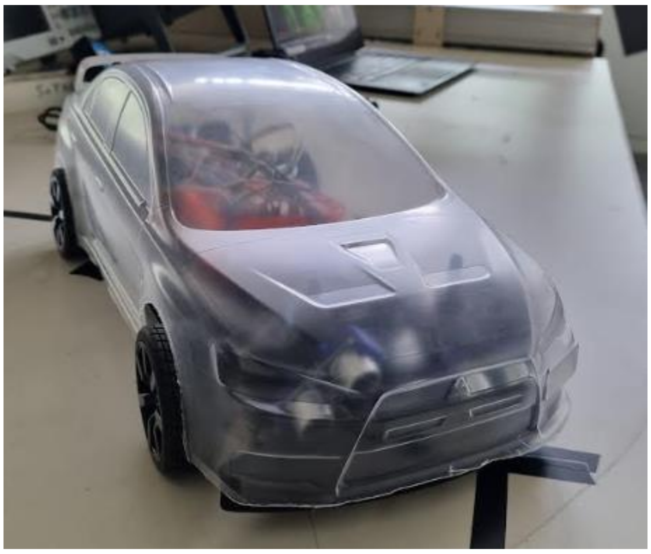
20/01/2025

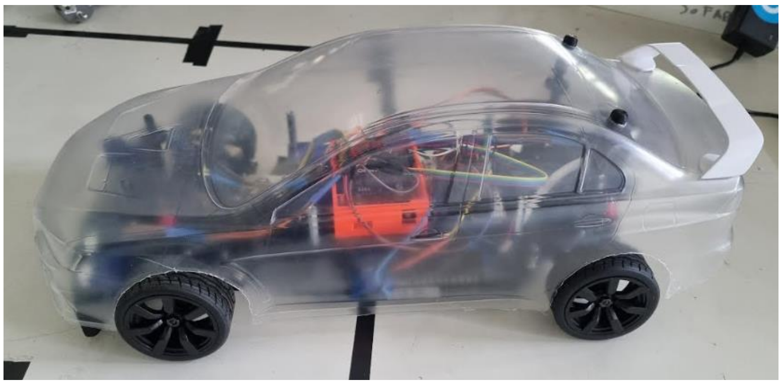
ROBO5 Academic Project:

AutoRCX

Presented by:

Anas Derkaoui





Contents

Introduction:

AutoRCX is an autonomous RC-scaled car capable of traveling long distances and navigating its surroundings while being able not only to detect obstacles, but also to dodge them. By leveraging advanced technologies such as the Jetson Nano card, an Arduino UNO card and a LiDAR, the AutoRCX aims to demonstrate the potential of autonomous driving in a scaled-down remote-controlled car.

The primary objective of this project is to create a fully autonomous RC car, the AutoRCX, that can operate independently, making informed decisions based on sensor data and providing a seamless and immersive experience. By using the ROS (Robot Operating System) middleware and with the integration of powerful computational capabilities offered by the Jetson Nano board, the car can process sensor inputs, analyze the environment, and execute intelligent maneuvers.

AutoRCX incorporates various cutting-edge components to achieve its goals. Some algorithms and packages available within ROS make the car very capable of self-navigating. A high precision LiDAR also enhances the precision of distance measurements and generates a map of the surroundings of the car, contributing to accurate mapping and obstacle avoidance.

Electrical diagram:

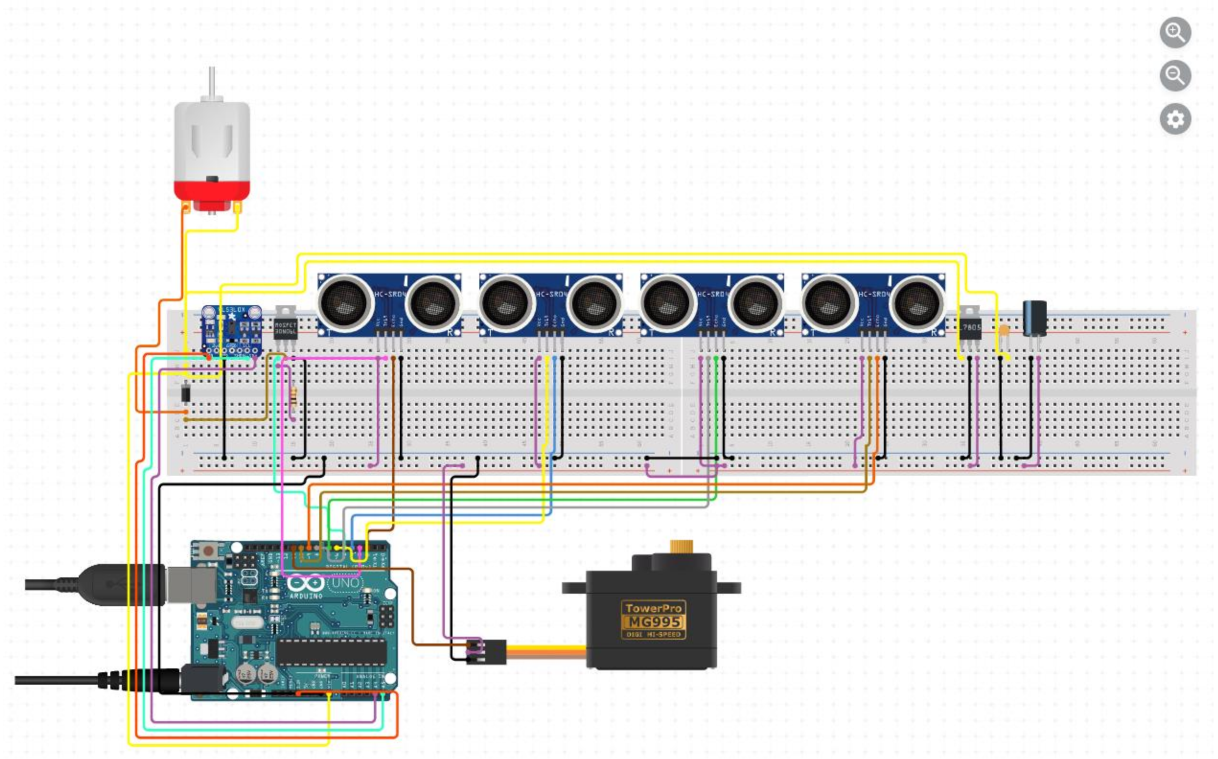


Figure 3, wiring of components

The above setup includes some additional ultrasonic sensors that won't be used during the autonomous navigation. On top of that, the AutoRCX also rocks some high-end hardware that will eventually enable it to navigate autonomously and perform Simultaneous Localization and Mapping. Some of which is the Jetson Nano board:

Figure 4, Jetson Nano board

And an RPLiDAR A1M8:

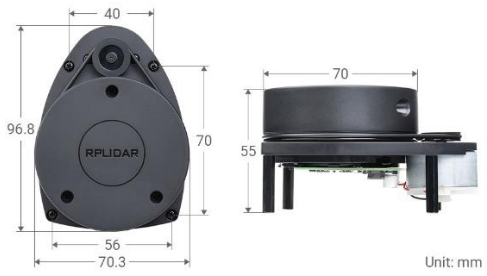


Figure 5, RPLiDAR A1M8

The Jetson board serves as the primary hub for all computational processing, handling intensive tasks such as data analysis and decision-making. In contrast, the Arduino UNO acts as a dedicated controller, connected to the Jetson board to execute commands for operating the servo motor and DC motor. Complementing these components is the LiDAR sensor, a sophisticated device that emits laser light to detect obstacles and map the environment. Capable of providing 360-degree coverage, the LiDAR will be directly connected to the Jetson board. Its primary role is to scan the car's surroundings and generate a detailed environmental map, crucial for navigation and obstacle avoidance.

Scope statement:

Objectives :

- Develop an autonomous car capable of operating without direct human intervention.
- Enable the car to navigate towards a specified destination by processing sensor data and making informed decisions.
- Implement obstacle detection and collision avoidance mechanisms using the LiDAR.
- Integrate the Jetson Nano board for powerful computing.
- Have a feedback mechanism to provide real-time updates on the car's location, progress, and actions.

- Document the entire development process, including hardware setup, software implementation, and testing procedures in my GitHub.

Estimated time spent on the project:

- Time on-campus: 78h
- Time off-campus: 24h
- Total estimated time: 126h
- Cost as an engineer (starting with a gross annual salary of 38 000€ for 1600 hours of work): 2992.5 €

Budget:

- Jetson Nano board: 227€
- Jetson Nano fan : 10€
- DC motor controller : 12,80€
- Battery : 40€
- Arduino UNO: 29,40€
- Power converter : 17€
- RPLIDAR A1M8 : 113€
- Car frame : 120€
- Car body shell : 57,46€
- DC motor : 19€
- Wheels and tires : 18€
- IMU : 42€

⇨ Total : 705.66 €

Risk and Challenges :

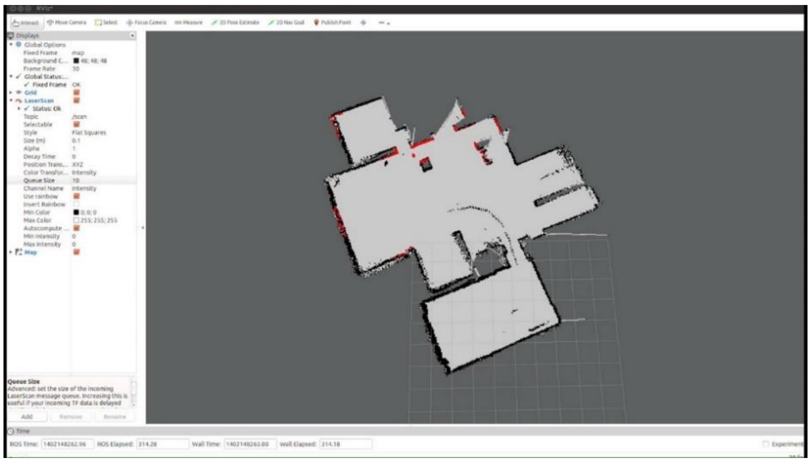
- Integration and compatibility challenges between hardware components and software libraries, such as ROS (Robotics Operating System), can arise during development: Certain sensors, like the MPU-9250 IMU, lack a compatible ROS 1 driver, significantly narrowing the options for 9 Degrees of Freedom (DoF) IMUs within this framework. These limitations necessitate careful selection of components to ensure seamless operation and compatibility with ROS.
- Ensuring reliable and accurate sensor data processing for precise decision-making: Some sensors should be calibrated before use. If not calibrated, they would give faulty information to the algorithms depending on them. Thus, the navigation will not work properly.
- Adding simulated odometry to the car: Since the car is not equipped with a position sensor that would output real-time odometry data, I resorted to a simulation of odometry.
- Fine tuning the parameters given to the algorithm in order to have high precision feedback: Since the robot type that I am using is an Ackermann robot, it has very different aspects than most of the other common robots, so this should be considered while developing the algorithms.

- Passing information from Jetson Nano card to the Arduino UNO card and eventually to the car components: This is a crucial role and it has to be robust enough to let the car navigate to the precise location defined by the algorithms.
- Balancing power consumption and battery life for extended operation: The power that the car components consume should be accounted for because it will define how long the car can operate. It is also important to know if there are any marginal trade-offs to take, in order to expand battery life.
- Dealing with unpredictable environmental factors that may affect sensor accuracy: The LiDAR sensor can detect obstacles around it, but there are some challenging environment parts, like black-color materials, that can give the LiDAR a hard time detecting obstacles.
- Handling random and unexpected behavior from hardware and software: The hardware is not always reliable, especially when the project is still under development. There could be some random behavior that should be assessed as soon as possible in order to ensure a good working implementation.

Algorithmic functioning:

The algorithmic functioning of the AutoRCX involves a series of steps and decision-making processes to enable the car to navigate towards a destination while avoiding obstacles. As a general view, when the autonomous car is supposed to go to a chosen destination, a bunch of algorithms like AMCL (Adaptive Monte-Carlo Localization), "base-local-planner" and "move-base" will try to calculate the pose, heading and the path in order for the car to reach its final position. All of these packages are accessible within the ROS middleware that I have implemented in the AutoRCX.

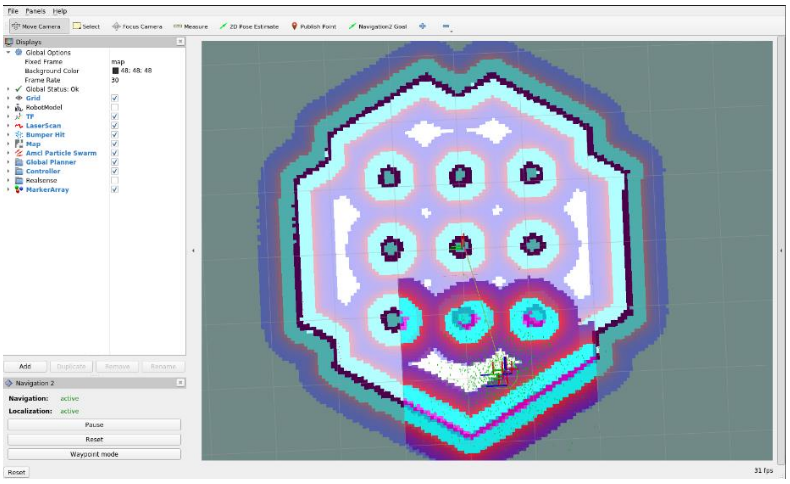
Firstly, a map of the environment is already available and can be visualized by RVIZ (Robot Visualization), which is the tool that will use all of the algorithms mentioned above. The RVIZ application enables robotics engineers to visualize a lot of aspects of the robot's surroundings and the robot itself. The map that I had saved of the robot's environment, was previously generated using the "hector mapping" SLAM (Simultaneous Localization and Mapping) algorithm. SLAM consists of generating a map of the environment surrounding the AutoRCX by receiving the data coming from the LiDAR and converting it to either a white pixel, meaning a place with no obstacles, or a black pixel, which means that an obstacle is present in that area. The dark grey area will remain unknown as long as the LiDAR has not scanned it yet or if it is out of reach of the lidar's scans.



```
Global Options
  Fixed Frame          map
  Background C...      48; 48; 48
  Frame Rate           30
Global Status:...
  ✓ Fixed Frame        OK
Grid                   ✓
LaserScan              ✓
  ✓ Status: Ok
  Topic                /scan
  Selectable           ✓
  Style                Flat Squares
  Size (m)             0.1
  Alpha                1
  Decay Time           0
  Position Trans...    XYZ
  Color Transfor...    Intensity
  Queue Size           10
  Channel Name         intensity
  Use rainbow          ✓
  Invert Rainbow
  Min Color            0; 0; 0
  Max Color            255; 255; 255
  Autocompute ...      ✓
  Min Intensity        0
  Max Intensity        0
Map                    ✓
```

Figure 6, RVIZ showing an example of a map generated by a LiDAR.

Secondly, some very important files should be created in order to do the navigation part. First and foremost, the “global_costmap”. This represents the actual map of the environment of the car with its static obstacles. It is used in order to define the obstacles and how much it is of a risk to be close to one of them. Then there is the “local_costmap”, which will serve as a smaller version of the previous map but will be needed in order to detect dynamic obstacles that are moving in real-time and display the trajectory of the car. Finally, a “common_costmap” which will contain data about places with high costs and some others with low costs. If the cost of a pixel is high, that means that the pixel represents an obstacle, if it is the opposite, then there is no object.

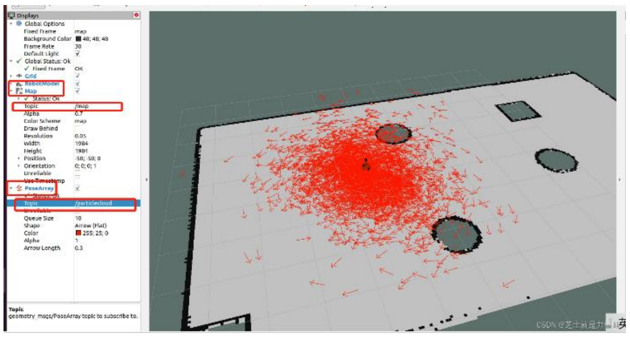


```
Global Options
  Fixed Frame          map
  Background Color     48; 48; 48
  Frame Rate           30
✓ Global Status: Ok
Grid                   ✓
RobotModel
TF                     ✓
LaserScan              ✓
Bumper Hit             ✓
Map                    ✓
Amcl Particle Swarm    ✓
Global Planner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

Navigation:    active
Localization:  active
```

Figure 7, RVIZ tool showing the “global_costmap” and the “local_costmap” which is smaller.

Another important algorithm is AMCL (Adaptive Monte Carlo Localization), which is widely recognized as a powerful tool for robot localization. AMCL is a probabilistic localization method based on the Monte Carlo approach. Its primary function is to help the robot estimate its pose, which includes its position and orientation within the saved map. The algorithm achieves this by maintaining a set of particles, each representing a possible pose of the robot. As the robot moves and collects odometry and IMU (Inertial Measurement Unit) data, the particles are updated iteratively to reflect these changes. During the motion update phase, the particles are adjusted based on the robot's motion model, incorporating randomness to account for movement uncertainty. The sensor’s update phase follows, where each particle is assigned a weight based on how well the robot’s LiDAR readings align with the expected readings for that particle's pose. Resampling is then performed to focus on particles with higher weights, discarding those with lower probabilities. Over time, this process causes the particle distribution to converge on the robot's true pose, even if the initial estimate was imprecise. AMCL is widely used in robotics due to its effectiveness in static environments, making it an essential component for this project.



```
Displays
▾ Global Options
    Fixed Frame          map
    Background Color     ■ 48; 48; 48
    Frame Rate           30
    Default Light        ✓
▾ ✓ Global Status: Ok
    ✓ Fixed Frame        OK
▸ Grid                   ✓
  RobotModel             ✓
▾ Map                    ✓
  ▸ ✓ Status: Ok
    Topic                /map
    Alpha                0.7
    Color Scheme         map
    Draw Behind
    Resolution           0.05
    Width                1984
    Height               1984
  ▸ Position             -50; -50; 0
  ▸ Orientation          0; 0; 0; 1
    Unreliable
    Use Timestamp
▾ PoseArray              ✓
    Topic                /particlecloud
    Queue Size           10
    Shape                Arrow (Flat)
    Color                ■ 255; 25; 0
    Alpha                1
    Arrow Length         0.3

Topic
geometry_msgs/PoseArray topic to subscribe to.
```

Figure 8, RVIZ tool showing the red arrows that represent a possible pose of the robot using AMCL.

Furthermore, a critical aspect of this project is path planning, which determines the car's trajectory to navigate its environment effectively. For this purpose, I utilized a readily available algorithm called “base_local_planner”, a widely used method in robotics. This algorithm calculates the optimal direction for the car to move and determines the corresponding linear and angular velocities required to follow that path. While numerous algorithms are available for path planning, “base_local_planner” stands out for its reliability and integration within robotic frameworks, making it a popular choice for such applications.

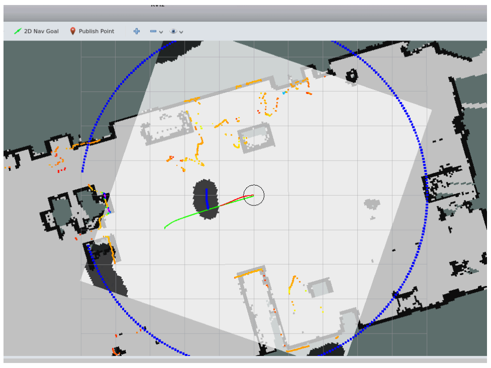


Figure 9, “base_local_planner” showing the trajectory in red and green that the robot will take.

Lastly, for the autonomous RC car to reach its defined destination on the map, it must receive appropriate commands to control its wheels and steering. This task is managed by the “move_base” package, a core component of the ROS navigation stack. The “move_base” package acts as the central node that integrates global and local path planning with robot motion control. Its primary role is to take the desired goal pose as input and orchestrate the process of reaching that goal safely and efficiently. “move_base” works by combining data from odometry, IMU, and the robot's environment map, to execute path planning and obstacle avoidance. It uses a global planner to generate an optimal path from the current position to the goal based on the map, while the “base_local_planner” handles real-time adjustments to avoid dynamic obstacles and ensure smooth motion. This enables the car to navigate autonomously to its target pose while dynamically responding to changes in the environment.

Now, here's a high-level overview of the algorithm described above:

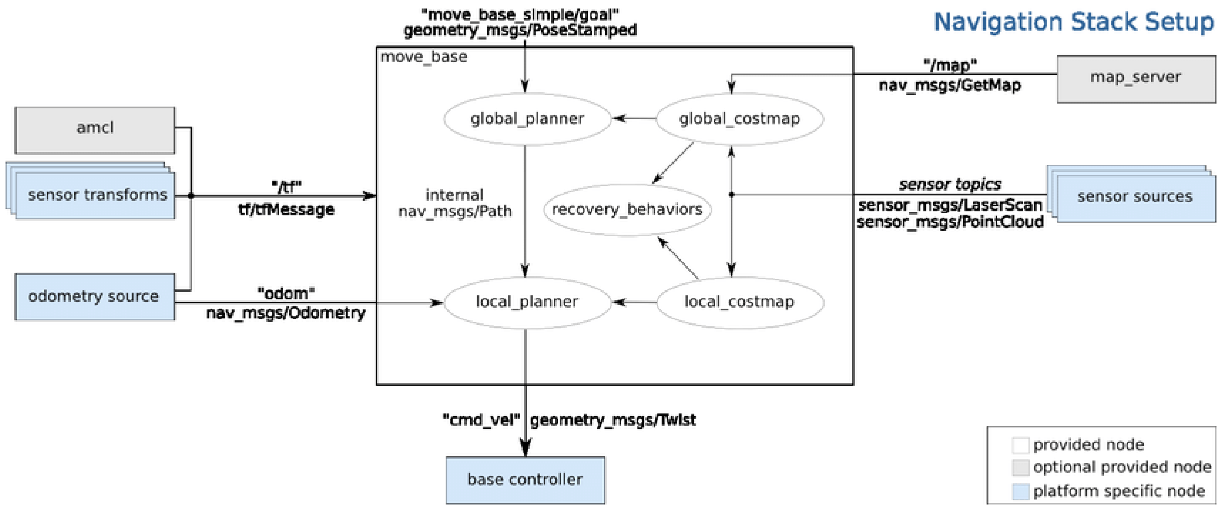


Figure 10, Setup of the navigation stack described previously.

Now let’s see the behavior of the AutoRCX at each stage:

1. Initialization:

- I plug the battery in the car, the servo motor and the DC motor receive power and the Jetson Nano board turns on and then delivers power to the Arduino board along with the LiDAR.
- Then I log into the Jetson Nano wirelessly and I launch the ROS navigation package.

2. Starting up the algorithms and Reading Sensor Data:

- I launch some nodes on ROS to control the car remotely. Then, I use a remote controller in order to move it around the map that I want to use for autonomous navigation.
- After finishing the mapping, the RVIZ tool opens up and shows the generated map calculated using the “hector_mapping” SLAM algorithm and the LiDAR.
- Then I save that map in order to use it for autonomous navigation.

3. Initialization of the navigation stack:

- After launching the autonomous navigation stack, using the AMCl package, an initial position of the robot on the map can be chosen using the RVIZ “2D Pose Estimate” tool.
- After displaying the “global_costmap” and the “local_costmap”, the obstacles are highlighted and the robot is surrounded by a perimeter that shows nearby static and dynamic obstacles. Just like in “Figure 7”.

4. Path Planning :

- Now the robot is placed at an estimated position, the “2D Nav Goal” tool available in RVIZ is used to define a target destination for the robot.
- The “move_base” will determine the optimal path to the destination based on the current location, detected objects, and available information on both the “local_costmap” and “global_costmap”.

5. Motor Control :

- The “base_local_planner” generates two commands of type linear and angular velocity in order to control the speed and steering of the AutoRCX.

6. Feedback and Monitoring :

- In the RVIZ tool we can see the robot moving to the target position as shown in “Figure 9”. And in the terminal of the navigation package, we can see the feedback of the algorithms running in the background.

Encountered problems and proposed solutions:

- The shell of the AutoRCX did not fit the frame that I had previously, so I had to cut some pieces and poke some holes. Now the shell can be held in place firmly.
- In order to hold the LiDAR in place, I had to cut 4 holes at the top of the car shell, and now it sits right above the Jetson Nano card. It is now stable and is not causing any interference problem with other hardware.
- The Jetson card is enclosed within the car shell. So, in order to connect to it and do the autonomous navigation, I use a Wireless HDMI cable that has a transmitter and a receiver enabling me to use the graphical interface of the Jetson card wirelessly.
- I used a piece of wood and some screws to hold the Jetson card in place. I cut the wooden piece so as to fit into the car frame both vertically and horizontally. The height can be easily adjusted with some screws and nuts.
- The navigation was throwing a lot of errors and warnings when I first tried it. Now, after some deep digging into the parameters, I am able to start the navigation stack and recognize the majority of the involved parameters.
- The Jetson Nano card throttles occasionally. And one way of dodging this problem is to connect it to a battery that is not fully charged to its maximum capacity, which is 12,2V.
- From time to time the card refuses to boot when the LiDAR is plugged in. And this causes some sort of power shortage. This is one of the persistent problems and I believe it can be sorted out by regulating the amperage output for each of the components of the car. Another solution involves separating the power source of each one of the car components so that the Jetson Nano is not affected by the other car parts.

Conclusion:

The AutoRCX project has successfully culminated in the development of an autonomous car powered by the computational capabilities of the Jetson Nano board. This project marks a significant step forward, achieving its primary objective of enabling the car to navigate a known environment independently. While there is still room for improvement, the progress made thus far provides a strong foundation for future advancements.

Looking ahead, integrating a depth camera will unlock exciting possibilities for the AutoRCX project. Depth perception will enhance sensor fusion capabilities, facilitate visual SLAM and odometry, enable advanced object detection, and support the implementation of more sophisticated decision-making algorithms. These improvements will elevate the car’s autonomous functionality to new levels of precision and adaptability.

Ultimately, incorporating modules such as GPS will further expand the car’s capabilities, allowing for enhanced connectivity, localization, and navigation performance. With continued innovation and exploration, AutoRCX has the potential to make significant contributions to the field of autonomous systems and robotics, paving the way for more advanced and versatile autonomous vehicles.

Bibliography:

https://www.theconstruct.ai/ros-qa-119-ros-mapping-tutorial-provide-map/

https://robotics.stackexchange.com/questions/24180/multiple-ros-installation-on-single-machine

https://answers.ros.org/question/395222/move_base-problem-unable-to-get-starting-pose-of-robot-unable-to-create-global-plan/

https://stackoverflow.com/questions/53545159/save-a-map-in-ros

https://robotics.stackexchange.com/questions/99344/move-base-wont-subscribe-to-scan-topic

http://wiki.ros.org/base_local_planner

http://wiki.ros.org/melodic/Installation/Ubuntu

http://wiki.ros.org/fr/ROS/Tutorials/InstallingandConfiguringROSEnvironment

http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes

https://github.com/bandasaikrishna/Autonomous_Mobile_Robot/tree/main/mobile_robot_autonomous_navigation

https://dev.to/ivanorsolic/a-self-driving-rc-car-and-a-complete-guide-to-build-your-own-23el

https://docs.ros.org/en/melodic/api/robot_localization/html/configuring_robot_localization.html