THE UNIVERSITY OF NEWCASTLE AUSTRALIA

FACULTY OF ENGINEERING AND BUILT ENVIRONMENT

www.newcastle.edu.au

Constrained Control using Optimisation

"Model-based predictive control (MPC) […] is the only advanced control technique—that is, more advanced than standard PID control—to have had a significant and widespread impact on industrial process control." — J. M. Maciejowski

Dr. Chris Renton
School of Engineering

Constraints

Sequential Decisions

Model Predictive Ctrl

Constraints & Constrained Control

When the performance of a controlled physical system is to be maximised, dealing with actuator saturation and the possibility of constraint violation often becomes unavoidable.

When the control action differs from the control command, this is due to the fact that control strategy does not contemplate the limited authority of the actuators. The consequences are, in general, a degradation of the closed-loop performance, a reduction of the lifespan of the actuator and, in some cases, stability problems.

Apart from actuator saturation, most controlled systems are required to operate at the limits of their capabilities so as to maximise performance. This is reflected in specifications that require quantities of the system (states, inputs, outputs or combinations of them) to be within certain limits, and often as close as possible to those limits.

All these cases call for the design of a control strategy that can handle constraints.

Classification of Constraints

Constraints, here, are expressed in terms of limits on the magnitude of the quantities and signals involved in the control system and/or their time derivatives or time differences (discrete-time case).

- **Hard constraints** are those limits that are imposed by physical characteristics of the system to be controlled. Hence, hard constraints cannot be exceeded.

- **Soft constraints**, on the other hand, are limits that may be exceeded, temporarily, without any consequences other than poor performance.

Constraints can also be classified into **input constraints** and **state or output constraints**.

Constrained Control Strategies

- **Serendipitous:** This strategy ignores the constraints in the control design. Constraints may be reached occasionally with minor loss of performance and no risk of instability. E.g., controlled systems that cannot go unstable.

- **Cautious:** The control tuning is such that constraints are never reached during the envelope of plant operational conditions. This often results in poor performance for not exploiting the full potential of the control action.

- **Remedial:** The control design is initially done without considering the constraints. Then, special features are added to ensure that the constraint requirements are met. Examples are anti-wind-up and automatic gain control.

- **Tactical:** The constraints are considered as part of the control design *ab initio*, and operating conditions on, or near, the constraints are allowed. These strategies, generally, accomplish the best performance and yield more profitable results. Example: Model Predictive Control (MPC).

Finite-Horizon Sequential Decisions

We can think of control as a sequential decision problem (SDP).

SDP are characterized by three elements (Bertsekas, 1976, 2000):

- Dynamic System $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$
- Constraint sets $\mathbf{x}_k \in \mathcal{X}_k, \qquad \mathbf{u}_k \in \mathcal{U}_k$
- Cost (functional) $\mathcal{V} = g_T(\mathbf{x}_{N+1}) + \sum_{k=1}^{N} g_k(\mathbf{x}_k, \mathbf{u}_k)$

The cost is a scalar that measures deviations from the desired performance. One component is the final or terminal cost, and other is the cumulative cost associated with each decision (control $\mathbf{u}_k$)

Finite-Horizon Sequential Decisions

$$\mathcal{V} = g_T(\mathbf{x}_{N+1}) + \sum_{k=1}^{N} g_k(\mathbf{x}_k, \mathbf{u}_k)$$

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$$

$$\mathbf{x}_k \in \mathcal{X}_k, \qquad \mathbf{u}_k \in \mathcal{U}_k$$

Since the cost is a measure of deviation from the desired performance, it should be minimised.

Hence, we seek the sequence of control values that minimises the cost:

$$\mathbf{U}^* = \arg\min_{\mathbf{U}} \mathcal{V}$$

$$\mathbf{U}^N = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N\} \qquad \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$$

$$\mathbf{x}_k \in \mathcal{X}_k, \qquad \mathbf{u}_k \in \mathcal{U}_k$$

Finite-Horizon Sequential Decisions

$$\mathbf{U}^* = \arg\min_{\mathbf{U}} \mathcal{V}$$

$$\mathbf{U}^N = \{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N\}$$

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$$

$$\mathbf{x}_k \in \mathcal{X}_k, \qquad \mathbf{u}_k \in \mathcal{U}_k$$

There are two ways of solving the problem:

- **Open-loop optimisation**: We seek the sequence of control values that minimise the cost,

$$\mathbf{U}_N^* = \{\mathbf{u}_1^*, \mathbf{u}_2^*, \ldots, \mathbf{u}_N^*\}$$

- **Closed-loop optimisation**: we seek the control policy, or sequence of control actions as a function of the state that minimises the cost,

$$\mathbf{\Pi}_N^* = \{\boldsymbol{\pi}_1^*(\cdot), \boldsymbol{\pi}_2^*(\cdot), \ldots, \boldsymbol{\pi}_N^*(\cdot)\}$$

Finite-Horizon Sequential Decisions

What is the difference in the end result?

$$\mathbf{U}_N^* = \{\mathbf{u}_1^*, \mathbf{u}_2^*, \ldots, \mathbf{u}_N^*\}$$

$$\mathbf{\Pi}_N^* = \{\boldsymbol{\pi}_1^*(\cdot), \boldsymbol{\pi}_2^*(\cdot), \ldots, \boldsymbol{\pi}_N^*(\cdot)\}$$

If the problem is purely deterministic (no uncertainty) there is no difference:

$$\mathbf{u}_k^* = \boldsymbol{\pi}_k^*(\mathbf{x}_k)$$

If there is uncertainty, the policy approach is better, in general—the control value is updated based on the current state information, whereas in the open-loop solution all the control values are optimised at the initial stage, and state information is ignored (because it assumes that given the initial state, the rest of the state sequence can be known with certainty using the model and there is no need to measure the state)

Infinite-Horizon Decisions

In a control setting, we would like to take N to infinity.

The open-loop solution is not practical, because we need to optimise over an infinite sequence of control values.

The closed loop solution is the only valid option. But a solution can be found only if the optimal policy is stationary, that is all functions in the policy are the same.

One problem for which this holds is LQR (no constraints):

$$V = \sum_{k=0}^{\infty} \mathbf{x}_k^{\mathsf{T}} \mathbf{Q} \mathbf{x}_k + \mathbf{u}_k^{\mathsf{T}} \mathbf{R} \mathbf{u}_k$$

$$\mathbf{x}_{k+1} = \mathbf{A}\,\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$$

$$\Rightarrow \qquad \mathbf{u}_k^* = \boldsymbol{\pi}^*(\mathbf{x}_k) = -\mathbf{K}_{LQ}\,\mathbf{x}_k$$

How do we deal with constraints?

Receding-Horizon Control

A trade off is to use Receding-Horizon open-loop Optimal Control (RHC):

1. Select the sequence of control values that solves a finite-horizon open-loop minimisation based on the current information, and apply the first control value.
2. Repeat the optimisation once new information is available (measurements). This process continues indefinitely.

This may require the optimisation problem to be solved on-line, and we can easily incorporate constraints on inputs and states.

The rationale behind the RHC approximation is that decisions made at current time instant will most likely have little effect in the system response beyond certain horizon (unlike some life changing decisions).

The length of the horizon depends on the dynamics of the system under consideration.

Explicit vs Implicit RHC

The **explicit RHC implementation** requires to solve for the sequence of functions for a fixed horizon:

$$\mathbf{\Pi}_N^* = \{\boldsymbol{\pi}_1^*(\cdot), \boldsymbol{\pi}_2^*(\cdot), \ldots, \boldsymbol{\pi}_N^*(\cdot)\}$$

and then fix the control law to the first element of the policy:

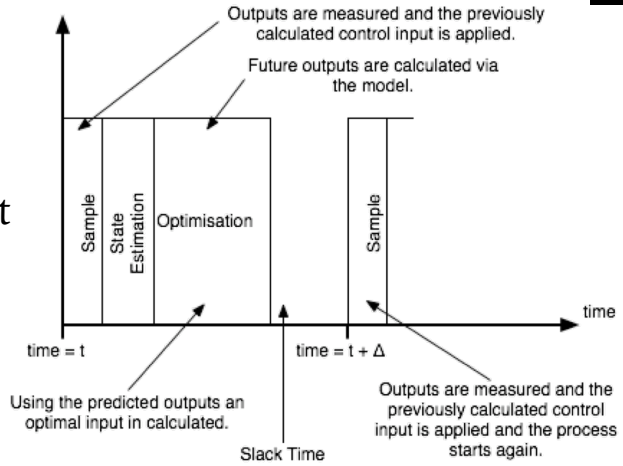
$$\mathbf{u}_{k+1}^* = \boldsymbol{\pi}_1^*(\mathbf{x}_k)$$

The **implicit RHC implementaion** requires solving the numerical optimisation problem online. In this case, the control law is not known explicitly; given the state, an optimisation algorithm computes the control values,

$$\mathbf{U}_N^* = \{\mathbf{u}_1^*, \mathbf{u}_2^*, \ldots, \mathbf{u}_N^*\}$$

$$\mathbf{u}_{k+1}^* \triangleq \mathbf{u}_1^*$$

Embedded implementation



For $k = 1, 2, \ldots$

1. Apply the stored control action $\mathbf{u}_k$ to the plant
2. Measure sensor data $\mathbf{y}_k$
3. Estimate state $\boldsymbol{\mu}_{k+1|k}$ from most recent measurements
4. Solve the optimisation problem to compute $\{\mathbf{u}_{k+1}, \mathbf{u}_{k+2}, \ldots \mathbf{u}_{k+N}\}$
5. Store $\mathbf{u}_{k+1}$ for the next time step

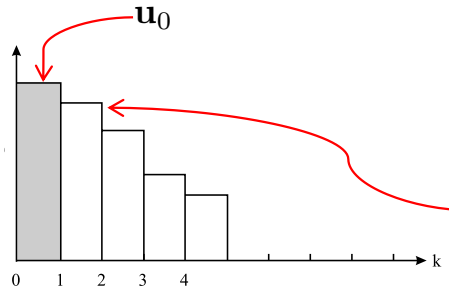
Since the system is time invariant, we can let the time indices in our notation refer to the change from the current time step (i.e., $k = 0$).

e.g., let $\mathbf{u}_0$ be the current control action, and $\boldsymbol{\mu}_{1|0}$ is the estimated step at the next time step and $\mathbf{u}_1$ is the next control action.

Implicit Receding-Horizon Control

Horizon
$N = 5$

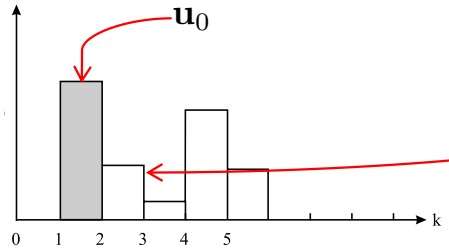
$k = 1$



Given $\mathbf{x}_1$

$\mathbf{U}_5^* = \{\mathbf{u}_1^*, \mathbf{u}_2^*, \ldots, \mathbf{u}_5^*\}$

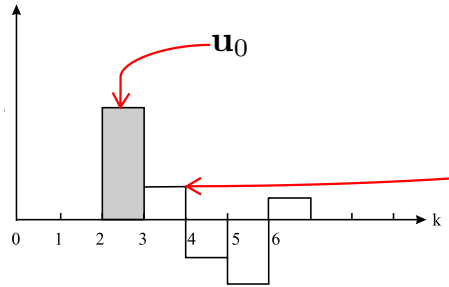
$k = 2$



Given $\mathbf{x}_2$

$\mathbf{U}_5^* = \{\mathbf{u}_1^*, \mathbf{u}_2^*, \ldots, \mathbf{u}_5^*\}$

$k = 3$



Given $\mathbf{x}_3$

$\mathbf{U}_5^* = \{\mathbf{u}_1^*, \mathbf{u}_2^*, \ldots, \mathbf{u}_5^*\}$

Model Predictive Control

Model Predictive Control *is a receding-horizon control strategy that solves, online, a constrained optimisation problem associated with a finite-horizon open-loop sequential-decision problem.*

The most important feature of MPC is its ability to handle multivariable constrained control problems with different types of constraints for which offline computations of a control law are in general impossible to obtain.

Because of this, MPC has become an common tool in industrial control engineering and a method of choice for some advanced control applications.

MPC is perhaps the only advanced control technology beyond PID that is significantly wide-spread in industry.

MPC was born in industry and it took 20 years before academics and researchers took interest in it and further contributed to its development (stability, robustness, implementation, etc.).

Unconstrained Linear Systems with Quadratic Cost

Let's start with a basic problem.

Consider a linear system: $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$

and the cost:
$$\mathcal{V} = \frac{1}{2}(\mathbf{x}_{N+1} - \mathbf{x}^*_{N+1})^\mathsf{T}\mathbf{Q}_f(\mathbf{x}_{N+1} - \mathbf{x}^*_{N+1}) + \sum_{k=1}^{N} \frac{1}{2}(\mathbf{x}_k - \mathbf{x}^*_k)^\mathsf{T}\mathbf{Q}(\mathbf{x}_k - \mathbf{x}^*_k) + \frac{1}{2}\mathbf{u}_k^\mathsf{T}\mathbf{R}\mathbf{u}_k$$

We seek
$$\mathbf{U}^* = \arg\min_{\mathbf{U}} \mathcal{V}(\mathbf{x}_1, \mathbf{U}^N)$$
$$\mathbf{U}^*_N = \{\mathbf{u}^*_1, \mathbf{u}^*_2, \ldots, \mathbf{u}^*_N\}$$

Unconstrained Linear Systems with Quadratic Cost

Let us propagate the model forward: $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k$

$$
\begin{aligned}
\mathbf{x}_2 &= \mathbf{A}\mathbf{x}_1 + \mathbf{B}\mathbf{u}_1 \\
\mathbf{x}_3 &= \mathbf{A}\mathbf{x}_2 + \mathbf{B}\mathbf{u}_2 = \mathbf{A}^2\mathbf{x}_1 + \mathbf{A}\mathbf{B}\mathbf{u}_1 + \mathbf{B}\mathbf{u}_2 \\
\mathbf{x}_4 &= \mathbf{A}\mathbf{x}_3 + \mathbf{B}\mathbf{u}_3 = \mathbf{A}^3\mathbf{x}_1 + \mathbf{A}^2\mathbf{B}\mathbf{u}_1 + \mathbf{A}\mathbf{B}\mathbf{u}_2 + \mathbf{B}\mathbf{u}_3 \\
&\vdots \\
\mathbf{x}_{N+1} &= \mathbf{A}^N\mathbf{x}_1 + \sum_{k=1}^{N} \mathbf{A}^{N-k}\mathbf{B}\mathbf{u}_k
\end{aligned}
$$

Unconstrained Linear Systems with Quadratic Cost

Let us express this in a matrix form for $N = 3$:

$$\mathbf{x}_2 = \mathbf{A}\mathbf{x}_1 + \mathbf{B}\mathbf{u}_1$$
$$\mathbf{x}_3 = \mathbf{A}\mathbf{x}_2 + \mathbf{B}\mathbf{u}_2 = \mathbf{A}^2\mathbf{x}_1 + \mathbf{A}\mathbf{B}\mathbf{u}_1 + \mathbf{B}\mathbf{u}_2$$
$$\mathbf{x}_4 = \mathbf{A}\mathbf{x}_3 + \mathbf{B}\mathbf{u}_3 = \mathbf{A}^3\mathbf{x}_1 + \mathbf{A}^2\mathbf{B}\mathbf{u}_1 + \mathbf{A}\mathbf{B}\mathbf{u}_2 + \mathbf{B}\mathbf{u}_3$$

$$\Downarrow$$

$$\mathfrak{x} = \mathcal{A}\mathbf{x}_1 + \mathcal{B}\mathfrak{u}$$

$$\mathfrak{x} \triangleq \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \mathbf{x}_4 \end{bmatrix} \qquad \mathcal{A} \triangleq \begin{bmatrix} \mathbf{I} \\ \mathbf{A} \\ \mathbf{A}^2 \\ \mathbf{A}^3 \end{bmatrix} \qquad \mathcal{B} \triangleq \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{B} & \mathbf{0} & \mathbf{0} \\ \mathbf{A}\mathbf{B} & \mathbf{B} & \mathbf{0} \\ \mathbf{A}^2\mathbf{B} & \mathbf{A}\mathbf{B} & \mathbf{B} \end{bmatrix} \qquad \mathfrak{u} \triangleq \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \end{bmatrix}$$

Unconstrained Linear Systems with Quadratic Cost

The forward propagation for a general horizon N is

$$\mathfrak{x} = \mathcal{A}\mathbf{x}_1 + \mathcal{B}\mathfrak{u}$$

$$\mathfrak{x} \triangleq \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \vdots \\ \mathbf{x}_{N+1} \end{bmatrix} \quad \mathcal{A} \triangleq \begin{bmatrix} \mathbf{I} \\ \mathbf{A} \\ \mathbf{A}^2 \\ \vdots \\ \mathbf{A}^N \end{bmatrix} \quad \mathcal{B} \triangleq \begin{bmatrix} \mathbf{0} & \cdots & \cdots & \mathbf{0} \\ \mathbf{B} & \ddots & & \vdots \\ \mathbf{AB} & \mathbf{B} & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{0} \\ \mathbf{A}^{N-1}\mathbf{B} & \cdots & \mathbf{AB} & \mathbf{B} \end{bmatrix} \quad \mathfrak{u} \triangleq \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_N \end{bmatrix}$$

Unconstrained Linear Systems with Quadratic Cost

We can work in a similar way with the cost:

$$\mathcal{V} = \frac{1}{2}(\mathbf{x}_{N+1} - \mathbf{x}^*_{N+1})^\mathsf{T}\mathbf{Q}_f(\mathbf{x}_{N+1} - \mathbf{x}^*_{N+1}) + \sum_{k=1}^{N} \frac{1}{2}(\mathbf{x}_k - \mathbf{x}^*_k)^\mathsf{T}\mathbf{Q}(\mathbf{x}_k - \mathbf{x}^*_k) + \frac{1}{2}\mathbf{u}_k^\mathsf{T}\mathbf{R}\mathbf{u}_k$$

$$\Downarrow$$

$$\mathcal{V} = \frac{1}{2}(\mathfrak{x} - \mathfrak{x}^*)^\mathsf{T}\mathcal{Q}(\mathfrak{x} - \mathfrak{x}^*) + \frac{1}{2}\mathfrak{u}^\mathsf{T}\mathcal{R}\mathfrak{u}$$

$$\mathcal{Q} = \mathrm{diag}(\mathbf{Q}, \ldots, \mathbf{Q}, \mathbf{Q}_f)$$
$$\mathcal{R} = \mathrm{diag}(\mathbf{R}, \ldots, \mathbf{R})$$

Unconstrained Linear Systems with Quadratic Cost

Substituting

$$\mathfrak{x} = \mathcal{A}\mathbf{x}_1 + \mathcal{B}\mathfrak{u}$$

into

$$\mathcal{V} = \frac{1}{2}(\mathfrak{x} - \mathfrak{x}^*)^\mathsf{T}\mathcal{Q}(\mathfrak{x} - \mathfrak{x}^*) + \frac{1}{2}\mathfrak{u}^\mathsf{T}\mathcal{R}\mathfrak{u}$$

we obtain a quadratic function:

$$\mathcal{V} = \frac{1}{2}\mathfrak{u}^\mathsf{T}\mathcal{H}\mathfrak{u} + \mathfrak{g}^\mathsf{T}\mathfrak{u} + \text{const}$$

$$\mathcal{H} = \mathcal{B}^\mathsf{T}\mathcal{Q}\mathcal{B} + \mathcal{R} \qquad \mathcal{Q} = \text{diag}(\mathbf{Q}, \ldots, \mathbf{Q}, \mathbf{Q}_f)$$

$$\mathfrak{g} = \mathcal{B}^\mathsf{T}\mathcal{Q}(\mathcal{A}\mathbf{x}_1 - \mathfrak{x}^*) \qquad \mathcal{R} = \text{diag}(\mathbf{R}, \ldots, \mathbf{R})$$

Unconstrained Linear Systems with Quadratic Cost

Then, the finite-horizon problem reduces to the minimisation of a quadratic function:

$$u^* = \arg\min_u \frac{1}{2} u^\mathsf{T} \mathcal{H} u + \mathfrak{g}^\mathsf{T} u$$

Since, we have not considered constraints, this minimum is given by

$$u^* : \frac{\partial}{\partial u}\left[\frac{1}{2} u^\mathsf{T} \mathcal{H} u + \mathfrak{g}^\mathsf{T} u\right]\Big|_{u=u^*} = \mathbf{0} \quad \Rightarrow \quad u^* = -\mathcal{H}^{-1}\mathfrak{g}$$

Provided that the Hessian is symmetric and positive definite:

$$\frac{\partial^2}{\partial u^2}\left[\frac{1}{2} u^\mathsf{T} \mathcal{H} u + \mathfrak{g}^\mathsf{T} u\right]\Big|_{u=u^*} = \mathcal{H} > 0 \quad \Leftrightarrow \quad \mathbf{R} > 0, \; \mathbf{Q} \geq 0, \; \mathbf{Q}_f \geq 0$$

Convexity: one global minimiser

Input constraints

Consider the following saturation and slew rate constraints:

$$\mathbf{u}_{\min} \leq \mathbf{u}_k \leq \mathbf{u}_{\max}, \quad k = 1, \ldots, N$$
$$\Delta\mathbf{u}_{\min} \leq \mathbf{u}_k - \mathbf{u}_{k-1} \leq \Delta\mathbf{u}_{\max}, \quad k = 1, \ldots, N$$

These can be expressed as $\mathcal{L}\mathfrak{u} \leq \mathfrak{m}$

$$\mathcal{L} = \begin{bmatrix} \mathbf{I} \\ -\mathbf{I} \\ \mathcal{W} \\ -\mathcal{W} \end{bmatrix}, \quad \mathfrak{m} = \begin{bmatrix} \mathfrak{u}_{\max} \\ -\mathfrak{u}_{\min} \\ \Delta\mathfrak{u}_{\max} \\ -\Delta\mathfrak{u}_{\min} \end{bmatrix} \qquad \mathfrak{u} \triangleq \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_N \end{bmatrix} \qquad \mathfrak{u}_{\min} \triangleq \begin{bmatrix} \mathbf{u}_{\min} \\ \mathbf{u}_{\min} \\ \vdots \\ \mathbf{u}_{\min} \end{bmatrix} \qquad \mathfrak{u}_{\max} \triangleq \begin{bmatrix} \mathbf{u}_{\max} \\ \mathbf{u}_{\max} \\ \vdots \\ \mathbf{u}_{\max} \end{bmatrix}$$

$$\mathcal{W} \triangleq \begin{bmatrix} \mathbf{I} & \mathbf{0} & \cdots & \cdots & \mathbf{0} \\ -\mathbf{I} & \mathbf{I} & \ddots & & \vdots \\ \mathbf{0} & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & -\mathbf{I} & \mathbf{I} \end{bmatrix} \qquad \Delta\mathfrak{u}_{\min} \triangleq \begin{bmatrix} \mathbf{u}_0 + \Delta\mathbf{u}_{\min} \\ \Delta\mathbf{u}_{\min} \\ \vdots \\ \Delta\mathbf{u}_{\min} \end{bmatrix} \qquad \Delta\mathfrak{u}_{\max} \triangleq \begin{bmatrix} \mathbf{u}_0 + \Delta\mathbf{u}_{\max} \\ \Delta\mathbf{u}_{\max} \\ \vdots \\ \Delta\mathbf{u}_{\max} \end{bmatrix}$$

State constraints

Consider the following state constraints:

$$\mathbf{x}_{\min} \leq \mathbf{x}_k \leq \mathbf{x}_{\max}, \quad k = 2, \ldots, N$$

Write constraints as $\mathfrak{x}_{\min} \leq \mathfrak{x}_k \leq \mathfrak{x}_{\max}$

where $\mathfrak{x} = \mathcal{A}\mathbf{x}_1 + \mathcal{B}\mathfrak{u}$

$$\mathfrak{x}_{\min} \triangleq \begin{bmatrix} \mathbf{x}_{\min} \\ \mathbf{x}_{\min} \\ \vdots \\ \mathbf{x}_{\min} \end{bmatrix} \qquad \mathfrak{x}_{\max} \triangleq \begin{bmatrix} \mathbf{x}_{\max} \\ \mathbf{x}_{\max} \\ \vdots \\ \mathbf{x}_{\max} \end{bmatrix}$$

These can be expressed as $\mathcal{L}\mathfrak{u} \leq \mathfrak{m}$

$$\mathcal{L} = \begin{bmatrix} \mathcal{B} \\ -\mathcal{B} \end{bmatrix}, \quad \mathfrak{m} = \begin{bmatrix} \mathfrak{x}_{\max} - \mathcal{A}\mathbf{x}_1 \\ -\mathfrak{x}_{\min} + \mathcal{A}\mathbf{x}_1 \end{bmatrix} \qquad \mathfrak{u} \triangleq \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_N \end{bmatrix}$$

Select the rows from $\mathbf{L}$ and $\mathbf{m}$ as needed to implement the desired constraints.

Output tracking and move penalty

A cost function of particular interest in industry is the following:

$$\mathcal{V} = \sum_{k=1}^{\infty} \frac{1}{2}(\mathbf{y}_k - \mathbf{y}_k^*)^\mathsf{T}\mathbf{Q}_y(\mathbf{y}_k - \mathbf{y}_k^*) + \frac{1}{2}(\mathbf{u}_k - \mathbf{u}_{k-1})^\mathsf{T}\mathbf{R}_u(\mathbf{u}_k - \mathbf{u}_{k-1})$$

This penalises the deviations of the output from a desired trajectory and penalises the change in actuator setting between time steps.

To do this, let's augment the state with the previous input

$$\underbrace{\begin{bmatrix}\mathbf{x}_{k+1}\\ \mathbf{u}_k\end{bmatrix}}_{\mathbf{z}_{k+1}} = \underbrace{\begin{bmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{0} & \mathbf{I}\end{bmatrix}}_{\mathbf{A}_z}\underbrace{\begin{bmatrix}\mathbf{x}_k\\ \mathbf{u}_{k-1}\end{bmatrix}}_{\mathbf{z}_k} + \underbrace{\begin{bmatrix}\mathbf{B}\\ \mathbf{I}\end{bmatrix}}_{\mathbf{B}_z}\underbrace{(\mathbf{u}_k - \mathbf{u}_{k-1})}_{\Delta\mathbf{u}_k},$$

$$\mathbf{y}_k = \underbrace{\begin{bmatrix}\mathbf{C} & \mathbf{D}\end{bmatrix}}_{\mathbf{C}_z}\underbrace{\begin{bmatrix}\mathbf{x}_k\\ \mathbf{u}_{k-1}\end{bmatrix}}_{\mathbf{z}_k} + \underbrace{\mathbf{D}}_{\mathbf{D}_z}\underbrace{(\mathbf{u}_k - \mathbf{u}_{k-1})}_{\Delta\mathbf{u}_k}.$$

Output tracking and move penalty

Then, the cost function can be written as

$$\mathcal{V} = \frac{1}{2}(\mathbf{z}_{N+1} - \mathbf{z}^*_{N+1})^\mathsf{T}\mathbf{Q}_f(\mathbf{z}_{N+1} - \mathbf{z}^*_{N+1}) + \sum_{k=1}^{N} \frac{1}{2}(\mathbf{z}_k - \mathbf{z}^*_k)^\mathsf{T}\mathbf{Q}_z(\mathbf{z}_k - \mathbf{z}^*_k) + \frac{1}{2}\Delta\mathbf{u}_k^\mathsf{T}\mathbf{R}_z\Delta\mathbf{u}_k + (\mathbf{z}_k - \mathbf{z}^*_k)^\mathsf{T}\mathbf{S}_z\Delta\mathbf{u}_k.$$

where $\mathbf{Q}_f$ is the solution to the algebraic Riccati equation

$$\mathbf{Q}_f = \mathbf{A}_z^\mathsf{T}\mathbf{Q}_f\mathbf{A}_z - (\mathbf{A}_z^\mathsf{T}\mathbf{Q}_f\mathbf{B}_z + \mathbf{S}_z)(\mathbf{B}_z^\mathsf{T}\mathbf{Q}_f\mathbf{B}_z + \mathbf{R}_z)^{-1}(\mathbf{B}_z^\mathsf{T}\mathbf{Q}_f\mathbf{A}_z + \mathbf{S}_z^\mathsf{T}) + \mathbf{Q}_z$$

and

$$\mathbf{Q}_z = \mathbf{C}_z^\mathsf{T}\mathbf{Q}_y\mathbf{C}_z \qquad \mathbf{z}^*_k = \begin{bmatrix}\mathbf{x}^*_k \\ \mathbf{u}^*_{k-1}\end{bmatrix} = \begin{bmatrix}\mathbf{N}_x\mathbf{y}^*_k \\ \mathbf{N}_u\mathbf{y}^*_{k-1}\end{bmatrix}$$

$$\mathbf{S}_z = \mathbf{C}_z^\mathsf{T}\mathbf{Q}_y\mathbf{D}_z$$

$$\mathbf{R}_z = \mathbf{D}_z^\mathsf{T}\mathbf{Q}_y\mathbf{D}_z + \mathbf{R}_u \qquad \mathbf{z}^*_{N+1} = \begin{bmatrix}\mathbf{x}^*_{ss} \\ \mathbf{u}^*_{ss}\end{bmatrix} = \begin{bmatrix}\mathbf{N}_x\mathbf{y}^*_\infty \\ \mathbf{N}_u\mathbf{y}^*_\infty\end{bmatrix}$$

$$\begin{bmatrix}\mathbf{A}-\mathbf{I} & \mathbf{B} \\ \mathbf{C} & \mathbf{D}\end{bmatrix}\begin{bmatrix}\mathbf{N}_x \\ \mathbf{N}_u\end{bmatrix} = \begin{bmatrix}\mathbf{0} \\ \mathbf{I}\end{bmatrix}$$

Output tracking and move penalty

The cost function can be written compactly as

$$\mathcal{V} = \frac{1}{2}(\mathfrak{z} - \mathfrak{z}^*)^\mathsf{T}\mathcal{Q}_z(\mathfrak{z} - \mathfrak{z}^*) + \frac{1}{2}\Delta\mathfrak{u}^\mathsf{T}\mathcal{R}_z\Delta\mathfrak{u} + (\mathfrak{z} - \mathfrak{z}^*)^\mathsf{T}\mathcal{S}_z\Delta\mathfrak{u}$$

$$\mathfrak{z} = \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \\ \mathbf{z}_N \\ \mathbf{z}_{N+1} \end{bmatrix}, \quad \mathfrak{z}^* = \begin{bmatrix} \mathbf{z}_1^* \\ \mathbf{z}_2^* \\ \vdots \\ \mathbf{z}_N^* \\ \mathbf{z}_{N+1}^* \end{bmatrix}, \quad \mathcal{Q}_z = \begin{bmatrix} \mathbf{Q}_z & \mathbf{0} & \cdots & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_z & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \mathbf{Q}_z & \mathbf{0} \\ \mathbf{0} & \cdots & \cdots & \mathbf{0} & \mathbf{Q}_f \end{bmatrix}, \quad \mathcal{S}_z = \begin{bmatrix} \mathbf{S}_z & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_z & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{0} \\ \vdots & & \ddots & \mathbf{S}_z \\ \mathbf{0} & \cdots & \cdots & \mathbf{0} \end{bmatrix}$$

$$\Delta\mathfrak{u} = \begin{bmatrix} \mathbf{u}_1 - \mathbf{u}_0 \\ \mathbf{u}_2 - \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_N - \mathbf{u}_{N-1} \end{bmatrix}, \quad \mathcal{R}_z = \begin{bmatrix} \mathbf{R}_z & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_z & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{R}_z \end{bmatrix}$$

Output tracking and move penalty

The $\mathbf{z}$ state over N+1 steps is given by

$$\underbrace{\begin{bmatrix}\mathbf{z}_1\\ \mathbf{z}_2\\ \mathbf{z}_3\\ \vdots\\ \mathbf{z}_{N+1}\end{bmatrix}}_{\mathfrak{z}} = \underbrace{\begin{bmatrix}\mathbf{I}\\ \mathbf{A}_z\\ \mathbf{A}_z^2\\ \vdots\\ \mathbf{A}_z^N\end{bmatrix}}_{\mathcal{A}_z} \underbrace{\begin{bmatrix}\mathbf{x}_1\\ \mathbf{u}_0\end{bmatrix}}_{\mathbf{z}_1} + \underbrace{\begin{bmatrix}\mathbf{0} & \cdots & \cdots & \mathbf{0}\\ \mathbf{B}_z & \ddots & & \vdots\\ \mathbf{A}_z\mathbf{B}_z & \mathbf{B}_z & \ddots & \vdots\\ \vdots & \ddots & \ddots & \mathbf{0}\\ \mathbf{A}_z^{N-1}\mathbf{B}_z & \cdots & \mathbf{A}_z\mathbf{B}_z & \mathbf{B}_z\end{bmatrix}}_{\mathcal{B}_z} \underbrace{\begin{bmatrix}\mathbf{u}_1-\mathbf{u}_0\\ \mathbf{u}_2-\mathbf{u}_1\\ \vdots\\ \mathbf{u}_N-\mathbf{u}_{N-1}\end{bmatrix}}_{\Delta\mathfrak{u}}$$

$$\mathfrak{z} = \mathcal{A}_z\mathbf{z}_1 + \mathcal{B}_z\Delta\mathfrak{u}$$

where

$$\underbrace{\begin{bmatrix}\mathbf{u}_1-\mathbf{u}_0\\ \mathbf{u}_2-\mathbf{u}_1\\ \mathbf{u}_3-\mathbf{u}_2\\ \vdots\\ \mathbf{u}_N-\mathbf{u}_{N-1}\end{bmatrix}}_{\Delta\mathfrak{u}} = \underbrace{\begin{bmatrix}\mathbf{0} & -\mathbf{I}\\ \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\\ \vdots & \vdots\\ \mathbf{0} & \mathbf{0}\end{bmatrix}}_{\mathcal{E}} \underbrace{\begin{bmatrix}\mathbf{x}_1\\ \mathbf{u}_0\end{bmatrix}}_{\mathbf{z}_1} + \underbrace{\begin{bmatrix}\mathbf{I} & \mathbf{0} & \cdots & \cdots & \mathbf{0}\\ -\mathbf{I} & \mathbf{I} & \ddots & & \vdots\\ \mathbf{0} & \ddots & \ddots & \ddots & \vdots\\ \vdots & \ddots & \ddots & \ddots & \mathbf{0}\\ \mathbf{0} & \cdots & \mathbf{0} & -\mathbf{I} & \mathbf{I}\end{bmatrix}}_{\mathcal{W}} \underbrace{\begin{bmatrix}\mathbf{u}_1\\ \mathbf{u}_2\\ \mathbf{u}_3\\ \vdots\\ \mathbf{u}_N\end{bmatrix}}_{\mathfrak{u}}$$

$$\Delta\mathfrak{u} = \mathcal{E}\mathbf{z}_1 + \mathcal{W}\mathfrak{u}$$

Output tracking and move penalty

Substituting $\mathfrak{z} = \mathcal{A}_z \mathbf{z}_1 + \mathcal{B}_z \Delta\mathfrak{u}$ and $\Delta\mathfrak{u} = \mathcal{E}\mathbf{z}_1 + \mathcal{W}\mathfrak{u}$

$$\begin{aligned}\mathfrak{z} &= \mathcal{A}_z \mathbf{z}_1 + \mathcal{B}_z(\mathcal{E}\mathbf{z}_1 + \mathcal{W}\mathfrak{u}) \\ &= (\mathcal{A}_z + \mathcal{B}_z\mathcal{E})\mathbf{z}_1 + \mathcal{B}_z\mathcal{W}\mathfrak{u}\end{aligned}$$

into the cost function and simplifying yields

$$\begin{aligned}\mathcal{V} &= \frac{1}{2}(\mathfrak{z} - \mathfrak{z}^*)^\mathsf{T}\mathcal{Q}_z(\mathfrak{z} - \mathfrak{z}^*) + \frac{1}{2}\Delta\mathfrak{u}^\mathsf{T}\mathcal{R}_z\Delta\mathfrak{u} + (\mathfrak{z} - \mathfrak{z}^*)^\mathsf{T}\mathcal{S}_z\Delta\mathfrak{u} \\ &= \frac{1}{2}((\mathcal{A}_z + \mathcal{B}_z\mathcal{E})\mathbf{z}_1 + \mathcal{B}_z\mathcal{W}\mathfrak{u} - \mathfrak{z}^*)^\mathsf{T}\mathcal{Q}_z((\mathcal{A}_z + \mathcal{B}_z\mathcal{E})\mathbf{z}_1 + \mathcal{B}_z\mathcal{W}\mathfrak{u} - \mathfrak{z}^*) \\ &\quad + \frac{1}{2}(\mathcal{E}\mathbf{z}_1 + \mathcal{W}\mathfrak{u})^\mathsf{T}\mathcal{R}_z(\mathcal{E}\mathbf{z}_1 + \mathcal{W}\mathfrak{u}) + ((\mathcal{A}_z + \mathcal{B}_z\mathcal{E})\mathbf{z}_1 + \mathcal{B}_z\mathcal{W}\mathfrak{u} - \mathfrak{z}^*)^\mathsf{T}\mathcal{S}_z(\mathcal{E}\mathbf{z}_1 + \mathcal{W}\mathfrak{u})\end{aligned}$$

$$\begin{aligned}\mathcal{V} &= \frac{1}{2}\mathfrak{u}^\mathsf{T}\underbrace{\mathcal{W}^\mathsf{T}(\mathcal{B}_z^\mathsf{T}\mathcal{Q}_z\mathcal{B}_z + \mathcal{B}_z^\mathsf{T}\mathcal{S}_z + \mathcal{S}_z^\mathsf{T}\mathcal{B}_z + \mathcal{R}_z)\mathcal{W}}_{\mathcal{H}}\mathfrak{u} \\ &\quad + \Big(\underbrace{\mathcal{W}^\mathsf{T}\left(((\mathcal{B}_z^\mathsf{T}\mathcal{Q}_z + \mathcal{S}_z^\mathsf{T})(\mathcal{A}_z + \mathcal{B}_z\mathcal{E}) + (\mathcal{B}_z^\mathsf{T}\mathcal{S}_z + \mathcal{R}_z)\mathcal{E}\right)\mathbf{z}_1 - (\mathcal{B}_z^\mathsf{T}\mathcal{Q}_z + \mathcal{S}_z^\mathsf{T})\mathfrak{z}^*\right)}_{\mathfrak{g}}\Big)^\mathsf{T}\mathfrak{u} \\ &\quad + \text{const}\end{aligned}$$

Quadratic Programming

Hence the MPC solution requires to solve at each sampling period a quadratic programming (QP) problem:

$$u^* = \arg\min_u \frac{1}{2} u^\top \mathcal{H} u + \mathfrak{g}^\top u$$
$$\text{s.t.} \quad \mathcal{L} u \leq \mathfrak{m}$$

QP is a standard problem in numerical optimisation.

Matlab has the function `quadprog` that solves this problem.

Every optimisation algorithm starts with a potential solution and iterates to modify this potential solution and obtain the one with the minimum cost.

In the case of MPC, we can use the solution of the previous time step to initialise the QP at the current time step. This is called **hot start**.

QP Solvers

- Active set methods (ASM): They have the advantage that in each iteration of the algorithm a feasible solution, which lies on the boundary of the feasible region, is obtained and the cost is reduced relative to the solution in the preceding iteration. This is a good feature when we run out of time to solve the full problem within the sample period because we can output a feasible, although not optimal, solution. A disadvantage is that the iteration needs to start from a feasible candidate solution, and this may require solution of a linear programming problem to initialise.

- Interior point methods (IPM): These methods have less computational complexity than ASM. The problem is transformed into a non-quadratic nonlinear unconstrained problem using barrier functions. They can be fast for some problems, but there is no guarantee of finding a solution in a particular time interval, and depending on the initialisation, we may not have feasible iterates until the iterations are finished.

- There is an extensive literature on these methods in optimisation and some on their application to MPC. See references at the end of the slides.

Feasibility and Constraint Relaxation

One of the main issues when we have inputs and output constraints at the same time, is that of feasibility.

It can happen that no feasible solution would satisfy the constraints simultaneously—this happens if we have an unexpected large disturbance.

Standard QP solvers can flag this situation, but we still need to compute a control value.

When this happens we need to relax the output constraints.

Feasibility and Constraint Relaxation

A way to relax constraints is by incorporating **slack variables** in the QP:

2-norm on slack variables

$$[\mathfrak{u}^*, \mathbf{s}^*] = \arg\min_{\mathfrak{u},\mathbf{s}} \frac{1}{2}\mathfrak{u}^\mathsf{T}\mathcal{H}\mathfrak{u} + \mathfrak{g}^\mathsf{T}\mathfrak{u} + \rho\,\mathbf{s}^\mathsf{T}\mathbf{s}$$
$$\text{s.t.} \quad \mathcal{L}\,\mathfrak{u} \leq \mathfrak{m} + \mathbf{s}$$
$$\mathbf{s} \geq \mathbf{0}$$

∞-norm on slack variables

$$[\mathfrak{u}^*, \mathbf{s}^*] = \arg\min_{\mathfrak{u},\mathbf{s}} \frac{1}{2}\mathfrak{u}^\mathsf{T}\mathcal{H}\mathfrak{u} + \mathfrak{g}^\mathsf{T}\mathfrak{u} + \rho \max_j s_j$$
$$\text{s.t.} \quad \mathcal{L}\,\mathfrak{u} \leq \mathfrak{m} + \mathbf{s}$$
$$\mathbf{s} \geq \mathbf{0}$$

1-norm on slack variables

$$[\mathfrak{u}^*, \mathbf{s}^*] = \arg\min_{\mathfrak{u},\mathbf{s}} \frac{1}{2}\mathfrak{u}^\mathsf{T}\mathcal{H}\mathfrak{u} + \mathfrak{g}^\mathsf{T}\mathfrak{u} + \rho \sum_j s_j$$
$$\text{s.t.} \quad \mathcal{L}\,\mathfrak{u} \leq \mathfrak{m} + \mathbf{s}$$
$$\mathbf{s} \geq \mathbf{0}$$

1-norm on slack variables

$$[\mathfrak{u}^*, s^*] = \arg\min_{\mathfrak{u},\mathbf{s}} \frac{1}{2}\mathfrak{u}^\mathsf{T}\mathcal{H}\mathfrak{u} + \mathfrak{g}^\mathsf{T}\mathfrak{u} + \rho s$$
$$\text{s.t.} \quad \mathcal{L}\,\mathfrak{u} \leq \mathfrak{m} + s\mathbf{1}$$
$$s \geq 0$$

The 2-norm always provides slack variables with a small value even if it is not needed.

The 1-norm or ∞-norm ensure that slack variables take a non-zero value only when there is no feasible solution.

Horizons & Sampling

The longer the horizon N, usually, the better the result since we can accommodate constraints further into the future. However, beyond certain values there will be no change in performance—this is because the impact a control action has will eventually lose importance relative to the more recent control actions.

There is a limit on how much we can increase the horizon since it increases the size of the online optimisation problem, and it takes longer to solve. In addition, if we have disturbances, these need to be predicted, and longer the horizon the more uncertain the predictions.

If you are sampling very fast, then you will need a long horizon. So there is a trade-off. In some cases you may have to sample a bit slower to reduce the complexity of the problem.

Tuning & Stability

$$\mathcal{V} = \frac{1}{2}(\mathbf{x}_{N+1} - \mathbf{x}^*_{N+1})^\mathsf{T}\mathbf{Q}_f(\mathbf{x}_{N+1} - \mathbf{x}^*_{N+1}) + \sum_{k=1}^{N}\frac{1}{2}(\mathbf{x}_k - \mathbf{x}^*_k)^\mathsf{T}\mathbf{Q}(\mathbf{x}_k - \mathbf{x}^*_k) + \frac{1}{2}\mathbf{u}_k^\mathsf{T}\mathbf{R}\mathbf{u}_k$$

The tuning requires the following parameters:

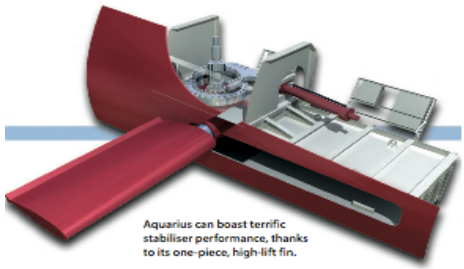
$$N \geq 1 \qquad \mathbf{R} > 0 \qquad \mathbf{Q} \geq 0 \qquad \mathbf{Q}_f \geq 0$$

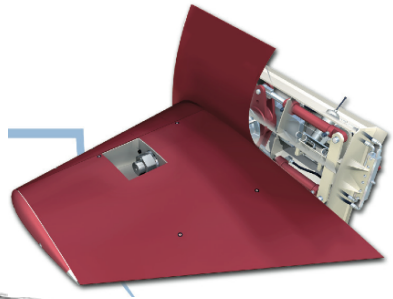
- The longer the horizon the better the stability. Stability can be proven using Lyapunov theory with the cost as a Lyapunov candidate.
- $\mathbf{Q}$ and $\mathbf{R}$ weigh the control effort and the deviations from the equilibrium.
- The final state weight $\mathbf{Q}_f$ is usually chosen as the solution of the Riccati equation associated with the infinite horizon problem (LQR). This accounts for the fact that if the horizon is long enough the state will eventually reach a set in the feasible region in which the LQR solution is the optimal solution and guarantees stability.

Example Ship Fin-roll Stabilisation

- 60% to 90% roll reduction (RMS)
- Performance depends on speed
- Control is important for performance

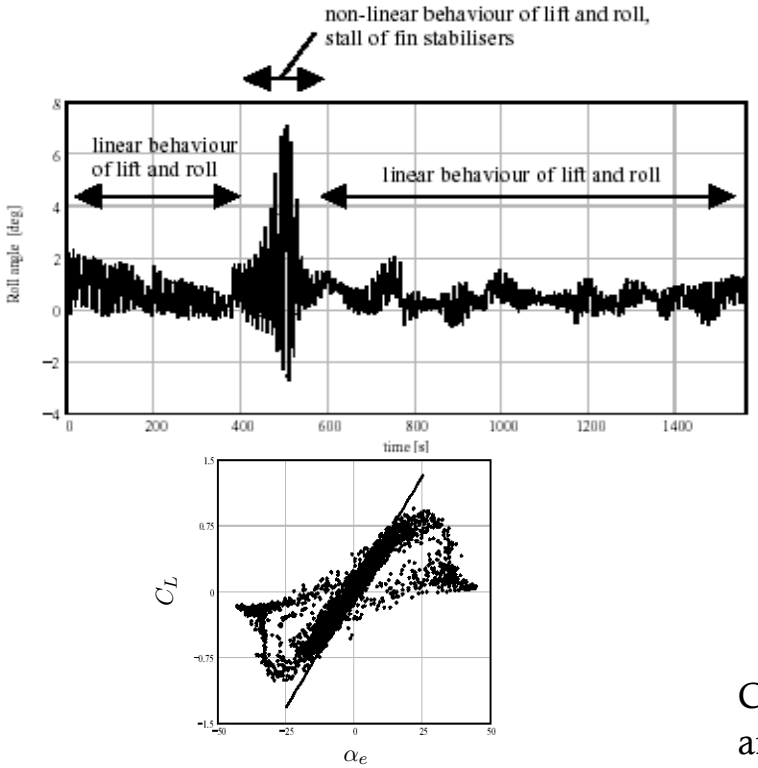
- Easy to damage
- Most expensive stabiliser
- Can produce noise affecting sonar
- Dynamic stall



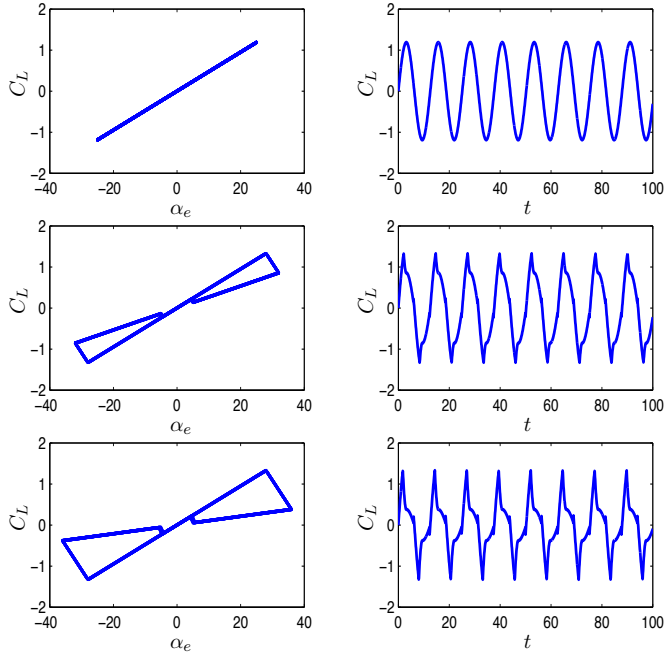


Example Ship Fin-roll Stabilisation

Strange experimental results



Dynamic Stall Model



Can we estimate and constrain the effective angle of attack?

Example Ship Fin-roll Stabilisation

1 DOF model

$$\dot{\phi} = p,$$

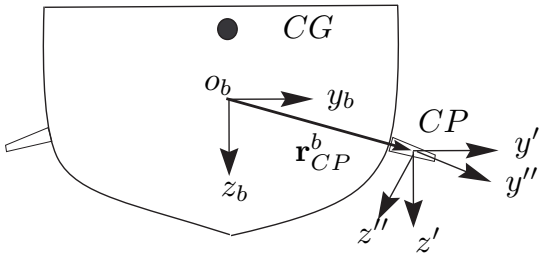
$$[I_{xx} + K_{\dot{p}}]\,\dot{p} + (K_p + 2\,r_f K_\alpha U)\,p + K_\phi\,\phi = K_w - 2U^2 K_\alpha\,\alpha$$

Discrete-time model

$$\mathbf{x}_{k+1} = \mathbf{A}\,\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{d}_k$$

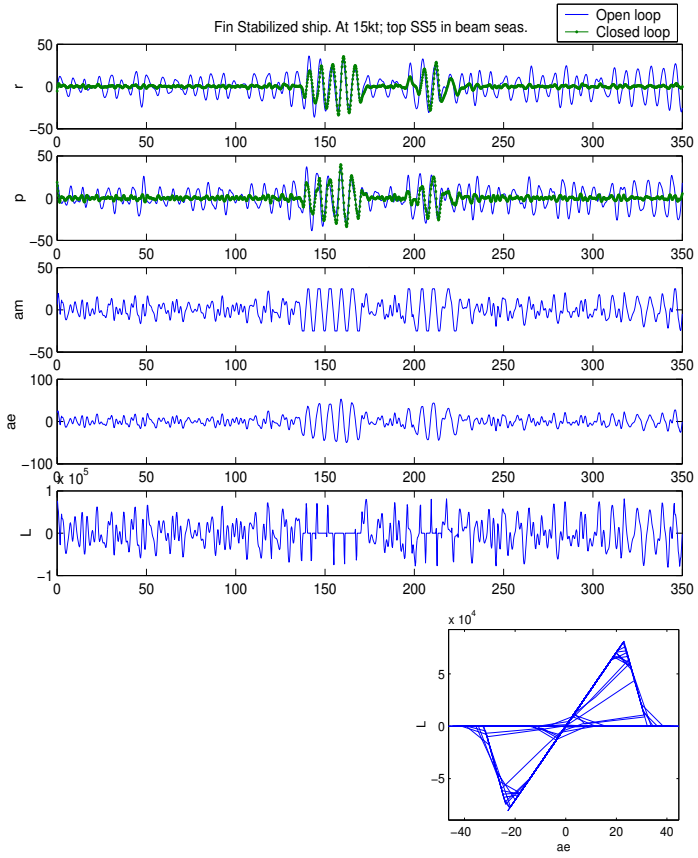
$$\mathbf{x}_k = [p_k, \phi_k]^\mathsf{T} \quad \mathbf{u}_k = \alpha$$

$$\mathbf{d}_k = K_{w,k}$$

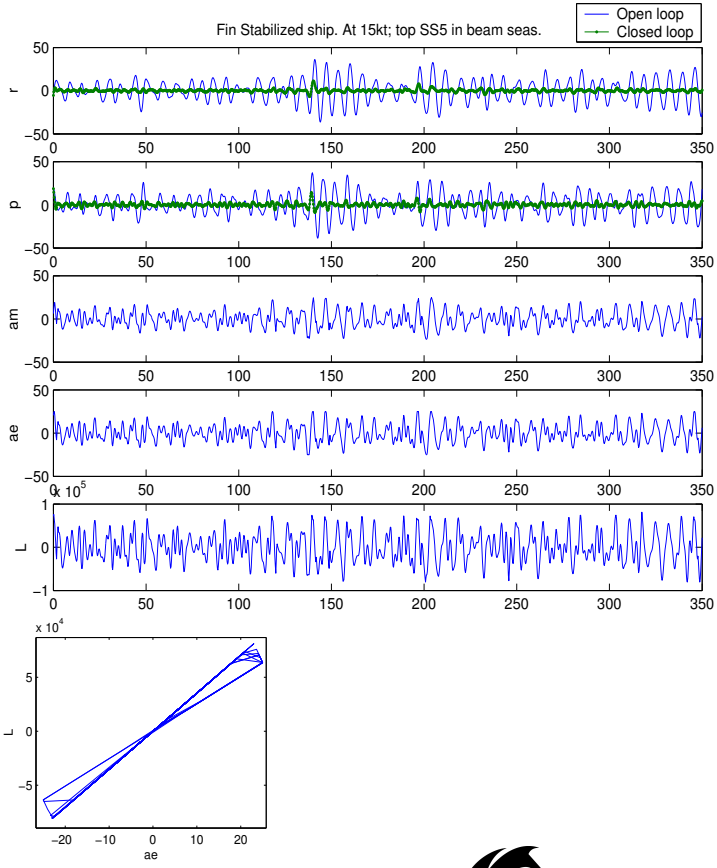


Example Ship Fin-roll Stabilisation

No output constraint



With output constraint



References

- Maciejowski, J.M. (2002). *Predictive Control with Constraints*. Prentice Hall.
- Bertsekas, D. P. (1976). *Dynamic Programming and Stochastic Control*. Vol. 125 of Mathematics in science and engineering. Academic Press, New York.
- Bertsekas, D. (2000). *Dynamic Programming and Optimal Control*. Vol. 1,2 of Optimization and Computation Series. Athena Scientific, Belmont, Massachusetts.
- Perez, T. (2005). *Ship Motion Control*. Springer London.