ROS 2 TurtleBot3 Burger: Vision-Based Lane Following, SLAM Mapping, and Navigation

Adel KANSO Mariam JANBEIN

Supervisor: Prof. Joaquin Jorge Rodriguez

Abstract

This project implements a complete pipeline on a TurtleBot3 Burger using ROS 2: (1) camera calibration and bird's-eye image projection, (2) robust lane detection of a two-color track (yellow left boundary, white right boundary), (3) closed-loop lane following to complete the track, (4) SLAM map generation using Cartographer (LiDAR-based), (5) pose logging using TF transforms, and (6) conversion of the recorded trajectory into a processed map suitable for navigation experiments using Navigation2. A key objective was reproducibility: after reading this report, a reader should be able to rebuild the same pipeline, tune parameters, and understand the main pitfalls we encountered (lighting sensitivity, scaling issues, and intermittent connectivity errors).

Contents

1 Introduction

Autonomous mobile robots commonly rely on either vision (for structured environments such as lanes/lines) or SLAM (for map-based localization and navigation). In this work we combined both approaches on a TurtleBot3 Burger:

- Vision-based lane following for completing a closed-loop track using color cues.
- LiDAR-based SLAM for map creation and later Navigation2 trials.
- Trajectory logging and offline map post-processing to attempt generating a simplified road-only map.

2 Platform and Tools

2.1 Hardware

- Robot: TurtleBot3 Burger
- Sensors used: RGB camera, 2D LiDAR

2.2 Software

- ROS 2 (TurtleBot3 stack)
- Cartographer for SLAM mapping (selected instead of SLAM Toolbox)
- Navigation2 (Nav2) for navigation experiments
- OpenCV for image processing (projection + lane detection)
- TF2 for robot pose extraction (map → base_link)

2.3 External Resources Consulted

We used the official TurtleBot3 e-manual, community discussions (e.g., StackOverflow), and tutorial videos (e.g., YouTube) for setup guidance and troubleshooting.

3 System Overview

Figure 1 summarizes the main data flow.

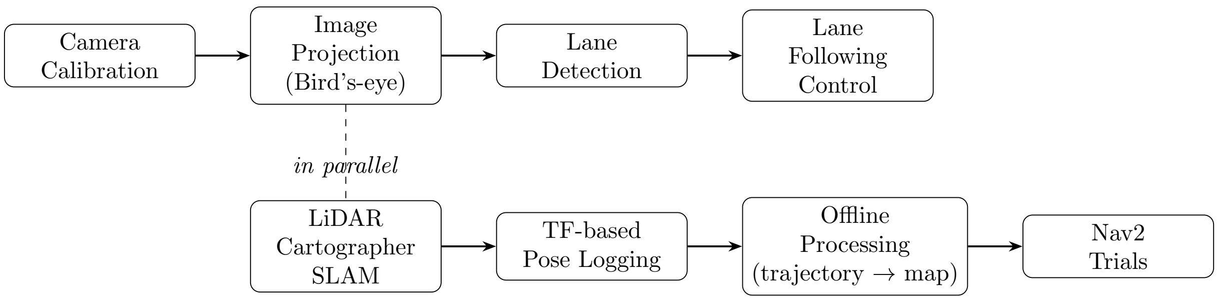


Figure 1: High-level pipeline: camera calibration → image projection → lane detection → lane following; and in parallel LiDAR Cartographer SLAM → TF-based pose logging → offline processing → Nav2 trials.

4 Camera Calibration

4.1 Objective

Calibration was required to correct lens distortion and improve the geometric consistency of the projected (top-down) image. We used a checkerboard pattern with a 9×7 inner-corner grid and a square size of 0.023 m.

4.2 Procedure

1. Place the checkerboard in front of the robot camera.
2. Collect images while varying distance and orientation (tilt/roll/pan) to cover the camera field of view.
3. Run the ROS 2 camera calibration tool configured for the checkerboard geometry and remapped to the robot camera topic.
4. Save the resulting intrinsic parameters and verify by checking straight-line consistency in the undistorted stream.

4.3 Calibration Output and Verification

We verified calibration quality by visually checking that:

- Distortion near the image edges was reduced.
- The checkerboard corners aligned consistently across frames.

Figure placeholder: include an example calibration view and/or undistorted stream.

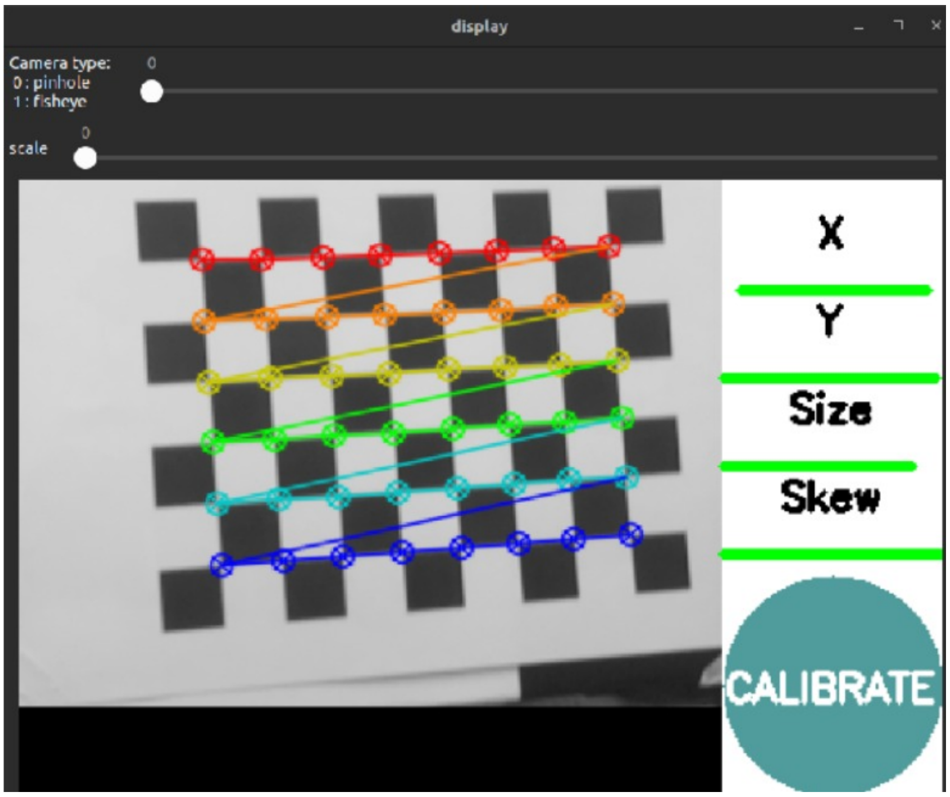


Figure 2: Example camera calibration view (checkerboard detection).

5 Bird's-Eye Image Projection (Homography)

5.1 Objective

Lane detection is easier in a bird's-eye view where lane boundaries become approximately parallel and changes in perspective are reduced. We implemented an image projection node that:

- selects a trapezoidal region of interest (ROI) in the original image,
- applies a homography transform to warp it into a rectangular top-down view,
- publishes both a calibration visualization and a warped output image.

5.2 Projection Parameters and Tuning

The ROI trapezoid is defined around the image center and uses four main parameters (top/bottom widths and vertical offsets), plus two additional offsets to shift the trapezoid when needed. Table 1 lists the final parameters used.

Table 1: Final image projection parameters used in our setup.

Parameter	Value
top_x	144
top_y	41
bottom_x	326
bottom_y	129
offset_x	−45
offset_y	160

5.3 Practical Notes

- **Offsets were essential:** the track was not always centered in the camera; shifting the ROI using offset_x/offset_y allowed aligning the trapezoid with the visible lane region.
- **Verification:** we displayed the trapezoid overlay and confirmed that the warped output produced stable lane geometry.

Figure: Projected Image

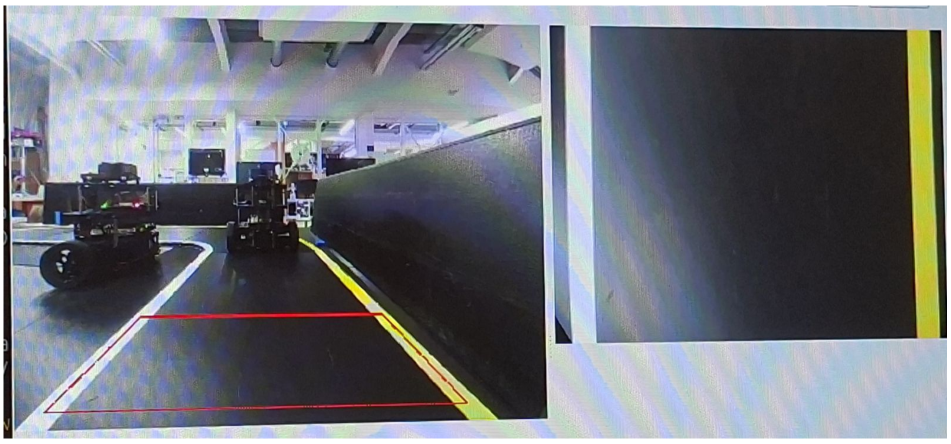


Figure 3: Projected Image.

6 Lane Detection

6.1 Goal

The track was defined by:

- **Yellow line** on the left boundary
- **White line** on the right boundary

We performed color-based segmentation in HSV space to generate binary masks for each lane boundary and then estimated lane curves.

6.2 HSV Thresholds

Lighting conditions strongly influenced detection performance. We tuned thresholds repeatedly because the working environment had changing illumination during the day. Table 2 lists the final parameters that performed best in our test area.

Table 2: Final HSV threshold parameters for lane segmentation.

Lane	Hue_L	Hue_H	Sat_L	Sat_H	Light_L	Light_H
White	0	163	0	58	195	255
Yellow	0	65	62	255	142	255

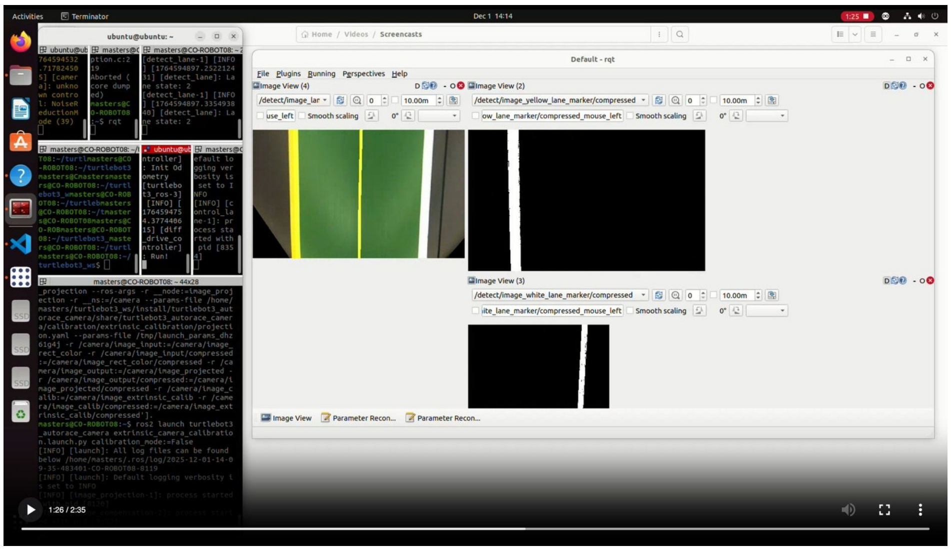
6.3 Detection Pipeline

1. Convert projected image from BGR to HSV.
2. Apply thresholding to obtain binary masks for yellow and white lanes.
3. Estimate lane geometry using polynomial fitting; fall back to a sliding-window method when tracking fails.
4. Compute a centerline:
   - If both lanes are reliable: centerline is the mean of left and right lane curves.
   - If only one lane is reliable: centerline is estimated using a fixed offset from the detected lane (empirically tuned).

6.4 Operational Challenge: Lighting Variations

This stage required the most frequent tuning. Small changes in shadows and reflections altered HSV distributions, making lane segmentation unstable. In practice, parameter adjustment became a recurring daily task.

Figure: lane masks and final overlay.



```
764594532
.71782450
5] [camer
a]: unkno
wn contro
l: NoiseR
eductionM
ode (39)
```

```
ption.c:2
19
Aborted (
core dump
ed)
masters@C
O-ROBOT08
:~$ rqt
```

```
[detect_lane-1] [INFO
] [1764594897.2522124
31] [detect_lane]: La
ne state: 2
[detect_lane-1] [INFO
] [1764594897.3354938
40] [detect_lane]: La
ne state: 2
```

```
T08:~/turtlmasters@CO
-ROBOT08:~/turtlebot3
masters@Cmastersmaste
rs@CO-ROBOT08:~/turtl
ebot3_wmasters@CO-ROB
OT08:~/turtlebmasters
@CO-ROBOT08:~/tmaster
s@CO-ROBOT08masters@C
O-ROBmasters@CO-ROBOT
08:~/turtlebot3_maste
rs@CO-ROBOT08:~/turtl
masters@CO-ROBOT08:~/
turtlebot3_ws$
```

```
ntroller]
: Init Od
ometry
[turtlebo
t3_ros-3]
 [INFO] [
176459475
4.3774406
15] [diff
_drive_co
ntroller]
: Run!
```

```
efault lo
gging ver
bosity is
 set to I
NFO
[INFO] [c
ontrol_la
ne-1]: pr
ocess sta
rted with
 pid [835
4]
```

```
_projection --ros-args -r __node:=image_proj
ection -r __ns:=/camera --params-file /home/
masters/turtlebot3_ws/install/turtlebot3_aut
orace_camera/share/turtlebot3_autorace_camer
a/calibration/extrinsic_calibration/projecti
on.yaml --params-file /tmp/launch_params_dhz
61g4j -r /camera/image_input:=/camera/image_
rect_color -r /camera/image_input/compressed
:=/camera/image_rect_color/compressed -r /ca
mera/image_output:=/camera/image_projected -
r /camera/image_output/compressed:=/camera/i
mage_projected/compressed -r /camera/image_c
alib:=/camera/image_extrinsic_calib -r /came
ra/image_calib/compressed:=/camera/image_ext
rinsic_calib/compressed'].
masters@CO-ROBOT08:~$ ros2 launch turtlebot3
_autorace_camera extrinsic_camera_calibratio
n.launch.py calibration_mode:=False
[INFO] [launch]: All log files can be found
below /home/masters/.ros/log/2025-12-01-14-0
9-35-483401-CO-ROBOT08-8119
[INFO] [launch]: Default logging verbosity i
s set to INFO
[INFO] [image_projection-1]: process started
```

Figure 4: Lane detection result: overlay with estimated centerline (computed from yellow and white lane masks after HSV thresholding).

7 Lane Following Control and Track Completion

7.1 Control Objective

Using the detected centerline, the robot was commanded to minimize lateral error relative to the desired center of the lane in the projected image.

7.2 Major Debugging Issue Encountered

A significant issue occurred after projection tuning: even with a visually correct projected image, the robot consistently drifted and followed the yellow lane rather than the centerline. After extensive troubleshooting, we identified that the cause was a **bug in the control/detection integration code**. We resolved it by recreating the relevant code module from scratch, after which the robot correctly tracked the lane center.

7.3 Result

After fixing the bug and repeatedly tuning HSV thresholds, we successfully ran the robot around the full track using only lane detection and closed-loop control.

8 SLAM Mapping with Cartographer

8.1 Objective

We generated a 2D SLAM map using LiDAR and Cartographer. We explicitly chose Cartographer rather than SLAM Toolbox to match the course direction and our available configuration.

8.2 Procedure

1. Launch Cartographer for TurtleBot3.
2. Teleoperate the robot to cover the environment and close loops where possible.
3. Save the final occupancy grid map (PGM + YAML metadata).

Figure placeholder: final SLAM map.

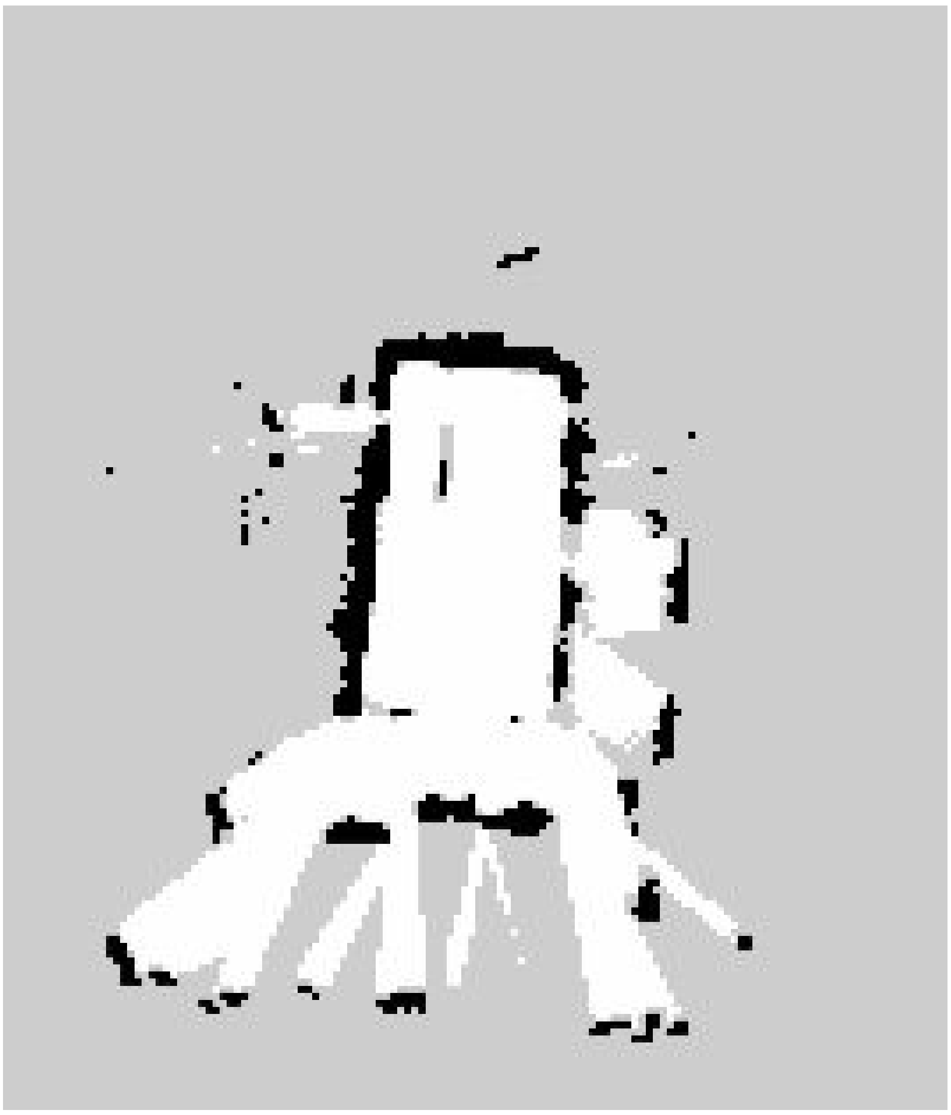


Figure 5: Example SLAM map generated with Cartographer (LiDAR-based).

9 Pose Logging Using TF

9.1 Motivation

Initially, we tried logging the robot pose from odometry, but after guidance from Prof. Rodriguez, we changed to TF-based logging because odometry was too noisy over long runs. We attempted an alternative pose source (likely SLAM-related), but it only provided the last pose and did not log a full trajectory as required.

9.2 Method

We logged pose at a fixed period by querying TF transforms from `map` to `base_link` and recording:

- time stamp,
- x, y position,
- yaw angle.

Data were stored in CSV format and later used for offline visualization and map post-processing.

9.3 Example Output Format

The CSV format used:

time	x	y	yaw
...	...	...	...

10 Offline Map Processing: From Trajectory to Road-Only Map

10.1 Goal

We attempted to convert the recorded trajectory into a simplified map that contains only the drivable road (track) for easier navigation planning. The workflow was:

1. Draw the trajectory over the SLAM PGM map using map resolution and origin from YAML. **Figure placeholder:** final SLAM map.

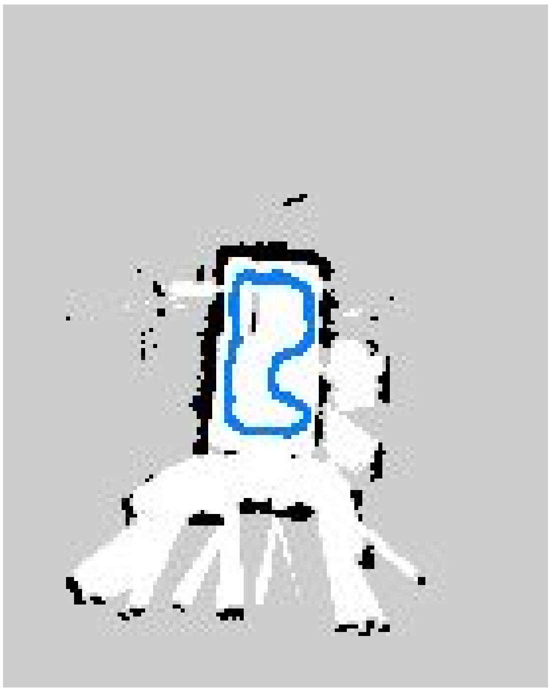


Figure 6: Example SLAM map generated with Cartographer (LiDAR-based).

2. Extract only the drawn road pixels and remove other content.

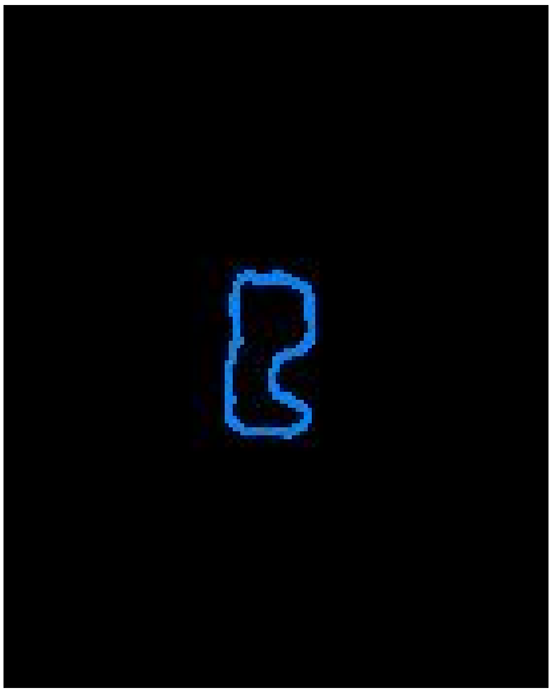


Figure 7: Example SLAM map generated with Cartographer (LiDAR-based).

3. Estimate road width in pixels from the extracted road image.
4. Use measured real road width to compute a new map resolution in meters per pixel.
5. Scale the road-only image

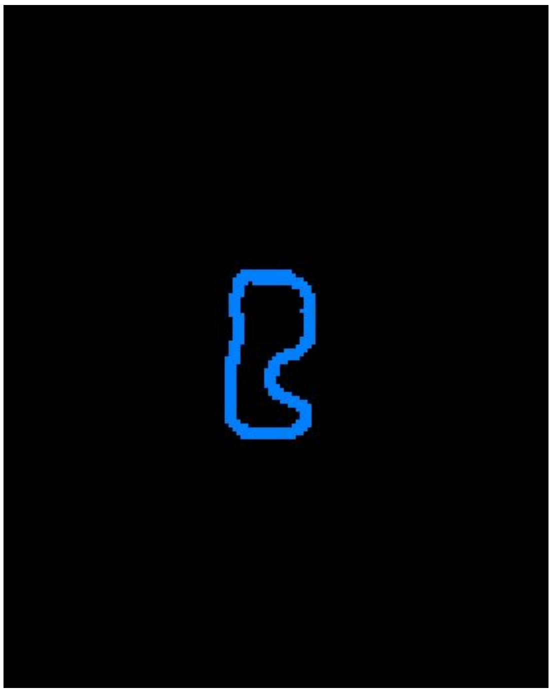


Figure 8: Example SLAM map generated with Cartographer (LiDAR-based).

6. Regenerate a centered PGM+YAML map.

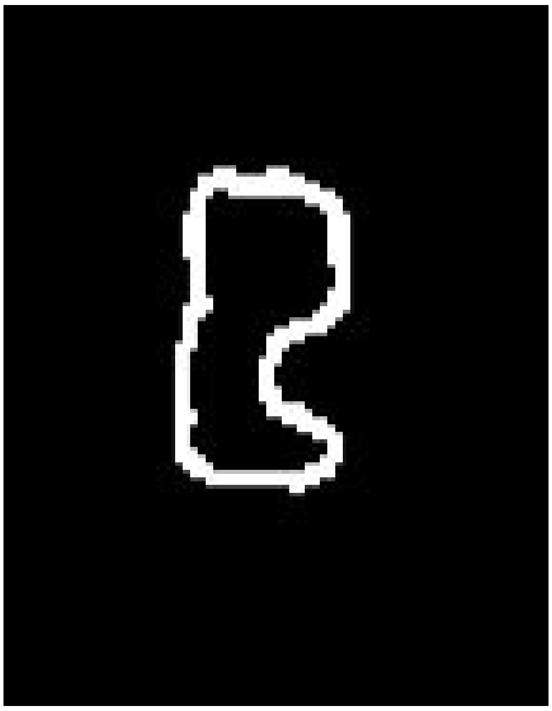


Figure 9: Example SLAM map generated with Cartographer (LiDAR-based).

10.2 Road Width Measurement and Resolution Estimation

We measured the physical road width as 26 cm. From the extracted road image, the road width was estimated as 13.17 pixels, producing a computed resolution of:

$$\text{resolution} \approx \frac{0.26}{13.17} \approx 0.0197\,\text{m/pixel}.$$

However, using this computed resolution caused a practical mismatch in Navigation2: the road became too narrow relative to the robot footprint, resulting in navigation failures. Due to measurement and scaling uncertainty, we used a coarser resolution of 0.0400 m/pixel in our generated map, but this introduced scaling drift for distant goals. We had an idea of trying to draw the road on the map, and try to modify this map exactly in this case no measurement step calculation needed, and it will be the perfect result of measurement compatibility.

10.3 Artifacts Produced

- Trajectory-overlay image (trajectory drawn on SLAM map)
- Road-only image (background removed)
- Scaled road-only image (nearest-neighbor scaling)
- Generated road PGM map with centered origin and chosen resolution

Figure placeholders: trajectory overlay and road-only extraction.

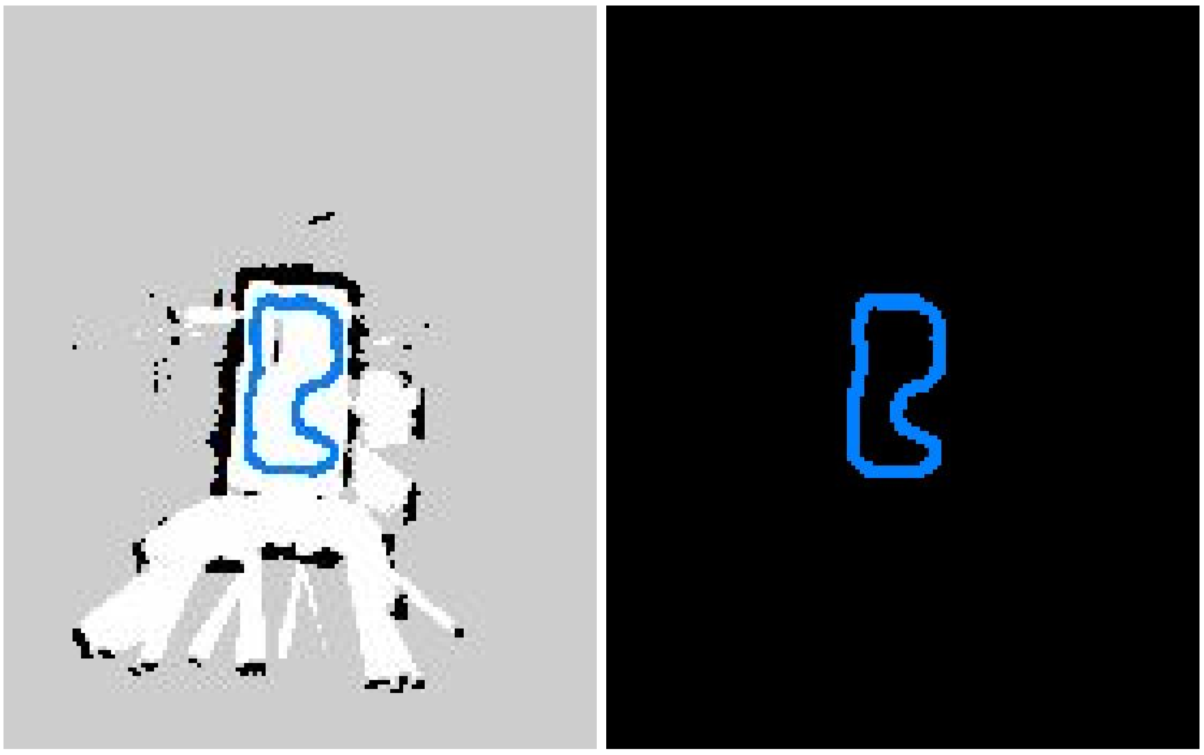


Figure 10: Left: trajectory plotted on the SLAM map. Right: extracted road-only image.

11 Navigation2 Experiments

11.1 Setup

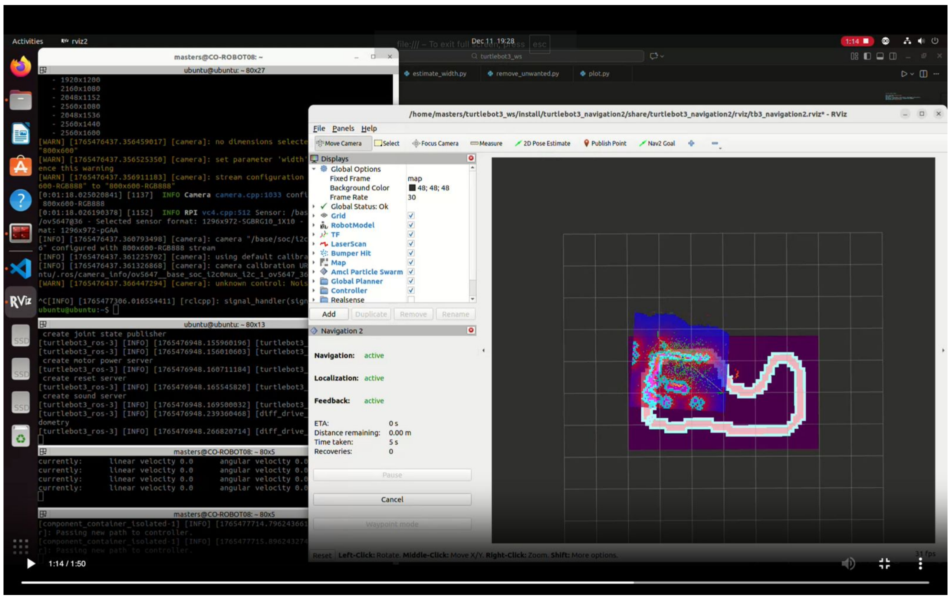
Using the generated map (PGM+YAML), we ran Navigation2 and performed:

- two pose initialization/estimation steps (initial alignment and refinement),
- manual motion using keyboard teleoperation to confirm localization consistency,
- goal-based navigation tests by sending target poses.

11.2 Observed Behavior

Navigation worked for short movements but failed when targeting far goals. The main reason was **incorrect map scaling** (resolution and/or origin mismatch), which caused accumulating localization/planning errors at longer distances.

Figure placeholder: Nav2 visualization (RViz) with map, costmaps, and goal.



```
ubuntu@ubuntu: ~ 80x27
   - 1920x1200
   - 2160x1080
   - 2048x1152
   - 2560x1080
   - 2048x1536
   - 2560x1440
   - 2560x1600
[WARN] [1765476437.356459017] [camera]: no dimensions selecte
"800x600"
[WARN] [1765476437.356525350] [camera]: set parameter 'width'
ence this warning
[WARN] [1765476437.356911183] [camera]: stream configuration
600-RGB888" to "800x600-RGB888"
[0:01:18.025020841] [1137]  INFO Camera camera.cpp:1033 confi
 800x600-RGB888
[0:01:18.026190378] [1152]  INFO RPI vc4.cpp:512 Sensor: /bas
/ov5647@36 - Selected sensor format: 1296x972-SGBRG10_1X10 -
mat: 1296x972-pGAA
[INFO] [1765476437.360793498] [camera]: camera "/base/soc/i2c
6" configured with 800x600-RGB888 stream
[INFO] [1765476437.361225702] [camera]: using default calibra
[INFO] [1765476437.361326868] [camera]: camera calibration UR
ntu/.ros/camera_info/ov5647__base_soc_i2c0mux_i2c_1_ov5647_36
[WARN] [1765476437.366447294] [camera]: unknown control: Nois

^C[INFO] [1765477306.016554411] [rclcpp]: signal_handler(sign
ubuntu@ubuntu:~$ 
```

```
ubuntu@ubuntu: ~ 80x13
 create joint state publisher
[turtlebot3_ros-3] [INFO] [1765476948.155960196] [turtlebot3_
[turtlebot3_ros-3] [INFO] [1765476948.156010603] [turtlebot3_
 create motor power server
[turtlebot3_ros-3] [INFO] [1765476948.160711184] [turtlebot3_
 create reset server
[turtlebot3_ros-3] [INFO] [1765476948.165545820] [turtlebot3_
 create sound server
[turtlebot3_ros-3] [INFO] [1765476948.169500032] [turtlebot3_
[turtlebot3_ros-3] [INFO] [1765476948.239360468] [diff_drive_
dometry
[turtlebot3_ros-3] [INFO] [1765476948.266820714] [diff_drive_
```

```
masters@CO-ROBOT08: ~ 80x5
currently:      linear velocity 0.0      angular velocity 0.0
currently:      linear velocity 0.0      angular velocity 0.0
currently:      linear velocity 0.0      angular velocity 0.0
currently:      linear velocity 0.0      angular velocity 0.0
```

```
masters@CO-ROBOT08: ~ 80x5
[component_container_isolated-1] [INFO] [1765477714.796243661
r]: Passing new path to controller.
[component_container_isolated-1] [INFO] [1765477715.896243274
r]: Passing new path to controller.
```

Displays
- Global Options
  - Fixed Frame: map
  - Background Color: ■ 48; 48; 48
  - Frame Rate: 30
- ✓ Global Status: Ok
- Grid ✓
- RobotModel ✓
- TF ✓
- LaserScan ✓
- Bumper Hit ✓
- Map ✓
- Amcl Particle Swarm ✓
- Global Planner ✓
- Controller ✓
- Realsense ☐

Navigation 2

Navigation: active

Localization: active

Feedback: active

ETA:	0 s
Distance remaining:	0.00 m
Time taken:	5 s
Recoveries:	0

Reset **Left-Click:** Rotate. **Middle-Click:** Move X/Y. **Right-Click:** Zoom. **Shift:** More options.

Figure 11: Navigation2 test: localized robot and goal target on the generated map.

12 Additional Challenges and Troubleshooting

- **Intermittent camera detection errors:** sometimes the camera node failed with "camera not found"-type issues. Restarting the robot or relaunching nodes typically resolved the problem.
- **Unreliable connection to the robot:** occasional refusal to connect disrupted experiments; restarting network components and relaunching helped.
- **Environment variability:** changing lighting caused frequent HSV retuning for lane detection.
- **Time loss due to hidden software bug:** the centerline-following bug led to extensive debugging time despite correct projection output.

13 Future Work

13.1 QR Code Lap Completion Marker

Prof. Rodriguez suggested using a QR marker to detect lap completion and switch robot behavior (e.g., exploring vs. path planning). The intended marker family was `dict_6x6_50`. We started implementing detection using the robot camera stream, but we did not finish due to time constraints. Automating lap completion remains an important improvement.

13.2 Map Scaling and Metric Consistency

To make Navigation2 reliable for long-range goals, future work should:

- improve the physical-to-pixel calibration (more precise measurements and controlled extraction thickness),
- validate resolution by comparing multiple known distances,
- ensure robot footprint and inflation parameters match the map scale.

13.3 Robust Lane Detection

To reduce daily retuning:

- use adaptive thresholding or illumination normalization,
- incorporate morphological filtering and temporal smoothing,
- consider learning-based segmentation if allowed.

14 Conclusion

We implemented and validated a full ROS 2 pipeline on TurtleBot3 Burger including camera calibration, bird's-eye projection, HSV-based lane detection, and lane following that completed the full track. We also generated a LiDAR-based SLAM map using Cartographer, logged trajectory using TF transforms, and built an offline workflow to extract a road-only map for Navigation2. The primary limitations were operational instability (intermittent device/connection errors), sensitivity to lighting for lane detection, and map scaling errors that prevented consistent long-distance navigation goals. Despite these issues, the project demonstrated an end-to-end autonomous workflow and identified clear steps for further improvements.