ITU ZES Solar Car Team - Autonomous Group

Qualification Problems Report

Prepared by:

Umut Şener

virdult@gmail.com

October 2025

Contents

1. Research Questions on Autonomous Systems

1.1 ROS Basics

ROS → "Robot Operating System", it is not an operating system but a set of libraries and tools to make life easier for robots designing.

//This Why ROS part is %80 rosroboticlearnings.com and %20 me. So don't bother if you already know

Why ROS?

Why ROS? → Well, it is widely used that's why. But why is it widely used?

1. Open source
2. ROS keeps track of the "Transforms" of robot. I guess this is something like, when you don't have this feature, you need to keep track of the robot's starting and ending point of any movement at any part and then calculate stuff, and this does it automatically?
3. ROS have "IK Slovers", IK is Inverse Kinematics, which is something that calculates joints' movements needed to reach a certain place. For example how much to rotate the wheel.
4. It have a function called "MoveIt" and as far as I understood, it helps about movement/pathfinding of the robot.
5. It has a "navigation stack" which helps robot move easier/autonomously with decided environments.
6. Helps robot moves smoother with "ROS control" package.
7. Can use "OpenCV". This helps ROS nodes to process camera images: object detection, lane detection, color tracking etc.

8. It got "Rviz" and "Gazebo". Gazebo is basically a simulation world you can create literally the same as outside and you can test it in there before going to the actual test. Rviz is a debugging thing which shows you the robot's experiences first hand both in simulation and in real tests.

ROS Concepts

Node: Node is basically a specific code that will run. There will be a lot of nodes for the solar car for example one for path finding, one for when car gets stuck, one for car's speed, more than one for car's environment detection etc. Also there is a master node apparently to combine these and run them simultaneously.

Topic: Topic is where node's communicate. There are things called "ROS Messages" which the Nodes will output at certain situations, or even repeatedly at close intervals such as GPS system. These messages are going to be sent to a place called "Topic", and in these topics, the nodes have 2 options, maybe even 3. One is that they can only send messages "Publisher mode". Second one is that they can only read messages that have been sent "subscriber mode". Third one, idk if this is a thing but (it should :D), nodes can be both publisher and the subscriber for that Topic. This message conversation between topics are happening anonymously.

Service: And this is basically a "Message" that doesn't being sent to topic to freely use but a message that you send to a specific node to make that node to the thing you want. Basically a function call across code files. Again for clarity they put names for the service users. Whoever calls this function (sender of the request) is Service Client Node and the node that function is in (request is done and replied from) is Service Server Node.

ROS 2 Humble Installation

I couldn't download Humble so I downloaded Jazzy for now. I'll swap my ubuntu to 22.x and then install Humble later on.

Screenshots

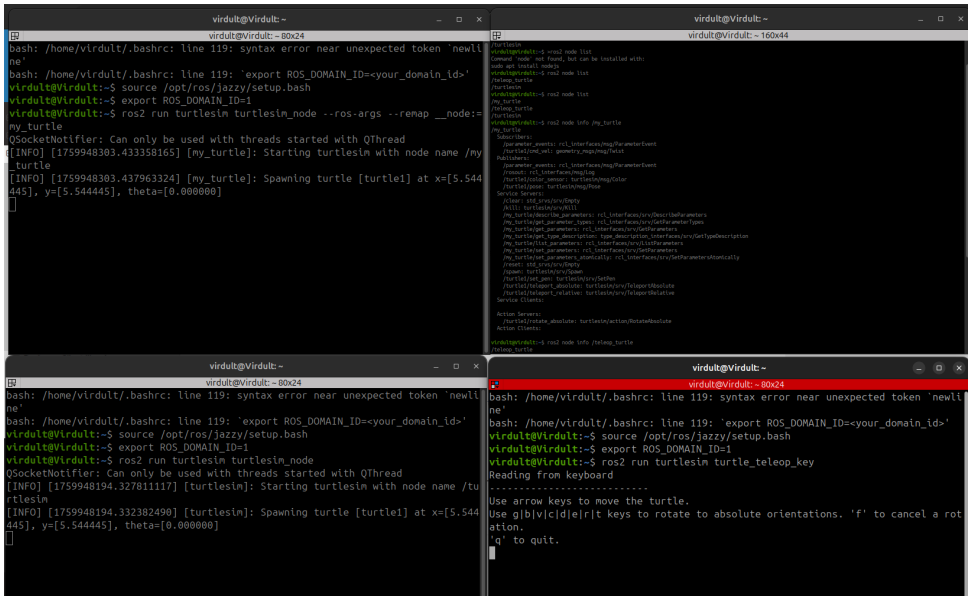

Figure 1.1: ROS2 Humble Tutorial Example (Nodes)

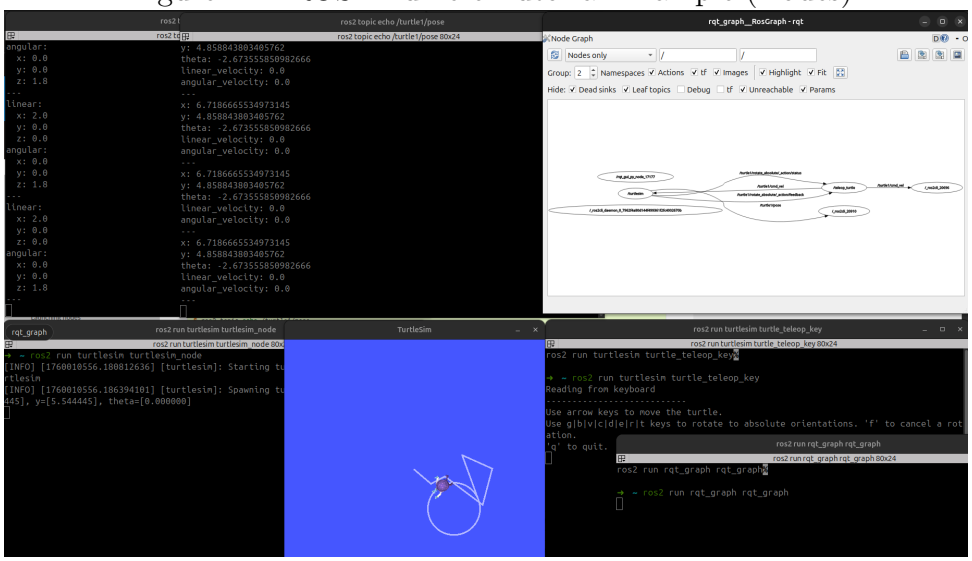

Figure 1.2: ROS2 Humble Tutorial Example (Topics)

1.2 Autonomous Vehicle Sensors

Camera: 2D color images provided as data. The things it can show is the objects around, lane lines, traffic signs/signals. Strengths are cheap, and high resolution, weaknesses are poor in low light or bad weather conditions. Stack modules → Perception and Tracking.

LiDAR → Light Detection and Ranging, it provides ranges to the objects as data, like a bat or a whale. Used for 3D mapping, obstacle detection and localization. Pros, works at day/night and precise shapes. Cons, expensive and struggles in rain/snow/fog. Stack modules → Mapping, Localization, Perception.

Radar: Data provided is again, distance and relative velocity. Used for vehicle tracking and collision avoidance. Pros are it doesn't care about rain/fog/dust and measures velocity. Cons, low resolution and noisy reflections. Stack modules → Perception, tracking, control.

Ultrasonic Sensors: Data provided is short range distance. Used for parking and wall detection. Pros are it is cheap and very reliable in low speeds. Cons are very short range, around 5 meters. Stack modules → perception (only near field).

IMU (Inertial Measurement Unit): Data provided is acceleration and angular rate. Used for estimating motion. Pros are high frequency updates and works by itself without signals from outside (Don't know why is this a pro of this thing only). Cons are drift over time when used by itself. Stack modules → localization, control.

GNSS/GPS: Data provided is Global Position. It is used for global localization and route planning. Pros is provides global reference frame. Cons is low accuracy in canyons/tunnels, signal loss. Stack modules → Localization.

Wheel Encoders: Data provided is wheel rotation so basically distance travelled because we know the wheel's radius. Used for velocity estimation and odometry (estimating change in position over time). Pros are accurate at low speed, also robust. Cons is, slippage causes errors. Stack modules → Localization, control.

Thermal/IR cameras: Data provided is heat signatures. Used for detection at night. Pros are it can detect humans and animals even at night. Cons are it is low resolution and high cost. Stack module → perception.

Microphones (experimental and rare): Data provided is sound directions and sound events. Usages might be siren sounds of ambulances and fire trucks etc. Pros, adds awareness for emergency situations. Cons are noisy environments. Stack module → Perception.

The reason we have a lot of sensors are you can't just plug one of these in and expect a perfect output. Everything have a place that they are good for and a place that they are bad for.

1.3 Mapping and Localization

Localization is estimating the vehicle's pose according to a chosen coordinate frame. It is used continuously by planning, control and tracking.

Mapping is basically mapping the environment that vehicle will be used in. It have different types such as metric maps, HD maps, topological maps etc.

So we need localization and mapping together so that car can stay at the place/ride at the line that we want it to do. Both should be accurate so there is no problem.

If localization fails, we basically don't know how the car is looking. Hence you don't know the car's position/direction accurately, this can make car to crash to somewhere even though it is looking fine on the screen. It depends on the amount of error but it is problematic. If there is a way to track when localization failed, program should directly stop the car if a big thing occurred or if it is getting bigger, program should pull the car to the side of the road and then stop the car I guess.

Well you already gave the answer to the question you asked :D. GPS/GNSS is not reliable alone by themselves because if we go in a tunnel and signal loss, we're just doomed. You don't know where the car is and where it is going from then on.

Let's add IMU + GPS to use the 2 as you want. I choose IMU with the GPS because GPS by itself is already pretty reliable, and the only con IMU had was the consecutive use. So I thought that you can keep using GPS all the time, and use IMU time to time to check if GPS data is correct. On normal roads, you open it and close it repeatedly with intervals. And when going to somewhere that GPS signal might be lost, you can open IMU fully and trust on it until GPS signal comes back.

Wheel encoder makes sense with GPS with this thought as well. If it is not extremely rainy and it is even affecting GPS signals, you're good to go using only wheel encoder + GPS. These are only with 2 of the sensors. A 3rd one would make these much more safer and accurate, and probably a must have as well. Probably even a 4-5th one.

Localization Failure Scenario

Imagine your car have GPS + LiDAR + Wheel encoder + Camera. Weather is extremely foggy and you're going in a tunnel which lights are broken. Since there is fog, LiDAR stopped working. Since you went in the tunnel, GPS signal is lost. Tunnel is dark because of we're in Turkey and the maintenance haven't been made to the lights so camera's are off as well.

We only have one thing to rely on, Wheel encoder. Now, it is definitely preference but I would stop the car after pulling it to the side of the road with the data we had before and our wheel encoders. But if you want to make it keeping going, you'll need more things. Wheel encoder is not enough to pass a place that might have curves because it basically don't have anything to see around and give feedback. It can only show you

where the car is going. So if LiDAR or Camera was not in a mess, you could still make the car going in the tunnel with environment detection + car's movement. You'll have 2 assurances. But with only 1, it is not trustable. Better to pull to the side and stop.

I searched for alternative approaches but, I think better safe than sorry. Though, "preloaded local HD maps" makes sense in our situation since we know the tunnel and we only have the wheel encoder, we can potentially ride through the tunnel. Others were not that reliable.

1.4 Path Planning

Path planning algorithm is literally it's name. Path, Planning. There is a place where your car at and there is a destination. You 'plan' the possible 'path"s that can reach to your destination with this algorithm. Car makes a global planning based on the information given to it, and then car starts to move according to the plan and updates the plan according to what's happening around or what happened to the road.

I've read about the path planning methods a little bit but I'm not very deep in any one them.

There are RRT (Rapidly-exploring Random Tree), Dijkstra etc.

RRT $\rightarrow$ (as far as I remember) $\rightarrow$ It expands towards open areas and quickly finds paths that have no collision. Useful for emergency maneuvering because it is fast.

1.5 Simulation Environments

Starting with the second question: why is simulation helpful for learning and testing?

It is because you don't want to test a random thing I write in code with the real car and risk it crashing in 1 second. Simulations are good for both not risking car and fast for testing since you don't have to move the car to the spot, start it, watch it, be careful with it etc. You just run the test and you get the result in 2 seconds from your computer. And make changes at the code accordingly.

Simulators List

1. CARLA (open-source) → Photorealistic graphics, realistic physics, supports camera/LiDAR/Radar simulation, Python API, ROS integration — great for ML training & research.

2. LGSVL / SVL Simulator → High-fidelity, cloud support, integrates with Autoware & Apollo stacks.

3. NVIDIA Drive Sim / Omniverse → Hardware-accelerated physics & sensor simulation, highly accurate sensor models — industry-grade.

4. AirSim → Good for reinforcement learning, supports drones & cars, Unreal Engine-based.

5. Gazebo / ROS Ignition → Lightweight physics sim, great for robotics & algorithm prototyping (less realistic visuals).

6. Webots → Educational and configurable, supports vehicles but less realistic vehicles.

#Best for photorealism/ML → CARLA
#Best for industry-level physics → NVIDIA Drive Sim
#Best for robotics research → Gazebo/Ignition
#Best for education → Webots
#Best for 'our' solar car team → Gazebo. Though CARLA would be better if everyone could run it.

2. Visualizing Point Clouds

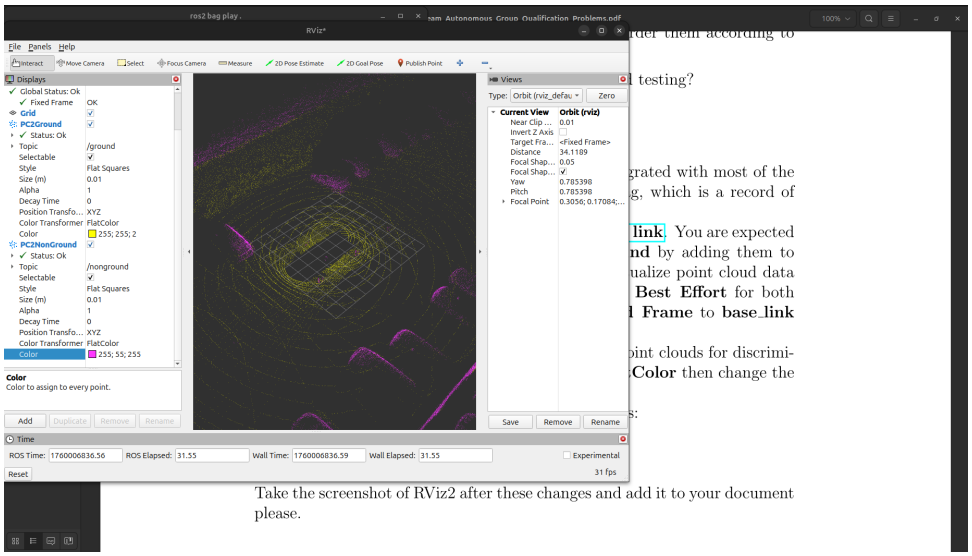

Figure 2.1: RViz2 visualization of /ground and /nonground point clouds