Journey with

ROS2
ROBOMAPPER ODYSSEY

ANUSHAA B
CS24BT032

THE START: COMPATIBILITY!

I spun up Ubuntu 24.04 without thinking twice — it was new, clean, and already on my drive. What I didn't check was whether ROS 2 Humble actually played nice with it. (It doesn't.)

Errors started popping up. I wasn't sure if I was the problem, or if Jazzy and 24.04 was a bad idea, because resources? Similar error experiences? With Humble not Jazzy.

Eventually, I gave up the high ground of stubbornness and set up 22.04 + Humble.

Same errors.
But this time? They felt honest. Predictable. My own bugs. No doubts if they could ever be fixed.

Turns out, Jazzy wasn't a mistake — just not the easiest place to begin!
It wasn't a dead end. Just a different kind of maze.

Would I start that way again? Probably not.
Would I try Jazzy again? I mean why not!
Would I trade what I learned? Definitely not.

PROJECT OVERVIEW

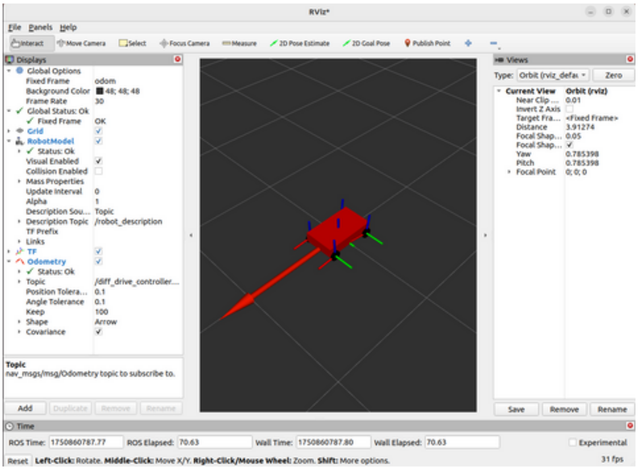
4-Wheeled Differential Drive Robot in ROS 2 Humble

Designed and simulated a custom 4-wheeled robot in ROS 2 Humble, using Gazebo for simulation and RViz2 for visualization. The robot is controlled using a 2-wheel differential drive setup (front-wheel drive, rear-wheel passive). The entire system has been successfully tested and can be manually driven using a keyboard teleoperation node

KEY COMPONENTS AND FEATURE

1. Custom URDF Robot Model
   Robot structure:
   - A central base_link body.
   - Four wheels: front_left, front_right, rear_left, and rear_right.
   - Only the front wheels are actuated and participate in differential drive.
   - Rear wheels are passive for support.
   - All wheels are correctly placed, connected via joints, and rotate realistically.

2. Visualization in RViz2
RViz2 is used for monitoring:
   - The robot model (/robot_description)
   - Live TF frames
   - Odometry data published by the controller
   - Arrow visuals show real-time movement direction and speed.



Global Options	
Fixed Frame	odom
Background Color	■ 48; 48; 48
Frame Rate	30
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
RobotModel	✓
✓ Status: Ok	
Visual Enabled	✓
Collision Enabled	☐
Mass Properties	
Update Interval	0
Alpha	1
Description Sou...	Topic
Description Topic	/robot_description
TF Prefix	
Links	
TF	✓
Odometry	✓
✓ Status: Ok	
Topic	/diff_drive_controller....
Position Tolera...	0.1
Angle Tolerance	0.1
Keep	100
Shape	Arrow
Covariance	✓

Topic
nav_msgs/msg/Odometry topic to subscribe to.

Type: Orbit (rviz_defau | Zero

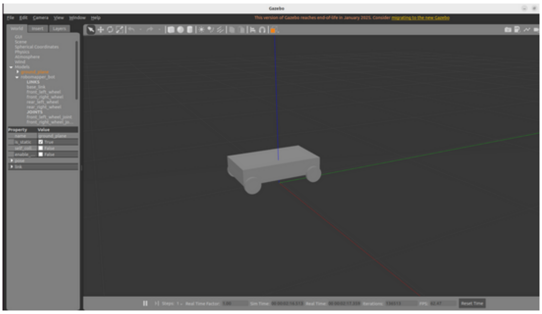
Current View	Orbit (rviz)
Near Clip ...	0.01
Invert Z Axis	☐
Target Fra...	<Fixed Frame>
Distance	3.91274
Focal Shap...	0.05
Focal Shap...	✓
Yaw	0.785398
Pitch	0.785398
Focal Point	0; 0; 0

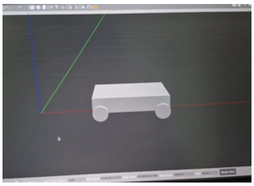
ROS Time: 1750860787.77 ROS Elapsed: 70.63 Wall Time: 1750860787.80 Wall Elapsed: 70.63 ☐ Experimental

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:** Zoom. **Shift:** More options. 31 fps

3. Simulation in Gazebo

- The robot is launched into the Gazebo environment using a custom gazebo.launch.py.
- Uses gazebo_ros2_control for simulating hardware interfaces.
- The model appears as expected and interacts with the simulated environment correctly.





4. ros2_control Integration

Configured with:

- controller_manager.yaml for initializing the controller manager
- diff_drive_controller.yaml for the differential drive controller
- Important parameters include:
- left_wheel_names and right_wheel_names
- Wheel separation, radius, and odometry setup
- Proper interfaces for velocity commands and position feedback
- All parameters passed correctly via --param-file to avoid launch errors.

5. Teleoperation Support

- The robot was successfully tested with the teleop_twist_keyboard package.
- Commands are sent to /diff_drive_controller/cmd_vel_unstamped.
- The robot responds accurately in simulation to keyboard input, enabling manual control.

Current Status

- The robot spawns successfully in Gazebo and RViz2.
- Controllers load correctly and publish odometry.
- TF and joint state data are broadcasting as expected.
- Teleop keyboard control works smoothly, confirming proper integration of the control stack.

APPROACH

I decided to start off by making my own urdf. On second thought that may not have been the best idea. It was time consuming,tackling errors and learning on the go, but now I have my own little bot, even if it’s just a little 4 wheeled cuboid and even if I did not have enough time to implement SLAM.

Challenges Faced and Solutions Implemented

From getting it to simply show up in RViz2 to driving it around in Gazebo under realistic physics, it was all misleading victories and cryptic failures. Here are the major(yes. only major.) issues faced:

1. <u>The Mock Component Mirage (The False Fix)</u>
At one point, I genuinely believed everything was working. The robot spawned in RViz2. The controllers loaded without complaints. No errors in the terminal. I could even send teleop commands. It looked perfect.
The catch? I was using mock_components/GenericSystem — a generic, non-physical hardware interface that doesn’t talk to Gazebo at all. Everything ran cleanly, but nothing actually moved in the simulation.
This was the first real trap: a system that technically launched, but was functionally dead in the water.

2. The Gazebo Reality Check
As soon as I switched from mock components to a proper gazebo_ros2_control interface, the illusion shattered. I was greeted by a wall of errors that had been hiding the whole time. Most notably:

```
[diff_drive_controller]: left_wheel_names cannot be empty
```

And if that got fixed:

```
[FATAL] [ros2_control_node]: Failed to parse parameters: no node names before 'ros__parameters'
```

Turns out, my original single YAML file — which lumped controller manager and diff drive parameters together — had been accepted by the mock system (because it didn't actually care). But now that Gazebo and ros2_control were involved for real, everything broke.

3. The Param File Rebuild
The core issue was that both the controller manager and the diff drive controller were trying to read from the same parameter file, and ROS 2 Humble didn't know how to separate them cleanly. Sometimes it failed hard. Sometimes it just misread values and pretended everything was fine — until it wasn't.
Solution: I split the config into two clean files:
controller_manager.yaml – strictly for controller manager settings
diff_drive_controller.yaml – scoped only to the diff drive plugin

Each file was passed with its own --param-file flag to its respective node. After that, the controller loaded properly, and the robot became responsive in simulation.
The confirmation was satisfying:

```
[controller_manager]: Successfully loaded controller 'diff_drive_controller'
[diff_drive_controller]: Parameters successfully configured
```

4. <u>Missing Physics</u>
Even with control working, simulation didn’t behave realistically. The robot would jitter, drift, or react strangely to even small commands. It quickly became clear that the URDF lacked proper physical definitions.
There were:

- No inertial tags
- No limit or dynamics on joints
- No surface friction or damping

Solution: I enriched the URDF with:

- Accurate inertial properties for every link
- Velocity and effort limits for wheel joints
- Friction and damping parameters using Gazebo <gazebo> tags

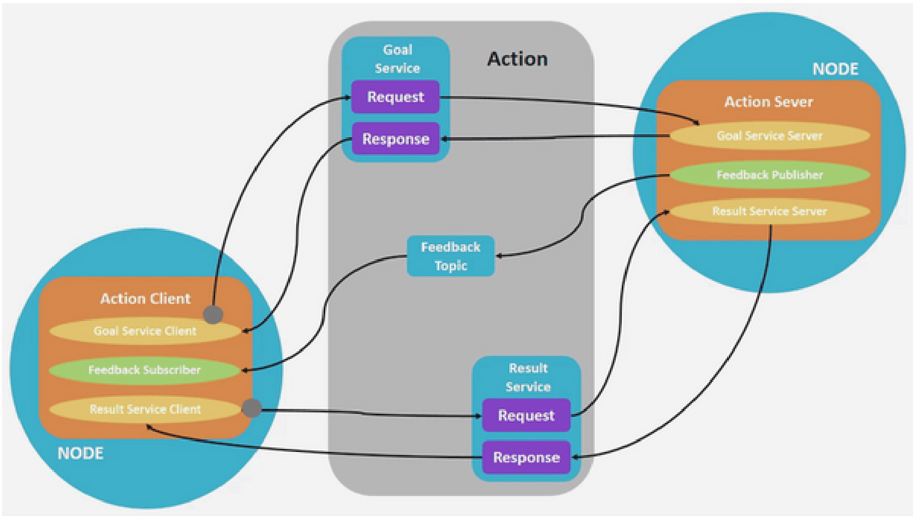
Once those were in place, the robot finally responded with realistic behavior: smooth motion, proper friction, and stable teleop control in Gazebo.

A SMALL LESSON

Reading documentation might feel like a detour when you're eager to get things working — but in reality, you realise it's the shortest path to understanding. Especially with something as layered as ROS 2, the official docs are the map — they help you catch mistakes and really know how you fixed them , instead of sheer hope, and actually understand what your robot is doing instead of just waiting till it behaves.

~~Yes I totally listened to all the advise and did that~~
Obviously learnt this the hard way. If there was one thing, it is this, that could've actually saved more time for me.

Example: One simple flowchart - all basic things!
Recommended to just get to a know a little of everything on it before beginning!



Performance Analysis and Future Plans

Right now, my robot is a humble rolling cuboid — four wheels, a base and a strong will to grow. It spawns cleanly in Gazebo, responds to teleop commands, publishes transforms and odometry, and holds up pretty well under basic simulated physics.
It works. It lives. But it's definitely not done.

What's Working (The Good Stuff)

- The robot is fully integrated into ROS 2 Humble, and works seamlessly with Gazebo and RViz2.
- The differential drive controller is properly loaded and parameterized, with joint interfaces behaving as expected.
- Teleop via teleop_twist_keyboard is responsive — commands are sent, and the robot moves accordingly.
- ros2_control is running through gazebo_ros2_control, no longer relying on mock systems.
- All relevant topics — /cmd_vel, /odom, /tf, and joint states — are alive and behaving.
- Everything's stable!

The Bugs (The Quirks)
Even though the robot is built around a clean 2-wheel differential drive setup, it doesn't always move perfectly straight. Instead, it glides sideways on Gazebo even though the wheels rotate perfectly on RViz2. The wheels rotate forward, the command is straight, and yet... sideways.

Possible causes:

- Improper friction coefficients or Gazebo contact physics.
- Slight misalignment of joint axes.
- Inertial values or mass distribution.
- It's subtle, but it definitely needs fixing, without exploding everything else.

What Could've Been Better

This version of the project hits the basics, but it could've gone further — and it still can.

- STL meshes: The robot is still all boxes and cylinders. Proper 3D models would not only make it look cleaner, but improve collision behavior in simulation.
- Better physics tuning: Right now, joint damping, friction, and dynamics are approximated. A more accurate configuration would result in more realistic movement and fewer surprises.
- URDF to Xacro: Refactoring the description into modular Xacro macros would make adding sensors, variants, and hardware interfaces way easier.
- Cleaner launch architecture: Splitting bringup and visualization into modular launch files would streamline testing and debugging.

Some of the must-haves on the roadmap!!

THE VISION

This little bot might just be a box on wheels today, but it has bigger plans and potential! One day, I'd love to see it out in the wild— navigating actual spaces, building maps, planning paths, reacting to its environment with real sensors and real brains.
Plug in a LIDAR, add an IMU, maybe a depth camera.
Bring in SLAM (Simultaneous Localization and Mapping) to let it build its own map of the world.
Set up autonomous navigation using the full ROS 2 Nav2 stack.
Let it run obstacle avoidance, path planning, recovery behaviors — the full package.
I ~~don't~~ imagine it dodging walls, rolling confidently across unfamiliar rooms, updating maps in real-time. Maybe even working with a swarm one day. You never know.
Sure, for now it slides a little. And it's just a cuboid. But it's got a personality. It tries.
And one day, I want to see it out there — running real code, learning, adapting, mapping its world like it was meant to! It needs to go places! (hopefully in a straight line)

THANK YOU!