Data Science tools & techniques

Student name: Ihor Tizengauzen

Student number: 2101247

Date: 07.12.2024

Table of Contents

[Introduction 2](#_Toc184485923)

[Project creation process 3](#_Toc184485924)

[COllaboration process 5](#_Toc184485925)

[Code Structure and Design 7](#_Toc184485926)

[Repository Overview 7](#_Toc184485927)

[src Directory Subfolders 7](#_Toc184485928)

[Key Files and Functions 8](#_Toc184485929)

[Program Demonstration 9](#_Toc184485930)

[CHALLENGES AND SOLUTIONS 10](#_Toc184485931)

[Making the Robot Model Appear in Gazebo and RViz 10](#_Toc184485932)

[Path Configuration Issues 10](#_Toc184485933)

[Integrating ROS2 Packages 10](#_Toc184485934)

[Tuning Navigation Stack Parameters 10](#_Toc184485935)

[Future Work and Improvements 11](#_Toc184485936)

[Improved Computational Efficiency 11](#_Toc184485937)

[Enhanced Multi-Map Management 11](#_Toc184485938)

[Conclusion 11](#_Toc184485939)

[References 12](#_Toc184485940)

# Introduction

The goal of this project is to develop a fully autonomous, navigation-enabled robotic vehicle, "SmartCAR," capable of operating both in a simulated environment and real-world scenarios. The project is built on the ROS2 framework, utilizing simulation tools such as Gazebo for environmental interaction testing and RViz2 for real-time visualization and validation of the robot's model and sensor placements.

This project addresses the challenge of creating a robotic system that can navigate autonomously, leveraging precise localization and path-planning functionalities. The solution involves designing a 3D model of the SmartCAR, configuring sensor inputs (LiDAR and IMU) for localization and mapping, and implementing control systems that allow autonomous and user-directed navigation. Through a structured workflow, the assignment guided the development from model creation and sensor integration to simulated testing, ultimately achieving a system capable of real-time autonomous navigation.

This report outlines the development process, focusing on each stage of the assignment, from the initial 3D model setup and simulation in Gazebo to implementing key functionalities like odometry and keyboard-based control. The outcome is a robust navigation system for SmartCAR, ready for deployment in both virtual and real environments.

# Project creation process

The development of the SmartCAR project followed a structured workflow designed to ensure a clear progression from concept to implementation. The process began with defining the project scope and functional requirements, focusing on autonomous navigation capabilities and seamless integration between simulation and real-world hardware. A 3D model of the robot was designed using URDF and XACRO files, which were subsequently tested in Gazebo and RViz for visualization and accuracy. Simultaneously, critical ROS2 packages were configured to handle sensor input, localization, and navigation. Regular iterations, involving testing and debugging of individual components, ensured compatibility and reliability. The final stages included integration of the navigation stack and odometry systems, with extensive simulation trials to validate performance. This meticulous approach enabled the creation of a robust, adaptable system ready for both virtual and physical deployment.

# COllaboration process

The successful completion of the SmartCAR project was achieved through close collaboration with mechanical engineers, who were solely responsible for designing and creating the robot's physical model. Their expertise ensured the development of a robust and functional 3D model that accurately reflected the project requirements. The mechanical engineers handled sensor placements, joint configurations, and structural integrity, which were seamlessly integrated into the project’s URDF and XACRO files. This interdisciplinary collaboration ensured the alignment of hardware and software components, enabling the creation of a cohesive and efficient system.Functional Requirements

|  |  |
| --- | --- |
| **General Requirements** | **Robot Definition** |
| Each ROS2 package must include a README file with comprehensive documentation.  Packages should be compiled successfully without errors.  Launch files and nodes need to be well-commented to facilitate understanding and maintenance, ensuring they function correctly across all required systems. | Develop a Unified Robot Description Format (URDF) file that accurately represents SmartCAR’s physical structure, sensor placements, and interface configuration.  Ensure the robot model can be visualized in RViz2, allowing for inspection and validation of the design. |
| **Simulation Requirements** |
| Integrate essential sensors, including IMU (Inertial Measurement Unit) and LiDAR, within the Gazebo environment.  Implement odometry and command response functionalities for movement tracking and control.  Configure localization and navigation features, enabling SmartCAR to autonomously navigate through simulated environments with accurate spatial awareness and obstacle avoidance. |

# Code Structure and Design

## Repository Overview

The ros2\_ws workspace is structured to support the development of the SmartCAR robot simulation and real-world implementation using ROS2. It contains the following primary directories:

|  |  |
| --- | --- |
| Directory | Description |
| install | |  | | --- | | Stores installed packages after the workspace is built, allowing ROS2 to recognize and utilize them effectively. |  |  | | --- | |  | |
| build | |  | | --- | | Contains compiled files generated during the build process. Each package has its own subdirectory here with generated object files and build configurations. |  |  | | --- | |  | |
| Log | |  | | --- | | Stores logs generated during build and runtime, essential for debugging and issue tracking. |  |  | | --- | |  | |
| src | Source directory containing the main packages for the project, such as smart\_car and smartcar\_msgs. |

## src Directory Subfolders

|  |  |  |
| --- | --- | --- |
| Package | Subdirectory | Description |
| smart\_car | .vscode | |  | | --- | | Contains Visual Studio Code configuration files (c\_cpp\_properties.json, settings.json) for managing environment paths. |  |  | | --- | |  | |
|  | config | |  | | --- | | Includes configuration files for simulation and navigation: ekf.yaml (EKF localization), nav2\_params.yaml (navigation parameters), and maps for simulation like smalltown\_world. |  |  | | --- | |  | |
|  | launch | |  | | --- | | Contains launch scripts (gazebo.launch, localization.launch, nav2.launch and etc.) to initiate different components in simulation. |  |  | | --- | |  | |
|  | rviz | |  | | --- | | Stores RViz configuration files (e.g., smart\_car.rviz, nav2\_default\_view.rviz) for visualizing the robot and its environment. |  |  | | --- | |  | |
|  | script | |  | | --- | | Houses executable scripts for automating tasks or specific runtime functions. |  |  | | --- | |  | |
|  | smart\_car | |  | | --- | | Includes Python initialization scripts that set up or initialize SmartCAR’s Python modules. |  |  | | --- | |  | |
|  | urdf | |  | | --- | | Contains URDF and XACRO files (smartcar.urdf, smartcar.urdf.xacro) that define the robot’s physical structure and setup for both simulation and visualization. |  |  | | --- | |  | |
|  | world | |  | | --- | | Stores Gazebo world files that specify the simulation environments. |  |  | | --- | |  | |
| |  | | --- | | smartcar**\_**msgs |  |  | | --- | |  | | |  | | --- | | **.**vscode |  |  | | --- | |  | | |  | | --- | | Contains VSCode settings specific to this package, similar to those in the smart\_car package. |  |  | | --- | |  | |
|  | msg | |  | | --- | | Contains custom message definitions used within the SmartCAR system. |  |  | | --- | |  | |
|  | include | |  | | --- | |  |  |  | | --- | |  | |
|  | src |  |

## Key Files and Functions

|  |  |  |
| --- | --- | --- |
| File/Function | Description | Parameters/Dependencies |
| |  | | --- | | URDF and XACRO Files |  |  | | --- | |  | | |  | | --- | | Define the robot’s physical structure, including sensors, joints, and links for simulation and visualization (smartcar.urdf, smartcar.urdf.xacro). |  |  | | --- | |  | | |  | | --- | | Dependencies: urdf and xacro ROS packages. |  |  | | --- | |  | |
| |  | | --- | | gazebo.launch |  |  | | --- | |  | | |  | | --- | | Launches the Gazebo simulation with the specified world files, setting up a virtual environment for SmartCAR. |  |  | | --- | |  | | |  | | --- | | Parameters: world file path. |  |  | | --- | |  | |
| localization.launch | |  | | --- | | Initializes localization nodes, using an Extended Kalman Filter (EKF) for sensor fusion and localization. |  |  | | --- | |  | | |  | | --- | | Parameters: EKF configuration (ekf.yaml). |  |  | | --- | |  | |
| |  | | --- | | ekf.yaml |  |  | | --- | |  | | |  | | --- | | Configures the EKF node for localization and sensor fusion. |  |  | | --- | |  | | |  | | --- | | Dependencies: robot\_localization package. |  |  | | --- | |  | |
| |  | | --- | | nav2\_params.yaml |  |  | | --- | |  | | |  | | --- | | Sets parameters for the navigation stack, including obstacle avoidance and path planning for autonomous navigation. |  |  | | --- | |  | | |  | | --- | | Dependencies: nav2 package. |  |  | | --- | |  | |
| RViz Configurations (smart\_car.rviz) | |  | | --- | | Sets up RViz for visualizing SmartCAR’s state, position, and sensor data, providing a real-time view of the robot’s environment and interactions. | |  |
| README.txt | Provides information about gazebo.launch.py |  |

# Program Demonstration

The demonstration video provides a comprehensive overview of the SmartCAR project, showcasing how the program fulfills its functional requirements. At the beginning of the video, the robot is tested on a pre-configured map with built-in objects, allowing for an initial assessment of its navigation and obstacle-detection capabilities. The video then proceeds to launch gazebo.launch.py, initializing the simulation environment in Gazebo and loading the SmartCAR model. To further test the robot’s obstacle-avoidance capabilities, an additional block is manually placed near the robot to simulate a new obstacle. In RViz, after visualizing the robot and setting a target destination, the SmartCAR begins navigating towards the goal. The video highlights how the robot detects both the pre-configured objects and the newly added obstacle, autonomously adjusting its path to go around them. This real-time demonstration emphasizes the robot’s effective obstacle avoidance and path-planning functionality.

CHALLENGES AND SOLUTIONS  
Throughout the development of the SmartCAR project, I encountered several challenges that required in-depth problem-solving and debugging. These challenges were often related to ROS2 functionality, sensor integration, and simulation configuration, as well as understanding engineering and modeling processes as an International Business student. Collaborating with mechanical engineers to grasp the technical aspects of model creation, such as URDF and XACRO file structures, sensor placements, and physical configurations, was particularly demanding. The steep learning curve in understanding the engineering terminology and requirements added complexity to the project but provided valuable interdisciplinary learning.

## Making the Robot Model Appear in Gazebo and RViz

One of the problems was getting the SmartCAR model to appear correctly in both Gazebo and RViz. There were times when the model simply wouldn't load, or it appeared distorted or incomplete, which hindered the ability to test and visualize the robot’s capabilities.

After troubleshooting, I identified that issues were in incorrect URDF configurations and sometimes missing dependencies in the launch files. I refined the URDF and XACRO files in the urdf folder and ensured all necessary parameters were set in the launch files. Additionally, I made adjustments in gazebo.launch and rviz. This solved the rendering issues and enabled consistent model visualization in both Gazebo and RViz.

## Path Configuration Issues

Another problem was incorrect or inconsistent paths to files and packages. Many times, scripts or launch files would fail to execute because they couldn’t locate a required file, especially in nested directories.

To solve this, I reviewed all file paths across the project to ensure they were relative and correct. For example, I adjusted paths in configuration files and ensured that the CMakeLists.txt and package.xml files were accurately pointing to dependencies. This not only reduced errors during runtime but also helped in maintaining organized file structure.

## Integrating ROS2 Packages

Integrating different ROS2 packages like nav2 (for navigation) and robot\_localization (for sensor) was complex, with compatibility issues arising when syncing data from multiple sensors.

By adjusting the configuration parameters in ekf.yaml and nav2\_params.yaml, I was able to get the navigation and localization systems to work together. This tuning improved the overall stability of the system.

## Tuning Navigation Stack Parameters

The nav2 stack's path planning and obstacle avoidance initially performed poorly, causing the robot to struggle around obstacles.

Through adjustments to nav2\_params.yaml, I was able to improve SmartCAR's obstacle avoidance and path planning.

# Future Work and Improvements

There are several things where the project could be further enhanced, both in terms of functionality and performance.

## Improved Computational Efficiency

* Optimize launch files to start only essential nodes and implement more efficient algorithms for localization and mapping. In a virtual machine (VM) environment, selectively disabling resource-heavy nodes, like high-resolution RViz or intensive Gazebo physics, can reduce CPU and memory load.
* These optimizations allow the system to run efficiently on lower-spec hardware, which is particularly helpful for mobile applications and VM-based testing with limited resources.

## Enhanced Multi-Map Management

* Implement a multi-map management system to allow the SmartCAR to store and switch between different pre-mapped environments. This enables the robot to operate more efficiently by loading specific maps environments, such as office layouts, warehouses, or outdoor paths, without needing to remap each time.
* Multi-map management enhances navigation accuracy across various environments, reduces re-mapping requirements, and enables more effective performance testing across different scenarios.

# Conclusion

The development of the SmartCAR project demonstrated success in building a functional autonomous navigation system capable of operating in a simulated environment. The project met the functional requirements, including robust simulation capabilities in Gazebo, visualization in RViz2, and sensor integration using ROS2 frameworks. The structured approach, from model creation to sensor configuration and testing, provided valuable insights into the complexities of robotic navigation and localization.

Key takeaways from the project include the importance of detailed configuration for URDF files, the role of precise path management, and the necessity of testing each component in isolation before integration. Throughout the process, several challenges were encountered, particularly with model visualization, file path configuration, and compatibility between ROS2 packages. These obstacles highlighted the need for careful debugging, systematic troubleshooting, and thorough documentation.

Working on this project was very challenging, and the results are not ideal. However, it provided valuable insights into the processes of coding and data gathering within the specific context of engineering.

# References

* 1. [Simulating Robots with Gazebo and ROS | Getting Ready to Build Robots with ROS #8 - YouTube](https://www.youtube.com/watch?v=laWn7_cj434)
  2. [(1) ROS2 robot URDF + Xacro creation, visualize in RVIZ2 and Gazebo + Teleoperation | Beginner Tutorial - YouTube](https://www.youtube.com/watch?v=Vbh3--etiwg)
  3. [(1) Intro: Install and Setup ROS2 Humble - ROS2 Tutorial 1 - YouTube](https://www.youtube.com/watch?v=0aPbWsyENA8&list=PLLSegLrePWgJudpPUof4-nVFHGkB62Izy)
  4. [(1) Start Your First ROS2 Node - ROS2 Tutorial 2 - YouTube](https://www.youtube.com/watch?v=c5DRTN2b2kY&list=PLLSegLrePWgJudpPUof4-nVFHGkB62Izy&index=2)
  5. [Create and Set Up a ROS2 Workspace - ROS2 Tutorial 3](https://www.youtube.com/watch?v=3GbrKQ7G2P0&list=PLLSegLrePWgJudpPUof4-nVFHGkB62Izy&index=3)
  6. [Create a ROS2 Python Package - ROS2 Tutorial 4](https://www.youtube.com/watch?v=iBGZ8LEvkCY&list=PLLSegLrePWgJudpPUof4-nVFHGkB62Izy&index=4)
  7. [Create a ROS2 Node with Python and OOP - ROS2 Tutorial 5](https://www.youtube.com/watch?v=wfCuPQ_6VbI&list=PLLSegLrePWgJudpPUof4-nVFHGkB62Izy&index=5)
  8. [What is a ROS2 Topic? - ROS2 Tutorial 6](https://www.youtube.com/watch?v=MwEXX6a-TWw&list=PLLSegLrePWgJudpPUof4-nVFHGkB62Izy&index=6)
  9. [Write a ROS2 Publisher with Python - ROS2 Tutorial 7](https://www.youtube.com/watch?v=Yy4OgGwEAj8&list=PLLSegLrePWgJudpPUof4-nVFHGkB62Izy&index=7)
  10. [Write a ROS2 Subscriber with Python - ROS2 Tutorial 8](https://www.youtube.com/watch?v=od3JwOeyEXc&list=PLLSegLrePWgJudpPUof4-nVFHGkB62Izy&index=8)
  11. [Create a Closed Loop System with a Publisher and a Subscriber - ROS2 Tutorial 9](https://www.youtube.com/watch?v=DXEnVEQjImo&list=PLLSegLrePWgJudpPUof4-nVFHGkB62Izy&index=9)
  12. [What is a ROS2 Service? - ROS2 Tutorial 10](https://www.youtube.com/watch?v=FSqm0fDfxrk&list=PLLSegLrePWgJudpPUof4-nVFHGkB62Izy&index=10)
  13. [Write a ROS2 Service Client with Python - ROS2 Tutorial 11](https://www.youtube.com/watch?v=vCTbUgw6k8U&list=PLLSegLrePWgJudpPUof4-nVFHGkB62Izy&index=11)
  14. [(1) 8 Adding LiDAR sensor to our simple Robot - gazebo plugin - YouTube](https://www.youtube.com/watch?v=jJzzw2jk-lY)