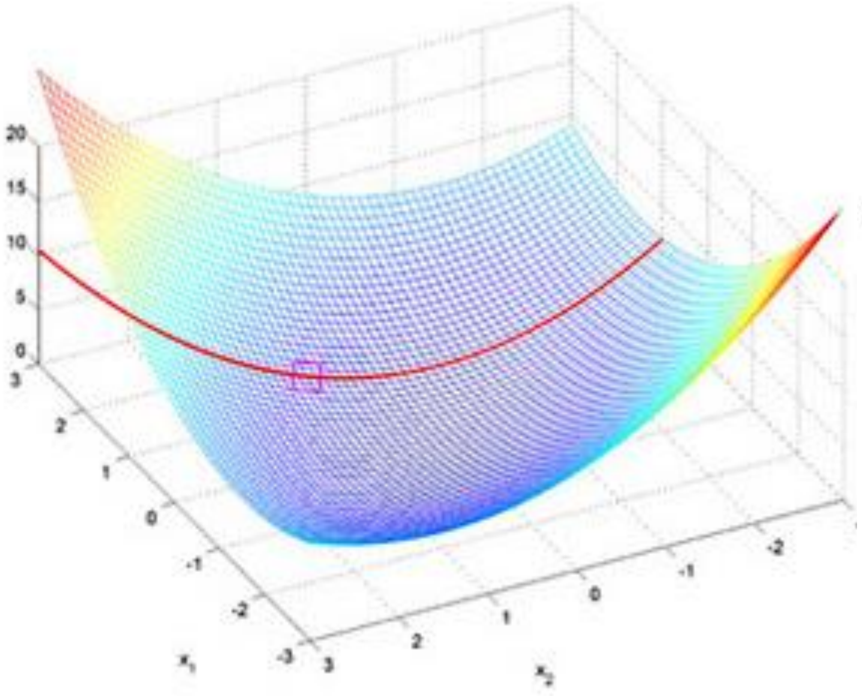
MEGR 7090/8090: Advanced Optimal Control

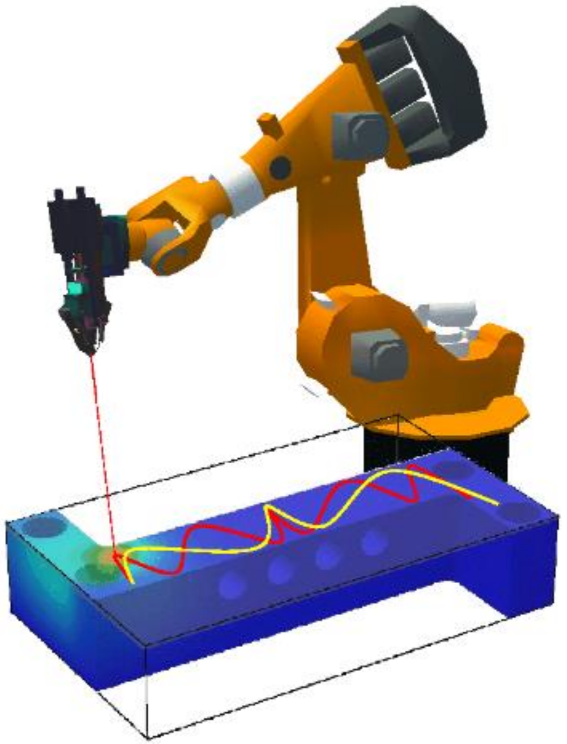
UNC CHARLOTTE



$$\boxed{V_n(\mathbf{x}_n) = \min_{\{\mathbf{u}_n,\mathbf{u}_{n+1},\cdots,\mathbf{u}_{N-1}\}} \left[\frac{1}{2}\sum_{k=n}^{N-1}\left(\mathbf{x}_k^T\mathbf{Q}_k\mathbf{x}_k + \mathbf{u}_k^T\mathbf{R}\mathbf{u}_k\right) + \frac{1}{2}\mathbf{x}_N^T\mathbf{Q}_N\mathbf{x}_N\right]}$$

$$\begin{aligned}
V_n(\mathbf{x}_n) &= \min_{\{\mathbf{u}_n,\mathbf{u}_{n+1},\cdots,\mathbf{u}_{N-1}\}} \left[\frac{1}{2}\sum_{k=n}^{N-1}\left(\mathbf{x}_k^T\mathbf{Q}_k\mathbf{x}_k + \mathbf{u}_k^T\mathbf{R}\mathbf{u}_k\right) + \frac{1}{2}\mathbf{x}_N^T\mathbf{Q}_N\mathbf{x}_N\right] \\
&= \min_{\mathbf{u}_n}\left[\frac{1}{2}\left(\mathbf{x}_n^T\mathbf{Q}_n\mathbf{x}_n + \mathbf{u}_n^T\mathbf{R}\mathbf{u}_n\right) + \underbrace{\min_{\{\mathbf{u}_{n+1},\cdots,\mathbf{u}_{N-1}\}}\left[\frac{1}{2}\sum_{k=n+1}^{N-1}\left(\mathbf{x}_k^T\mathbf{Q}_k\mathbf{x}_k + \mathbf{u}_k^T\mathbf{R}\mathbf{u}_k\right) + \frac{1}{2}\mathbf{x}_N^T\mathbf{Q}_N\mathbf{x}_N\right]}_{V_{n+1}(\mathbf{x}_{n+1})}\right] \\
&= \min_{\mathbf{u}_n}\left[\frac{1}{2}\left(\mathbf{x}_n^T\mathbf{Q}_n\mathbf{x}_n + \mathbf{u}_n^T\mathbf{R}\mathbf{u}_n\right) + V_{n+1}(\mathbf{x}_{n+1})\right]
\end{aligned}$$

$$\boxed{V_n(\mathbf{x}_n) = \min_{\mathbf{u}_n}\left[\frac{1}{2}\left(\mathbf{x}_n^T\mathbf{Q}_n\mathbf{x}_n + \mathbf{u}_n^T\mathbf{R}\mathbf{u}_n\right) + V_{n+1}(\mathbf{x}_{n+1})\right]}$$



Lecture 17

October 19, 2017

Summary of Discrete-Time Control Trajectory Optimization Techniques Learned So Far

UNC CHARLOTTE

Convex optimization tools:

- Main idea: Treat the control trajectory as a set of design variables
- Special-case tools: Linear programming (LP), quadratic programming (QP)
- More general-case tool for nonlinear problems (NLP): sequential quadratic programming (SQP)
- Pros: Generally computationally efficient, with off-the-shelf solvers available
- Con: Only ***local*** optimality is guaranteed for ***non-convex problems***

Global, grid-based optimization tools:

- Exhaustive search: Extremely computationally demanding, treats control trajectory optimization as design optimization
- Dynamic programming: Leverages Bellman's principle of optimality (and the concept of ***sequence*** and ***states***) to relieve some of the computational intensity of an exhaustive search
- Pro: Global optimality up to the grid resolution, regardless of convexity
- Con: Fine grid/lots of states requires significant computational resources ("***curse of dimensionality***")

Discrete-Time Control Trajectory Optimization Techniques – Common Theme

So far, the process of optimal control has involved two steps:

- **Step 1**: Determine the trajectory, $\mathbf{u}^*$, that minimizes an objective function over a future horizon, subject to constraints:

Minimize $J(\mathbf{u};\mathbf{x}(0)) = \sum_{i=0}^{N-1} g(\mathbf{x}(i), u(i)) + h(\mathbf{x}(N))$

Subject to: $\mathbf{x}(i) \in X, i = 0 \ldots N$, $u(i) \in U, i = 0 \ldots N-1$ and $\mathbf{x}(k+1) = f(\mathbf{x}(k), u(k))$

- **Step 2**: Implement the above trajectory, starting at step 0:

$$u(0) = u^*(0), u(1) = u^*(1), \ldots, u(N-1) = u^*(N-1)$$

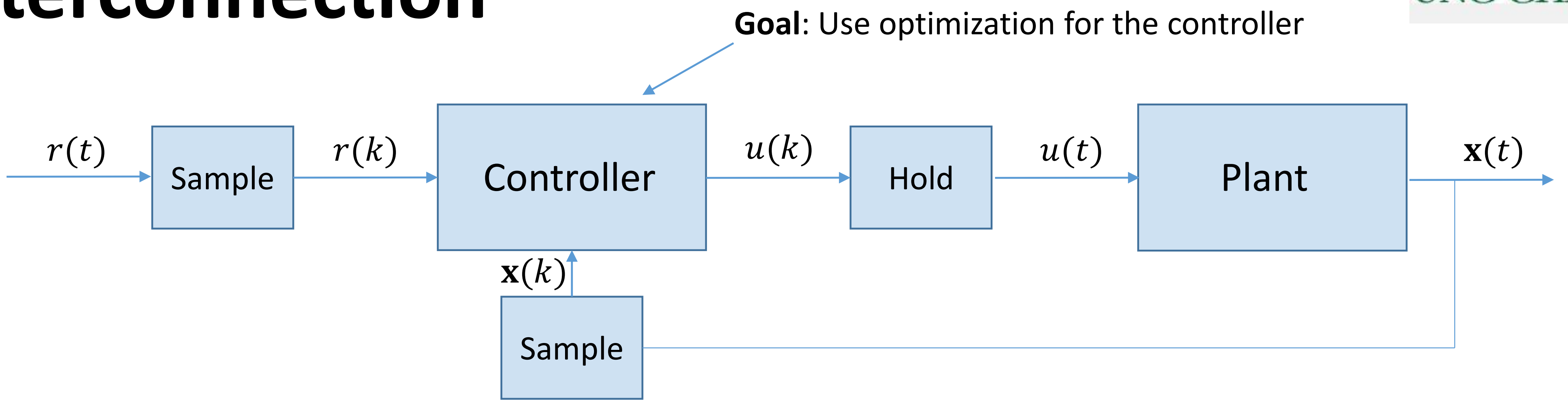
  - No ability to adjust $\mathbf{u}^*$ based on new knowledge garnered at steps 1…N-1

Analogy: Imagine if you were asked to race a car along a complex course/track by:

- First studying the track carefully and coming up with a detailed plan
- Then putting on a blindfold
- Then driving around the track

[Video link]

Typical Discrete-Time Feedback Interconnection



Observations from the familiar block diagram above:

- Control signal is adjusted each time step based on observations up until this time step
- Traditional design techniques use tools like root locus and Bode plots to choose the controller structure, i.e., the controller does **not** minimize an objective function
- Key challenge: Incorporate ***feedback*** (incorporated in the block diagram above) into a controller that ***minimizes an objective function*** (not incorporated in most traditional feedback controllers you've learned about)

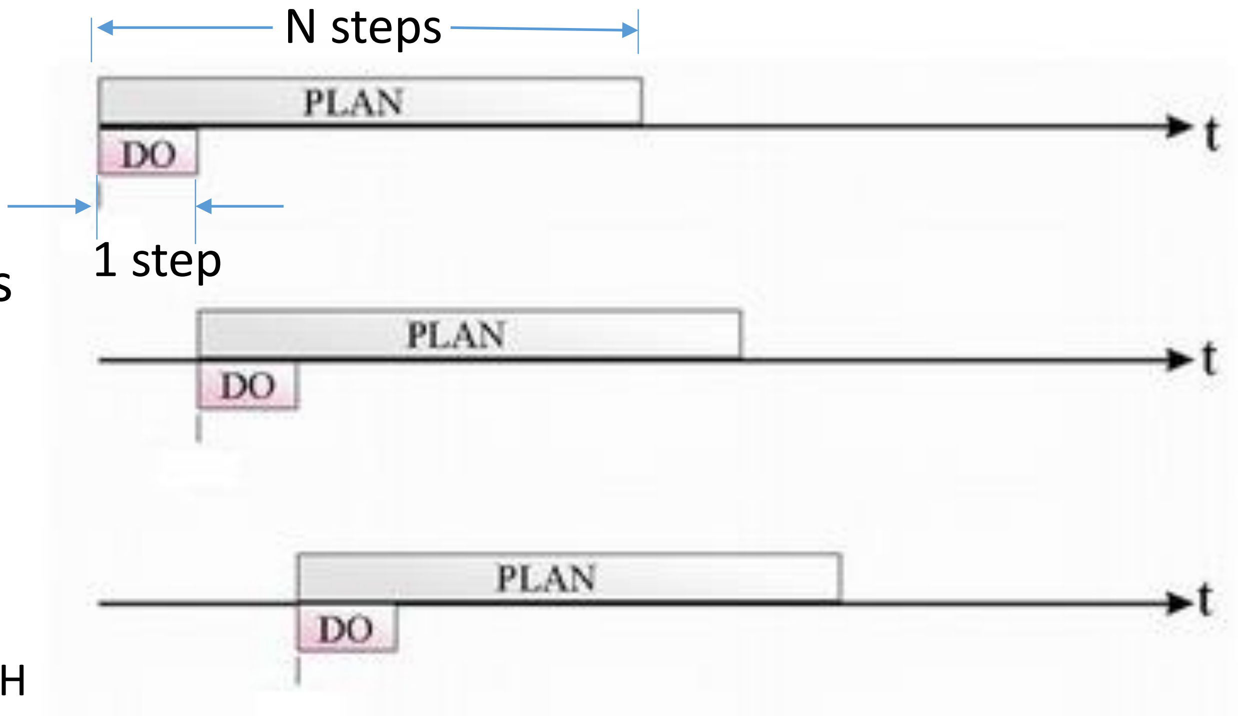
Incorporating Control Trajectory Optimization into Optimal Control – Main Idea

- At time 0, compute the control trajectory that minimizes $J(\mathbf{u}, \mathbf{x}(0)) = \sum_{i=0}^{N-1} g(\mathbf{x}(i), u(i)) + h(\mathbf{x}(N))$, subject to constraints. Implement $u^*(0)$.

- At the next time step (1), compute the control signal that minimizes $J(\mathbf{u}, \mathbf{x}(1)) = \sum_{i=1}^{N} g(\mathbf{x}(i), u(i)) + h(\mathbf{x}(N+1))$, subject to constraints

- Repeat every time step

- This process is known as ***model predictive control (MPC)***, also known as ***receding horizon control***



Source: ETH

MPC Notation

UNC CHARLOTTE

Note: At every time step, MPC optimizes a **sequence** of control signals. Those sequences overlap. (Example: If $N = 10$), then a value for $u(9)$ is computed at steps 0, 1,…, 9. To differentiate between these different values, we use what Dr. V calls "slash notation":

- $\mathbf{u}(k) = [u(k|k) \quad \dots \quad u(k+N-1|k)]^T$ = candidate control sequence at step k
- $\mathbf{x}_{seq}(k) = [\mathbf{x}(k|k) \quad \dots \quad \mathbf{x}(k+N|k)]^T$ = predicted state sequence at step k
- In general, the notation is interpreted as (time step at which the variable is evaluated|time step at which the optimization is performed)

General mathematical formulation for MPC:

$$\mathbf{u}^*(k) = \arg\min \left[\sum_{i=k}^{N-1} g\big(\mathbf{x}(i|k), u(i|k)\big) + h(\mathbf{x}(k+N|k)) \right]$$

$$u(k) = u^*(k|k)$$

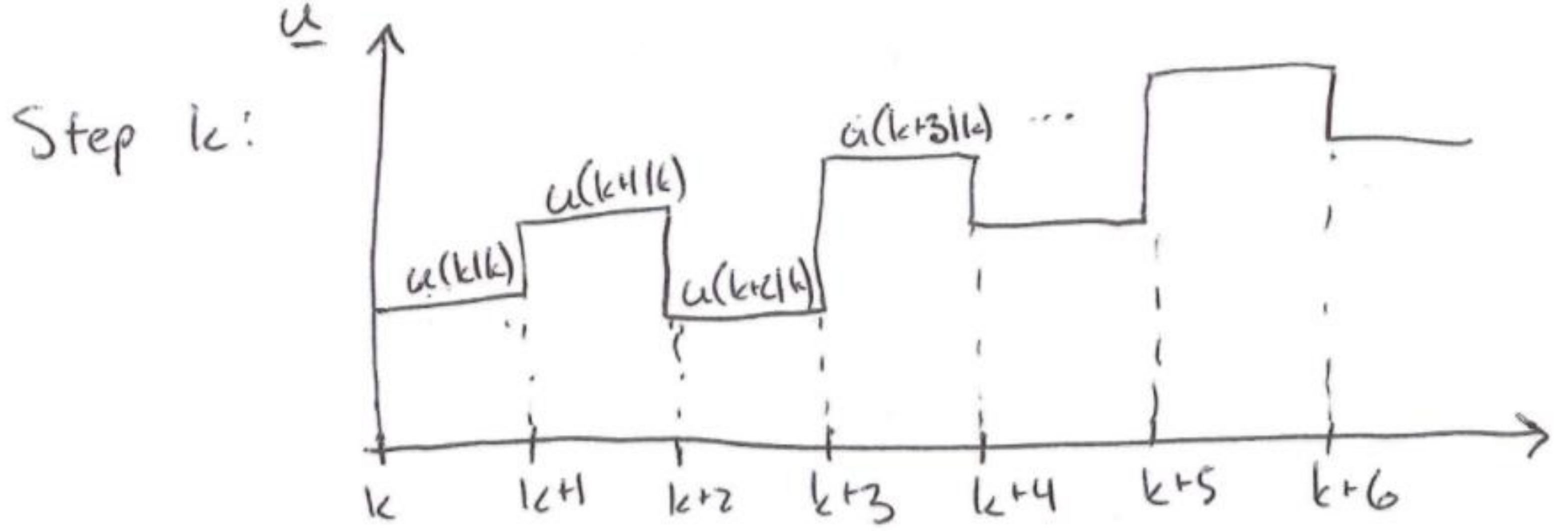
(Implement the first step of the optimized sequence)

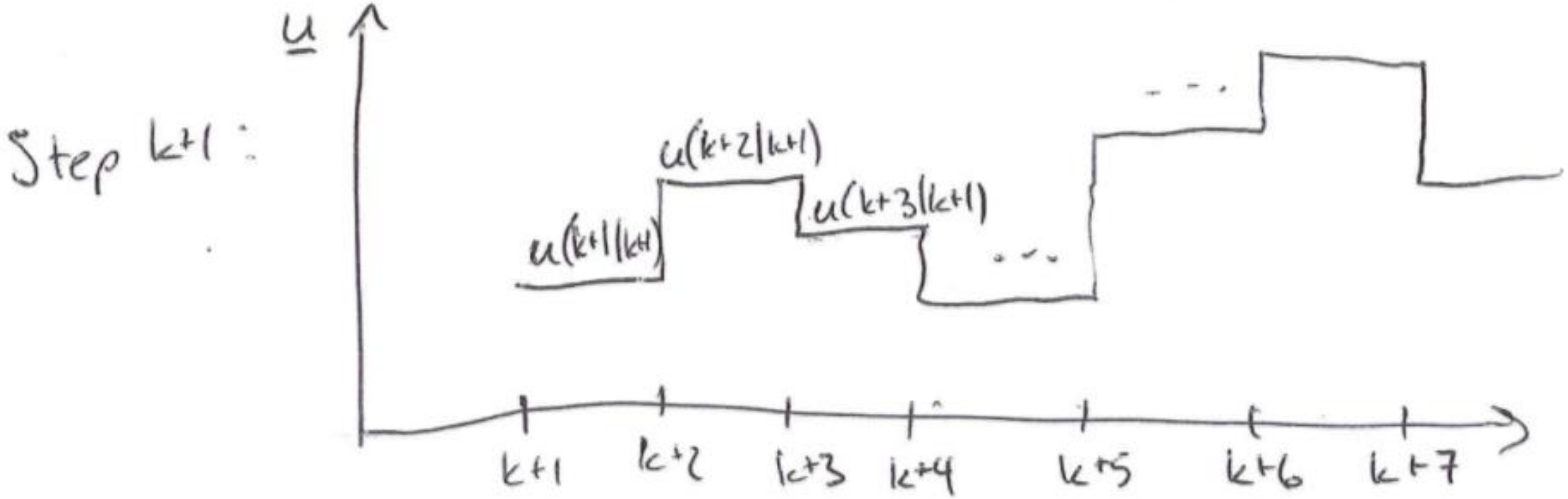
subject to: $\mathbf{x}(i|k) \in X, i = k \dots k+N$
$u(i|k) \in U, i = k \dots k+N-1$

and: $\mathbf{x}(i+1|k) = f\big(\mathbf{x}(i|k), u(i|k)\big)$

MPC Notation

"Slash" notation:





MPC Notation

$$\underbrace{u(k)}_{\text{what I actually implement at step } k} = u^*(k|k)$$

optimized control sequence.

optimization was done at step k

decision variable value at step k

MPC Prediction vs. Control Horizon

Sometimes it is important to consider effects of control signals over a long horizon, but not computationally feasible to optimize that many control variables.

To resolve this, MPC can assume a **prediction horizon** (N_p) that is longer than the control horizon (N_c):

$$\mathbf{u}^*(k) = \arg\min \left[\sum_{i=k}^{N_p-1} g\big(\mathbf{x}(i|k), u(i|k)\big) + h\big(\mathbf{x}(k+N_p|k)\big) \right] \quad \Rightarrow \quad u(k) = u^*(k|k)$$

(Implement the first step of the optimized sequence)

subject to: $\mathbf{x}(i|k) \in X, i = k \ldots k+N$

$u(i|k) \in U, i = k \ldots k+N-1$ and: $\mathbf{x}(i+1|k) = f\big(\mathbf{x}(i|k), u(i|k)\big)$

$$\boxed{u(i|k) = u(k+N_c-1|k), i = N_c \ldots N_p - 1}$$

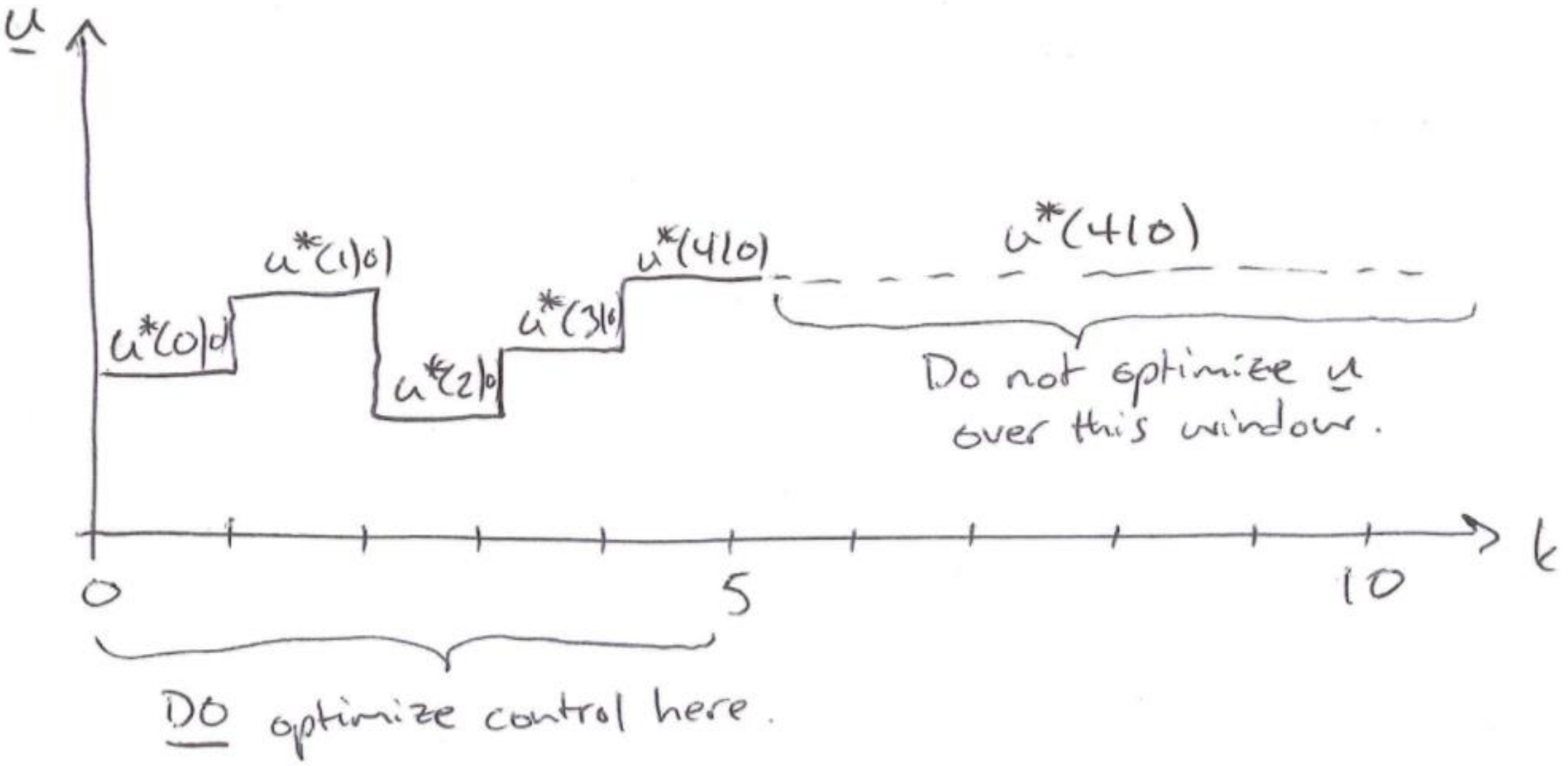
Hold the control signal constant after N_c steps

MPC Prediction vs. Control Horizon

Control vs. prediction horizon:

Ex. $N_c = 5$, $N_p = 10$

Sample control sequence at step 0:



Special Case of MPC – Unconstrained Discrete-Time LQR

Unconstrained discrete-time LQR problem:

$$\mathbf{u}^*(k) = \arg\min \left[\sum_{i=k}^{N-1} \left(\mathbf{x}^T(i|k) Q \mathbf{x}(i|k) + R u^2(i|k) \right) + \mathbf{x}^T(k+N|k) S \mathbf{x}(k+N|k) \right] \Rightarrow u(k) = u^*(k|k)$$

Subject to: $\mathbf{x}(i+1|k) = A\mathbf{x}(i|k) + Bu(i|k)$

Solution – *Linear feedback control:* $u(k) = u^*(k|k) = -K\mathbf{x}(k)$

- When N is finite, K can be found through ***backward recursion*** (see lecture 16 notes)
- When $N = \infty$, K can be found by solving the ***algebraic Riccati equation*** (see lecture 16 notes)

Limitations to unconstrained discrete-time LQR:

- Aside from the system dynamics, constraints are not considered (it's in the name!)
- Objective function ***must be quadratic*** in states and control variable
- System dynamics ***must be linear***

A Slightly More General Case of MPC – *Constrained* Discrete-Time LQR

Constrained discrete-time LQR problem – also known as _linear MPC_:

$$\mathbf{u}^*(k) = \arg\min\left[\sum_{i=k}^{N-1}\left(\mathbf{x}^T(i|k)Q\mathbf{x}(i|k) + Ru^2(i|k)\right) + \mathbf{x}^T(k+N|k)S\mathbf{x}(k+N|k)\right] \Rightarrow u(k) = u^*(k|k)$$

Subject to:

$$M_1\mathbf{x}(i|k) - \mathbf{b}_1 \le 0, i = k \ldots k+N$$
$$M_2u(i|k) - \mathbf{b}_2 \le 0, i = k \ldots k+N-1$$

and $\mathbf{x}(i+1|k) = A\mathbf{x}(i|k) + Bu(i|k)$

Solution – _Quadratic programming (QP):_

- Refer to lecture 9 notes for proof that the above problem is convex and therefore QP can be used reliably

Simulink's built-in MATLAB toolbox can be used to perform linear MPC (constrained discrete-time LQR)

Introduction to MPC Toolbox

Basic features:

- MPC toolbox generates an MPC design for a **linear, discrete-time model** and **quadratic cost function**
- MPC toolbox will automatically **convert your plant model to discrete time** (given a time step) and **linearize the model** (if there are nonlinearities)
- User guide provided on Canvas

Let's learn how to use MPC toolbox with an example:

System dynamics:

$$\dot{\mathbf{x}} = A\mathbf{x} + Bu$$

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Objective function: $$J\big(\mathbf{u}(k), \mathbf{x}(k)\big) = \sum_{i=k}^{k+N-1} (\mathbf{x}^T(i|k)Q\mathbf{x}(i|k) + Ru^2(i|k))$$

Constraints: $$-1 \leq u(i|k) \leq 1, i = 0 \ldots N-1$$

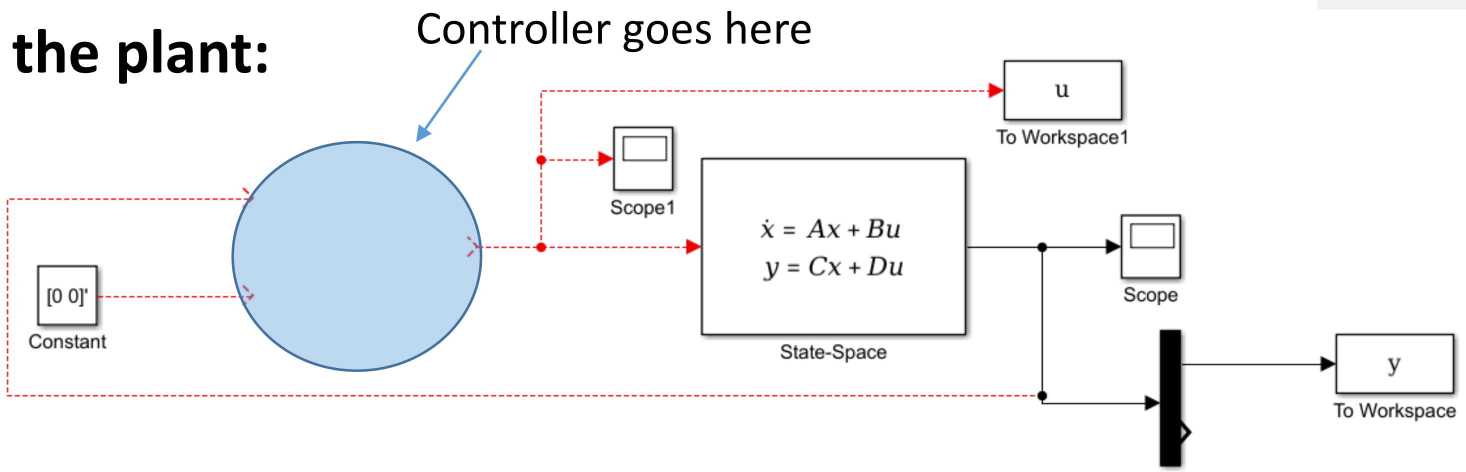
$$-0.5 \leq \mathbf{x}(i|k) \leq 1, i = 0 \ldots N-1$$

Time step: $T = 0.1$

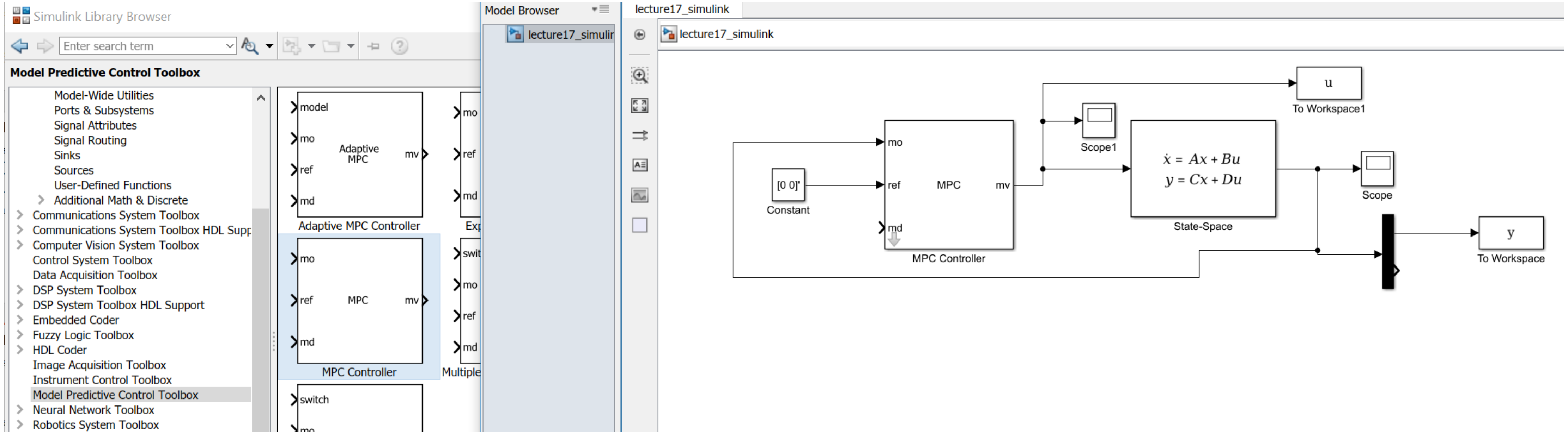
Horizon length: $N_c = N_p = N = 10$

Introduction to MPC Toolbox

Step 1 – create a model for the plant:

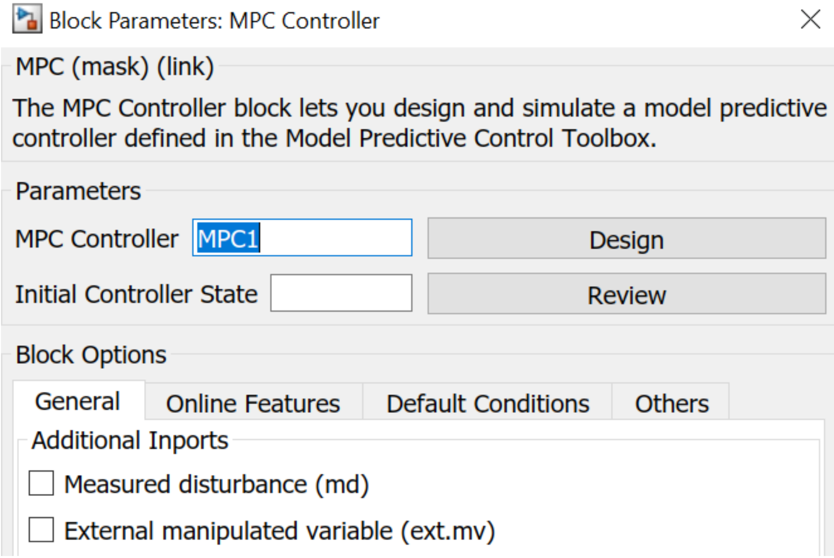


Step 2 – drag and drop a blank MPC block into the position of the controller:

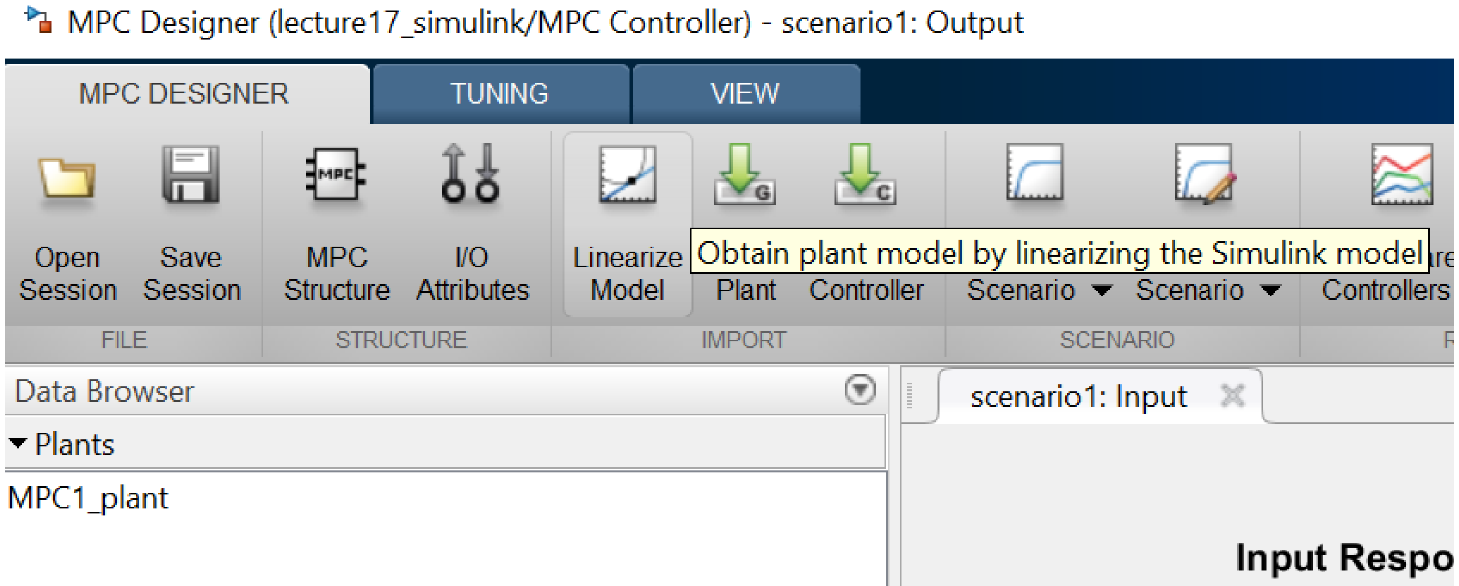


Introduction to MPC Toolbox

Step 3 – Open up the MPC controller, select "Design":

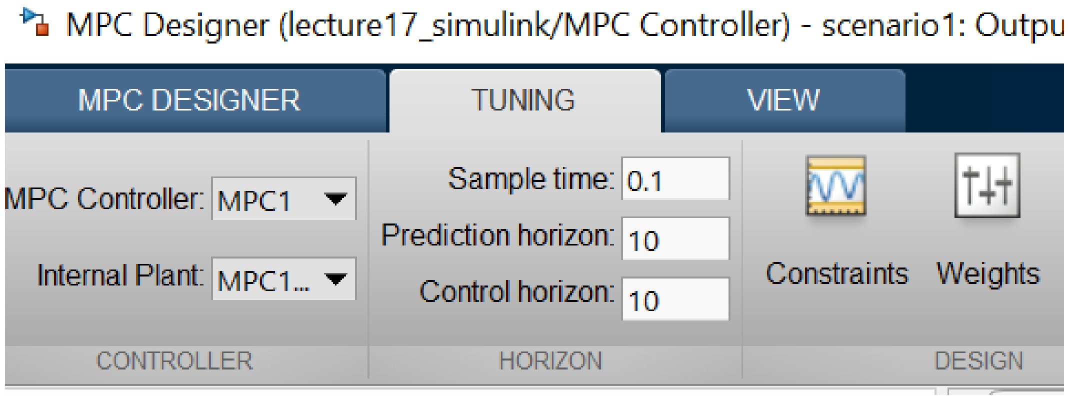


Step 4 – Select the MPC structure and linearize the model (already linear for us):



Introduction to MPC Toolbox

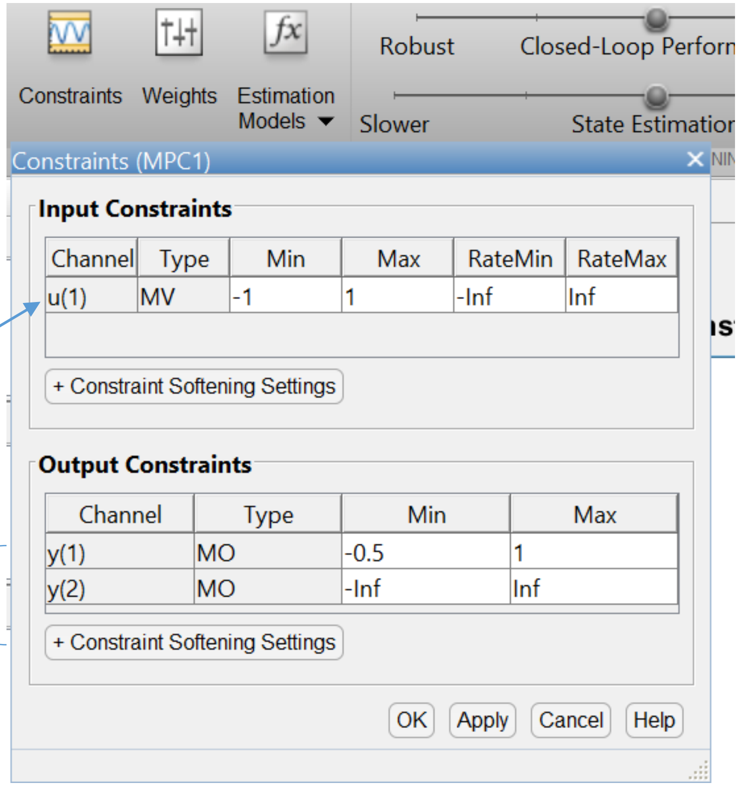
Step 5 – Set the time step and horizon lengths:



Step 6 – Set constraints (to the right of time step and horizon lengths):

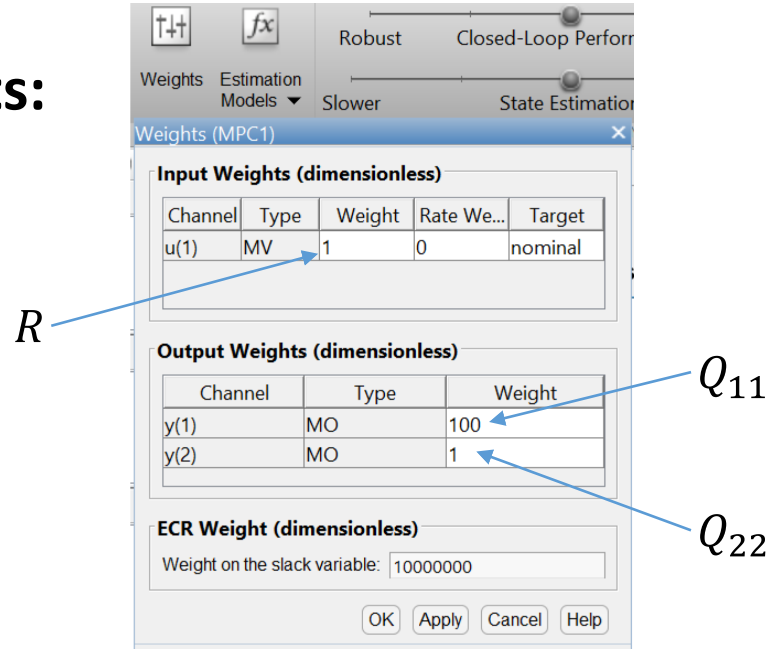
Control signal constraints

State constraints

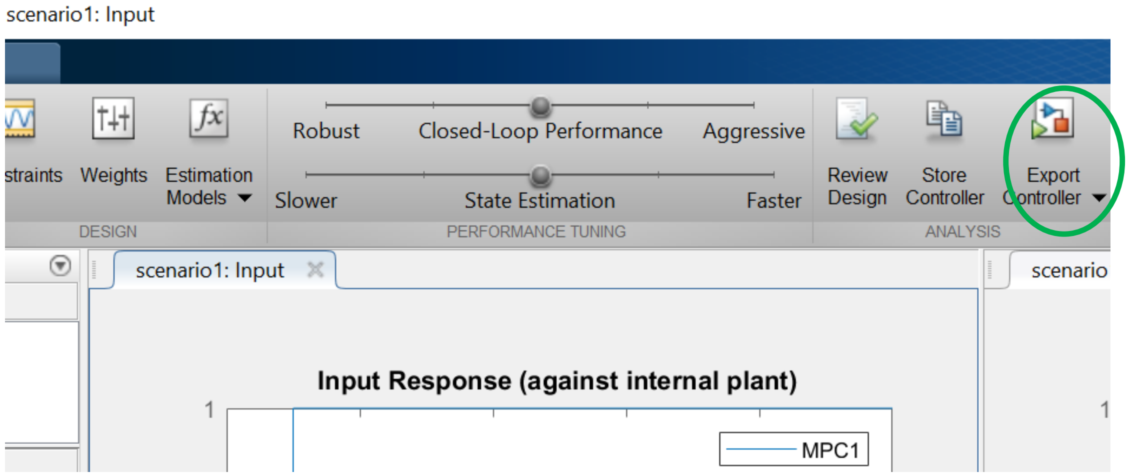


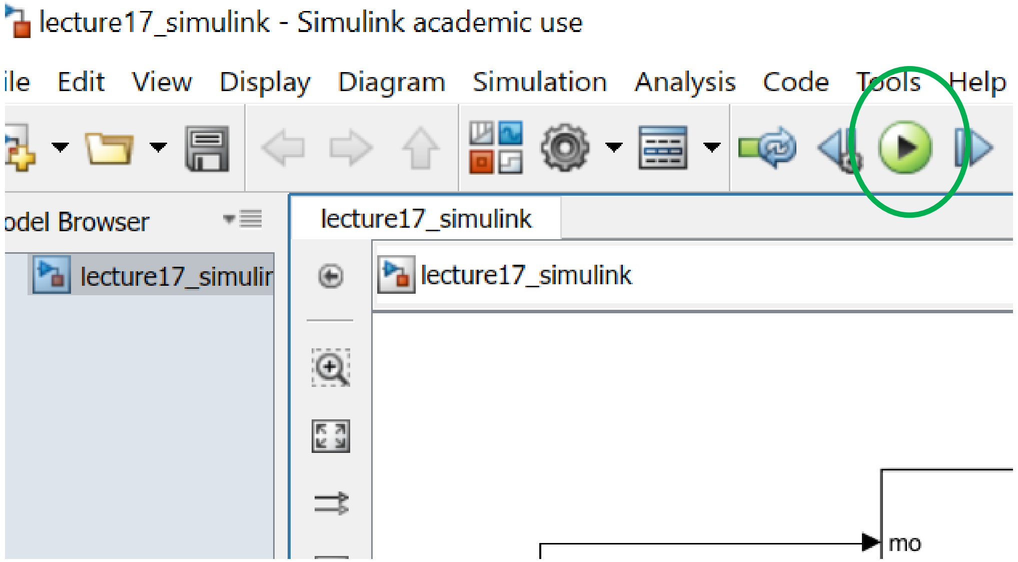
Introduction to MPC Toolbox

Step 7 – Set the cost function weights:



Step 8 – Export the controller and run the Simulink model:





Preview of Upcoming Lectures

Continuation of our MPC study:

- MPC implementation in MATLAB and Simulink (October 24)
- Stability and robustness of MPC (October 26)

After October 26 – Optimal control for continuous-time systems