Software Reusability in Autonomous Vehicle Steering

Emiko Soroka
PI: Sanjay Lall

Stanford University

May 9, 2021

Software Reusability: A Growing Problem

Advanced driver assistance systems (ADAS) and autonomous vehicles are a rapidly growing segment of the automotive industry

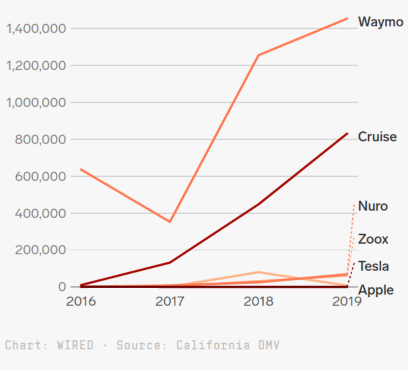


Figure: Autonomous miles driven by top California companies

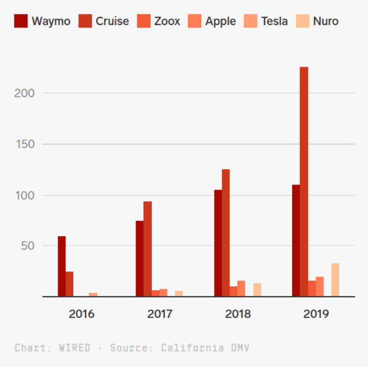


Figure: Number of self-driving vehicles on the road

(Chart source: `https://www.wired.com/story/california-self-driving-cars-log-most-miles/`)

Software Reusability: A Growing Problem

The rise of autonomous driving comes with new challenges in software reusability.

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

Software Reusability: A Growing Problem

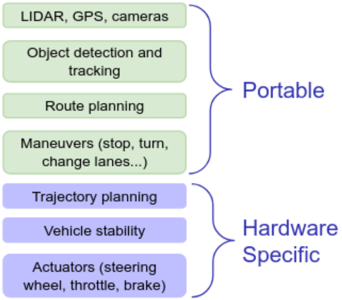
The rise of autonomous driving comes with new challenges in software reusability.

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

Software Portability in the Autonomy Stack

High-level software is easily ported to different vehicles. However, low-level control depends on vehicle-specific hardware for multiple reasons.

- Fuel efficiency
- Vehicle dynamics
- Engine performance



In this presentation, we focus on steering and acceleration control. Our goal is to **disentangle hardware-dependent control algorithms from higher-level software.**

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.

Literature review

- Driving in an allowable corridor (trajectory optimization) using inner model control (Li et al. Chinese Control Conference 2013).
- Using Bezier curves to represent roads: algorithm first generates a smooth path, then applies MPC to determine the vehicle's velocity along the path (Qian et al. (ITSC 2016).
- Switching between energy-efficient and sport mode using nonlinear MPC (Daoud et al. ICVES 2019). Assumes the vehicle drives along a known path (trajectory tracking).
- More complex MPC objectives addressing both tracking error and passenger comfort (Farag, Journal of Intelligent Transportation Systems 2020).

Trajectory Tracking with Waypoints

Tracking controllers follow a given path as closely as possible.
Many approaches are possible, including PID and tracking MPC.
Challenges with trajectory tracking:

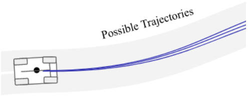
- Tuning PID gains
- Proving a PID controller is stable
- Developing controllers for complex vehicle models that represent the interactions between tires, axles, vehicle body, suspension, etc.

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

Trajectory Optimization: Beyond Waypoints?

Some controllers optimize the vehicle's trajectory: for example, by determining the best path in a safe driving corridor.

High-level software could provide this safe corridor and driving goals. The exact path is determined by the steering controller.

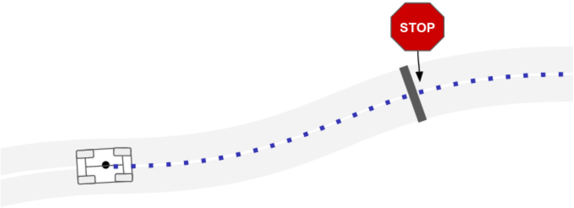


Some challenges with this approach:

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

Example Applications

- Dynamic maneuvers such as emergency braking / steering
  - Requires accurate knowledge of hardware capabilities
- Stopping at a stop sign (an interesting problem)
  - Need a flexible interface so the controller can smoothly switch between objectives



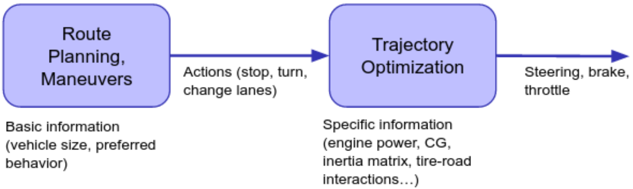
Proposed Division

High-level software operates 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)

Vehicle-specific 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.

- Weighting different objectives is an intuitive way to adjust driving behavior
- Constraints can represent an allowable corridor to drive in, guaranteeing a trajectory won't violate safety requirements.
- Vehicle kinematics models can be switched out without requiring major modifications to the controller.
- The controller provides an abstraction to higher levels of the autonomy stack.

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 can be difficult to discretize.
- Adding complexity worsens this issue, but is necessary to handle a full range of driving conditions.

Implementation of MPC controller

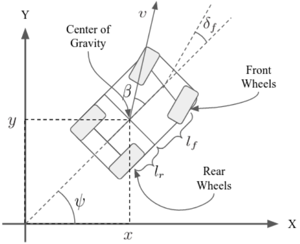


Figure: Kinematic bicycle model, commonly used in MPC controllers.

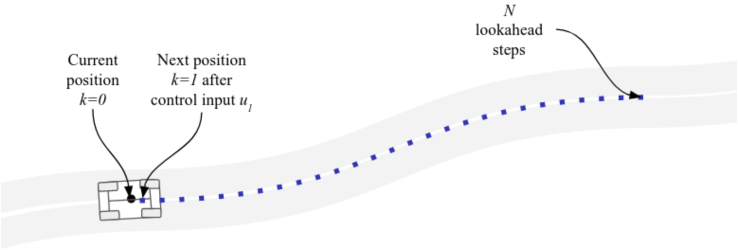
$$\text{State } z = \begin{bmatrix} x \\ y \\ v \\ \psi \end{bmatrix}, \quad \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{v} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} v\cos(\psi+\beta) \\ v\sin(\psi+\beta) \\ a \\ \frac{v}{l_r}\sin(\beta) \end{bmatrix} \tag{1}$$

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 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 at each step



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).

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

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

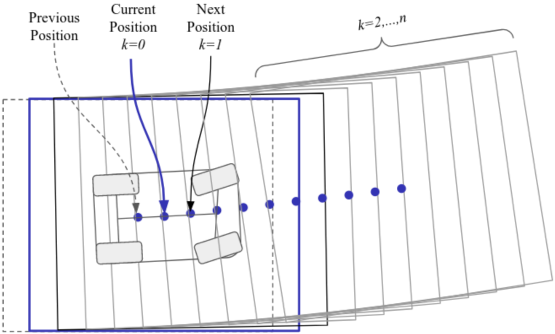
Define the nonlinear MPC problem:

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

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

Implementation Challenges

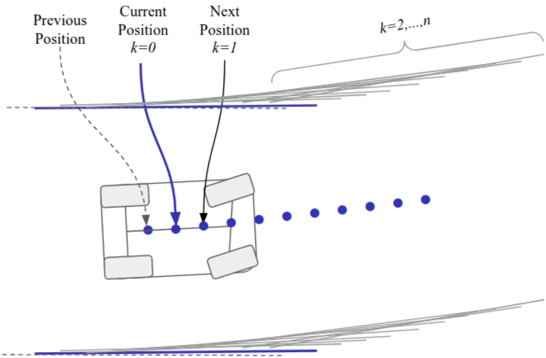
A major issue was representing the road accurately and efficiently. Proposed solution: Use polygons to approximate the road at each step $k = 1, \ldots, N$.



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

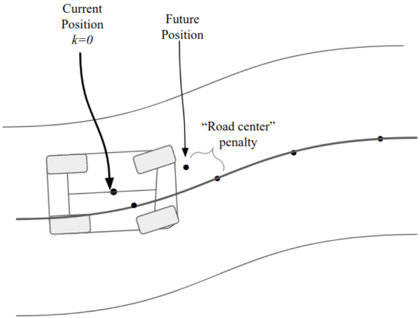
Implementation Challenges

This caused a lot of issues because it was difficult to generate the polygons without unnecessarily constraining the vehicle speed. Simplification: 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 without unnecessarily penalizing velocity changes.
If we start with a (suboptimal) speed estimate, choosing points on the road center for steps $1, \ldots, N$ without introducing "noise" in the cost function is difficult.



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

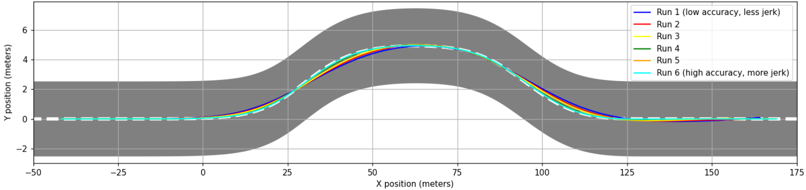
Performance on ISO Lane Change

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.

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.



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

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

Performance on ISO Lane Change

The error between the road center and vehicle path decreases with increased weight on $J_{accuracy}$.

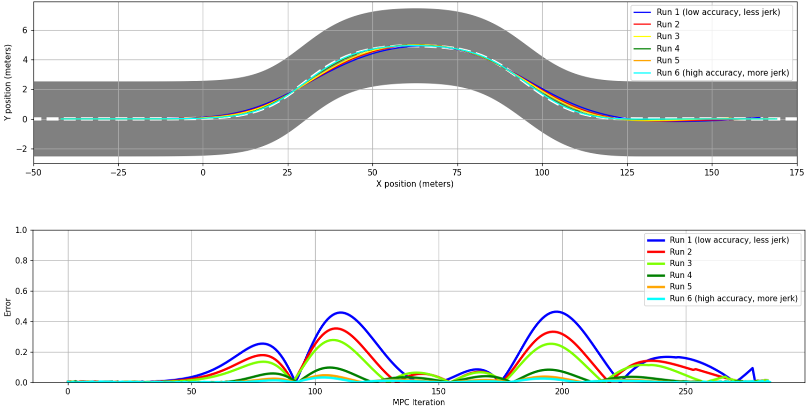
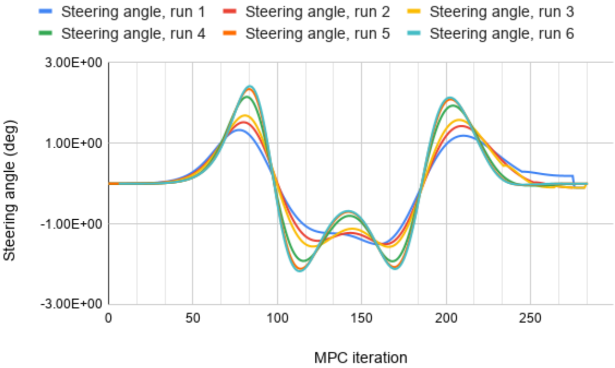


Figure: Error $\|x - x_{center}\|_2$

Performance on ISO Lane Change

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

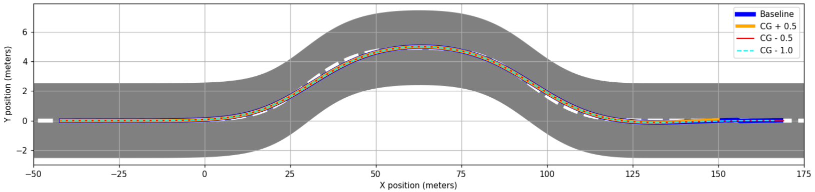


$$J = \alpha J_{accuracy} + 10\alpha J_{speed} + 100 J_{jerk} + 10\frac{180}{\pi} J_{steering}$$

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

Changing Model Parameters

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



Changing these parameters does not destabilize the system, though larger control inputs are computed.
All 3 runs were computed with cost:

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

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

Changing Model Parameters

The error between the road center and vehicle path is almost unchanged.

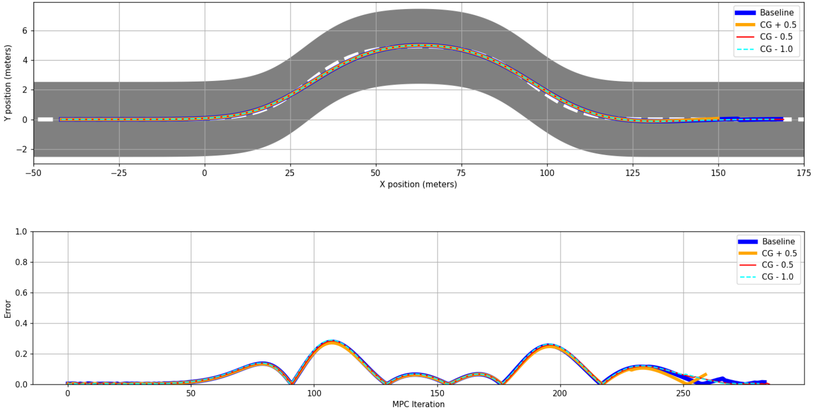
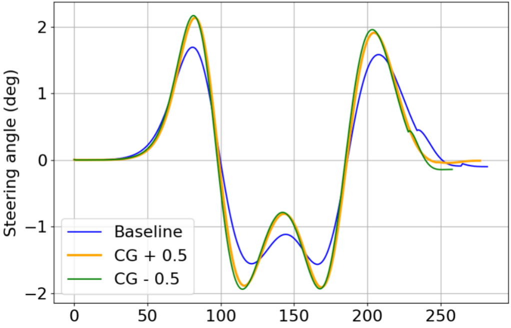


Figure: Error $\|x - x_{center}\|_2$

Changing Model Parameters

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



This proof of concept shows how the controller could improve software reusability. A high-level autonomous system providing instructions to this optimizing controller does not have to “know” about the change in CG.

Summary

Current accomplishments:

- Programmed a nonlinear MPC controller for vehicle steering and longitudinal acceleration
- Demonstrated ability to easily change vehicle parameters
- Configurable driving behavior via weighted multiobjective cost function.

Future research directions:

- Determine appropriate cost function weights for modal changes (example: coming to a stop vs traveling at constant speed).
- Faster and more reliable methods of solving nonlinear MPC problems.
- Real-time estimation: vehicle parameters can change slowly (tire degradation, temperature change), rapidly (driving over a patch of ice) and while parked (loading/unloading cargo).

Questions?