Politecnico di Torino

Department of Electronics and Telecommunications

Master's degree in Mechatronic Engineering

2024/2025

Sensors, Embedded Systems and Algorithms for Service Robotics

Report 3 - Final project

Subject B: Obstacle avoidance and robot following

Date 24/01

Group n° 1

Gabriel Cunha – S335404

Baptiste Hooghe – S334817

Thiago Levin – S334838

Paul Novoa – S339335

1 OBJECTIVE

Objective of the Project

The project aims to develop a ROS 2 package enabling the execution of planning and control algorithms on a mobile robot. The project includes 3 key aspects for the robot's autonomy and navigation purposes.

Firstly, a path planning method will allow the robot to get an optimal trajectory to a target within a mapped environment. This includes creating precise control commands to ensure accurate movements.

Secondly, obstacle avoidance and dynamic goal chasing are implemented using the Dynamic Window Approach (DWA). This ensures that the robot can autonomously navigate around obstacles while continuously adapting to follow a moving target, even in changing environments.

Our group will exclusively work on Track B, which involves the development of a navigation script to follow another moving robot. The moving target robot will be detected visually using landmarks tag placed on it, allowing the navigation system to track its position accurately with a camera.

The Dynamic Window Approach will be implemented within a ROS 2 node. The heading term will be computed based on the estimated X-Y coordinates of the detected robot. The controller will operate at a frequency of 15 Hz, using a ROS 2 timer to manage the DWA sequence efficiently. The robot will rely on LiDAR scan data for obstacle perception, processing this data to remove invalid readings, limit the detection range, and extract obstacle positions relative to the robot’s pose. A safety function will stop the robot if a collision tolerance is reached.

Additionally, the goal manager will dynamically update the navigation target during operation. The system will evaluate navigation outcomes as either successful goal achievement, collision, or timeout, with periodic feedback on the robot’s progress toward the goal.

Beyond the standard DWA, enhancements will include:

- Modifying the velocity scoring term to slow down the robot as it approaches the target.
- Adding a term to the cost function that maintains optimal distance and visibility of the target robot to ensure effective tracking.

Real-world experiments will involve the navigation system following a teleoperated or autonomously controlled target robot across different obstacle-laden trajectories. Performance metrics such as success rate, trajectory precision, obstacle distances, and velocity profiles will be analyzed over multiple trials.

2 DESIGN

2.1 DWA NODE

The primary goal of DWA is to move the robot towards a goal while avoiding collisions and ensuring smooth movement.

- **Key Functions and Classes:**

Node Class: Managing communication with the robot's sensors and actuators.

- **scan_callback**: Processes incoming LiDAR data to update obstacle information.
- **odom_callback**: Updates the robot's current position based on odometry data.
- **goal_pose_callback**: Dynamically updates the robot's goal position.
- **update_robot_pose**: Computes and publishes velocity commands based on the current state and the goal.
- **publish_obstacle_markers**: Visualizes detected obstacles for debugging in RViz.

DWA Class: Creating and selecting trajectories

- **go_to_pose**: Moves the robot towards the goal while avoiding obstacles.
- **compute_cmd**: Computes the velocity commands based on the robot's current state, goal, and obstacles.
- **get_trajectories**: Generates possible trajectories based on the robot's velocity and acceleration.
- **evaluate_paths**: Scores the generated trajectories to determine the optimal path.

2.2 CONTROLLER NODE

The **Controller node** is tasked with guiding the robot from its initial position to a designated goal while avoiding obstacles.

At the heart of the Controller node, **DWA function** evaluates potential trajectories within a dynamic window determined by the robot's velocity and acceleration constraints. This evaluation optimizes three key aspects: heading alignment with the goal, velocity maximization for efficiency, and obstacle avoidance to ensure safety. These priorities are balanced using a cost function:

$$J = \alpha \cdot heading + \beta \cdot velocity + \gamma \cdot obstacle\ avoidance$$

Here, α, β, and γ are tunable parameters that allow for balancing these priorities based on the specific requirements of the environment and task.

The controller computes paths over a fixed time horizon (ex: 2.0 s) and selects the one with the lowest cost. Velocity commands are then published to the robot's actuators.

For collision avoidance, the Controller node relies on **LiDAR data** to detect nearby obstacles. A safety mechanism is in place to halt the robot if an obstacle is detected within a predefined collision tolerance (ex: 0.2 m). Visualization is a key feature of the Controller node, enabling it to monitor the robot's trajectory, velocities, and path during navigation, mainly for debugging purposes.

The custom **DWA class** encapsulates the core logic for trajectory evaluation and motion planning. The system's configuration parameters, including velocity limits, angular acceleration, and weight factors (α, β, γ), allows fine-tuning for specific environments and tasks.

In practice, the **Controller node** has demonstrated its ability to navigate the robot to the target goal efficiently while avoiding obstacles. The robot's path and velocity profiles, visualized during operation, provide valuable feedback for optimizing navigation strategies and enhancing overall system performance.

3 EXPERIMENTS

3.1 SIMULATION

In the gazebo simulation environment, the robot was tested in diverse scenarios, such as narrow hallways, open spaces, and cluttered areas. The robot was tasked with navigating to both stationary and moving goals, allowing us to verify its ability to adapt to different conditions. Parameter configuration was evaluated over multiple runs to observe its impact on robot behaviour.

The tuning process began by emphasizing the /vel parameter, set to high a high value to bring fast movement. This initially caused the robot to follow the target flawlessly but this was heavily relied on the low /dist parameter and not representative of complex path cases. Then, /dist parameter has been adjusted from 2 to 1, improving obstacle avoidance while maintaining smooth navigation.

Additionally, /heading parameter was reduced to 0.4, to prevent the robot from steering too close to obstacles, and /vel parameter increased to 2 to restore faster speed, leading to balance between speed and safety.

To evaluate deceleration, tests were performed with both stationary and moving goals. The weight /target parameter was set to 0.1, corresponding to a stopping distance of 10 cm. The robot consistently decelerated smoothly and stopped precisely at the desired distance. These tests validated the robot's ability to handle dynamic environments effectively.

3.2 Real Robot

Real-world experiments were conducted using a turtlebot equipped with LiDAR and odometry sensors in environments similar to the simulation setup following a real robot equipped with landmarks signs on its backside.

The parameters tuned in the previous simulation phase were directly transferred to the real robot without significant modifications. The robot successfully avoided collision, but failed to consistently follow the second robot across multiple runs and parameter tuning (~ 5 runs). Despite repeatedly trying to tune the parameters to obtain better results, the performance of the robot in the real world remained rather poor. It was observed during experiments that the robot failed to consistently follow the second robot during turn maneuvers, especially if the turn was too tight.

On the other hand, deceleration tests in real-world scenarios showed that the robot accurately stopped at the desired distance of 10 cm from both static and moving goals. The /target parameter ensures smooth deceleration and safe behavior near the target. This confirmed that in the limited case of straight paths, the robot was able to follow the goal pretty reliably.

4 Results and Discussion

In this section, we will present and discuss the results of our experiments. The performance of the DWA algorithm will be measured using the following performance metrics where applicable:

- Distance Root Mean Square Error (RMSE): RMS of the error between desired and observed distance to the dynamic goal
- Bearing RMSE: RMS of the dynamic goal bearing
- Mean Obstacle Distance: average distance of detected obstacles
- Minimum Obstacle Distance: minimum distance of detected obstacles
- Tracking Time Rate: percent of the total experiment time where the goal bearing was within pi/6 rad (30 degrees)
- Goal Success Rate: percent of dynamic goals that were reached by the robot within 1 cm

4.1 Task 1

In task 1, the objective was to perform a simulation of the DWA in Gazebo. The cost function for this task includes obstacle avoidance, goal tracking, and velocity. As shown in table 1, the performance metrics for task 1 are good, especially in the time tracking rate and goal success rate. This means that the robot was able to maintain sight of the dynamic goal and reach it consistently throughout the simulation. This can be confirmed in the XY plot of the robot trajectory in figure 1. Clearly, the robot is able to track the dynamic goal and reach it at every step.

The RMSE distance metric shows that the robot followed the dynamic goal too closely and did not manage to maintain the desired 2 m distance from the goal. The bearing RMSE metric being less than pi/6 confirms that the robot is able to track the dynamic goal visually and keep it in its line of sight. The mean obstacle distance metric shows that the robot was able to navigate around the obstacles that were presented and maintain a safe distance. The minimum obstacle distance metric shows that there may

have been a close call at some point, however this may also be due to having to navigate a narrow hallway where there is no other option.

RMSE Distance (m)	RMSE Bearing (rad)	Mean Obstacle Distance (m)	Minimum Obstacle Distance (m)	Tracking Time Rate	Goal Success Rate
1.85	0.27	1.81	0.30	99.2%	99.0%

Table 1. Task 1 Performance Metrics

The command profile generated by the DWA for the linear and angular velocities are shown in figure 2. Clearly, the command profile is jittery, which may be undesirable as it can produce undesirable results when applied to a real system, as these systems may not be able to accelerate and decelerate as fast as the DWA algorithm is demanding it does. This in general is one of the problems of using velocity as a command input.

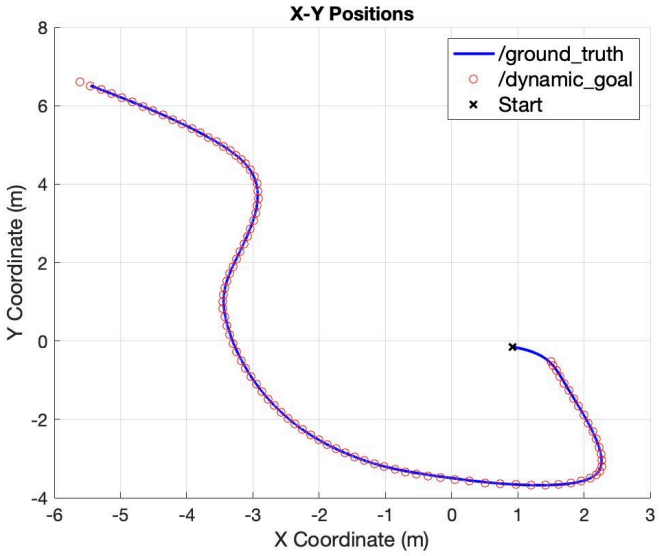

Figure 1. Task 1 XY Trajectory

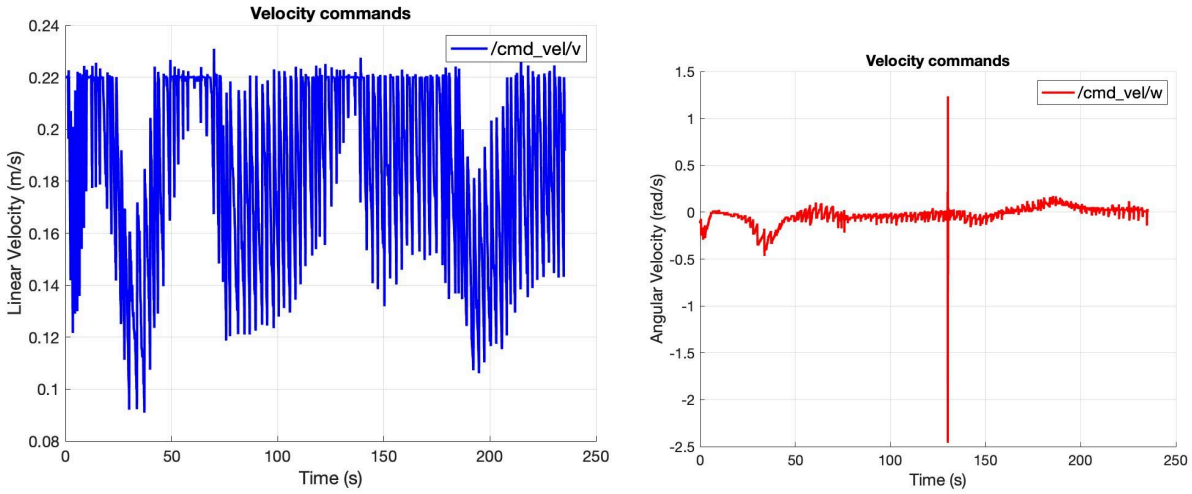

Figure 2. Linear Velocity Commands (left) and Angular Velocity Commands (right)

4.2 Task 2

In task 2, the objective was to perform a simulation of the DWA in Gazebo. The cost function for this task includes obstacle avoidance, goal tracking, velocity, and, additionally, distance to the goal. With the addition of the distance to the cost function, the hope is that the robot will maintain a distance of 2 m with the dynamic goal throughout the simulation.

As shown in table 2, the performance metrics for task 2 are good, especially in the time tracking rate and goal success rate, although the goal success rate went down slightly compared to task 1. This means that the robot was able to maintain sight of the dynamic goal and reach it consistently throughout the simulation. This can be confirmed in the XY plot of the robot trajectory in figure 3. Clearly, the robot is able to track the dynamic goal and reach it at every step.

The RMSE distance metric improved from task 1 by about 0.15 m. Clearly, the addition of the distance to the dynamic goal to the cost function helped improve the performance of the robot. The bearing RMSE metric also improved. This may be due to the fact that since the robot was slightly further away from the dynamic goal, it was easier to maintain the 0 bearing. The mean obstacle distance metric also improved, showing that the robot was able to better navigate around the obstacles. The minimum obstacle distance metric shows that, again, there may be a narrow hallway to navigate where there is no other option.

RMSE Distance (m)	RMSE Bearing (rad)	Mean Obstacle Distance (m)	Minimum Obstacle Distance (m)	Tracking Time Rate	Goal Success Rate
1.66	0.23	2.11	0.30	99.0%	97.5%

Table 2. Task 2 Performance Metrics

The command profile generated by the DWA for the linear and angular velocities are shown in figure 4. Similarly to task 1, the command profile is jittery. In addition, there are some very large spikes in the requested commands at around 125 seconds. This may be due to the additional distance requirement on

the DWA which could have provoked it to request a very large command. Again, this sort of behavior can be problematic when it comes to a real world robot because the robot may have enough acceleration capacities to keep up with the large swings in velocity commands.

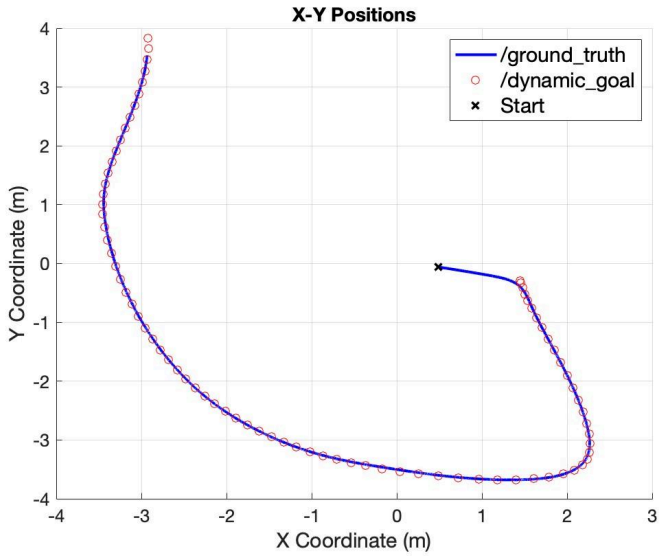

Figure 3. Task 2 XY Trajectory

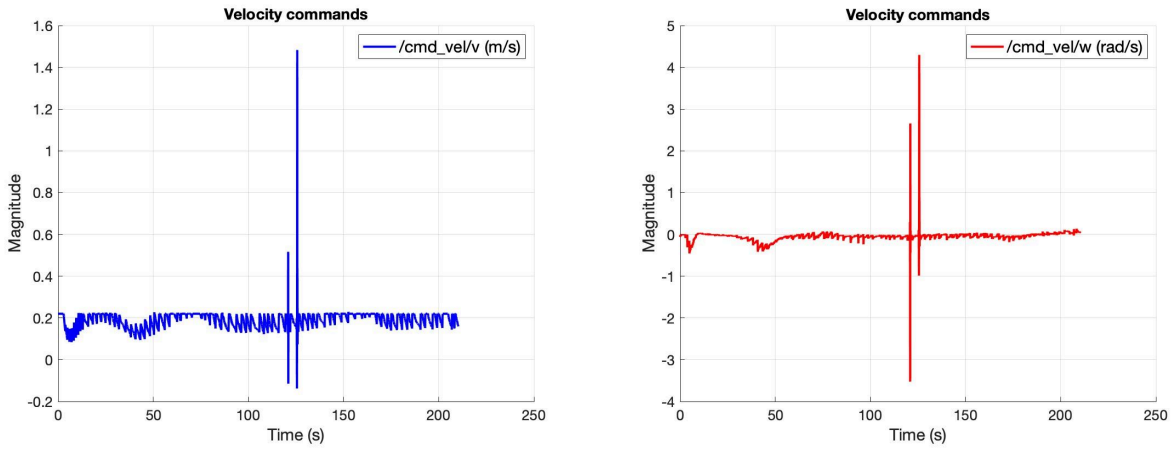

Figure 4. Linear Velocity Commands (left) and Angular Velocity Commands (right)

4.3 Task 3

In task 3, the objective was to implement the DWA algorithm on a real robot and try to have it follow a second robot. The cost function for this task includes obstacle avoidance, goal tracking, velocity, and, additionally, distance to the goal, exactly as in task 2. In order to calculate the performance metrics, the XY position of the second robot was estimated based on the bearing and range measurements from the camera as well as the yaw measurement from /odom.

The performance metrics for task 3, runs 1 and 2 are shown in table 3. Clearly, the real-world setting proved to be much harder for the DWA algorithm, as both the tracking time and goal success rates are

much lower than in the simulations. The RMSE distance metric is relatively good, even better than in tasks 1 and 2. However, the RMSE bearing metric is much poorer at nearly triple the value of task 2. The mean obstacle distance is better, but this may be due to the fact there were much less obstacles to avoid in the real world setting. The overall drop in performance shown in the metrics of table 3 can also be observed in the XY plot of the trajectories shown in figure 5. It is important to note that, since the position of the second robot is estimated based on the camera measurements, it is not necessarily accurate. Nevertheless, it can be seen from the trajectories that the robot struggles to follow the dynamic goal accurately from the beginning, and it loses the second robot even further when it executes the turn.

Run	RMSE Distance (m)	RMSE Bearing (rad)	Mean Obstacle Distance (m)	Minimum Obstacle Distance (m)	Tracking Time Rate	Goal Success Rate
1	1.37	0.60	3.03	0.24	80.0%	39.6%
2	1.52	0.50	3.16	0.16	85.1%	71.6%

Table 3. Task 3 Performance Metrics

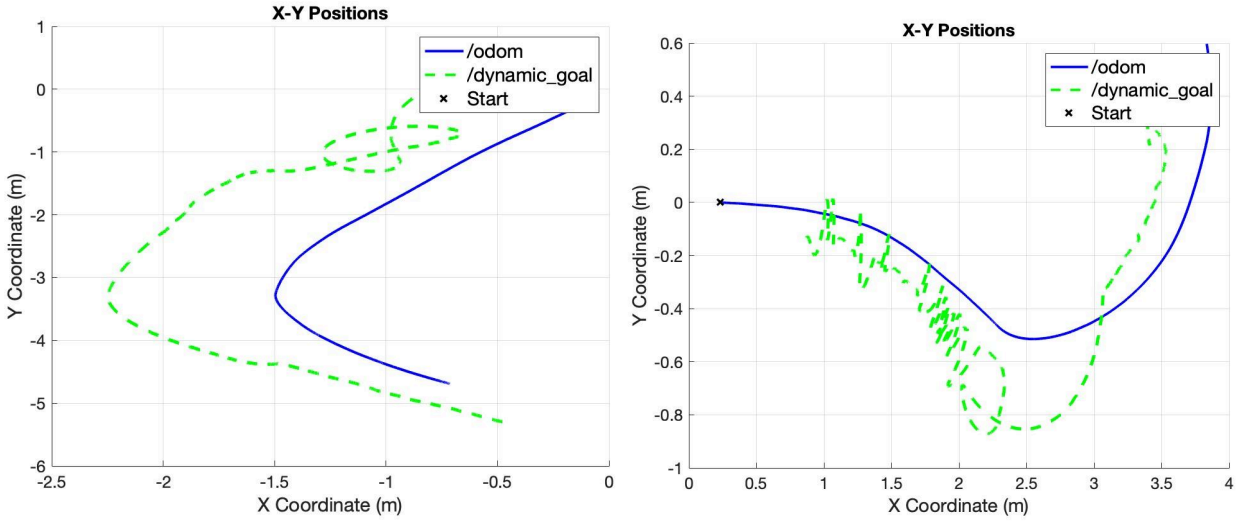

Figure 5. Task 3 XY Trajectories for Run 1 (left) and Run 2 (right)

The velocity profiles for runs 1 and 2 are shown in figures 6 and 7, respectively. Although the commands are much less jittery than they were in tasks 1 and 2, there are still very large jumps in the requested velocity profiles, especially in run 1, where they are more frequent. This may be one of the reasons that the performance was poor during the real world testing of the DWA algorithm. Large velocity jumps as displayed in the data are unrealistic for real world systems, and may cause problems. In addition, since the DWA algorithm is based on having a good model of the system for control sequence optimization, discrepancies with the real world can severely affect the performance, and not including the acceleration needed by the robot to reach the requested speed in the model may have been one of those discrepancies.

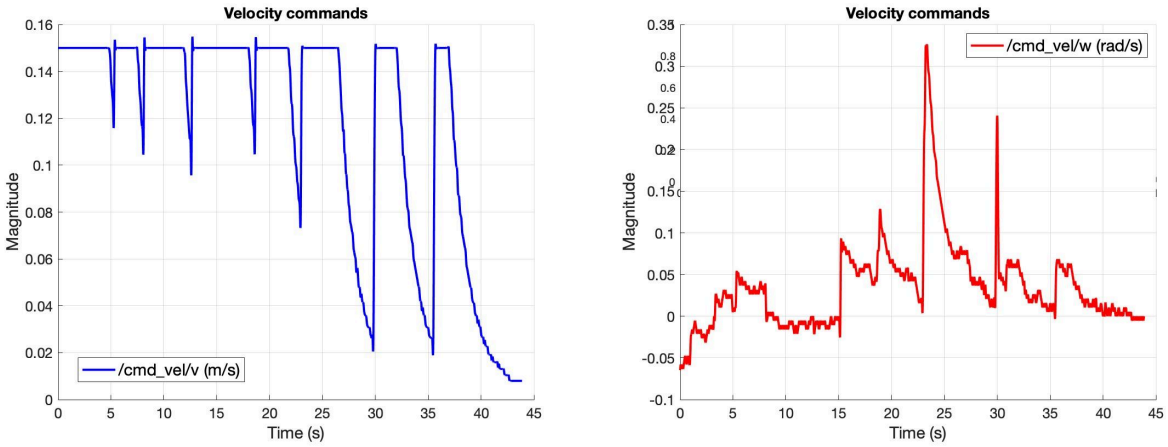

Figure 6. Run 1 Linear Velocity Commands (left) and Angular Velocity Commands (right)

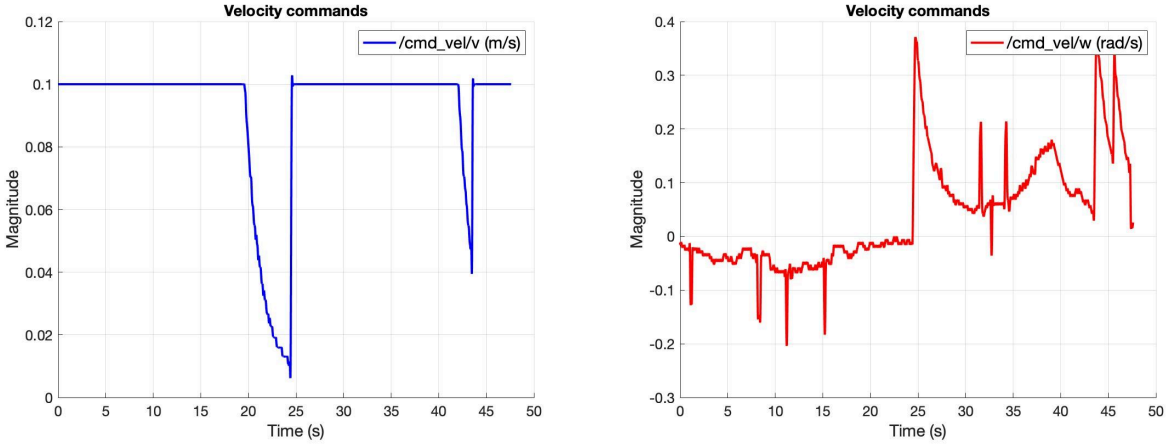

Figure 7. Run 2 Linear Velocity Commands (left) and Angular Velocity Commands (right)