How to: Controls

ETH Zurich

Manuel Dangel
Jose Vazquez

powered by

Control Approaches at FS Driverless

One control system to rule all disciplines.

- ✓ Less development time
- ✗ Generality is hard to achieve

Specific controllers for each discipline

- ✗ More development time
- ✓ Exploits the structure of each discipline

Disciplines:

- Acceleration
- Skidpad
- AutoX
- Trackdrive

Unstructured vs Structured Environments

Unstructured

- **Objective:** Go as fast as possible without a global map
- ✗ Noisy signals from upper stack

Structured

- **Objective:** Go as fast as possible with an a-priori knowledge of the map
- ✓ Aggressive controls benefit from the cleaner signals

Disciplines:

- AutoX
- Acceleration
- Skidpad
- Trackdrive

Control as tracking

Control in FSD can be seen as a standard **reference tracking** problem

The main ingredients are:

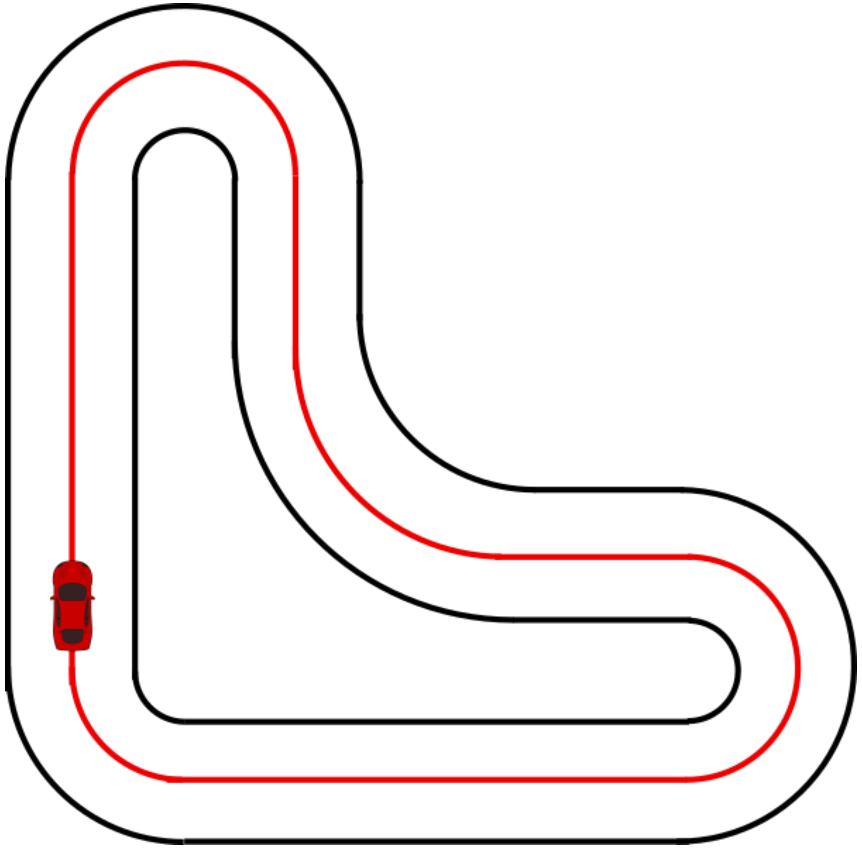
- A reference
- A feedback controller

And there is **plenty** of to choose from in each category

Reference:
- Centerline
- Precomputed Race-line
- Constant Speed
- Optimal speed profile

Feedback Controller:
- PID
- Pure Pursuit
- LQR
- Linear MPC
- Non-linear MPC

Simple Reference Trajectory



First steps:

- The simplest lateral reference to track is the track's center-line.
- You can also control your vehicle to reach a "safe" constant longitudinal speed

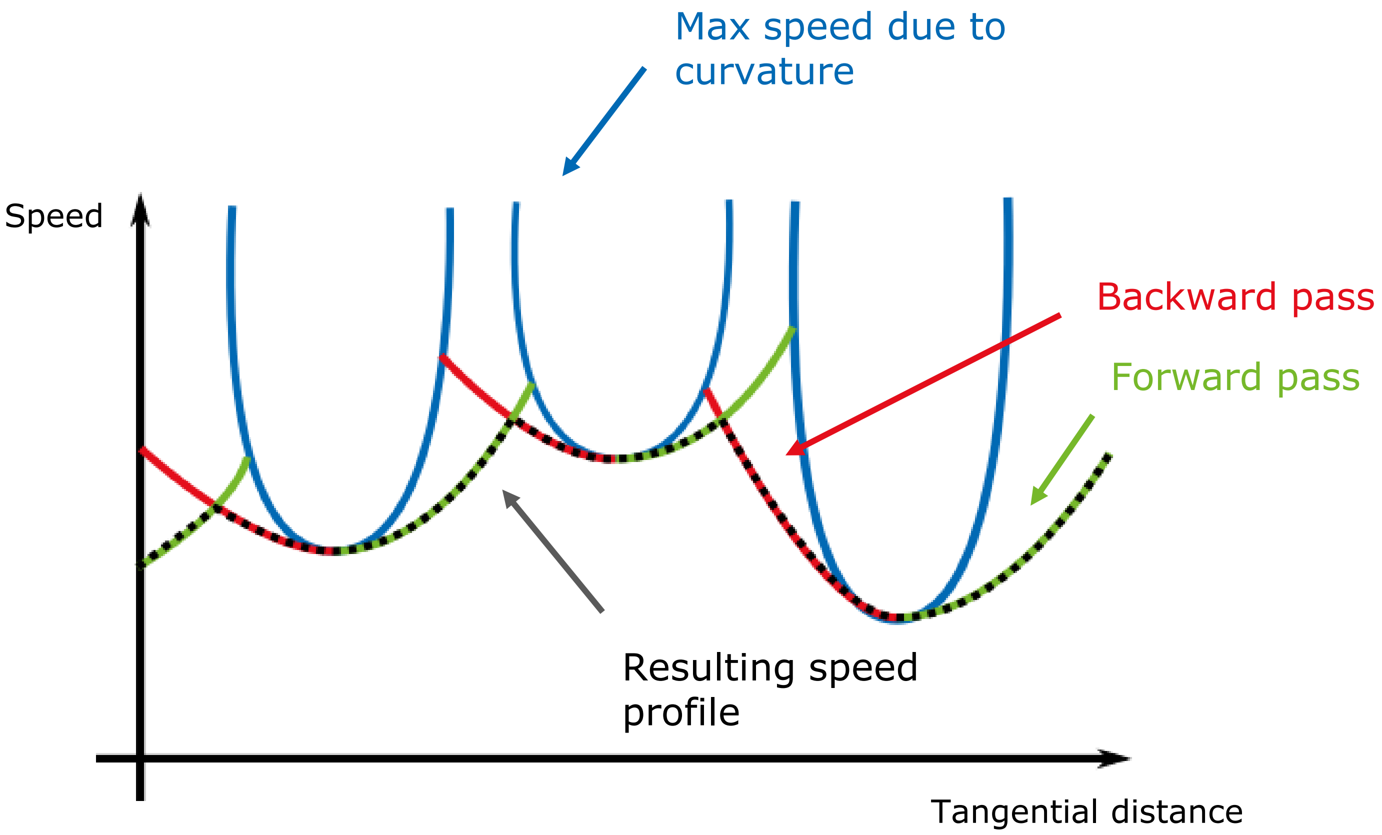
Steady State Velocity Profile

There is a **simple way** to compute a velocity profile

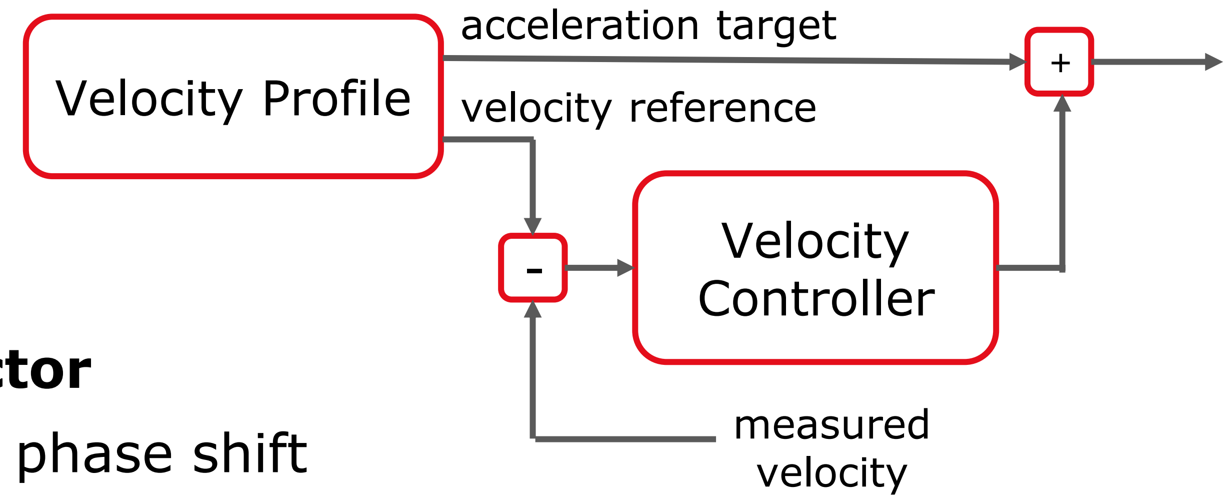
- Max speed given curvature and max acceleration

$$v = \sqrt{\frac{a_y}{\kappa}}$$

- **Forward pass:** simulate with acceleration limit
- **Backward pass:** simulate with deceleration limit
- Min of all is the achievable velocity

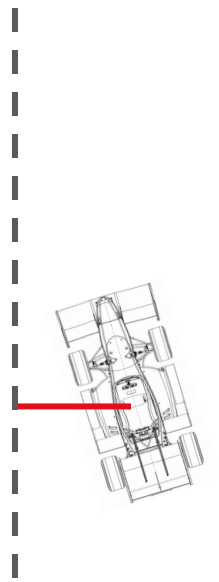


PID + Feedforward



Remarks:

- A **PID** alone is only a **disturbance rejector**
  - It follows a changing reference with a phase shift
- To track a varying reference always use a feedforward
- Beware of Integrator Windup
- If possible measure the derivative error directly
- Choose control gains on **"the right errors"**



Alternatives:

- Optimal Control: For example LQR
  - Different only if higher order system is modeled and measured

Pure Pursuit Controller

Simple all-in-one Lateral Controller

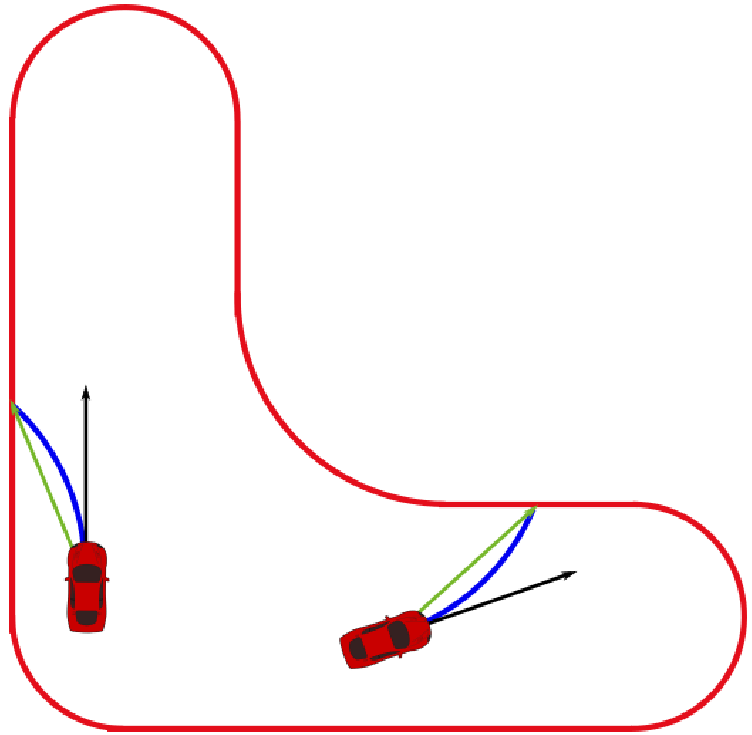
- Combines feedforward and feedback
- bias-free tracking of steady state corners

Tips and Tricks:

- Keep controller frequency constant with speed
- Compensate for an estimated Slip Angle

Alternatives:

- Curvature Feedforward & Feedback P(I)D
- Stanley Control



[Couler] Implementation of the Pure Pursuit Path tracking Algorithm

Intermezzo - Optimization

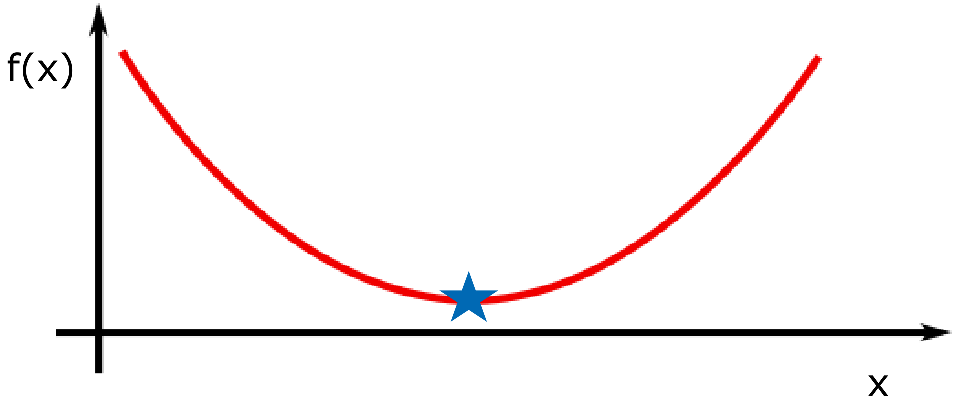
Objective function: What are you minimizing?

- Time? Energy? Space?

$$\min_x f(x)$$

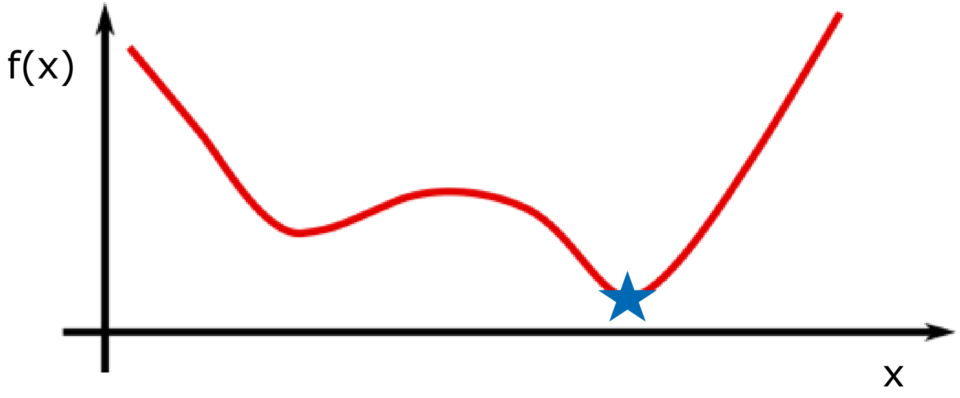
Convex objective functions:

- ✓ Easier to optimize
- ✗ Sometimes less expressive
- ✓ Only one optimum



Non-Convex objective functions:

- ✗ Harder to optimize
- ✓ Sometimes more expressive
- ✗ Can have several optima



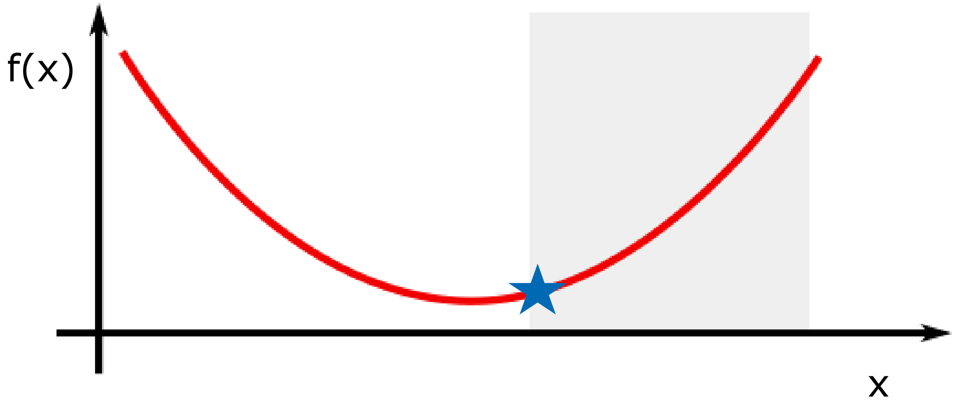
Intermezzo - Optimization

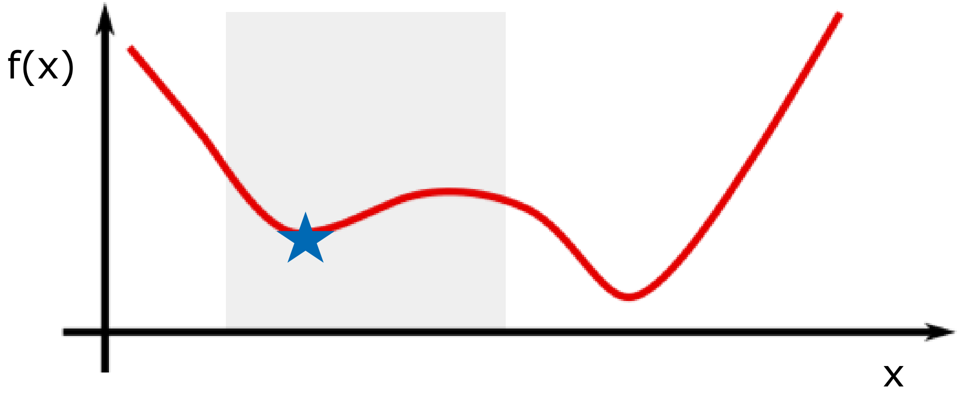
Objective function: What do you want to do?

Constraints: What you really don't want to do?

- Defines where your solution <u>shouldn't</u> be

$$\min_{x} f(x)$$
$$\text{s.t.} \quad g_1(x) \leq 0$$
$$g_2(x) = 0$$



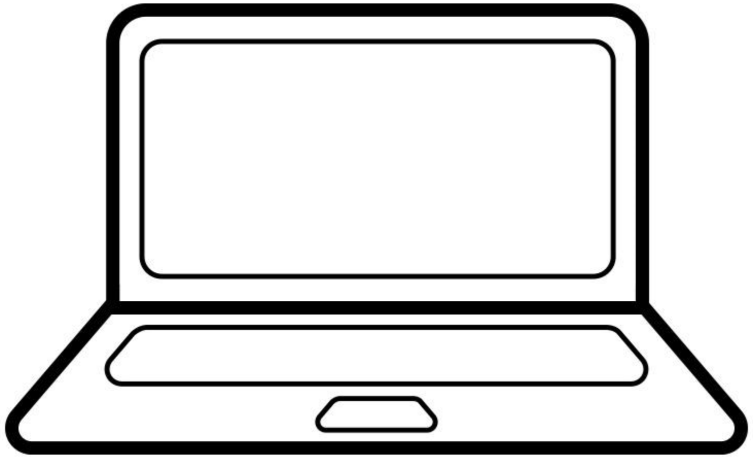


Intermezzo - Solvers for Optimization

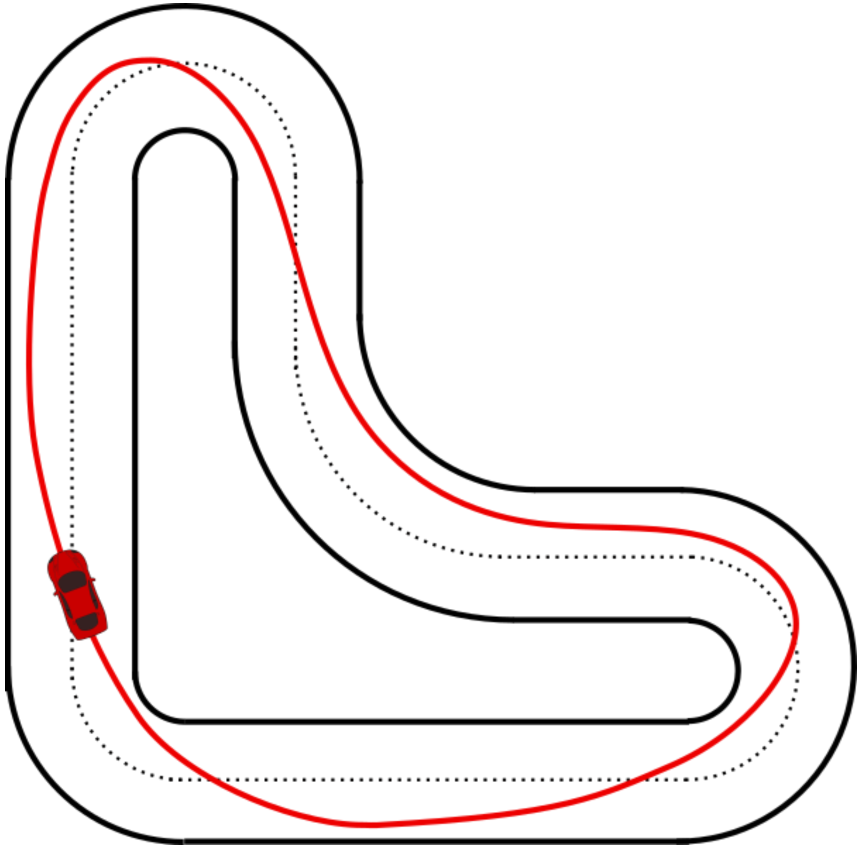
Examples of open-source and commercial solvers:

- **Convex Optimization**
  - OSQP
  - CVX
  - qpOASES
  - HPIPM

- **Non-convex optimization**
  - IPOPT
  - ACADOS/ACADO (For control applications)
  - FORCES PRO (For control applications)



Curvature Optimal Reference



$$\min_{x} f(x)$$ Raceline curvature

$$\text{s.t.} \quad g_1(x) \leq 0$$ Staying within the track bounds

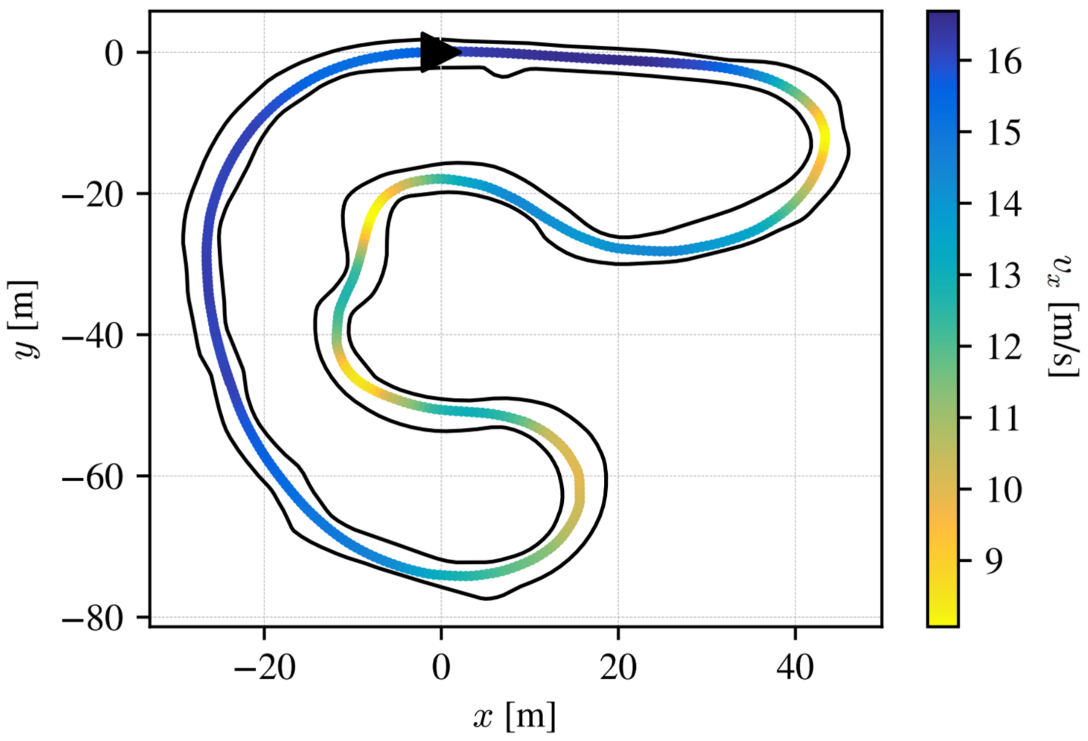
$$g_2(x) = 0$$ Curvature as a function of tangential distance

Remarks:

- Easy solution to get a smoother, better reference than the middle line
- Velocity profile can be computed using the curvature

[Heilmeier, et al.] Minimum curvature trajectory planning and control for an autonomous race car

Time Optimal Reference



$$\min_x f(x)$$

$$\text{s.t.} \quad g_1(x) \leq 0$$
$$g_2(x) = 0$$

Lap-time

Obeying vehicle dynamics

Staying within the track bounds

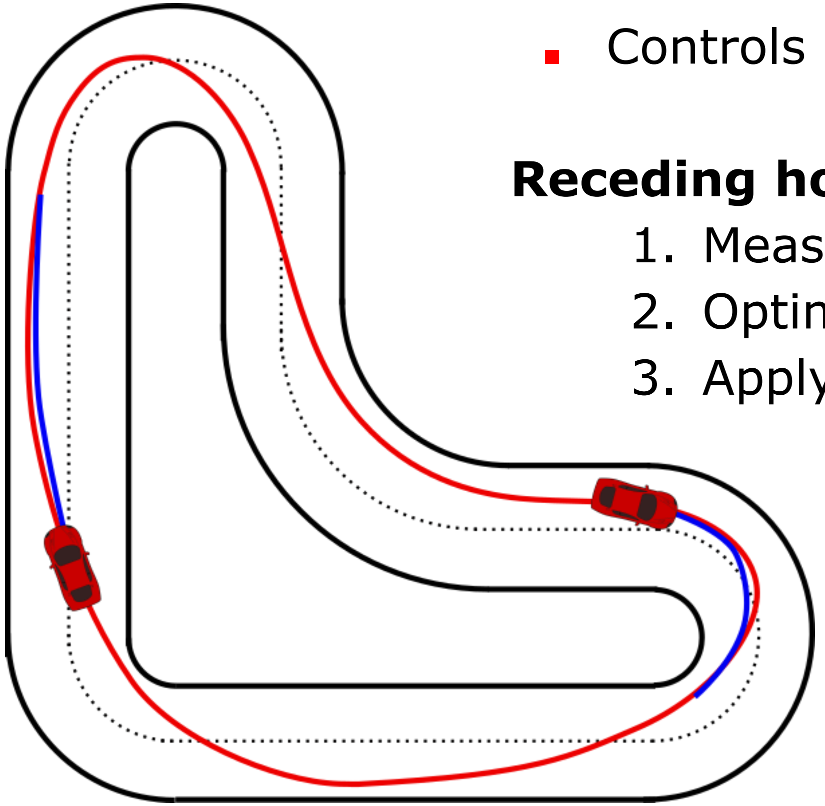
Remarks:

- Uses a vehicle model to minimize lap-time.
- The output is a full-state trajectory that can be used by a full-state tracking controller

[Vazquez, et al.] Optimization-Based Hierarchical Motion Planning for Autonomous Racing

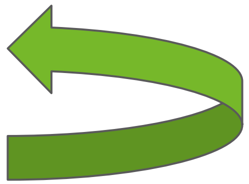
Model Predictive Control (MPC)

- Systematic way to **encode** specifications into the controller
- **Predicts** what the vehicle would do in the future
- **Minimizes** cost along **<u>horizon</u>** (prediction)
- Controls in a **<u>Receding Horizon</u>** fashion



Receding horizon:

1. Measure State
2. Optimize
3. Apply Action



Repeat

$$\min_{u} \sum_{t=0}^{T} j(x_t, u_t)$$

Cost along horizon

$$\text{s.t.} \quad x_{t+1} = f(x_t, u_t)$$

Discrete Dynamics

$$g(x_t, u_t) \leq 0$$

Constraints

$$x_0 = \hat{x}$$

State estimate

Different flavours of MPC

Linear MPC:

- ✓ Numerically easier (Quadratic problem)
- ✗ More Assumptions made
- ✗ Harder to formulate (Linearization)

$$\min_u \sum_{t=0}^{T} x^T Q x + u^T R u \quad \leftarrow \text{Quadratic cost}$$

$$\text{s.t.} \quad x_{t+1} = A x_t + B u_t \quad \leftarrow \text{Linear dynamics}$$

$$M x \leq 0 \quad \leftarrow \text{Linear constraints}$$

$$x_0 = \hat{x}$$

Non-Linear MPC:

- ✗ Numerically harder (Non-convex)
- ✓ Less Assumptions made
- ✓ Easier to formulate
- ✗ Gets stuck in local minima

$$\min_u \sum_{t=0}^{T} j(x_t, u_t) \quad \leftarrow \text{General cost}$$

$$\text{s.t.} \quad x_{t+1} = f(x_t, u_t) \quad \leftarrow \text{Non-linear dynamics}$$

$$g(x_t, u_t) \leq 0 \quad \leftarrow \text{General constraints}$$

$$x_0 = \hat{x}$$

Non-linear MPC in Trackdrive

Model Predictive Contouring Control MPCC

- Optimization-based autonomous racing of 1:43 scale RC cars [Liniger, et al.]
- Amz driverless: The full autonomous racing system [Kabzan, et al.]

Contouring Control in curvilinear coordinates

- Optimization-Based Hierarchical Motion Planning for Autonomous Racing [Vazquez, et al.]

$$\min_{u} \sum_{t=0}^{T} j(x_t, u_t)$$

$$\text{s.t.} \quad x_{t+1} = f(x_t, u_t)$$
$$g(x_t, u_t) \leq 0$$
$$x_0 = \hat{x}$$

Maximize tangential distance along track

Minimize deviation from the track

Non-linear bicycle model

Tire forces within friction ellipse

Stay within track boundaries

Practical tips and tricks for MPC

Common problems in MPC for trackdrive:

- Solving the MPC takes time (computation delay):
  - Control action is sent too late!
- Actuators add have a lot of delay
  - The vehicle acts too late!
- MPC solution is too jerky
  - Could break actuators if not careful!

Quick solutions:

1. Measure avg computation delay
2. Integrate measurement before solving MPC

1. Measure actuator delay
2. Choose a control action in the "future" (hacky?)

1. Reformulate problem to use control input rates instead of control inputs

Improving Feedforward: Better Models

- **Example:**
  - **curvature = f(steering)**

How to achieve better models:

- Make fewer assumptions
- Remove simplifications
- Use domain knowledge
- Compensate for system dynamics
- Validate the Model Parameters

Fewer simplifications

$$\kappa = \frac{\delta}{l}$$

$$\kappa = \frac{atan(\delta)}{l}$$

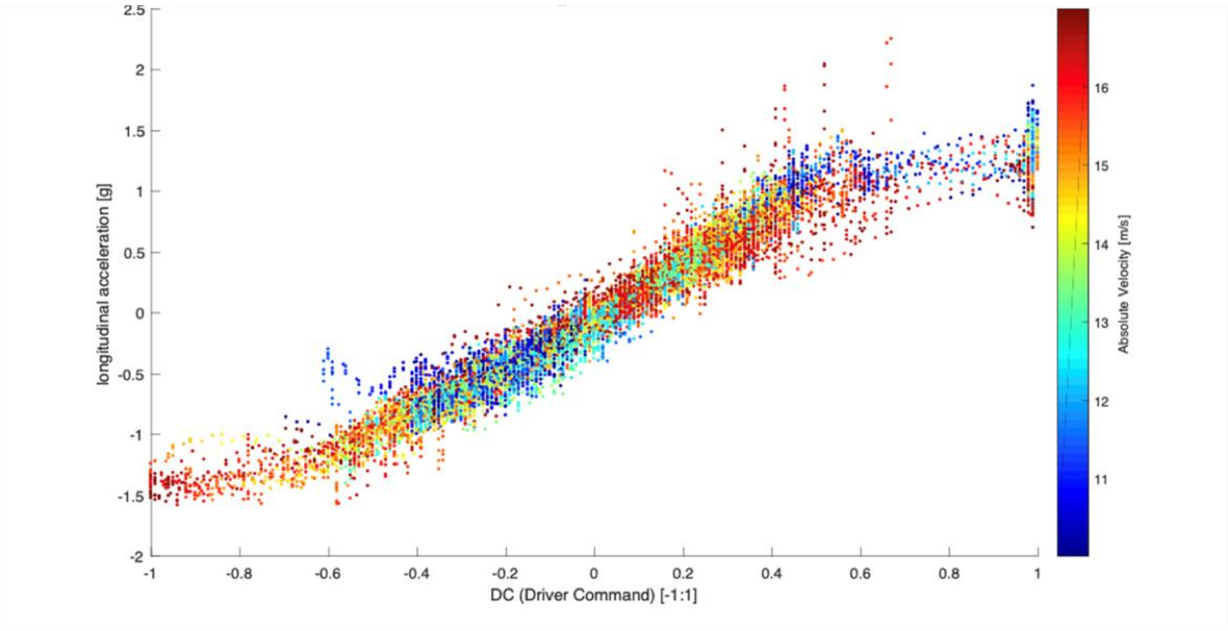
$$\kappa = \frac{atan(\delta)}{l(1+Kv^2)}$$

Improving Feedforward: System Identification

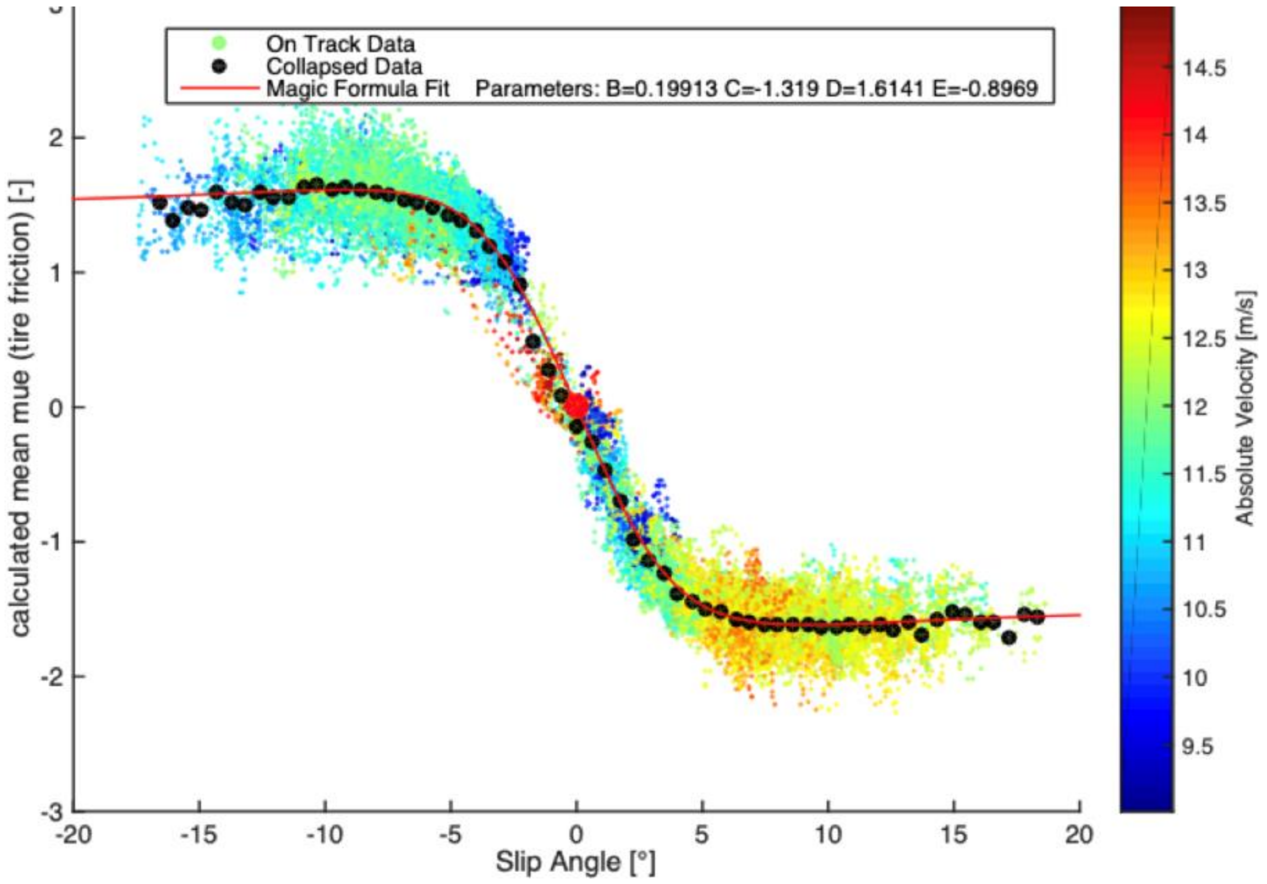
Know your vehicle

- Accurate models for easier control!

Drivetrain Model: command -> acceleration



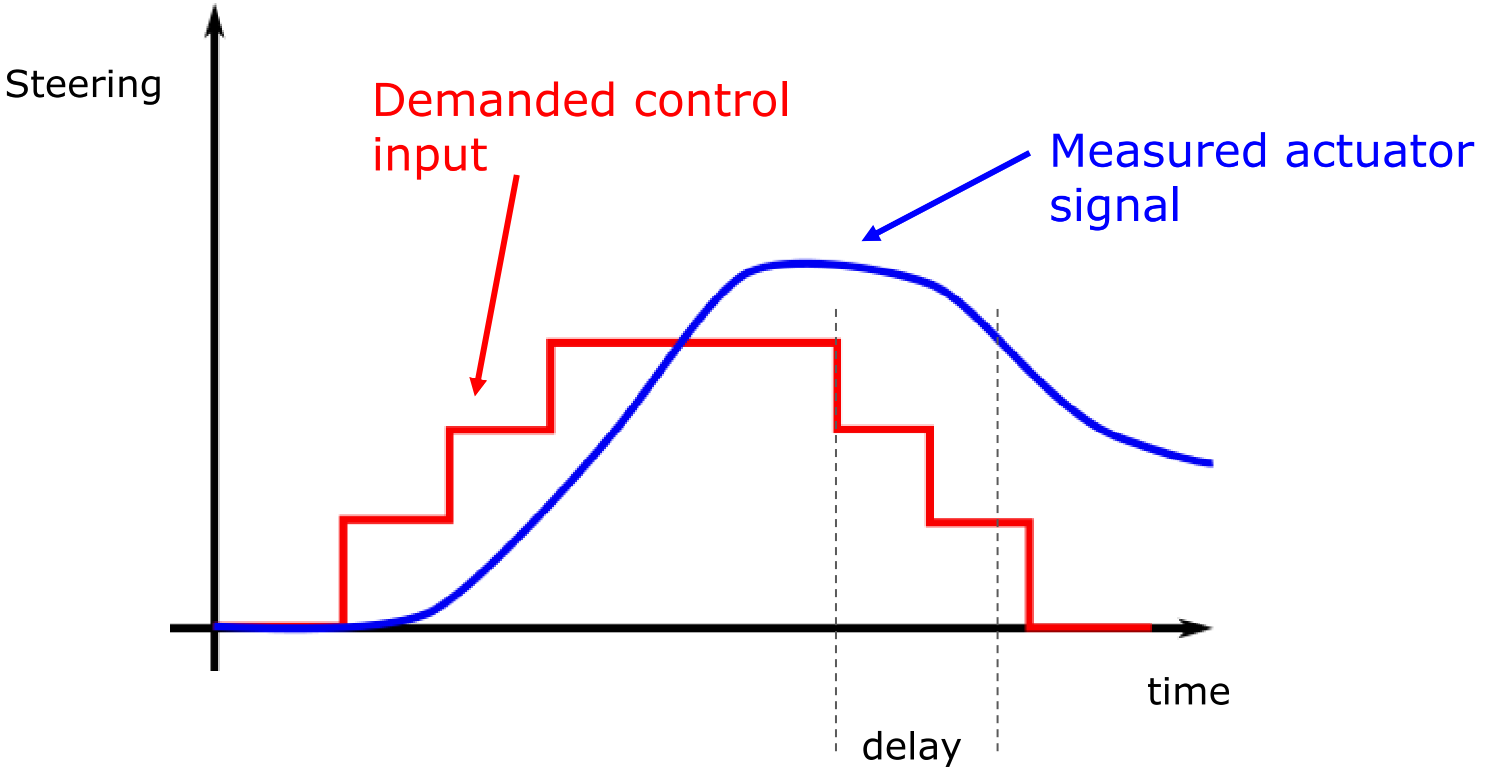
Tire Model fitting from Skidpad data



Improving Feedforward: Better Actuators

How can you improve your actuators?

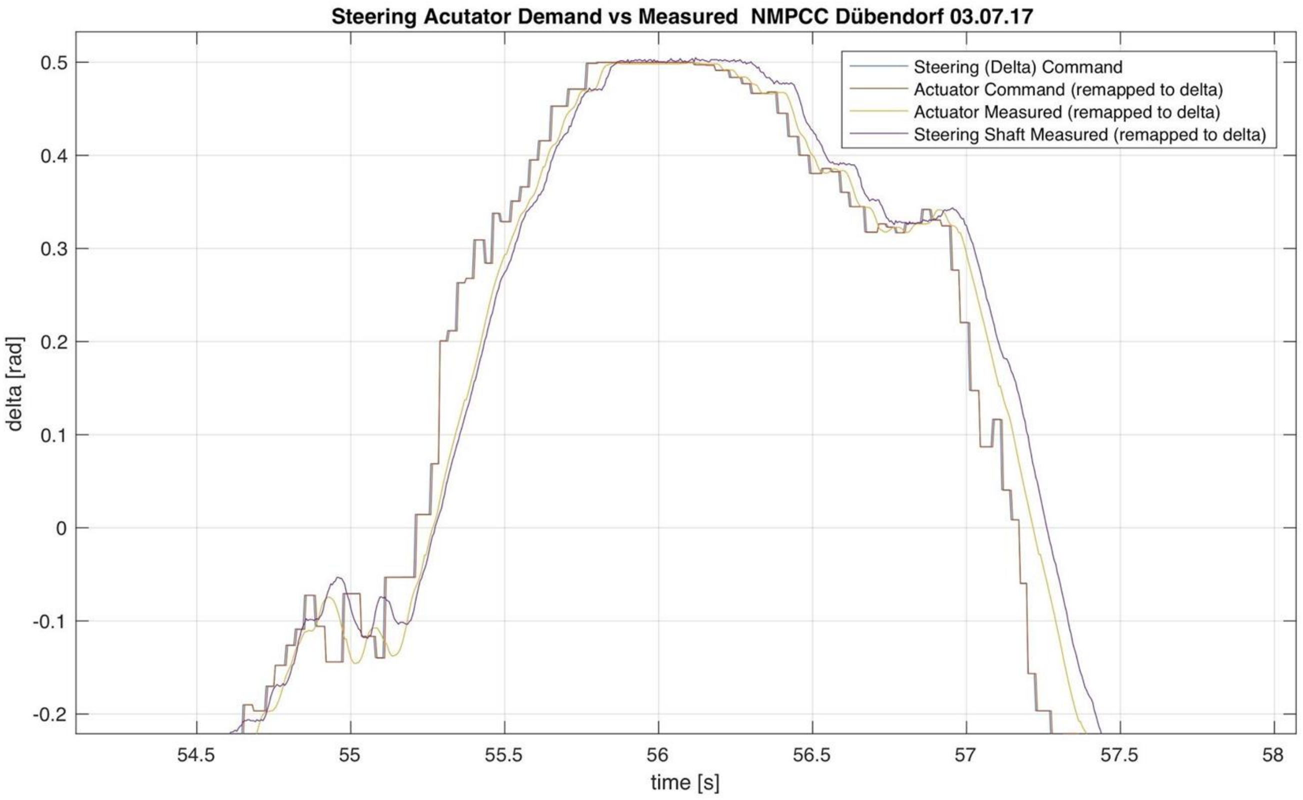
- increase speed
- reduce delay
- no overshoot
- **no backlash!**



Improving Feedforward: Better Actuators

How can you improve your actuators?

- increase speed
- reduce delay
- no overshoot
- **no backlash!**



Improving Feedforward: ML & Estimation

- **Online Parameter Estimation**
  - Grip estimation
- **Supervised ML to improve the model**
  - MPC with Gaussian processes [Kabzan, et al.]
- **Online Model adaptation is adaptive control**
  - Difficult and potentially dangerous
  - Very active research field

[Kabzan, et al.] Learning-based Model Predictive Control for Autonomous Racing

Improving the Feedback: Embed Structure

Use **domain knowledge** to embed the problem structure into the control architecture

- Know some **vehicle dynamics**
- Choose the right variables to control for
- Place controllers on errors whose dynamics are not strongly state dependent
- Example: Stanford Matry [Goh, et al.]

https://dynamicdesignlab.sites.stanford.edu/content/beyond-the-limits

Domain Knowledge Example: Skidpad

Goal: Minimize Skidpad time

- Maximize lateral acceleration given radius
- Control speed given max acceleration

Which control set-point to choose?

- Should we control for target velocity?

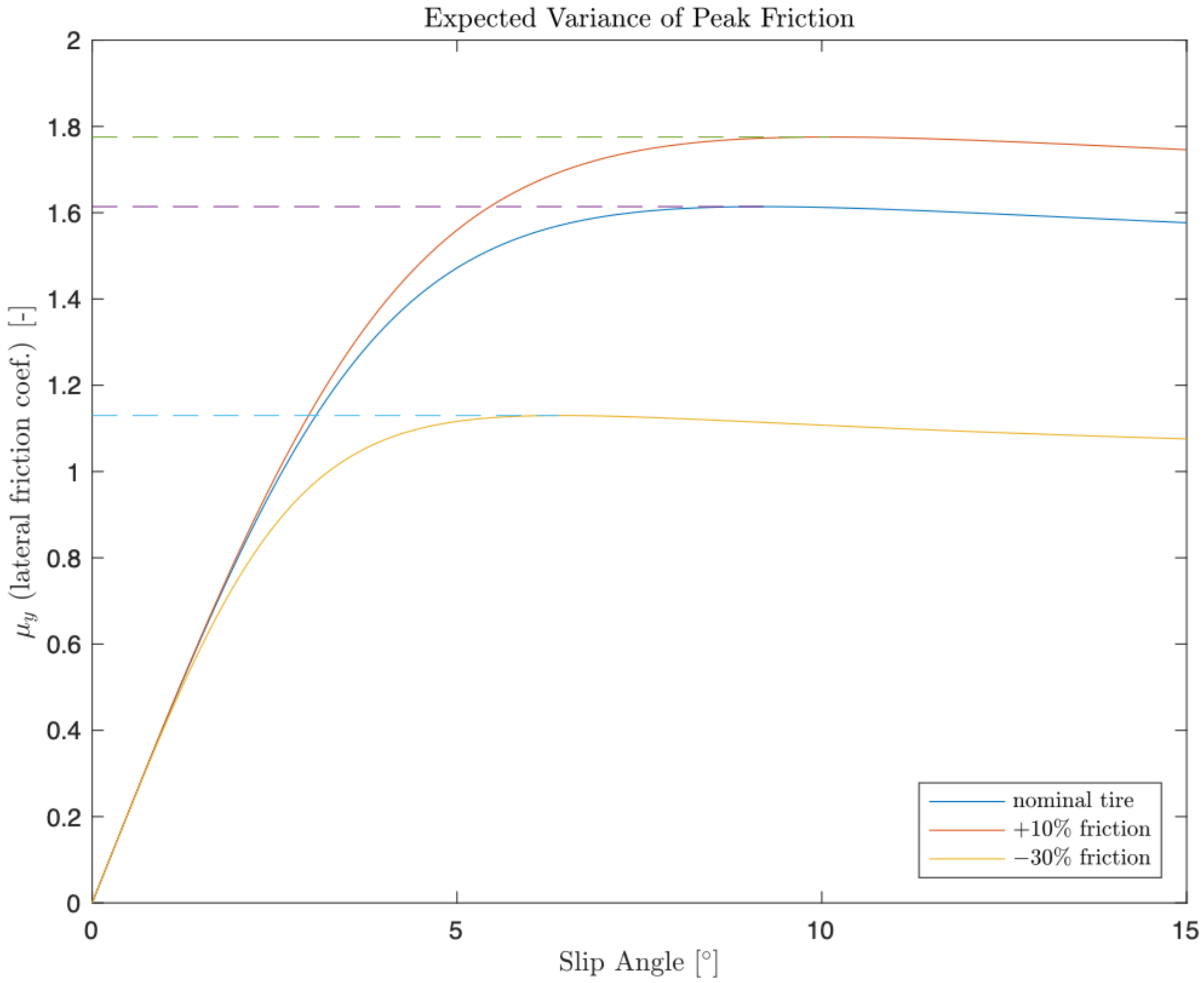
$$v = \sqrt{a_y R}$$

- **Caution!** maximum lateral acceleration is uncertain

Domain Knowledge Example: Skidpad

Caution! Friction level is very uncertain

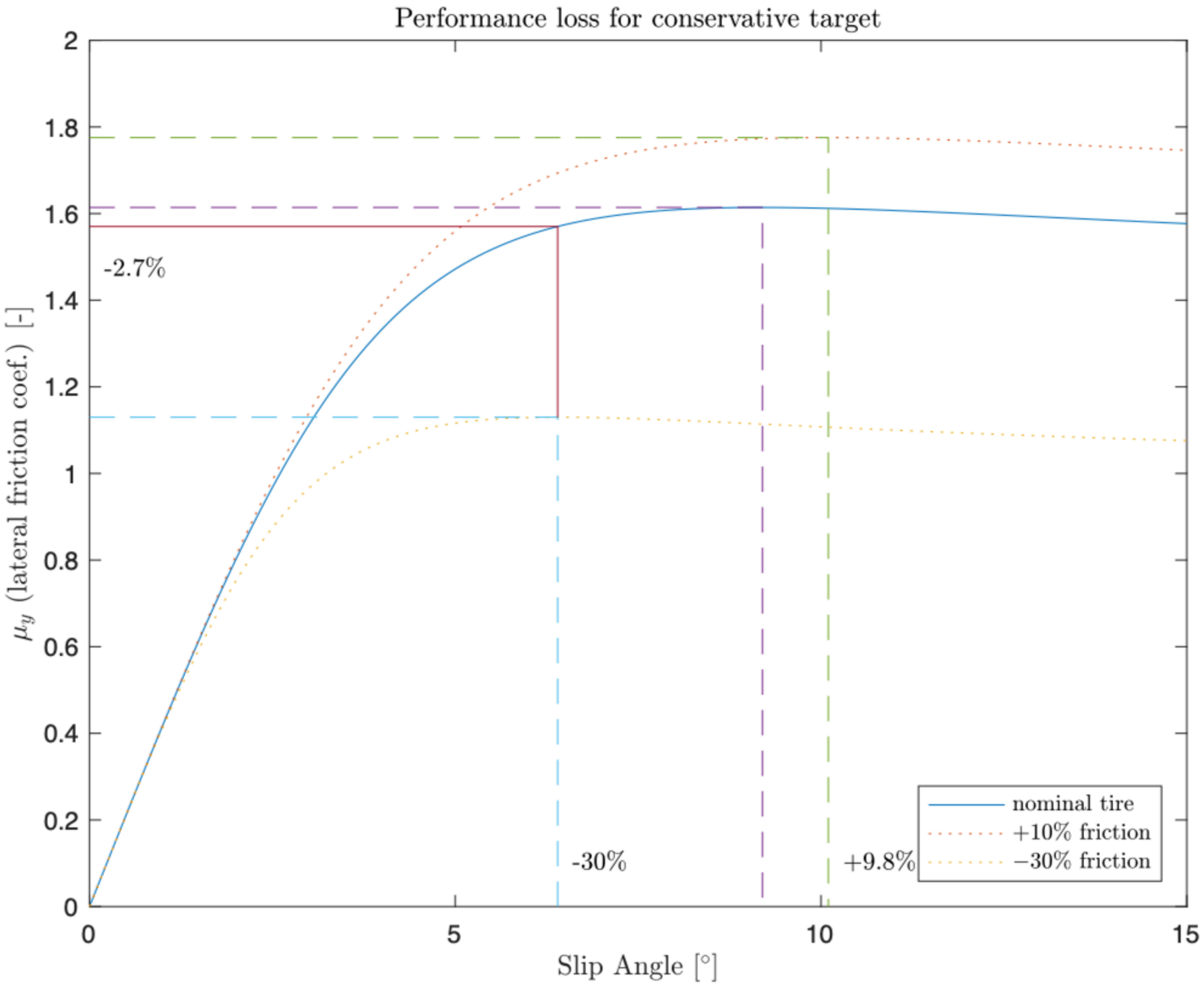
- We need a **not so fast**, conservative velocity target



Domain Knowledge Example: Skidpad

A **better idea** is to use a target **slip angle** instead / additionally

- At the peak, slip angle’s influence on friction is smaller.
- Thus, the **velocity uncertainty** is reduced compared to targeting a peak friction
- **Disadvantage:** Needs accurate slip angle measurement

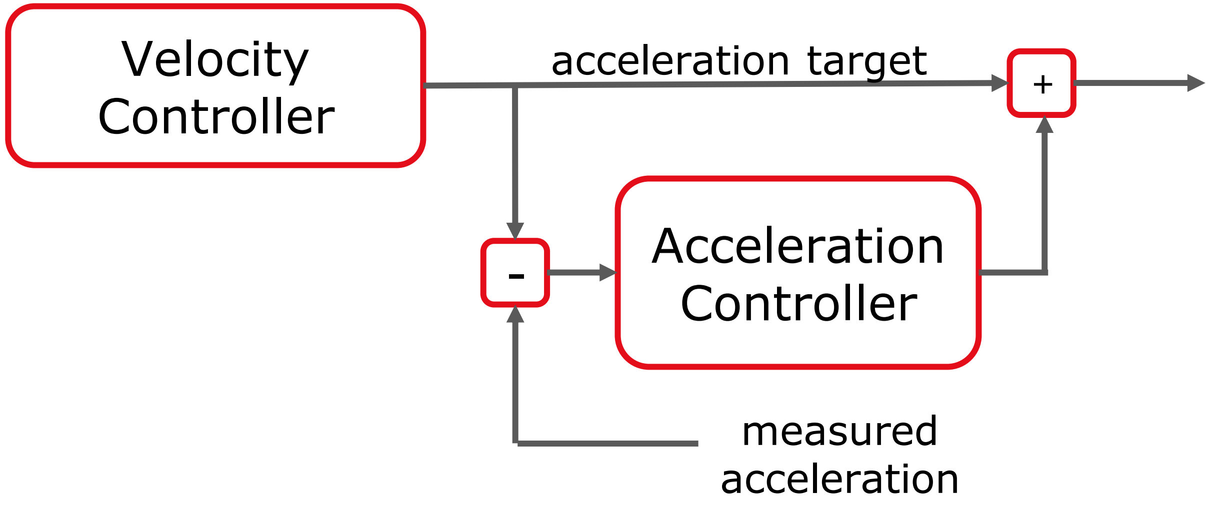


Domain Knowledge Example: Skidpad

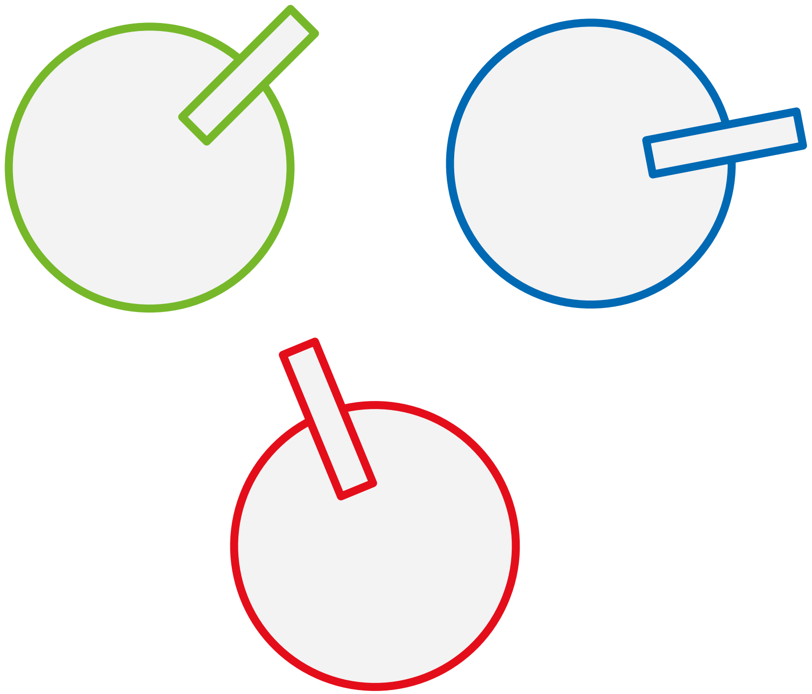
Improving the Feedback: Low-level Controls

Better lower-level controllers will improve your performance

- Good steering and e-motor controllers are crucial!
- **Low-level Controllers** can be used to better track **higher level** signals
  - Cascaded control
  - Longitudinal acceleration
  - Curvature / Yaw-rate



Improving the Feedback: Tune Closed Loop



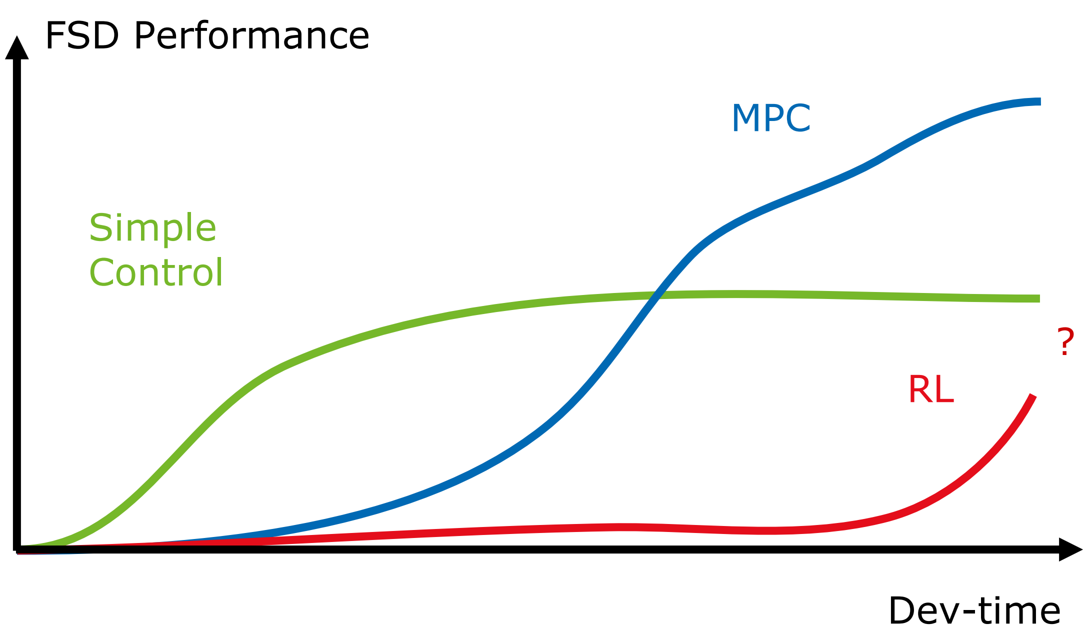
Planning and Controls are the **only** part of the autonomous stack that need testing in closed loop

- Perception and Estimation can be tuned mostly on recorded data
- Control needs **on-track** testing!
- Efficient Testing is important
  - Have **real-time** tuning capability
  - Have **log data** analysis ready
  - Have **visualization** ready
  - Have **simulation** to compare

Reinforcement Learning

RL in FSD is still a **big open question**

- Does it follow the “start simple” approach?
- How do you do reward engineering on track?
  - Track time is expensive
  - Cars are too expensive
- **Sim2Real is hard!**
- **Idea:** RES could be a good query signal for DAGGER (Dataset aggregation)



What kind of controls should you do?

Be aware of the strengths and weaknesses within your team

Look for expertise inside your university:

- This can range from **Vehicle Dynamics** to **Optimal Control** practitioners

Start simple → **Make it work** → **Make it better**

Thank you for your attention