3 Parameter Optimization of Local Planning Algorithms

The performance of local planning algorithms is significantly impacted by their parameter configuration. Parameters influence how algorithms detect and respond to their environment, affecting the robot's ability to avoid obstacles, remain below speed limitations, and follow routes smoothly. However, determining the optimal set of parameters can be difficult due to the variety of scenarios that a robot may experience. Optimizing these parameters is crucial since it increases the algorithm's efficiency and dependability, resulting in safer and more predictable robot behavior. Therefore, this thesis extensively investigates and improves the parameters of three frequently used local planning algorithms in ROS 2: DWB, MPPI, and RPP. Each algorithm has individual features that address different areas of navigation and obstacle avoidance, but they all have the same aim of improving navigational efficiency and safety.

3.1 Parameter Tuning for Dynamic Window Approach (DWB)

A detailed description of the functionality of the DWB algorithm can be found in 2.2.3. Next, an in-depth discussion of the DWB parameters will be presented in the following sub-sections.

3.1.1 DWB: Forward Simulation

DWB is a reactive controller that adjusts the path of the robot in real-time to avoid obstacles based on sensor inputs and local maps. It works independently of global planning by using critic functions, which are heuristic methods, to analyze and select optimal trajectories depending on the local environment, ensuring efficient and collision-free navigation. The algorithm starts by sampling the control space of the robot $(dx, dy, d\theta)$ using the vx, vy, and $vtheta$ parameters. Given that rotation requires more complicated control than simple linear motion, a higher number of velocity samples in the angular direction N_θ is preferred [38] compared to the number of velocity samples in the x velocity direction N_x. Specifically, N_θ is set to 40 while N_x is set to 20. The velocity samples in the y direction, N_y, are set to 0 because the robot does not generate velocity samples in the y-direction. In contrast, omni-directional robots, which account for lateral motion, require configuring the number

of velocity samples in the y-direction, often matching N_x. With these settings, the planner generates a total of 800 trajectories, as calculated by the equation 5, where N_{traj} represents the number of generated trajectories.

$$N_{\text{traj}} = N_{\text{x}} \times N_{\theta} \tag{5}$$

Each velocity sample is simulated as if it applies to the robot over a specified time interval controlled by the simulation time t_{sim}. Each trajectory is forward simulated by setting this parameter. Setting t_{sim} to a value below 1.5 s resulted in sub-optimal trajectories in scenarios involving confined spaces. This is because a shorter simulation time restricts the predictive horizon during forward simulation, causing the planner to evaluate only the immediate future of each trajectory. Therefore, this reduced foresight may not provide enough information to determine the optimal trajectory, thus the robot gets stuck or even could collide. As a result, configuring $t_{\text{sim}} = 1.5\,\text{s}$ showed great results in immediate response and minimizing the trade-offs. On the other hand, a value greater than 2.5 s creates longer trajectory arcs. While this broad view is useful in free places, it can become restrictive in dynamic or congested situations. The robot sticks to a longer path before re-planning and adjusting based on current sensory information. This makes it difficult to respond quickly to unexpected changes in the environment, such as dynamic obstacles.

When configuring the t_{sim} parameter value it is crucial to consider the size of the local costmap. If the robot is moving with a speed of $0.5\,\text{m}\,\text{s}^{-1}$, t_{sim} of 3 s, and a local costmap size of (3 m x 3 m) (width and height respectively). Calculating the maximum achievable velocity (V_{max}) within the local costmap size S_{LCM} would be $0.5\,\text{m}\,\text{s}^{-1}$ using the following equation:

$$V_{\text{max}} = \frac{S_{\text{LCM}}/2}{t_{\text{sim}}} \tag{6}$$

As a result, the robot would reach the maximum velocity within the local costmap. However, if the velocity were reconfigured to a higher value using the described costmap size and simulation duration, the robot would not attain the desired V_{max}. Therefore, to address this issue, either the size of the costmap should be increased or the value of t_{sim} should be decreased. It should be noted that before determining the V_{max} of the robot, equation 6 must meet the desired velocity requirements. After ensuring that the V_{max} can be achieved within the local costmap another parameter needs to be configured. The forward prune distance d_{prune} parameter is responsible for pruning a predefined distance from the global path and utilizing it in calculating the generated trajectories. The following piece-wise function illustrates more on how the generated velocity v_{gen} is selected:

$$V_{\text{gen}} = \begin{cases} \frac{d_{\text{prune}}}{t_{\text{sim}}} & \text{if } \frac{d_{\text{prune}}}{t_{\text{sim}}} < V_{\text{max}}, \\ V_{\text{max}} & \text{otherwise.} \end{cases} \tag{7}$$

The *forward_prune_distance* parameter controls the speed of the robot based on the configured distance. If d_{prune} divided by t_{sim} is below the desired V_{max}, the robot speed moves with the calculated speed obtained from the d_{prune}. Otherwise, the robot moves with the desired V_{max}. A demonstration to show the effect of changing the d_{prune} value can be shown in Figure14, using a maximum velocity of $V_{\text{max}} = 0.5\,\text{m}\,\text{s}^{-1}$ and a simulation time of $t_{\text{sim}} = 3\,\text{s}$ as an example. Given that the robot is moving forward.

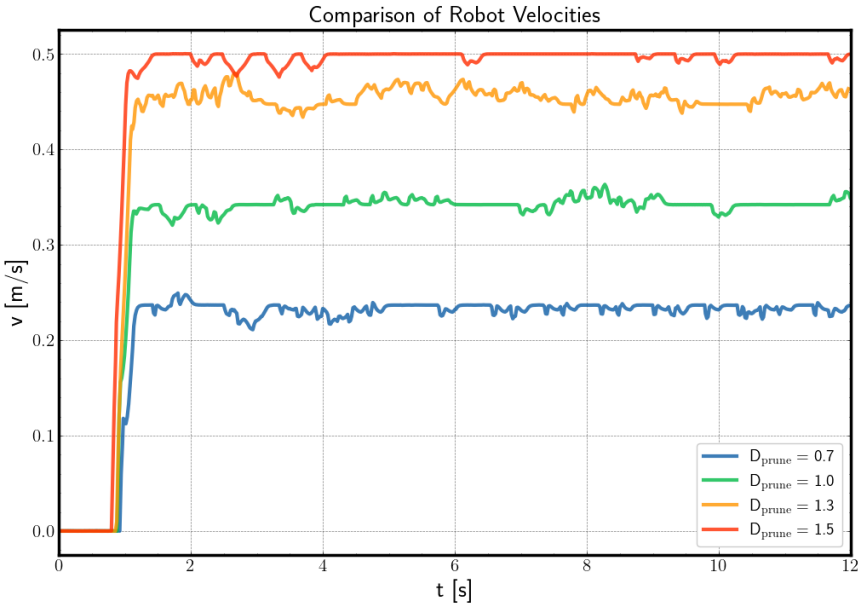

Figure 14: *Effect of changing the prune distance parameter*

Moreover, several experiments were conducted to see how the robot responds to dynamic obstacles. The simulation time when set to 1.5 second produced the best results for re-planning pathways closer to the global path when dealing with obstacles and allowing the robot to travel faster past the obstacle with a safer distance. Using higher t_{sim} resulted in longer paths in which the robot first avoids the obstacle but eventually moves closer to it. This proximity to the obstacle does not provide the robot enough time to stick to the global path, perhaps resulting in collisions. Nevertheless, the prune distance is set to $d_{\text{prune}} = 0.9\,\text{m}$ to achieve a V_{max} of $0.5\,\text{m}\,\text{s}^{-1}$ through satisfying equation 7.

3.1.2 DWB: Critic Functions

Six main critic functions that scores the generated trajectories for the algorithm to select the most optimal one based on the one with the lowest cost. The following critics are used:

- **PathDist Critic** C_{PD}: This critic measures how far each cell in the local costmap is from the nearest point on the global path. It then assigns a score to each cell based on that distance. The further away from the path, the higher the score for the cell.
- **PathAlign Critic** C_{PA}: This critic evaluates each generated trajectory by setting a forward point at a predetermined distance along it. From this point, the critic calculates the Euclidean distance to the closest point on the global path. The trajectory whose forward point is closest to the global path receives the lowest score, indicating it is the most aligned with the intended path.
- **GoalDist Critic** C_{GD}: Plays an important role in evaluating trajectories by computing the cost associated with the distance between each cell in the local costmap and the last valid location on the global plan that is inside the local costmap.
- **GoalAlign Critic** C_{GA}: Considers how the robot's orientation will finish up to the goal, choosing paths that properly align the robot for the final approach and goal heading.
- **BaseObstacle Critic** C_{BO}: Ensures that the trajectories considered for navigation are free from obstacles, reducing the risk of collisions.
- **Oscillation Critic** C_{O}: This critic gets triggered in dynamic situations or scenarios where the robot's motions may be uncertain, alternating between moving forward and backward or turning left and right without making significant progress. Thus, eliminating trajectories that cause the same performance.

Each critic function has a parameter that serves as a scale β representing the planner's dependence on that specific critic feature. Selecting the appropriate combination of critics based on specific requirements is crucial. For example, if the robot operates in an environment without dynamic obstacles, C_{PA} is unnecessary to be highly weighted. Instead, the *PathDist* can be used to prevent over-constraining the robot to strictly follow the global path's heading. The main goal is to choose a suitable combination of critics that can effectively address all potential scenarios simultaneously.

In some situations, moving the robot backward can lead to a high risk of collision due to limited LiDAR sensor coverage or sensor placement that creates blind spots, preventing the detection of approaching objects and causing collisions. Therefore, tuning and selecting the critics must be very precise to ensure the robot rotates to prefer the forward movement whenever possible. The DWB planner does not consider the robot's heading; for example, if the goal is behind the robot, it may move backward to reach the goal if all the critic's objectives, such as following the path, reaching the goal with the best possible trajectory, or being aligned with the global path, are satisfied. This behavior, however, can be a high risk of collision if the mentioned constraints play a role. Therefore, the first critic that is utilized is the *GoalAlign* critic, this critic selects the trajectories that can allow the robot to face the final heading of the goal.

Setting this critic the highest scale is essential to always consider trajectories that can align the robot's heading with the goal. Firstly, the forward point distance parameter d_{point} is used. This option provides a point in front of the robot with a certain distance where it will look forward to calculate the angle change in the present heading to align with the goal. A distance of 0.1 m enables the robot to be more sensitive to angular deviations from the goal direction. As a result, the robot detects even small early misalignment and performs rotations to correct its heading toward the goal. However, larger values showed that the robot becomes less sensitive to small angular deviations. This means, the robot may consider its current trajectory to be satisfactory and thus delay making sharp rotations or keep moving backward. Therefore, a value of 0.1 m is chosen.

Additionally, the C_{GA} is assigned a β of 35.0, the highest among all the critics, to ensure it is heavily relied upon, even when dealing with dynamic obstacles. This is important because the robot might move backward to avoid an obstacle and continue moving backward thinking that the current motion is aligning on the global path. With C_{GA} taking priority, the robot will focus on aligning with the goal, ensuring it adjusts its heading correctly.

However, depending on the C_{GA} only is insufficient. This critic adjusts the robot based only on the heading of the final point on the global path within the local costmap. When the planner selects the optimal trajectory for the robot to reach the goal, the C_{GA} may give a high score for a trajectory that drives the robot within the inflation radius of an obstacle, aligning it best with the goal. However, this approach can increase the risk of collision. Therefore, the *Path critics* should be used to align the current position of the robot with the global path. Both critics are heavily tuned for optimal performance, each assigned a scale of 30.0. However, these weights should be always configured to be lower than C_{GA}. This is because the robot might consider following the path in the reverse direction, as the DWB planner

generates trajectories from its rear when it starts moving backward, aligning it more closely with the path. Hence, assigning a higher weight to the C_{GA} is essential in this scenario.

After configuring most of the critics, the robot can avoid both dynamic and static obstacles, though it maintains only a limited proximity distance from them. For example, if a dynamic obstacle is directly in front of the robot, it will directly turn either left or right, and then proceed. However, this maneuver occurs right in front of the obstacle, indicating a need for a safer distance to consider a sharper rotation moving the robot away from the obstacle. To address this, the C_{BO} is utilized to penalize trajectories that would result in a collision.

When the robot detects a sudden obstacle, it generates and evaluates trajectories in real-time. Without the usage of the *BaseObstacle* critic, the robot might consider trajectories that, although not in direct collision, but still close to obstacles. In such instances, even if all trajectories are in a "*semi-collision*" state, the C_{BO} helps by penalizing these paths. This encourages the robot to consider backward motion, allowing it to create an extra proximity distance from the obstacle to perform a safer rotation. Additionally, the algorithm will start penalizing more trajectories that are closer to the obstacle, even if the robot did not consider moving backward at times to create extra distance from the obstacle before performing the rotation. As a result, the algorithm can rely on backward trajectories to escape the tight distance between the robot and the obstacle. Once the robot moves out of the collision risk zone, the other critics carry on their roles, guiding the robot to select once again the optimal trajectory and eliminating the need for continued backward motion.

However, assigning a high weight to the C_{BO} can limit the ability of the robot to move in narrow spaces, as it might consider all possible paths as unsafe or invalid if they appear in a collision. Choosing a β greater than 0.1 would cause the robot to get stuck in confined spaces as shown in Figure15. Where cells shown in red discourage the robot from moving inside. Choosing a lower value allows the robot greater flexibility to explore trajectories in confined spaces, as illustrated in Figure16. In both Figures, the costs associated with each cell are shown, illustrating the robot's decision-making process. These figures demonstrate how cell costs are allocated to observe the effect of changing the critic weights.

Additionally, in certain scenarios, the robot may overly penalize trajectories when navigating between closely positioned objects, causing it to slow down as it waits for the algorithm to find a viable path. This often leads to delays in the navigation process. To mitigate this issue, the *GoalDist* critic is utilized. This critic prioritizes trajectories that allow the robot to reach its destination as quickly as possible. However, there are instances where the influence of the C_{GD} can overshadow the C_{BO}, which penalizes trajectories near obstacles. By assigning $\beta = 5.0$ to the C_{GD},

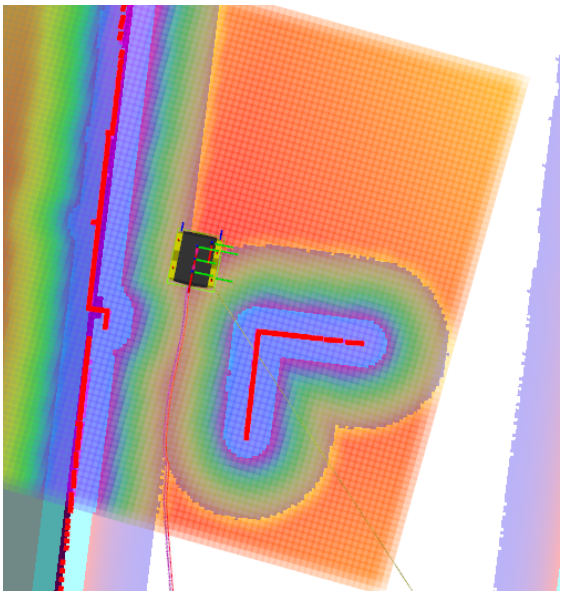

Figure 15: *Effect of setting $\beta = 1.0$ for the BaseObstacle*

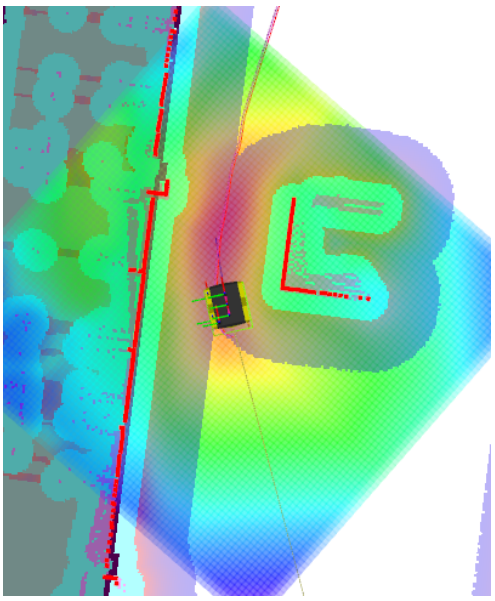

Figure 16: *Effect of setting $\beta = 0.1$ for the BaseObstacle*

a balanced approach is achieved. This ensures that the robot maintains a safe distance from obstacles during maneuvering, thereby reducing delays in choosing the optimal path through confined spaces between obstacles.

Moreover, numerous experiments revealed that in certain situations, the robot frequently faces a problem where the global planner repeatedly re-plans without making an absolute decision, especially when approaching an obstacle directly or attempting a sharp turn. This problem often leads to the robot oscillating between moving to the left or right. To address this, the *Oscillation* critic is utilized. This critic records the robot's last known position and observes both current and future actions. If the robot alternates directions a specific number of times, the critic activates, causing the robot to make an immediate decision and choose the optimal trajectory. Setting a β above 30.0 did not have a significant impact, so this value is used.

To sum up, DWB has been finely tuned to enhance the robot's navigation and decision-making capabilities across various scenarios. By optimizing parameters like velocity sampling and critic function weights, the robot achieves a balance between safety and efficiency. The following Tables represent the tuned parameters: