Technical Cognitive Systems

Michael Beetz

Intelligent Autonomous Systems
Technische Universität München

Summer Term 2012
Part XI

Control Frameworks
Outline

Robot Arms for Manipulation

Robot Kinematics

Control Theory Overview

Trajectory Following
Robot Motion Control

Controlled Process

Physics (Plant)  Sensing

Objective

Controlling Process

Decision Making

Models

Learning Element

Perception

Belief State

Motion Control

Robot Arms  Robot Kinematics: Representation  Control  Trajectory Following

Michael Beetz  Summer Term 2012
Outline

Robot Arms for Manipulation

Robot Kinematics

Control Theory Overview

Trajectory Following
Do Stuff in the Real World
A Jungle of Robotic Arms
Links and Joints

- robot arms consist of **joints** and **links**
- two type of joints: **revolute** (rotary motion) and **prismatic** (linear motion)
- classical industrial application: high stiffness in links and joints
  - advantage: position accuracy
  - disadvantage: may lead to high forces exerted by robot → danger!
  - note: link deflection under load or joint play might reduce precision in the real-world

![Revolute Joint](image1)
![Prismatic Joint](image2)
Links and Joints – example

How would you model this chain of links and joints?
Links and Joints – example

How would you model this chain of links and joints?
Joint Limits
Outline

Robot Arms for Manipulation

Robot Kinematics
  C-Space
  Task Space
  Forward and Inverse Kinematics

Control Theory Overview

Trajectory Following
**Configuration Space vs. Task Space**

Given the robot, the most fundamental question we can ask is:

“where is the robot?”

Think about it!
Configuration Space vs. Task Space

Given the robot, the most fundamental question we can ask is:

“where is the robot?”

The answer is ..

**Configuration**

The *configuration* of a robot is a complete specification of the positions of every point of the robot.
Configuration Space vs. Task Space

The following question is...

“How can we represent the configuration?”
Configuration Space vs. Task Space

“How can we represent the configuration?”

We can express all configurations of the robot using only $n$-independent joint angles.
Configuration Space vs. Task Space

Now look at this picture:

How can we get the configuration/joint trajectory to do this?
The robot interacts with objects and obstacles which are better defined by an external set of coordinates (e.g., \((x, y, z)\) coordinates in a global frame.)
Configuration Space vs. Task Space

- The robot interacts with **objects** and **obstacles** which are better defined by an external set of coordinates (e.g., \((x, y, z)\) coordinates in a **global frame**.)
- The part interacting with environments such as obstacles and objects is called the **end-effector**.
Configuration Space vs. Task Space

- The robot interacts with **objects** and **obstacles** which are better defined by an external set of coordinates (e.g., \((x, y, z)\) coordinates in a **global frame**.)
- The part interacting with environments such as obstacles and objects is called the **end-effector**.
- It is useful to attach a **reference frame** to the **end-effector** and denote the configuration of the end-effector in the global/object frame.
Configuration Space/ Joint Space

Degrees of Freedom

The number of degrees of freedom is the minimum number of independent parameters needed to represent the configuration of the robot.

It is equivalently called the mobility of the robot.
Configuration Space/ Joint Space

Degrees of Freedom

The number of degrees of freedom is the minimum number of independent parameters needed to represent the configuration of the robot.

It is equivalently called the mobility of the robot.

Configuration Space

If the robot has \( n \) degrees of freedom, the \( n \)-dimensional space containing all possible configurations of the robot is called the configuration space.

The robot’s configuration is usually expressed in terms of joint variables, so the configuration space is also called the joint space.
Task Space/ Operational Space

Task Space

Given a robot with a reference frame attached to its end-effector, the **task space** is the set of all positions and/or orientations achievable by the end-effector frame (task frame).

\[ T = \{(x, y, z) | r_1 \leq x^2 + y^2 + z^2 \leq r_2 \} \]
Task Space/ Operational Space

Task Space

Given a robot with a reference frame attached to its end-effector, the **task space** is the set of all positions and/or orientations achievable by the end-effector frame (task frame).

\[ T = \{(x, y, z) | r_1 \leq x^2 + y^2 + z^2 \leq r_2\} \]

\[ \mathbf{T} = \{(x, y, z, \alpha, \beta, \gamma) | r_1 \leq x^2 + y^2 + z^2 \leq r_2, (\alpha, \beta, \gamma) \in SO(3)\} \]
Forward Kinematics

Forward Kinematics Problem

Given an open chain robot arm with a prescribed task frame, the goal is to determine the task frame’s position and orientation as a function of the joint values.

Given: \((q_1, q_2, \ldots, q_7)\)
Forward Kinematics

Forward Kinematics Problem

Given an open chain robot arm with a prescribed task frame, the goal is to determine the task frame’s position and orientation as a function of the joint values.

For a given set of joint angles there will usually exist a unique end-effector position and orientation.
Differential Kinematics

Differential Kinematics Problem

A robot’s **differential kinematics** refers to the relationship between the robot’s joint velocities and the linear and angular velocity of its end-effector.

- Given forward kinematics: \( x = f(q), q \in \text{C-space}, x \in \text{Task space} \), the differential kinematics can be obtained simply by multiplying the joint velocity by a configuration-dependent **Jacobian** matrix:

\[
\dot{x} = J(q)\dot{q}, \quad J(q) = \frac{\partial f(q)}{\partial q} \quad \text{(linear)}
\]
Differential Kinematics

Differential Kinematics Problem

A robot’s differential kinematics refers to the relationship between the robot’s joint velocities and the linear and angular velocity of its end-effector.

- Given forward kinematics: \( \mathbf{x} = f(\mathbf{q}), \mathbf{q} \in \text{C-space}, \mathbf{x} \in \text{Task space} \), the differential kinematics can be obtained simply by multiplying the joint velocity by a configuration-dependent Jacobian matrix:

\[
\dot{\mathbf{x}} = J(\mathbf{q})\dot{\mathbf{q}}, \quad J(\mathbf{q}) = \frac{\partial f(\mathbf{q})}{\partial \mathbf{q}} \quad \text{(linear)}
\]

- The linear relationship is exploited to great effect in many applications, ranging from algorithms for inverse kinematics and trajectory generation to manipulation planning and control.
Inverse Kinematics

Inverse Kinematics Problem

Given a desired position and orientation of the end-effector frame, one seeks to determine the set of joint angles that achieves this desired end-effector configuration.

Given: \((x, y, z, \alpha, \beta, \gamma)\)
Inverse Kinematics

Inverse Kinematics Problem
Given a desired position and orientation of the end-effector frame, one seeks to determine the set of joint angles that achieves this desired end-effector configuration.

For a particular end-effector position and orientation, there may exist multiple solutions, or even none at all. (e.g., elbow-up and elbow-down configurations)
Kinematics – An Example

- forward kinematics:

\[
\begin{pmatrix}
x \\
y
\end{pmatrix} = \begin{pmatrix}
L_1 \cos q_1 + L_2 \cos(q_1 + q_2) \\
L_1 \sin q_1 + L_2 \sin(q_1 + q_2)
\end{pmatrix}
\]

Figure: Two links planar manipulator
Kinematics – An Example

- **forward kinematics**: 
  \[
  \begin{pmatrix}
  x \\
  y
  \end{pmatrix} = \begin{pmatrix}
  L_1 \cos q_1 + L_2 \cos(q_1 + q_2) \\
  L_1 \sin q_1 + L_2 \sin(q_1 + q_2)
  \end{pmatrix}
  \]

- **inverse kinematics**:
  \[
  \alpha = \cos^{-1}\left(\frac{x^2 + y^2 + L_1^2 - L_2^2}{2L_1 \sqrt{x^2 + y^2}}\right), \quad
  \beta = \cos^{-1}\left(\frac{L_1^2 + L_2^2 - x^2 - y^2}{2L_1 L_2}\right)
  \]
  \[
  q_1 = \tan^{-1}\frac{y}{x} - \alpha, \quad
  q_2 = \pi - \beta
  \]
  \[
  q_1 = \tan^{-1}\frac{y}{x} + \alpha, \quad
  q_2 = \pi + \beta
  \]

Figure: Two links planar manipulator

Michael Beetz
Summer Term 2012

Robot Arms
Robot Kinematics: Representation
Control
Trajectory Following

Technical Cognitive Systems
280
Kinematics – An Example

**Figure:** Two links planar manipulator

- **forward kinematics:**
  \[
  \begin{pmatrix}
  x \\
  y
  \end{pmatrix} = \begin{pmatrix}
  L_1 \cos q_1 + L_2 \cos(q_1 + q_2) \\
  L_1 \sin q_1 + L_2 \sin(q_1 + q_2)
  \end{pmatrix}
  \]

- **inverse kinematics:**
  \[
  \alpha = \cos^{-1}\left(\frac{x^2 + y^2 + L_1^2 - L_2^2}{2L_1 \sqrt{x^2 + y^2}}\right),
  \beta = \cos^{-1}\left(\frac{L_1^2 + L_2^2 - x^2 - y^2}{2L_1 L_2}\right)
  \]
  \[
  q_1 = \tan^{-1}\frac{y}{x} - \alpha, q_2 = \pi - \beta
  \]
  \[
  q_1 = \tan^{-1}\frac{y}{x} + \alpha, q_2 = \pi + \beta
  \]

- **differential kinematics:**
  \[
  \begin{pmatrix}
  \dot{x} \\
  \dot{y}
  \end{pmatrix} = J(q) \begin{pmatrix}
  \dot{q}_1 \\
  \dot{q}_2
  \end{pmatrix}
  \]
  \[
  J(q) = \begin{pmatrix}
  -L_1 \sin q_1 - L_2 \sin(q_1 + q_2) & -L_2 \sin(q_1 + q_2) \\
  L_1 \cos q_1 + L_2 \cos(q_1 + q_2) & L_2 \cos(q_1 + q_2)
  \end{pmatrix}
  \]
The Nullspace

- **Redundancy**: Robots whose joint degrees of freedom exceed the dimension of its task space are called *kinematically redundant*. For the redundant robot, the inverse kinematics problem has an infinite number of solutions.
The Nullspace

- For a redundant robot arm, there are extra degrees of freedom so that the null motion not changing the end-effector velocity (i.e., producing a zero velocity) can be generated by multiplying the nullspace matrix to any arbitrary joint velocity.
Outline

Robot Arms for Manipulation

Robot Kinematics

Control Theory Overview
  Control Frameworks

Trajectory Following
Example of Control System

Water temperature control system
Example of Control System

What is control?

Input  →  System  →  Output
Feed-forward Control/ Open-loop Control

Diagram:
- Desired output response
- Disturbances
- Controller
- System
- Output

Diagram shows the flow of information from desired output response to output, with disturbances affecting the controller and system.
Feed-forward Control/ Open-loop Control

Our model is not perfect!

- It’s really **hard** to measure mass, inertia, and kinematic parameters **accurate**
- There is also **friction**.
- **External disturbances** such as temperature, sudden breaking, bending, **etc.**
Feedback Control/ Closed-loop Control

Control means that we have to take the \textit{measurement} and \textit{correct} our input.

The goal is to make the \textit{error go to zero} within acceptable time.
PID control as a Feedback Control

PID (Proportional-Integral-Derivative) controller

- $K_p$ acts as a virtual spring that tries to reduce the position errors.
- the derivative gain $K_d$ acts as a virtual damper that tries to reduce the velocity error.
- The integral gain $K_i$ can be used to eliminate steady-state errors when the system is at rest.
Control System Design(1)

Very simple feedback control in velocity level:

\[ u = \dot{x} = \dot{x}_d + k_p(x_d - x) \]
\[ \dot{e} + k_p e = 0 \]
\[ \rightarrow e = e^{-k_p t} e(0) \]
Control System Design(1)

Very simple feedback control in velocity level:

\[ u = \dot{x} = \dot{x}_d + k_p (x_d - x) \]
\[ \dot{e} + k_p e = 0 \]
\[ \rightarrow e = e^{-k_p t} e(0) \]

By setting \( k_p \) positive, we can make the error go to zero.
Example of a simple second order P(I)D control:

\[ m\ddot{x} + c\dot{x} + kx = 0 \]

\[
\begin{pmatrix}
\dot{x} \\
\ddot{x}
\end{pmatrix} =
\begin{pmatrix}
0 & 1 \\
-\frac{k}{m} & -\frac{c}{m}
\end{pmatrix}
\begin{pmatrix}
x \\
\dot{x}
\end{pmatrix}
\]

\[ \rightarrow v = e^{At}v(0) \]

Depending on \( A \), it is overdamped, underdamped, and critical damped.
Control System Design(2)

Example of a simple second order P(I)D control:

**Natural response**

\[ m\ddot{x} + c\dot{x} + kx = 0 \]

\[
\begin{pmatrix}
\dot{x} \\
\ddot{x}
\end{pmatrix} =
\begin{pmatrix}
0 & 1 \\
-\frac{k}{m} & -\frac{c}{m}
\end{pmatrix}
\begin{pmatrix}
x \\
\dot{x}
\end{pmatrix}
\]

\[ \rightarrow v = e^{At} v(0) \]

Depending on \( A \), it is overdamped, underdamped, and critical damped.

**PD(PID) control design:** Determine gains to be critically damped

\[ f = kp(x_d - x) + (ki\int (x_d - x)dt) + kd(\dot{x}_d - \dot{x}) \]

In general, \( \dot{x}_d = \ddot{x}_d = 0 \), then, we can get the error dynamics (\( e = x_d - x \)):

\[ m\ddot{e} + (c + kd)\dot{e} + (k + kp)e = 0 \]
Complementary Control

Feedback control with feedforward control

For the previous example,

\[ f = m \{ \ddot{x}_d + k_p (x_d - x) + k_i \int (x_d - x) \, dt + k_d (\dot{x}_d - \dot{x}) \} + c \dot{x} + kx \]

the error dynamics is derived as:

\[ m \ddot{e} + k_p e + k_i \int e \, dt + k_d \dot{e} = 0 \]
Complementary Control

Feedback control with feedforward control

Example: gravity compensation, computed torque control (later)
Outline

Robot Arms for Manipulation

Robot Kinematics

Control Theory Overview

Trajectory Following
- Trajectory Following in Joint Space
- Trajectory Following in Task Space – using position IK
- Trajectory Following in Task Space – using velocity IK
Trajectory Following

Given a time-varying joint reference trajectory $q_d(t) = [q_1(t), \ldots, q_n(t)]^T$, the task of trajectory following is to track the specified joint positions, velocities and accelerations as close as possible.

The trajectories…

- joint velocity $\dot{q}_d(t)$ and acceleration trajectories $\ddot{q}_d(t)$ follow through simple time derivation
- specified trajectories are assumed to comply with the manipulators physical capabilities (joint limits, max velocities accelerations)
Trajectory Following in Joint Space

Typical applications

• 'teaching': recording of a specific motion for repeated execution, e.g. in factory assembly tasks

• building block for more sophisticated control approaches
Trajectory Following in Joint Space

Advantages
- decoupling of control problem into control of single joints possible
- linear control of single joints

Disadvantages
- straight-line interpolations in task space are most likely highly non-linear in joint space, and visa versa
- no direct correspondence to task space
Trajectory Following in Joint Space

Robot Dynamics model:

$$\tau = \underbrace{M(q) \ddot{q}}_{\text{inertia}} + \underbrace{C(q, \dot{q}) \dot{q}}_{\text{coriolis}} + \underbrace{g(q)}_{\text{gravity}}$$

Control law

- Computed torque control:

  $$\tau = M(q)(\ddot{q}_d + K_p(q_d - q) + K_d(\dot{q}_d - \dot{q})) + C(q, \dot{q}) \dot{q} + g(q)$$
Trajectory Following in Task Space – using position IK

Now, change of problem input:

- reference task space trajectory
  \[ x_d(t) = [p_x(t), p_y(t), p_z(t), \alpha(t), \beta(t), \gamma(t)]^T \in \mathbb{R}^m, \text{ here } m = 6 \]
Trajectory Following in Task Space – using position IK

Now, change of problem input:

- reference task space trajectory
  \[ x_d(t) = [p_x(t), p_y(t), p_z(t), \alpha(t), \beta(t), \gamma(t)]^T \in \mathbb{R}^m, \text{ here } m = 6 \]
  
  *translation* \hspace{1cm} *orientation*

Note:

- other orientation representations possible – we skip this topic
- overall goal still is to move the robot arm; but we changed the way we specify what we want
Trajectory Following in Task Space – using position IK

Now, change of problem input:

- reference task space trajectory
  \[ \mathbf{x}_d(t) = [p_x(t), p_y(t), p_z(t), \alpha(t), \beta(t), \gamma(t)]^T \in \mathbb{R}^m, \text{ here } m = 6 \]

Advantages:
- changes in desired task can be incorporated online
- task specification more intuitive

Disadvantages:
- position IK might be slow or 'misbehave' in certain configurations
Now, change of problem input:

- reference task space trajectory

\[
x_d(t) = [p_x(t), p_y(t), p_z(t), \alpha(t), \beta(t), \gamma(t)]^T \in \mathbb{R}^m, \text{ here } m = 6
\]

Control law: Same as the trajectory following in joint space (not repeated here)
Trajectory Following in Task Space – using velocity IK

- so far, inverse kinematics on position level
- one can also do inverse kinematics on velocity level
Trajectory Following in Task Space – using velocity IK

- so far, inverse kinematics on position level
- one can also do inverse kinematics on velocity level

- forward kinematics: $x(t) = f(q(t))$
Trajectory Following in Task Space – using velocity IK

- so far, inverse kinematics on position level
- one can also do inverse kinematics on velocity level

\[
\dot{x}(t) = J(q(t)) \dot{q}(t)
\]

- forward kinematics: \( x(t) = f(q(t)) \)
- time derivative of that: \( \dot{x}(t) = J(q(t)) \dot{q}(t) \)

\[\text{Jacobian}\]
Trajectory Following in Task Space – using velocity IK

The Jacobian...

- partial derivative matrix to map from one representational space to another
Trajectory Following in Task Space – using velocity IK

The Jacobian...

- partial derivative matrix to map from one representational space to another
- general mathematical concept that can be found in different fields, e.g. mapping from image to Cartesian space in augmented reality applications
Trajectory Following in Task Space – using velocity IK

The Jacobian...

- partial derivative matrix to map from one representational space to another
- general mathematical concept that can be found in different fields, e.g. mapping from image to Cartesian space in augmented reality applications

\[
\dot{x}(t) = J(q(t)) \dot{q}(t)
\]

*Jacobian*
Trajectory Following in Task Space – using velocity IK

The Jacobian...

- partial derivative matrix to map from one representational space to another
- general mathematical concept that can be found in different fields, e.g. mapping from image to Cartesian space in augmented reality applications

\[
\dot{x}(t) = J(q(t)) \dot{q}(t)
\]

\[
J(q(t)) = \begin{bmatrix}
\frac{\partial f_1(q(t))}{\partial q_1} & \cdots & \frac{\partial f_1(q(t))}{\partial q_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial f_m(q(t))}{\partial q_1} & \cdots & \frac{\partial f_m(q(t))}{\partial q_n}
\end{bmatrix} 
= \begin{bmatrix}
\frac{\partial x(q(t))}{\partial q_1} & \cdots & \frac{\partial x(q(t))}{\partial q_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial \gamma(q(t))}{\partial q_1} & \cdots & \frac{\partial \gamma(q(t))}{\partial q_n}
\end{bmatrix} 
\in \mathbb{R}^{m \times n}
\]
Trajectory Following in Task Space – using velocity IK

The Jacobian...
- partial derivative matrix to map from one representational space to another
- general mathematical concept that can be found in different fields, e.g. mapping from image to Cartesian space in augmented reality applications

$$\dot{x}(t) = J(q(t)) \dot{q}(t)$$

$$J(q(t)) = \begin{bmatrix}
\frac{∂f_1(q(t))}{∂q_1} & \ldots & \frac{∂f_1(q(t))}{∂q_n} \\
\vdots & \ddots & \vdots \\
\frac{∂f_m(q(t))}{∂q_1} & \ldots & \frac{∂f_m(q(t))}{∂q_n}
\end{bmatrix} \in \mathbb{R}^{m×n}$$

The Jacobian tells us how the individual task space coordinates change if we slightly change either of the joint values. And all that specific to our current configuration.
Trajectory Following in Task Space – using velocity IK

- Jacobian to map from task space velocities to joint velocities:
  \[ \dot{x}(t) = J(q(t))\dot{q}(t) \]
Trajectory Following in Task Space – using velocity IK

- Jacobian to map from task space velocities to joint velocities:
  \[ \dot{x}(t) = J(q(t))\dot{q}(t) \]
- its inverse maps the other way:
  \[ \dot{q}(t) = J^\dagger(q(t))\dot{x}(t) \]
Trajectory Following in Task Space – using velocity IK

- Jacobian to map from task space velocities to joint velocities:
  \[ \dot{x}(t) = J(q(t)) \dot{q}(t) \]
- Its inverse maps the other way:
  \[ \dot{q}(t) = J^\dagger(q(t)) \dot{x}(t) \]
- Pseudo-inverse needed because the Jacobian is in general not square
- \[ J^\dagger = J^T (JJ^T)^{-1} \] – note the left-out dependency on \( q(t) \)
Recall the dynamics model, and $\dot{q}_d(t) = J^\dagger(q(t))\dot{x}_d(t)$ with

$$q_d(t) = \int_{t_0}^{t} \dot{q}_d(t) \, dt,$$

$$\ddot{q}_d(t) = \frac{d}{dt} \dot{q}_d(t).$$

Computed torque control law again:

$$\tau = M(q)(\ddot{q}_d + K_p(q_d - q) + K_d(\dot{q}_d - q)) + C(q, \dot{q})\dot{q} + g(q)$$
Trajectory Following in Task Space – using velocity IK

Advantages

- possible to introduce the notion of nullspaces of a motion
- also possibility to adapt desired motion depending on task space formulation online

Disadvantages

- around and in singularities the resulting joint velocities may become very high – danger!
- gains still do not have meaning in the task space
Do Stuff in the Real World – revisited
The big picture...

Controlled Process

Physics (Plant)

Sensing

Objective

Controlling Process

Decision Making

Models

Learning Element

Belief State

Perception

Motion Control

Robot Arms

Robot Kinematics: Representation

Control

Trajectory Following

Michael Beetz
Summer Term 2012

Technical Cognitive Systems

320
Questions
Where to go from here?

Recommendations for further reading and/or experimenting...

- tutorials for arm control with the PR2 in simulation:
  http://www.ros.org/wiki/pr2_mechanism/Tutorials

- good online class on robotic basics from Standford:
  http://www.youtube.com/view_play_list?p=65CC0384A1798ADF

- reference book for the robot control:

- reference book for the robot kinematics: