Long-range Trajectory Generation with Obstacle Avoidance using Non-linear Model Predictive Control

Jonas Berlin jberlin@student.chalmers.se
Georg Hess georghe@student.chalmers.se
Anton Karlsson antka@student.chalmers.se
William Ljungbergh willju@student.chalmers.se

***Abstract*—This paper presents a novel approach to collision-free, long-range trajectory generation for a differential drive robot in an industrial environment with arbitrarily shaped obstacles. In order to handle larger distances than conventional trajectory generation approaches, the graph-search algorithm A* is used to find initial waypoints while a Non-linear Model Predictive Control (NMPC) solver produces a smooth trajectory by following said waypoints. Further, the NMPC solver ensures that the produced trajectory complies with vehicle dynamics and constraints, while minimizing a given objective function. The minimization 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.**

***Index Terms*—Trajectory generation, NMPC, Motion Planning, Obstacle avoidance, PANOC**

I. Introduction

The transportation industry is projected to undergo large change in the future for several reasons. Climate change and stricter emission restrictions have led to a surge in electrified products and hybrid technology. Manufacturers are expected to have larger variation in their product line-up, with more customization for each product. To account for this rise in product diversity, the next-generation manufacturing plants are required to be more versatile.

Simultaneously, large technology advancements have enabled the autonomization of vehicles traditionally operated by humans. One key component to enable a flexible manufacturing environment are Autonomous Transport Robots (ATRs). These can be used to transport the required materials to the assembly line as they are needed rather than having large storage devices in close proximity to the assembly line. However, rather than traveling along given static paths like traditional Automated Guided Vehicles (AGV), ATRs must be allowed to move more freely throughout the factory, similar to how humans would move to transport goods through a plant. This poses new challenges, as one has to find feasible paths for ATRs in a dynamic environment where obstacles such as pallets might be stationed at different locations over time.

Path planning, or motion planning, is a field within robotics and autonomous systems that is concerned with finding a collision-free path from a given state to a target. The topic has been explored extensively and is generally divided into global path planning and local path planning [1]. This division is concerned with the amount of information available to the algorithm when deciding on a path. For global path planning the path is optimized given the starting and terminal state as well as all known obstacles. In contrast, local path planning is optimized only for a smaller patch and obstacles present in that patch.

An overview of available approaches to solving the global path planning problem can be found in [2], but common approaches include graph search algorithms such as A* [3] and D* [4], or tangent graph-based planning, e.g. see [5] for an application. Global path planners are able to cover large distances in a computationally efficient manner, which makes them suitable to use for scales encountered in a manufacturing plant. Another advantage of using global path planners is that some of the algorithms, A* for instance, guarantee an optimal solution [2], allowing them to solve the shortest path problem globally. However, these algorithms are not able to consider vehicle dynamics or other constraints when generating a solution. Thus, there is no guarantee that the solution is feasible for a given motion model nor does the solution provide sensible control actions to follow the generated path.

Local path planners are concerned with generating a collision-free path for a smaller portion of the surrounding environment. These planners also consider vehicle dynamics and constraints such as velocity and acceleration limits, i.e. they yield feasible *trajectories* for a given vehicle or robot. Common approaches to finding trajectories include Timed-Elastic-Band (TEB)

[6], Dynamic Window Approach (DWA) [7] and recently also Model Predictive Control (MPC). The latter is a control strategy where an optimization problem is solved at every time step to determine appropriate control actions [8]. From the resulting sequence of control actions, only the first one is applied, and then the optimization problem is solved again at the next time instance. By using the resulting control actions as inputs to a motion model, MPC can be used for trajectory generation. The optimization problem can be formulated to easily incorporate these motion models as well as constraints on accelerations while minimizing a cost function, e.g. minimizing jerk, energy consumption, or combinations of other goal functions. While computationally demanding, MPC has risen in popularity in recent years, enabled by increasing processing power and efficient solvers for the given optimization problems.

Traditional solvers for numerical optimization include Interior Point (IP) methods and Sequential Quadratic Programming (SQP), however, these require costly operations and are not computationally feasible for large problem spaces or if there are real-time requirements. A new method, namely the Proximal Averaged Newton-type method for Optimal Control (PANOC) was proposed by [9] and shows computation times multiple orders faster than IP and SQP in many applications. In [10] and [11], these three 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 [11] or [12].

When using NMPC for obstacle avoidance, one of the design choices is how to model the obstacles. In [12] a novel framework was introduced for modeling generic, possibly non-convex, shapes of obstacles by describing them as a set of non-linear inequalities. In previous approaches to obstacle avoidance for motion planning, many rely on modeling obstacles as spheres and ellipsoids [13], or handle only convex obstacles [14], thus limiting what obstacles can be expressed. While allowing general obstacle shapes, the novel framework used in [12] comes with its own shortcomings. As described by [11], it has the risk of getting stuck in local minima. To avoid this, [11] combined the modeling approach with several heuristics such as using graph search to guide the solution out from local minima.

While all the approaches above could yield collision-free trajectories for an ATR, they are only able to do so for relatively small distances, a few meters at most. In order to generate trajectories over an entire factory, we propose a novel approach leveraging the global optimality and computational efficiency of the A*-algorithm with the constraints handling of NMPC by 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. The proposed algorithm is, to the best of our knowledge, the first to generate collision-free trajectories using NMPC for distances larger than a few meters while being able to handle arbitrary shapes of obstacles. In the following sections, the algorithm is described in more detail.

II. Method

This section presents the different building blocks of the proposed solution. First, an overview of the complete algorithm is presented, followed by an in-depth review of the NMPC formulation.

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 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 obstacle polygons P, 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 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 position. The waypoints found when running the A* algorithm on the visibility graph then consist of $\mathbf{x}_{init}$, $\mathbf{x}_{end}$ 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 $\mathbf{o} = [x_{obs}, y_{obs}]^T$ in $\mathcal{O}$, e.g. in Fig. 2 this corresponds to $\mathcal{O} = \{[3, 7]^T\}$. Further, a sequence L with line segments of equal length is extracted from the waypoints.

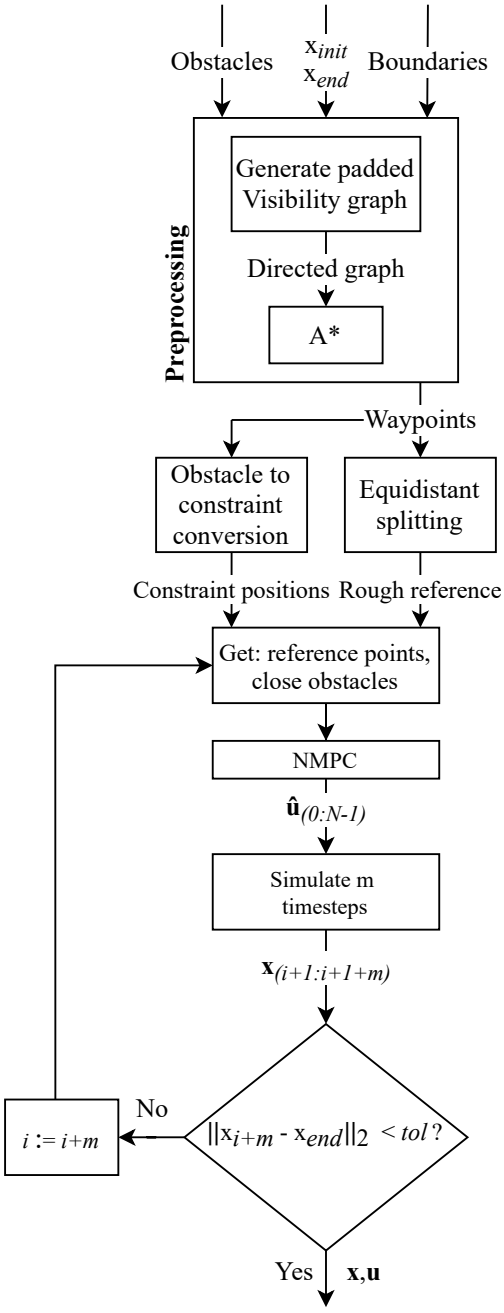

Fig. 1. Flowchart of the proposed algorithm.

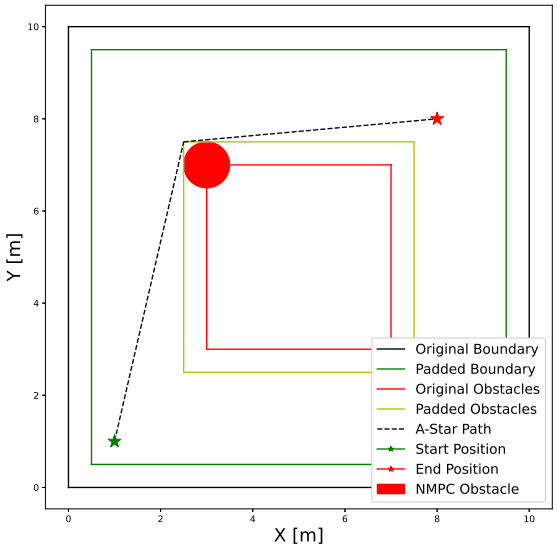

Fig. 2. Example environment displaying padding of boundary and obstacle, A* path and obstacle modelling in NMPC.

Once the initial processing is finished, the starting state is appended to the trajectory $\mathcal{X}$. Next, the NMPC solver is called in an iterative manner. Let N denote the horizon used in the NMPC and let n_{obs} be the maximum number of 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} closest points in $\mathcal{O}$. Given these inputs, the NMPC solver is used to find a sequence $\mathcal{U}$ of feasible actions $\mathbf{u}$ 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 A* path, keep at least a distance of r to the selected n_{obs} obstacle points in $\mathcal{O}$ and respect vehicle dynamics and constraints. From the sequence $\mathcal{U}$, the first m actions are used to calculate m new states to append to the trajectory $\mathcal{X}$. If the last state is not close to the target state the next iteration is 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. We claim that this is possible which will be empirically shown in the evaluation section.

B. NMPC Formulation

The dynamics of the ATR is modeled using a discrete motion model for a differential drive robot as in (1), where i denotes the current time step and T_s denotes the step size . The states in the model are the 2D coordinates x, y, and the heading θ. The inputs to the system are velocity v and angular velocity ω. 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 ATR use-case.

$$\mathbf{x}_i = \begin{bmatrix} x_i \\ y_i \\ \theta_i \end{bmatrix}, \mathbf{u}_i = \begin{bmatrix} v_i \\ \omega_i \end{bmatrix} \tag{1a}$$

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

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, while the states and control actions that define the generated trajectory are denoted $\mathbf{x}$ and $\mathbf{u}$ respectively. The selected cost function consists of multiple parts,

$$J_{\text{cte},j} = d(\hat{\mathbf{x}}_j, \mathbf{l}_{\text{A}*})\, Q_{cte}\, d(\hat{\mathbf{x}}_j, \mathbf{l}_{\text{A}*}) \tag{2a}$$

$$J_{v,j} = (\hat{v}_j - v_{r,j})\, R_v\, (\hat{v}_j - v_{r,j}) \tag{2b}$$

$$J_{\text{acc},j} = (\hat{\mathbf{u}}_{j+1} - \hat{\mathbf{u}}_j)^T\, R_{\text{d}}\, (\hat{\mathbf{u}}_{j+1} - \hat{\mathbf{u}}_j) \tag{2c}$$

where J_{cte} is the cross-track error (CTE) cost. For a state $\hat{\mathbf{x}}_i$ and a set of line segments $\mathbf{l}_{\text{A}*}$, $d(\hat{\mathbf{x}}_i, \mathbf{l}_{\text{A}*})$ returns the minimum distance between the state and any of the line segments. J_v is a cost for difference between selected velocity $\hat{v}_i$ and the reference velocity $v_{r,i}$ 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. All Q and R are tuning parameters that can be modified for desired controller behaviour. Tuning of these and their intuition will be explained in the following section. These cost, constraints from obstacles and on dynamics make up the optimization problem in (3).

$$\min_{\hat{\mathbf{u}}_{(0:N-1)}} \sum_{j=0}^{N-1} J_{\text{cte},j} + J_{v,j} + J_{\text{acc},j} \tag{3a}$$

$$\text{s.t.: } \hat{\mathbf{x}}_0 = \mathbf{x}_i \tag{3b}$$

$$||[\hat{x}_j, \hat{y}_j]^T - \mathbf{o}||_2 \geq r,\ \forall \mathbf{o} \in \mathcal{O}_l, j = 0, \ldots, N \tag{3c}$$

$$\hat{\mathbf{x}}_{j+1} = f(\hat{\mathbf{x}}_j, \hat{\mathbf{u}}_j) \tag{3d}$$

$$\hat{\mathbf{u}}_j \in [\mathbf{u}_{min}, \mathbf{u}_{max}] \tag{3e}$$

$$\frac{\Delta \hat{\mathbf{u}}}{T_s} \in [\dot{\mathbf{u}}_{min}, \dot{\mathbf{u}}_{max}] \tag{3f}$$

Here $\mathcal{O}_l$ is a subset of all obstacles $\mathcal{O}$, specifically, the n_{obs} 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 $\Delta \hat{\mathbf{u}} = \hat{\mathbf{u}}_{j+1} - \hat{\mathbf{u}}_j$ is also applied to $\hat{\mathbf{u}}_0 - \mathbf{u}_{i-1}$, again to ensure coherence between iterations.

The optimization problem in (3) is solved using the PANOC optimizer, this yields a sequence of control actions. A segment of this sequence is supplied to the motion model and the ATRs movements are simulated m time steps ahead, with m ranging from 1 to N see (4).

$$\mathbf{x}_{(i+1:i+m)} := \hat{\mathbf{x}}_{(1:m)} \tag{4}$$

The intuition behind choosing m is that fewer steps will likely result in a somewhat better trajectory whereas a larger amount of steps will reduce the computation time. After these steps are taken the process will be iterated until the final state is reached as is shown in the flowchart in Fig. 1.

C. Dynamic obstacles

While this paper is concerned with static obstacles only, the NMPC formulation above can easily be modified to incorporate simple dynamic obstacles as well. Rather than using static positions $\mathbf{o}$ in (3c), one has to feed the predicted obstacle location at each time instance over the NMPC horizon, i.e. each obstacle is described as a sequence of positions $[\mathbf{o}_0, \ldots, \mathbf{o}_{N-1}]$. Further, for modeling uncertainty in the positional predictions, one can potentially adjust the required distance r with time, e.g. increasing the radius for time steps far ahead.

III. Evaluation

The proposed solution was evaluated via simulation and throughout the evaluation a differential drive motion model, as described in (1), was used to simulate the movement of the vehicle. The algorithm was tested on multiple constellations of layouts and obstacle configurations to ensure robustness. Furthermore, several different tuning profiles were investigated to explore the influence of their parameters. In Fig. 3 a successful trajectory generation is shown for one combination of tuning and obstacle configuration.

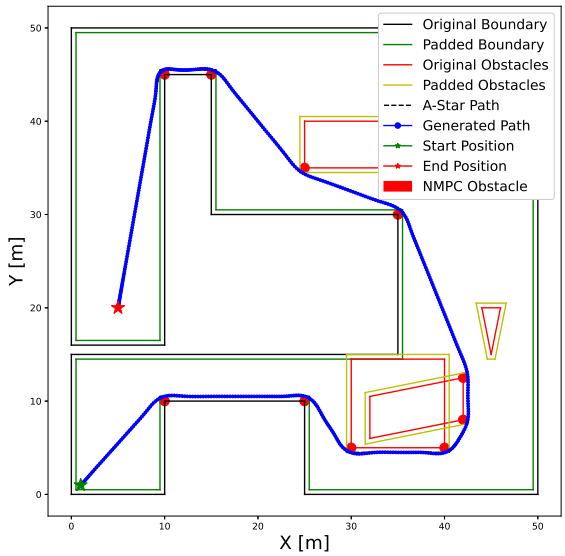

Fig. 3. One specific test-case and tuning profile. A collision-free reference trajectory through an environment containing polygon-shaped environment boundary and obstacles .

In the example shown in Fig. 3 the tuning profile consisted of the following values. The cost for velocity deviation from the reference speed R_v was set to $R_v = 10.0$ and the cost for linear and angular 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 an ATR, which is shown in (5).

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

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

The linear and angular velocity profiles from the simulation shown in Fig. 3 are shown in Fig. 4. Clearly, the NMPC-controller decreases the linear velocity as it increases the angular velocity, i.e. it slows down when taking sharp corners to avoid deviating too much from the reference path. However, it should be noted that this phenomenon is heavily dependent on the specific tuning profile and that the effects of different tuning profiles will be discussed further.

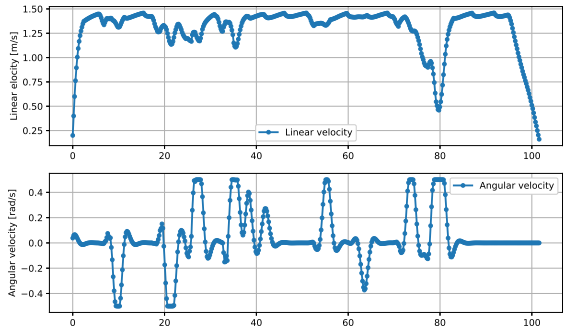

Fig. 4. Linear and angular velocity profile for the simulation shown in Fig. 3

.

Several challenging test-cases were generated and the most relevant sections from a selection of these are shown in Fig. 5. Here, the same tuning profile as in Fig. 3 was used. From the figure it is evident that

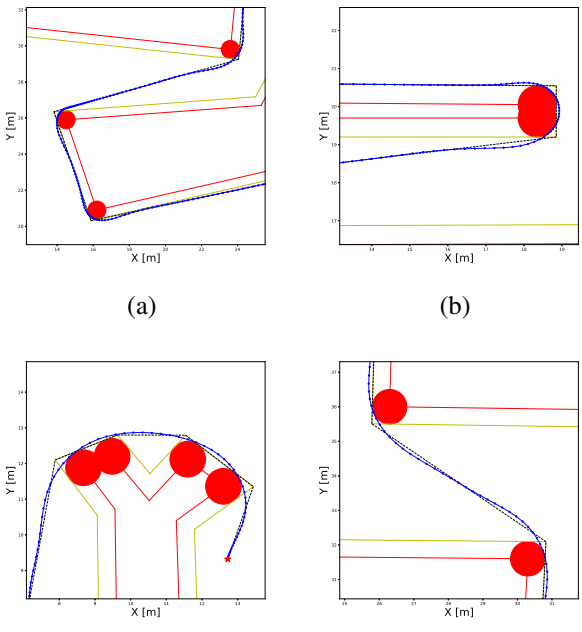

Fig. 5. Interesting sections of different reference trajectories. (a) - (d) show trajectories that have been generated using the same tuning profile as in Fig. 3.

the proposed approach computes a smooth reference trajectory, even around very sharp corners, that satisfies the non-holonomic motion constraints of the differential drive robot while keeping the deviation from the initial reference path small.

As mentioned previously, the behavior of the proposed solution is heavily dependent on its tuning profile. The tuning profile consists of a set of design choices, such as the length of the receding horizon in the NMPC-controller, and a set of cost parameters that determine how the different terms in the NMPC objective function are weighted relative to each other. The relative nature of the cost parameters makes them inherently hard to tune. Furthermore, as the optimal performance is based on the specific use-case, one can not find a set of cost-parameters that are optimal for all cases. However, the intuition regarding the parameters of the tuning profile is listed below.

R_v*:* Increasing the cost of the velocity deviation R_v incentivizes the solution to always stay close to the target velocity. As traversing at the target speed is not always viable, e.g. in sharp corners, setting this parameter too large can yield sub-optimal behavior, most commonly that the vehicle strays too far away from the reference path. However, when the parameter is set too low, there is not enough incentive to move forward and the vehicle will simply stand still or move very slowly.

R_d^{ang}*:* By penalizing the angular accelerations harder, i.e. giving R_d^{ang} a larger value, the solution will yield smoother turning behavior. However, when set too large, it is not allowed to turn as quickly and strays away from the reference path during sharp corners.

R_d^{lin}*:* Increasing the cost of linear acceleration, R_d^{lin} makes it hard to follow sharp corners. This incentivizes the solution to have a low linear acceleration, which does not allow it to slow down quickly enough when a sharp corner is coming up. This, again, causes the solution to stray away from the reference path that it is intended to follow.

Q_{cte}*:* Increasing the cross-track-error cost forces the solution to strictly follow the initial reference path. This causes the solution to sometimes have very high linear and angular accelerations because it has to perform swift maneuvers to maintain a close distance to the non-smooth reference path.

N*:* The negative consequences obtained from increasing the costs of linear and angular accelerations can be countered by increasing the length of the receding horizon, N. This will allow the algorithm to simulate further into the future and therefore make a better decision at the current time. However, as one increases the receding horizon length one also increases the size of the problem, causing longer runtimes which may or may not be of concern depending on the specific use-case and available hardware.

While the algorithm can handle a variety of scenarios as shown in Fig. 5, there are certain cases where the NMPC solver generates trajectories that intersect the padded obstacles. In Fig. 6, 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 A* path. While the trajectory intersects the padded obstacle, the safety margin used when padding results in the trajectory to be collision-free. For the set robot width of 0.25 meters, 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 of the proposed algorithm.

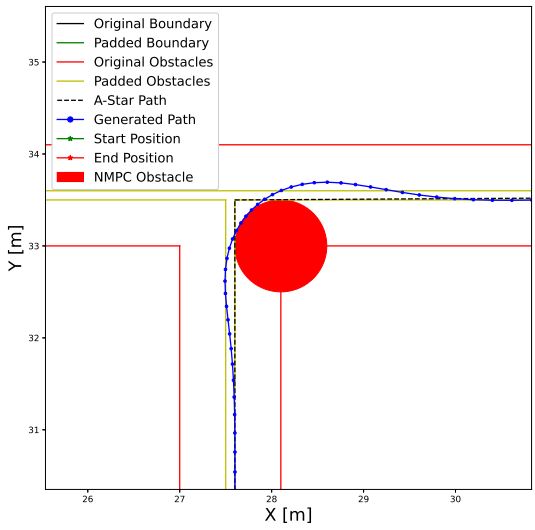

Fig. 6. Scenario for which the NMPC solver generated a trajectory intersecting with a padded obstacle. However, the safety margin ensures that the robot keeps sufficient distance to the original obstacle. The trajectory is going from the bottom to the right side.

To ensure a computationally tractable solution, the runtime of the algorithm has been empirically analyzed by running the algorithm on a large set of test-cases with variable complexity using a 3.59 GHz AMD Ryzen 7 3700X 8-core CPU. Specifically, using the same tuning-profile as in Fig. 3, the algorithm was run on 11 test-cases and for each case, every iteration was timed. In Fig. 7 the total runtime per iteration for all test-cases is shown as a histogram. The mean runtime per iteration was 3.5 ms and as evident from the figure, one iteration of the algorithm rarely exceeds 10 ms. To put this in perspective, generating the trajectory shown in Fig. 3 took 1.52 seconds.

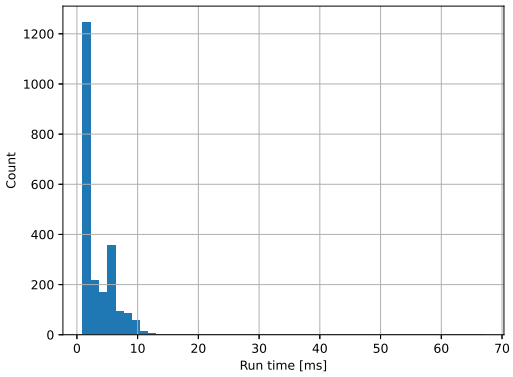

Fig. 7. Runtime per iteration for the proposed solution. Evaluated on a 3.59 GHz AMD Ryzen 7 3700X 8-core CPU.

The proposed approach to incorporate dynamic obstacles by feeding obstacle positions at each time step was evaluated by adding three moving circles to the test case in Fig. 3. The NMPC controller successfully avoided said obstacles without colliding with any static obstacles. However, further investigations have to be conducted to identify potential fail-modes and to ensure robust performance.

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 [12] 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 modeling with our algorithm for cases where our original approach fails. However, one has to investigate how a more complex obstacle modeling 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. Finally, one might investigate whether the NMPC formulation is able to handle dynamic obstacles as well. As mentioned earlier, the formulation can be adjusted to incorporate moving obstacles, and while initial tests showed promising results no extensive evaluation has been conducted.

References

[1] A. V. Savkin, A. S. Matveev, M. Hoy, and C. Wang, *Safe robot navigation among moving and steady obstacles*. Butterworth-Heinemann, 2015.

[2] C. Goerzen, Z. Kong, and B. Mettler, "A survey of motion planning algorithms from the perspective of autonomous uav guidance," *Journal of Intelligent and Robotic Systems*, vol. 57, no. 1-4, p. 65, 2010.

[3] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[4] A. Stentz, "Optimal and efficient path planning for partially known environments," in *Intelligent unmanned ground vehicles*, Springer, 1997, pp. 203–220.

[5] A. V. Savkin and M. Hoy, "Reactive and the shortest path navigation of a wheeled mobile robot in cluttered environments," *Robotica*, vol. 31, no. 2, p. 323, 2013.

[6] C. Rösmann, W. Feiten, T. Wösch, F. Hoffmann, and T. Bertram, "Trajectory modification considering dynamic constraints of autonomous robots," in *ROBOTIK 2012; 7th German Conference on Robotics*, VDE, 2012, pp. 1–6.

[7] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

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

[9] 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.

[10] M. Gerle and S. Johansson, "Evaluation of optimization algorithms in MPC applications," M.S. thesis, Chalmers University of Technology, Gothenburg, Sweden, 2020.

[11] 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.

[12] 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.

[13] P. Wang and B. Ding, "A synthesis approach of distributed model predictive control for homogeneous multi-agent system with collision avoidance," *International Journal of Control*, vol. 87, no. 1, pp. 52–63, 2014.

[14] T. Mercy, R. Van Parys, and G. Pipeleers, "Spline-based motion planning for autonomous guided vehicles in a dynamic environment," *IEEE Transactions on Control Systems Technology*, vol. 26, no. 6, pp. 2182–2189, 2017.