Defining an Interface for Autonomous Vehicle Steering Software

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

***Abstract*—TODO: write abstract.**

***Index Terms*—autonomous driving, steering controller, mpc, optimal control,**

Introduction

Advanced driver assistance systems (ADAS) and autonomous vehicles are a rapidly growing segment of the automotive industry. In 2004, Audi researchers reported that software advances accounted for 80% of innovations in new vehicles [1]. Since then, the prevalence of ADAS and autonomous driving has rapidly increased.

The rise of automotive software presents new challenges in software reusability. To equip more vehicles with autonomous software, manufacturers must be able to develop autonomous driving stacks for vehicles with different capabilities and sensor configurations, verify safety-critical software, and comply with numerous regulations and standards. Software reuse is a key component of this process, enabling rapid development of autonomous driving for different vehicle platforms.

Some portions of the software, such as route planning, object recognition, and other high-level decision-making processes, are hardware-agnostic. However, lower-level software such as steering and acceleration control is more dependent on the vehicle's specific hardware. To optimize fuel efficiency, for example, a gasoline-powered vehicle must be driven differently than an electric-gas hybrid or fully electric vehicle with regenerative braking. Tire-road interactions, vehicle dynamics, engine performance, and other mechanical characteristics inform this optimization. Clearly, this part of the autonomy stack is much less portable between different vehicles.

Steering and Acceleration for Autonomous Vehicles

Major concerns with steering and acceleration control for autonomous vehicles include fuel efficiency, passenger comfort, and safety. However, different applications of autonomy put different weight on these objectives. An autonomous truck delivering cargo would prioritize fuel efficiency, but may not carry any passengers. An autonomous taxi must avoid making maneuvers that would cause discomfort or alarm to customers. Many ADAS-equipped vehicles on the market today offer the option of switching between "sport" and "eco-friendly" modes, which change the driving behavior of the car, raising the possibility that fully autonomous consumer vehicles would offer the same type of configurable experience.

Software Reusability Considerations

Given the hardware dependencies of low-level control software, an initial idea to tackle software reusability may be to minimize the amount of software being written. One could seek to make the division between hardware-agnostic and hardware-specific software as far "down" the stack as possible to maximize software reuse. However, this presents issues in practice. Suppose our high-level software plans driving maneuvers (such as turning, stopping, holding a certain speed or distance from another vehicle) and decomposes these maneuvers into a trajectory of closely spaced waypoints that specify the vehicle's state over time. Then the low level controller must only track this trajectory, assuming it is correct. Unfortunately, without hardware information the vehicle will follow a sub-optimal path. In dynamic maneuvers such as emergency braking, this could become dangerous.

Performance Considerations

A more performant approach is to divide the software when driving maneuvers are specified. The low-level controller would be provided data about these maneuvers, such as the road boundaries and specific, desired states (stopping at a at a stop sign, keeping a safe distance behind another car). The specific trajectory taken to execute these maneuvers is then computed by the low-level controller, using hardware-specific information to optimize the trajectory. 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. A vehicle that is aware its tires could slip can steer more conservatively. However, the optimizing controller is coupled to the hardware. The performance benefit must be weighed against the development cost of these vehicle-specific controllers.

Trajectory Tracking vs Optimization

Are waypoints the best representation of the vehicle's path? A higher-level representation, such as the road boundaries and obstacles, allows the steering controller to determine its own

trajectory but is more difficult to develop and verify. In this paper we explore the question of tracking vs optimization by discussing the implementation of an optimizing controller.

I. Implementations of Steering and Acceleration Controllers

Many steering and acceleration controllers have been developed for varying purposes. We focus on examples of specific categories: classical control, and optimal control (such as model-predictive control (MPC)). Most of these examples are tracking controllers: the objective is to follow a reference trajectory as closely as possible.

A. Classical Control for Trajectory Tracking

PID controllers can be 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 passenger comfort, then apply a genetic algorithm to optimize their controller's PID gains offline [2]. This significantly reduced path tracking error and undesirable high-frequency control inputs over their baseline of hand-tuned PID gains. Information about the vehicle's hardware is required to complete the PID tuning. The authors use 28 numerical parameters describing the vehicle's geometry, inertia matrices, and shock absorber performance to simulate the PID controller during the optimization process. This suggests in a practical application, the tuning procedure would need to be rerun if, for example, the vehicle is loaded with heavy cargo. Alternatively, PID gains could be precomputed for a variety of different configurations.

Classical control has also been applied to more complex vehicle models. Chebly et al. develop a multi-body model of the vehicle using robotics methods, using 21 bodies connected by joints to represent the vehicle's chassis, suspension, steering, and wheels. [3] This model uses 12 parameters including vehicle mass, inertia, and cornering stiffnesses of tires. The high fidelity of this vehicle model enables the design of a steering and acceleration controller that successfully tracks a reference trajectory across a wide range of driving conditions and performs well in highly nonlinear situations [3]. The authors also discusses the controller's robustness to errors caused by variations in the vehicle's mass or tire stiffness, verifying the controller can continue to track the reference trajectory in with a $\pm 30\%$ error in these parameters.

B. MPC Trajectory Tracking

Farag et al. apply MPC for trajectory tracking to achieve high performance driving around "complex" tracks featuring tight curves and hairpin turns [4]. Notably, this performance is achieved with only two parameters listed: the distances l_f and l_r from the vehicle's center of mass to its front and rear wheels, respectively. This is achieved by using a simple kinematic bicycle model for the vehicle. The authors note that this increases the algorithm's portability to different vehicles.

The use of few physical parameters appears in several other MPC algorithms. Daoud et al. develop a dual-objective nonlinear MPC formulation for trajectory tracking, allowing an electric vehicle to switch between "sport" and "eco-friendly" driving modes. This algorithm uses the kinematic bicycle model with an additional simplification: the center of gravity (CG) is assumed to be at the center of the wheelbase, so only one parameter is needed: the wheelbase length l. The front and rear axles are then a distance $l/2$ away from the CG. An additional parameter describes the efficiency of the electric motor.

C. Trajectory Optimization

There are fewer optimizing controllers, likely due to the increased difficulty of design. One paper on this topic applies an inner model control framework to the problem of path tracking: computing an optimal trajectory and controlling the vehicle to follow it [5]. The road is represented as a corridor of allowable space. The authors note that due to the added complexity of accurate nonlinear vehicle models, their controller uses the simplified bicycle model (with three parameters: mass, and the two distances between the CG and front/rear axles). They also use a linear approximation to model tire forces, with two additional parameters for front and rear tire stiffness.

In this algorithm, the path tracking problem is split in two parts. First, given a road corridor where it is safe to drive, a trajectory is selected from the space of possible trajectories in the corridor. An optimal control input is computed. The authors then design an inner model controller driven by this signal to account for disturbances in the vehicle [5].

II. Practical Considerations

Many considerations inform the selection of a vehicle model, including nonlinearity, accuracy in extreme situations such as emergency maneuvers with high steering curvature and acceleration, ease of computation (particularly for optimal controllers such as MPC), and ease of system identification. Additional practical concerns include the controller's robustness against inaccurate or changing parameters.

In real life, disturbances such as passengers, heavy cargo and fuel can perturb the vehicle's weight and CG. Weather and temperature effects can change the tire performance. A practically applicable controller must be robust against inaccuracies in its vehicle model. Alternatively, the controller could implement an online estimation method to update parameters, allowing a high standard of performance to be maintained under changing configurations.

III. Implementation of a Nonlinear MPC Trajectory Optimizer

To study the challenges and benefits associated with optimizing controllers, we implemented one using nonlinear MPC (NMPC). The controller used a simple kinematic bicycle model for the vehicle, shown in Figure 1. This is the same model used in [4] with $n = 4$ state variables, $m = 2$ control signals, and two system parameters: l_f, the distance from the

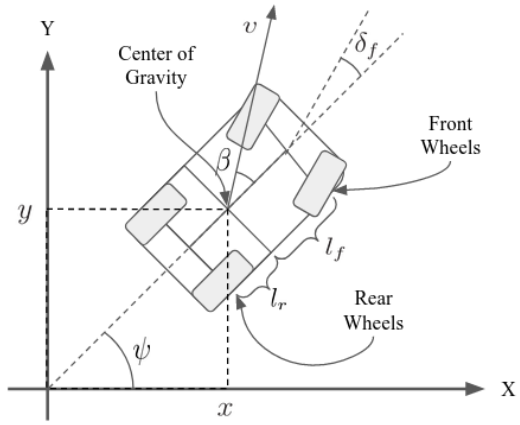

Fig. 1. Kinematic bicycle model.

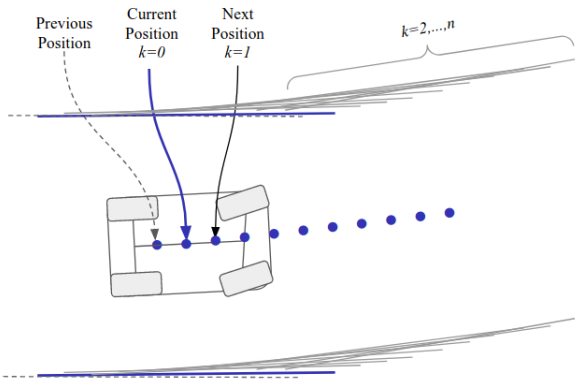

Fig. 2. A short sample of overlapping lines to define the road corridor. 2 lines are used per step: for an MPC problem with steps $x_1, \ldots, x_N$, $2N$ lines are required.

center of mass (CoM) to the front axle, and l_r, the distance from the CoM to the rear axle.

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

where β, the angle between the car velocity and its longitudinal axis, is defined in (2). The state vector is $z \in \mathbb{R}^4$.

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

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 $k = 1, \ldots, N$ spaced a distance Δ_t apart. The nonlinearity is confined to the dynamics model (1). The cost function is quadratic and the state and control constraints are all representable by linear inequalities.

The problem is initialized with the initial vehicle state and a desired speed for each lookahead step $v_{des,1}, \ldots, v_{des,N}$. Obviously, the desired speed should not vary much over the lookahead distance.

Taking inspiration from a paper in which a "driveable corridor" is defined [6], allowing the optimization algorithm to choose the vehicle's path within the corridor, we used a list of linear inequalities to represent the road as shown in Figure 2. Each right and left boundary corresponds to one step: if there are N lookahead steps, $2N$ lines must be generated.

This representation was selected to simplify the optimization problem. At each step, the allowable $x-y$ position of the vehicle can be represented by a set of linear inequalities. By generating a new polygon for each lookahead step, arbitrarily tight curvatures can be represented by decreasing the time between each step. In our implementation, four-sided polygons are used to simplify software development.

The cost function contains terms to penalize sharp turns and acceleration, which degrade passenger comfort, and to penalize deviations from the center of each polygon (otherwise the vehicle may drive close to the boundaries of the road, which is confusing to other drivers and alarming to passengers).

The cost terms are:

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

where $(x_{center,k}, y_{center,k}.\psi_{center,k}$ describe the $x-y$ position and angle of the center of the kth polygon.

$$J_{speed} = \sum_{k=1}^{N} (v_k - v_{des,k})^2 \tag{4}$$

The cost terms in (3) and (4) are separated so they can be weighted differently. Equation (3) controls the accuracy of the path-following behavior, while (4) controls how closely the vehicle stays at the desired speed.

Equations (5) and (6) penalize undesirable sharp changes in acceleration and steering angle.

$$J_{jerk} = \sum_{k=2}^{N} (a_k - a_{k-1})^2 \tag{5}$$

$$J_{steering} = \sum_{k=2}^{N} (\delta_{f,k} - \delta_{f,k-1})^2 \tag{6}$$

By separating these terms, they can be weighted to produce different behaviors as shown in (7).

$$J = a_1 J_{accuracy} + a_2 J_{speed} + a_3 J_{jerk} + a_4 J_{steering} \tag{7}$$

The final NMPC problem is:

$$\begin{aligned}
\text{minimize} \quad & J(z_1, \ldots, z_N, u_1, \ldots, u_N) && (8)\\
\text{subject to} \quad & z_k = f(z_{k-1}, u_{k-1}),\ k = 1, \ldots, N && (9)\\
& u_{min} \leq u_k \leq u_{max},\ k = 1, \ldots, N && (10)\\
& \begin{bmatrix} v_{min} \\ \psi_{min} \end{bmatrix} \leq \begin{bmatrix} v_k \\ \psi_k \end{bmatrix} \leq \begin{bmatrix} v_{max} \\ \psi_{max} \end{bmatrix},\ k = 1, \ldots, N && (11)\\
& A_k \begin{bmatrix} x_k \\ y_k \end{bmatrix} \geq 0,\ k = 1, \ldots, N && (12)
\end{aligned}$$

where A_k is a matrix of linear inequalities that describe the road boundaries at the kth step.

We use the bold $\mathbf{z} = z_1, \ldots, z_N$ and $\mathbf{u} = u_1, \ldots, u_N$ to denote a full trajectory of N steps. The final algorithm is then:

Require: z_0, v_{min}, v_{max}, u_{min}, u_{max},`weights`
{Weights for cost function (7).}
$z \leftarrow z_0$
while true do
$\mathbf{v}_{des} \leftarrow$ `get_desired_speed(`z`)`
$A_1, \ldots, A_N \leftarrow$ `generate_road(`z, $\mathbf{v}_{des}$`)`
`bounds` = $(A_1, \ldots, A_N, v_{min}, v_{max}, u_{min}, u_{max})$
$\mathbf{z}$, $\mathbf{u} \leftarrow$ `solve_nmpc(`z_0`, bounds, weights)`
$z \leftarrow f(z, \mathbf{u}_1)$
end while

A. Implementation Details

The control limits are given in (13). Because the $x - y$ position of the vehicle will be constrained by the linear inequalities, there is no need to impose any other condition on it. The velocity was constrained to $[0, 50.0\ m/s]$.

$$\begin{bmatrix} -5 \\ -\pi/4 \end{bmatrix} \leq \begin{bmatrix} a \\ \delta_f \end{bmatrix} \leq \begin{bmatrix} 2.5 \\ \pi/4 \end{bmatrix} \tag{13}$$

The problem was defined using CasADi in Python and solved with ipopt [7] [8]. Ipopt requires initialization when solving a nonlinear optimization problem: a rough guess of speed and position must be provided. To reduce computation, the initial guess for the $k+1$th NMPC problem is computed using the solution of the kth problem, This provided a significant speedup.

IV. Results

Figure 3 shows the road as described by the polygon boundaries. Though each boundary is rectangular, they overlap to produce an accurate representation of the road.

Figures 5 and 6 show the steering angle signal and position error computed by the nonlinear MPC for several sets of weights given in Table I. In this test the simulated vehicle navigates the ISO double lane-change path (Figure 3). The initial speed provided to the optimizing controller was 10 m/s.

The error can be evaluated several ways: Because the desired velocity was constant ($10m/s$), the velocity error can be seen in Figure 4.

The position error (14) measures the distance of each x-y position (x_k, y_k), $k = 1, \ldots, N$ from the center of its polygonal bound. This is the same measurement used in the cost function (3).

$$\text{Position error} = \left\| \begin{bmatrix} x_k \\ y_k \end{bmatrix} - \begin{bmatrix} x_{center,k} \\ y_{center,k} \end{bmatrix} \right\|_2 \tag{14}$$

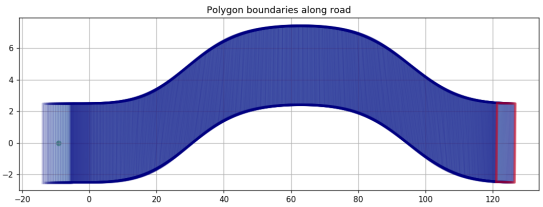

Fig. 3. Polygon boundaries describing the ISO double lane change path. The path has a width of 5 meters.

As expected by the weights in Table I, increasing the cost associated with the jerk (change in acceleration) and change in steering angle produces a smoother velocity profile and smoother control signals.

$$J = \alpha J_{accuracy} + 10\alpha J_{speed} + 100 J_{jerk} + 10\frac{180}{\pi} J_{steering}$$

TABLE I
COST WEIGHTS FOR (7) USED IN THE MPC SIMULATIONS. THE WEIGHTS OF (4) AND (5) ARE INCREASED TO PRODUCE INCREASED SMOOTHING OF THE VEHICLE TRAJECTORY.

Run	1	2	3	4	5	6
α	0.01	0.05	0.1	1.0	5.0	10.0

A. Changing Vehicle Parameters

The kinematic bicycle model used in this paper has 2 parameters, l_r and l_f, describing the vehicle CG with respect

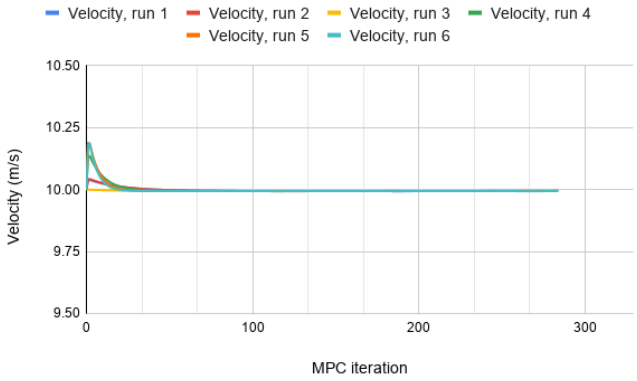

Fig. 4. Velocity profile (desired speed is 10 m/s). There is some initialization difficulty, likely due to the issue shown in Figure 10 when generating the first set of road center points. Once this error is overcome, the vehicle stays almost exactly at the desired speed.

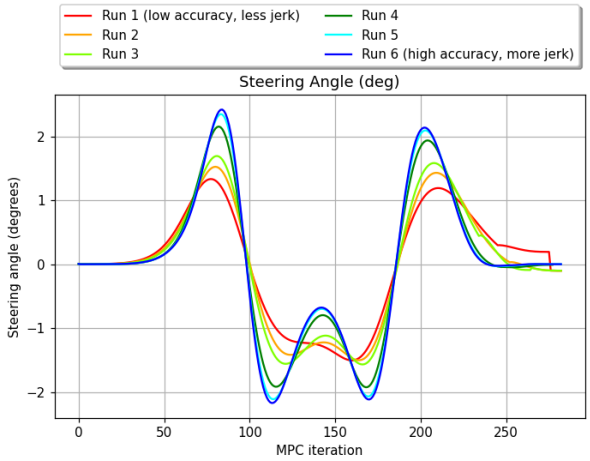

Fig. 5. Steering angle control signal for a low-speed test (desired velocity is 10 m/s).

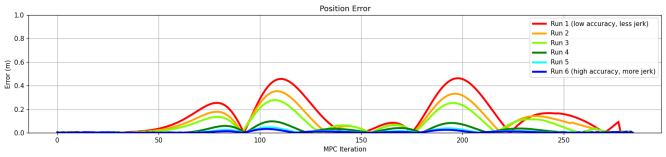

Fig. 6. Position error (14). Runs with less weight on accuracy (3) had higher error but required smaller control inputs as shown in Figure 5.

to the front and rear axles. To study the effects of parameter changes on the controller output, we moved the CG forwards and backwards, as might occur in a vehicle transporting heavy cargo.

All runs were computed with cost weights corresponding to the "mid-range" performance of the previous section:

$$J = 0.1J_{accuracy} + J_{speed} + 100J_{jerk} + 10\frac{180}{\pi}J_{steering}.$$

Ideally, the accuracy and control signals should be similar since the same cost function is being applied.

Figures 7, 8, and 9 show these expected results. With the CG moved forward or back, the vehicle becomes less maneuverable. The speed lowers slightly from the initial estimate of 10 m/s.

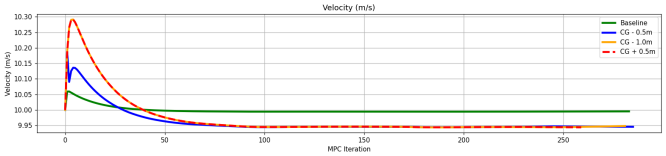

Fig. 7. Velocity profile (desired velocity is 10 m/s). There is some initialization difficulty, likely due to the issue shown in Figure 10 when generating the first set of road center points. The optimizing controller decreases the velocity slightly when the CG moves forwards or back.

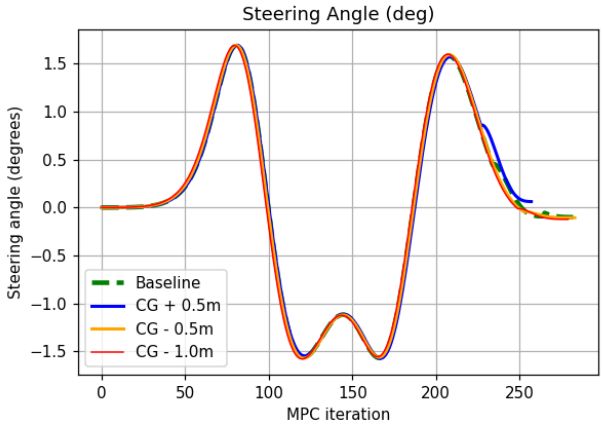

Fig. 8. Steering angle control signal with varying CG. The control signals are very similar, likely because the cost function weights were not changed.

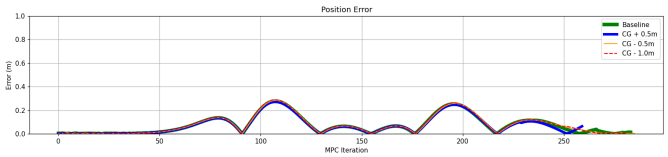

Fig. 9. Position error (14) with varying CG. The error is very similar, likely because the cost function weights were not changed..

B. Implementation Difficulties

The primary implementation difficulties encountered while programming the NMPC controller were related to road representation: the problem of converting numerical data representing the road (such as a list of center points and road widths) to a mathematically useful format. Other papers apply a variety of methods: fitting polynomials, Bezier curves, or other smooth functions to the data. We chose to represent the road center and width using Bezier curves. Unfortunately, in trajectory optimization the vehicle can stray from the center curve if it provides better performance on other metrics (such as reducing jerk and steering angle). This complicates the cost function.

A major issue was formulating a cost function that causes the vehicle to move forward without unnecessarily penalizing velocity changes. If the MPC problem is initialized with a (suboptimal) speed estimate, choosing points on the road center for steps $1, \ldots, N$ without introducing "noise" in the accuracy term of the cost function (3) is difficult (Figure 10). This issue was resolved by using previously computed trajectories to generate the center points, but is still present in the first MPC iteration.

C. Limitations

One limitation of the road representation in Fig. 2 is that if there is an obstacle in the middle of the road, the vehicle cannot evaluate both paths around it. One must be selected by higher-level software.

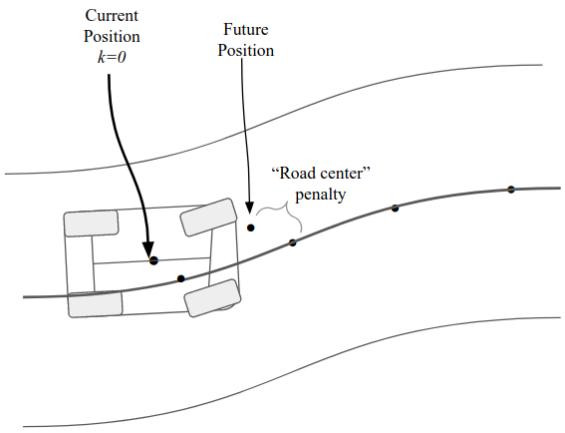

Fig. 10. Using poorly chosen center points adds a penalty if the optimal speed deviates from the initial guess.

V. Conclusions

The controllers discussed in this paper take several approaches to the problem of autonomous vehicle steering. On one side is the question of modeling the vehicle: complex robotics-inspired models can provide more accurate dynamics, but are likely too computationally difficult for an optimal control approach such as MPC. Classical control techniques can be used to analyze these models and design stabilizing controllers as done in [3]. Additional concerns include uncertainty and changes in vehicle parameters that could affect the controller's performance in real life, and the difficulty of system identification to port the controller to new vehicle hardware.

Optimal control approaches are also common, but many of them focus on trajectory tracking, which can be achieved via MPC, as opposed to the difficult nonlinear problem of trajectory optimization. New advances in nonlinear MPC may alleviate this issue in the future: more advanced dimensional lifting techniques can approximate the problematic nonlinear dynamics, transforming the problem into a convex quadratic formulation that can be quickly and reliably solved [9].

VI. Future Work

In this paper, we have discussed the benefits and drawbacks of different steering and acceleration controllers from a hardware portability view. It is difficult to directly compare these controllers because of the lack of standardized tests defined for them. Some researchers select clothoids, commonly used in designing freeway ramps, and Euler spirals to construct test roads [2]. Others use the roads around their home institution for test data [4]. One open question is the design of standard driving tests for these controllers. Defining a set of standard paths and vehicle maneuvers for benchmarking controllers will benefit future researchers by enabling easy quantitative comparison of algorithms.

Another avenue of future research is parameter estimation and controller robustness against changing vehicle dynamics. Vehicle mass and CG can change for many reasons: for example, delivery vehicles picking up and dropping off heavy cargo, fuel tanks emptying over a long drive, and trucks transporting liquids that can move in their tanks. To be practically useful, a steering and acceleration controller must be able to adapt as parameters change. Additionally, some important parameters are difficult to measure and change with road conditions. The parameters describing tire-road interactions cannot be directly measured, but have been estimated using a complex two-step method, accounting for both slow changes (tires degrading over time) and rapid ones (such as driving into a puddle) [10]. Environmental data such as the road gradient and bank angle are relevant for stability control / anti-rollover systems, but cannot be easily measured: some algorithms exist to estimate them using velocity and engine torque data [11].

VII. Acknowledgments

This work was completed under Professor Sanjay Lall in the Information Systems Lab. The NMPC code was based on several open-source files provided with CasADi.

References

[1] Bernd Hardung, Thorsten Kölzow, and Andreas Krüger. Reuse of software in distributed embedded automotive systems. In *Proceedings of the 4th ACM International Conference on Embedded Software*, EMSOFT '04, page 203210, New York, NY, USA, 2004. Association for Computing Machinery.

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

[7] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

[8] Andreas Wchter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 2006.

[9] Milan Korda and Igor Mezi. Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control. *Automatica*, 93:149160, Jul 2018.

[10] Simone Garatti and Sergio Bittanti. Parameter estimation in the pacejka's tyre model through the ts method. *IFAC Proceedings Volumes*, 42(10):1304–1309, 2009. 15th IFAC Symposium on System Identification.

[11] M. N. Mahyuddin, J. Na, G. Herrmann, X. Ren, and P. Barber. An adaptive observer-based parameter estimation algorithm with application to road gradient and vehicle's mass estimation. In *Proceedings of 2012 UKACC International Conference on Control*, pages 102–107, 2012.