# MODEL PREDICTIVE CONTROL

## LINEAR TIME-VARYING AND NONLINEAR MPC

**Alberto Bemporad**

imt.lu/ab

# COURSE STRUCTURE

- ✔ **Basic concepts** of model predictive control (MPC) and **linear MPC**
- Linear time-varying and nonlinear MPC
- MPC computations: quadratic programming (QP), explicit MPC
- Hybrid MPC
- Stochastic MPC
- Data-driven MPC

**Course page**:

`http://cse.lab.imtlucca.it/~bemporad/mpc_course.html`

# LINEAR TIME-VARYING MODEL PREDICTIVE CONTROL

# LPV MODELS

- **Linear Parameter-Varying (LPV)** model

$$\begin{cases} x_{k+1} &= A(p(t))x_k + B(p(t))u_k + B_v(p(t))v_k \\ y_k &= C(p(t))x_k + D_v(p(t))v_k \end{cases}$$

  that depends on a vector $p(t)$ of parameters

- The weights in the quadratic performance index can also be LPV

- The resulting optimization problem is still a QP

$$\begin{array}{ll} \min\limits_{z} & \frac{1}{2}z'H(p(t))z + \begin{bmatrix} x(t) \\ r(t) \\ u(t-1) \end{bmatrix}' F(p(t))'z \\ \text{s.t.} & G(p(t))z \leq W(p(t)) + S(p(t)) \begin{bmatrix} x(t) \\ r(t) \\ u(t-1) \end{bmatrix} \end{array}$$

- The QP matrices must be constructed online, contrarily to the LTI case

# LINEARIZING A NONLINEAR MODEL: LPV CASE

- An LPV model can be obtained by linearizing the **nonlinear model**

$$\left\{\begin{array}{rcl} \frac{dx_c(t)}{dt} & = & f(x_c(t), u_c(t), p_c(t)) \\ y_c(t) & = & g(x_c(t), p_c(t)) \end{array}\right.$$

- $p_c \in \mathbb{R}^{n_p}$ = a vector of exogenous signals (e.g., ambient conditions)
- At time $t$, let $\bar{x}_c(t), \bar{u}_c(t), \bar{p}_c(t)$ be **nominal values**, that we assume constant in prediction, and linearize

$$\frac{d}{d\tau}(x_c(t+\tau) - \bar{x}_c(t)) = \frac{d}{d\tau}(x_c(t+\tau)) \simeq \underbrace{\left.\frac{\partial f}{\partial x}\right|_{\bar{x}_c(t),\bar{u}_c(t),\bar{p}_c(t)}}_{A_c(t)} (x_c(t+\tau) - \bar{x}_c(t)) +$$

$$\underbrace{\left.\frac{\partial f}{\partial u}\right|_{\bar{x}_c(t),\bar{u}_c(t),\bar{p}_c(t)}}_{B_c(t)} (u_c(t+\tau) - \bar{u}_c(t)) + \underbrace{f(\bar{x}_c(t), \bar{u}_c(t), \bar{p}_c(t))}_{B_{vc}(t)} \cdot 1$$

- Convert $(A_c, [B_c\ B_{vc}])$ to discrete-time and get prediction model $(A, [B\ B_v])$
- Same thing for the output equation to get matrices $C$ and $D_v$

# LTV MODELS

- **Linear Time-Varying (LTV)** model

$$\begin{cases} x_{k+1} & = & A_k(t)x_k + B_k(t)u_k \\ y_k & = & C_k(t)x_k \end{cases}$$

- At each time $t$ the model can also change over the prediction horizon $k$
- The measured disturbance is embedded in the model
- The resulting optimization problem is still a QP

$$\begin{array}{ll} \min\limits_{z} & \frac{1}{2}z'H(t)z + \begin{bmatrix} x(t) \\ r(t) \\ u(t-1) \end{bmatrix}' F(t)'z \\ \text{s.t.} & G(t)z \leq W(t) + S(t)\begin{bmatrix} x(t) \\ r(t) \\ u(t-1) \end{bmatrix} \end{array}$$

- As for LPV-MPC, the QP matrices must be constructed online

# LINEARIZING A NONLINEAR MODEL: LTV CASE

- LPV/LTV models can be obtained by linearizing **nonlinear models**

$$\begin{cases} \frac{dx_c(t)}{dt} & = & f(x_c(t), u_c(t), p_c(t)) \\ y_c(t) & = & g(x_c(t), p_c(t)) \end{cases}$$

- At time $t$, consider **nominal trajectories**

$$U = \{\bar{u}_c(t), \bar{u}_c(t+T_s), \ldots, \bar{u}_c(t+(N-1)T_s)\}$$

(example: $U$ = shifted previous optimal sequence or input ref. trajectory)

$$P = \{\bar{p}_c(t), \bar{p}_c(t+T_s), \ldots, \bar{p}_c(t+(N-1)T_s)\}$$

(no preview: $\bar{p}_c(t+k) \equiv \bar{p}_c(t)$)

- **Integrate** the model and get nominal state/output trajectories

$$\begin{aligned} X &= \{\bar{x}_c(t), \bar{x}_c(t+T_s), \ldots, \bar{x}_c(t+(N-1)T_s)\} \\ Y &= \{\bar{y}_c(t), \bar{y}_c(t+T_s), \ldots, \bar{y}_c(t+(N-1)T_s)\} \end{aligned}$$

- Examples: $\bar{x}_c(t) =$ current state / equilibrium state / reference state

- While integrating, also compute the **sensitivities**

$$\begin{aligned} A_k(t) &= \frac{\partial \bar{x}_c(t+(k+1)T_s)}{\partial \bar{x}_c(t+kT_s)} \\ B_k(t) &= \frac{\partial \bar{x}_c(t+(k+1)T_s)}{\partial \bar{u}_c(t+kT_s)} \\ C_k(t) &= \frac{\partial \bar{y}_c(t+kT_s)}{\partial \bar{x}_c(t+kT_s)} \end{aligned}$$

- Approximate the NL model as the LTV model

$$\left\{ \begin{aligned} \overbrace{x_c(k+1)-\bar{x}_c(k+1)}^{x_{k+1}} &= A_k(t)\overbrace{(x_c(k)-\bar{x}_c(k))}^{x_k} + B_k(t)\overbrace{(u_c(k)-\bar{u}_c(k))}^{u_k} \\ \underbrace{y_c(k)-\bar{y}_c(k)}_{y_k} &= C_k(t)\underbrace{(x_c(k)-\bar{x}_c(k))}_{x_k} \end{aligned} \right.$$

(the notation "$(k)$" is a shortcut for "$(t+kT_s)$")

# LINEARIZATION AND TIME-DISCRETIZATION

- Getting the discrete-time LTV model $A_k(t), B_k(t), C_k(t)$ requires **linearizing** and **discretizing** in time the nonlinear continuous-time dynamical model

$$\frac{dx_c(t)}{dt} = f(x_c, u_c, p_c) \approx \underbrace{f(\bar{x}_c, \bar{u}_c, \bar{p}_c)}_{\frac{d\bar{x}_c}{dt}} + \underbrace{\left.\frac{\partial f}{\partial x_c}\right|_{\bar{x}_c, \bar{u}_c, \bar{p}_c}}_{\text{Jacobian matrix } A_c} (x_c - \bar{x}_c) + \underbrace{\left.\frac{\partial f}{\partial u_c}\right|_{\bar{x}_c, \bar{u}_c, \bar{p}_c}}_{\text{Jacobian matrix } B_c} (u_c - \bar{u}_c)$$

- Let $x = x_c - \bar{x}_c, u = u_c - \bar{u}_c$. We get the continuous-time linear system

$$\frac{dx}{dt} = A_c x + B_c u$$

- Similarly, we linearize the output equation and get

$$y = y_c - \bar{y}_c \approx \underbrace{\left.\frac{\partial g}{\partial x_c}\right|_{\bar{x}_c, \bar{u}_c, \bar{p}_c}}_{\text{Jacobian matrix } C} x$$

- The continuous-time linear system $(A_c, B_c, C)$ can be converted to a discrete-time system $(A, B, C)$ with sample time $T_s$

- **Forward Euler method**

$$\begin{aligned}\bar{x}_c(k+1) &= \bar{x}_c(k)+T_s f(\bar{x}_c(k),\bar{u}_c(k),\bar{p}_c(k))\\ A(k) &= I+T_sA_c(k)\\ B(k) &= T_sB_c(k)\end{aligned}$$

Leonhard Paul Euler
(1707-1783)

- For improved accuracy we can use smaller integration steps $\frac{T_s}{N}$, $N\geq 1$:

1. $x=\bar{x}_c(k), A=I, B=0$
2. for $n=1$ to $N$ do
    - $A \leftarrow \left(I+\frac{T_s}{N}\frac{\partial f}{\partial x_c}(x,\bar{u}_c(k),\bar{p}_c(k)\right)A$
    - $B \leftarrow \left(I+\frac{T_s}{N}\frac{\partial f}{\partial x_c}(x,\bar{u}_c(k),\bar{p}_c(k))\right)B+\frac{T_s}{N}\frac{\partial f}{\partial u}(x,\bar{u}_c(k),\bar{p}_c(k))$
    - $x \leftarrow x+\frac{T_s}{N}f(x,\bar{u}_c(k),\bar{p}_c(k))$
3. return $\bar{x}_c(k+1)\approx x$ and matrices $A(k)=A$, $B(k)=B$

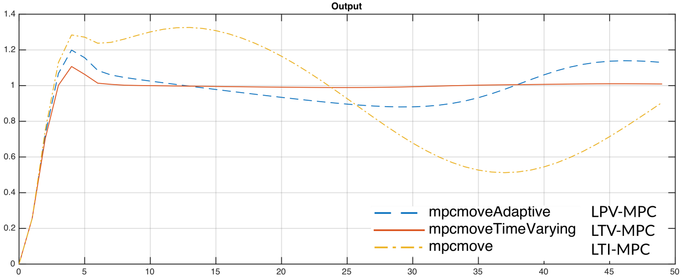
- Note that integration, linearization, and time-discretization are combined
- See also references in (Gros, Zanon, Quirynen, Bemporad, Diehl, 2020)

# LTV-MPC EXAMPLE

- Process model is LTV

$$\frac{d^3y}{dt^3} + 3\frac{d^2y}{dt^2} + 2\frac{dy}{dt} + (6 + \sin(5t))y = 5\frac{du}{dt} + \left(5 + 2\cos\left(\frac{5}{2}t\right)\right)u$$




- LTI-MPC cannot track the setpoint, LPV-MPC tries to catch-up with time-varying model, LTV-MPC has preview on future model values

```
>> openExample('mpc/TimeVaryingMPCControlOfATimeVaryingLinearSystemExample')
```

# LTV-MPC EXAMPLE

- Define LTV model

```
Models = tf; ct = 1;
for t = 0:0.1:10
    Models(:,:,ct) = tf([5 5+2*cos(2.5*t)],[1 3 2 6+sin(5*t)]);
    ct = ct + 1;
end

Ts = 0.1; % sampling time
Models = ss(c2d(Models,Ts));
```

- Design MPC controller

```
sys = ss(c2d(tf([5 5],[1 3 2 6]),Ts)); % average model time
p = 3; % prediction horizon
m = 3; % control horizon
mpcobj = mpc(sys,Ts,p,m);

mpcobj.MV = struct('Min',-2,'Max',2); % input constraints
mpcobj.Weights = struct('MV',0,'MVRate',0.01,'Output',1);
```

# LTV-MPC EXAMPLE

- Simulate LTV system with **LTI-MPC** controller

```
for ct = 1:(Tstop/Ts+1)
    real_plant = Models(:,:,ct); % Get the current plant
    y = real_plant.C*x;
    u = mpcmove(mpcobj,xmpc,y,1); % Apply LTI MPC
    x = real_plant.A*x + real_plant.B*u;
end
```

- Simulate LTV system with **LPV-MPC** controller

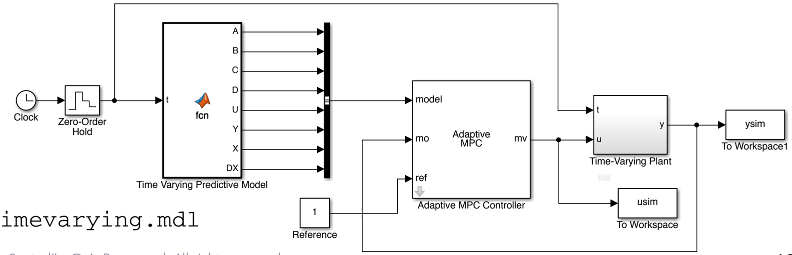
```
for ct = 1:(Tstop/Ts+1)
    real_plant = Models(:,:,ct); % Get the current plant
    y = real_plant.C*x;
    u = mpcmoveAdaptive(mpcobj,xmpc,real_plant,nominal,y,1);
    x = real_plant.A*x + real_plant.B*u;
end
```

# LTV-MPC EXAMPLE

- Simulate LTV system with **LTV-MPC** controller

```
for ct = 1:(Tstop/Ts+1)
    real_plant = Models(:,:,ct); % Get the current plant
    y = real_plant.C*x;
    u = mpcmoveAdaptive(mpcobj,xmpc,Models(:,:,ct:ct+p), ...
          Nominals,y,1);
    x = real_plant.A*x + real_plant.B*u;
end
```

- Simulate in Simulink

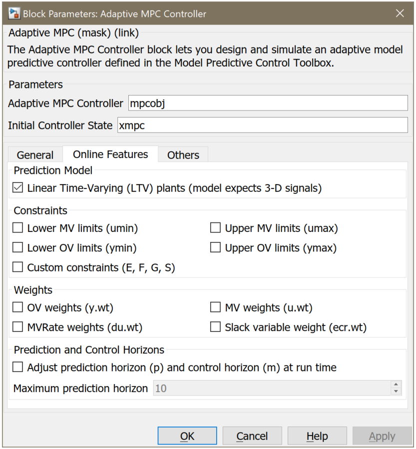



mpc_timevarying.mdl

# LTV-MPC EXAMPLE

- Simulink block

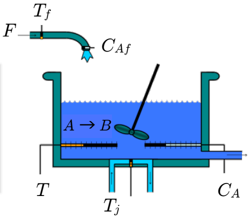
need to provide 3D array
of future models




`mpc_timevarying.mdl`

# EXAMPLE: LPV-MPC OF A NONLINEAR CSTR SYSTEM

- MPC control of a diabatic **continuous stirred tank reactor (CSTR)**
- Process model is nonlinear




$$
\begin{aligned}
\frac{dC_A}{dt} &= \frac{F}{V}(C_{Af} - C_A) - C_A k_0 e^{-\frac{\Delta E}{RT}} \\
\frac{dT}{dt} &= \frac{F}{V}(T_f - T) + \frac{UA}{\rho C_p V}(T_j - T) - \frac{\Delta H}{\rho C_p} C_A k_0 e^{-\frac{\Delta E}{RT}}
\end{aligned}
$$

  - $T$ : temperature inside the reactor $[K]$ (state)
  - $C_A$ : concentration of the reactant in the reactor $[kgmol/m^3]$ (state)
  - $T_j$ : jacket temperature $[K]$ (input)
  - $T_f$ : feedstream temperature $[K]$ (measured disturbance)
  - $C_{Af}$ : feedstream concentration $[kgmol/m^3]$ (measured disturbance)
- Objective: **manipulate** $T_j$ to **regulate** $C_A$ on desired setpoint

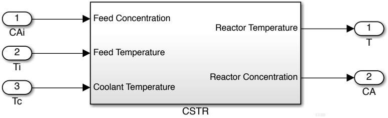
```
>> edit ampccstr_linearization
```

(MPC Toolbox)

# EXAMPLE: LPV-MPC OF A NONLINEAR CSTR SYSTEM

Process model:

```
>> mpc_cstr_plant
```




```
% Create operating point specification.
plant_mdl = 'mpc_cstr_plant';
op = operspec(plant_mdl);

op.Inputs(1).u = 10; % Feed concentration known @initial condition
op.Inputs(1).Known = true;
op.Inputs(2).u = 298.15; % Feed concentration known @initial condition
op.Inputs(2).Known = true;
op.Inputs(3).u = 298.15; % Coolant temperature known @initial condition
op.Inputs(3).Known = true;

 [op_point, op_report] = findop(plant_mdl,op); % Compute initial condition

x0 = [op_report.States(1).x;op_report.States(2).x];
y0 = [op_report.Outputs(1).y;op_report.Outputs(2).y];
u0 = [op_report.Inputs(1).u;op_report.Inputs(2).u;op_report.Inputs(3).u];

% Obtain linear plant model at the initial condition.
sys = linearize(plant_mdl, op_point);
sys = sys(:,2:3); % First plant input CAi dropped because not used by MPC
```

# EXAMPLE: LPV-MPC OF A NONLINEAR CSTR SYSTEM

- MPC design

```
% Discretize the plant model
Ts = 0.5; % hours
plant = c2d(sys,Ts);

% Design MPC Controller

% Specify signal types used in MPC
plant.InputGroup.MeasuredDisturbances = 1;
plant.InputGroup.ManipulatedVariables = 2;
plant.OutputGroup.Measured = 1;
plant.OutputGroup.Unmeasured = 2;
plant.InputName = 'Ti','Tc';
plant.OutputName = 'T','CA';

% Create MPC controller with default prediction and control horizons
mpcobj = mpc(plant);

% Set nominal values in the controller
mpcobj.Model.Nominal = struct('X', x0, 'U', u0(2:3), 'Y', y0, 'DX', [0 0]);
```

# EXAMPLE: LPV-MPC OF A NONLINEAR CSTR SYSTEM

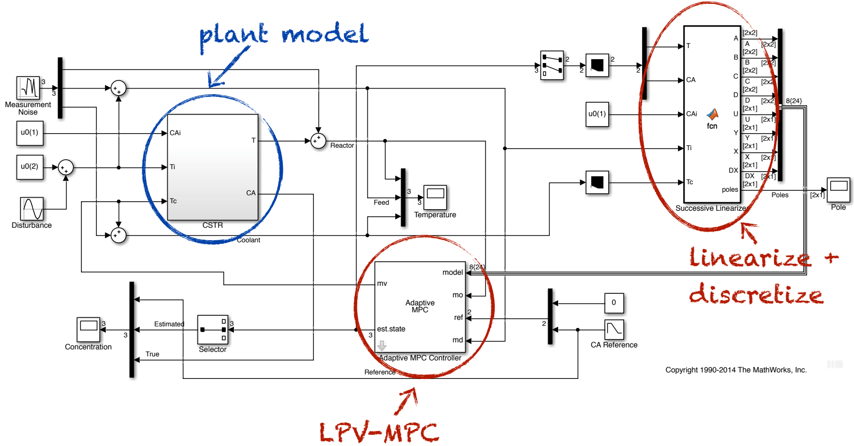
- MPC design (cont'd)

```
% Set scale factors because plant input and output signals have different
% orders of magnitude
Uscale = [30 50];
Yscale = [50 10];
mpcobj.DV(1).ScaleFactor = Uscale(1);
mpcobj.MV(1).ScaleFactor = Uscale(2);
mpcobj.OV(1).ScaleFactor = Yscale(1);
mpcobj.OV(2).ScaleFactor = Yscale(2);

% Let reactor temperature T float (i.e. with no setpoint tracking error
% penalty), because the objective is to control reactor concentration CA
% and only one manipulated variable (coolant temperature Tc) is available.
mpcobj.Weights.OV = [0 1];

% Due to the physical constraint of coolant jacket, Tc rate of change is
% bounded by degrees per minute.
mpcobj.MV.RateMin = -2;
mpcobj.MV.RateMax = 2;
```

# EXAMPLE: LPV-MPC OF A NONLINEAR CSTR SYSTEM

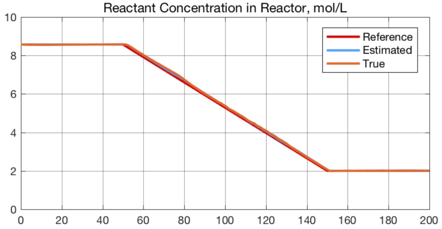
- Simulink diagram

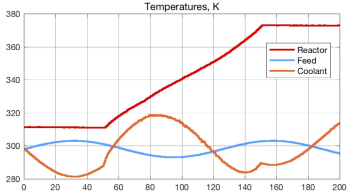


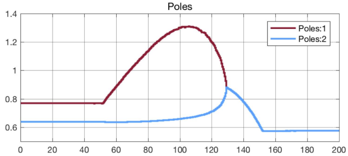

```
>> edit ampc_cstr_linearization
```

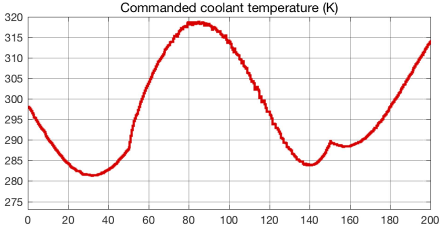
# EXAMPLE: LPV-MPC OF A NONLINEAR CSTR SYSTEM

- Closed-loop results



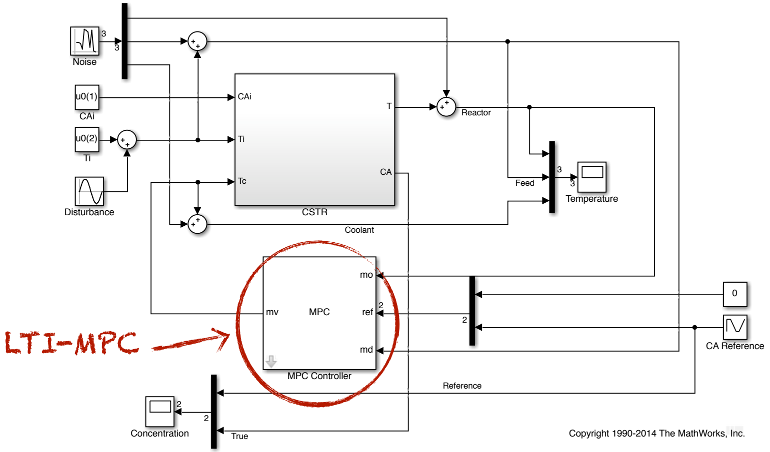



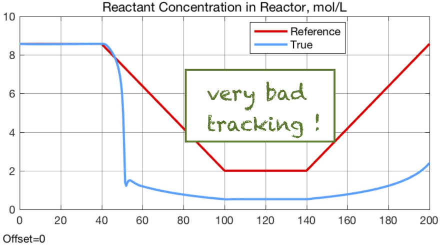
# EXAMPLE: LTI-MPC OF A NONLINEAR CSTR SYSTEM

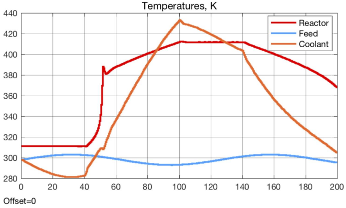
- Closed-loop results with **LTI-MPC**, same tuning

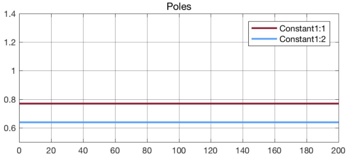


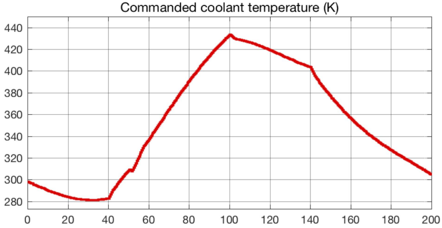
# EXAMPLE: LTI-MPC OF A NONLINEAR CSTR SYSTEM

- Closed-loop results



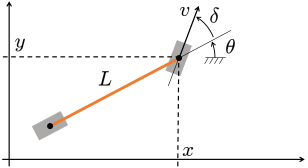



# AUTONOMOUS DRIVING EXAMPLE

- **Goal**: Control longitudinal acceleration and steering angle of the vehicle simultaneously for autonomous driving with obstacle avoidance

- **Approach**: MPC based on a bicycle-like kinematic model of the vehicle in Cartesian coordinates




$$\begin{cases} \dot{x} = v\cos(\theta+\delta) \\ \dot{y} = v\sin(\theta+\delta) \\ \dot{\theta} = \frac{v}{L}\sin(\delta) \end{cases}$$

| | | | |
|---|---|---|---|
| $(x,y)$ | Cartesian position of front wheel | $v$ | velocity at front wheel |
| $\theta$ | vehicle orientation | $\delta$ | steering input |
| $L$ | vehicle length = $4.5$ m | | |

- Let $x_n, y_n, \theta_n, v_n, \delta_n$ nominal states/inputs satisfying

$$\begin{bmatrix} \dot{x}_n \\ \dot{y}_n \\ \dot{\theta}_n \end{bmatrix} = \begin{bmatrix} v_n \cos(\theta_n + \delta_n) \\ v_n \sin(\theta_n + \delta_n) \\ \frac{v_n}{L} \sin(\delta_n) \end{bmatrix}$$

feasible nominal trajectory

- Linearize the model around the nominal trajectory:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} \approx \begin{bmatrix} \dot{x}_n \\ \dot{y}_n \\ \dot{\theta}_n \end{bmatrix} + A_c \begin{bmatrix} x - x_n \\ y - y_n \\ \theta - \theta_n \end{bmatrix} + B_c \begin{bmatrix} v - v_n \\ \delta - \delta_n \end{bmatrix}$$

linearized model

where $A_c, B_c$ are the **Jacobian matrices**

$$A_c = \begin{bmatrix} 0 & 0 & -v_n \sin(\theta_n + \delta_n) \\ 0 & 0 & v_n \cos(\theta_n + \delta_n) \\ 0 & 0 & 0 \end{bmatrix} \quad B_c = \begin{bmatrix} \cos(\theta_n + \delta_n) & -v_n \sin(\theta_n + \delta_n) \\ \sin(\theta_n + \delta_n) & v_n \cos(\theta_n + \delta_n) \\ \frac{1}{L} \sin(\delta_n) & \frac{v_n}{L} \cos(\delta_n) \end{bmatrix}$$

- Use first-order Euler method to discretize model:

$$A = I + T_s A_c, \quad B = T_s B_c, \quad T_s = 50\,\mathrm{ms}$$

# AUTONOMOUS DRIVING EXAMPLE

- Constraints on inputs and input variations $\Delta v_k = v_k - v_{k-1}, \Delta\delta_k = \delta_k - \delta_{k-1}$:

$$
\begin{array}{lll}
-20 \leq v \leq 70 & \mathrm{km/h} & \text{velocity constraint} \\
-45 \leq \delta \leq 45 & \mathrm{deg} & \text{steering angle} \\
-5 \leq \Delta\delta \leq 5 & \mathrm{deg} & \text{steering angle rate}
\end{array}
$$

- Stage cost to minimize:

$$(x - x_{\mathrm{ref}})^2 + (y - y_{\mathrm{ref}})^2 + \Delta v^2 + \Delta\delta^2$$

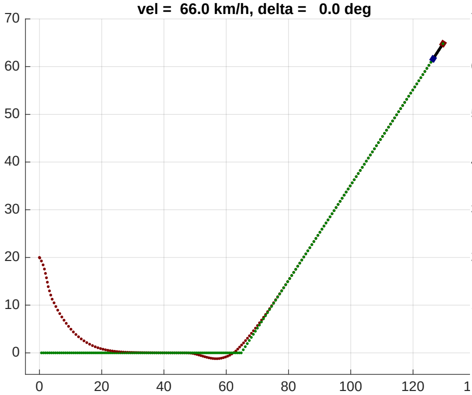
- Prediction horizon: $N = 30$ (prediction distance = $NT_s v$, for example 25 m at 60 km/h)

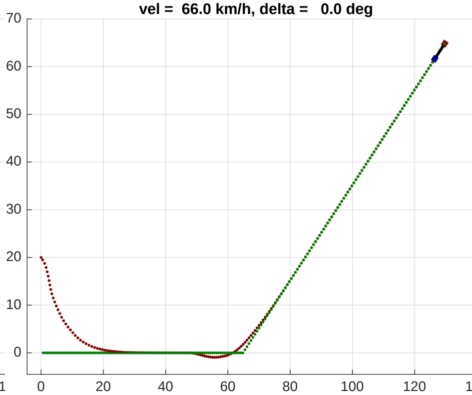
- Control horizon: $N_u = 4$

- Preview on reference signals available

# AUTONOMOUS DRIVING EXAMPLE

- Closed-loop simulation results



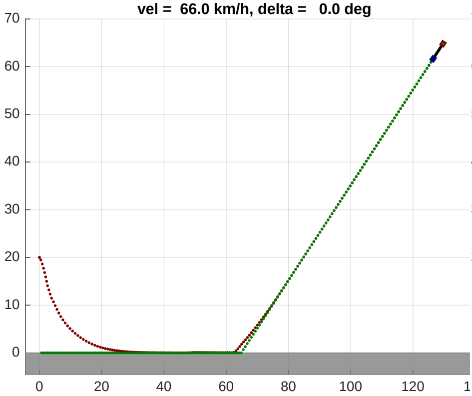

LPV-MPC

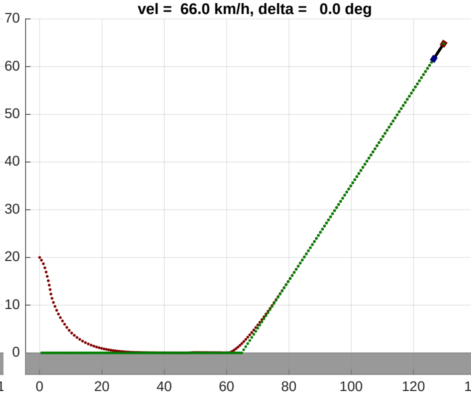
Model linearized @$t$

LTV-MPC

Model linearized @$t+k$, $k=0,\ldots,N-1$

- Add position constraint $y \geq 0\,\mathrm{m}$




LPV-MPC

Model linearized @$t$

LTV-MPC

Model linearized @$t+k$, $k=0,\ldots,N-1$

# LTV KALMAN FILTER

- Process model = **LTV model with noise**

$$\begin{aligned} x(k+1) &= A(k)x(k) + B(k)u(k) + G(k)\xi(k) \\ y(k) &= C(k)x(k) + \zeta(k) \end{aligned}$$

$\xi(k) \in \mathbb{R}^q$ = zero-mean white **process noise** with covariance $Q(k) \succeq 0$

$\zeta(k) \in \mathbb{R}^p$ = zero-mean white **measurement noise** with covariance $R(k) \succ 0$

- **measurement update**:

$$\begin{aligned} M(k) &= P(k|k-1)C(k)'[C(k)P(k|k-1)C(k)' + R(k)]^{-1} \\ \hat{x}(k|k) &= \hat{x}(k|k-1) + M(k)\left(y(k) - C(k)\hat{x}(k|k-1)\right) \\ P(k|k) &= (I - M(k)C(k))P(k|k-1) \end{aligned}$$

- **time update**:

$$\hat{x}(k+1|k) = A(k)\hat{x}(k|k) + B(k)u(k)$$

$$P(k+1|k) = A(k)P(k|k)A(k)' + G(k)Q(k)G(k)'$$

- Note that here the observer gain $L(k) = A(k)M(k)$

# EXTENDED KALMAN FILTER

- Process model = **nonlinear model with noise**

$$\begin{aligned} x(k+1) &= f(x(k),u(k),\xi(k)) \\ y(k) &= g(x(k),u(k))+\zeta(k) \end{aligned}$$

- **measurement update**:

$$\begin{aligned} C(k) &= \frac{\partial g}{\partial x}(\hat{x}_{k|k-1},u(k)) \\ M(k) &= P(k|k-1)C(k)'[C(k)P(k|k-1)C(k)'+R(k)]^{-1} \\ \hat{x}(k|k) &= \hat{x}(k|k-1)+M(k)\left(y(k)-g(\hat{x}(k|k-1),u(k))\right) \\ P(k|k) &= (I-M(k)C(k))P(k|k-1) \end{aligned}$$

- **time update**:

$$\hat{x}(k+1|k) = f(\hat{x}(k|k),u(k))$$

$$A(k) = \frac{\partial f}{\partial x}(\hat{x}_{k|k},u(k),E[\xi(k)]),\ G(k) = \frac{\partial f}{\partial \xi}(\hat{x}_{k|k},u(k),E[\xi(k)])$$

$$P(k+1|k) = A(k)P(k|k)A(k)'+G(k)Q(k)G(k)'$$

# NONLINEAR MODEL PREDICTIVE CONTROL

# NONLINEAR MPC

(Mayne, Rawlings, Diehl, 2017)

- Nonlinear prediction model

$$\begin{cases} x_{k+1} & = & f(x_k, u_k) \\ y_k & = & g(x_k, u_k) \end{cases}$$

- Nonlinear constraints $h(x_k, u_k) \leq 0$
- Nonlinear performance index $\min \ \ell_N(x_N) + \sum_{k=0}^{N-1} \ell(x_k, u_k)$
- Optimization problem: **nonlinear programming problem (NLP)**

$$\begin{array}{rl} \min_z & F(z, x(t)) \\ \text{s.t.} & G(z, x(t)) \leq 0 \\ & H(z, x(t)) = 0 \end{array} \qquad z = \begin{bmatrix} u_0 \\ \vdots \\ u_{N-1} \\ x_1 \\ \vdots \\ x_N \end{bmatrix}$$

# NONLINEAR OPTIMIZATION

- (Nonconvex) NLP is harder to solve than QP
- Convergence to a **global optimum** may not be guaranteed
- Several NLP solvers exist (such as **Sequential Quadratic Programming (SQP)**)

  (Nocedal, Wright, 2006)
- NLP can be useful to deal with strong dynamical nonlinearities and/or nonlinear constraints/costs
- NL-MPC is less used in practice than linear MPC

# FAST NONLINEAR MPC

(Lopez-Negrete, D'Amato, Biegler, Kumar, 2013)

- **Fast MPC**: exploit **sensitivity analysis** to compensate for the computational delay caused by solving the NLP
- **Key idea**: pre-solve the NLP between time $t-1$ and $t$ based on the predicted state $x^*(t) = f(x(t-1), u(t-1))$ in background
- Get $u^*(t)$ and sensitivity $\left.\frac{\partial u^*}{\partial x}\right|_{x^*(t)}$ within sample interval $[(t-1)T_s, tT_s)$
- At time $t$, get $x(t)$ and compute

$$u(t) = u^*(t) + \frac{\partial u^*}{\partial x}(x(t) - x^*(t))$$

- A.k.a. **advanced-step MPC** (Zavala, Biegler, 2009)
- Note that still one NLP must be solved within the sample interval

# FROM LTV-MPC TO NONLINEAR MPC

- **Key idea**: Solve a **sequence of LTV-MPC** problems at each time $t$

For $h = 0$ to $h_{\max} - 1$ do:

1. **Simulate** from $x(t)$ with inputs $U_h$ and get state trajectory $X_h$
2. **Linearize** around $(X_h, U_h)$ and **discretize** in time
3. Get $U^*_{h+1}$ = **QP solution** of corresponding LTV-MPC problem
4. **Line search**: find optimal step size $\alpha_h \in (0, 1]$;
5. Set $U_{h+1} = (1 - \alpha_h)U_h + \alpha_h U^*_{h+1}$;

Return solution $U_{h_{\max}}$

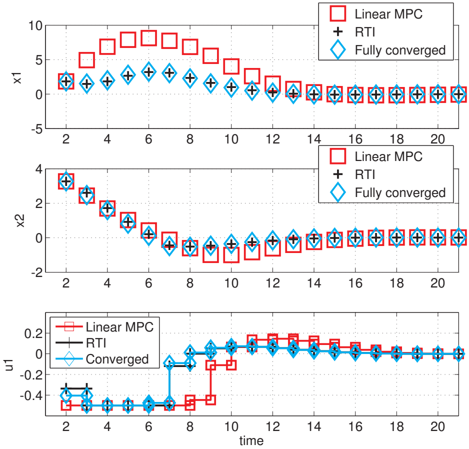
- The above method is a form of **Sequential Quadratic Programming** (SQP) applied to solve the full nonlinear MPC problem

- Special case: just solve one iteration with $\alpha = 1$ (a.k.a. **Real-Time Iteration**) (Diehl, Bock, Schloder, Findeisen, Nagy, Allgower, 2002) = LTV-MPC

# NONLINEAR MPC

(Gros, Zanon, Quirynen, Bemporad, Diehl, 2020)

- Example



- The (unconstrained) NLMPC problem can be rephrased as

$$\begin{array}{rl} \min_{X,U} & \frac{1}{2}\left[\begin{smallmatrix} X \\ U \end{smallmatrix}\right]' H \left[\begin{smallmatrix} X \\ U \end{smallmatrix}\right] + c' \left[\begin{smallmatrix} X \\ U \end{smallmatrix}\right] \\ \text{s.t.} & G(X,U) = 0 \end{array} \qquad X = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}, \; U = \begin{bmatrix} u_0 \\ \vdots \\ u_{N-1} \end{bmatrix}$$

$G(X,U) = 0$ summarizes the **nonlinear dynamics** $x_{k+1} = f_k(x_k, u_k)$ for all $k$

- Given the simulated nominal trajectories $\bar{X}^0, \bar{U}^0$, let us **linearize**

$$G(X,U) \approx \underbrace{G(\bar{X}^0, \bar{U}^0)}_{\text{0 by construction}} + \underbrace{\nabla G(\bar{X}^0, \bar{U}^0) \left[\begin{smallmatrix} \Delta X \\ \Delta U \end{smallmatrix}\right] = 0}_{\text{LTV model equations}} \qquad \begin{array}{l} \Delta X = X - \bar{X}^0 \\ \Delta U = U - \bar{U}^0 \end{array}$$

and solve the **quadratic program**

$$\begin{bmatrix} \Delta X^1 \\ \Delta U^1 \end{bmatrix} = \arg\min_{\Delta X, \Delta U} \; \frac{1}{2}\left[\begin{smallmatrix} \Delta X \\ \Delta U \end{smallmatrix}\right]' H \left[\begin{smallmatrix} \Delta X \\ \Delta U \end{smallmatrix}\right] + \left(H \begin{bmatrix} \bar{X}^0 \\ \bar{U}^0 \end{bmatrix} + c\right)' \left[\begin{smallmatrix} \Delta X \\ \Delta U \end{smallmatrix}\right]$$
$$\text{s.t.} \quad \nabla G(\bar{X}^0, \bar{U}^0) \left[\begin{smallmatrix} \Delta X \\ \Delta U \end{smallmatrix}\right] = 0$$

(we have neglected $\nabla^2 G(X,U)$ in formulating the above QP)

# SEQUENTIAL QUADRATIC PROGRAMMING

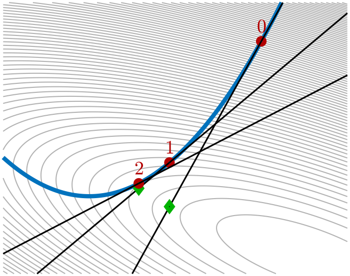
- Set $U^1 = U^0 + \alpha_1 \Delta U^1$ and **simulate the NL model** to get new states $X^1$, where $0 \leq \alpha_1 \leq 1$ (ideally, $\alpha_1 = 1$)

- The **step-size** $\alpha_1$ is chosen by **line search**, to ensure the cost fcn decreases

- Linearize again around $X^1, U^1$ and solve a new QP to get the optimal $\Delta U^2$. And so on ...

- After solving $M$ QPs, we have the optimal solution $U^* = U^{M-1} + \alpha_M \Delta U^M$

- Linear inequality constraints $AX + BU \leq f$ are also included (if they appear in NLMPC)



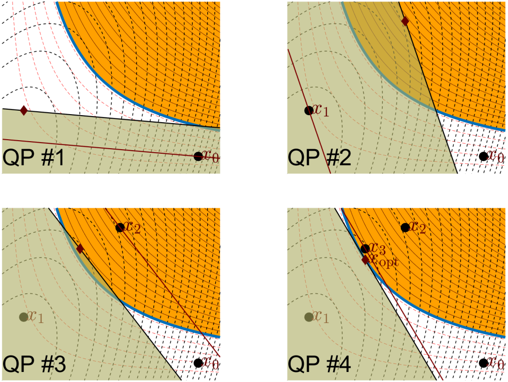
# ADVANTAGES OF NONLINEAR MPC

- Better exploits **nonlinear prediction models** than LTV-MPC
  - **Physics-based** models (= white-box models)
  - **Machine-learned** models (= black-box models, e.g., neural networks)

- Can handle **nonlinear inequality constraints** (and nonlinear cost functions)

$$g(x) \leq 0$$

$$g(x_k) + \nabla g(x_k)(x - x_k) \leq 0$$



# ODYS EMBEDDED MPC TOOLSET

- **ODYS Embedded MPC** is a software toolchain for design and deployment of MPC solutions in industrial production
- Support for **linear & nonlinear MPC** and **extended Kalman filtering**
- Extremely flexible, all MPC parameters can be changed at runtime (models, cost function, horizons, constraints, ...)
- Integrated with **ODYS QP Solver** for max speed, low memory footprint, and robustness (also in single precision)

`odys.it/qp`

- Library-free C code, **MISRA-C 2012 compliant**
- Currently used worldwide by several automotive OEMs in R&D and production
- Support for **neural networks** as prediction models (**ODYS Deep Learning**)

`odys.it/embedded-mpc`