Obstacle Avoidance for a Car-like Robot using the Dynamic Window Approach

Luciana Ercolanese - P38000197

February 7, 2025

Field and Service Robotics - Final Project

Abstract

This project focuses on the autonomous navigation of a car-like robot using the Dynamic Window Approach (DWA). The method accounts for the geometry and dynamics of the robot, ensuring the selection of physically feasible control inputs. The algorithm was tested in simulation using Simulink, performing multiple navigation tasks on three different maps. The results demonstrate the effectiveness of the DWA in handling real-world constraints while achieving efficient path planning.

1 Introduction

Obstacle avoidance techniques are a fundamental component of autonomous navigation systems, which enables robots to plan a collision-free trajectory towards a predetermined target. This type of planning is essential to ensure that the robot operates safely and efficiently in environments cluttered with static and dynamic obstacles. Beyond simply avoiding collisions, the planned trajectory must adhere to the robot's inherent constraints, including its kinematic, dynamic, and geometric limitations. These constraints define the feasible movements of the robot and significantly influence the overall trajectory planning process.

For car-like robots, which are fully non-holonomic under the assumption that the wheels roll without slipping, additional considerations arise. Non-holonomic constraints restrict the robot's instantaneous motions, such as lateral movements, making certain configurations temporarily unreachable. However, despite these restrictions, the robot retains global controllability, which means it can eventually navigate to any point in its configuration space.

The Dynamic Window Approach (DWA) offers a powerful solution to these challenges by integrating constraints directly into the trajectory planning process. Unlike other techniques that may require separate stages to handle kinematic and dynamic constraints, DWA inherently considers these factors during decision making. By evaluating the robot's possible velocities within a dynamically feasible window, this method ensures that all selected control inputs are physically realizable and compliant with the robot's operational limitations. Consequently, DWA streamlines the planning process while maintaining robust performance in various navigation tasks.

2 Methodology

The navigation system for the car-like robot is structured into three interconnected modules: the vehicle model, the workspace perception system, and the Dynamic Window Approach (DWA). The block diagram is shown in Figure 1, inspired by the design presented in [1].

The workspace perception module simulates a LIDAR sensor to acquire real-time data about obstacles in the robot's environment. This data is processed into an obstacle representation and passed to the Dynamic Window Approach (DWA) block. The DWA also takes as input the robot pose in the

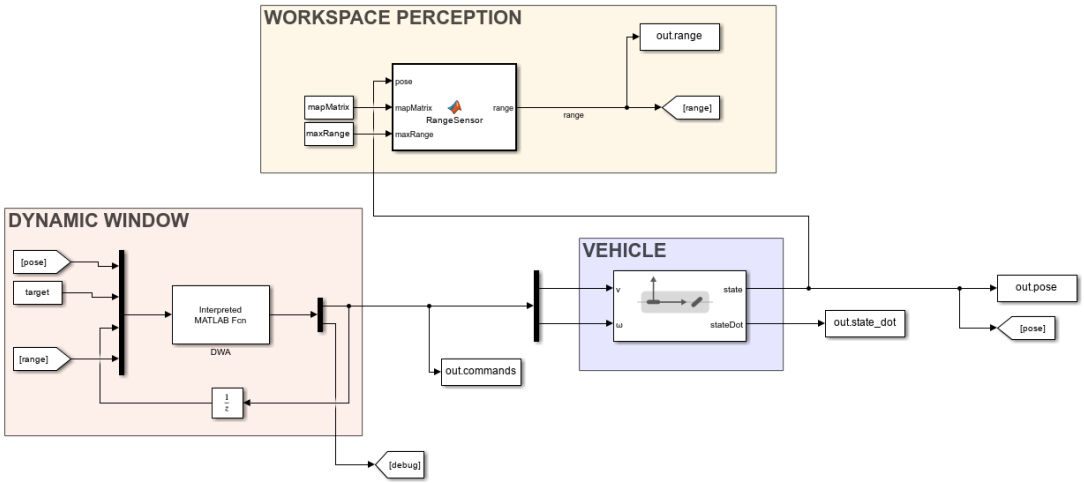

Figure 1: Block diagram of the navigation system.

world frame, provided by the model of the wheeled robot, and computes optimal control inputs, e.g. linear velocity v and angular velocity ω, that ensure collision-free and dynamically feasible motion. The computed commands are sent directly to the vehicle model, implemented using a kinematic bicycle model, to update the robot's pose (x, y, θ) in real time.

2.1 Wheeled Robot Model

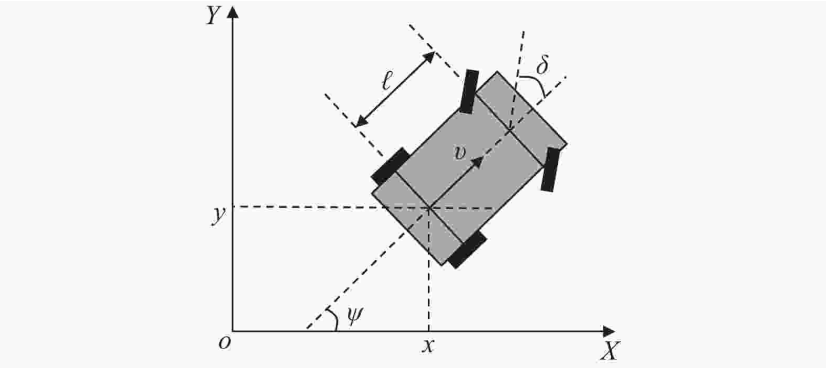

Figure 2: Kinematic bicycle model of the car-like robot.

As shown in Figure 2, the car-like robot is modeled using a kinematic bicycle model, which assumes front-wheel steering and rear-wheel drive. The model operates under the assumption that the wheels roll without slipping, imposing nonholonomic constraints on the robot's motion.

The kinematic equations governing the vehicle's motion are given by:

$$\dot{x} = v \cos\theta, \tag{1}$$

$$\dot{y} = v \sin\theta, \tag{2}$$

$$\dot{\theta} = \omega = \frac{v}{L} \tan\phi, \tag{3}$$

where:

- x and y are the position coordinates of the robot in the world frame,
- θ is the robot's heading (orientation) in the world frame,
- v is the linear velocity,

- L is the distance between the front and rear axles of the robot, and
- ϕ is the steering angle of the front wheels.

2.2 Workspace Perception

The robot navigates within a predefined map, represented as a binary occupancy grid. In this grid, a value of 1 (`true`) indicates an occupied cell, while 0 (`false`) represents a free cell. Although the map is defined a priori, the navigation algorithm does not have prior knowledge of it. Instead, the robot perceives the environment in real-time using a simulated LIDAR sensor.

The LIDAR is implemented as a `rangeSensor` object in MATLAB. It has a field of view of 180° and a resolution of $\pi/128$ radians. The sensor provides distance measurements to obstacles within its range, enabling the robot to build a local understanding of the workspace. For simplicity, no measurement noise was added to the LIDAR simulation.

2.3 Dynamic Window Approach (DWA)

The Dynamic Window Approach (DWA) is a real-time motion planning algorithm used for obstacle avoidance and trajectory optimization. Unlike traditional methods that rely on separately computed global paths and obstacle avoidance, DWA integrates the robot's kinematic and dynamic constraints directly into the trajectory generation process.

The procedure of DWA consists of two main steps: defining the admissible control commands and selecting the optimal control inputs based on a cost function.

2.3.1 Defining Admissible Control Inputs

The first step involves defining a set of control inputs that are feasible for the robot. This set is a subset of the control space, denoted by $\mathcal{U}_R$, and consists of the intersection of three subsets:

1. **Physically reachable velocities**: The subset $\mathcal{U}$ includes the linear velocity $v \in [0, v_{max}]$ and the angular velocity $\omega \in [-\omega_{max}, \omega_{max}]$ that the robot can physically achieve. For simplicity, we assume that the robot cannot move in reverse (i.e., no negative linear velocity). the maximum angular velocity $\omega_{\max}$, which determines the maximum steering rate of the robot,is given by

$$\omega_{\max} = \frac{\tan(\varphi_{\max})}{l}$$

where $\varphi_{\max}$ is the maximum steering angle of the robot, and l is the distance between the front and rear axles of the robot. Therefore

$$\mathcal{U} = \{(v, \omega) \in \mathbb{R}^2 | 0 \leq v \leq v_{max}, -\omega_{max} \leq \omega \leq \omega_{max}\}$$

2. **Safe velocities**: The subset $\mathcal{U}_A$ includes control inputs that generate safe trajectories, meaning that the robot is capable of stopping in time to avoid a collision. Mathematically, this is expressed as

$$\mathcal{U}_A = \{(v, \omega) \in \mathbb{R}^2 | v \leq \sqrt{2 \cdot \text{dist}(v, \omega) \cdot a_v}, \omega \leq \sqrt{2 \cdot \text{dist}(v, \omega) \cdot a_\omega}\ \}$$

where a_v and a_ω are the maximum linear and angular decelerations, respectively, and $\text{dist}(v, \omega)$ is the distance to the nearest obstacle, considering that the robot follows either straight-line trajectories or arcs of a circle with radius $r = \frac{v}{\omega}$. The details of the function $\text{dist}(v, \omega)$, as presented in the work of [3], are included in Section 2.3.3.

3. **Physically reachable control inputs in the next timestep**: The subset $\mathcal{U}_D$ includes the control inputs that are reachable by the robot in the next control timestep, based on its current state. The linear and angular velocities must satisfy:

$$\mathcal{U}_D = \{(v, \omega) \in \mathbb{R}^2 | v \in [v_{current} - a_v \cdot T, v_{current} + a_v \cdot T], \quad \omega \in [\omega_{current} - a_\omega \cdot T, \omega_{current} + a_\omega \cdot T]\}$$

, where $v_{current}$ and $\omega_{current}$ are the current velocities of the robot, a_v and a_ω are the maximum linear and angular accelerations, and T is the timestep. This subset gives the technique its name, as it defines a window of reachable control inputs that is centered around the current inputs and dynamically adapts during the execution, as illustrated in Figure 3.

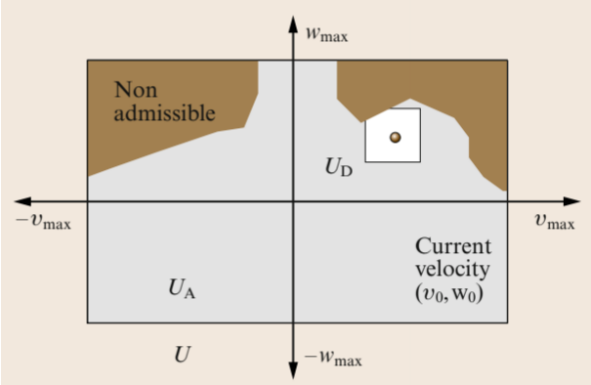

Figure 3: Dynamic Window representation: The big rectangle is the set of control inputs $\mathcal{U}$, within the maximum allowed values. The dot represents the current input given to the robot, and the square surrounding it represents $\mathcal{U}_D$, i.e. the control inputs reachable in a timestep starting from that configuration. Finally, the darker regions are unfeasible control inputs, that would drive the robot to a collision. The figure is courtesy of [4].

The set of candidate control inputs is the intersection of these three subsets, i.e.:

$$\mathcal{U}_R = \mathcal{U}_A \cap \mathcal{U}_D \cap \mathcal{U}.$$

2.3.2 Selecting Optimal Control Control Inputs

The second step of the DWA involves selecting the optimal control inputs, $\mathbf{v}^*$ and ω^*, which maximize an objective function. The objective function is defined as:

$$\text{objective} = k_1 \cdot \text{heading} + k_2 \cdot \text{velocity} + k_3 \cdot \text{clearance}$$

Where:

1. **Heading**: This term represents the alignment between the robot's trajectory and the target direction. It is computed as:
$$\text{heading} = \frac{1 - |\theta'|}{\pi}$$
where θ' is the angle between the robot's heading at the next timestep, θ_{t+1}, and the direction of the target. If we denote the robot's position at the next timestep as (x_{t+1}, y_{t+1}) and the target's position as (x_{target}, y_{target}), then $\theta' = \theta_{t+1} - \tan\left(\frac{y_{target} - y_{t+1}}{x_{target} - x_{t+1}}\right)$. This term is higher when the robot's trajectory is better aligned with the target direction.

2. **Velocity**: This term penalizes low velocities when the robot is far from the target and adjusts its behavior as the robot gets closer to the target. It is computed as follows:

   - When the robot's Euclidean distance from the target is above a certain threshold:
   $$\text{velocity} = \frac{|v|}{v_{\max}}$$
   where v is the linear velocity, and $v_{\max}$ is the maximum allowed velocity.
   - When the robot's Euclidean distance from the target is below the threshold:
   $$\text{velocity} = 1 - \frac{|v|}{v_{\max}}$$

3. **Clearance**: This term ensures that the robot maintains a safe distance from obstacles. It is calculated as:

$$\text{clearance} = \text{dist}(v, \omega)$$

where $\text{dist}(v, \omega)$ represents the minimum distance to a potential collision if the robot follows the trajectory determined by the control inputs v and ω. The details of the function $\text{dist}(v, \omega)$ are provided in Section 2.3.3.

The optimal control inputs, $\mathbf{v}^*$ and ω^*, are the values that maximize this objective function, balancing the trade-offs between heading alignment, velocity, and clearance.

2.3.3 Computation of Distance to Collision considering Robot Geometry

The procedure described in this section is directly adapted from [3]. The method accounts for the polygonal geometry of the robot, ensuring accurate distance calculations to avoid collisions.

The robot is assumed to follow a circular arc trajectory with a radius defined by

$$r = \frac{v}{w},$$

where v and w are the linear and angular velocities, respectively. The position of an obstacle is first transformed into the robot's reference frame, using its relative distance and angle.

With the robot's frame fixed, the obstacle point is assumed to move along a circular trajectory with a radius r_o, computed as

$$r_o = \sqrt{(x_o - r)^2 + y_o^2},$$

where x_o and y_o are the obstacle's coordinates in the robot's frame.

- For the **front side**, the intersection is determined by solving the following system:

$$\begin{cases} (x_c - r)^2 + y_c^2 = r_o^2, \\ y_c = y_{\text{front}}, \\ x_c \in [x_{\text{left}}, x_{\text{right}}]. \end{cases}$$

  Substituting $y_c = y_{\text{front}}$ into the equation of the circle, the solution for x_c is:

$$x_c = r \pm \sqrt{r_o^2 - y_{\text{front}}^2}.$$

- For the **left side**, the intersection is determined by solving the following system:

$$\begin{cases} (x_c - r)^2 + y_c^2 = r_o^2, \\ x_c = x_{\text{left}}, \\ y_c \in [y_{\text{back}}, y_{\text{front}}]. \end{cases}$$

  Substituting $x_c = x_{\text{left}}$ into the equation of the circle, the solution for y_c is:

$$y_c = \pm\sqrt{r_o^2 - (x_{\text{left}} - r)^2}.$$

- For the remaining two sides, the computations are done analogously.

The collision distance d is calculated as:

$$d = \alpha r,$$

where $\alpha = \arccos\left(\frac{\mathbf{P_C} \cdot \mathbf{O_C}}{\|\mathbf{P_C}\| \|\mathbf{O_C}\|}\right)$, $\mathbf{P_C}$ are the coordinates of the collision point obtained above, and $\mathbf{O_C}$ is the vector $\mathbf{O} - \mathbf{C}$, i.e., the position of the obstacle minus the center of rotation of the robot. For each side where a collision is detected, the distance to collision is calculated. The overall distance to collision d_{coll} is defined as:

$$d_{\text{coll}} = \min(d_{\text{front}}, d_{\text{back}}, d_{\text{left}}, d_{\text{right}}),$$

where $d_{\text{front}}, d_{\text{back}}, d_{\text{left}}, d_{\text{right}}$ are the distances to collision for the respective sides of the robot.

This calculation is repeated for all obstacle points detected by the LIDAR sensor, and the minimum distance across all points is selected. Finally, this distance is normalized by dividing it by the maximum range of the LIDAR, $d_{\max} = 12\,\text{m}$:

$$\text{dist}(v, \omega) = \frac{d_{\text{coll}}}{d_{\max}}.$$

This approach ensures a geometrically precise evaluation of the distance to collision, accommodating the robot's shape and its kinematic constraints.

3 Simulations

The proposed method was tested on three different maps, referred to as *Map 1*, *Map 2*, and *Map 3*. Map 1 presents a dense environment with many obstacles and narrow turns, requiring precise maneuvering. Map 2 is larger, with obstacles distributed more sporadically, leaving more free space for navigation. Map 3 is the largest and includes a sequence of target waypoints that the robot must reach in order.

The experiments were conducted using a car-like robot with a maximum velocity of $1.5\,\text{m/s}$. The robot's dimensions are $1.8 \times 0.9\,\text{m}$, matching the model described in [1].

For Map 1, the robot was tested on four different target locations, while for Map 2, it navigated toward two different targets. The corresponding paths are shown in Figures 4 for Map 1, and Figures 5 for Map 2.

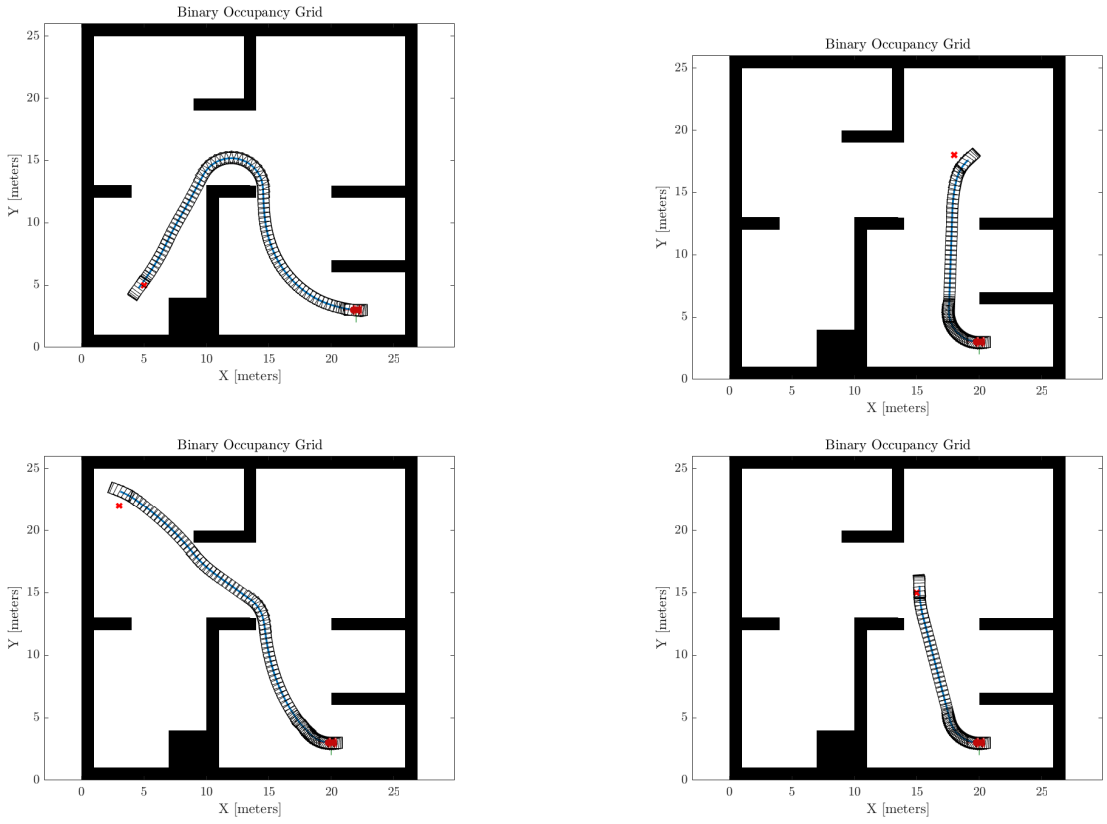

Figure 4: Overview of paths to different targets in Map 1

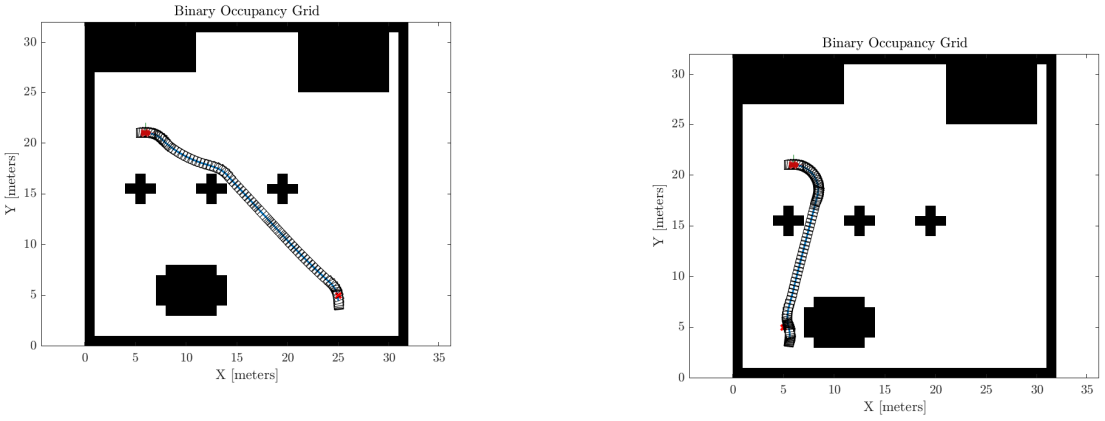

Figure 5: Overview of paths to different targets in Map 2

Map 3 was tested with a predefined sequence of waypoints. Two different velocity settings were considered: the standard robot with a maximum velocity of 1.5 m/s and a high-speed version reaching up to 15 m/s. The results, shown in Figures 7, highlight a key difference: the faster robot exhibits more frequent looping behavior. This occurs because the velocity term in the objective function becomes more dominant, leading the robot to prioritize maintaining speed over reaching the next waypoint efficiently.

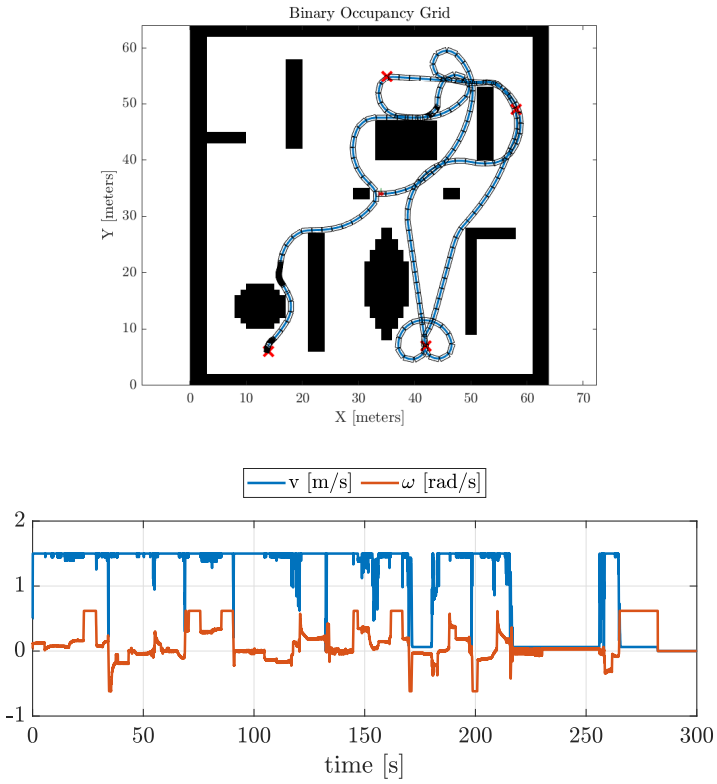

Figure 6: Velocity limited to $1.5m/s$

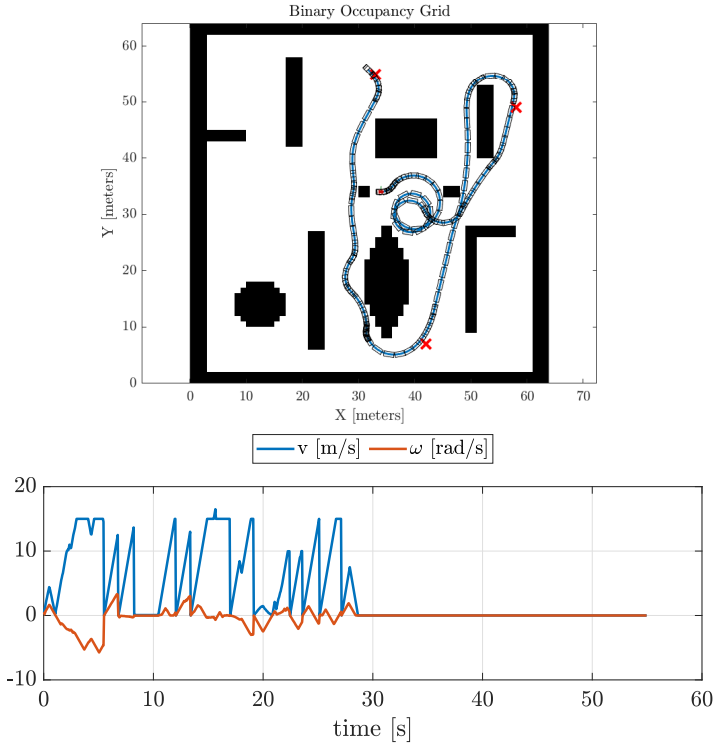

Figure 7: Velocity limited to $15m/s$

With the chosen tuning parameters, the robot does not reach the target with extreme precision. However, this is a necessary trade-off to prevent issues with local minima and excessive self-looping behavior. The advantage of the proposed method lies in its ability to inherently consider the kinematic

and dynamic constraints of the robot, making it adaptable to different vehicle configurations without requiring extensive modifications.

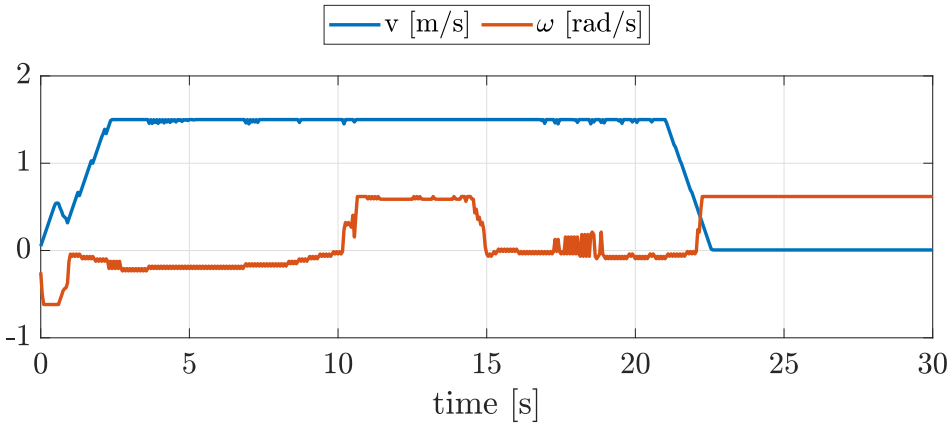

Figure 8: Control inputs for the robot in Map 1, when travelling to the first target

Another important observation, visible in the control inputs (Figure 8), is that the algorithm discourages high velocities near the target but does not explicitly force the robot to stop. As a result, after reaching the target, the robot maintains a nonzero angular velocity ω while keeping its linear velocity close to zero for a short period. This behavior ensures that the heading term in the objective function reaches its maximum contribution (heading = 1).

This does not pose any practical issues, as the commanded linear velocity in this phase remains extremely low (approximately 0.05 m/s). The robot thus effectively stabilizes its orientation without unintended excessive motion.

The results confirm that the algorithm is effective in a variety of environments and under different vehicle constraints. However, as discussed in the next subsection, fine-tuning is essential. The same parameter set does not always generalize across different scenarios, and improper tuning can lead to inefficient paths or local minima issues, that are examined in depth in Section 3.1.

3.1 Tuning of the Control Parameters

The tuning of the parameters k_1, k_2, and k_3 in the cost function is a delicate task, as each parameter influences different aspects of the robot's behavior.

- **Parameter k_1 (Heading)**: This parameter influences the robot's tendency to move directly toward the target. If k_1 is too high, the robot is driven aggressively toward the goal, but this increases the risk of getting stuck in local minima. Finding a balance in k_1 is crucial to ensure smooth trajectory tracking without falling into traps.

- **Parameter k_2 (Velocity)**: This parameter controls the importance of the robot's speed. If k_2 is too low, the robot tends to prioritize heading alignment over velocity, which may result in the robot remaining stationary. Conversely, if k_2 is too high, the heading term loses its influence, and the robot may start oscillating in circular loops instead of heading toward the target.

- **Parameter k_3 (Clearance)**: This parameter adjusts the importance of avoiding obstacles. If k_3 is too low, the robot might pursue the heading and velocity at all costs, even if this results in driving into dead ends or getting too close to obstacles. On the other hand, if k_3 is too high, the robot may become overly cautious, blocking itself in the center of the map and failing to approach the target. This happens because it tries to maintain maximum clearance, even when this means not getting closer to the target.

Examples of these behaviors can be observed in the videos included in the project repository on GitHub[1] and in the figures below.

[1] https://github.com/LErcolanese/fsr_final_project

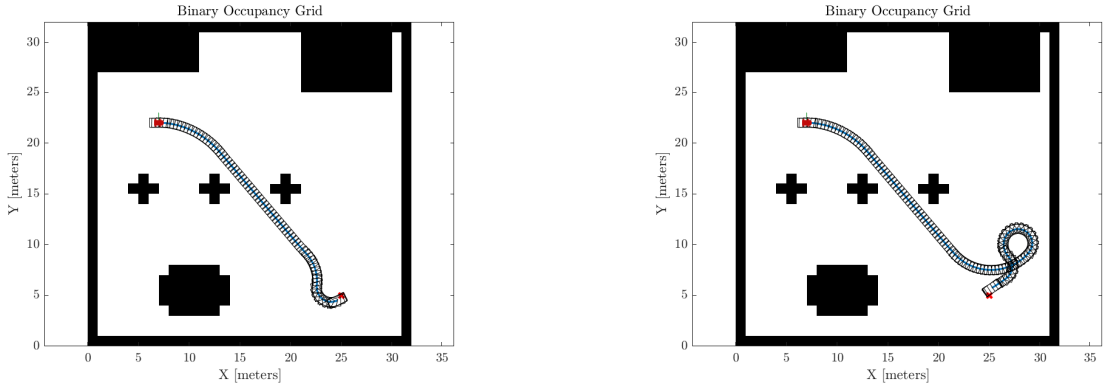

Figure 9: Different tunings of the parameters of the objective function: on the right, a higher value of the velocity gain was set, that resulted in the robot performing a loop around itself before getting to the target.

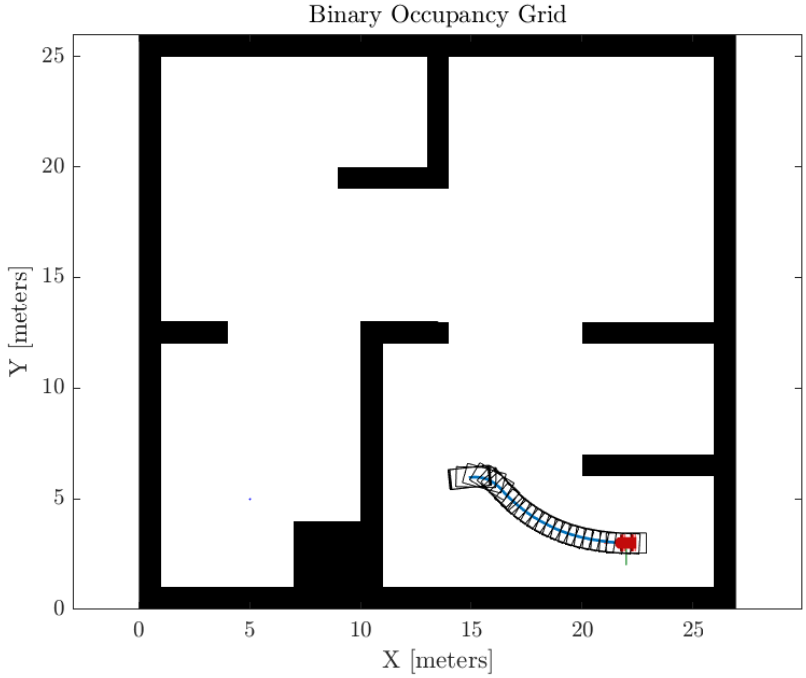

Figure 10: Example of a local minimum

4 Conclusions

The main advantage of the Dynamic Window Approach (DWA) is that, unlike many other obstacle avoidance techniques, it integrates the robot's kinematic, dynamic, and geometric constraints directly into the planning process. This makes the method highly adaptable to different types of robots, as the adaptation to various robot configurations is relatively straightforward. Additionally, the approach is computationally efficient, making it well-suited for real-time applications.

The method has proven effective on a variety of maps and robot characteristics, demonstrating its versatility. However, one of the key challenges lies in the need for fine-tuning. A tuning configuration that works well for one path or environment may not necessarily perform well in a different scenario. Furthermore, the method is susceptible to local minima, which is a known issue for techniques of this

type. As a result, DWA is typically used as a local planner in combination with a global planner, which might leverage prior knowledge about the environment to improve performance.

In the context of global planning, some works, such as [2], propose extending the DWA method by integrating it with a navigation function, allowing it to handle global path planning more effectively. While the Dynamic Window Approach is a robust and efficient method for local path planning, its limitations in terms of fine-tuning and susceptibility to local minima should be addressed in more complex planning frameworks.

References

[1] K. Rebai, O. Azouaoui, M. Benmami, and A. Larabi, "Car-like robot navigation at high speed," in *Proceedings of the 2007 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2007, pp. 2053–2057.

[2] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in *Proceedings of the 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*, vol. 1, 1999, pp. 341–346.

[3] K. O. Arras, J. Persson, N. Tomatis, and R. Siegwart, "Real-time obstacle avoidance for polygonal robots with a reduced dynamic window," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, vol. 3, 2002,

[4] B. Siciliano and O. Khatib, "Robotics and the Handbook," in *Springer Handbook of Robotics*, Springer, 2016, pp. 1–6.