]: Succeeded to change the baudrate!
[turtlebot3_ros-3] [INFO] [1733745545.573438171] [turtlebot3_node]: Start Calibration of Gyro
[turtlebot3_ros-3] [INFO] [1733745550.573618183] [turtlebot3_node]: Calibration End
[turtlebot3_ros-3] [INFO] [1733745550.573706829] [turtlebot3_node]: Add Motors
```

Image Counter (terminal 2)

```

arms@arms-Legion-5-15IAH7H: ~/turtlebot3_frontier
File "/opt/ros/humble/local/lib/python3
.10/dist-packages/rclpy/context.py", line
102, in shutdown
    self.__context.shutdown()
rclpy._rclpy_pybind11.RCLError: failed to
shutdown: rcl_shutdown already called on
the given context, at ./src/rcl/init.c:2
41
[ros2run]: Process exited with failure 1
arms@arms-Legion-5-15IAH7H:~/turtlebot3_f
arms@arms-Legion-5-15IAH7H:~/turtlebot3_f
rontier_based_ws$ source install/setup.ba
sh
arms@arms-Legion-5-15IAH7H:~/turtlebot3_f
arms@arms-Legion-5-15IAH7H:~/turtlebot3_f
rontier_based_ws$ ros2 run image_count im
age_counter
yaw: -0.5376896858215331
odom_x: 0.05257817371249128
odom_y: -0.03142174604929573
yaw: -0.5377789735794067
odom_x: 0.05257817371249128
odom_y: -0.03142174604929573
yaw: -0.5376291275024415
odom_x: 0.05259991272817088
odom_y: -0.03143470930665801

```

Yolo (terminal 3, via SSH)

```

arms@arms-LP: ~/ros2_ws 58x27
Learn more about enabling ESM Apps service at https://ubun
tu.com/esm

Failed to connect to https://changelogs.ubuntu.com/meta-re
lease-logs. Check your Internet connection or proxy setting
s

Last login: Mon Dec 9 18:53:09 2024 from 192.168.0.122
arms@arms-LP:~$ cd ros2_ws/
arms@arms-LP:~/ros2_ws$ source install/setup.bash
arms@arms-LP:~/ros2_ws$ ros2 launch yolobot_recognition la
unch_yolov8.launch.py
[INFO] [launch]: All log files can be found below /home/ar
ms/.ros/log/2024-12-09-18-58-48-936027-arms-LP-2969
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [yolov8_ros2_pt.py-1]: process started with pid [29
70]
[yolov8_ros2_pt.py-1] /home/arms/.local/lib/python3.10/sit
e-packages/matplotlib/projections/_init_.py:63: UserWarn
ing: Unable to import Axes3D. This may be due to multiple
versions of Matplotlib being installed (e.g. as a system p
ackage and as a pip package). As a result, the 3D projecti
on is not available.
[yolov8_ros2_pt.py-1] warnings.warn("Unable to import Ax
es3D. This may be due to multiple versions of "
[yolov8_ros2_pt.py-1]

```

Usb webcam (terminal 4, via SSH)

```

arms@arms-LP: ~/ros2_ws 100x27
arms@arms-LP:~$ cd ~/ros2_ws
arms@arms-LP:~/ros2_ws$ source install/setup.bash
arms@arms-LP:~/ros2_ws$ ros2 launch usb_cam camera
camera_info.yaml camera.launch.py
arms@arms-LP:~/ros2_ws$ ros2 launch usb_cam camera.launch.py
[INFO] [launch]: All log files can be found below /home/arms/.ros/log/2024-12-09-18-57-47-134853-arms-LP-2533
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [usb_cam_node_exe-1]: process started with pid [2534]
[usb_cam_node_exe-1] [INFO] [1733745467.799844919] [rgb_cam]: camera_name value: test_camera
[usb_cam_node_exe-1] [WARN] [1733745467.799941111] [rgb_cam]: framerate: 10.000000
[usb_cam_node_exe-1] [INFO] [1733745467.803230753] [rgb_cam]: camera calibration URL: package://usb_cam/config/camera_info.yaml
[usb_cam_node_exe-1] [INFO] [1733745468.052834216] [rgb_cam]: Starting 'test_camera' (/dev/video0) at 320x240 via mmap (mjpeg2rgb) at 10 FPS
[usb_cam_node_exe-1] This device supports the following formats:
[usb_cam_node_exe-1]   Motion-JPEG 1920 x 1080 (30 Hz)
[usb_cam_node_exe-1]   Motion-JPEG 1920 x 1080 (20 Hz)
[usb_cam_node_exe-1]   Motion-JPEG 1920 x 1080 (15 Hz)
[usb_cam_node_exe-1]   Motion-JPEG 1920 x 1080 (10 Hz)

```

Nadja Frontier Exploration (terminal 5)

```

arms@arms-Legion-5-15IAH7H: ~/turtlebot3_frontier
timestamp on the message is earlier than
all the data in the transform cache'
^C[WARNING] [launch]: user interrupted with ctrl-c (SIGINT)
[rviz2-2] [INFO] [1733905157.349778246] [rclcpp]: signal_handler(signum=2)
[async_slam_toolbox_node-1] [INFO] [1733905157.349781631] [rclcpp]: signal_handler(signum=2)
[INFO] [async_slam_toolbox_node-1]: process has finished cleanly [pid 14353]
[ERROR] [rviz2-2]: process has died [pid 14355, exit code -11, cmd '/opt/ros/humble/lib/rviz2/rviz2 -d /home/arms/turtlebot3_frontier_based_ws/install/autonomous_exploration/share/autonomous_exploration/rviz/exploration_default_view.rviz --ros-args -r __node:=rviz2 --params-file /tmp/launch_params_5hkw4opl'].
arms@arms-Legion-5-15IAH7H:~/turtlebot3_frontier_based_ws$ source install/setup.bash
arms@arms-Legion-5-15IAH7H:~/turtlebot3_frontier_based_ws$ ros2 launch autonomous_exploration autonomous_exploration.launch.py

```

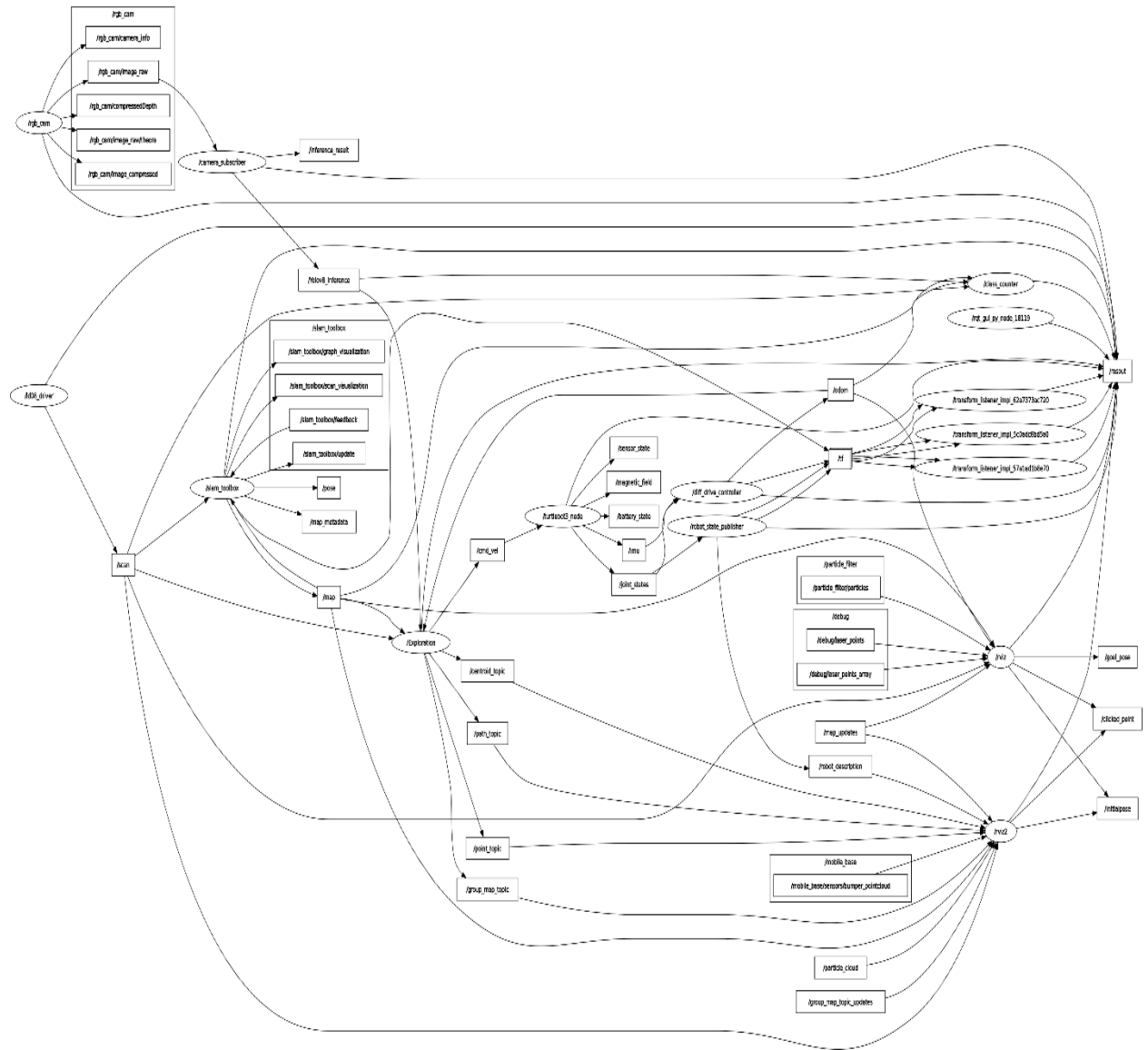
Nadja Autonomous Control (terminal 6)


```

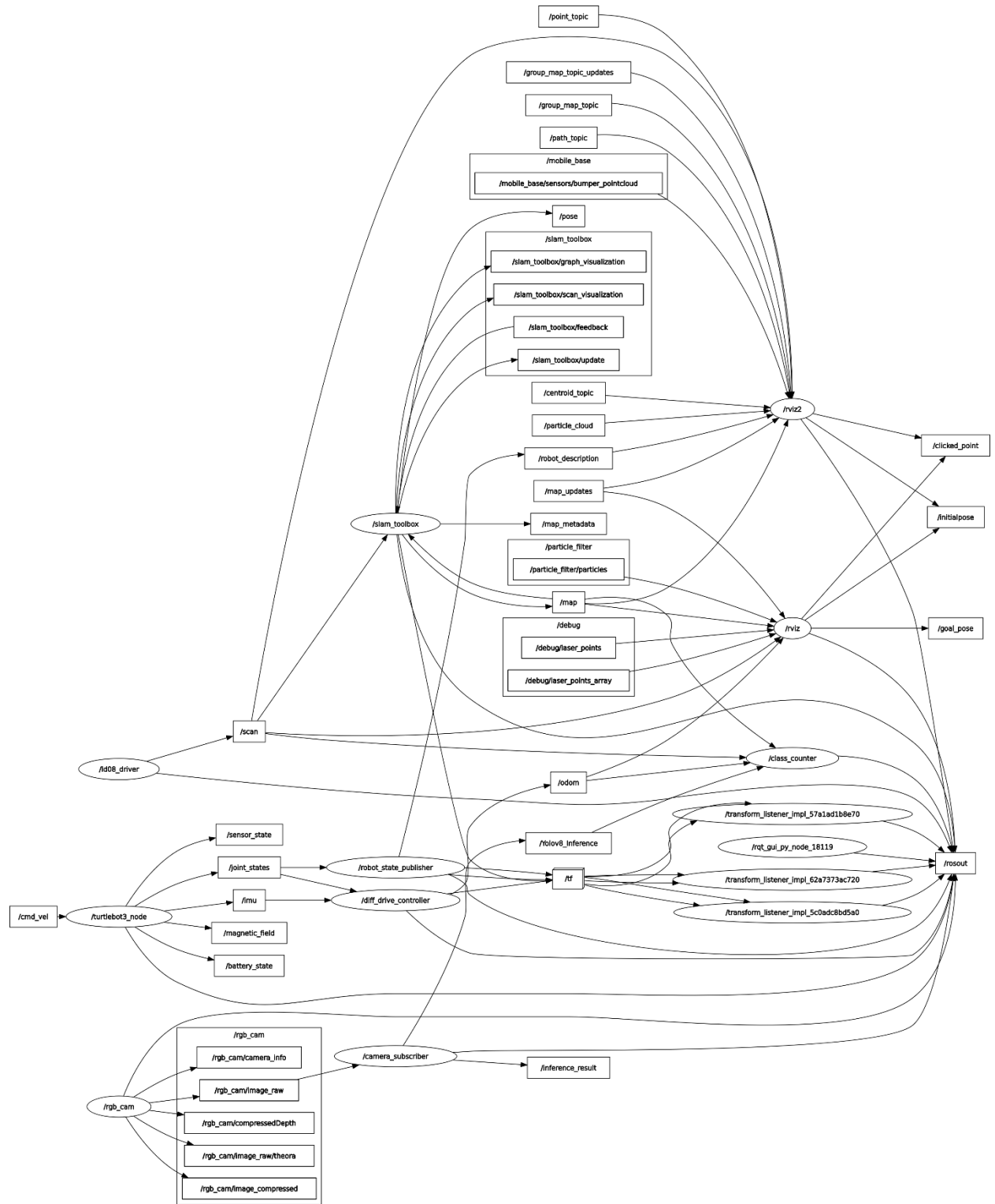
arms@arms-Legion-5-15IAH7H: ~/turtlebot3 frontier based ws 58x27
tonomous_exploration/lib/python3.10/site-packages/autonomo
us_exploration/control.py", line 588, in run_exploration
    publish_cmd_vel(self, v, w)
  File "/home/arms/turtlebot3_frontier_based_ws/install/au
tonomous_exploration/lib/python3.10/site-packages/autonomo
us_exploration/control.py", line 401, in publish_cmd_vel
    self.publisher.publish(twist)
  File "/opt/ros/humble/local/lib/python3.10/dist-packages
/rclpy/publisher.py", line 70, in publish
    self.__publisher.publish(msg)
rclpy._rclpy_pybind11.RCLError: Failed to publish: publish
er's context is invalid, at ./src/rcl/publisher.c:389
arms@arms-Legion-5-15IAH7H:~/turtlebot3_frontier_based_ws$
arms@arms-Legion-5-15IAH7H:~/turtlebot3_frontier_based_ws$
source install/setup.bash
arms@arms-Legion-5-15IAH7H:~/turtlebot3_frontier_based_ws$
ros2 run autonomous_exploration control
/usr/lib/python3/dist-packages/scipy/__init__.py:146: User
Warning: A NumPy version >=1.17.3 and <1.25.0 is required
for this version of SciPy (detected version 1.26.2
  warnings.warn(f"A NumPy version >={np_minversion} and <{
np_maxversion}")
Initialization done. Start Thread
Wait for data...
Wait for data...
Wait for data...

```


RQT_graph (full):



RQT_graph (without autonomous_exploration node):



3) Algorithm for Counting Images Using Python

In order to perform the task, we created a separate package for counting and modified the existing Nadja4 Frontier-based Exploration package to stop the robot when desired. The image_count package can be found here: https://github.com/Jotlane/image_count

3.1 Location-based counting using LIDAR

For our final attempt, we created a ROS2 package that utilises the following subscriptions.

Subscriptions:

1. /Yolov8_Inference (type: Yolov8Inference)
 - a. The detected classes of
2. map (type: OccupancyGrid)
 - a. Occupancy Grid information, including map origin, which robot odometry is relative to
3. odom (type: Odometry)
 - a. Robot odometry, including x/y/yaw relative to the map origin
4. scan (type: LaserScan)
 - a. LIDAR, including distance between the robot and the first obstacle encountered at each angle

With these subscriptions, the callback functions store the latest data but do not act on it, with the exception being the callback to the /Yolov8_Inference topic. When it receives a message from that topic, it will first process the center of the bounding box to determine the location of the object on the screen. To avoid counting images outside of the maze, a height limit is implemented to filter them out.

Once the image is confirmed to be within the maze, the distance between the robot and the nearest object in front of it is obtained from the latest stored LaserScan data. Using trigonometry with the distance and the latest stored robot odometry data, the position of the detected object relative to the map origin can be obtained.

Sending this data to the add_to_count function, we determine if the new object should be counted as a new object or not. Firstly, the class is checked against a list of known classes to ensure that other objects are not stored. If this is the first instance that this particular class has been detected, its position is immediately stored and its count is incremented by 1. Otherwise, if there are existing stored positions, the function checks whether the new position is within a certain distance of the old position. If it is, then it is determined that the robot is detecting the same picture, and will not count it again. If it is not, then the robot will treat it as a new image and increment the count by 1. Regardless, this new detected position will be stored in the list of

stored positions for that particular object, in order to reduce the chances of double counting the object.

Each time an object's count increases, it prints to the terminal the current counted values for each object. The latest print contains the final count for the exploration.

3.2 Stop sign detection

As the Nadja4 Frontier-based Exploration package handled the movement of the robot, it would have to be modified in order to get the robot to stop at a desired position. By adding a subscriber to the /Yolov8_Inference topic, we can obtain the class names of detected objects. A callback function runs upon receiving a message, which checks if the class name is "stop sign". If it is, it sends the running_state to be equal to 5. This ends the robot exploration, as if it has fully explored a location. To prevent the robot from following the previously sent cmd_vel, a new cmd_vel of 0 velocity and 0 angular velocity is sent, ensuring that the robot comes to a full stop.

3.3 Location-based counting using Occupancy Grid

Our first attempt, which we decided not to pursue further, used the occupancy grid data to determine the positions of the detected image instead of the LIDAR data. We decided to abandon this attempt as there was insufficient data; the map origin yaw was always being published as 0 when it may not necessarily be. The algorithm for the attempt is as described below.

Subscriptions:

1. /Yolov8_Inference (type: Yolov8Inference)
 - a. The detected classes of
2. map (type: OccupancyGrid)
 - a. Occupancy Grid information, including map origin, which robot odometry is relative to
3. odom (type: Odometry)
 - a. Robot odometry, including x/y/yaw relative to the map origin
4. scan (type: LaserScan)
 - a. LIDAR, including distance between the robot and the first obstacle encountered at each angle

With these subscriptions, the callback functions store the latest data but do not act on it, with the exception being the callback to the /Yolov8_Inference topic. When it receives a message from that topic, it will first process the center of the bounding box to determine the location of the object on the screen. To avoid counting images outside of the maze, a height limit is implemented to filter them out.

Once the image is confirmed to be within the maze, the distance between the robot and the nearest object in front of it is obtained from the latest stored OccupancyGrid data. Using trigonometry with the robot odometry and the map position and yaw, a line can be “drawn” from the robot’s location to the wall in the direction that the robot is facing. This was done using a custom implementation of Bresenham’s Line Algorithm, normally used for drawing lines in computer graphics. With each “step” taken down the line, the function checks if that pixel is a wall on the occupancy grid. If so, it will mark it as the location of the image. Otherwise, it will continue going down the line. Once an image is discovered, its position and class are sent to the `add_to_count` function.

Sending this data to the `add_to_count` function, we determine if the new object should be counted as a new object or not. Firstly, the class is checked against a list of known classes to ensure that other objects are not stored. If this is the first instance that this particular class has been detected, its position is immediately stored and its count is incremented by 1. Otherwise, if there are existing stored positions, the function checks whether the new position is within a certain distance of the old position. If it is, then it is determined that the robot is detecting the same picture, and will not count it again. If it is not, then the robot will treat it as a new image and increment the count by 1. Regardless, this new detected position will be stored in the list of stored positions for that particular object, in order to reduce the chances of double counting the object.

Each time an object’s count increases, it prints to the terminal the current counted values for each object. The latest print contains the final count for the exploration.

4) Tuned Frontier_Exploration parameters

Here, there are the 5 parameters with their default values:

lookahead_distance : 0.24

speed : 0.1 #max speed

expansion_size : 2 #wall expansion coefficient

target_error : 0.15 #margin of error to target

min_distance_to_obstacles : 0.2 #robot distance for local security

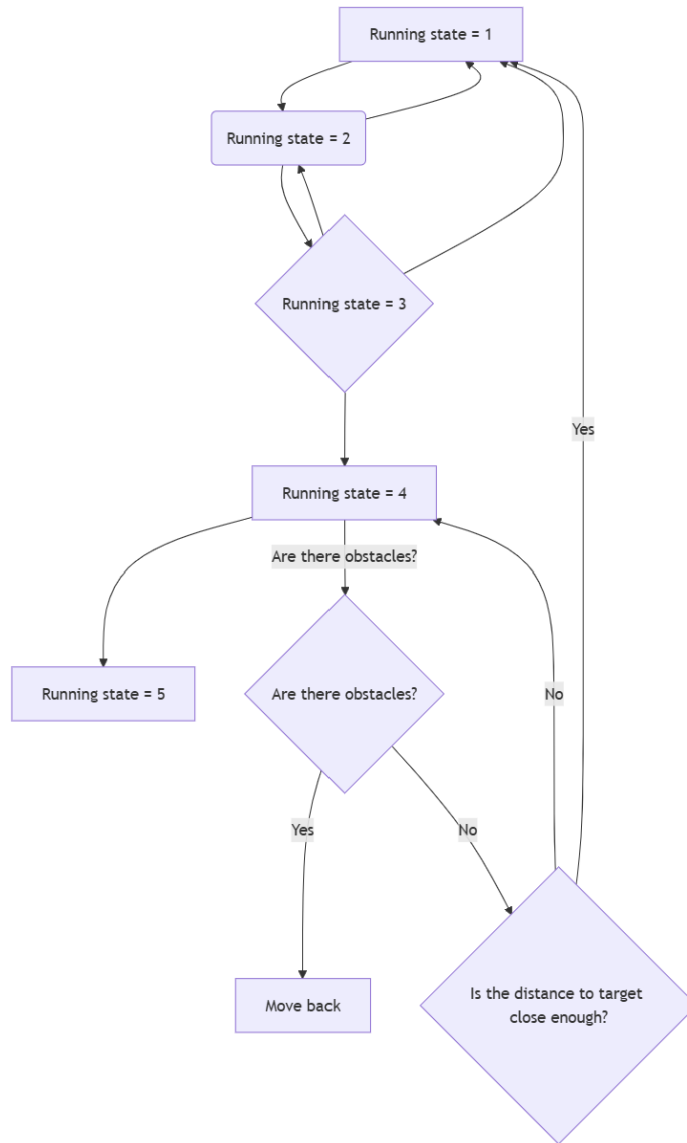
For the project, only the ‘lookahead_distance’ parameter was changed as the other parameters are irrelevant for us to meet the objectives listed under 1) Introduction. The lookahead_distance is the radius of a circle that extends from the TurtleBot3 robot. The robot

creates equidistant waypoints, where some of the waypoints lie within the circle and are thereby ignored. The next waypoint that lies outside the circle will be the next waypoint that the TurtleBot3 moves towards. We increased our lookahead distance to 0.3m as it enabled the robot to have a slightly smoother trajectory when pathfinding using the waypoints.

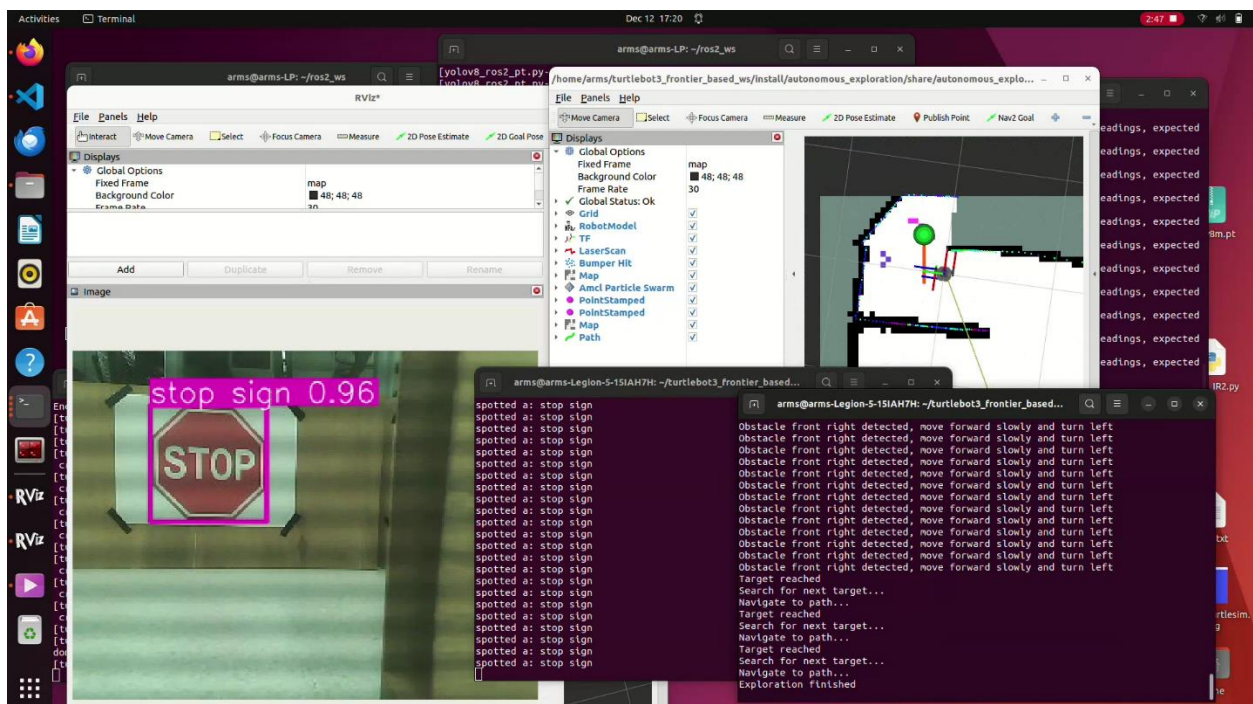
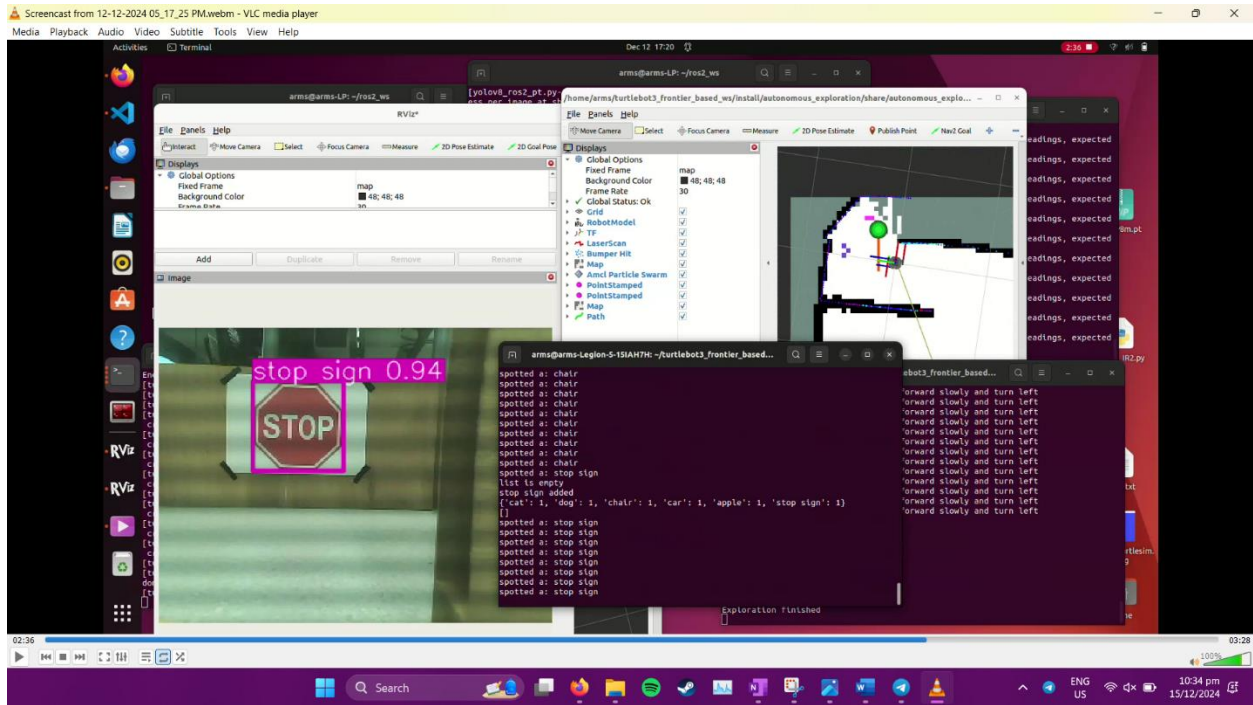
To figure out the other relevant parameters to change, we analysed the code in the control.py file in the frontier_exploration package. To summarise, the code works as follows:

- **State 1: Preparing the Map**
 - Calculates the robot's position on the map.
 - Groups unexplored areas ("targets") to decide where to explore next.
 - If there are no groups, exploration ends.
- **State 2: Choosing a Target**
 - Selects the nearest unexplored area as the target and plans a path to it.
 - If planning fails, it retries up to 5 times before stopping exploration.
- **State 3: Path Planning**
 - Smoothens the calculated path for better movement.
 - Publishes the target point and path to other parts of the system.
- **State 4: Navigating to the Target**
 - Moves along the planned path using a "pure pursuit" algorithm.
 - Handles obstacles by stopping and recalculating the path if necessary.
 - If the target is reached, it resets to plan the next exploration.
- **State 5: Exit**
 - Stops the robot and ends the program.

The following figure shows how the code above works:



5) Results with images



6) A video (separate files consisting of screen recording & camera) illustrating the Performance.

External recording: https://youtu.be/yrHY_x0VieU



Screen recording: <https://youtu.be/wB-KBI7VnYw>

7) References

- [1] A. Sojasingarayar, "Faster R-CNN vs YOLO vs SSD — Object Detection Algorithms," *IBM Data Science in Practice*, Medium. Available: <https://medium.com/ibm-data-science-in-practice/faster-r-cnn-vs-yolo-vs-ssd-object-detection-algorithms>. [Accessed: Dec. 10, 2024].
- [2] "YOLO Object Detection Explained: A Beginner's Guide," *DataCamp*. Available: <https://www.datacamp.com/tutorial/yolo-object-detection>. [Accessed: Dec. 10, 2024].
- [3] R. Jayarajan, "GitHub - rohithjayarajan/frontier_exploration: Package for Frontier Exploration Robot developed as the final project for the course ENPM808X at the University of Maryland, College Park. The application of this project is to enable a mobile robot to autonomously map an unknown environment," GitHub. [Online]. Available: https://github.com/rohithjayarajan/frontier_exploration. [Accessed: 12-Dec-2024].