Software Reusability in Autonomous Vehicle Steering

Emiko Soroka
PI: Sanjay Lall

Stanford University

April 27, 2021

Software Reusability: A Growing Problem

Advanced driver assistance systems (ADAS) and autonomous vehicles are a rapidly growing segment of the automotive industry. However, this success comes with new challenges in software reusability.
To equip more vehicles for autonomous driving, manufacturers must be able to:

- Port an existing autonomous driving stack to vehicles with different capabilities and sensor configurations.
- Verify safety-critical software operates correctly on different platforms.
- Comply with regulations and standards.

Software Portability in the Autonomy Stack

Some high-level modules are relatively independent of vehicle hardware, such as route planning, computer vision, and other high-level processes.
However, low-level control depends on the vehicle's specific hardware for a variety of reasons.

- Fuel efficiency: hybrid vehicle vs electric vehicle, regenerative braking.
- Physical parameters: weight, center of gravity (CG), tire size and tire-road interactions.
- Engine performance (number of cylinders, front-wheel drive vs all-wheel drive, etc).

In this presentation, we focus on low-level steering and acceleration control. High-level modules provide a path to follow: our goal is to track it.

Steering an Autonomous Vehicle

Major concerns when steering an autonomous vehicle:

- Safety: vehicle must not lose control or stray from a safe path.
- Comfort: Steer and accelerate smoothly, reducing jerk (change in acceleration).
- Fuel efficiency: avoid unnecessary control inputs.

However, not all applications weight these equally.

- Autonomous delivery truck: maximize fuel efficiency.
- Autonomous taxi: provide a comfortable and enjoyable ride.

Configurable behavior is already on the market: many ADAS systems allow switching between "sport" and "eco-friendly" driving modes.

Trajectory Tracking with Waypoints

In trajectory tracking (as opposed to trajectory planning or optimization), the goal is to follow a given path as closely as possible. Many approaches are possible, including PID and tracking MPC. Some challenges with trajectory tracking include:

- Tuning PID gains
- Proving a PID controller is stable
- Developing controllers for complex vehicle models that represent the interactions between tires, axles, vehicle body, suspension, etc.

Problem: A higher-level module must have enough information about the vehicle hardware to generate the waypoints.

Trajectory Optimization: Beyond Waypoints?

Some controllers have also been developed to optimize the vehicle's trajectory: for example, by determining the best path in a safe driving corridor.
The high-level software could provide this safe corridor and some high-level goals ("drive at this speed", "stop at this point"). The exact path is determined by the steering controller.
Some challenges with this approach include:

- The vehicle has nonlinear dynamics that add difficulty to the optimization problem.
- The safe corridor the vehicle can drive in is more difficult to represent.
- The vehicle model must be computationally tractable.
- The optimization must run in real time.

Software reusability perspective

Changing behavior is arguably a high-level goal, affecting what speed the vehicle drives at, what lanes it drives in, and other decisions.
However, to plan these behaviors the autonomy software must know what its hardware is capable of.

- If our trajectory planner is platform-independent, it can't take advantage of the full capabilities of the hardware.
- If our planner is hardware-dependent, it will be difficult to port the software to new vehicles.

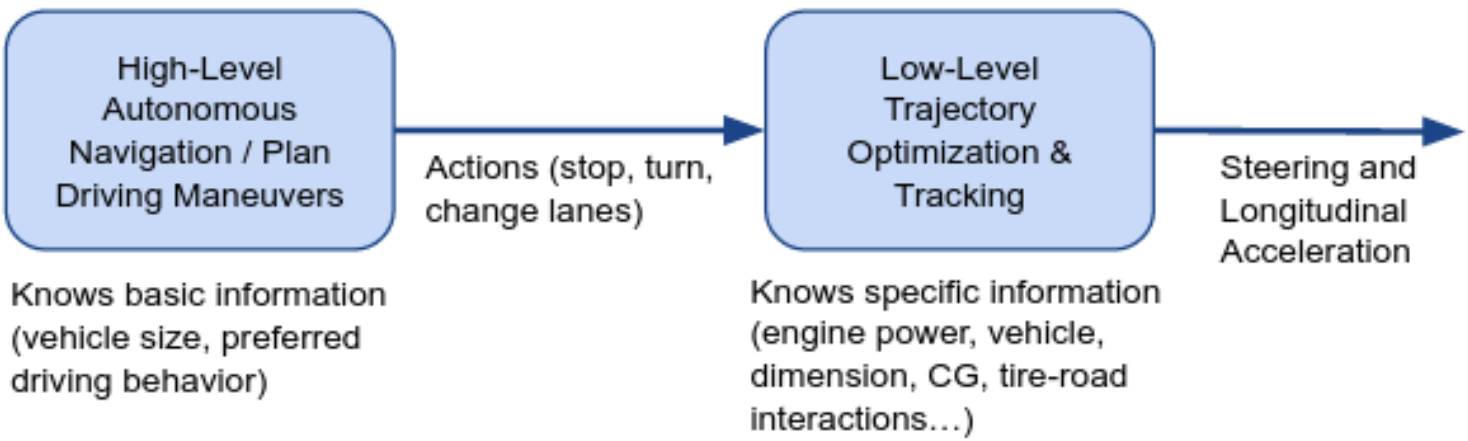
Proposed Division

High-level software generates an inexact representation of the path without much knowledge of hardware capabilities.

- Provides a desired speed
- Provides a safe driving corridor
- Provides specific goals for low-level controller (stop at stop sign, hold constant speed on freeway)

Low-level controller optimizes this path.

- Accounts for hardware capabilities
- Accounts for user preferences (comfort, fuel efficiency, etc.)



MPC Trajectory Optimization

Model-predictive control (MPC) is a good choice for the low-level controller.

- Optimization approach: weighting different objectives is an intuitive way to adjust driving behavior.
- State constraints can be used to represent a safe corridor to drive in.
- Constraints can be used to guarantee a trajectory won't violate safety requirements.
- Vehicle kinematics models can be switched out without requiring major modifications to the controller.

Trajectory Optimization Challenges

Nonlinear optimization is difficult to implement.

- Convergence to an optimal solution isn't guaranteed.
- In a complex problem, it can be difficult to determine why the solver failed or produced an unexpected result.
- Discretization of the vehicle dynamics and road corridor can introduce problems if not accurate enough.

Trade-off between model accuracy and computational tractability.

- Even "simple" models like the kinematic bicycle can be difficult to work with.
- Adding complexity worsens this issue, but is necessary to handle a full range of driving conditions (example: knowledge of tire-road interactions is important in an emergency stop).

Implementation of MPC controller

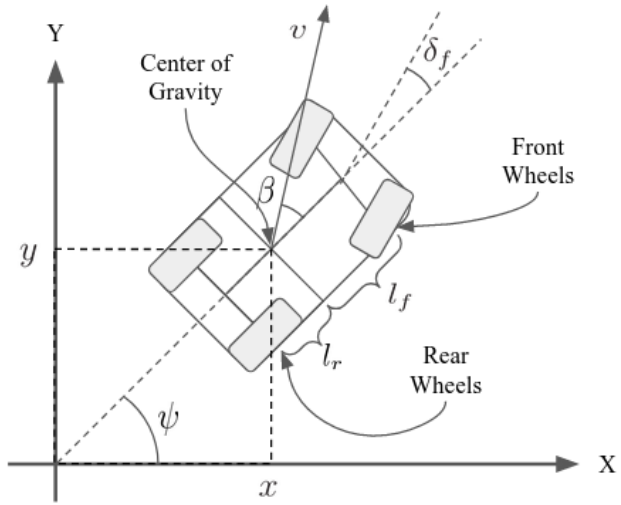


Figure: Kinematic bicycle model, commonly used in MPC controllers.

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

β is the angle between the car velocity and its longitudinal axis (2).

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

Implementation

- Consider N lookahead steps $k = 1, \ldots, N$ spaced a distance Δ_t apart
- The control signals are a, the longitudinal acceleration of the car, and δ_f, the steering angle of its front wheels. $u = [a \ \delta_f]$
- The nonlinearity is confined to the dynamics model (1)
- The cost function is quadratic in state z and control u
- The state and control constraints are representable by linear inequalities

Implementation: Cost Function

Define a multi-objective cost function. Changing the weights on each term will change the driving behavior.

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

where $(x_{center,k}, y_{center,k}.\psi_{center,k}$ describe the $x - y$ position and angle of a desired point on the road (more on this later).

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

Equation (3) controls the accuracy of the path-following behavior, while (4) controls how closely the vehicle stays at the desired speed.

Implementation: Cost Function

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

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

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

Combine these terms to define the cost function:

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

Implementation: MPC Problem

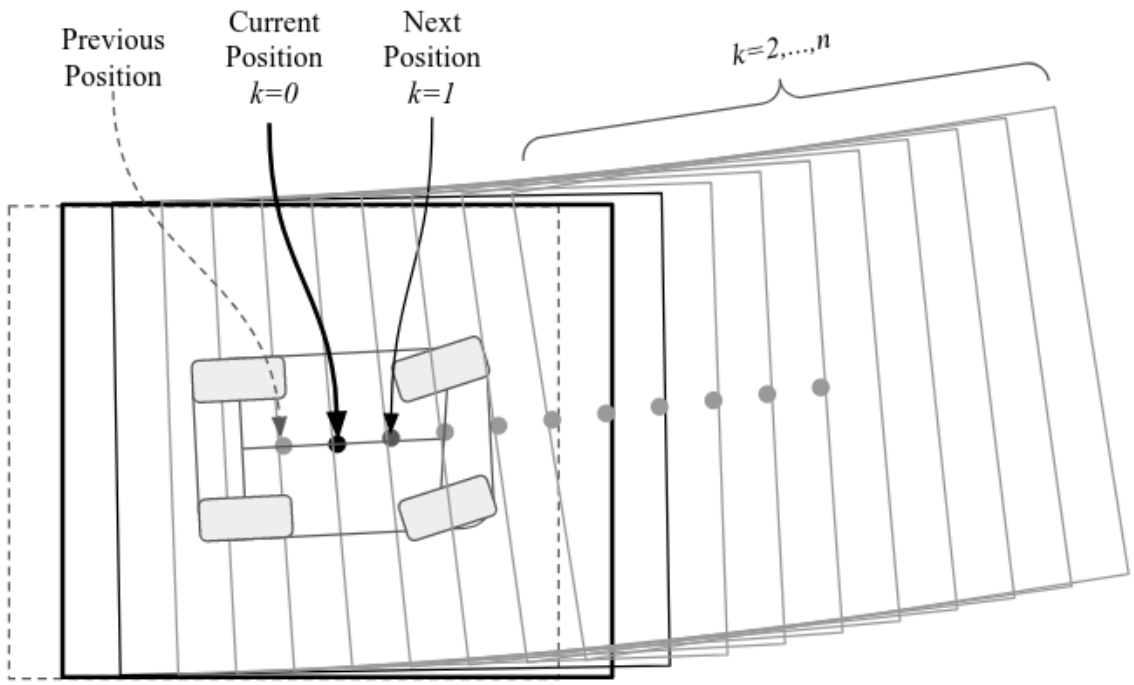
Define the nonlinear MPC problem:

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

A_k is the matrix of linear inequalities describing the road boundaries at the kth step.
The "min" and "max" bounds on u, v and ψ are constant.

Implementation Challenges

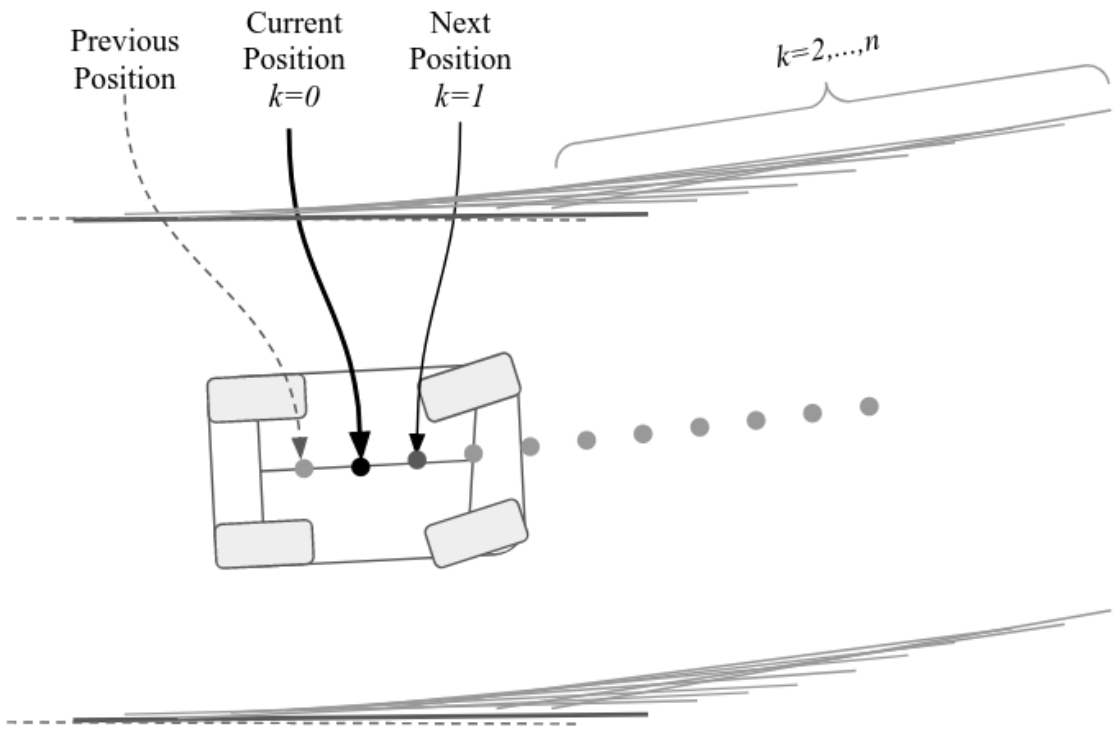
A major issue was representing the road accurately and efficiently. Proposed solution: Use polygons to approximate the road at each step $k = 1, \ldots, N$.



Then the position constraint on $x_k, y_k,\ k = 1, \ldots, N$ is simply a set of linear inequalities.

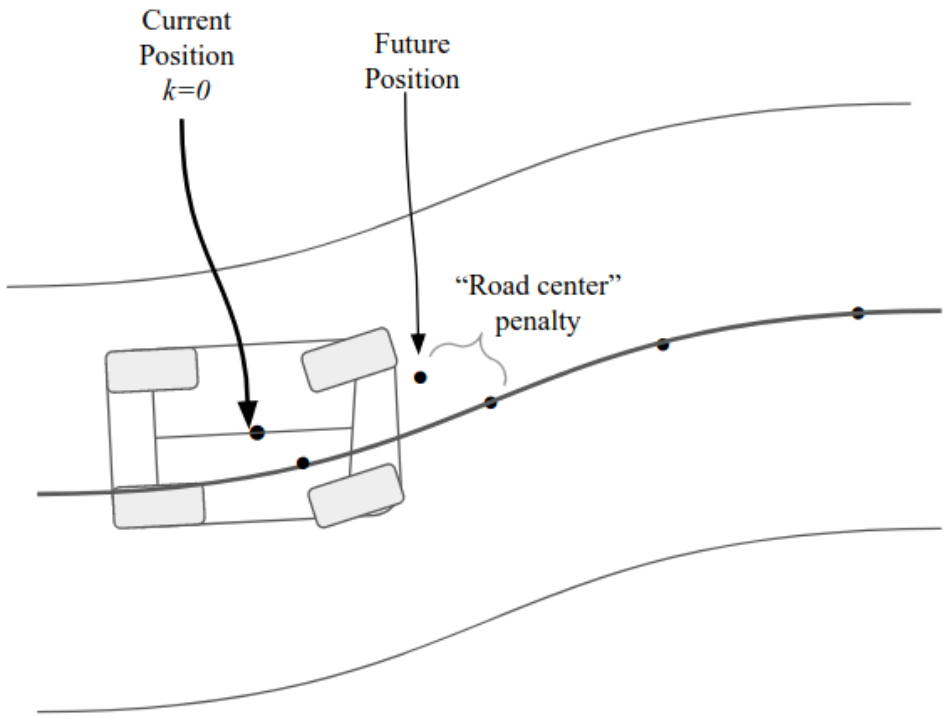
Implementation Challenges

This actually caused a lot of issues because it was difficult to generate the polygons without unnecessarily constraining the vehicle speed. A simplification I thought of to reduce coding issues (which loses the guarantee that the vehicle will stay in a bounded region, although in practice large accelerations are penalized) is to use the right and left boundaries, but leave the rear and front open.



Implementation Challenges

Another major issue was formulating a cost function that causes the vehicle to move forward while allowing it to change its velocity. If we start with a speed estimate (that may not be optimal), choosing points on the road center for steps 1,...,N without introducing "noise" in the cost function is difficult.



Solution: use previously computed trajectories to generate the center points.

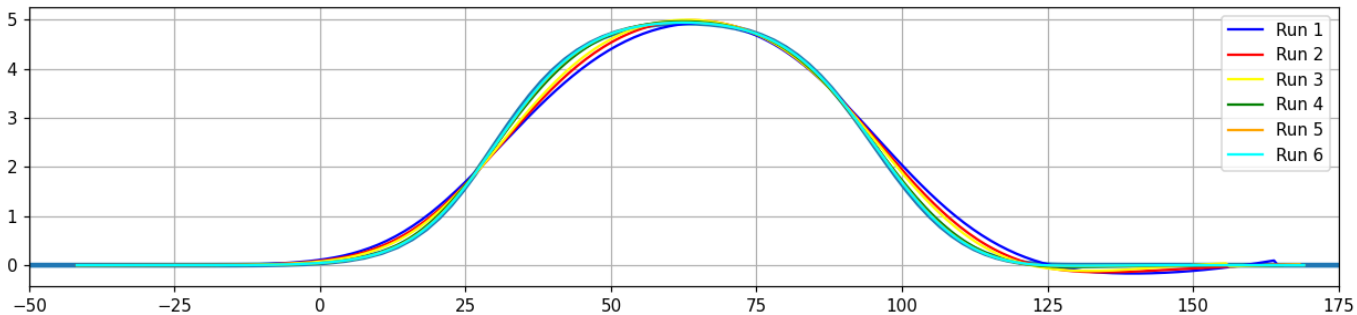
Validation: Performance on ISO Lane Change

The ISO double lane change path is a test performed by human drivers on a real track, but is also used in some papers on autonomous driving.
The controller was implemented in Python using CasADi for symbolic math and Ipopt.

- The linear boundaries successfully overlap to represent curved roads.
- The vehicle behavior can be changed using the weights on the cost terms.
- Because the road boundaries are a hard constraint,the vehicle cannot stray out of the road corridor. Deviations from the center of the road are only penalized, allowing it to make small adjustments to its path.

Validation: Performance on ISO Lane Change

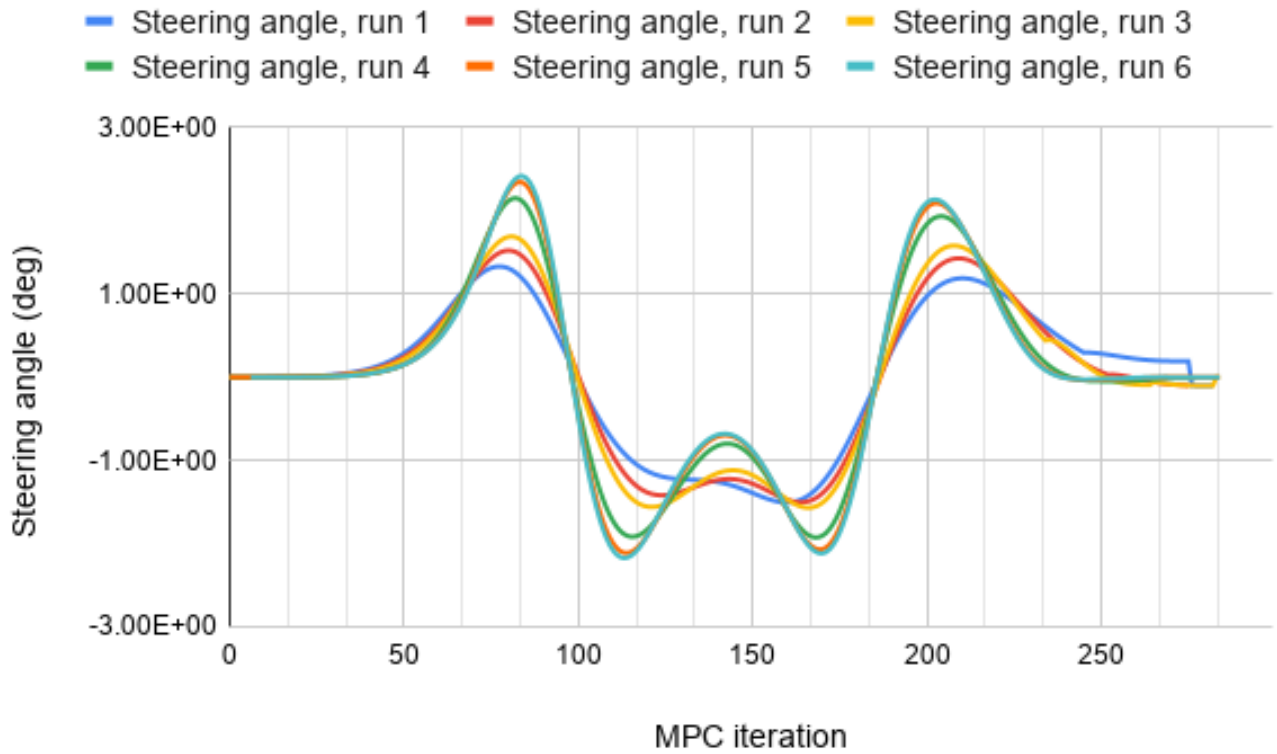
In this slide, we see the results of several test runs. The "accuracy" and "speed" weights were successively increased: jerk and steering change weights remained the same at 100 and 10, respectively.



Run	1	2	3	4	5	6
$J_{accuracy}$	0.01	0.05	0.1	1.0	5.0	10.0
J_{speed}	0.1	0.5	1.0	10.0	50.0	100.0

Validation: Performance on ISO Lane Change

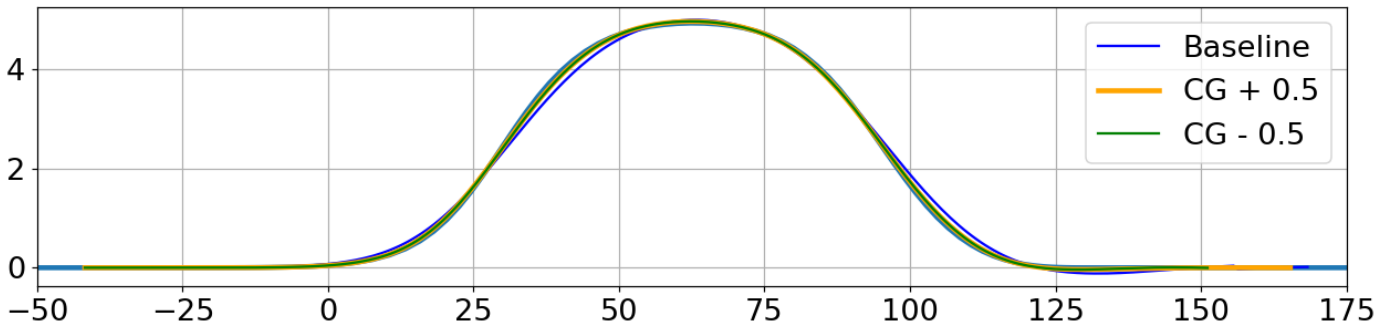
As expected, the magnitude of the steering input increases to keep the vehicle closer to the road center.



Run	1	2	3	4	5	6
$J_{accuracy}$	0.01	0.05	0.1	1.0	5.0	10.0
J_{speed}	0.1	0.5	1.0	10.0	50.0	100.0

Changing Model Parameters

The kinematic bicycle model has only 2 parameters describing the distances between the rear/front axles and the vehicle CG.



Changing these parameters does not destabilize the system, though we will see larger control inputs are required to keep the vehicle on the path.

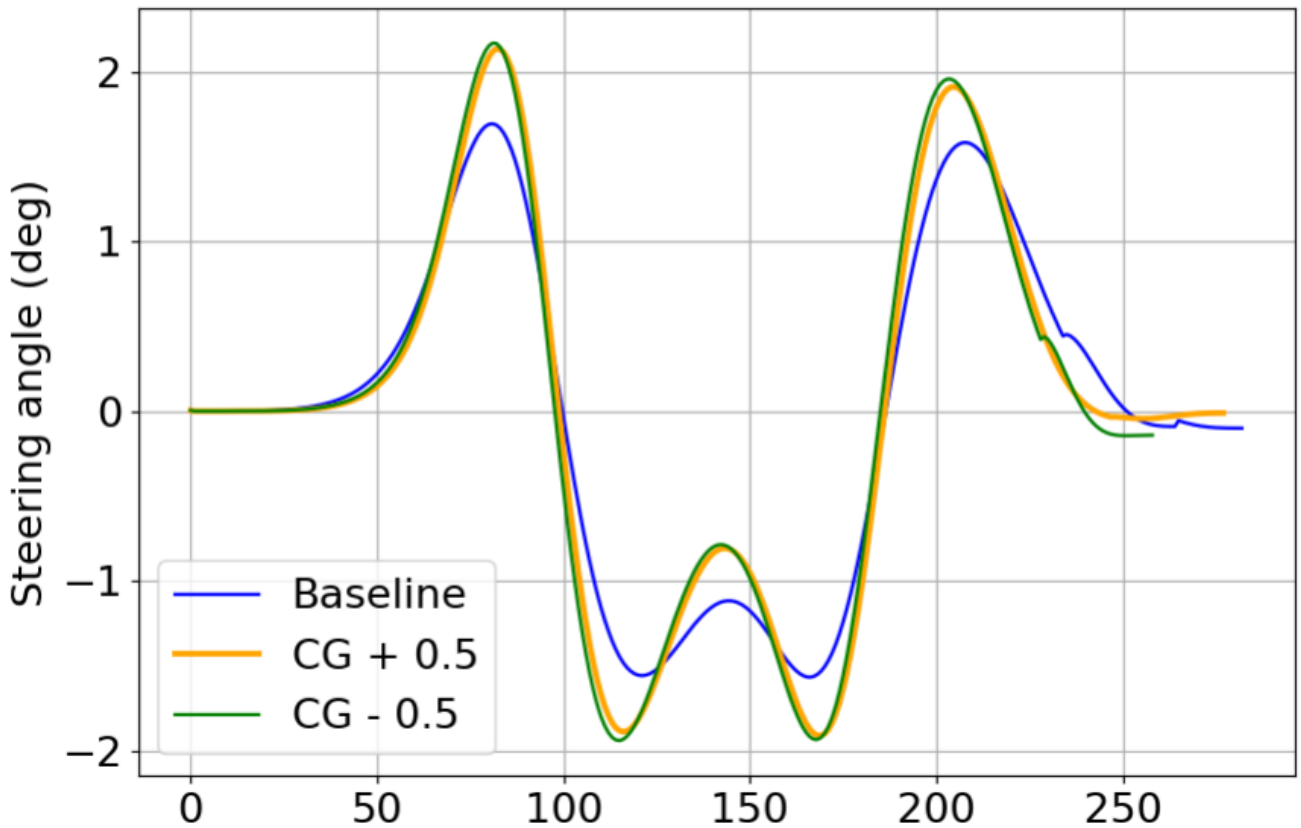
All 3 runs were computed with cost:

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

which corresponds to the "mid-range" performance seen on the previous graphs.

Changing Model Parameters

When the CG is pushed forward or back, as expected, larger control inputs are generated.



This proof of concept shows how an MPC controller provides software reusability. A high-level autonomous system providing instructions to this optimizing controller does not have to "know" about the change in CG.