CHALMERS
UNIVERSITY OF TECHNOLOGY

Trajectory Generation for Mobile Robots in a Dynamic Environment using Nonlinear Model Predictive Control

Citation for the original published paper (version of record):

Berlin, J., Hess, G., Karlsson, A. et al (2021). Trajectory Generation for Mobile Robots in a Dynamic Environment using Nonlinear Model
Predictive Control. IEEE International Conference on Automation Science and Engineering, 2021-August: 942-947. http://dx.doi.org/10.1109/CASE49439.2021.9551644

Trajectory Generation for Mobile Robots in a Dynamic Environment using Nonlinear Model Predictive Control

Jonas Berlin[1] Georg Hess[1] Anton Karlsson[1] William Ljungbergh[1]
Ze Zhang[2] Per-Lage Götvall[3] Knut Åkesson[2]

***Abstract*— This paper presents an approach to collision-free, long-range trajectory generation for a mobile robot in an industrial environment with static and dynamic obstacles. For the long-range planning a visibility graph together with A* is used to find a collision-free path with respect to the static obstacles. This path is used as a reference path to the trajectory planning algorithm that in addition handles dynamic obstacles while complying with the robot dynamics and constraints. A Nonlinear Model Predictive Control (NMPC) solver generates a collision-free trajectory by staying close to the initial path but at the same time obeying all constraints. The NMPC problem is solved efficiently by leveraging the new numerical optimization method Proximal Averaged Newton for Optimal Control (PANOC). The algorithm was evaluated by simulation in various environments and successfully generated feasible trajectories spanning hundreds of meters in a tractable time frame.**

I. Introduction

Mobile robots that autonomously navigate in an environment with static and dynamic obstacles have to solve two main problems, (i) they have to find a global path from the source to the destination, and (ii) they have to locally plan a trajectory that avoids collision with moving obstacles, e.g., humans and forklifts. There exists a large number of approaches to do motion planning for autonomous vehicles. An overview of different motion planning techniques for autonomous vehicles are presented in [1, 2]. The problem is typically divided into a global *route planner* and a local *trajectory planner*. For the route planner the task is to find a path from source to destination. In a factory setting this typically involves following preferred road networks and avoiding static obstacles. The route planning problem can be formulated as minimum-cost path problems defined over a graph where a weight on an edge describe the distance between the nodes that are connected by the edge. The nodes can originate from a discretization of the 2-D space or from geometric methods where obstacles are models as polyhedrals [3]. In both these cases shortest path problems can be solved using A* [4] or approximate, but faster, variants like Anytime Dynamic A* [5].

[1] Department of Electrical Engineering Chalmers University of Technology, 41296 Gothenburg, Sweden {jberlin, georghe, antka, willju}@student.chalmers.se

[2] Department of Electrical Engineering Chalmers University of Technology, 41296 Gothenburg, Sweden {zhze, knut.akesson}@chalmers.se

[3] AB Volvo per-lage.gotvall@volvo.com

We gratefully acknowledge financial support from Chalmers AI Research Centre (CHAIR) and AB Volvo (Project ViMCoR). Code is available: https://github.com/wljungbergh/mpc-trajectory-generator

Route planners do, typically not, consider robot kinematics and dynamics, nor do they handle dynamic obstacles. However, global route planners are able to cover large distances in a computationally efficient manner, which makes them suitable for the long-horizon planning problems, but they have to be complemented with local trajectory planners that will take robot kinematics, dynamics, and also dynamic obstacles into account.

Local trajectory planners are concerned with generating a collision-free path for a part of the path between the source and destination. These planners generally consider vehicle kinematics and dynamics and constraints on velocity and acceleration, and yield feasible *trajectories* for a given robot. A local trajectory planning may also take moving obstacles into account by generating a trajectory that avoids a collision between the robot and the moving obstacle. There are many approaches to trajectory planning, for example sampling-based planners like RRT[6], or planners based on interpolating curves that computes a smooth path given a set of way-points, see e.g. [7]. However, to handle situations with moving obstacles it is advantageous to use an on-line approach that computes control actions for the robot based on the observed and predicted future states of the environment. Optimization-based trajectory planning algorithms, such as Model-Predictive Control (MPC), are very promising due to two reasons, (i) they can naturally encode constraints on velocity, acceleration, and jerk as well as distance constraints to obstacles inside its formulation, and (ii) the algorithms for solving optimization problems has improved considerably over the last years, this together with the increased performance of common processors has made it feasible to solve complex optimization problems in real-time.

The method Proximal Averaged Newton-type method for Optimal Control (PANOC) was proposed in [8] and shows computation times multiple orders faster than interior point (IP) and sequential quadratic programming (SQP) for many applications. In [9], different solvers were compared for non-linear MPC (NMPC) obstacle avoidance, showing the superiority of PANOC when it comes to robustness and solving time. Further, the PANOC solver has successfully been used for real-time NMPC with obstacle avoidance in applications such as [9, 10].

In [10] a novel framework was introduced for modeling generic, possibly non-convex, shapes of obstacles by describing them as a set of non-linear inequalities. However, as described by [9], the framework has the risk of getting stuck in local minima. To avoid this, [9] combined the modeling

approach with several heuristics such as using graph search to guide the solution out from local minima. Nevertheless, none of these approaches are suitable to find long-range trajectories.

In order to generate trajectories at larger scales, we propose an approach leveraging the global route planners optimality and computational efficiency with the a local trajectory based planner based on an NMPC controller solving several NMPC problems in an iterative manner. Specifically, A* is used to find the optimal path while NMPC follows this path closely and generates a feasible trajectory. To solve the NMPC problems iteratively in a computationally efficient manner, the PANOC solver is used. Our contribution is a new combination of global and local path planning techniques, yielding an algorithm that can generate collision-free trajectories using NMPC for distances larger than a few meters while being able to handle arbitrary shapes of static obstacles as well as avoiding collision with dynamic obstacles.

II. Method

A. Algorithm overview

In Fig. 1, a flowchart of the proposed algorithm is shown. The algorithm mainly consists of two parts. First, A* is used to find waypoints for the shortest path. Secondly, an NMPC solver is used iteratively to follow said waypoints and generate a smooth trajectory. The A* algorithm can consider arbitrary polygon shapes as static obstacles, which are translated to simpler constraints in the NMPC setting. The reasoning is that this simplifies the optimization problem and thus should improve the robustness and computation time of the NMPC solver.

As input, the algorithm is fed a set of static obstacle polygons P_{static}, an outer boundary polygon B, initial state $\mathbf{x}_{init}$ and a target state $\mathbf{x}_{end}$, an illustrative example is shown in Fig. 2. To account for A* considering the robot to be a point object, all polygons in P_{static} are inflated by half the robot width together with some additional safety margin. Similarly, B is deflated by the same amount. The set of all polygons is then converted to a visibility graph where the set of vertices also includes the starting and target positions extracted from $\mathbf{x}_{init}$ and $\mathbf{x}_{end}$ respectively. The waypoints found when running the A* algorithm on the visibility graph then consist of start and target position, as well as polygon vertices that must be cornered, e.g. the upper left polygon vertex in Fig. 2. Before starting the NMPC solver, the waypoints are correlated to the non-padded polygon vertices and their coordinates are saved as $\mathbf{o}_{\text{static}} = [x_{obs}, y_{obs}]^T$ in $\mathcal{O}_{\text{static}}$, e.g. in Fig. 2 this corresponds to $\mathcal{O}_{\text{static}} = \{[3, 7]^T\}$. Further, a sequence L with line segments of equal length is extracted from the waypoints.

Once the initial processing is finished, the trajectory $\mathcal{X}$ is initiated as the initial state. Next, the NMPC solver is called in an iterative manner. Let N denote the horizon used in the NMPC and let $n_{obs,static}$ and $n_{obs,dynamic}$ be the maximum number of static and dynamic obstacles considered in each NMPC iteration. In each iteration, the latest state in the trajectory is set as the current state and based on this the N coming line segments from L are extracted as well as the $n_{obs,static}$ closest obstacle coordinates in $\mathcal{O}_{\text{static}}$. Further, the predicted location and shape of $n_{obs,dynamic}$ dynamical obstacles for the next N time steps are extracted. Given these inputs, the NMPC solver is used to find a sequence of feasible actions $\hat{\mathbf{u}}_{(0:N-1)}$ for a given motion model. The corresponding states $\mathbf{x}$ should be close to the line segments, i.e. have a low cross-track error to the desired path, keep at least a distance of r to the selected $n_{obs,static}$ obstacle points in $\mathcal{O}_{\text{static}}$, keep a sufficient distance to the $n_{obs,dynamic}$ dynamical obstacles and respect vehicle dynamics and constraints. From the sequence $\hat{\mathbf{u}}_{(0:N-1)}$, the first action is used to calculate the new state to append to the trajectory $\mathcal{X}$. If the last state is not close to the target state the next iteration is

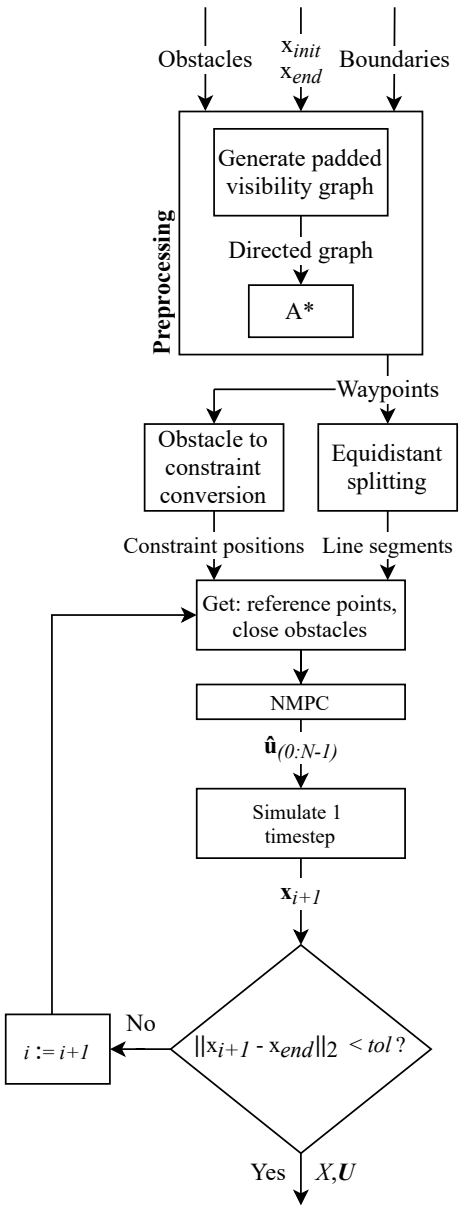


Fig. 1. Flowchart of the proposed algorithm.

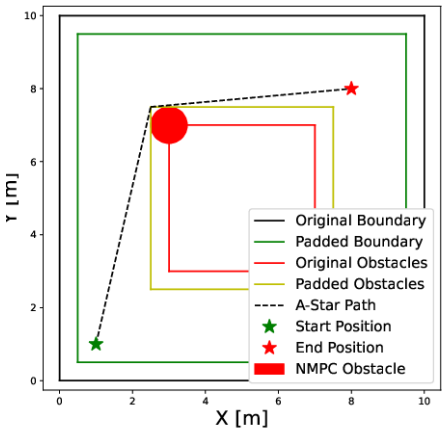


Fig. 2. Example environment displaying padding of boundary and obstacle, A* path and obstacle modelling in NMPC. The original enclosing boundary is shown as a black line, while the green line is used for the A* path. Similarly, the original static obstacle is shown with a red line, while the padded yellow version is used when deploying the A*-algorithm. Lastly, the obstacle corner where the A* path traverses is translated to a circular NMPC obstacle, such that the robot does not collide with the static obstacle.

started, i.e. the current state is updated and new line segments and obstacle locations are fed to the NMPC solver.

As described above and shown in Fig. 2, obstacles are modeled in different ways in the visibility graph and the NMPC formulation. In the visibility graph, obstacles are allowed to have any polygon shape. For the NMPC formulation, these are translated such that the trajectory must keep a certain distance of at least r to polygon vertices that are along the A* path, e.g. in Fig. 2 r is shown as the radius of the red circle. To ensure that the trajectory does not overlap with polygons that are not modeled in the NMPC, e.g. the boundary in Fig. 2, the cross-track error to the reference path must be kept sufficiently small. While not shown in the figure, dynamic obstacles are modelled as ellipses. This is due to the fact that their true locations in general are not known beforehand, and ellipses can be used to model their position as a Gaussian distribution.

B. NMPC Formulation

The dynamics in the NMPC formulation are modelled using a discrete motion model for a differential drive robot as in (1), where i denotes the current trajectory time step and T_s denotes the step size. The state $\mathbf{x}_i$ in the model consists of the 2D coordinates x_i, y_i, and the heading θ_i, while the inputs $\mathbf{u}_i$ to the system are velocity v_i and angular velocity ω_i. Acceleration is not included in the motion model but the change in the input is bounded in the coming NMPC constraints, and thus acceleration is not neglected. Note that the proposed algorithm is not dependent on this specific motion model but the differential drive is used since it is illustrative for the mobile robot use-case.

$$f(\mathbf{x}_i, \mathbf{u}_i) = \begin{bmatrix} x_i + v_i \cos(\theta_i) T_s \\ y_i + v_i \sin(\theta_i) T_s \\ \theta_i + \omega_i T_s \end{bmatrix} \tag{1}$$

The NMPC-controller calculates an optimal control sequence in regard to a cost function and constraints for the predicted states and control actions over a horizon of length N. Predicted states and inputs within the NMPC controller are denoted as $\hat{\mathbf{x}}$ and $\hat{\mathbf{u}}$ respectively and use time index j, while the states and control actions that define the generated trajectory are denoted $\mathbf{x}$ and $\mathbf{u}$ respectively and use time index i. The selected cost function consists of multiple parts,

$$J_{\text{cte},j} = d(\hat{\mathbf{x}}_j, \mathbf{l}_{\text{A}^*})^T \, Q_{cte} \, d(\hat{\mathbf{x}}_j, \mathbf{l}_{\text{A}^*}) \tag{2a}$$
$$J_{v,j} = (\hat{v}_j - v_{r,j})^T \, R_v \, (\hat{v}_j - v_{r,j}) \tag{2b}$$
$$J_{\text{acc},j} = (\hat{\mathbf{u}}_j - \hat{\mathbf{u}}_{j-1})^T \, R_{\text{d}} \, (\hat{\mathbf{u}}_j - \hat{\mathbf{u}}_{j-1}) \tag{2c}$$

where J_{cte} is the cross-track error (CTE) cost. For a state $\hat{\mathbf{x}}_j$ and a set of line segments $\mathbf{l}_{\text{A}^*}$, $d(\hat{\mathbf{x}}_i, \mathbf{l}_{\text{A}^*})$ finds the minimum distance between the state and any of the line segments. J_v is a cost for difference between predicted velocity $\hat{v}_j$ and the reference velocity $v_{r,j}$ at each time instance. Finally, J_{acc} is the cost for change in input i.e acceleration. For coherence between iterations, this cost is also applied to $\hat{\mathbf{u}}_0 - \mathbf{u}_{i-1}$ where $\mathbf{u}_{i-1}$ is the last applied input from the previous iteration. Conventionally a cost for the predicted terminal state is included to provide incentive for the robot move forward, however this cost has been omitted as that incentive is provided by other costs. Q_{cte}, R_v and R_d are tuning parameters that can be modified for desired controller behaviour.

Dynamical obstacles are modeled as ellipses with five time dependent parameters namely the coordinates $\mathbf{o}_{x,j}$ and $\mathbf{o}_{y,j}$, width $\mathbf{o}_{w,j}$, height $\mathbf{o}_{h,j}$ and heading $\mathbf{o}_{\alpha,j}$. The reason width and height are time dependent is to potentially model uncertainty in obstacle location. The robot is at time j not allowed to violate the area $\mathcal{D}_j$ occupied by all ellipses at time j. Let $\mathcal{O}_{dynamic,j}$ denote the dynamic obstacle states at time j, then $\mathcal{D}_j$ can be written as,

$$\begin{aligned} \mathcal{D}_j = \{(x,y) | \mathbf{o}_j \in \mathcal{O}_{dynamic,j}, \\ \frac{\left((x - \mathbf{o}_{x,j})\cos(\mathbf{o}_{\alpha,j}) + (y - \mathbf{o}_{y,j})\sin(\mathbf{o}_{\alpha,j})\right)^2}{\mathbf{o}_{w,j}^2} + \\ \frac{\left((x - \mathbf{o}_{x,j})\sin(\mathbf{o}_{\alpha,j}) - (y - \mathbf{o}_{y,j})\cos(\mathbf{o}_{\alpha,j})\right)^2}{\mathbf{o}_{h,j}^2} \leq 1\} \end{aligned} \tag{3}$$

These cost, constraints from obstacles and dynamics make

up the optimization problem in (4).

$$\min_{\hat{\mathbf{u}}_{(0:N-1)}} \sum_{j=0}^{N} J_{\text{cte},j} + J_{v,j} + J_{\text{acc},j} \tag{4a}$$

$$\text{subject to: } \hat{\mathbf{x}}_0 = \mathbf{x}_i \tag{4b}$$

$$||[\hat{x}_j, \hat{y}_j]^T - \mathbf{o}||_2 \geq r, \ \forall \mathbf{o} \in \mathcal{O}_l, j = 0, \ldots, N \tag{4c}$$

$$(\hat{x}_j, \hat{y}_j) \notin \mathcal{D}_j \tag{4d}$$

$$\hat{\mathbf{x}}_{j+1} = f(\hat{\mathbf{x}}_j, \hat{\mathbf{u}}_j) \tag{4e}$$

$$\hat{\mathbf{u}}_j \in [\mathbf{u}_{min}, \mathbf{u}_{max}] \tag{4f}$$

$$\frac{\Delta \hat{\mathbf{u}}}{T_s} \in [\dot{\mathbf{u}}_{min}, \dot{\mathbf{u}}_{max}] \tag{4g}$$

Here $\mathcal{O}_l$ is a subset of all static obstacles $\mathcal{O}_{\text{static}}$, specifically, the $n_{obs,static}$ closest ones as described earlier. To ensure coherence, the first state of the NMPC solver $\hat{\mathbf{x}}_0$ is set to the latest state from the trajectory generated so far $\mathbf{x}_i$. Finally, the constraint on $\Delta\hat{\mathbf{u}} = \hat{\mathbf{u}}_j - \hat{\mathbf{u}}_{j-1}$ is also applied to $\hat{\mathbf{u}}_0 - \mathbf{u}_{i-1}$, again to ensure coherence between iterations.

The optimization problem in (4) is solved using the PANOC optimizer, this yields a sequence of control actions. Note, that we, in this work, assume that no disturbances are present. The first control action is supplied to the motion model and the robots movements are simulated one time step ahead. This process will be iterated until the final state is reached as is shown in the flowchart in Fig. 1.

III. Evaluation

The proposed solution was evaluated via simulation and the algorithm was tested on multiple constellations of layouts and obstacle configurations to ensure robustness. However, only the test-case shown in Fig. 3 will be considered in this section. Note that the dynamical obstacles are not shown in this figure as their positions vary with time, but are displayed in Fig. 5 to 7.

Throughout the evaluation, a differential drive motion model, as described in (1) was used to simulate the movement of the mobile robot and the following values were selected as the tuning profile. The cost for velocity deviation from the reference speed R_v was set to $R_v = 10.0$, the cost of deviation from the global path Q_{cte} was set to $Q_{cte} = 200$ and the cost for linear and angular acceleration was set to $R_d = \text{diag}(\begin{bmatrix}10.0 & 5.0\end{bmatrix})$ respectively. Furthermore, the length of the receding horizon was set to $N = 20$ and the algorithm operated with a time-step size of $T_s = 0.2$. Lastly, the linear and angular constraints imposed on the vehicle was chosen to reflect that of a mobile robot, which is shown in (5).

$$\mathbf{u}_{min} = \begin{bmatrix}-0.5 & -0.5\end{bmatrix}^T \quad \mathbf{u}_{max} = \begin{bmatrix}1.5 & 0.5\end{bmatrix}^T \tag{5a}$$

$$\dot{\mathbf{u}}_{min} = \begin{bmatrix}-1.0 & -3.0\end{bmatrix}^T \quad \dot{\mathbf{u}}_{max} = \begin{bmatrix}1.0 & 3.0\end{bmatrix}^T \tag{5b}$$

Our proposed solution have displayed several different desirable characteristics that can be highlighted using the example shown in Fig. 3. By inspecting the generated linear and angular velocity profiles, shown in Fig. 4, it is clear that the NMPC-controller decreases the linear velocity as it increases the angular velocity. In other words it slows down when taking sharp corners to avoid deviating too much from the reference path. Furthermore, from Fig. 3 and 4, it is evident that the proposed approach computes a smooth trajectory, even around very sharp corners, that satisfies the non-holonomic motion constraints of the differential drive robot while keeping the deviation from the global reference path small.

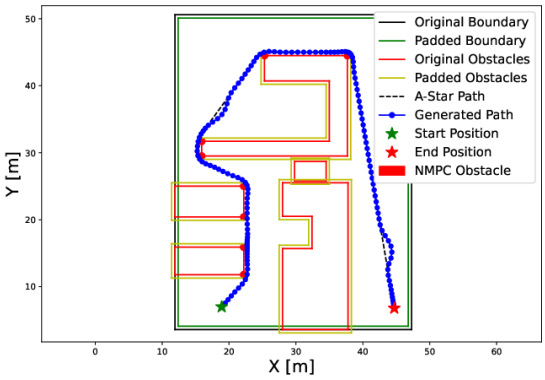


Fig. 3. Generated reference trajectory throughout the layout. Note that the dynamical obstacles (shown in Fig. 5 - 7) are not shown as their positions are varying over time. Video of the simulation can be found `https://youtu.be/jiRxyRfNJK8`

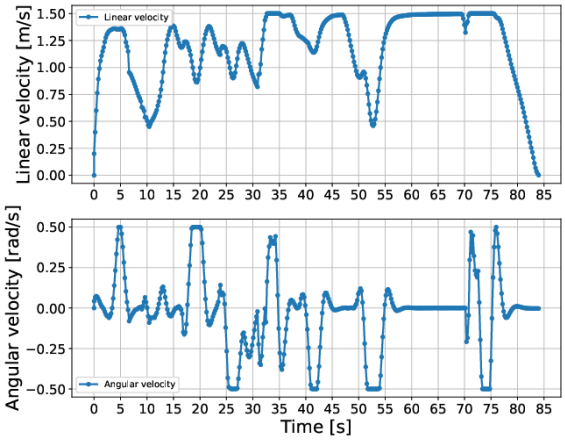
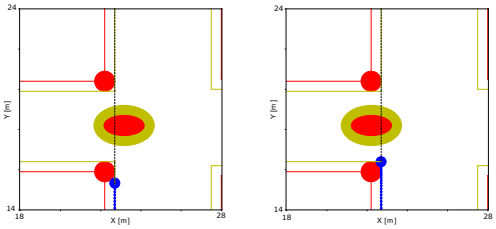


Fig. 4. Linear and angular velocity profile for the reference trajectory shown in Fig. 3

.

The time-lapse in Fig. 5 displays a dynamic obstacle crossing the global path perpendicularly as the mobile robot is to traverse it. As the NMPC-controller has access to the future positions of said obstacle it can calculate that decreasing the velocity and allowing the obstacle to pass before proceeding would enable it to maintain a small distance to the global path.

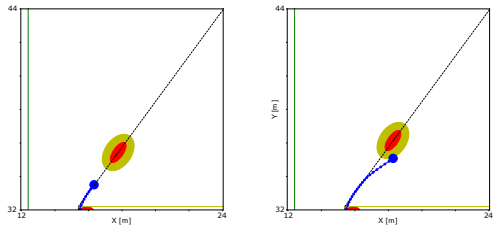
In Fig. 6 the mobile robot is blocked by a slow-moving obstacle travelling along the same path. Here, the NMPC-



(a) $t = 9.0$ (b) $t = 11.0$

(c) $t = 13.0$ (d) $t = 15.0$

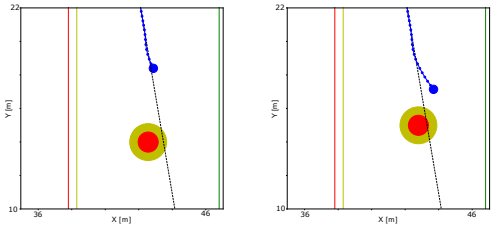
Fig. 5. A dynamical obstacle crossing the global path perpendicularly, causing the mobile robot to slow down in order for the obstacle to pass. The extent of the moving obstacle is shown as a red ellipse, while the yellow ellipse is used in the NMPC formulation to account for the extent of the ego-robot. Video at `https://youtu.be/95Or9fLlIpM`



(a) $t = 31.0$ (b) $t = 33.0$

(c) $t = 34.0$ (d) $t = 36.0$

Fig. 6. Slow-moving dynamical obstacle forces the mobile robot to deviate from the global path and increase its velocity to perform an overtaking. Video at: `https://youtu.be/aN73iwWjQUA`



(a) $t = 72.0$ (b) $t = 73.0$

(c) $t = 74.0$ (d) $t = 76.0$

Fig. 7. Mobile robot strays from the global path in order to stay clear of the oncoming dynamical obstacle. Video at: `https://youtu.be/DQarRKinpTw`

controller can not simply wait for the obstacle to pass but rather it can reason about the cost of deviating from the global path versus the cost of remaining behind the slowly moving obstacle. Depending on the tuning profile this can lead to different behaviors, however, using the profile described above leads to the overtaking that can be seen in the time-lapse in Fig. 6.

Finally, in Fig. 7 the mobile robot is on course for a collision with an oncoming obstacle. Even in this seemingly challenging case, where the obstacle is approaching with a relatively large velocity, the NMPC-controller can compute a smooth and collision-free trajectory by deviating from the global path. While the extent of the ego-robot intersects with the padded obstacle ellipse, it never collides with the true obstacle. Together, the scenarios shown in Fig. 5 to 7 display that the NPMC and algorithm can handle a range of different interaction scenarios with dynamic obstacles. It further illustrates that the algorithm is robust not only to different static obstacle configuration but also different types of dynamic obstacle interactions.

However, while the algorithm can handle a large variety of scenarios, one of which is shown in Fig. 3, there are certain cases where the NMPC solver generates trajectories that intersect the padded obstacles. In Fig. 8, one such case is depicted, where the path is heading north through a narrow corridor before turning 90 degrees into a different narrow corridor. For the NMPC solver, the only obstacle constraint is to keep a distance of at least 0.5 meters to the polygon corner marked with a red circle. Since the solver has no notion of other obstacles, there is nothing restricting it to stay within the drivable area. To stay close to the reference speed of 1.5 m/s, while respecting constraints on the angular velocity, the trajectory must deviate from the desired path. While the trajectory intersects the padded obstacle, the safety margin used when padding results in the trajectory to be collision-free. The robot width is set to 0.25 meters, and there is no contact between the robot and any obstacle. However, this shows that proper selection of all parameters is of utmost importance to generate collision-free trajectories and they must be selected based on each specific use-case. While this can be considered an edge-case it highlights the limitations

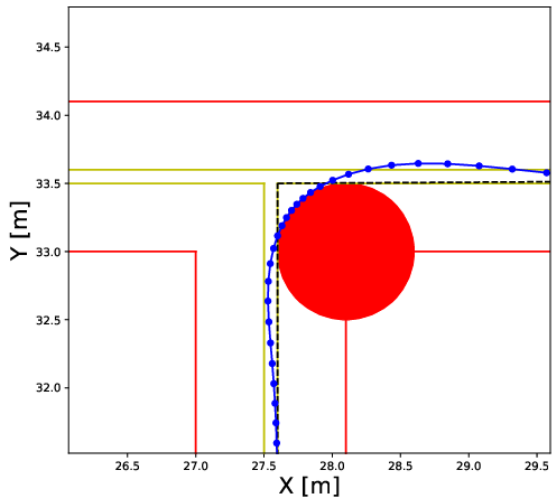


Fig. 8. Scenario for which the NMPC solver generated a trajectory intersecting with a padded obstacle. However, the safety margin ensures that the robot keeps sufficient distance to the original obstacle. The trajectory is going from the bottom to the right side.

of the proposed algorithm.

IV. Conclusion

In this paper we propose a novel combination of already established techniques to perform long-range trajectory generation. We leverage the optimality and computational efficiency of the A* algorithm to generate a reference path, consisting of linearly connected waypoints, from which we form a cross-track error loss in the NMPC objective function. The recent development in numerical optimization methods, namely PANOC, enables our approach to leverage the NMPC in an iterative manner while maintaining computation time sufficiently low. The iterative usage of the NMPC allows us to generate reference trajectories that span over longer distances than that of what previous solutions showcase.

To improve the robustness of the proposed approach, we suggest further investigations within three areas. Firstly, as mentioned earlier, in [10] a framework was presented on how to model general obstacle shapes in the NMPC formulation. It might be of interest to combine this type of obstacle modelling with our algorithm for cases where our original approach fails. However, one has to investigate how a more complex obstacle modelling affects the run-time. Secondly, the presented approach does not work for all possible motion models. There are situations where the A* algorithm will produce paths that cannot be followed closely by some motion models, e.g. one describing the motion of a car. More work has to be done to find ways to adjust our approach such that it can handle all common motion models.

References

[1] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016. DOI: 10.1109/TIV.2016.2578706.

[2] 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. DOI: 10.1109/TITS.2015.2498841.

[3] T. Lozano-Perez and M. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Commun. ACM*, vol. 22, pp. 560–570, 1979.

[4] N. J. Nilsson, *Principles of Artificial Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1980, ISBN: 0934613109.

[5] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic A*: An anytime, replanning algorithm.," Jan. 2005, pp. 262–271.

[6] S. M. LaValle and J. J. K. Jr., "Randomized kinodynamic planning.," *Int. J. Robotics Res.*, vol. 20, no. 5, pp. 378–400, 2001.

[7] M. Brezak and I. Petrović, "Real-time approximation of clothoids with bounded error for path planning applications," *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 507–515, 2014. DOI: 10.1109/TRO.2013.2283928.

[8] L. Stella, A. Themelis, P. Sopasakis, and P. Patrinos, "A simple and efficient algorithm for nonlinear model predictive control," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, IEEE, 2017, pp. 1939–1944.

[9] B. Hermans, P. Patrinos, and G. Pipeleers, "A penalty method based approach for autonomous navigation using nonlinear model predictive control," *IFAC-PapersOnLine*, vol. 51, no. 20, pp. 234–240, 2018.

[10] A. Sathya, P. Sopasakis, R. Van Parys, A. Themelis, G. Pipeleers, and P. Patrinos, "Embedded nonlinear model predictive control for obstacle avoidance using PANOC," in *2018 European control conference (ECC)*, IEEE, 2018, pp. 1523–1528.