nMPC-Local-Planner with Integrated Dynamic Obstacle Avoidance

Student: Eppacher Kevin, BSc. **PK:** 2310331013
Peer ReviewerIn: Schweiger Simon, MSc.

***Abstract*—Mobile robots are widely used in industries, households, and public spaces, such as Autonomous Mobile Robots (AMRs) for transporting goods in intralogistics, vacuum/mower robots, and service robots in restaurants. However, mobile robots face several challenges, including highly dynamic environments, external disturbances such as slippage, which can complicate localization over long distances, and other environmental factors. The goal of this work is to develop a system that enables a mobile robot to navigate optimally and collision-free. Several established approaches, such as the Dynamic Window Approach (DWA) and Move Base Flex, have already proven effective. The Dynamic Window Approach limits the robot's velocity space to ensure collision avoidance, while Move Base Flex provides a flexible interface for path planning and execution, allowing for the integration of different navigation frameworks. In this work, a control system is developed for dynamic obstacle avoidance using a nonlinear Model Predictive Controller (nMPC). The results are compared with the well-established Move Base DWA from ROS1. Both controllers are evaluated in a Gazebo simulation under three conditions: with static objects, with dynamically detected obstacles, and without obstacle avoidance (i.e., without laser scan).**

(Results will follow...)

***Keywords*—Model Predictive Control, Differential Drive Robots, Obstacle Avoidance, Autonomous Mobile Robots, Control Algorithms**

I. Introduction

Mobile robots are becoming increasingly indispensable in various fields such as intralogistics, public, and private services, with applications ranging from autonomous floor-cleaning robots to vacuum cleaners. In dynamic environments, these robots are frequently confronted with fast-moving obstacles, which they must detect and avoid in real-time. The conventional approach to navigation involves the mobile robot first creating a map of its surroundings, typically using an occupancy grid to store information about static objects such as walls. Based on this map, a global path is planned using algorithms such as Rapidly Exploring Random Tree (RRT), which the robot subsequently follows [1].

However, it is important to note that while the global path provides an initial route, it is updated at a lower frequency compared to the local planner, which operates at a higher update rate. The local planner is responsible for adjusting the robot's trajectory in real-time to avoid newly detected obstacles, while still adhering to the waypoints of the global path [2]. This separation between global path planning and local obstacle avoidance is critical in ensuring that the robot can navigate effectively in both known and dynamic environments [3].

[4] conducted a comprehensive comparison of its developed local planner, ASAP, with other seminal planners, including DWA, TEB, and MPC. In Figure 1, the performance of these planners is compared as they guide a mobile robot to follow a global path while avoiding obstacles.

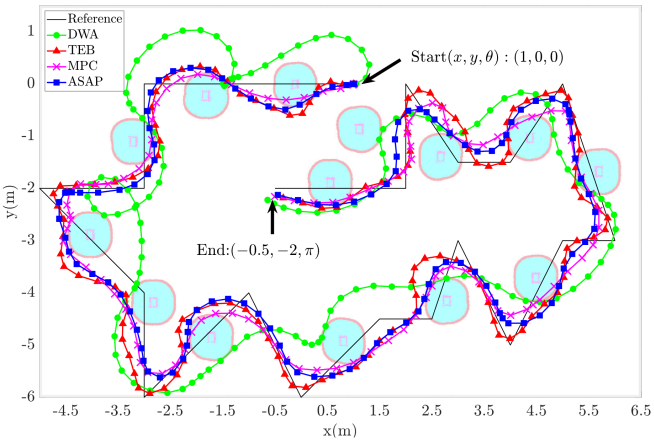

Fig. 1: Blockdiagram of nMPC for future Implementation

In this study, [4] evaluated the local planners based on metrics such as arrival time, cross track error, and computation time.

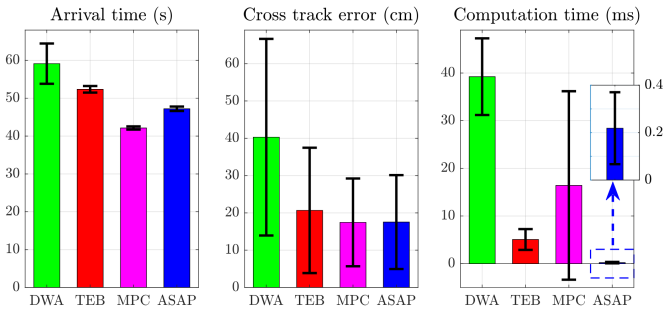

Fig. 2: Blockdiagram of nMPC for future Implementation

The findings in Figure 2 indicate that the MPC local planner achieved the shortest arrival time, while maintaining the same cross track error as the ASAP planner. However, when it comes to computation time, the ASAP controller was the fastest, with a processing time of just 0.2 ms, compared to DWA's 40 ms, TEB's 5 ms, and MPC's 18 ms. Furthermore, [4] highlights that the DWA planner had the highest collision rate, which also corresponds to its longer arrival time.

Similarly, [5] performed a comparison between MPC and TEB planners. The study evaluated both the TEB planner and various forms of MPC in terms of arrival time, path length, control effort, and CPU time. The results indicated that the TEB planner and the quadratic form of MPC had comparable arrival times, around 126.9 seconds. However, the time-optimal MPC, which is specifically optimized for minimizing arrival time, completed the task in just 116 seconds. In terms of path length and control effort, both planners showed similar performance. However, the computational time for MPC was found to be three times higher than that of the TEB planner, demonstrating the trade-off between efficiency and computational cost.

[4] and [5] illustrate promising results for the MPC as local planner, especially due to the high adaptability forming the cost function for the required needs and the model adaptability, as for example omni-drive-directional-wheels. However, it is shown, that the MPC planner in combination with dynamic obstacle avoidance lack in computation time.

The main objective is to design a controller that effectively addresses the challenges of dynamic obstacle avoidance, offering improved performance in real-time navigation within dynamic environments. This work contributes to the state-of-the-art development, of a nonlinear Model Predictive Controller for a mobile robot with integrated dynamic obstacle avoidance. This work proposed a nMPC local Planner, in which the nMPC is integrated with a obstacle avoidance algorithm, with the aim of reducing computation time and increasing the performance, which is evaluated by reducing the arrival time and the cross tracking error.

The rest of the paper is organized as follows: Section II discusses related work in local path planning and obstacle avoidance. Section III outlines the methodology used. Section IV describes how the methods were implemented and evaluated. In Chapter IV, the results of the experiments are presented. In Chapter V, the results are discussed and critically reflected upon. Chapter VI summarizes this work and gives an outlook on future research.

II. State-of-the-Art

In this section, we provide an overview of several state-of-the-art local planners, including the Dynamic Window Approach (DWA), Timed Elastic Band (TEB), Model Predictive Control (MPC), Move Base Flex (MBF), Agile and Safe Pursuit (ASAP), and Reinforcement Learning (RL)-based planners. We explain how these planners function and highlight their key limitations, especially when operating in dynamic environments. Additionally, we discuss how nonlinear Model Predictive Control (nMPC) addresses these limitations and provides improved performance.

A. Dynamic Window Approach (DWA)

DWA is a popular local planner that selects safe and feasible velocities by sampling translational (v) and rotational (ω) velocities within a dynamic window. These sampled velocities are evaluated through an objective function that balances forward movement and obstacle avoidance. DWA computes a set of trajectories and selects the one that maximizes progress toward the goal while ensuring the robot avoids collisions [6].

The velocities v and ω are constrained within bounds, as shown in Equation (1):

$$v_{min} \leq v \leq v_{max}, \quad \omega_{min} \leq \omega \leq \omega_{max} \tag{1}$$

The objective function $G(v, \omega)$ evaluates the alignment with the goal, distance to obstacles, and forward velocity, as described in Equation (2):

$$G(v, \omega) = \alpha \cdot \text{heading}(v, \omega) + \beta \cdot \text{clearance}(v, \omega) + \gamma \cdot v \tag{2}$$

While DWA is efficient and handles real-time obstacle avoidance well, it struggles with dynamic obstacles due to its lack of future obstacle movement prediction. The extensive velocity sampling can lead to increased computation times in complex environments [6].

Recent enhancements to DWA focus on improving computational efficiency and addressing the limitations in dynamic environments. A faster implementation of DWA based on non-discrete path representation [7] reduces the computational complexity by eliminating discrete path representations and employing polar coordinate transformations. Another improved version integrates reinforcement learning (RL) techniques [8], dynamically adjusting the sampling window size to better handle obstacles in real-time, which improves efficiency and accuracy in dynamic environments.

B. Timed Elastic Band (TEB)

TEB models the robot's trajectory as a dynamic band that adjusts to avoid obstacles while satisfying kinodynamic constraints. The trajectory is continuously optimized using a cost function that takes into account the goal, collisions, and system dynamics [9].

The cost function $J(\mathbf{x}, \mathbf{v})$ over the series of poses $\mathbf{x}$ and velocities $\mathbf{v}$ is defined in Equation (3):

$$J(\mathbf{x}, \mathbf{v}) = \sum_{i=1}^{n-1} \left(\text{cost}_{\text{goal}}(x_i) + \text{cost}_{\text{collision}}(x_i) + \text{cost}_{\text{kinodynamics}}(v_i) \right) \tag{3}$$

TEB is effective at generating smooth trajectories that adapt to obstacles and kinodynamic constraints. However, in highly dynamic environments, TEB's computational complexity can increase significantly. Additionally, its performance is sensitive to initial conditions and parameter tuning.

Recent improvements to TEB incorporate dynamic obstacle velocities using Kalman filtering to predict obstacle movement and optimize the trajectory in real time. This improved TEB method [10] enhances real-time performance by reducing the overall computational burden while maintaining a high success rate for dynamic obstacle avoidance.

C. Model Predictive Control (MPC)

MPC computes optimal control inputs over a prediction horizon by solving an optimization problem that accounts for both system dynamics and environmental constraints [11]. Given the system model $x_{k+1} = f(x_k, u_k)$, MPC minimizes the cost function defined in Equation (4):

$$\min_{\vec{u}_k} J(\vec{x}_k, \vec{u}_k) = \phi(\vec{x}(N)) + \sum_{k=0}^{N-1} L(\vec{x}(k), \vec{u}(k)) \quad (4)$$

MPC offers smooth, optimized control over nonlinear dynamics, making it ideal for complex environments. However, it is computationally demanding, especially in real-time applications. Additionally, MPC requires highly accurate models of the robot and environment, which may not always be feasible in dynamic situations [12].

Compared to DWA and TEB, MPC offers improved handling of dynamic obstacles by continuously adjusting the robot's trajectory in real-time, optimizing for both safety and performance. It also handles nonlinear dynamics more effectively, making it a robust solution for highly dynamic environments.

III. Methods

This section describes the methodology applied in this work, focusing on the integration of the trajectory planner, nMPC, and the Obstacle Detection Algorithm, culminating in the nMPC local planner. Figure 3 illustrates the individual components of the nMPC local planner. The main elements are the trajectory planner, nMPC, and optionally, the obstacle detection algorithm, which will be explained in more detail later.

The trajectory planner receives the global path from MoveBase. It processes the global plan into a local plan, taking into account the number of predictions and the prediction horizon length specified by the nMPC. After the nMPC receives the reference trajectory, the robot's current pose, and the target state, the optimizer of the nMPC calculates the optimal control input and sends it back to the trajectory planner. Depending on whether the differential drive mobile robot (DDMR) has reached its target or not, the trajectory planner then sends the optimal control input $\mathbf{u} = \begin{bmatrix} v & \omega \end{bmatrix}^\top$

The robot's absolute position is determined using AMCL, which helps eliminate cumulative slip and drift errors. However, a map must be created in advance [1]. Based on the scan data from the LiDAR, the DDMR can detect obstacles. The Obstacle Detection Node filters the LiDAR data to define a search cone, reducing unnecessary computational load on the nMPC. It then sends the positions of the obstacles, along with the predefined safety radius of the obstacles, to the trajectory planner, which passes this information to the nMPC for further processing.

A. nMPC

The nMPC is responsible for planning a path based on the current pose, the target pose, and the reference trajectory, using an optimizer that not only satisfies the kinematics of a differential drive mobile robot (DDMR) but also avoids obstacles and respects limitations on control inputs and states. Figure 4 illustrates the components of an MPC [13].

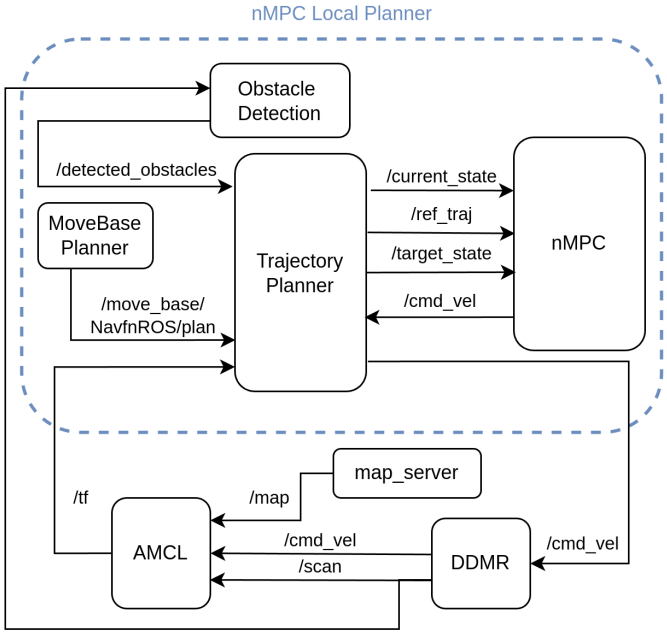

Fig. 3: nMPC-Local-Planner Block Diagram

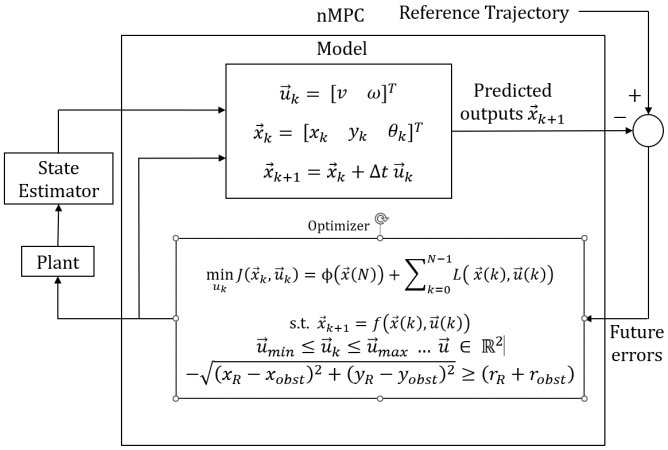

Fig. 4: Detailed Blockdiagram of nMPC [14]

First, the Optimal Control Problem (OCP) must be converted into a Nonlinear Programming Problem (NLP) using the multiple shooting discretization method. The goal is to minimize the cost function, as defined in Equation 5, which consists of the states and control actions for all predictions [15].

$$\min_{\vec{u}_k} J(\vec{x}_k, \vec{u}_k) = J_{\text{obs}} + \phi(\vec{x}(N)) + \sum_{k}^{k+N-1} (\vec{x}_{R,k} - \vec{x}_{\text{ref},k})^T Q (\vec{x}_{R,k} - \vec{x}_{\text{ref},k}) + \vec{u}_k^T R \vec{u}_k \quad (5)$$

The cost function J consists of three main components:

- $\phi(\vec{x}(N))$ represents the terminal cost, ensuring the final state $\vec{x}(N)$ is close to the target state, as defined in Equation 6 [15].

$$\phi(\vec{x}(N)) = \frac{1}{2}\vec{x}_N^T S \vec{x}_N \tag{6}$$

- J_{obs} is a penalty term that increases the cost if the robot comes close to an obstacle, calculated according to Equation 7 [16].

$$J_{\text{obs}} = \sum_{k=0}^{N} \sum_{i=1}^{n_{\text{obs}}} \frac{w_{\text{pen}}}{\sqrt{(x_k - x_{\text{obs},i})^2 + (y_k - y_{\text{obs},i})^2} + \epsilon} \tag{7}$$

where $x_{\text{obs},i}$ and $y_{\text{obs},i}$ represent the coordinates of the i-th obstacle, and ϵ is a small value to avoid singularities. The state vector $\mathbf{x}$ and control vector $\mathbf{u}$ are defined in Equations 8 and 9, respectively [17].

$$\mathbf{x}_k = \begin{bmatrix} x_k & y_k & \theta_k \end{bmatrix}^T \tag{8}$$

$$u_k = \begin{bmatrix} v_k & \omega_k \end{bmatrix}^T \tag{9}$$

These vectors describe the position, orientation, and control inputs of the robot at each time step k. The evolution of the state is constrained to follow the robot's kinematics, as enforced by Equation 10 [17].

$$\vec{x}(k+1) = f(\vec{x}(k), \vec{u}(k)) \tag{10}$$

$$\vec{x}(0) = \vec{x}_0 \tag{11}$$

For a differential drive mobile robot, the discrete state-space dynamics are modeled as shown in Equation 12 [1].

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} + \Delta T \begin{bmatrix} \cos(\theta_k) & 0 \\ \sin(\theta_k) & 0 \\ 0 & 1 \end{bmatrix} u_k \tag{12}$$

To ensure safe navigation, the control inputs are constrained between the bounds specified in Equation 13.

$$\vec{u}_{\min} \leq \vec{u}_k \leq \vec{u}_{\max}, \quad \forall k \tag{13}$$

In addition, a collision avoidance constraint, as shown in Equation 14, ensures that the robot maintains a safe distance from obstacles.

$$-\sqrt{(x_R - x_{\text{obst}})^2 + (y_R - y_{\text{obst}})^2} + (r_R + r_{\text{obst}}) \leq 0 \tag{14}$$

This inequality guarantees that the robot's distance from any obstacle is greater than or equal to the sum of their radii, preventing collisions [16].

The control strategy computes the optimal sequence of control inputs $\mathbf{u}_k$ by solving the NLP at each time step, taking into account the robot's current and predicted future states. The horizon N and the sampling time ΔT influence how far ahead the controller predicts and adjusts the control actions. By combining trajectory planning, obstacle avoidance, and constrained optimization, nMPC provides an integrated approach for navigating in dynamic environments.

B. Trajectory Planner

The trajectory planner processes the global path provided by the MoveBase and transforms it into a local path that adheres to the predictive horizon N of the nMPC and the predicted distance d_{pred}. By default, the global planner in ROS assigns the same orientation to all waypoints, which is suboptimal for precise trajectory tracking. To address this, an additional step has been implemented to adjust the orientation of each waypoint based on the direction to the next point along the path. This results in waypoints that are aligned with the trajectory, improving navigation accuracy.

Once the waypoints are oriented correctly, they are passed to the trajectory interpolation algorithm. This algorithm ensures that the number of waypoints matches the number of nMPC predictions. Based on the robot's maximum linear velocity v_{max}, the algorithm calculates the appropriate time stamps for the interpolated local path. This guarantees that the nMPC maintains the correct prediction length. Additionally, the time constant is multiplied by a scaling factor α to generate a feedforward effect, ensuring that the reference waypoints are always ahead of the robot's current state, guiding the robot to follow the trajectory effectively. This technique also allows for subtle manipulation of speed along the trajectory.

A plot illustrating this interpolation process is provided in Figure 5. In the example, three waypoints are given, and the algorithm linearly interpolates 10 points for 10 predictions.

Algorithm 1 Trajectory Interpolation

Require: $P = \{(x_i, y_i, \text{yaw}_i)\}$, d_{pred}, N, v_{max}, α
Ensure: P_{new}, T

1: Compute cumulative distance between points:

$$D_i = \sum \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

2: Retain points where $D_i \leq d_{\text{pred}}$
3: Compute interpolated distance per point:

$$d_{\text{point}} = \frac{d_{\text{pred}}}{N}$$

4: Compute time step based on maximum velocity:

$$T = \frac{d_{\text{point}}}{v_{\text{max}}}$$

5: Adjust time step with scaling factor:

$$T = T \cdot \alpha$$

6: Interpolate x, y, and yaw to get N prediction points
7: **return** P_{new}, T =0

This interpolation algorithm takes the reference global path and generates N evenly spaced points along the trajectory. The associated time step T ensures smooth and continuous control of the robot's motion, with feedforward control helping to keep the reference trajectory ahead of the robot's actual state. This approach is analogous to the method used in the TEB Local Planner [18].

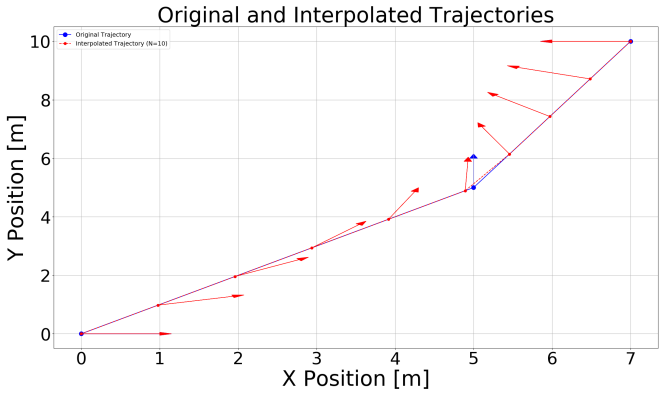

Fig. 5: Interpolation process: Given 3 initial waypoints, 10 linearly interpolated waypoints are generated for 10 nMPC predictions.

C. Obstacle Detection

Unlike DWA and TEB, the obstacle detection algorithm does not subscribe to the local costmap but instead directly processes laser scan data. The laser scan data, once received, are initially filtered based on a search cone, defined by the search radius and search angle, as illustraded in Algorithm 2. This filtering step helps to exclude obstacles that are behind the robot and no longer relevant. The nMPC is currently designed to operate only with positive linear velocities, meaning the robot is expected to move forward. If backward motion were required, the search angle would need to be adjusted by 180°.

After filtering, the scan data are transformed into 2D coordinates to create a LiDAR image. The LiDAR image is then smoothed using Gaussian blur, which helps reduce noise and makes obstacle detection more robust. The smoothing process is critical for ensuring the corner detection algorithm functions effectively, as it eliminates irregularities and helps detect more consistent object edges. The Shi-Tomasi corner detection algorithm is applied to identify obstacles. Initially intended for corner detection, the algorithm can also detect walls or flat surfaces depending on the quality level, likely due to the inherent noise in the sensor data.

Once corners are detected, they are sorted by their distance from the robot. The detected positions are then transformed from the /map frame to the /base_footprint frame. Finally, only the maximum number of obstacles that the nMPC can process are considered, with any additional obstacles being discarded.

The algorithm filters the laser scan data to focus on the area within the specified search radius and angle. The data are then converted into 2D coordinates and used to create a LiDAR image, which is smoothed with Gaussian blur to reduce noise and improve obstacle detection accuracy. The Shi-Tomasi corner detection algorithm identifies obstacles, which are then transformed into the robot's frame of reference and used to inform the nMPC. Only the closest obstacles, up to the maximum number that the nMPC can process, are considered. A visual representation of the obstacle detection algorithm is shown in Figure 6, illustrating how the laser scan data are filtered, converted, and processed to detect obstacles before passing the data to the nMPC.

Algorithm 2 Filtered Obstacle Detection

1: Filter laser scan data (r_i, θ_i) using:

$$r_i < r_{\text{search}} \quad \text{and} \quad \left(-\frac{\theta_{\text{search}}}{2} \leq \theta_i \leq \frac{\theta_{\text{search}}}{2}\right)$$

2: Convert filtered data to 2D coordinates:

$$x_i = r_i \cos(\theta_i), \quad y_i = r_i \sin(\theta_i)$$

3: Create a LiDAR image for obstacle detection:

$$x_{\text{img}} = \left(\frac{x_i}{r_{\text{max}}} \times I_{\text{size}}\right) + \frac{I_{\text{size}}}{2}, \quad y_{\text{img}} = \left(\frac{y_i}{r_{\text{max}}} \times I_{\text{size}}\right) + \frac{I_{\text{size}}}{2}$$

4: Apply Gaussian blur to smooth the image:

$$I_{\text{blur}}(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \otimes I_{\text{img}}(x, y)$$

5: Detect corners using the Shi-Tomasi corner detection algorithm:

$$\{(x_{\text{corner}}, y_{\text{corner}})\} = \text{Shi-Tomasi}(I_{\text{blur}})$$

6: Sort detected corners by distance from the robot:

$$d_i = \sqrt{x_{\text{corner}}^2 + y_{\text{corner}}^2}$$

7: Transform detected corners into the robot's frame of reference:

$$x_{\text{trans}} = x_{\text{robot}} + (x_{\text{corner}} \cos(\text{yaw}) - y_{\text{corner}} \sin(\text{yaw}))$$

$$y_{\text{trans}} = y_{\text{robot}} + (x_{\text{corner}} \sin(\text{yaw}) + y_{\text{corner}} \cos(\text{yaw}))$$

8: Select N_{max} closest points

9: **return** $(x_{\text{trans}}, y_{\text{trans}}, r_{\text{corner}})$ to nMPC =0

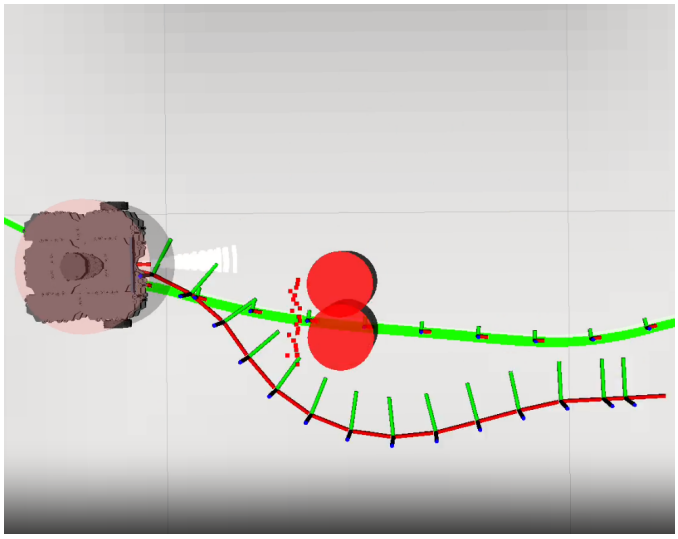

Fig. 6: Obstacle Detection Algorithm Visualization

The integration of the obstacle detection algorithm with

the trajectory interpolation ensures that the nMPC can safely navigate through the environment while considering both the robot's kinematics and the presence of dynamic obstacles. By sending only the most relevant obstacles to the nMPC, the computational load is minimized, allowing for real-time decision-making.

IV. Implementation Detail

A. CasADi

CasADi was chosen for this work due to its robust framework, which integrates with IPOPT to perform nonlinear optimizations. For the implementation of the nMPC in this project, it was necessary to predefine the optimization problem in CasADi. This involved defining all the required matrices for the state vector, control vector, optimization variables, and weighting matrices Q, R and S, as well as the cost function.

When the optimization problem is called, all necessary parameters are passed, including the current state, the reference trajectory, obstacles (represented as n-times (x, y, radius)), and the computed time constant T. The parameter settings for the optimization solver were configured as follows:

```
opts = {'ipopt.max_iter': 2000,
        'ipopt.print_level': 0,
        'print_time': 0,
        'ipopt.acceptable_tol': 1e-8,
        'ipopt.acceptable_obj_change_tol': 1e-6}
```

B. MoveBase Integration in ROS1

The system was implemented in ROS1 due to its existing Navigation Stack, which includes state-of-the-art local planners such as DWA and TEB. ROS1 provides a simple and efficient way to integrate, test, and validate the nMPC alongside these well-established planners. Additionally, ROS1 offers an open-source platform, making it possible for the community to further test and develop the system. The project is made available on GitHub for public use and contributions at the following link: https://github.com/KevinEppacher/walle_ws.git

C. Evaluation Metrics

The experiments were conducted using three key metrics:

- Cross Tracking Error: The deviation of the robot's path from the first reference trajectory.
- Arrival Time: The time taken for the robot to reach its target waypoint.
- Computation Time: The average time required for the local planner to compute the control inputs.

1) Experiment 1: Dynamic Obstacle Avoidance: In the first experiment, the robot is tasked with navigating to a waypoint on the map. During this task, a dynamic obstacle moves in a circular pattern within the Gazebo simulation. The obstacle is detected only by the LiDAR sensors and the local costmap, but is not accounted for in the global plan. The update frequency of the system is reduced to 1 Hz, making the local planner responsible for most of the planning decisions, instead of the global planner. The performance of the three local planners—nMPC, DWA, and TEB—was measured based on the predefined metrics, with particular attention given to observing whether any of the planners collided with the rotating obstacle.

2) Experiment 2: Narrow Passage with Static Obstacle: In the second experiment, the robot must pass through a narrow corridor. However, the passage is partially blocked by a static obstacle, which is not present in the pre-mapped environment. Consequently, the global planner generates a plan that passes through the "invisible" obstacle, requiring the local planner to detect and avoid it. The performance of the local planners was again evaluated using the same metrics.

3) Ablation Experiment: Disabled LiDAR Scans: In the ablation experiment, each local planner is required to navigate to a target point. The direct path is obstructed by a mapped object, and the LiDAR scan data are disabled to test the robustness of the planners without the aid of real-time obstacle detection. This scenario was designed to assess how well the planners perform when relying solely on pre-mapped data.

D. Hardware

The experiments were conducted on a system equipped with an AMD Ryzen 5 7600X 6-Core Processor running at a maximum clock speed of 5.45 GHz. The system has 12 logical CPUs (2 threads per core) and 30 GB of RAM, with no swap space configured.

The experiments ran inside a Docker container, which utilized up to 4.808 GB of RAM (approximately 15.78 % of the available memory) and had a peak CPU usage of 855.20 %, indicating multi-core processing. The container handled a maximum of 393 processes during the experiments.

V. Discussion and Results

Each experiment is first presented with an XY plot showing the paths generated by the three local planners (nMPC, DWA, and TEB) alongside the initial global path. Following this, a bar chart is provided that compares the three metrics—cross tracking error, computation time, and arrival time—across all three planners. This structure is consistently applied across all three experiments.

A. Experiment 1: Dynamic Obstacle Avoidance

In the first experiment in Figure 7, the TEB planner encountered significant challenges with the rotating obstacle. The TEB planner entered a deadlock situation, where the elastic band surrounding the robot's path became entangled with the rotating obstacle. This prevented the planner from finding a valid path, resulting in a collision with the obstacle. Once the planner switched to recovery mode, causing the robot to rotate in place and clear the local costmap, it was able to find a new path and reach the target. However, this process led to an extended arrival time of over 90 seconds, which can be seen in Figure 8, and the cross tracking error with respect to the global path was also considerable due to the repeated rotations caused by the rotating obstacle.

In contrast, both the nMPC and DWA planners successfully reached their targets in 40 and 39 seconds, respectively. The

DWA planner achieved a slightly better cross tracking error than the nMPC (0.01 m difference), though the nMPC planner outperformed the DWA planner by being 2.3 times faster in computation time. While this difference is not captured in the plot, it was observed in the simulation that the DWA planner detected the obstacle only when it was directly in front of the robot, whereas the nMPC was able to predict and avoid the obstacle up to 2 meters in advance.

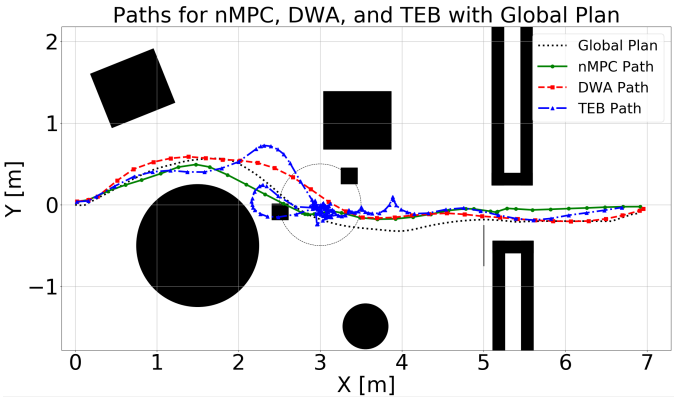

Fig. 7: Comparison of paths for nMPC, DWA, and TEB in Experiment 1

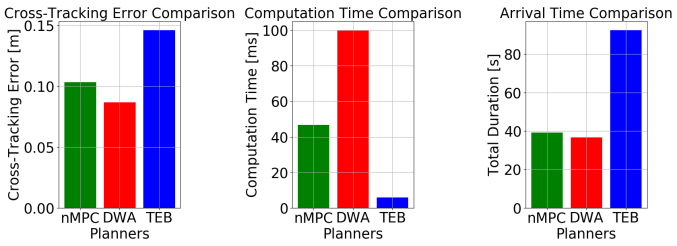

Fig. 8: Metrics comparison (cross tracking error, computation time, arrival time) for nMPC, DWA, and TEB in Experiment 1

B. Experiment 2: Narrow Passage with Static Obstacle

In contrast to Experiment 1, the TEB planner was the fastest to reach the target in Experiment 2, completing the task in 15 seconds, followed closely by the nMPC planner at 18 seconds, which can be seen in Figure 10. The DWA planner, however, took significantly longer, with an arrival time of approximately 35 seconds. The delay in the DWA planner was attributed to its failure to account for the static obstacle blocking the passage. The global planner generated a path through the obstacle, and the DWA local planner was unable to compute a correction in time, resulting in repeated collisions with the wall.

Although the TEB planner reached the goal fastest, the nMPC planner achieved the lowest cross tracking error, demonstrating more precise path adherence. The TEB planner continued to show a consistent computation time of 5-10 ms as seen in Figure 12, 8 and 10.

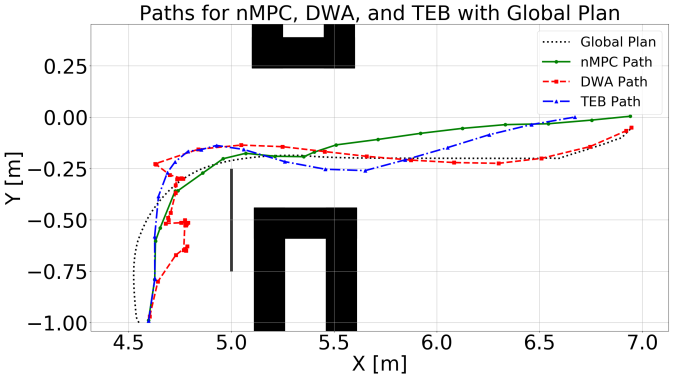

Fig. 9: Comparison of paths for nMPC, DWA, and TEB in Experiment 2

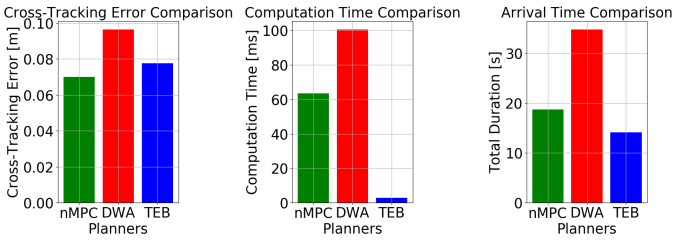

Fig. 10: Metrics comparison (cross tracking error, computation time, arrival time) for nMPC, DWA, and TEB in Experiment 2

C. Ablation Experiment: Lidar Scan Disabled

In the final experiment, the nMPC planner failed to reach its target, colliding with an object due to the absence of scan data. A key limitation of the current nMPC implementation is that when the final goal falls within the prediction horizon, the controller switches from reference tracking to point stabilization to optimize performance. This behavior needs to be addressed to ensure the nMPC follows the entire trajectory, rather than stopping prematurely at the prediction horizon.

Despite reaching its goal, the TEB planner also collided with the object during this experiment. The DWA planner, in contrast, exhibited the best performance in terms of arrival time and cross tracking error when lidar scan data were disabled, demonstrating its robustness in situations where real-time obstacle detection is not available.

VI. Summary and Outlook

References

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005.

[2] R. Siegwart, I. Nourbakhsh, and D. Scaramuzza, *Autonomous Mobile Robots*, 2nd ed. Cambridge, MA: MIT Press, 2011.

[3] O. Khatib, "Real-time obstacle avoidance for fast mobile robots," *International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[4] D.-H. Lee, S. Choi, and K.-I. Na, "Asap: Agile and safe pursuit for local planning of autonomous mobile robots," *IEEE Access*, vol. PP, pp. 1–1, 01 2024.

[5] C. Rösmann, A. Makarow, and T. Bertram, "Online motion planning based on nonlinear model predictive control with

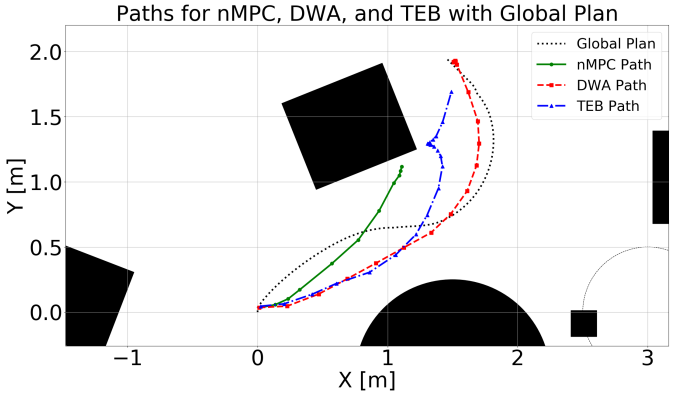

Fig. 11: Comparison of paths for nMPC, DWA, and TEB in Ablation Experiment

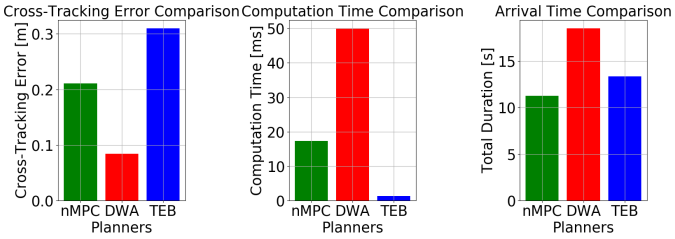

Fig. 12: Metrics comparison (cross tracking error, computation time, arrival time) for nMPC, DWA, and TEB in Ablation Experiment

non-euclidean rotation groups," *CoRR*, vol. abs/2006.03534, 2020. [Online]. Available: https://arxiv.org/abs/2006.03534

[6] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *Robotics Automation Magazine, IEEE*, vol. 4, pp. 23 – 33, 04 1997.

[7] Z. Lin and R. Taguchi, "Faster implementation of the dynamic window approach based on non-discrete path representation," *Mathematics*, vol. 11, no. 21, p. 4424, 2023.

[8] Y. Zhou, C. Zeng, and S. Zhu, "Improved dynamic window approach for unmanned surface vehicles' local path planning considering the impact of environmental factors," *Sensors*, vol. 22, no. 14, p. 5181, 2022.

[9] C. Rösmann, W. Feiten, T. Woesch, F. Hoffmann, and T. Bertram, "Trajectory modification considering dynamic constraints of autonomous robots," 01 2012, pp. 1–6.

[10] X. Ou, Z. You, and X. He, "Local path planner for mobile robot considering future positions of obstacles," *Processes*, vol. 12, no. 5, 2024. [Online]. Available: https://www.mdpi.com/2227-9717/12/5/984

[11] C. Rösmann, A. Makarow, and T. Bertram, "Online motion planning based on nonlinear model predictive control with non-euclidean rotation groups," *CoRR*, vol. abs/2006.03534, 2020. [Online]. Available: https://arxiv.org/abs/2006.03534

[12] A. Rezaee, "Model predictive controller for mobile robot," *Transactions on Environment and Electrical Engineering*, vol. 2, p. 17, 06 2017.

[13] P. Falcone, F. Borrelli, J. Asgari, H. P. Tseng, and D. Hrovat, "Predictive control for autonomous vehicle systems: Dynamic motion planning and execution," *Proceedings of the IEEE*, vol. 95, no. 2, pp. 464–490, 2007.

[14] F. Yakub and Y. Mori, "Effects of roll dynamics for car stability control by laguerre functions," 08 2013, pp. 330–335.

[15] J. B. Rawlings and D. Q. Mayne, "Model predictive control: Theory and design," *Nob Hill Pub.*, 2009.

[16] J. V. Frasch, M. Diehl, A. Geiger, and M. Steinbach, "Parallel algorithms for real-time motion planning based on nonlinear model predictive control," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 3, pp. 1497–1507, 2013.

[17] A. Liniger and J. Lygeros, "Nonlinear model predictive control for autonomous driving with an economic objective," *2015 European Control Conference (ECC)*, pp. 502–507, 2015.

[18] C. Rösmann, F. Hoffmann, and T. Bertram, "Integrated online trajectory planning and optimization in distinctive topologies," *Robotics and Autonomous Systems*, vol. 88, pp. 142–153, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889016300495