Overview

1 Dynamic System Models

2 From Continuous to Discrete Time

3 Input Output Models

4 Stochastic Models
Slides contain some figures from slides by Rien Quirynen and from the textbook "Model Predictive Control: Theory, Computation, and Design" (by Rawlings, Mayne, and Diehl)
Optimal Control based on Dynamic System Models

- optimal control = optimization of dynamic systems

- each optimal control problem (OCP) is characterized by three ingredients:
  - dynamic system model
  - constraints
  - objective function, i.e., cost or reward
optimal control = optimization of dynamic systems

each optimal control problem (OCP) is characterized by three ingredients:
- dynamic system model (focus of this talk)
- constraints
- objective function, i.e., cost or reward
Dynamic System Models

- system model describes evolution of system as function of
  - system state $s$ from state space $\mathcal{S} \subset \mathbb{R}^{n_s}$ (or $\subset \mathbb{Z}^{n_s}$ for discrete states)
  - control action $a$ from action space $\mathcal{A} \subset \mathbb{R}^{n_a}$ (or $\subset \mathbb{Z}^{n_a}$ for discrete actions)
  - random disturbance $\epsilon$ from some disturbance space $\mathcal{D}$

- examples:
  - stochastic discrete time system, for $k = 0, 1, 2, \ldots$
    $$s_{k+1} = f(s_k, a_k, \epsilon_k)$$
    with "evolution function" $f : \mathcal{S} \times \mathcal{A} \times \mathcal{D} \rightarrow \mathcal{S}$
  - deterministic continuous time ordinary differential equation (ODE), for $t \in [0, \infty)$
    $$\frac{ds}{dt}(t) = f_c(s(t), a(t))$$
    with "right hand side function" $f_c : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^{n_s}$

(stochastic continuous time systems need intricate notation and are therefore omitted here)
Notation for Ordinary Differential Equation (ODE) Models

- denote $\frac{ds}{dt}(t)$ by $\dot{s}(t)$
- drop time argument, abbreviate $\dot{s}(t) = f_c(s(t), a(t))$ by
  \[ \dot{s} = f_c(s, a) \]
- In this course, we use the RL notation: $s$ for state and $a$ for control action
- But in control engineering, one uses: $x$ for state and $u$ for control action, i.e.,
  \[ \dot{x} = f_c(x, u) \]
  (this notation might accidentally "slip through" on some slides)
ODE Example: Harmonic Oscillator

Mass $m$ with spring constant $k$ and friction coefficient $\beta$:

\[
\begin{align*}
\dot{x}_1(t) &= x_2(t) \\
\dot{x}_2(t) &= -\frac{k}{m}(x_2(t) - u(t)) - \frac{\beta}{m}x_1(t)
\end{align*}
\]

- state $x(t) \in \mathbb{R}^2$
- position of mass $x_1(t)$ ← measured
- velocity of mass $x_2(t)$
- control action: spring position $u(t) \in \mathbb{R}$ ← manipulated

Can summarize as $\dot{x} = f_c(x, u)$ with

\[
f_c(x, u) = \begin{bmatrix} x_2 \\ -\frac{k}{m}(x_2 - u) - \frac{\beta}{m}x_1 \end{bmatrix}
\]
ODE Example: Harmonic Oscillator

Mass \( m \) with spring constant \( k \) and friction coefficient \( \beta \):

\[
\begin{align*}
\dot{s}_1(t) &= s_2(t) \\
\dot{s}_2(t) &= -\frac{k}{m}(s_2(t) - a(t)) - \frac{\beta}{m}s_1(t)
\end{align*}
\]

- state \( s(t) \in \mathbb{R}^2 \)
- position of mass \( s_1(t) \leftarrow \text{measured} \)
- velocity of mass \( s_2(t) \)
- control action: spring position \( a(t) \in \mathbb{R} \leftarrow \text{manipulated} \)

Can summarize as \( \dot{s} = f_c(s, a) \) with

\[
f_c(s, a) = \begin{bmatrix} -\frac{k}{m}(s_2 - a) - \frac{\beta}{m}s_1 \\ s_2 \end{bmatrix}
\]
Some ODE Examples - what are their state vectors?

- Pendulum
- Hot plate with pot
- Continuously Stirred Tank Reactors (CSTR)
- Robot arms
- Moving robots
- Race cars
- Airplanes in free flight
Overview

1. Dynamic System Models

2. From Continuous to Discrete Time

3. Input Output Models

4. Stochastic Models
Transform continuous \( \dot{s}(t) = f_c(s(t), a(t)) \) into discrete time \( s_{k+1} = f(s_k, a_k) \) as follows:

1. define \( s_k := s(t_k) \) on **equidistant time grid** \( t_k = k \Delta t \) with sampling time \( \Delta t \)

2. use **zero order hold** control \( a(t) = a_k \) on \( t \in [t_k, t_{k+1}] \)

3. use **numerical simulation** to compute ODE solution \( x(t) \equiv x(t; s_k, a_k) \) satisfying

\[
\begin{align*}
  x(t_k) &= s_k \\
  \dot{x}(t) &= f_c(x(t), a_k) \quad \text{for} \quad t \in [t_k, t_{k+1}]
\end{align*}
\]

4. define \( f(s_k, a_k) := x(t_{k+1}; s_k, a_k) \)
Transform continuous \[ \dot{s}(t) = f_c(s(t), a(t)) \] into discrete time \[ s_{k+1} = f(s_k, a_k) \] as follows:

**Exact ODE solution**

\[
\begin{align*}
  x(0) &= s, \\
  \dot{x}(t) &= f_c(x(t), a), \\
  &\quad \text{for } t \in [0, \Delta t] \\
  f(s, a) &= x(\Delta t)
\end{align*}
\]

How to simulate ODE numerically?
Numerical Simulation/Integration, Three Examples

- simplest (but not recommended) implementation is a single step of an Euler integrator:

\[ f(s, a) := s + \Delta t f_c(s, a) \]

- more accurate are \( N \) steps of an Euler integrator:

\[
\begin{align*}
x_0 &:= s \\
\text{for } i = 0 \text{ to } N - 1 \text{ do} \\
x_{i+1} &:= x_i + (\Delta t/N) f_c(x_i, a) \\
f(s, a) &:= x_N
\end{align*}
\]

- more efficient are higher order **Runge Kutta (RK)** methods, e.g. a single RK4 step:

\[
\begin{align*}
v_1 &:= f_c(s, a) \\
v_2 &:= f_c(s + (\Delta t/2) v_1, a) \\
v_3 &:= f_c(s + (\Delta t/2) v_2, a) \\
v_4 &:= f_c(s + \Delta t v_3, a) \\
f(s, a) &:= s + (\Delta t/6) (v_1 + 2v_2 + 2v_3 + v_4)
\end{align*}
\]
Euler vs 4th Order Runge Kutta Method (RK4) for Test Problem

Aim: solve $\dot{s} = s + a$ for $\Delta t = 1$, $s = 1$, $a = 0$. Exact solution is $f(s, a) = e = 2.718$.

- **Four Euler steps give**
  
  \[
  \begin{align*}
  x_0 &:= 1 \\
  x_1 &:= x_0 + 1/4x_0 & [ = (1 + 1/4)x_0 ] \\
  x_2 &:= (1 + 1/4)x_1 \\
  x_3 &:= (1 + 1/4)x_2 \\
  x_4 &:= (1 + 1/4)x_3 \\
  f_{\text{Euler}}(s, a) &= x_4 & [ = (1+1/4)^4 = 2.441], \text{ error } > 10\%
  \end{align*}
  \]

- **One RK4 step gives**
  
  \[
  \begin{align*}
  v_1 &:= 1 \\
  v_2 &:= 1 + 1/2v_1 & [ = 6/4 ] \\
  v_3 &:= 1 + (1/2)v_2 & [ = 7/4 ] \\
  v_4 &:= 1 + v_3 & [ = 11/4 ] \\
  f_{\text{RK4}}(s, a) &= 1+(1/6) (v_1+2v_2+2v_3+v_4) & [ = 2.708 ]
  \end{align*}
  \]

RK4 is $27\times$ more accurate than Euler for same number $M = 4$ of function evaluations.
Classes of Numerical Simulation Methods

General Linear Methods

Multistep

Linear Multistep explicit implicit

Single-step

Runge-Kutta explicit implicit

and others ...
Fourth order RK method most efficient for typically desired accuracies

- each integration method is characterized by
  - integration order $P$ and
  - number of internal stages $S$
- can increase accuracy by more integration steps $N$
- total number of function evaluations is $M = N \cdot S$
- integration error proportional to $M^{-P}$
- for small $M$, low order methods are most accurate, e.g., Euler with $P = 1$
- for large $M$, high order methods are more accurate
- humans typically want errors smaller than 10%, but rarely smaller than $10^{-6}$
- accidentally, this favours the RK4 method ($P = 4$)
Classes of Numerical Simulation Methods

General Linear Methods

Multistep

Single-step

Linear Multistep

Runge-Kutta

explicit implicit explicit implicit and others ...
Discretization equations for general Runge Kutta (RK) methods

**Exact ODE solution**

\[
\begin{align*}
    x(0) &= s, \\
    \dot{x}(t) &= v(t) \\
    v(t) &= f_c(x(t), a), \\
    f(s, a) &= x(\Delta t)
\end{align*}
\]

**N steps of general RK method with S stages**

\[
\begin{align*}
    x_0 &= s, & x_{k+1} &= x_k + h \sum_{j=1}^{S} b_j v_{k,j} \\
    x_{k,i} &= x_k + h \sum_{j=1}^{S} a_{i,j} v_{k,j} \\
    v_{k,i} &= f_c(x_{k,i}, a), \\
    v_{k,i} &= f_c(x_{k,i}, a), \\
    f(s, a) &= x_N
\end{align*}
\]

- \(a_{i,j}\) and \(b_j\) are **Butcher tableau entries** of (potentially implicit) Runge Kutta method
- step length \(h := \Delta t/N\); intermediate states \(x_k, x_{k,i}, v_{k,i} \in \mathbb{R}^{2n}\) with integration step index \(k \in \{0, 1, \ldots, N\}\) and RK stage index \(i, j \in \{1, \ldots, S\}\)
- \(N\) nonlinear equation systems with each \(2Sn_s\) equations in \(2Sn_s\) unknowns \((x_{k,i}, v_{k,i})\)
- solved by Newton’s method (or imposed as equality constraints in optimization)
**Butcher Tableau, Six Examples**

<table>
<thead>
<tr>
<th>Implicit Euler</th>
<th>Midpoint rule (GL2)</th>
<th>Gauss-Legendre of order 4 (GL4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/2</td>
<td>(1/2 - \sqrt{3}/6)</td>
</tr>
<tr>
<td>1</td>
<td>1/2</td>
<td>(1/4)</td>
</tr>
<tr>
<td>1</td>
<td>1/2</td>
<td>(1/2)</td>
</tr>
<tr>
<td>1/2</td>
<td>1/2</td>
<td>(1/2 + \sqrt{3}/6)</td>
</tr>
<tr>
<td>1/2</td>
<td>1/2</td>
<td>(1/4 + \sqrt{3}/6)</td>
</tr>
<tr>
<td>1/2</td>
<td>1/2</td>
<td>(1/4)</td>
</tr>
<tr>
<td>1/2</td>
<td>1/2</td>
<td>(1/2)</td>
</tr>
</tbody>
</table>

**Example 8.2:**

![Graph showing function evaluations](image)

**Euler**

\[
\begin{array}{c|c|c|c|c}
0 & 0 & 1/2 & 1/2 & 1/6 \\
\hline
1 & 1 & 1 & 1/2 & 2/6 \\
\end{array}
\]

**Heun**

\[
\begin{array}{c|c|c|c|c}
0 & 0 & 1/2 & 0 & 1/6 \\
\hline
1 & 1 & 0 & 1/2 & 2/6 \\
\end{array}
\]

**RK4**

\[
\begin{array}{c|c|c|c|c}
0 & 1/2 & 0 & 1 & 2/6 \\
\hline
1 & 0 & 0 & 1 & 2/6 \\
\end{array}
\]

**Performance of different integration methods.**

- **Euler:** 3 function evaluations
- **Heun:** 2 function evaluations
- **RK4:** 3 function evaluations

**Implicit Runge-Kutta (IRK) methods**

- **Gauss-Legendre**
  - Order 4:
    \[c_1 \quad \text{a}_{11} \quad \cdots \quad \text{a}_{15}\]
    \[c_2 \quad \text{a}_{21} \quad \cdots \quad \text{a}_{25}\]
    \[
    \vdots
    \]
    \[c_s \quad \text{a}_{s1} \quad \cdots \quad \text{a}_{ss}\]
    \[b_1 \quad \cdots \quad b_s\]

An interesting fact is that an s-stage explicit Runge-Kutta method can never have coefficients on the bottom always add to one.
Intermediate Milestone: Deterministic State Space Models

From now on, throughout the course, we exclusively focus on discrete time models

\[ s_{k+1} = f(s_k, a_k) \]

with integer time index \( k = 0, 1, 2, \ldots \). We often simplify notation to

\[ s^+ = f(s, a) \]

Aim of optimal feedback control (including both MPC and RL) is to design a map, or policy, \( \pi : \mathcal{S} \rightarrow \mathcal{A}, \ s \mapsto a := \pi(s) \) such that closed-loop system \( s^+ = f(s, \pi(s)) \) has desirable properties, such as respecting constraints and minimizing a cost.

In practice, however, we might not be able to directly measure the state \( s \) ...
Overview

1. Dynamic System Models
2. From Continuous to Discrete Time
3. Input Output Models
4. Stochastic Models
In practice, we cannot measure the state. And the state representation is not even unique.

A system model should allow us to predict, for any horizon length $N$ and sequence of control actions $(a_1, \ldots, a_N)$, the sequence of measured outputs $(y_0, \ldots, y_N)$.

Typically, we need to also specify some initial conditions (e.g. the initial state $s_0$)
Two Ways to Represent Deterministic Systems with Outputs

- State Space Models with outputs:

  \[ s_{k+1} = f(s_k, a_k) \]
  \[ y_k = g(s_k, a_k) \quad \text{for} \quad k = 0, 1, 2, \ldots \]

  Initial conditions = initial state \( s_0 \).

- Input Output Models (of order \( n \)):

  \[ y_k = h(y_{k-1}, \ldots, y_{k-n}, a_k, \ldots, a_{k-n}) \quad \text{for} \quad k = n, n + 1, n + 2, \ldots \]

  Initial conditions: \( y_0, \ldots, y_{n-1} \) and \( a_0, \ldots, a_{n-1} \).
Recurrence Equation in Input Output Models of order $n$

Visualization of recurrence $y_k = h(y_{k-1}, \ldots, y_{k-n}, a_k, a_{k-1}, \ldots, a_{k-n})$:
can always transform input-output to state-space models:

state: \( s_k = (y_{k-1}, a_{k-1}, \ldots, y_{k-n}, a_{k-n}) \) (defined for \( k \geq n \))

state transition \( s \mapsto s^+ = f(s, a) \) described by

\[
\begin{bmatrix}
y_{k-1} \\
a_{k-1} \\
\vdots \\
y_{k-n+1} \\
a_{k-n+1} \\
y_{k-n} \\
a_{k-n}
\end{bmatrix}
\mapsto
\begin{bmatrix}
y_k \\
a_k \\
y_{k-1} \\
a_{k-1} \\
\vdots \\
y_{k-n+1} \\
a_{k-n+1}
\end{bmatrix}
= f(s_k, a_k) :=
\begin{bmatrix}
h(y_{k-1}, \ldots, y_{k-n}, a_k, \ldots, a_{k-n}) \\
a_k \\
y_{k-1} \\
a_{k-1} \\
\vdots \\
y_{k-n+1} \\
a_{k-n+1}
\end{bmatrix}
\]

output equation: \( y_k = g(s_k, a_k) := h(y_{k-1}, \ldots, y_{k-n}, a_k, \ldots, a_{k-n}) \).

conversely, we can sometimes transform state-space to input-output models, e.g. in case of observable and controllable linear time invariant (LTI) models.
Linear Time Invariant (LTI) Input Output Models

- Difference equation for **Auto Regressive models with eXogenous inputs (ARX):**

  \[ y_k = c_1 y_{k-1} + \ldots + c_n y_{k-n} + b_0 a_k + \ldots + b_n a_{k-n} \]

  for \( k = n, n + 1, \ldots \), with initial conditions: \( y_0, \ldots, y_{n-1} \) and \( a_0, \ldots, a_{n-1} \).

- also called **Infinite Impulse Response (IIR)** model (if some \( c_i \) coefficients are nonzero)

- If all \( c_i = 0 \) we speak of **Finite Impulse Response (FIR)** models:

  \[ y_k = b_0 a_k + \ldots + b_n a_{k-n} \]

- There exist also auto regressive (AR) models without inputs:

  \[ y_k = c_1 y_{k-1} + \ldots + c_n y_{k-n} \]

  Example: Fibonacci numbers 1,1,2,3,5,8,13,21, ... (with \( c_1 = c_2 = 1 \) and \( y_0 = y_1 = 1 \))
Some ODE Examples - what can be measured?

- Pendulum
- Hot plate with pot
- Continuously Stirred Tank Reactors (CSTR)
- Robot arms
- Moving robots
- Race cars
- Airplanes in free flight
Overview

1. Dynamic System Models
2. From Continuous to Discrete Time
3. Input Output Models
4. Stochastic Models
in reality, we always have some random noise $\epsilon_k$, e.g., disturbances or measurement errors
also, we usually have unknown, but constant system parameters $p$

(parameters can be seen as states that obey the dynamics $p_{k+1} = p_k$ and will often be omitted)
Stochastic Systems in State Space and Input Output Form

General Form (with random $\epsilon_k$):

**Stochastic State Space Model**

\[
s_{k+1} = f(s_k, a_k, \epsilon_k) \\
y_k = g(s_k, a_k, \epsilon_k)
\]

**Stochastic Input Output Model**

\[
y_k = h(y_{k-1}, \ldots, y_{k-n}, a_k, \ldots, a_{k-n}, \epsilon_k, \ldots, \epsilon_{k-n})
\]

Special Cases:

- **State Noise and Output Errors:**
  \[
s_{k+1} = f(s_k, a_k) + \epsilon_{k}^{SN} \\
y_k = g(s_k, a_k) + \epsilon_{k}^{OE}
\]

- **Equation Errors:**
  \[
y_k = h(y_{k-1}, \ldots, y_{k-n}, a_k, \ldots, a_{k-n}) + \epsilon_{k}^{EE}
\]
  (note: different than output error)
MPC needs System Identification and State Estimation

Prior to implementing an MPC controller, one needs to address two tasks:

- **System Identification (offline):**
  use a long sequence of recorded input and output data, \((a_0, \ldots, a_N)\) and \((y_0, \ldots, y_N)\), to identify parameters \(p\) using e.g. least squares optimization or subspace identification

- **State Estimation (online):**
  estimate the state \(s_k\) by using the previous control actions \((..., a_{k-2}, a_{k-1})\) and the past measurements \((..., y_{k-2}, y_{k-1})\) using e.g. Extended Kalman Filter (EKF) or moving horizon estimation (MHE) (MHE uses a fixed window of past data for fitting)

*Learning-based MPC* typically refers to an online model adaptation, i.e., to estimating parameters online (for which MHE is particularly suitable) ("learning a model" = "system identification")

Note: need state estimation only for partially observable markov decision processes (POMDP)
**Fully and Partially Observable Markov Decision Processes (MDP)**

**State Space View:**

**Partially Observable MDP**

\[ s_{k+1} = f(s_k, a_k, \epsilon_k) \]
\[ y_k = g(s_k, a_k, \epsilon_k) \]

with independent identically distributed \( \epsilon_k \)

**Fully Observable MDP**

\[ s_{k+1} = f(s_k, a_k, \epsilon_k) \]
\[ y_k = s_k \]

with \( y_k \in \mathbb{R}^{n_s} \)

**Probabilistic View:**

**Partially Observable MDP**

\[ P_{\text{state}}(s_{k+1} | s_k, a_k) \]
\[ P_{\text{meas}}(y_k | s_k, a_k) \]

with probability density functions \( P(\cdot) \)

**Fully Observable MDP**

\[ P_{\text{state}}(s_{k+1} | s_k, a_k) \]
\[ P_{\text{meas}}(y_k | s_k, a_k) = \delta(y_k - s_k) \]

with Dirac’s Delta function \( \delta(\cdot) \) in \( \mathbb{R}^{n_s} \)
Input output (I/O) models avoid need for state estimation

- We can avoid estimation task by assuming input-output (I/O) models of fixed order $n$
- This assumption leads to a **fully observable** Markov decision process (MDP)
- State $s_k$ at time $k$ is then given by $s_k = (y_{k-1}, a_{k-1}, \ldots, y_{k-n}, a_{k-n})$
- Reinforcement Learning (RL) algorithms often use I/O-models ("end-to-end learning")
- I/O-models also used in some **linear MPC** implementations based on LTI models, e.g.

$$y_k = \sum_{i=0}^{n} b_i a_{k-i} + (y_{k-1} - \sum_{i=0}^{n} b_i a_{k-i-1}) + \epsilon_k$$

- I/O-models also used for nonlinear black-box MPC or model-based RL which use neural networks for the mapping $y_k = h(y_{k-1}, \ldots, y_{k-n}, a_k, \ldots, a_{k-n})$
Summary

- We distinguish different model types
  - continuous vs discrete state and control
  - continuous vs discrete time
  - linear vs nonlinear
  - state space vs input output
  - deterministic vs stochastic
  - fully or partially observable
    (not to be confused with "observability" in systems theory)

- We transform differential equations to discrete time via numerical simulation
- We denote deterministic discrete time models and Markov Decision Processes (MDP) by

\[
\begin{align*}
  s^{+} &= f(s, a) \\
  P(s^{+} | s, a) &= P(s^{+} | s, a)
\end{align*}
\]

with state \( s \in \mathbb{R}^{n_s} \) and control action \( a \in \mathbb{R}^{n_a} \)