F1TENTH Autonomous Racing

Model Predictive Control

Ahmad Amine
University of Pennsylvania
aminea@seas.upenn.edu

Rahul Mangharam
University of Pennsylvania
rahulm@seas.upenn.edu

F1 TENTH

Penn Engineering

Lecture Outline

1. MPC overview
2. System dynamics review
3. MPC implementation on F1/10

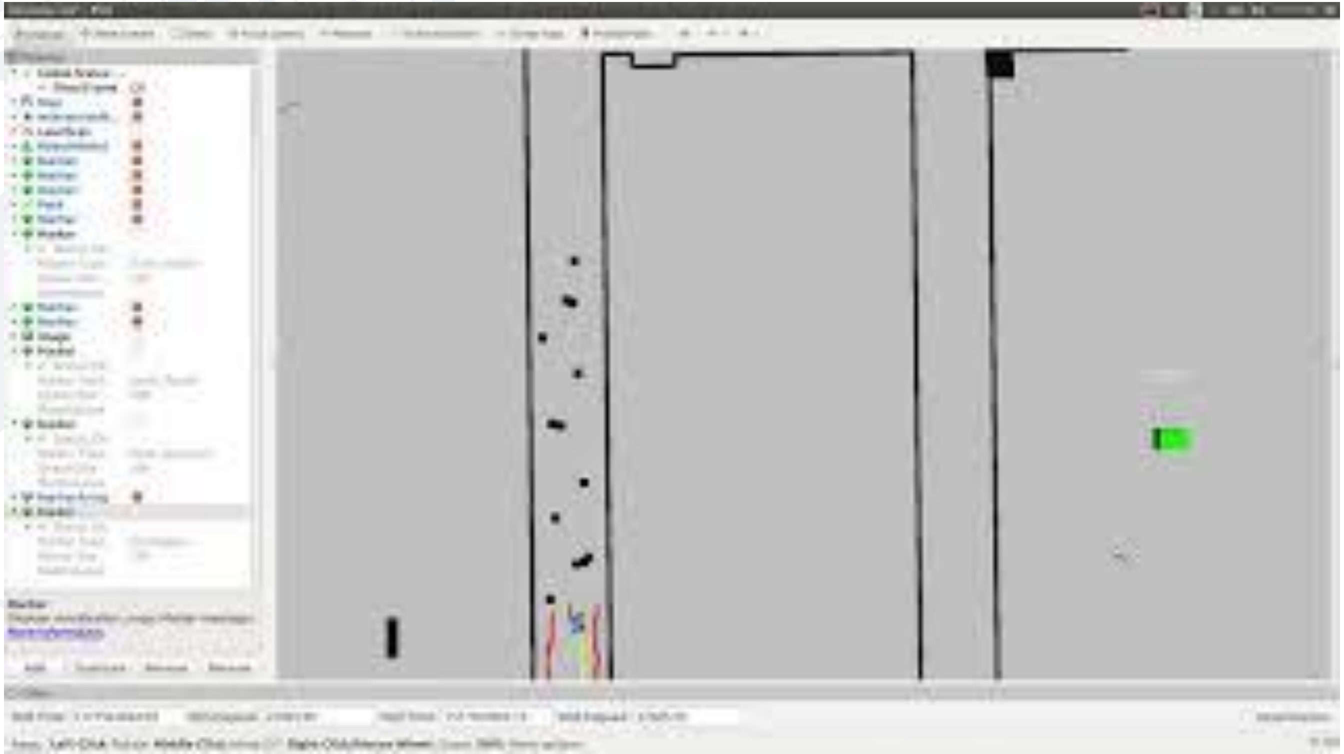
Lecture Outline

1. MPC overview
2. System dynamics review
3. MPC implementation on F1/10

Lecture Outline

1. MPC overview
2. System dynamics review
3. MPC implementation on F1/10

Applications: Trajectory Tracking



Green: reference trajectory Yellow: MPC trajectory Red: Safety constraints

Applications: Autonomous Drifting

https://asmedigitalcollection.asme.org/dynamicsystems/article/142/2/021004/1066044

Applications: Learning MPC

https://arxiv.org/pdf/1901.08184.pdf

PID Drawbacks

$$u(t) = K_\text{p} e(t) + K_\text{i} \int_0^t e(t')\,dt' + K_\text{d} \frac{de(t)}{dt}$$

- A car takes multiple inputs (steering angle, acceleration).
- Independent PID controllers may give dynamically infeasible control commands, e.g., car may flip over.
- E.g. angle = steering angle = $\pi/3$, velocity = 70mph

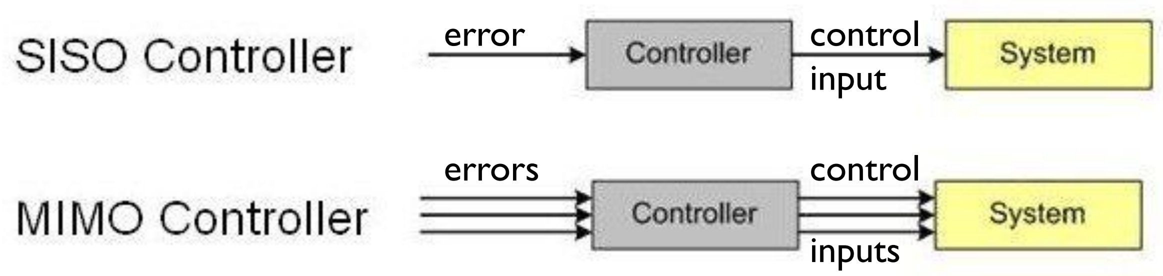


image: quora

PID Drawbacks

$$u(t) = K_\mathrm{p} e(t) + K_\mathrm{i} \int_0^t e(t')\, dt' + K_\mathrm{d} \frac{de(t)}{dt}$$

- Handles **only a single input (e(t)) and a single output (u(t)) (SISO systems)**. E.g. angle error → steering angle input
- Alternative: Use **MIMO Controllers** like LQR

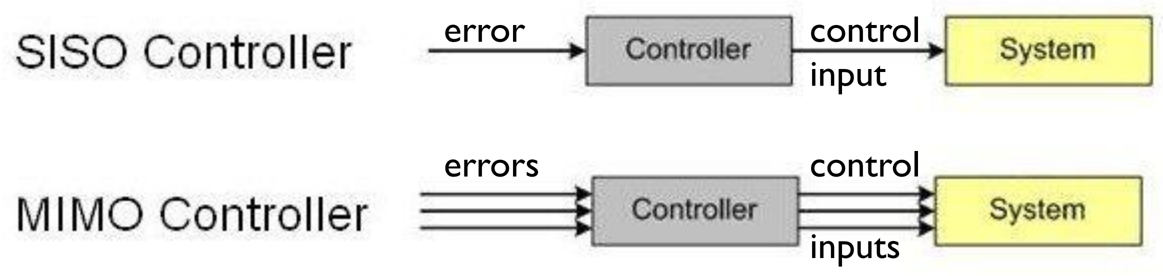
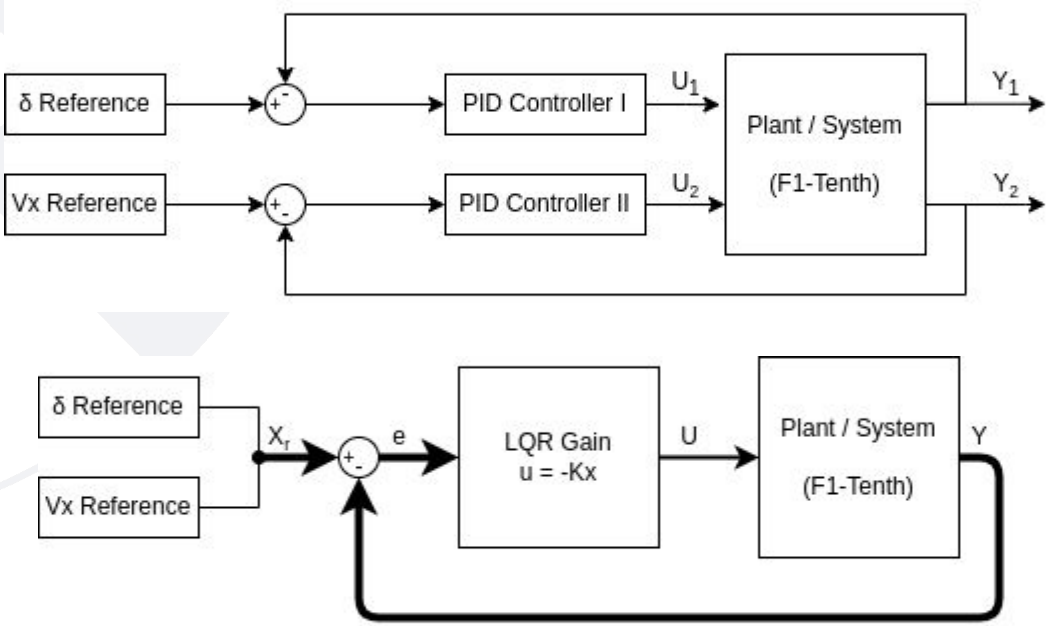


image: quora

Penn Engineering

PID Drawbacks

- MIMO (Multi-Input Multi-Output) VS SISO in PID



LQR Drawbacks

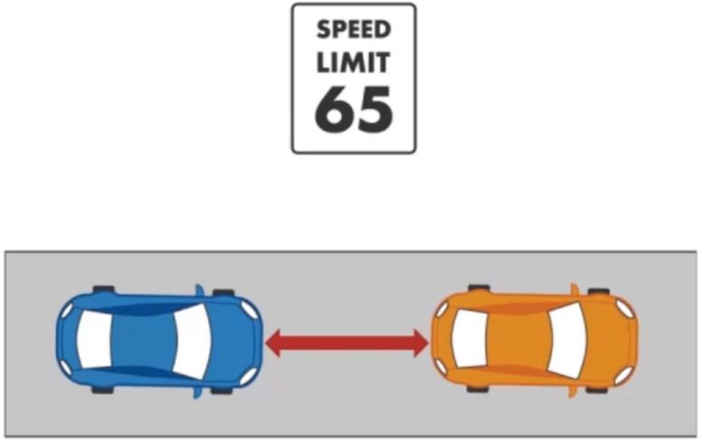
$$u^*(k) = \underbrace{-(B'P_\infty B + R)^{-1}B'P_\infty A}_{F_\infty} x(k), \qquad k = 0, \cdots, \infty$$

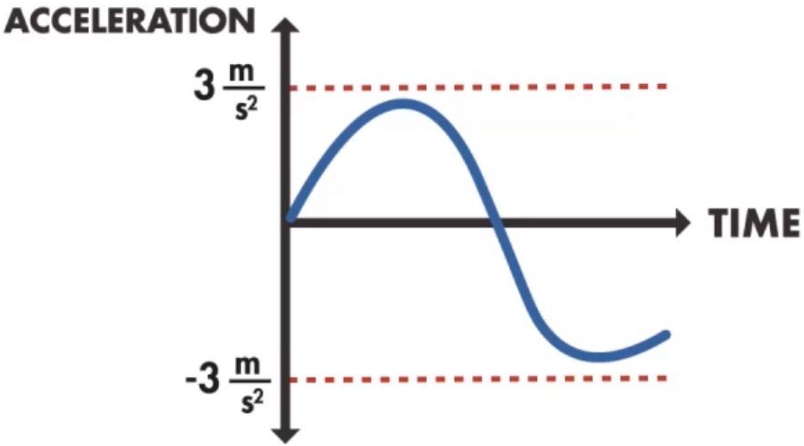
- **Cannot deal with constraints.** May generate impossible control inputs (steering angle = $\pi/2$) for the car to follow.

https://www.youtube.com/watch?v=wqmP-y-a2qY

Why Use MPC for Racing?

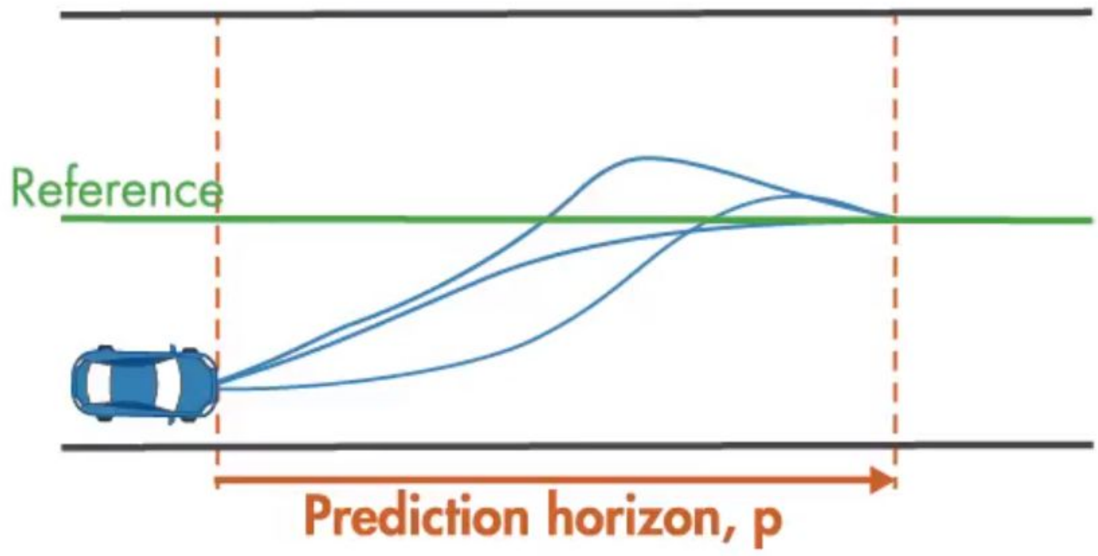
- **Satisfy safety constraints.**(velocity, acceleration, track bounds, etc…)





Why Use MPC for Racing?

- **Satisfy physics constraints.** (i.e vehicle dynamics, dynamically feasible trajectory)



How powerful is MPC?

- **Locally** optimal trajectory.
- Can handle MIMO systems.
- Satisfies controller limits (i.e avoid saturation).
- Can incorporate obstacle avoidance constraints.
- Leverages first-principle (physics) models to ensure dynamical feasibility.
- Can guarantee* stability of control law.

*If certain conditions are met

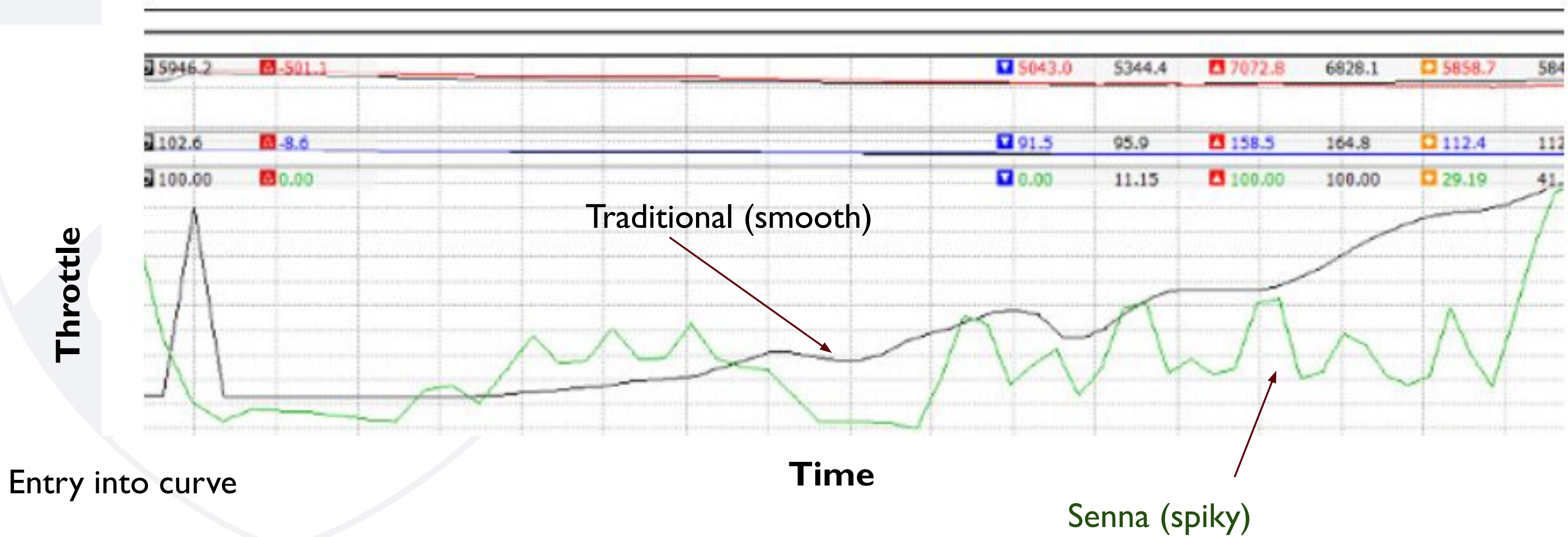
MPC: Humans do it too!

Example: Ayrton Senna's bizzare technique

Figure: Figure from (and detailed analysis at): https://alandovecoaching.wordpress.com/2018/05/27/trying-to-master-sennas-throttle-technique-update/

Senna’s Throttle Technique

A (possible) explanation



Turbocharger lag compensation: Senna had a mental ***model*** of turbocharger behavior, and is ***predictively*** trying to ***maximize*** acceleration upon exiting a curve.

Figure from: https://alandovecoaching.wordpress.com/2018/05/27/trying-to-master-sennas-throttle-technique-update/

if you're curious for more, see: https://www.youtube.com/watch?v=N4kcLyYhThE

MPC: An Optimization Problem

Essentially, we are trying to solve a constrained optimization problem here.

A general setup looks like this

$$
\begin{aligned}
U_t^\star(x(t)) := \operatorname*{argmin}_{U_t} \sum_{k=0}^{N-1} q(x_{t+k}, u_{t+k}) \\
\text{subj. to } x_t &= x(t) && \text{measurement} \\
x_{t+k+1} &= Ax_{t+k} + Bu_{t+k} && \text{system model} \\
x_{t+k} &\in \mathcal{X} && \text{state constraints} \\
u_{t+k} &\in \mathcal{U} && \text{input constraints} \\
U_t &= \{u_t, u_{t+1}, \ldots, u_{t+N-1}\} && \text{optimization variables}
\end{aligned}
$$

k goes from 0 (current time-step) to N (N time-steps ahead of now)

Problem is defined by

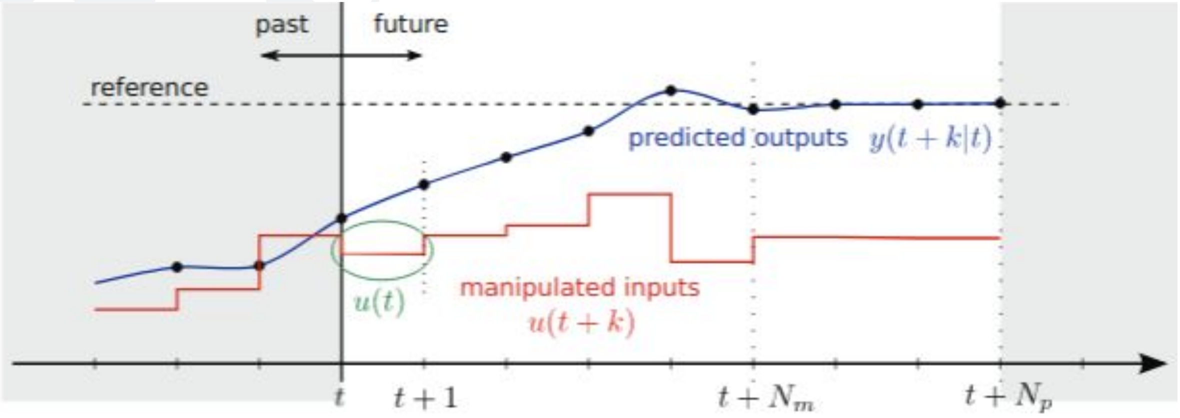
- **Objective** that is minimized e.g., lap time, tracking error, etc.
- Internal **system model** to predict system behavior i.e., vehicle dynamics
- **Constraints** that have to be satisfied e.g., track limits, steering limits etc.

What are the decision variables in context of our MPC problem? What are we trying to solve for?

MPC: Decision Variables

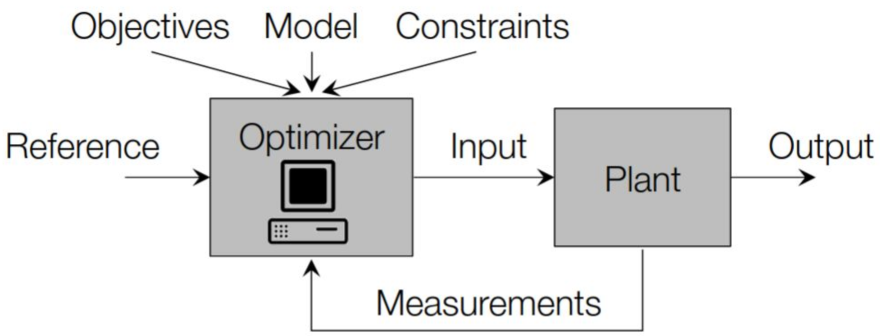
We are trying to solve for a sequence of control actions (and predicted states very often)

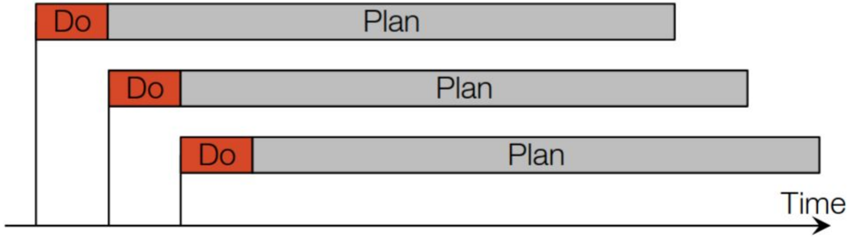
$$[x_0 \ldots x_N \quad u_0 \ldots u_N]^T$$



MPC: Receding Horizon Control

- Get your current state
- The optimizer computes the control sequence that minimizes cost function
- Apply the first input
- Repeat

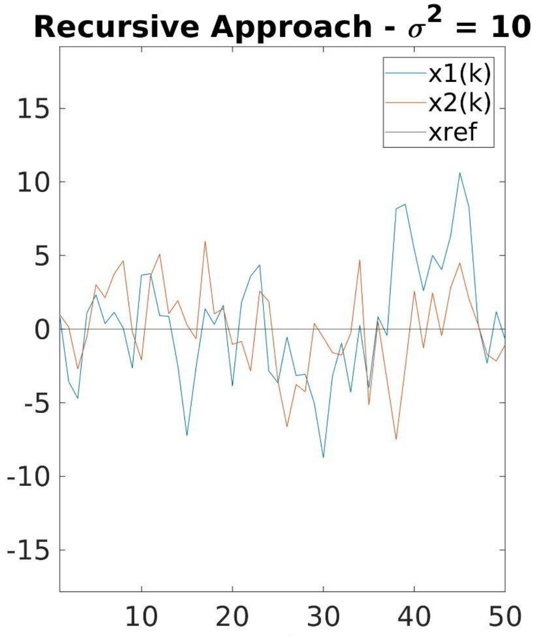


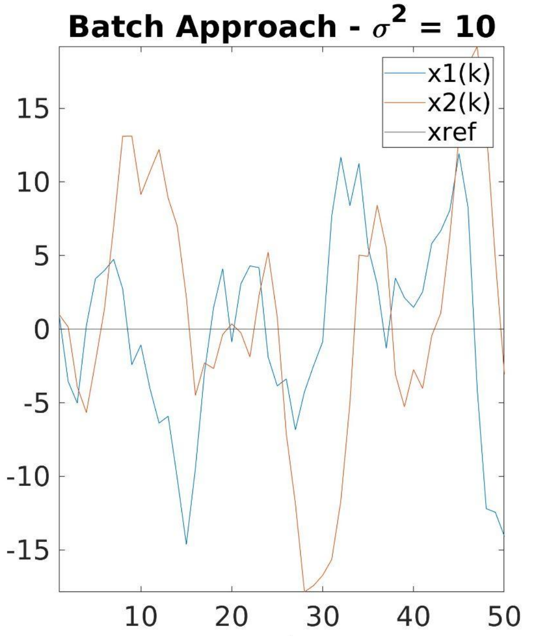


Receding Horizon Control: Advantages

- Allows for a computationally tractable optimization problem as opposed to infinite horizon control.
  - Can be guaranteed to converge to infinite horizon control

- Resulting controller is more robust to disturbances as opposed to open-loop control.





MPC: Cost Function (tracking)

- The cost function (objective) is often divided into two pieces:
  - State Error cost
  - Actuation Effort cost
- Q (Positive semi-definite) and R (Positive Definite) are weights you get to choose.
- Very Often, the cost is formulated as quadratic (L2-norm of error)

Terminal State Error

State Error

State error penalty weights

Control input

$$(x_N - x_r)^T Q_N (x_N - x_r) + \sum_{k=0}^{N-1} (x_k - x_r)^T Q (x_k - x_r) + u_k^T R u_k$$

Actuation effort penalty weights

Why do we have a seperate penalty term for the terminal state?

MPC: Dynamics Constraints

- A model which predicts future states (x), can be time-varying
- Some modern solvers (Casadi) can handle nonlinear dynamics.
- Often you will linearize these dynamics for faster computation
- The linearized dynamics is usually of the form:

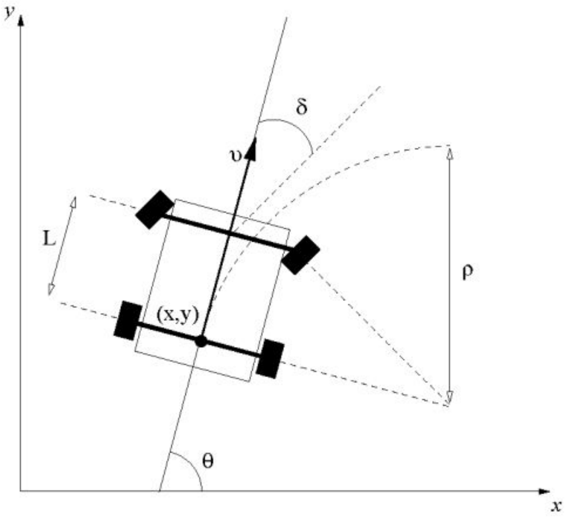
$$x(k+1) = Ax(k) + Bu(k)$$

For each x and u in $\begin{bmatrix} x_0 \ldots x_N & u_0 \ldots u_N \end{bmatrix}^T$

This constraint on two consecutive states makes sure the states sequence is a realistic trajectory (one that satisfy system dynamics)

What if you don't have this constraints on dynamics?

MPC: Dynamics Constraints



Some Nonlinear Dynamics in continuous time:

$$\dot{p}_1 = v \cos \Psi$$

$$\dot{p}_2 = v \sin \Psi$$

$$\dot{\Psi} = \frac{\tan(\delta)}{l_F + l_R} v$$

States: $x = [p_1, p_2, \Psi]$

Control inputs: $u = [v, \delta]$

Linearization and Discretization

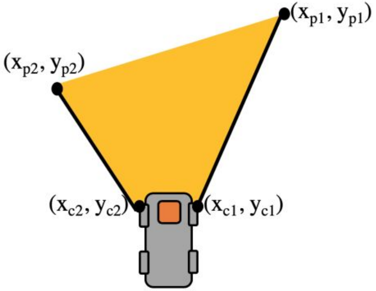
$$x(k+1) = Ax(k) + Bu(k)$$

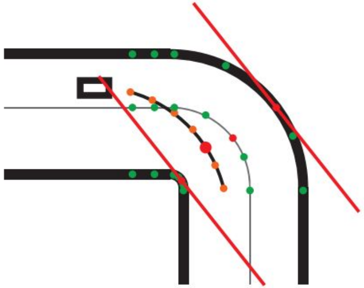
Why is discretization necessary?

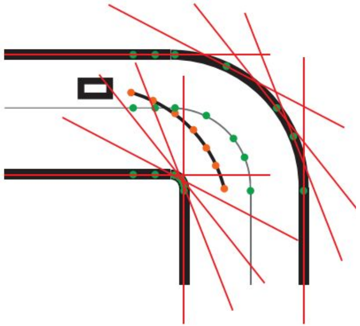
MPC: State and Input Constraints

Inequality Constraints:

- Actuator Limit: Steering angle limits, maximum speed, …
- Track boundaries: can be interpreted as region bounded by a set of lines(half spaces), i.e. Polyhedron.
- Generally written as $x \in X,\ u \in U$ to denote some desired set of states and inputs X and U.
- Can be written as Ax ≤ b, where each row of A, b corresponds to a constraint.







Penn Engineering

MPC: Putting it together

Optimal Cost

Cost Function

$$
\begin{aligned}
J_0^*(x(0)) = \quad & \min_{U_0} \quad J_0(x(0), U_0) \\
& \text{subj. to} \quad x_{k+1} = Ax_k + Bu_k, \ k = 0, \ldots, N-1 \\
& \qquad\qquad x_k \in \mathcal{X}, \ u_k \in \mathcal{U}, \ k = 0, \ldots, N-1 \\
& \qquad\qquad x_N \in \mathcal{X}_f \\
& \qquad\qquad x_0 = x(0)
\end{aligned}
$$

$(Ax \leq b)$

Another equality constraint: enforce x0 in $[x_0 \ldots x_N \quad u_0 \ldots u_N]^T$ to be equal to current measured state!

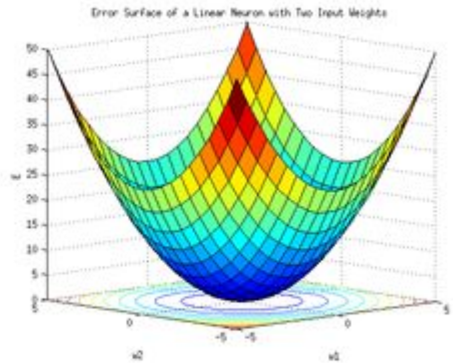
How are we going to solve this optimization problem?

Quadratic Programming Overview

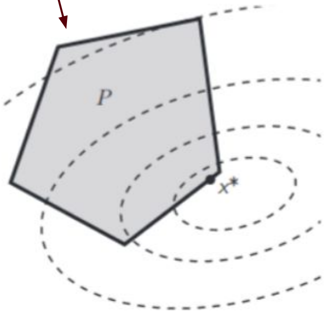
$$\begin{aligned} \min \quad & \frac{1}{2} z^T H z + g^T z \\ \text{s.t.} \quad & lb \le A_c z \le ub \end{aligned}$$

z: n x 1
H: n x n
g: n x 1
A: m x n (m constraints)

Visualization for a two dimensional QP
z = [z1, z2]



linear constraints



Convex! (only one global minimum)

fast to solve!

what happens when the constraints are not convex?

Quadratic Programming Overview

- Not all optimization problems are easy to solve. Most of them are not in fact.
- Our MPC setup is a quadratic programming problem.
- Quadratic Programming (Quadratic Cost & Linear Constraints) is convex: only one global optima exist.
- Can be solved efficiently in real time!
- Many solvers available: CVXGen, OSQP, QuadProg …
- Casadi for non-convex optimization (Matlab, Python, C++ support)
- Multi-Parametric Toolbox (MPT3) for MPC design, analysis (Linear), deployment.
- Recommend OSQP: nice EIGEN interface, easy to use in C++
  https://robotology.github.io/osqp-eigen/doxygen/doc/html/index.html

MPC: QP Formulation

How can we convert our general form MPC formulation to a QP?

MPC

$$
\begin{aligned}
J_0^*(x(0)) = \quad \min_{U_0} \quad & J_0(x(0), U_0) \\
\text{subj. to} \quad & x_{k+1} = Ax_k + Bu_k, \; k = 0, \ldots, N-1 \\
& x_k \in \mathcal{X}, \; u_k \in \mathcal{U}, \; k = 0, \ldots, N-1 \\
& x_N \in \mathcal{X}_f \\
& x_0 = x(0)
\end{aligned}
$$

?

Standard QP

$$
\begin{aligned}
\min \quad & \frac{1}{2} z^T H z + g^T z \\
\text{s.t.} \quad & lb \leq A_c z \leq ub
\end{aligned}
$$

where $z = [x_0 \ldots x_N \quad u_0 \ldots u_N]^T$

MPC: QP Formulation

$$
\begin{aligned}
u_0^* = \arg\min_{x_k, u_k} \quad & (x_N - x_r)^T Q_N (x_N - x_r) + \textstyle\sum_{k=0}^{N-1} (x_k - x_r)^T Q (x_k - x_r) + u_k^T R u_k \\
\text{subject to} \quad & x_{k+1} = A x_k + B u_k \\
& x_{\min} \le x_k \le x_{\max} \\
& u_{\min} \le u_k \le u_{\max} \\
& x_0 = \bar{x}
\end{aligned}
$$

$$\Downarrow$$

The idea is to write this summation in compact matrix form

$$
\begin{aligned}
\min \quad & \frac{1}{2} z^T H z + g^T z \\
\text{s.t.} \quad & lb \le A_c z \le ub
\end{aligned}
$$

MPC: QP Formulation - Cost Function

Most of the work is just index management.

Cost Function: $\frac{1}{2}z^T H z + g^T z$

$$H = \mathrm{diag}(Q, Q, \ldots, Q_N, R, \ldots, R)$$

$$g = \begin{bmatrix} -Qx_r \\ -Qx_r \\ \vdots \\ -Q_N x_r \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

MPC: QP Formulation - Constraints

Constraints:

$$lb \le A_c z \le ub$$

lower bound

$$l = \begin{bmatrix} -x_0 \\ 0 \\ \vdots \\ 0 \\ x_{min} \\ \vdots \\ x_{min} \\ u_{min} \\ \vdots \\ u_{min} \end{bmatrix}$$

$$A_c = \left[\begin{array}{ccccc|cccc} -I & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ A & -I & 0 & \cdots & 0 & B & 0 & \cdots & 0 \\ 0 & A & -I & \cdots & 0 & 0 & B & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & -I & 0 & 0 & \cdots & B \\ \hline I & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & I & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & I & \cdots & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & I & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 & I & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & I & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & I \end{array}\right]$$

upper bound

$$u = \begin{bmatrix} -x_0 \\ 0 \\ \vdots \\ 0 \\ x_{max} \\ \vdots \\ x_{max} \\ u_{max} \\ \vdots \\ u_{max} \end{bmatrix}$$

MPC: Control Law

What does an MPC Control Law look like?

Consider the double integrator

$$\begin{cases} x(t+1) & = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t) \\ y(t) & = \begin{bmatrix} 1 & 0 \end{bmatrix} x(t) \end{cases}$$

subject to constraints

$$-1 \leq u(k) \leq 1, \;\; k = 0, \ldots, 5$$

$$\begin{bmatrix} -10 \\ -10 \end{bmatrix} \leq x(k) \leq \begin{bmatrix} 10 \\ 10 \end{bmatrix}, \;\; k = 0, \ldots, 5$$

Compute the **state feedback** optimal controller $u^*(0)(x(0))$

MPC: Control Law

What does an MPC Control Law look like?

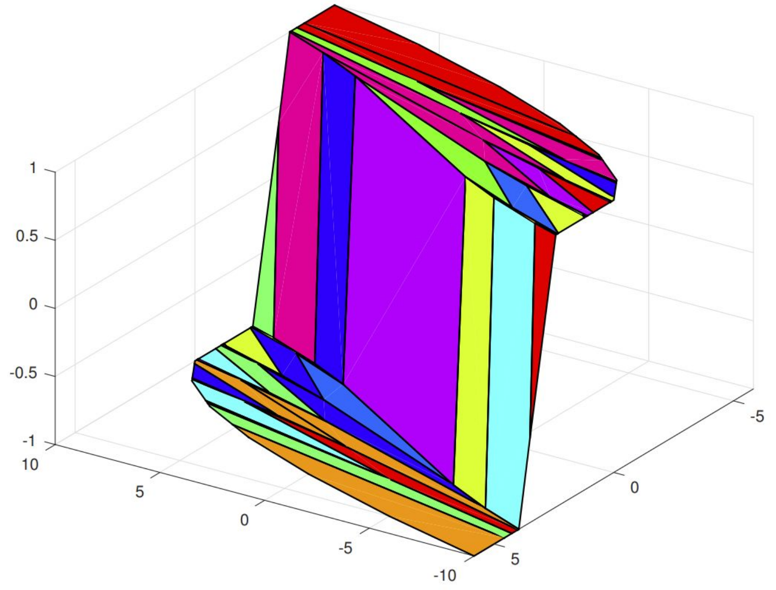


Figure: Optimal control input for the affine control law $u^*(0)$ ($N_0^r = 61$)

Penn Engineering

https://kwonlecture.snu.ac.kr/wp-content/uploads/2019/08/MPC_Course_v1.pdf

Lecture Outline

1. MPC overview
2. System dynamics review
3. MPC implementation on F1/10

System Dynamics Basics

State Space Models

System dynamics can be represented as a vector of ordinary differential equations

continuous-time

$$\dot{x} = g(x, u)$$
$$y = h(x, u)$$

discrete-time

$$x(k+1) = g(x(k), u(k))$$
$$y(k) = h(x(k), u(k))$$

$x \in \mathbb{R}^n$	state vector	$g(x, u) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$	system dynamics
$u \in \mathbb{R}^m$	input vector	$h(x, u) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^p$	output function
$y \in \mathbb{R}^p$	output vector		

Linear Systems

continuous-time

$$\dot{x} = A^c x + B^c u$$

discrete-time

$$x(k+1) = Ax(k) + Bu(k)$$

$x \in \mathbb{R}^n$	state vector	$A^c \in \mathbb{R}^{n \times n}$	system matrix
$u \in \mathbb{R}^m$	input vector	$B^c \in \mathbb{R}^{n \times m}$	input matrix

Forward-Euler Discretization

- Given CT model

$$\dot{x}^c(t) = g^c(x^c(t), u^c(t))$$

- Approximate $\dot{x}^c(t) \approx \frac{x^c(t+T_s)-x^c(t)}{T_s} = \frac{x(k+1)-x(k)}{T_s}$
- T_s is the **sampling time**
- With $u(k) := u^c(t_0 + kT_s)$ the DT model is

$$x(k+1) = x(k) + T_s g^c(x(k), u(k)) = g(x(k), u(k))$$

- Under regularity assumptions, if T_s is small and CT and DT have 'same' initial conditions and inputs, then outputs of CT and DT systems 'will be close'

Exact Discretization

Solution to linear ODEs

- Consider the ODE (written with explicit time dependence) $\dot{x}(t) = A^c x(t) + B^c u(t)$ with initial condition $x_0 := x(t_0)$, then its solution is given by

$$x(t) = e^{A^c(t-t_0)} x_0 + \int_{t_0}^{t} e^{A^c(t-\tau)} B u(\tau) \mathrm{d}\tau$$

where $e^{A^c t} := \sum_{n=0}^{\infty} \frac{(A^c t)^n}{n!}$

Exact Discretization

- Choose $t_0 = t_k$ (hence $x_0 = x(t_0) = x(t_k)$), $t = t_{k+1}$ and use $t_{k+1} - t_k = T_s$ and $u(t) = u(t_k) \;\; \forall t \in [t_k, t_{k+1})$

$$
\begin{aligned}
x(t_{k+1}) &= e^{A^c T_s} x(t_k) + \int_{t_k}^{t_{k+1}} e^{A^c(t_{k+1}-\tau)} B^c \mathrm{d}\tau\, u(t_k) \\
&= \underbrace{e^{A^c T_s}}_{\triangleq A} x(t_k) + \underbrace{\int_0^{T_s} e^{A^c(T_s-\tau')} B^c \mathrm{d}\tau'}_{\triangleq B}\, u(t_k) \\
&= A x(t_k) + B u(t_k)
\end{aligned}
$$

- We found the **exact** discrete-time model predicting the state of the continuous-time system at time t_{k+1} given $x(t_k)$, $k \in \mathbb{Z}_+$ under the assumption of a constant $u(t)$ during a sampling interval
- $B = (A^c)^{-1}(A - I)B^c$, if A^c invertible

Linearization

- **Problem:** Most physical systems are nonlinear but linear systems are much better understood
- Nonlinear systems can be well approximated by a linear system in a 'small' neighborhood around a point in state space
- **Idea:** Control keeps the system around some operating point $\rightarrow$ replace nonlinear by a linearized system around operating point

First order Taylor expansion of $f(\cdot)$ around $\bar{x}$

$$f(x) \approx f(\bar{x}) + \left.\frac{\partial f}{\partial x^\top}\right|_{x=\bar{x}} (x - \bar{x}), \text{ with } \frac{\partial f}{\partial x^\top} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

Linearization

u_s keeps the system around stationary operating point x_s
$\rightarrow \dot{x}_s = g(x_s, u_s) = 0,\ y_s = h(x_s, u_s)$

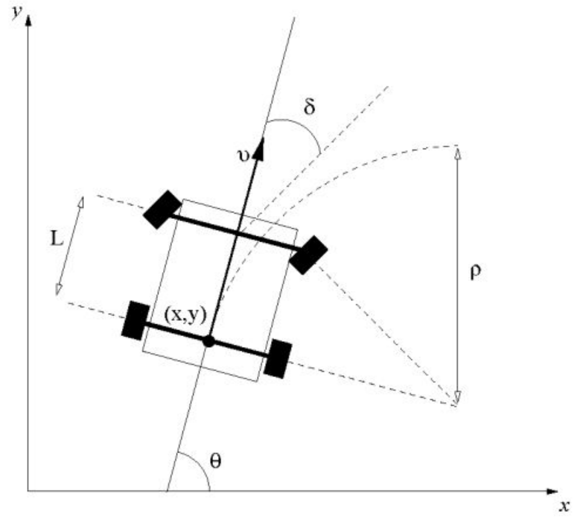
$$
\begin{aligned}
\dot{x} &= \underbrace{g(x_s, u_s)}_{=0} + \underbrace{\left.\frac{\partial g}{\partial x^\top}\right|_{\substack{x=x_s\\u=u_s}}}_{=A^c} \underbrace{(x - x_s)}_{=\Delta x} + \underbrace{\left.\frac{\partial g}{\partial u^\top}\right|_{\substack{x=x_s\\u=u_s}}}_{=B^c} \underbrace{(u - u_s)}_{=\Delta u} \\
\Rightarrow \dot{x} - \underbrace{\dot{x}_s}_{=0} &= \Delta\dot{x} = A^c \Delta x + B^c \Delta u \\
y &= \underbrace{h(x_s, u_s)}_{y_s} + \underbrace{\left.\frac{\partial h}{\partial x^\top}\right|_{\substack{x=x_s\\u=u_s}}}_{=C} (x - x_s) + \underbrace{\left.\frac{\partial h}{\partial u^\top}\right|_{\substack{x=x_s\\u=u_s}}}_{=D} (u - u_s)
\end{aligned}
$$

Why Linear Discrete Systems?

- Linear systems are much better understood than nonlinear systems.

- In context of MPC, linear systems allow us to translate the dynamics as linear constraints in the MPC formulation, which allows us to rewrite the MPC problem as a standard QP.
  - The resulting optimization problem is fast to solve.

Example - Kinematic Bicycle Model



$$\dot{p}_1 = v\cos\Psi$$

$$\dot{p}_2 = v\sin\Psi$$

$$\dot{\Psi} = \frac{\tan(\delta)}{l_F + l_R} v$$

States: $x = [p_1, p_2, \Psi]$

Control inputs: $u = [v, \delta]$

Linearization and Discretization

$$x(k+1) = Ax(k) + Bu(k)$$

Example - Kinematic Bicycle Model

Linearize around a reference trajectory $(\mathbf{x}_r(k), \mathbf{u}_r(k))$

$(\mathbf{x}_r(k), \mathbf{u}_r(k))$

$$A(k) = \frac{\partial f}{\partial \mathbf{x}}\bigg|_{\mathbf{x}_r(k),\mathbf{u}_r(k)} = \begin{bmatrix} 0 & 0 & -vsin(\psi) \\ 0 & 0 & vcos(\psi) \\ 0 & 0 & 0 \end{bmatrix}\Bigg|_{\mathbf{x}_r(k),\mathbf{u}_r(k)}$$

$$B(k) = \frac{\partial f}{\partial \mathbf{u}}\bigg|_{\mathbf{x}_r(k),\mathbf{u}_r(k)} = \begin{bmatrix} cos(\psi) & 0 \\ sin(\psi) & 0 \\ \frac{tan(\delta)}{C_L} & \frac{v}{C_L cos^2(\delta)} \end{bmatrix}\Bigg|_{\mathbf{x}_r(k),\mathbf{u}_r(k)}$$

$$x_{k+1} = A^d(k)x_k + B^d(k)u_k + h^d(k)$$

Lecture Outline

1. MPC overview
2. System dynamics review
3. MPC implementation on F1/10

MPC Implementation on F1/10

Penn Engineering

A Hierarchical Structure

- A high level path planner: chooses a trajectory that maximizes progress from a precomputed trajectory table
- A low level MPC to track the planned trajectory from the path planner.
- This approach is based on the first method described in the MPCC Paper https://arxiv.org/pdf/1711.07300.pdf

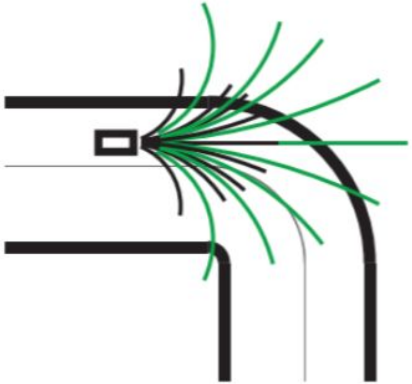
High Level Path Planner

- Grid the stationary velocities and steering angles within their ranges to form a table, where the rows represent steering angles δ and the columns represent speeds v at different increments.

- For each combination of v and δ, a trajectory over a horizon of N time steps can be simulated by integrating the dynamics.

The trajectories will look like this.

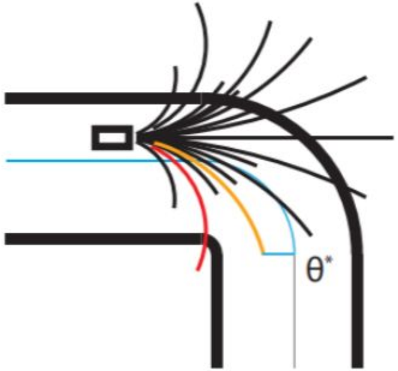
Important: Each trajectory assumes constant speed and constant steering angle.

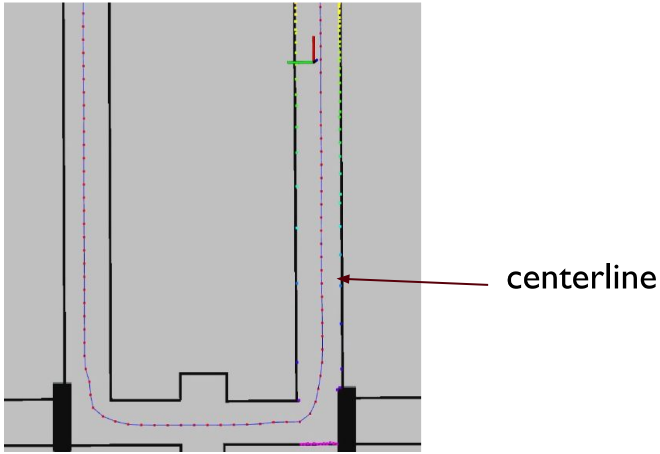


High Level Planner

- Select the trajectory that maximize progress along track centerline (θ)
- To do this, you need to have a centerline beforehand and be able to find projection on centerline.

The red one goes out of bounds, therefore not a candidate.





The orange one makes the most progress when projected on the centerline. Pick this one!

Penn Engineering

MPC: Low-level Tracking Control

State error penalty weights

Terminal State Error

State Error

Control input

$$(x_N - x_r)^T Q_N (x_N - x_r) + \sum_{k=0}^{N-1} (x_k - x_r)^T Q (x_k - x_r) + u_k^T R u_k$$

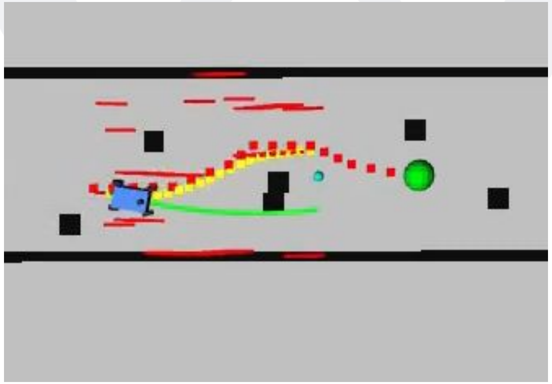
$$\begin{aligned} \text{subj. to} \quad & x_{k+1} = Ax_k + Bu_k, \; k = 0, \ldots, N-1 \\ & x_k \in \mathcal{X}, \; u_k \in \mathcal{U}, \; k = 0, \ldots, N-1 \\ & x_N \in \mathcal{X}_f \\ & x_0 = x(0) \end{aligned}$$

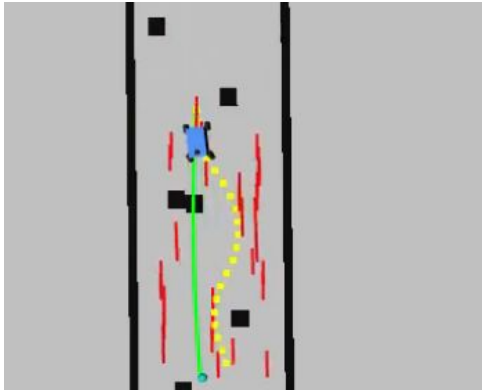
Actuation effort penalty weights

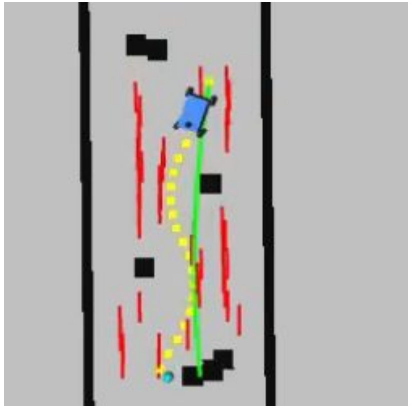
MPC minimizes deviation from the reference trajectory while satisfying all constraints

Hierarchical Planner - Visualized

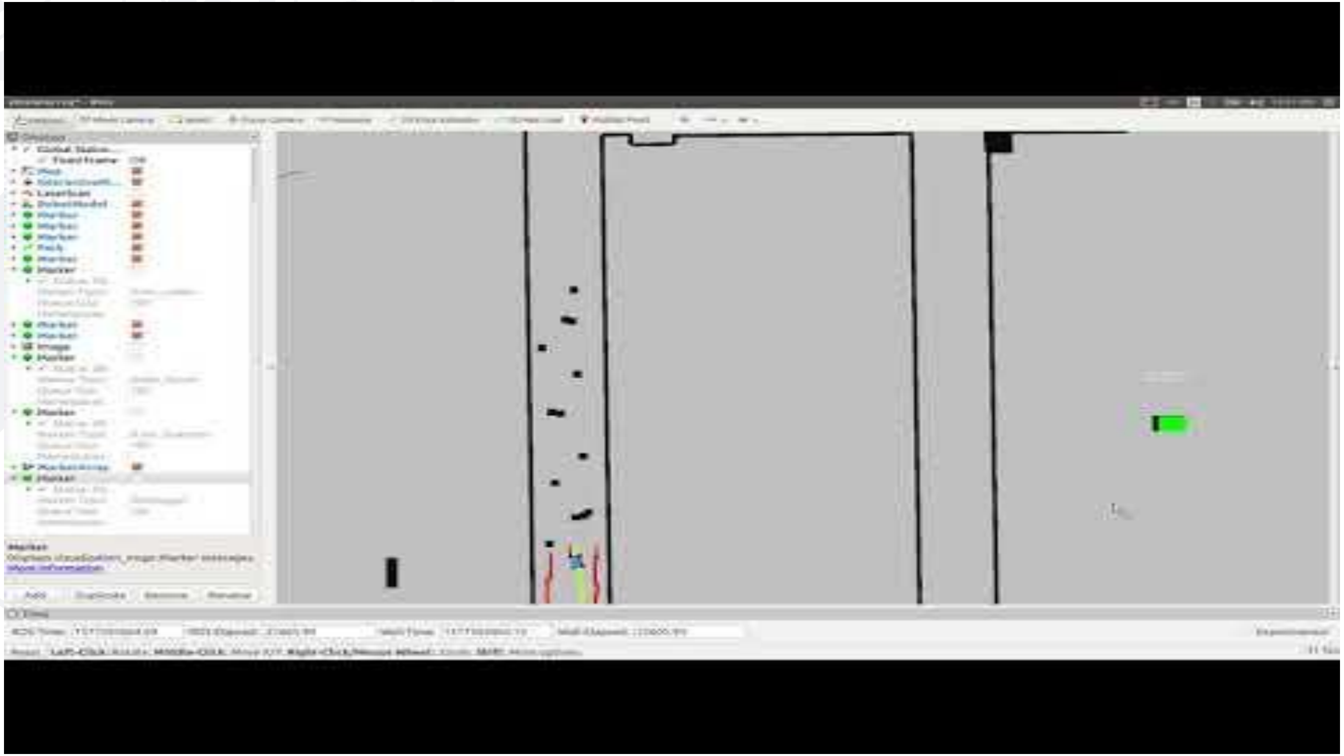
Green line: Reference trajectory
Yellow dots: Solution from MPC which minimizes deviation from reference trajectory
Red lines: Feasible space defined by the region between each pair of parallel redlines (linear half-space constraints)







Demo



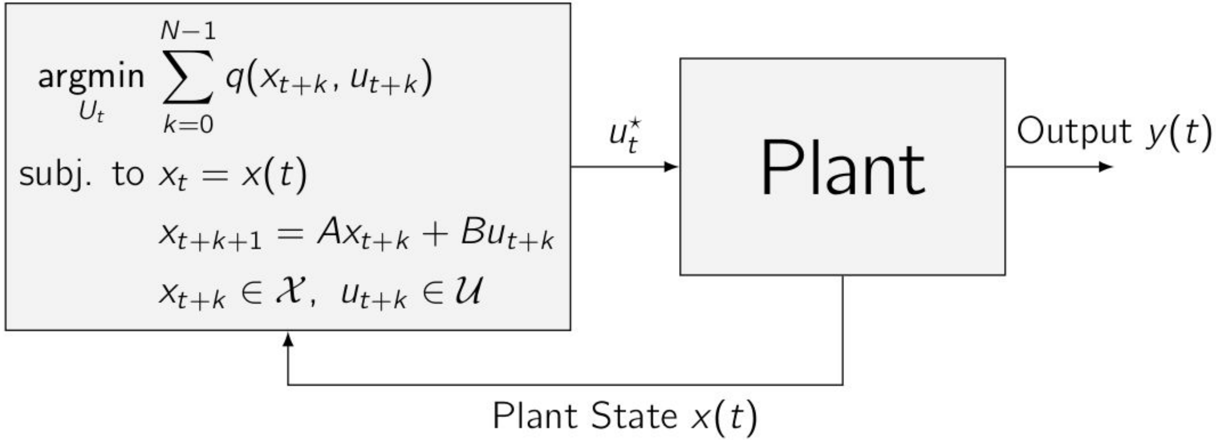
MPC Project Ideas

- Hierarchical Receding Horizon Control
  - Trajectory generator + MPC for trajectory following.
- Model Predictive Contouring Control (MPCC)
  - MPC as a local trajectory planner w.r.t centerline.
- Learning Model Predictive Control
  - Safe Set, minimum time formulation with local linear regression.

See: Lininger, Domahidi, Morari. *Optimization-Based Autonomous Racing of 1:43 Scale RC Cars*, 2017. https://arxiv.org/pdf/1711.07300.pdf

- For obstacle avoidance: incorporate RRT* or A* to adjust half-space constraints for MPC in real time.

Summary: Model Predictive Control



At each sample time:

- Measure /estimate current state $x(t)$
- Find the **optimal input sequence** for the entire planning window N: $U_t^\star = \{u_t^\star, u_{t+1}^\star, \ldots, u_{t+N-1}^\star\}$
- Implement only the **first** control action $u_t^\star$

MPC: Advantages and Limitations

Main Advantages:

- High performance controller that systematically handles constraints.
- Flexible formulation that can incorporate additional objectives.
- Can be formulated for nonlinear system dynamics.
- Can handle time-varying dynamics.

Main Limitations/Challenges:

- Stability is not always guaranteed.
- Feasibility is not always guaranteed.
- Computationally expensive - optimization problem needs to be solved real-time to be used as a controller.
- Robustness to system model errors is not guaranteed.

Practical MPC Tips

- MPC performance is heavily influenced by the model you choose for your vehicle
  - A kinematic model might not be the best for high-speeds.
  - Ensure proper linearization and discretization.
- Avoid using the horizon length N as a tuning parameter:
  - Choose this based on system settling time or computational limits.
- MPC can only be used as a controller if it can be solved real-time:
  - The choice of solver can have a large impact on solve time.
  - Are you using warm start for your optimization loop?
  - Only define the optimization problem once and update its parameters:
    - Optimization problem creation has a large overhead especially in CVXPY.
  - Python is fast enough if you use sparse matrices and good optimization code.