Nonlinear Model Predictive Control-Based Local Planner with Integrated Dynamic Obstacle Avoidance for Mobile Robots

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

***Abstract*—Mobile robots play a crucial role in various domains, including logistics and service industries, where they must navigate dynamic environments filled with fast-moving obstacles. Traditional hierarchical navigation systems separate global path planning and local obstacle avoidance, with global planners focusing on static maps and local planners using real-time sensor data for immediate adjustments. However, local planners such as Dynamic Window Approach (DWA), Timed Elastic Band (TEB), and Model Predictive Control (MPC) struggle in highly dynamic environments due to issues such as slow responsiveness, difficulty in handling sharp turns or dense obstacles, and high computational demands, which can result in suboptimal paths or failure to avoid collisions in real time. This paper proposes a Nonlinear Model Predictive Controller (nMPC) integrated with an obstacle detection algorithm to improve dynamic obstacle avoidance performance. The nMPC framework selectively extracts only the most relevant obstacles for avoidance, reducing the computational load while maintaining safety and efficiency. This optimization minimizes computation time, reduces cross-track errors, and ensures effective navigation. Through comprehensive simulations, the nMPC demonstrated superior performance compared to state-of-the-art planners in dynamic obstacle avoidance tasks, achieving faster arrival times and lower tracking errors. Future work will extend the application of this controller to real-world systems, enhancing the nMPC's adaptability to various robotic platforms. The project is made available on GitHub for public use and contributions at the following link: https://github.com/KevinEppacher/walle_ws.git**

***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.

Navigation in mobile robotics is generally approached using a hierarchical paradigm, where a global planner generates a high-level path and a local planner refines this path based on real-time sensory data. The global planner operates over a static map, which is typically generated using techniques like occupancy grids, and focuses on finding an optimal or feasible path from the start to the goal location. This path is often generated by algorithms such as Dijkstra, A*, or more advanced techniques like Rapidly Exploring Random Trees (RRT) [1].

However, the global path alone is insufficient in dynamic environments where obstacles may move or change over time. To handle these situations, the local planner operates at a higher frequency, continuously refining the robot's trajectory to avoid newly detected obstacles, while ensuring the robot still follows the general direction of the global path. The local planner relies on real-time sensory input (e.g., Light Detection and Ranging (LiDAR) or camera data) to make instantaneous adjustments, balancing both collision avoidance and adherence to the global plan [2], [3].

This separation between global path planning, which is concerned with long-term planning over a static map, and local obstacle avoidance, which focuses on immediate reactive control, is key to ensuring that the robot can navigate effectively in both known and dynamic environments. By maintaining this hierarchical structure, robots are able to achieve long-distance navigation goals while remaining responsive to dynamic obstacles in real-time [2], [3].

[4] conducted a comprehensive comparison of its developed local planner, ASAP, with other seminal planners, including DWA, TEB, and MPC. In [4], we compare the performance of these planners in guiding a mobile robot to follow a global path while avoiding densely placed obstacles. In this study, [4] evaluated the local planners based on metrics such as arrival time, cross track error, and computation time.

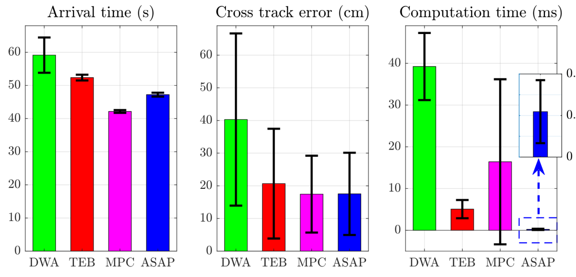

Fig. 1: Simulation results for arrival time, cross track error and computation time for DWA, TEB, MPC and ASAP (directly taken from [4])

The findings in Figure 1 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 Central Processing Unit (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.

Based on the findings in [4], the weaknesses of the local planners DWA, TEB and MPC are highlighted, particularly in the context of dynamic obstacle avoidance.

DWA faces significant challenges when obstacles are directly in front of the robot, especially in environments with narrow passages or sharp turns. This is due to the fact that DWA samples a wide range of speeds to find a feasible collision-free trajectory, but this method increases computation time and is prone to failure in tight spaces. In addition, in environments with dynamic obstacles, velocity sampling often cannot react quickly enough, making DWA less reliable in real-time applications [4].

TEB, known for generating smooth and collision-free paths, works by solving a non-convex optimisation problem. While TEB performs well in static or moderately dynamic environments, its performance degrades in highly dynamic environments due to the need for frequent re-optimisation. This can lead to suboptimal paths or deadlocks where the planner cannot find a valid solution, especially when obstacles move rapidly or unpredictably. In addition, TEB's computation time increases significantly in dense environments, further limiting its real-time applicability [4], [5].

MPC, on the other hand, provides near-optimal trajectories by predicting future robot states and optimising control inputs based on system dynamics and constraints. However, solving these constrained optimisation problems at each time step imposes a high computational burden, especially when a longer prediction horizon is required to handle complex or dynamic environments. The real-time performance of MPC is highly dependent on the available computational resources and the complexity of the environment. Furthermore, the effectiveness of MPC is highly dependent on an accurate model of the robot and the environment, which can be difficult to achieve in real-world scenarios with unpredictable obstacles [4], [5].

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 sparse obstacle avoidance. This work proposed an nMPC local planner, in which the nMPC is integrated with an obstacle avoidance algorithm, to reduce computation time and increase 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

This section provides an overview of several state-of-the-art local controllers, including the DWA, TEB and MPC. In contrast, nMPC is presented to overcome these limitations and provide improved performance.

A. Preliminary Information on Optimization-Based Methods

Optimization-based methods like DWA, TEB, and MPC solve an Optimal Control Problem (Optimal Control Problem (OCP)) by minimizing a cost function while respecting system constraints. The cost function is an objective measure that typically balances factors such as trajectory tracking, control effort, and obstacle avoidance. Its goal is to find control inputs that minimize deviation from the desired path while maintaining safety and efficiency.

In OCPs, equality constraints ensure the system dynamics are followed, while inequality constraints handle limits such as collision avoidance and control input bounds. The problem is often discretized using methods like multiple shooting or single shooting, which transform the continuous problem into a finite-dimensional optimization problem.

These optimization problems are solved in real-time using solvers such as Quadratic Programming (QP), Interior Point Optimizer (IPOPT), or Sequential Quadratic Programming (SQP), which can handle both the constraints and optimization efficiently over the prediction horizon.

B. Dynamic Window Approach

DWA is a popular local planner that selects safe and feasible velocities by sampling translational (v) and rotational (ω) velocities within a dynamic window, reinitialising the current pose and velocities and the current obstacle position. 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], [7].

The velocities v and ω are constrained within bounds (see Eq. 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 (see Eq. 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 [8] reduces the computational complexity by eliminating discrete path representations and employing polar coordinate transformations. Another improved version integrates reinforcement learning (RL) techniques [9], dynamically adjusting the sampling window size to better handle obstacles in real-time, which improves efficiency and accuracy in dynamic environments.

C. Timed Elastic Band

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 [10].

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

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

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 [11] enhances real-time performance by reducing the overall computational burden while maintaining a high success rate for dynamic obstacle avoidance.

D. Model Predictive Control

MPC computes optimal control inputs over a prediction horizon by solving an optimization problem that accounts for both system dynamics and environmental constraints. The fundamental principle of MPC is to generate a trajectory by forward-simulating the system's dynamics over a prediction horizon, while adhering to constraints and minimizing a cost function. At each time step, the optimization problem is solved, and only the first control input is applied, making the process iterative and reactive to dynamic environments.

The optimization problem involves the system model $x_{k+1} = f(x_k, u_k)$, where x_k represents the system state and u_k represents the control input at time step k. The MPC aims to minimize the cost function $J(\vec{x}_k, \vec{u}_k)$, as shown in Equation (4):

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

where $\phi(\vec{x}(N))$ is the terminal cost, and $L(\vec{x}(k), \vec{u}(k))$ represents the running cost over each time step k. The term $\phi(\vec{x}(N))$ penalizes the deviation of the final state $\vec{x}(N)$ from the desired target state, ensuring that the robot closely follows the planned trajectory by the end of the prediction horizon. The running cost $L(\vec{x}(k), \vec{u}(k))$ is a quadratic function of both the state and the control input, where:

- Q is the state weighting matrix, which penalizes deviations of the robot's state from the desired reference trajectory. Larger values in Q correspond to higher penalties for errors in specific state variables, such as position or velocity.

- R is the control input weighting matrix, which penalizes large or sudden control inputs. This encourages smoother and more efficient control actions, avoiding erratic behavior.

- S is the terminal cost weighting matrix, which emphasizes the importance of reaching the target state at the end of the prediction horizon, ensuring that the final state is as close as possible to the desired state. The overall objective of the cost function is to balance minimizing the tracking error while ensuring efficient use of control inputs.

Compared to DWA and TEB, MPC offers superior handling of dynamic obstacles by continuously adjusting the robot's trajectory in real-time, optimizing for both safety and performance. Unlike DWA, which samples a range of velocities and often struggles with sharp turns or dense obstacle fields due to its reactive nature, MPC employs a predictive approach that anticipates future states, making it more adaptable to dynamic scenarios [4]. Additionally, while TEB excels at generating smooth paths, it frequently encounters computational challenges in complex environments, especially when faced with multiple moving obstacles. The non-convex optimization problem that TEB solves can result in suboptimal paths or deadlocks, particularly when obstacles move unpredictably [4], [5].

MPC, on the other hand, directly integrates the system's nonlinear dynamics and constraints into its optimization process. By predicting future states and control inputs over a finite prediction horizon, MPC is capable of generating near-optimal paths that ensure collision-free navigation, even in highly dynamic environments. An additional advantage of the nMPC controller is its ability to enforce both hard and soft constraints. The soft constraints allow for minor violations (penalizing the cost function when predicted states approach obstacles), thereby guiding the robot towards safer paths. The hard constraints, on the other hand, guarantee a strictly collision-free trajectory by setting firm limits that cannot be breached. This dual-constraint structure provides flexibility in optimization while ensuring safety [12].

However, one of the long-standing challenges of MPC has been its computational intensity. Solving a constrained optimization problem in real-time, especially with a large prediction horizon, demands significant computational resources [13], [14]. To overcome this challenge, a sparse obstacle representation is integrated into the proposed nMPC framework. By selectively focusing on the most relevant obstacles and discarding less critical data, the computational load is reduced, allowing for real-time performance while retaining the predictive capabilities of MPC. This integration not only lowers the computational cost but also enhances the planner's ability to efficiently handle dynamic environments with high obstacle density [15].

III. Methods

Figure 2 illustrates the individual components of the nMPC local planner. The main elements are the trajectory planner, nMPC and the obstacle detection algorithm.

The trajectory planner receives the global path from a planner based on Dijkstra's search. 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 signal $\mathbf{u} = \begin{bmatrix} v & \omega \end{bmatrix}^\top$, until the target is reached.

The robot is localised in a predefined map using probabilistic localisation methods [1]. Based on the scan data from the LiDAR, the Differential Drive Mobile Robot (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. By sending only the most relevant obstacles to the nMPC, the computational load is minimized, allowing for real-time decision-making.

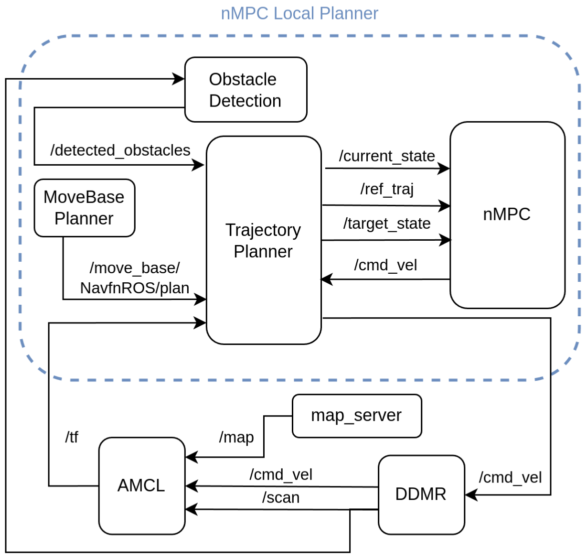

Fig. 2: nMPC-Local-Planner Block Diagram

A. Nonlinear Model Predictive Controller

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 3 illustrates the components of an nMPC [16].

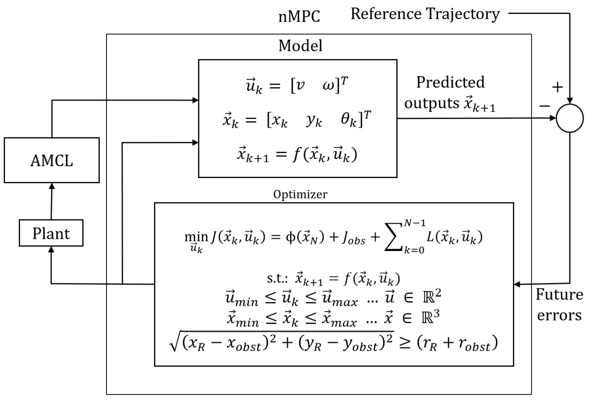
nMPC
Reference Trajectory
Model
$\vec{u}_k = [v \quad \omega]^T$
$\vec{x}_k = [x_k \quad y_k \quad \theta_k]^T$
$\vec{x}_{k+1} = f(\vec{x}_k, \vec{u}_k)$
Predicted outputs $\vec{x}_{k+1}$
+
−
AMCL
Plant
Optimizer
$\min_{\vec{u}_k} J(\vec{x}_k, \vec{u}_k) = \phi(\vec{x}_N) + J_{obs} + \sum_{k=0}^{N-1} L(\vec{x}_k, \vec{u}_k)$
s.t.: $\vec{x}_{k+1} = f(\vec{x}_k, \vec{u}_k)$
$\vec{u}_{min} \leq \vec{u}_k \leq \vec{u}_{max} \ \dots \ \vec{u} \in \mathbb{R}^2$
$\vec{x}_{min} \leq \vec{x}_k \leq \vec{x}_{max} \ \dots \ \vec{x} \in \mathbb{R}^3$
$\sqrt{(x_R - x_{obst})^2 + (y_R - y_{obst})^2} \geq (r_R + r_{obst})$
Future errors

Fig. 3: Detailed Blockdiagram of nMPC (modified from [17])

First, the Optimal Control Problem (OCP) must be converted into a Nonlinear Programming Problem (NLP) using the multiple shooting discretization method. Unlike single shooting, where only the control vector is optimized and the entire system is forward-simulated, multiple shooting introduces both the control vector and the state vector as optimization variables. This means that not only are the control inputs $\mathbf{u}$ optimized, but the system states $\mathbf{x}$ at each prediction step are also included in the optimization [18].

The cost function (see Eq. 5) is not merely a summation over all time steps, but instead, each segment's dynamics must adhere to the system's model equations, expressed as equality constraints (see Fig. 3). This enforces that the model dynamics are satisfied over the entire prediction horizon, and discrepancies between predicted and simulated states are minimized, improving both stability and accuracy [19].

Moreover, this approach significantly enhances the computational time efficiency. By breaking the problem into smaller segments, the optimization solver can converge faster and more reliably, especially in the presence of complex nonlinear dynamics or large prediction horizons. The ability to treat state and control variables as separate optimization entities helps in avoiding divergence [18].

The goal is to minimize the cost function (see Eq. 5), which consists of the States and control actions for all predictions [20].

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

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 [20].

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

- J_{obs} is a penalty term that increases the cost if the robot comes close to an obstacle, calculated according to Equation 7 [21].

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

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 [22].

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

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

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 [22].

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

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

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.

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

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 [21].

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 global planner 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, the orientation of each waypoint based on the direction to the next point along the path has been implemented. 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 4. In the example, three waypoints are given, and the algorithm linearly interpolates 10 points for 10 predictions.

Algorithm 1 Trajectory Interpolation. Variables: $P = \{(x_i, y_i, \text{yaw}_i)\}$, d_{pred}, N, $v_{\max}$, α

```
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:
   $$T = \frac{d_{\text{point}}}{v_{\max}}$$
5: Adjust time step with scaling factor:
   $$T = T \cdot \alpha$$
6: Interpolate $x$, $y$, and yaw to get $N$ prediction points
7: return $P_{\text{new}}$, $T$
```

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 [23].

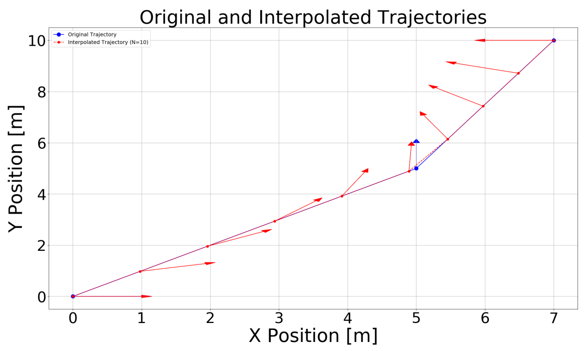

Fig. 4: Interpolation process: Given 3 initial waypoints, 10 linearly interpolated waypoints are generated for 10 nMPC predictions.

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, by detecting corners [24]. 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. By sending only the most relevant obstacles to the nMPC, the computational load is minimized, allowing for real-time decision-making.

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

A visual representation of the obstacle detection algorithm is shown in Figure 5, illustrating how the laser scan data are filtered, converted, and processed to detect obstacles before passing the data to the nMPC.

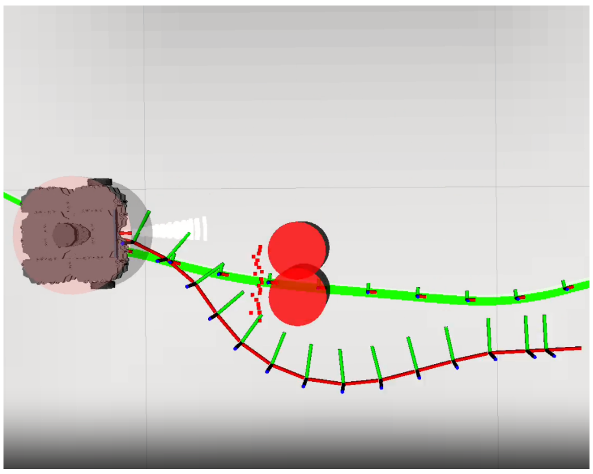

Fig. 5: Obstacle Detection Algorithm in which the DDMR detects a cube with the scan measurement and representing it as two circles

The integration of the obstacle detection algorithm

Algorithm 2 Shi-Thomas Obstacle Detection

```
1: Filter laser scan data $(r_i, \theta_i)$ 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:
     $$x_{\text{img}} = \left(\frac{x_i}{r_{\max}} \times I_{\text{size}}\right) + \frac{I_{\text{size}}}{2}, \quad y_{\text{img}} = \left(\frac{y_i}{r_{\max}} \times I_{\text{size}}\right) + \frac{I_{\text{size}}}{2}$$
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
```

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.

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, 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. Software Setup

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. Additionally, this work was containerized using Docker to ensure a consistent development environment, simplifying deployment and facilitating collaboration.

C. 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

In this section, the results are presented in detail and evaluated on the basis of the previously defined metrics, such as cross-tracking error, arrival time and calculation time. It analyses how well the developed nMPC Local Planner performed in comparison to other methods such as DWA and TEB. In particular, the extent to which the soft and hard constraints used contributed to obstacle avoidance and the role that parameters such as prediction length and obstacle detection played in performance in dynamic environments are discussed.

A. Evaluation Metrics

The experiments were conducted using three key metrics:

- Cross Tracking Error: The CTE measures the accuracy of path tracking by calculating the minimum distance from the robot's path to the reference trajectory. This metric helps evaluate how well the robot follows the desired path [25].

$$CTE = \frac{1}{N}\sum_{i=1}^{N}\min\left(\sqrt{(x_{\text{robot},i} - x_{\text{global},j})^2} + \sqrt{(y_{\text{robot},i} - y_{\text{global},j})^2}\right) \quad (15)$$

where N is the total number of points along the path. (x_i, y_i) represents the coordinates of the actual position of the robot at point i, and $(x_{\text{ref},i}, y_{\text{ref},i})$ are the coordinates of the reference path. The term inside the square root is the Euclidean distance between the actual position and the reference path at each point, and the minimum selects the closest point on the path.

- Arrival Time: This metric measures the time taken for the robot to reach its target waypoint. It is crucial for evaluating the efficiency of local planners, especially in dynamic environments where timely responses are important [26].
- Computation Time: The average time required for the local planner to compute the control inputs is used to assess real-time performance (see Eq. 16). This helps evaluate the planner's efficiency in generating control inputs quickly [27].

$$t_{\text{comp}} = \frac{1}{N-1}\sum_{i=1}^{N-1}(t_{i+1} - t_i) \quad (16)$$

where N is the total number of 'cmd_vel' messages, t_i is the timestamp of the i-th message, and $t_{i+1} - t_i$ represents the time difference between consecutive messages.

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.

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.

B. Experiment 1: Dynamic Obstacle Avoidance

In the first experiment in Figure 6, 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 7, 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.

C. 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 9. 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.

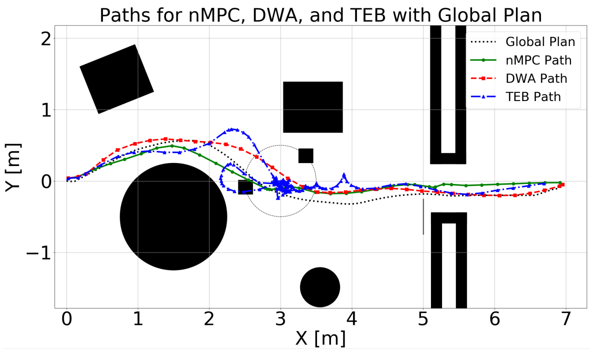

Fig. 6: Comparison of paths for nMPC, DWA, and TEB in Experiment 1

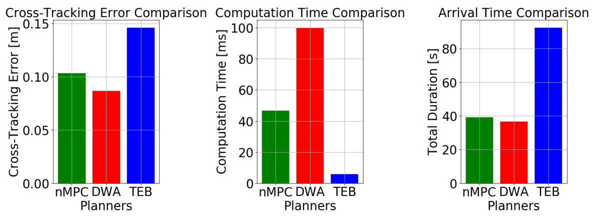

Fig. 7: Metrics comparison (cross tracking error, computation time, arrival time) for nMPC, DWA, and TEB in Experiment 1

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 11, 7 and 9.

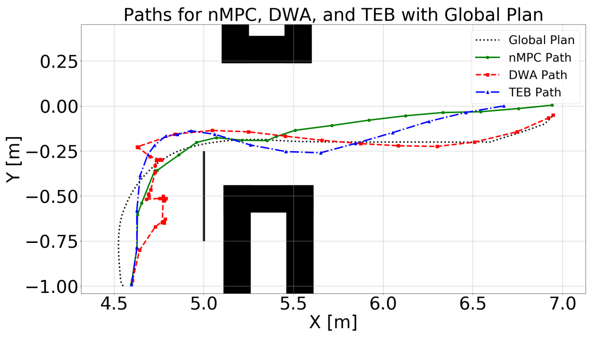

Fig. 8: Comparison of paths for nMPC, DWA, and TEB in Experiment 2

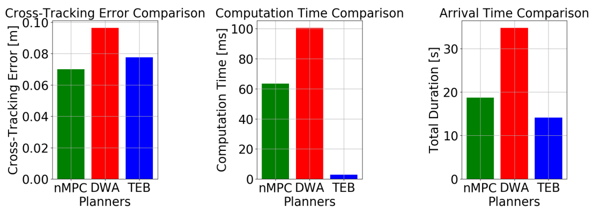

Fig. 9: Metrics comparison (cross tracking error, computation time, arrival time) for nMPC, DWA, and TEB in Experiment 2

D. 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.

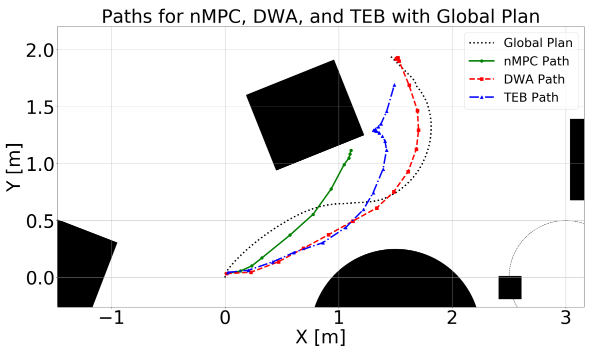

Fig. 10: Comparison of paths for nMPC, DWA, and TEB in Ablation Experiment

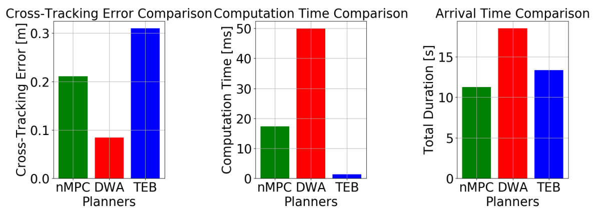

Fig. 11: Metrics comparison (cross tracking error, computation time, arrival time) for nMPC, DWA, and TEB in Ablation Experiment

VI. Summary and Outlook

The results demonstrate that the nMPC with soft constraints for obstacle clearance penalization and hard constraints to ensure collision-free paths outperformed the TEB planner in dynamic environments, reducing arrival time by approximately 50 seconds. Compared to DWA, the nMPC was 18 seconds faster in narrow navigation tasks. Although the TEB planner was 2 seconds quicker in very tight passages, the nMPC achieved a lower cross-tracking error relative to the global path. The implemented nMPC offers a flexible local planner, configurable with prediction length, number of obstacles to avoid, and Shi-Tomasi quality level for corner detection. However, further improvements in computation time are necessary to avoid more obstacles without significantly impacting performance. Future work will involve testing the nMPC on real hardware and extending the model to control tracked vehicles using mathematical, data-driven, or hybrid models.

References

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

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

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

[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 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] ——, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[8] 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.

[9] 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.

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

[11] 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

[12] Y. Wan, J. Tang, S. Lao, and Z. Zhao, "A distributed autonomous system for multi-uavs with limited visualization: Employing dual-horizon nmpc controller," *IEEE Transactions on Aerospace and Electronic Systems*, pp. 1–16, 2024.

[13] 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

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

[15] B. Lindqvist, S. S. Mansouri, A.-a. Agha-mohammadi, and G. Nikolakopoulos, "Nonlinear mpc for collision avoidance and control of uavs with dynamic obstacles," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6001–6008, 2020.

[16] 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.

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

[18] H. Bock and K. Plitt, "A multiple shooting algorithm for direct solution of optimal control problems*," *IFAC Proceedings Volumes*, vol. 17, no. 2, pp. 1603–1608, 1984, 9th IFAC World Congress: A Bridge Between Control Science and Technology, Budapest, Hungary, 2-6 July 1984. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667017612059

[19] M. Diehl, H. Bock, J. P. Schlöder, R. Findeisen, Z. Nagy, and F. Allgöwer, "Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations," *Journal of Process Control*, vol. 12, no. 4, pp. 577–585, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959152401000233

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

[21] 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.

[22] 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.

[23] 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

[24] J. Shi and Tomasi, "Good features to track," in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1994, pp. 593–600.

[25] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135–1145, 2016.

[26] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a frenét frame," in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 987–993.

[27] J. Bala, S. Adeshina, and A. Aibinu, "Implementing nonlinear model predictive control for enhanced trajectory tracking and road anomaly avoidance in autonomous vehicles," 11 2023, pp. 1–5.