Designing a Hardware-Agnostic Interface Between Route and Trajectory Planning in Self-Driving Cars

Emiko Soroka[1] and Sanjay Lall[2]

***Abstract*— Software reusability is critical to the rapid development and certification of autonomous vehicles (AVs). However, little attention has been given to designing AV path planners for increased software reusability. In this paper we design a trajectory-optimizing nonlinear MPC planner that takes a driveable corridor, desired speed profile and constraints as input to compute the AV's trajectory. These inputs are hardware-independent, confining hardware dependencies to the planner software itself and clearly separating higher-level route planning from hardware-dependent algorithms. We implement the planner in simulation to demonstrate its feasibility in representing common traffic scenarios.**

I. INTRODUCTION

Advanced driver assistance systems (ADAS) and autonomous vehicles are a rapidly growing segment of the automotive industry. However, developing an autonomy stack for a single vehicle is only the first step in bringing autonomous vehicles (AVs) to market. After developing a prototype AV, manufacturers must adapt their stack to vehicles with different capabilities. Reusable software allows AV manufacturers to rapidly enter all segments of the vehicle market.

Autonomous route planning, object recognition, and other high-level decision-making processes are hardware-agnostic. However, steering and acceleration control is tightly coupled to hardware. In emergency situations, knowledge of tire-road interactions, vehicle inertia, center of gravity, and engine performance is critical to averting an accident [1]. Thus, lower level control algorithms are much less portable between different vehicles.

We design a trajectory planner for AV steering and longitudinal acceleration from a software reusability perspective. The proposed planner receives as input a driveable corridor, a desired speed profile and any constraints on its future trajectory: information a higher-level route planning module can provide without hardware-specific knowledge. Thus, hardware dependencies are thus confined to the lower-level planner. This approach improves reusability and decreases software development time.

II. THE AV STEERING AND ACCELERATION CONTROL PROBLEM

A. Configurability

Major concerns with steering and acceleration control for autonomous vehicles include energy efficiency, passenger comfort, and safety. However, different applications of autonomy, such as autonomous trucking or passenger taxis, put different weight on these objectives. Additionally, many ADAS-equipped vehicles offer the option of switching between different driving modes, for example 'sport' and 'eco-friendly' modes, which alter performance. This suggests that fully autonomous consumer vehicles will be expected to offer the same type of configurable experience.

B. Software Division at Waypoints

To reduce hardware dependencies, one could seek to simplify the low-level planner. Suppose our high-level software plans driving maneuvers (such as turning, stopping, or maintaining a safe distance from another vehicle) and decomposes these maneuvers into a trajectory of closely spaced waypoints specifying the vehicle's state over time. The low level planner tracks this trajectory. Unfortunately, without hardware knowledge informing the trajectory generation, the vehicle follows a sub-optimal path, limiting its ability to handle dynamic emergency situations. With hardware knowledge, the high-level route planner is tied to a specific AV platform.

C. Software Division at Maneuvers

Alternatively, suppose we divide the software where driving maneuvers are specified. The low-level planner receives data about these maneuvers, including the road boundaries and specific, desired states (such as stopping at a stop sign or maintaining a safe distance behind another car). The planner then computes a trajectory using hardware-specific information. For example, a hybrid with regenerative braking can recover electric energy, while a vehicle without this capability may spend more time coasting to a stop and less time actively braking. While this trajectory optimization is coupled to specific AV hardware, the higher-level route planner is not.

D. The Design Task

In this paper we divide hardware-dependent and hardware-independent software at the level of driving maneuvers: the low-level planner receives a driveable corridor, desired speed, and constraints. The planner then computes a trajectory using hardware-specific knowledge. We propose an interface between low level and high level software capable of representing arbitrary driving maneuvers and present an example planner.

[1]S. Lall is Professor of Electrical Engineering at Stanford University, Stanford, CA 94305, USA. lall@stanford.edu

[2]E. Soroka is a PhD student in Aeronautics and Astronautics at Stanford University, Stanford, CA 94305, USA. esoroka@stanford.edu

III. PRIOR WORK

A. AV Software Architecture

Several papers have discussed specific AV architectures: for example an autonomy stack designed for the CARLA Autonomous Driving Challenge [2] or for completing the dynamic maneuvers required of an autonomous racecar [3]. Researchers have also proposed a requirements-driven approach to the AV stack [4] and recently, AUTOSAR released guidance for AV software design [5]. Less work has been done on software reusability; therefore this paper focuses specifically on these challenges in the interface between route planning and trajectory optimization.

B. Classical Control for Trajectory Tracking

Proportial-integral-derivative (PID) controllers have been applied to steering and acceleration control; however, tuning the PID gains remains challenging. Mohajer et al. define a multi-objective problem to evaluate a trajectory for efficiency and comfort, then apply a genetic algorithm to optimize PID gains offline [6]. This significantly reduces path tracking error and smooths control inputs over a baseline of hand-tuned PID gains. The authors use 28 numerical parameters to describe the vehicle and simulate the PID controller during the optimization, suggesting the tuning procedure would need to be rerun for different vehicle configurations. Complex multi-body vehicle models have been used to improve performance in highly nonlinear situations [7]; however, this approach also increases the complexity of tuning the controller to specific AV platforms.

C. MPC Trajectory Tracking

Farag et al. apply model-predictive control (MPC) trajectory tracking to achieve high performance on tracks featuring tight curves and hairpin turns [8]. Notably, this performance is achieved with only two parameters: the distances l_f and l_r from the vehicle's center of mass to its front and rear wheels, respectively. The authors use a simple kinematic bicycle model for the vehicle, noting that this increases the algorithm's portability to different vehicles.

Simpler models appear in many MPC algorithms. Daoud et al. propose a dual-objective nonlinear MPC (NMPC) formulation for trajectory tracking, allowing an electric vehicle to switch between driving modes [9]. They use the kinematic bicycle model with an additional simplification: the center of gravity (CG) is assumed to be at the center of the wheelbase. Only two parameters are needed: electric motor efficiency and wheelbase length l (the front and rear axles are $l/2$ away from the CG).

Finally, Beal et al. develop an MPC tracking controller for stabilizing a vehicle during extreme maneuvers [1]. Their approach uses a more detailed model, including tire behavior, to account for the complex dynamics of these situations.

D. Trajectory Optimization

There are fewer trajectory planners, likely due to the increased difficulty of design. Li et al. apply an inner model control framework to split the task into 2 parts: first, given a safe corridor, a trajectory and optimal control input is computed. An inner model controller tracks this signal to correct disturbances [10]. The authors use a bicycle model with three parameters, plus a linear approximation of tire forces with parameters for front and rear tire stiffness.

More recently, NMPC has been applied to directly compute the vehicle's state and control signals [11]. Micheli et al. use penalties and hard constraints to represent road boundaries, static obstacles, and moving obstacles in the nonlinear optimization problem. Though NMPC is computationally difficult, zero-order optimization approaches are also possible: Arrigoni et al. apply genetic algorithms to solve the difficult nonlinear optimization problem in real time [12]. NMPC trajectory planning has also been demonstrated on real AV hardware [13].

IV. AN ARCHITECTURE FOR AV TRAJECTORY PLANNERS

We propose the following division (Fig. 1), focusing on the Trajectory Planning module. We define the module's inputs and outputs and provide an example planner that implements our interface.

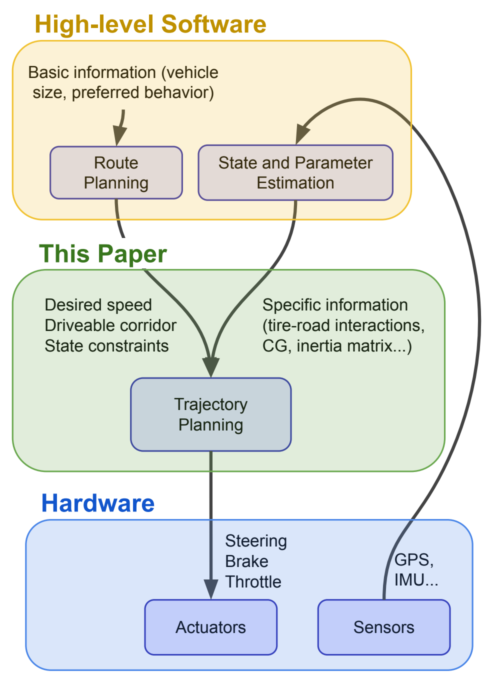

Fig. 1. Block diagram showing how this paper fits into a larger autonomy stack.

The proposed planner is initialized with hardware-specific information. In addition it may query three functions provided by higher-level software, described in Sections IV-A, IV-B, and IV-C.

A. Desired Speed

The desired speed profile is an estimate of the vehicle's future speed provided by high-level software: for example, the speed limit or speed of nearby traffic. The planner is not required to remain at this speed if a minor deviation yields improvement in other objectives.

Desired speed is a function of position and timestep: the vehicle can accelerate according to some profile. Providing both position and time enables the function to represent moving obstacles. The function accepts a point (x, y) in the driveable corridor and a timestep index $k \in 1, \ldots, N$, then returns the desired speed at step k: $v_{des,k}$ (a floating-point value).

$$\texttt{desired\_speed}(x,\ y,\ k) \rightarrow v_{des,k}$$

This can be used to construct a list of desired speeds for N timesteps: $v_{des} = [v_{des,1}, \ldots, v_{des,N}]$.

B. Driveable Corridor

The driveable corridor function takes the vehicle's current position (x, y) and an offset s, returning a new center point (x_c, y_c) and angle ψ_c of the road centerline a distance s away from (x, y). Additionally, it returns the distances to the left and right boundaries d_l and d_r (measured perpendicular to the center line) at (x_c, y_c) (Fig. 2).

$$\texttt{driveable\_corridor}(x,\ y,\ s) \rightarrow (x_c, y_c, \psi_c, d_l, d_r)$$

The center point is not necessarily the geometric center of the corridor; providing two boundaries allows turnouts and wide shoulders to be represented (Fig. 3). Additionally, the driveable corridor is not necessarily the entire road. If an obstacle encroaches on the road, the corridor will be reduced. Finally, the corridor representation fails to account for obstacles with free space on both sides. This is addressed in Section IV-C.

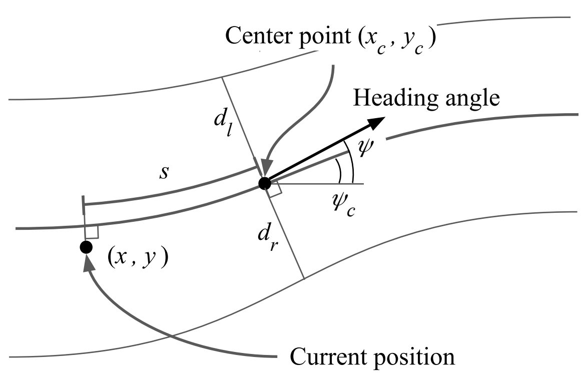

Fig. 2. The driveable corridor function accepts the current position (x, y) and offset s, then returns a new position (x_c, y_c) and distances d_l and d_r a distance s away from the current position.

C. State Constraints

State constraints allow high-level software to ensure a safety or legal requirement is met. For example, a constraint could ensure the vehicle has 0 velocity at an $x-y$ position corresponding to a stop sign. A time-dependent constraint could limit position to ensure a minimum following distance behind another vehicle. Finally, a constraint could keep the vehicle away from an obstruction in the driveable corridor. This accounts for the situation mentioned in Section (IV-B).

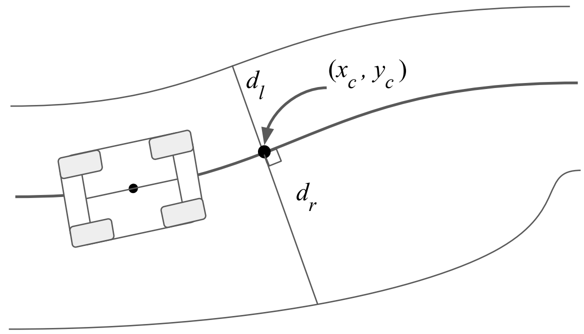

Fig. 3. This road has a wide shoulder. Using a left and right distance allows the centerline to represent the "center" a reasonable human driver would follow.

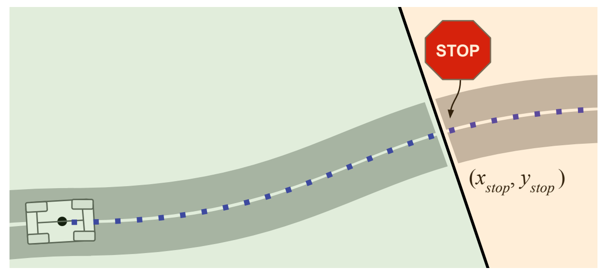

Fig. 4. When stopping, the vehicle must stay behind the line. Its position is constrained using a linear inequality to restrict it to the green region.

To accommodate these and other situations, a constraint generator function takes an $x-y$ position, a speed v (which may be the desired or estimated speed), and a timestep $k = 1, \ldots, N$. It returns a function g representing a vector of inequality constraints on x_k, y_k, v_k: the vehicle's position and speed at timestep k. These constraints may be nonlinear and nonconvex (Fig. 6).

$$\texttt{constraint\_generator}(z_k,\ k)\ \texttt{->} g(\cdot, \cdot, \cdot)$$

The constraints are satisfied at step k if $g(z_k) \leq 0$ (where $z_k = \begin{bmatrix} x_k & y_k & \psi_k & v_k \end{bmatrix}$ is the state vector).

V. IMPLEMENTATION

We implemented an NMPC planner using this interface. To model the vehicle, we used a kinematic bicycle model (1), also used in [8] and shown in Fig. 7. There are two system parameters: l_f, the distance from the center of mass (CoM) to the front axle, and l_r, the distance from the CoM to the rear axle. We used $l_r = 2.10$ m and $l_f = 2.67$ m for all simulations: the same values as [8].

$$z = \begin{bmatrix} x \\ y \\ v \\ \psi \end{bmatrix}, \quad \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{v} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} v\cos(\psi + \beta) \\ v\sin(\psi + \beta) \\ a \\ \frac{v}{l_r}\sin(\beta) \end{bmatrix} \tag{1}$$

$$\beta = \tan^{-1}\left(\frac{l_r}{l_f + l_r}\tan(\delta_f)\right) \tag{2}$$

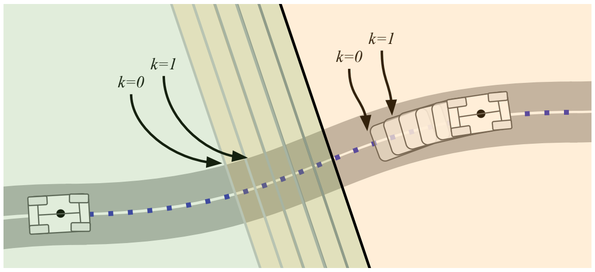

Fig. 5. The ego vehicle is following another car. The linear inequality moves at each timestep $k = 1, \ldots, N$. to enforce a minimum safe distance.

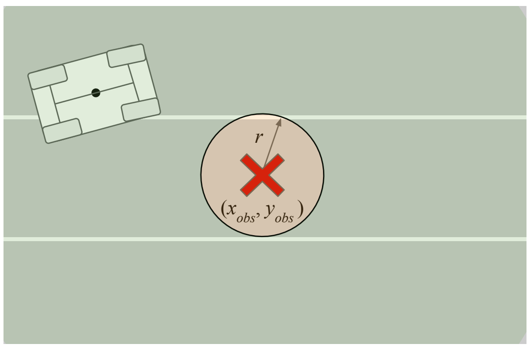

Fig. 6. An obstacle (X) obstructs one lane, creating a circular keep-out region of radius r. The vehicle must stay out of the circle, creating a nonconvex constraint $(x - x_{obs})^2 + (y - y_{obs})^2 \geq r$

The control signals are a, the longitudinal acceleration of the car, and δ_f, the steering angle of its front wheels. The control vector is $u = [a \ \delta_f] \ \in \mathbb{R}^2$.

We consider an NMPC problem with lookahead steps $k = 1, \ldots, N$ with $N = 30$ steps spaced a distance $\Delta_t = 0.075$ s apart. The nonlinearity is confined to the dynamics model (1) and constraints (Fig. 6). The cost function is quadratic.

In our implementation, we used the driveable corridor function to generate a list of linear inequalities representing the road as shown in Fig. 8. Each right and left boundary constrains the vehicle's $x - y$ position at one step: for N lookahead steps, $2N$ lines are generated. Arbitrarily tight curvatures can be represented by decreasing the time between each step.

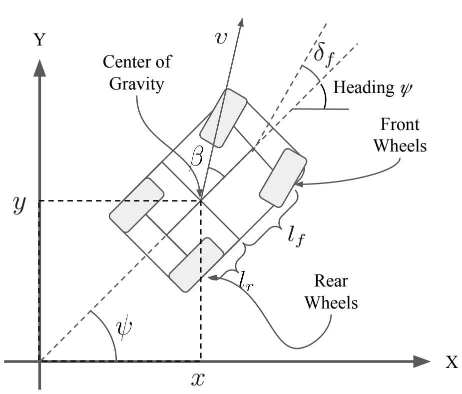

Fig. 7. Kinematic bicycle model.

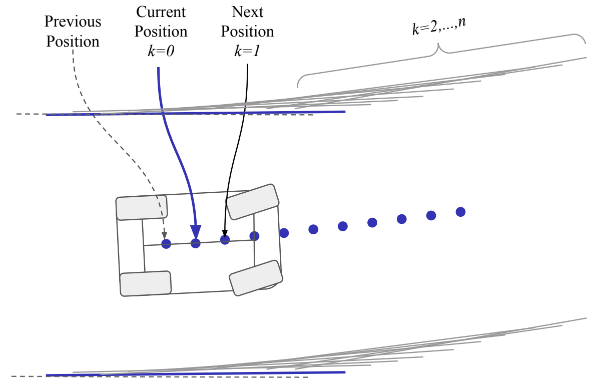

Fig. 8. A short sample of overlapping lines to define the road corridor. 2 lines are used to constrain the vehicle's position at eack step k.

A. Cost function

Our multi-objective cost function penalizes sharp accelerations which degrade passenger comfort and deviations from the desired speed and center of the driveable corridor. We use 2 terms to control this accuracy:

$$J_{\text{position}} = \sum_{k=1}^{N} \left\| \begin{bmatrix} x_k \\ y_k \end{bmatrix} - \begin{bmatrix} x_{\text{center},k} \\ y_{\text{center},k} \end{bmatrix} \right\|_2^2 \tag{3}$$

$$J_{\text{angle}} = \sum_{k=1}^{N} (\psi_k - \psi_{\text{center},k})^2 \tag{4}$$

$$J_{\text{speed}} = \sum_{k=1}^{N} (v_k - v_{\text{des},k})^2 \tag{5}$$

where $(x_{\text{center},k}, \ y_{\text{center},k}, \ \psi_{\text{center},k})$ is returned by `driveable_corridor` and $v_{\text{des},k}$ is returned by `desired_speed`. These two functions are provided by the higher-level software (Fig. 1).

Equations (3) and (4) controls path-following accuracy, while (5) controls how closely the vehicle stays at the desired speed. Because the desired speed could be small or large, (3), (4), and (5) are separated so they can be weighted differently.

Equations (6) and (7) penalize undesirable sharp changes in acceleration (m/s^2) and steering angle (deg).

$$J_{\text{jerk}} = \sum_{k=2}^{N} (a_k - a_{k-1})^2 \tag{6}$$

$$J_{\text{accel}} = \sum_{k=2}^{N} (\delta_{f,k} - \delta_{f,k-1})^2 \tag{7}$$

By separating these terms, they can be weighted to produce different behaviors (8).

$$J = a_1 J_{\text{position}} + J_{\text{angle}} + a_2 J_{\text{speed}} + a_3 J_{\text{jerk}} + a_4 J_{\text{steering}} \tag{8}$$

We use the notation z to denote a list of states $z_1, \ldots, z_N$

and controls $u = u_1, \ldots, u_N$. The final NMPC problem is:

$$\begin{aligned}
\text{minimize} \quad & J(z, u) && (9)\\
\text{subject to} \quad & z_k = f(z_{k-1}, u_{k-1}), \;\; k = 1, \ldots, N && (10)\\
& u_{\min} \leq u \leq u_{\max} && (11)\\
& v_{\min} \leq v \leq v_{\max} && (12)\\
& \psi_{\min} \leq \psi \leq \psi_{\max} && (13)\\
& A_k \begin{bmatrix} x_k \\ y_k \end{bmatrix} \geq 0, \;\; k = 1, \ldots, N && (14)\\
& g(x_k, y_k, \psi_k, v_k) \leq 0, \;\; k = 1, \ldots, N && (15)
\end{aligned}$$

where A_k is a matrix of linear inequalities constructed from `driveable_corridor` and `desired_speed` describing the corridor boundaries at the kth step.

B. Implementation Details

The $x - y$ position of the vehicle is constrained by the linear inequalities, so there is no need to impose any other condition on it. The velocity was constrained to $0 \leq v \leq 50$ m/s. The control signals were limited to $-5 \leq a \leq 2.5$ and $-\pi/4 \leq \delta_f \leq \pi/4$.

The problem was defined using CasADi in Python and solved with ipopt [14] [15]. Ipopt requires initialization when solving a nonlinear optimization problem: an estimated speed and position must be provided. Using the solution of the kth problem as the initial guess for the $k+1$th NMPC problem provided a significant reduction in computation time.

We provide an outline of this implementation:

Listing 1. Nonlinear MPC trajectory planner using proposed API.

```python
class TrajectoryPlanner():
    def run(self, initial_state:np.array,
                  driveable_corridor   :callable,
                  desired_speed        :callable,
                  constraint_generator:callable):
    # See section IV for definitions of these
        callables
    self.z0 = initial_state
    self.initialize_first_mpc_problem(
        driveable_corridor, desired_speed,
        constraint_generator)

    while True:
        # Construct the MPC problem
        problem = self.build_mpc_problem(
            driveable_corridor, desired_speed,
            constraint_generator)
        # Compute the trajectory z and control u:
            a list of states and control signals
            from 1,..N
        z, u    = self.solve_mpc_problem(problem)
        # Move forward one timestep
        self.z0 = self.apply_control(u[0])
        # Initialize next problem with previous
            result
        self.initialize_nth_mpc_problem(z, u)
```

VI. Results

We tested the planner on three scenarios. In the double-lane-change simulation, the vehicle is driving at 10 m/s (36 km/h) and must navigate the ISO double lane-change path. Positional accuracy (3), (4) and steering smoothness (7) contribute to the cost function.

The stop sign scenario consists of a straight road with a stop sign. The vehicle is driving at 4 m/s (14.4 km/h), detects the stop sign from 10 m away, and must smoothly come to a stop.

Finally, the vehicle following scenario consists of a straight road with another vehicle. The ego vehicle is initially driving at 4 m/s but must slow down to maintain a minimum safe distance behind a slower driver.

A. Double lane change

In this scenario, we tested a range of weights for the cost function (8). Accuracy terms (3) and (5) were grouped. Jerk (6) and steering change terms (7) were also grouped.

Initial tuning was performed to keep the grouped terms within an order of magnitude of each other, resulting in a cost function with one adjustable weight a:

$$J = \Big(J_{\text{position}} + J_{\text{angle}} + 10^3 J_{\text{speed}}\Big) + a\Big(10 J_{\text{jerk}} + J_{\text{steering}}\Big),$$

A large value of a corresponds to a high weight on comfort (smaller and smoother control signals) while a small value corresponds to a high weight on position and speed accuracy. Results are provided for several weights of a to illustrate this tradeoff. In practice the weights would be application-dependent or adjusted on the fly as discussed in [11].

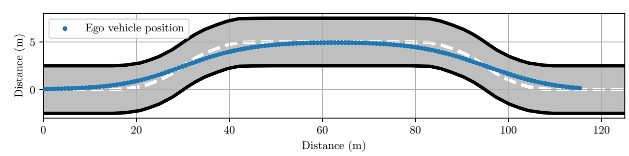

Fig. 9. The double-lane-change scenario. The road width is 5 m and the desired speed is 10 m/s. To minimize jerk and sharp changes in steering angle, the vehicle deviates from the road centerline.

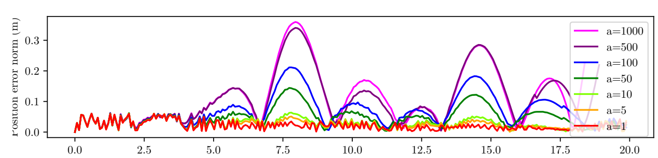

Fig. 10. Position error from the road centerline (meters) in the double-lane-change scenario with different MPC weights.

As expected, the position error is low for high accuracy runs and increases for high comfort runs (Fig. 10).

B. Stop sign

In this test, the road is straight: the car never deviates from the road centerline so there is no position error as defined in Fig. 10. Similarly, the steering signal remains at 0. The goal is to minimize acceleration and jerk.

The desired velocity profile was a rough estimate: the velocity was set at 4 m/s until the stop sign is detected at 10 m/s away; after that it decreases linearly (with distance) to 0. Despite this discontinuity, the NMPC planner produces a smooth velocity profile (Fig. 12).

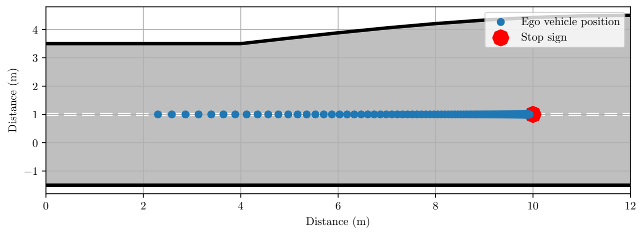

Fig. 11. Using a hard constraint to enforce 0 velocity at a stop sign.

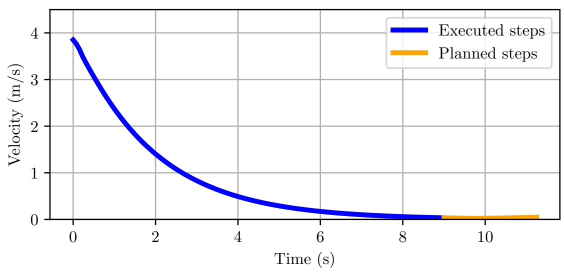

Fig. 12. Velocity in stop sign test vs time. The desired speed was 4 m/s.

C. Following another vehicle

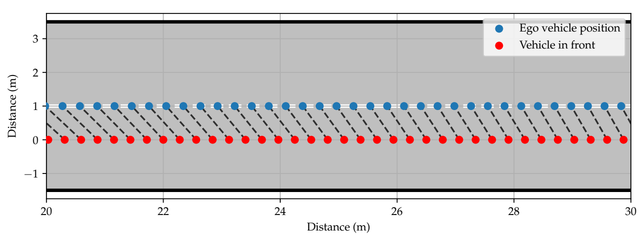

Fig. 13. Following a slower car. The desired speed is 4 m/s but the car ahead is traveling at 3.75 m/s. The white lines connect the two vehicle's positions at each time step, showing how a safe distance is maintained.

The hard constraint causes the vehicle's speed to decrease, maintaining a safe distance from a slower car (Fig. 13).

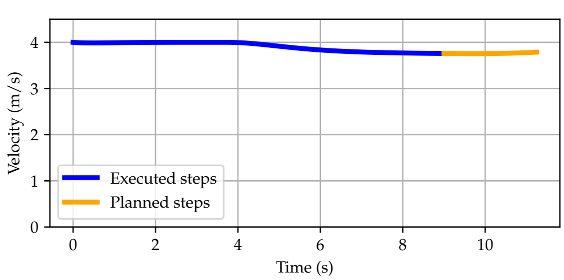

Fig. 14. The ego vehicle decelerates to maintain a safe distance from another car.

VII. CONCLUSIONS

Our proposed planner requires as input a driveable corridor, constraints imposed by obstacles and rules of the road, and a desired speed: information that can be provided by higher-level software without knowledge of vehicle hardware. Consequently, the higher-level software can be certified once and installed on different AV platforms. This input interface, designed to increase software reusability, is our main contribution. We demonstrate an NMPC implementation of this planner which can be ported to different vehicles by changing vehicle-specific parameters. The planner is of similar design to recent work in NMPC steering and acceleration control [11] [13], suggesting the relevance of this software reusability-focused design to current AV research.

VIII. ACKNOWLEDGMENTS

The NMPC code was based on several open-source files provided with CasADi. Code for this paper is published at `https://github.com/elsoroka/AutonomousCarMPC/tree/master/code_for_paper`

REFERENCES

[1] C. Beal and Gerdes J. Model predictive control for vehicle stabilization at the limits of handling. *IEEE Transactions on Control Systems Technology*, 21(4):1258–1269, 2013.

[2] Luis Alberto Rosero, Iago Pachêco Gomes, Júnior Anderson Rodrigues da Silva, Tiago Cesar dos Santos, Angelica Tiemi Mizuno Nakamura, Jean Amaro, Denis Fernando Wolf, and Fernando Santos Osório. A software architecture for autonomous vehicles: Team LRM-B entry in the first CARLA autonomous driving challenge. *CoRR*, abs/2010.12598, 2020.

[3] Johannes Betz, Alexander Wischnewski, Alexander Heilmeier, Felix Nobis, Leonhard Hermansdorfer, Tim Stahl, Thomas Herrmann, and Markus Lienkamp. A software architecture for the dynamic path planning of an autonomous racecar at the limits of handling. In *2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE)*, pages 1–8. IEEE, 2019.

[4] Alexandru Constantin Serban, Erik Poll, and Joost Visser. A standard driven software architecture for fully autonomous vehicles. In *2018 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 120–127, 2018.

[5] S. Fürst and M. Bechter. Autosar for connected and autonomous vehicles: The autosar adaptive platform. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*, pages 215–217, 2016.

[6] N. Mohajer, S. Nahavandi, H. Abdi, and Z. Najdovski. Enhancing passenger comfort in autonomous vehicles through vehicle handling analysis and optimization. *IEEE Intelligent Transportation Systems Magazine*, pages 0–0, 2020.

[7] A. Chebly, R. Talj, and A. Charara. Coupled longitudinal and lateral control for an autonomous vehicle dynamics modeled using a robotics formalism. *IFAC-PapersOnLine*, 50(1):12526–12532, 2017. 20th IFAC World Congress.

[8] W. Farag. Complex-track following in real-time using model-based predictive control. *International Journal of Intelligent Transportation Systems Research*, 2020.

[9] M. A. Daoud, M. Osman, M. W. Mehrez, and W. W. Melek. Path-following and adjustable driving behavior of autonomous vehicles using dual-objective nonlinear MPC. In *2019 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pages 1–6, 2019.

[10] X. Li, Z. Sun, Q. Chen, and J. Wang. A novel path tracking controller for ackerman steering vehicles. In *Proceedings of the 32nd Chinese Control Conference*, pages 4177–4182, 2013.

[11] F. Micheli, M. Bersani, S. Arrigoni, F. Braghin, and F. Cheli. NMPC trajectory planner for urban autonomous driving. *CoRR*, abs/2105.04034, 2021.

[12] S. Arrigoni, F. Braghin, and F. Cheli. MPC path-planner for autonomous driving solved by genetic algorithm technique, 2021.

[13] S. Arrigoni, S. Mentasti, F. Cheli, M. Matteucci, and F. Braghin. Design of a prototypical platform for autonomous and connected vehicles. *CoRR*, abs/2106.09307, 2021.

[14] J. Andersson, J. Gillis, G. Horn, J. Rawlings, and M. Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

[15] A. Wächter and L. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106, 2006.