

Mobile Robotics Minor Robotics Architecture Final Assignment

Ernesto Tersluijsen

1667131

2024-11-07

Table of Contents

Introduction.....	3
Functional Requirements.....	3
Use case.....	3
Code Structure and Design.....	4
Repository Overview.....	4
Function Documentation.....	5
Wheel Odometry.....	5
Joint State Publisher.....	6
Program Demonstration.....	7
Testing Process.....	7
Video Explanation.....	7
Challenges and Solutions.....	7
Performance and Optimisation.....	8
Future Work and Improvements.....	8
Conclusion.....	8
References.....	9

Introduction

For Robotics Architecture we have been given the assignment of programming a smartcar application. This smartcar is equipped with an IMU and a LiDAR it can use in Gazebo, which it uses to localise itself in the world.

The goal of this project is to to familiarise ourselves and learn about the different functionality in ROS2 and robot localisation. This assignment also touches on the basics of nav2, xacro and URDF.

Functional Requirements

Based on the assignment description containing the steps that need to be completed to achieve the desired result, I have created the following requirements.

Code	Requirement
FR1	Each package must be able to build with colcon.
FR2	The robot model must adhere to the definition described in Assignment doc. 2.1.2.
FR3	The robot must be visualised in Rviz and Gazebo.
FR4	The robot must use a LiDAR & IMU for localisation.
FR5	The robot must publish the sensor data on their respective topics
FR6	The robot must be able to ride a route using the localisation

While these requirements do not capture all the intricacies of the assignment description, they give a rough outline of what the program must be able to adhere to.

Use case

















The main use case the user will perform with the program will be to select an estimate start position and an end position in nav2, which should cause the smartcar in the simulation to drive to the specified goal.

Code Structure and Design

Repository Overview

The following image shows a general overview of the repository. The repository contains different packages, including the provided ackermann_msgs and car_gazebo_plugin packages. The created packages are the smartcar_msgs package and the smartcar_simulation package. In this repository most files of importance are located in the smartcar_simulation package. Apart from packages a couple scripts have been provided; the build, start_rviz, start_gazebo, start_localisation and start_nav2 scripts, these scripts simplify the testing process by automatically building all the required packages, converting the xacro to an URDF file, sourcing the setup script and running their respective launch files. The repository can be found on GitHub with the following link:

<https://github.com/ErnestoTersluijsen/smartcar-simulation>

 ErnestoTersluijsen added document and doxygen 099cea0 · 25 minutes ago 27 Commits		
 .vscode	amazing commit message	last month
 ackermann_msgs	added sensors, gazebo & driving	2 months ago
 car_gazebo_plugin	added sensors, gazebo & driving	2 months ago
 docs	added document and doxygen	25 minutes ago
 smartcar_msgs	Initial commit	2 months ago
 smartcar_simulation	added document and doxygen	25 minutes ago
 .clang-format	odometry code done???	last month
 .gitattributes	Initial commit	2 months ago
 .gitignore	added document and doxygen	25 minutes ago
 README.md	added document and doxygen	25 minutes ago
 build.sh	not finished code	last month
 start_gazebo.sh	fixed autocomplete & stuff	last month
 start_localisation.sh	added state publisher	3 weeks ago
 start_nav2.sh	current code	20 hours ago
 start_rviz.sh	fixed autocomplete & stuff	last month

Function Documentation

Wheel Odometry

```
25
26 void WheelOdometry::update_position(int32_t rpm, double steering_angle, rclcpp::Time current_time)
27 {
28     if (position_.timestamp.nanoseconds() == 0)
29     {
30         position_.timestamp = this->get_clock()->now();
31         return;
32     }
33
34     rclcpp::Duration time_step = current_time - position_.timestamp;
35     position_.timestamp = current_time;
36
37     double linear_velocity = calc_linear_velocity(rpm, wheel_diameter_);
38     double angular_velocity = calc_angular_velocity(rpm, steering_angle);
39
40     double phi = position_.phi + angular_velocity * time_step.seconds();
41     double x = position_.x + linear_velocity * std::cos(phi) * time_step.seconds();
42     double y = position_.y + linear_velocity * std::sin(phi) * time_step.seconds();
43
44     position_.phi = phi;
45     position_.x = x;
46     position_.y = y;
47
48     nav_msgs::msg::Odometry message;
49
50     message.header.stamp = get_clock()->now();
51     message.header.frame_id = "odom";
52     message.child_frame_id = "base_link";
53
54     message.pose.pose.position.x = x;
55     message.pose.pose.position.y = y;
56
57     tf2::Quaternion q;
58     q.setRPY(0, 0, phi);
59     message.pose.pose.orientation = tf2::toMsg(q);
60
61     message.pose.covariance = pose_covariance_;
62
63     message.twist.twist.linear.x = linear_velocity;
64     message.twist.twist.linear.y = 0.0;
65     message.twist.twist.linear.z = 0.0;
66
67     message.twist.twist.angular.z = angular_velocity;
68
69     message.twist.covariance = twist_covariance_;
70
71     publisher_->publish(message);
72 }
```

The code above shows the update position function, which calculates and publishes the odometry messages “/smartcar/wheel/odom” topic. The function has arguments for the RPM, the steering angle and the current time. RPM and steering angle both come from the smartcar status messages from the car_gazebo_plugin and current_time is calculated by using the previous message timestamp. The first thing the function does is check if it’s been called before by using the saved timestamp. If the timestamp is zero, or rather not saved before, the timestamp will be set to the current time and the function will return nothing.

The next time the function is ran the time difference between the messages will be calculated and the functions to calculate linear and angular velocity are called. Next the linear and angular velocity are used to calculate phi, x and y. Next a new odometry message is created and filled with the correct headers. After the headers are set correctly the appropriate values will be added to the odometry message and sent out by the publisher.

Joint State Publisher

```
28
29 void JointStatePublisher::update_wheel_position(int32_t engine_speed, double steering_angle)
30 {
31     rclcpp::Time current_time = get_clock()->now();
32     rclcpp::Duration duration = static_cast<rclcpp::Time>(msg_.header.stamp) - current_time;
33     msg_.header.stamp = current_time;
34
35     double radian_diff = (engine_speed / 60) * (M_PI * 2) * (static_cast<double>(duration.nanoseconds()) / 1e-9);
36
37     for (size_t i = 0; i < msg_.position.size(); ++i)
38     {
39         if (i == 0 || i == 2)
40         {
41             msg_.position.at(i) = steering_angle;
42         }
43         else
44         {
45             msg_.position.at(i) = std::fmod(msg_.position.at(i) + radian_diff, M_PI * 2);
46         }
47     }
48 }
49
```

The update wheel position function in the joint state publisher updates the actual positions of the wheels in Rviz using the position of the wheels from Gazebo. The function takes in the engine speed and the steering angle from the vehicle status messages of the smartcar. It then gets the current time and compares it to the time the previous message was sent, which is still stored in `msg_.header.stamp`. Using the difference, the rotation of the wheels is calculated and set for all the wheels and the steering angle is set. The message is then sent outside of this function from.

Program Demonstration

Testing Process

The testing process for the smartcar simulation will be to mainly check the use case if the use case described earlier in the document. This is due to the fact many of the functional requirements are hard to explicitly show in the video, but are part and required to be able to achieve the described use case.

Video Explanation

Video is available on the previously linked GitHub repository.

Direct link to the video: <https://github.com/ErnestoTersluijsen/smartcar-simulation/blob/main/videos/robotics%20architecture.mp4>

At the start of the video the log build and install folders have been deleted, so when the script `start_nav2.sh` is run, the program is compiled using colcon build. After the program is build and everything is sourced, Gazebo and Rviz2 are launched. The smartcar and the world are visible in Gazebo and the Rviz2 windows shows the nav2 interface. The estimated starting position and goal are selected and the smartcar starts driving. The goal was placed there to show that all the smartcar makes use of both the IMU and the LiDAR. Then eventually the smartcar reaches it's goal and I close the program.

This video does not contain the visualisation of the robot in Rviz, as Rviz is being used for nav2, but when another launch file for example `localisation.launch.py` the visualisation of the smartcar with it's wheels and steering angle are clearly shown.

Challenges and Solutions

One problem I encountered when I was working on the nav2 launch file absolutely baffled me. I started off making my nav2 launch file by copying the contents of the previous launch file, `localisation`, and proceeded to add the action to run the `nav2_bringup` launch file from within my nav2 launch file. For the `nav2_bringup` launch file to work it was necessary to pass the `smalltown_world.yaml` file. I proceeded to use the same way I declared arguments before and added it to the launch description.

When I then ran the launch file I encountered a problem, the argument "map" was empty and thus the `nav2_bringup` launch file threw an error. After many hours of debugging and trying every way I knew to pass arguments in a launch file, it turned out that for no logical reason, if I placed the `nav2_bringup` launch file above the `gazebo.launch.py` launch file the argument would not be empty. I still do not fully understand why this behaviour is present, as the `gazebo` launch file shouldn't affect how an argument is passed to a launch file but I was able to move past it and finish the nav2 functionality.

Performance and Optimisation

After looking through the different nodes written, I personally can not find optimisations I explicitly used. I thought the goal of each node was quite clear and used my previous experience of working with C++ and ROS2 to write a pretty standard program which is easy to read and understand and thus not focused on many performance optimisations. One tiny performance optimisation I could have used is passing more arguments to functions by reference instead of by value and checking which arguments and functions can be const. These optimisations I would consider good practise in programs where large objects are passed to functions, but considering the functions I wrote only use primitive types as arguments the difference in performance would be in the terms of a couple CPU cycles.

Future Work and Improvements

One of the first things I would improve about my project would be making the code more versatile. Currently nodes like `wheel_odometry`, are initialised with covariance matrices specific to the current smartcar. These matrices are at least set in the initialiser list of the class, but I would ideally want to load these settings from a configuration file, this would mean the source code won't have to be changed and recompiled for a different smartcar which needs different matrices.

Conclusion

Looking at what I have delivered I do believe I have achieved all the requested requirements stated in this document and the assignment document.

Personally I have used ROS2 before during the course of my study, which did give me a large advantage during this course. Luckily there were multiple subjects I do before or wasn't the most familiar with. For example at the start of the assignment there was a focus on working on a xacro URDF model, which I didn't do before and using nav2 for localisation and routing was a new experience. As a result I'm quite content with everything I delivered for this assignment.

References

<https://github.com/ErnestoTersluijsen/smartcar-simulation>

<https://docs.ros.org/en/humble/index.html>

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

https://docs.nav2.org/getting_started/index.html

https://github.com/ros-navigation/navigation2/tree/main/nav2_bringup