

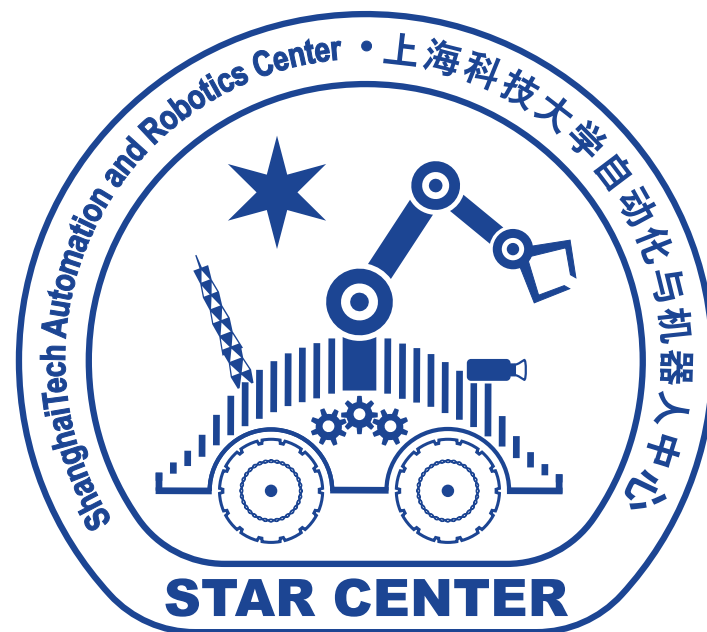


上海科技大学
ShanghaiTech University

CS289: Mobile Manipulation Fall 2025

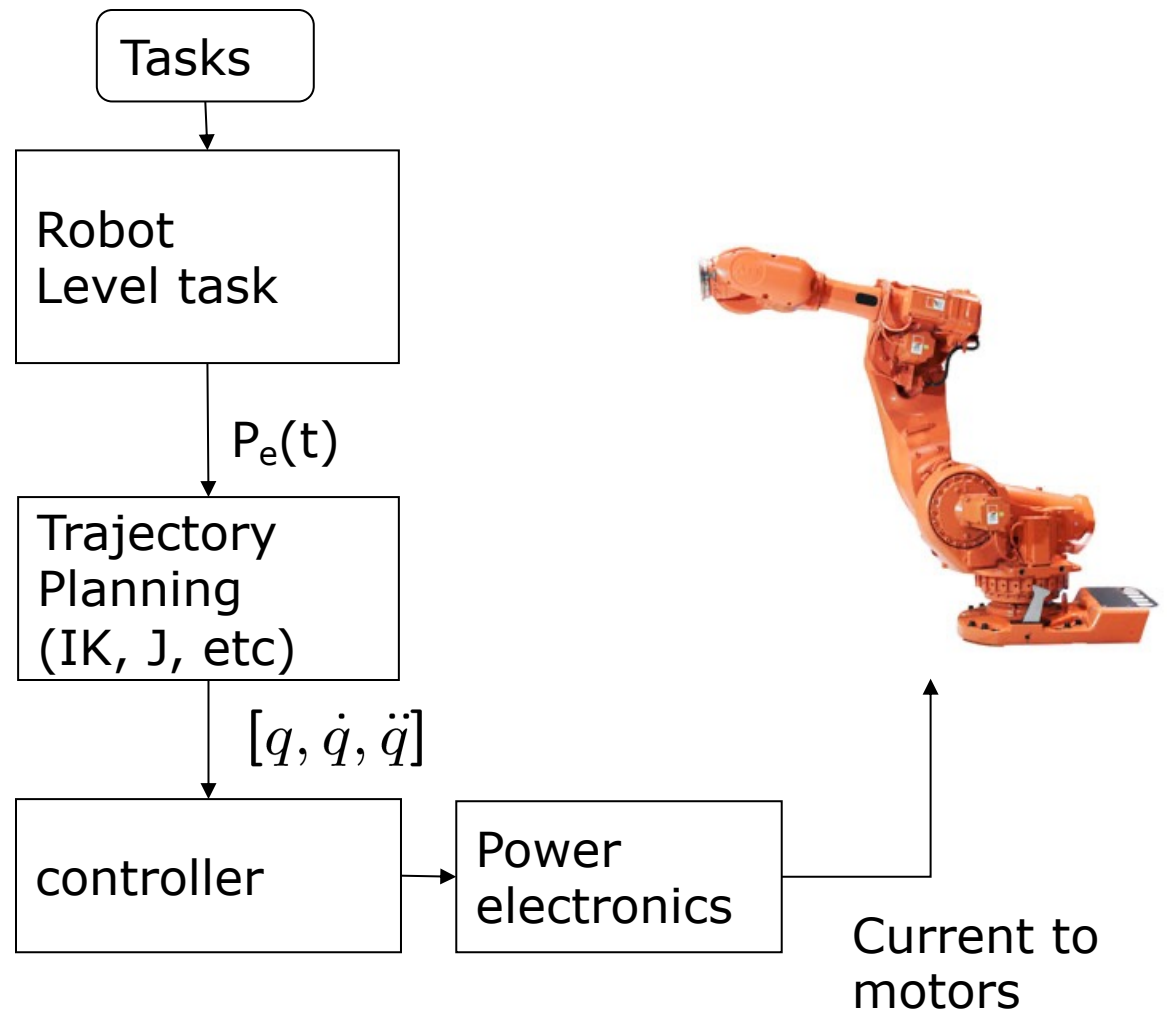
Sören Schwertfeger

ShanghaiTech University



Motivation & Overview

- We covered Kinematics, Planning, Perception, etc.
- How to make the robot actually move?
- **Control** the robot motion
 - Dynamics (forces, mass, inertia etc.) =>
 - Kinematics of speeds: Jacobian
 - Control Introduction
 - PID
- Hardware
 - PWM
 - Motor Drivers
 - Motor
 - Gears



What are kinematics?

- Describes the motion of points, bodies (objects), and systems of objects
 - Does not consider the forces that cause them (that would be kinetics)
 - Also known as “the geometry of motion”
- For manipulators
 - Describes the motion of the arm
 - Puts position/ angle and their rate of change (speed) of joints in relation with 3D pose of points on the arm, especially tool center point (tcp, end effector)

Kinematics

Forward Kinematics (angles to pose) (it is straight-forward -> easy)

What you are given: The constant arm parameters (e.g. DH parameters)
The angle of each joint

What you can find: The pose of any point (i.e. it's (x, y, z) coordinates & 3D orientation)

Inverse Kinematics (pose to angles) (more difficult)

What you are given:

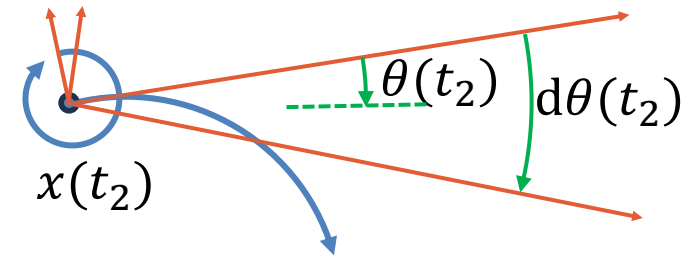
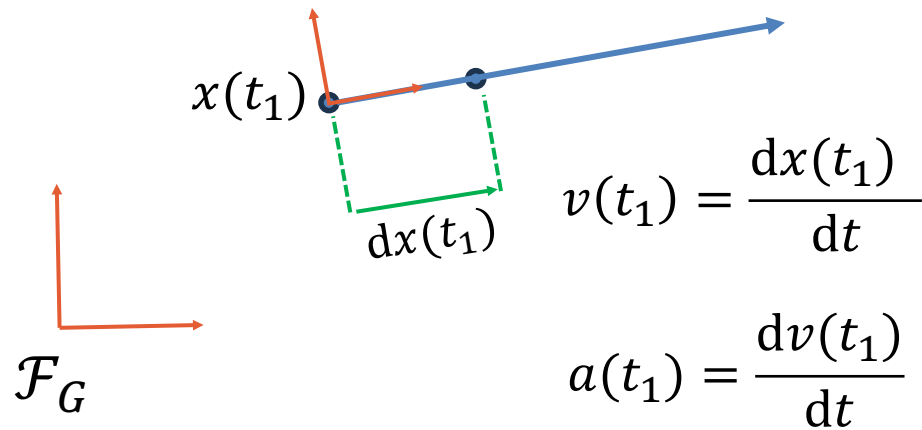
- The constant arm parameters (e.g. DH parameters)
- The pose of some point on the robot

What you can find: The angles of each joint needed to obtain that pose

What are dynamics?

- Kinematics:
Describes the motion of points, bodies (objects), and systems of objects
 - Including position, speed, acceleration, etc.
- Dynamics:
Kinematics + physics: forces, mass, inertia, moments – linear and angular
- For manipulators
 - Describes the motion of the arm
 - Puts position/ angle and their rate of change (speed) of joints in relation with 3D pose of points on the arm, especially tool center point (tcp, end effector)
 - Predict motion more accurately
 - Better control because we can predict the force (== motor power) needed

Kinematic State x



\mathcal{F}_G	Global frame
$x(t_1)$	Position (of robot) at time t_1 in \mathcal{F}_G . It is a vector of $\{x, y\}$ or $\{x, y, z\}$ in meter (two different x !)
$dx(t_1)$	Discrete example: the robot moved that much during the time dt (e.g. within $dt = 0.5$ s). This is a motion vector: $\{x, y\}$ or $\{x, y, z\}$ in meter.
$v(t_1)$	(Linear) velocity at time t_1 in \mathcal{F}_G . It is a motion vector in meters/ second
$a(t_1)$	(Linear) acceleration at time t_1 in \mathcal{F}_G . It is a motion vector in meters/ second ²
$\theta(t_2)$	Orientation at time t_2 in \mathcal{F}_G (measured against the x-axis \hat{x} of \mathcal{F}_G). In 2D: one scalar (θ) in radian. In 3D: a 3DoF rotation, e.g. Rotation Matrix or Quaternion.
$d\theta(t_2)$	Discrete example: rotation of robot during dt
$\omega(t_2)$	Angular/ rotation speed;
$\alpha(t_2)$	Angular acceleration

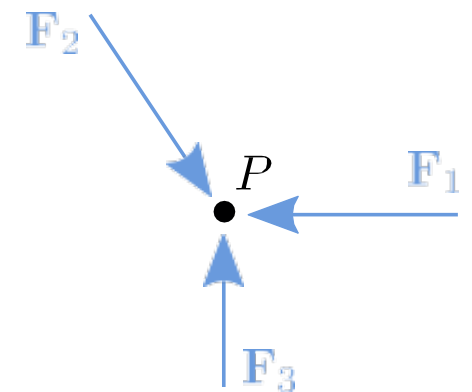
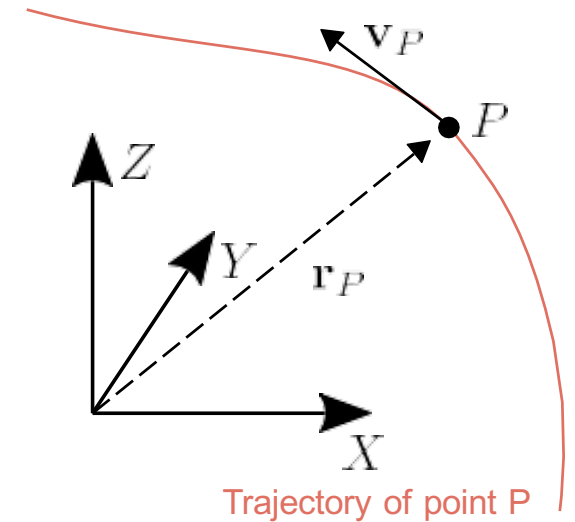
Dynamics of a Rigid Body

Translatory Motion of a Point:

- ▶ Consider **point** P with mass m in \mathbb{R}^3
- ▶ Let $\mathbf{r}_p(t) \in \mathbb{R}^3$ be its **position** in an inertial reference frame
- ▶ Let $\mathbf{v}_p(t)$ denote its **velocity** and $\mathbf{a}_p(t)$ its **acceleration**
- ▶ The **linear momentum** of P is defined as $\mathbf{p}_p(t) = m \mathbf{v}_P(t)$
- ▶ By **Newton's second law** we have

$$\frac{d}{dt} \mathbf{p}_p(t) = m \mathbf{a}_p(t) = F_{net}(t) = \sum_i \mathbf{F}_i(t)$$

where $\mathbf{F}_i(t)$ represent all forces acting on the point mass P



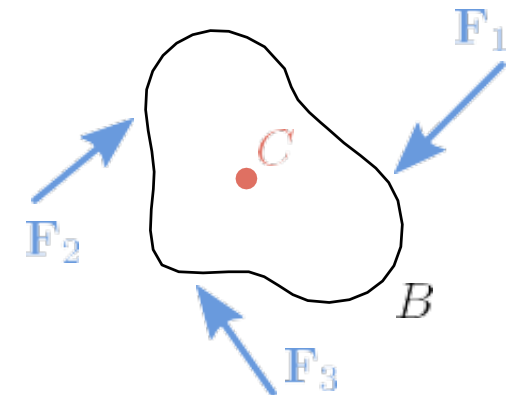
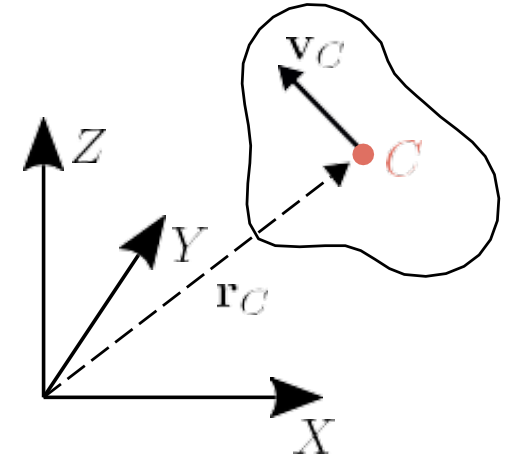
Dynamics of a Rigid Body

Translatory Motion of a Rigid Body:

- ▶ Consider a **rigid body** B with mass m in \mathbb{R}^3
- ▶ Let $\mathbf{r}_C(t) \in \mathbb{R}^3$ be the **position** of its **center of gravity** C
- ▶ Let $\mathbf{v}_C(t)$ denote its **velocity** and $\mathbf{a}_C(t)$ its **acceleration**
- ▶ The **linear momentum** of B is defined as $\mathbf{p}_B(t) = m \mathbf{v}_C(t)$
- ▶ The **center of gravity** of a rigid body **behaves like a point mass** with mass m and as if all forces act on that point

$$\frac{d}{dt} \mathbf{p}_B(t) = m \mathbf{a}_C(t) = F_{net}(t) = \sum_i \mathbf{F}_i(t)$$

where $\mathbf{F}_i(t)$ represent all forces acting on the rigid body B



Dynamics of a Rigid Body

Rotatory Motion of a Rigid Body:

- For the **rotatory motion**, also the geometric shape of B and the spatial distribution of its mass is important
- Let $\rho(x, y, z)$ be the **body's density function**:

$$m = \int_B \rho(x, y, z) dx dy dz = \int_B dm$$

- The **inertia tensor** of B is defined as

$$\Theta = \begin{bmatrix} I_x & I_{xy} & I_{xz} \\ I_{yx} & I_y & I_{yz} \\ I_{zx} & I_{zy} & I_z \end{bmatrix}$$

$$I_x = \int_B (y^2 + z^2) dm$$

$$I_y = \int_B (x^2 + z^2) dm$$

$$I_z = \int_B (x^2 + y^2) dm$$

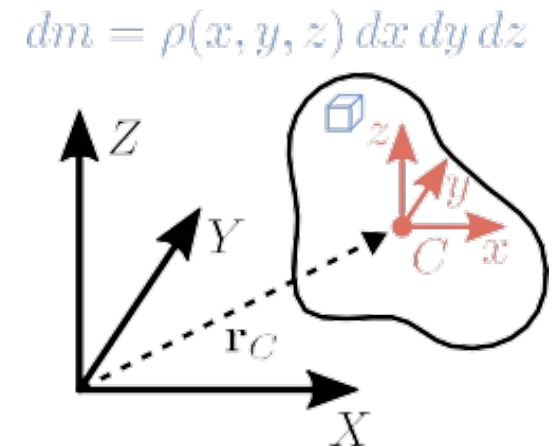
moments of inertia

$$I_{xy} = I_{yx} = - \int_B xy dm$$

$$I_{xz} = I_{zx} = - \int_B xz dm$$

$$I_{yz} = I_{zy} = - \int_B yz dm$$

moments of deviation



Dynamics of a Rigid Body

Rotatory Motion of a Rigid Body:

- Let $\boldsymbol{\omega}$ be the vector of **angular velocities**:

$$\boldsymbol{\omega} = (\omega_x \ \omega_y \ \omega_z)^T$$

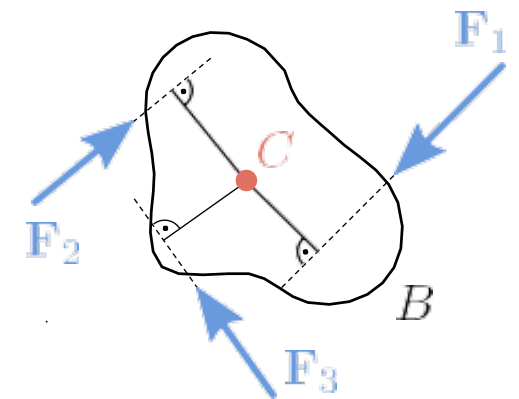
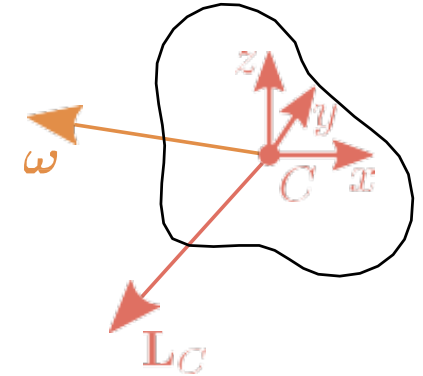
- The **angular momentum** \mathbf{L}_C of the rigid body B is given by

$$\mathbf{L}_C = \mathbf{\Theta} \boldsymbol{\omega}$$

- By the **angular momentum principle**

$$\frac{d}{dt} \mathbf{L}_C(t) = \mathbf{\Theta} \dot{\boldsymbol{\omega}} = \mathbf{M}_{net}(t) = \sum_i \mathbf{M}_i(t)$$

where $\mathbf{M}_i(t)$ are the moments of all forces acting on B with respect to the center of gravity C .



Dynamics of a Rigid Body

Rotatory Motion of a Rigid Body with Canonical Coordinates:

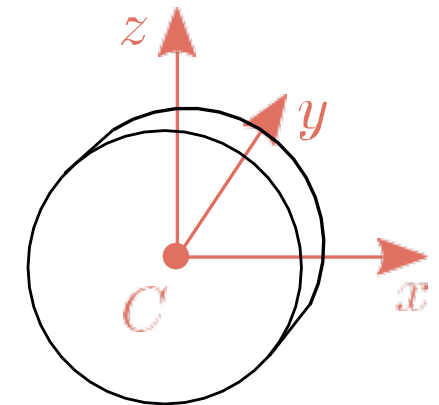
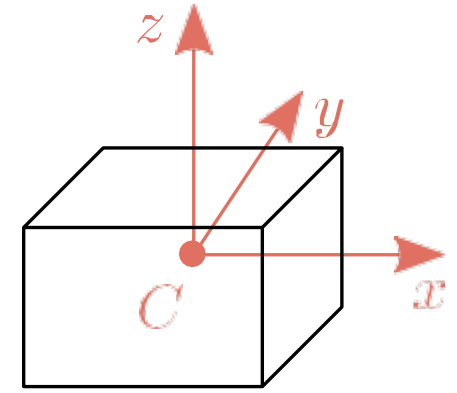
- If the body frame is chosen as a principal axis system for the rigid body (symmetry axes), the inertia tensor is diagonal:

$$\Theta = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix}$$

- For the planar motion of a rigid body in the x/y-plane:

$$\omega_x = \omega_y = 0 \quad \text{and} \quad M_x = M_y = 0$$

- Hence the angular momentum becomes $L_z = I_z \omega_z(t)$
and the angular momentum principle yields $I_z \dot{\omega}_z = \sum_i M_i$



Kinematics with speeds

- We need linear velocities and accelerations:
 $\mathbf{v}_p(t)$ velocity and $\mathbf{a}_p(t)$ its acceleration
- We need angular velocities $\boldsymbol{\omega}$ and accelerations $\dot{\boldsymbol{\omega}}$
- => use Kinematics with speeds => use Jacobians

Jacobian Matrix

- We need to know and to represent the relationship between the rates of change of the individual joint values:

$$\dot{q} = (\dot{q}_1, \dot{q}_2, \dots, \dot{q}_N)$$

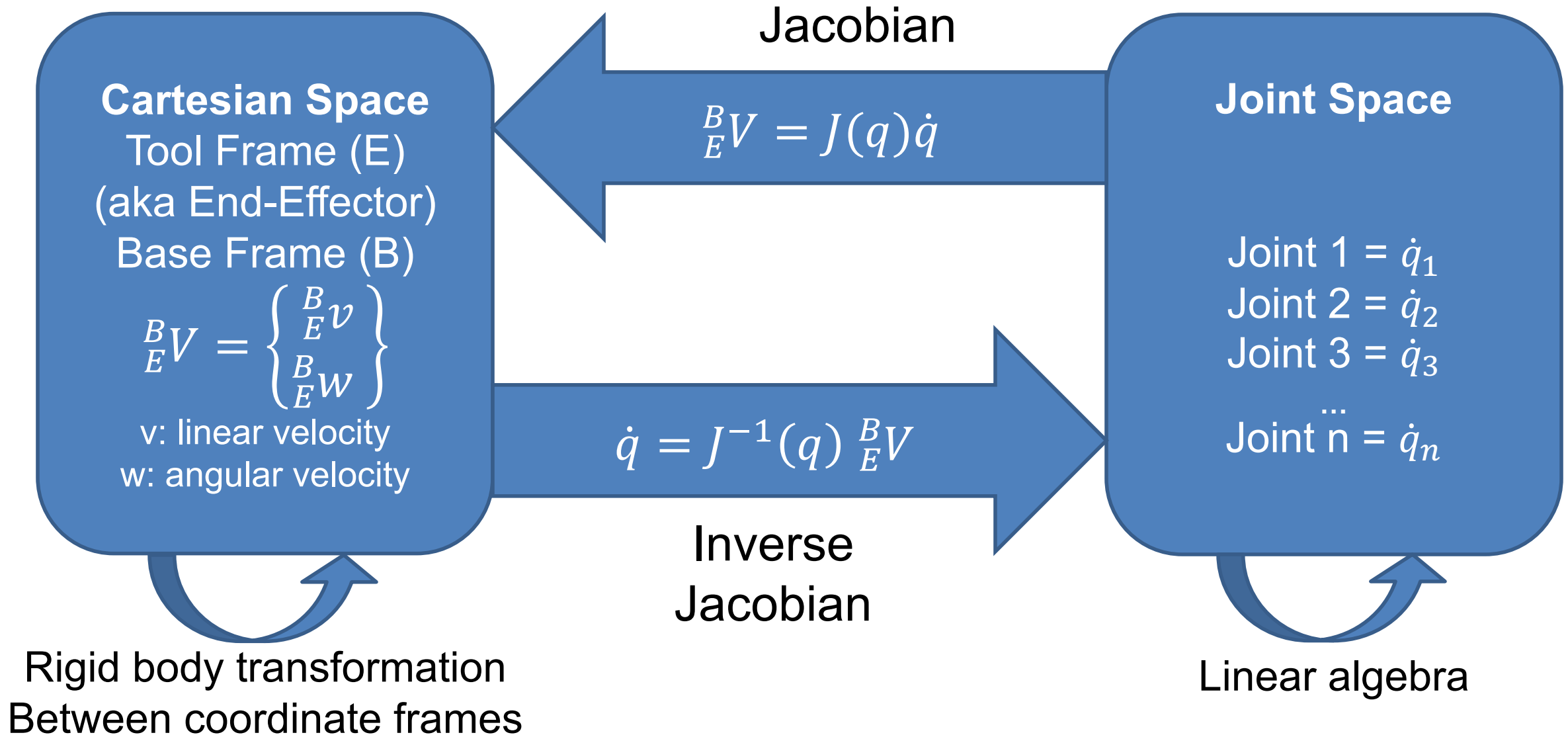
- and the rate of change of **pose** == angular ω and linear velocity v

$$\dot{X} = (\dot{\phi}, \dot{\psi}, \dot{\theta}, \dot{x}, \dot{y}, \dot{z})$$

- the matrix which represents this relationship the is called the Jacobian Matrix, J

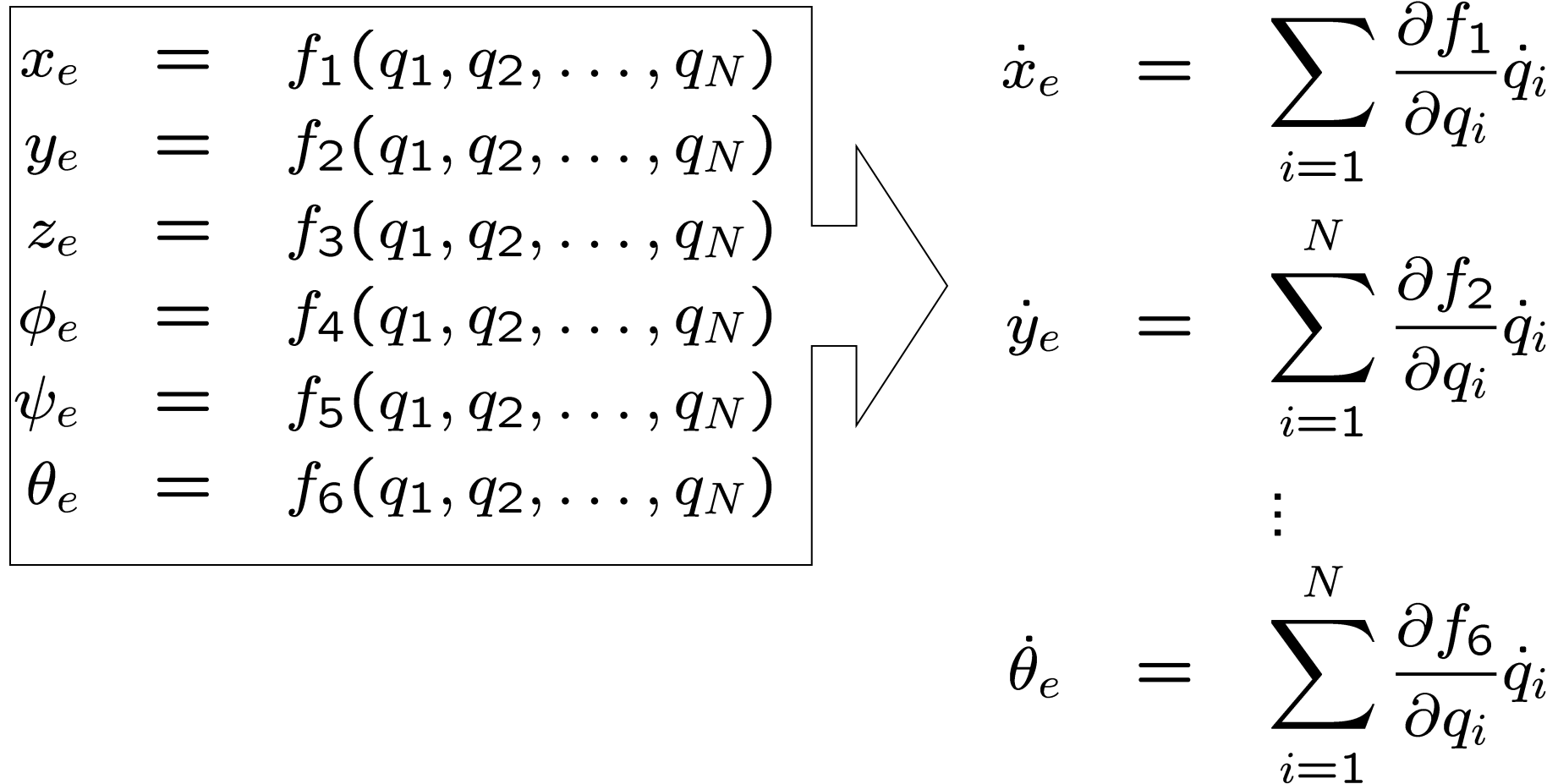
$$\dot{X} = J\dot{q}$$

Kinematics: Velocities



Jacobian Calculation

- We can obtain J by differentiating the forward kinematic relationships



The diagram illustrates the process of calculating the Jacobian matrix by differentiating forward kinematic relationships. On the left, a box contains six equations relating end-effector coordinates to joint variables q_1, q_2, \dots, q_N :

$$\begin{aligned}x_e &= f_1(q_1, q_2, \dots, q_N) \\y_e &= f_2(q_1, q_2, \dots, q_N) \\z_e &= f_3(q_1, q_2, \dots, q_N) \\\phi_e &= f_4(q_1, q_2, \dots, q_N) \\\psi_e &= f_5(q_1, q_2, \dots, q_N) \\\theta_e &= f_6(q_1, q_2, \dots, q_N)\end{aligned}$$

A large right-pointing arrow indicates the differentiation of these equations. On the right, the resulting Jacobian equations are shown:

$$\begin{aligned}\dot{x}_e &= \sum_{i=1}^N \frac{\partial f_1}{\partial q_i} \dot{q}_i \\\dot{y}_e &= \sum_{i=1}^N \frac{\partial f_2}{\partial q_i} \dot{q}_i \\\vdots \\\dot{\theta}_e &= \sum_{i=1}^N \frac{\partial f_6}{\partial q_i} \dot{q}_i\end{aligned}$$

Written in Matrix Form

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\psi} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial q_1} & \cdots & \frac{\partial f_1}{\partial q_6} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_6}{\partial q_1} & \cdots & \frac{\partial f_6}{\partial q_6} \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \\ \dot{q}_5 \\ \dot{q}_6 \end{bmatrix}$$

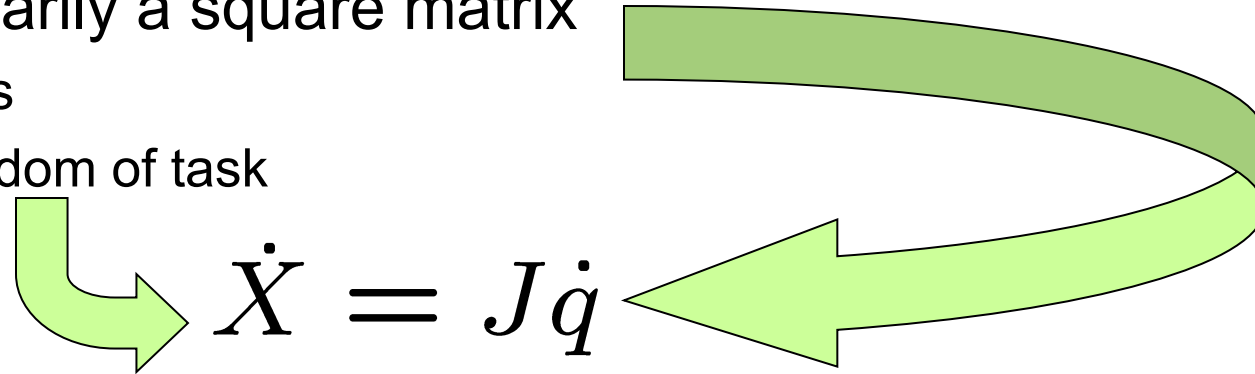
$$\dot{X} = J \dot{q}$$

Rate of change of Cartesian pose

Rate of change of joint position

Properties of J

- J is not necessarily a square matrix
 - number of joints
 - degrees of freedom of task



A diagram illustrating the relationship between joint velocities and task velocities. It features the equation $\dot{X} = J\dot{q}$ in the center. A green arrow points from the text 'degrees of freedom of task' to the \dot{X} term. Another green arrow points from the text 'number of joints' to the \dot{q} term. A large, thick green curved arrow originates from the right side of the equation and points back towards the left, indicating a feedback or dependency.

- J depends upon the instantaneous values of q_i , $i = 1, \dots, N$ so J will be *different* for each different set of joint values (q_1, q_2, \dots, q_N) , i.e. for each different robot arm configuration.

$$\dot{X} = J(q_1, q_2, \dots, q_N)\dot{q}$$

Robot Jacobian

- To obtain the inverse Jacobian relation we need to invert \mathbf{J} , which is, in general, hard. Three methods:
 1. Invert \mathbf{J} symbolically, which is only really practical for very simple robot geometries.
 2. Numerically invert \mathbf{J} for each configuration of the robot. This is computationally expensive, not always possible (e.g. when $\det(\mathbf{J})=0$) and difficult if $n \neq 6$
 - Use pseudo-inverse $(\mathbf{J} \mathbf{J}^T)^{-1} \mathbf{J}^T$
 3. Derive \mathbf{J}^{-1} directly from the Inverse Kinematics equations, much as we uses the Forward Kinematics equations to obtain \mathbf{J} above.

Singularities

- Robot is in a *singular configuration* when $\det(J)=0$ $\dot{X} = J\dot{q}$
 - i.e. when the relationship can't be inverted
- Singular configuration occur when two or more joint axes become aligned in space.
 - When this happens the robot geometry effectively loses one (or more) independent degrees of freedom: two more more of the degrees of freedom become mutually dependent.

Singularities

- The loss of one or more effective degrees of freedom thus occurs not just at a singular configuration, but also in a region (a volume in joint space) around it.
 - Not just when $\det(J) = 0$ but nearby (J is ill-conditioned).
 - Condition number is a useful (scale-independent) measure for matrix condition

Types of Singularities

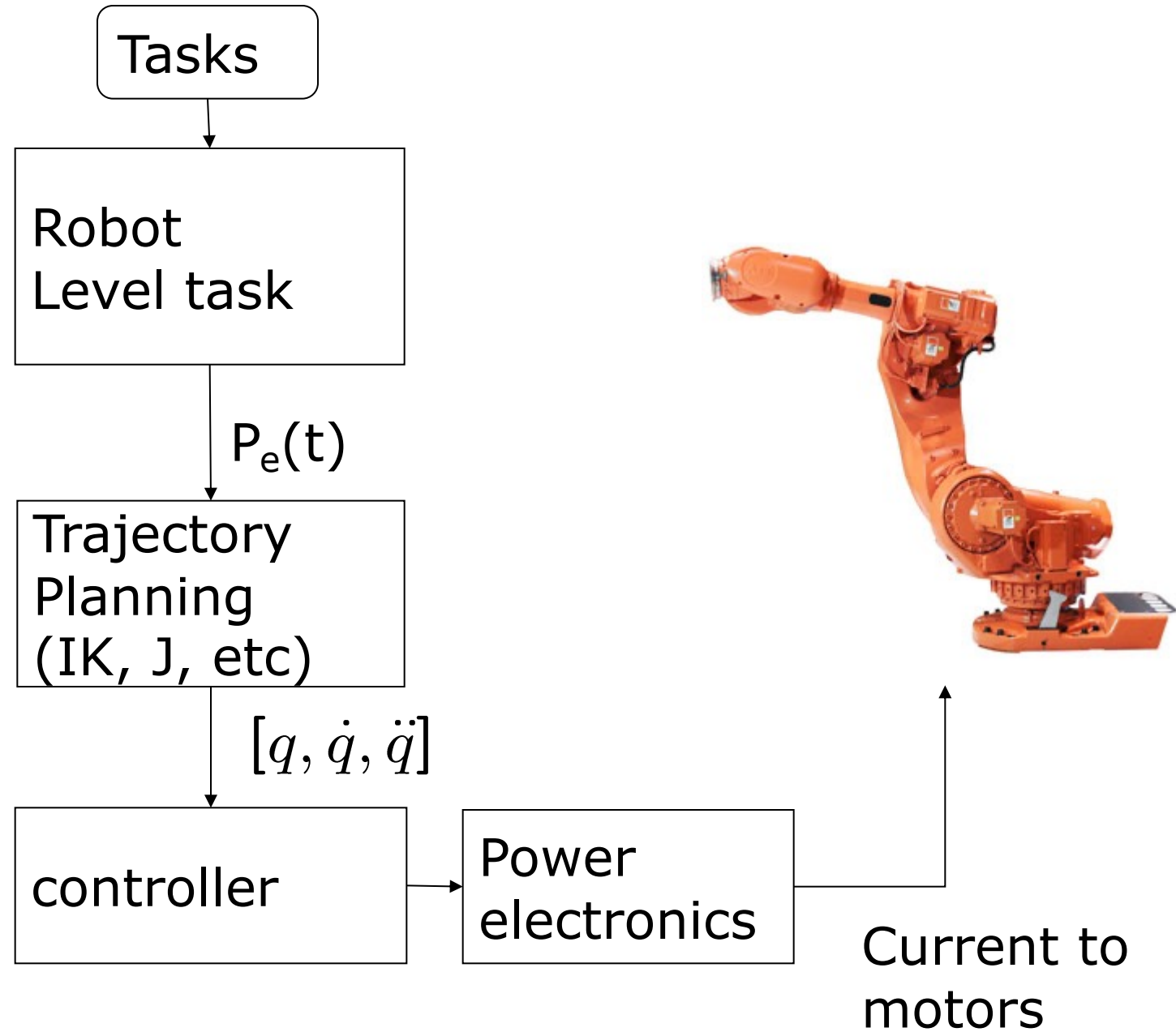
- *Workspace Boundary Singularities*: the robot manipulator is fully extended or folded onto itself so that P_e is at or near the boundary of the robot workspace.
 - In such configurations, one or more joints will be at their limits of range of movement, so that they will not be able to maintain any movement at some particular speed. This effectively makes \mathbf{J} a singular matrix.
- *Workspace Interior Singularities*: occur inside the robot workspace away from any of the boundaries, and they are generally caused by two or more joint axes becoming aligned.

Avoid Singularities

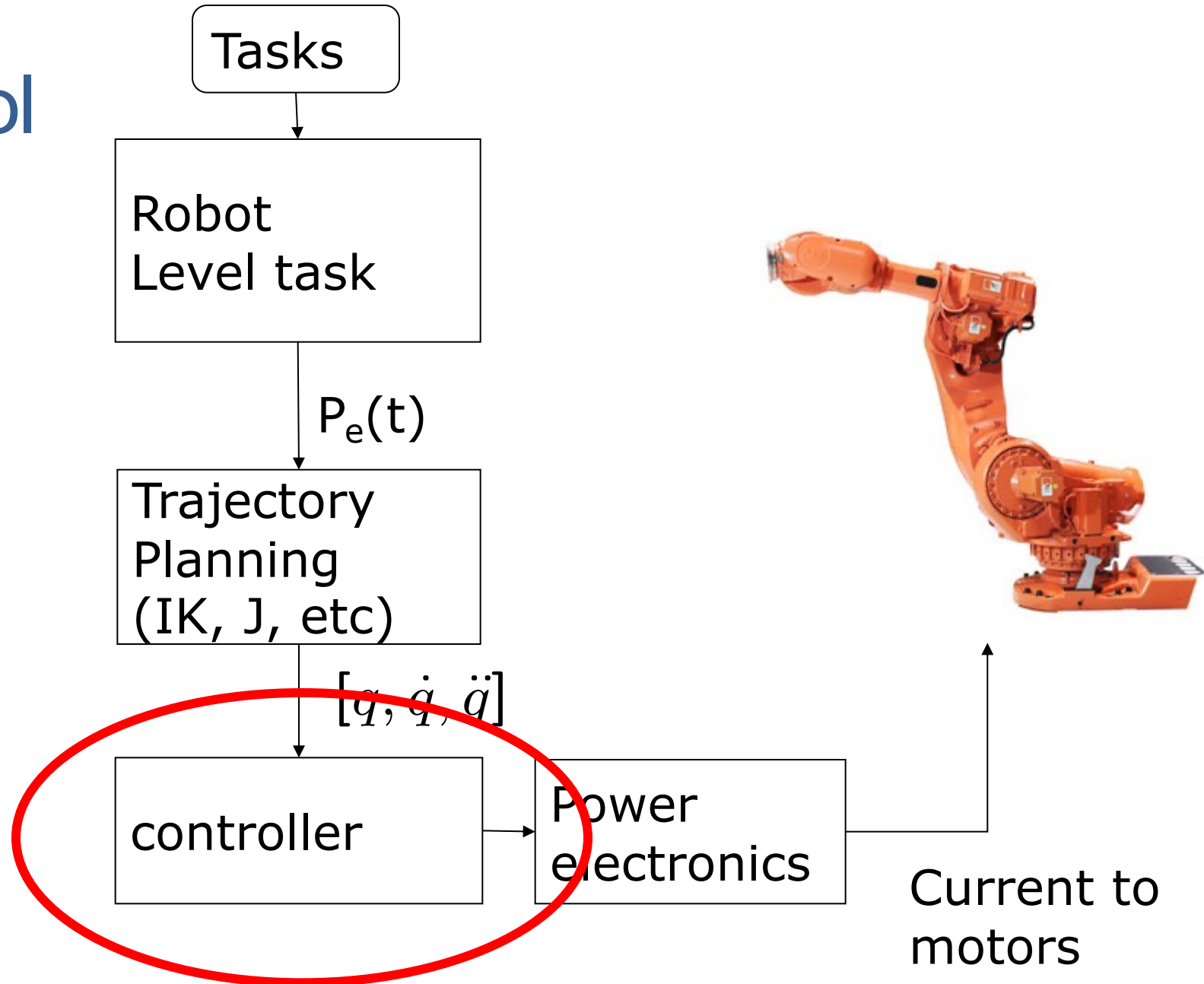
- To avoid singular configurations there are three different possibilities:
 1. Increase the number of degrees of freedom of the robot manipulator, perhaps by attaching to P_e a tool or gripper which has one or two degrees of freedom itself.
 2. Restrict the movements that the robot can be programmed to make so as to avoid getting to or near to any singular configurations.
 3. Dynamically modify \mathbf{J} to remove the offending terms, and thus return $\det(\mathbf{J}) \neq 0$. This means identifying the column and row of \mathbf{J} that need to be removed.

CONTROL

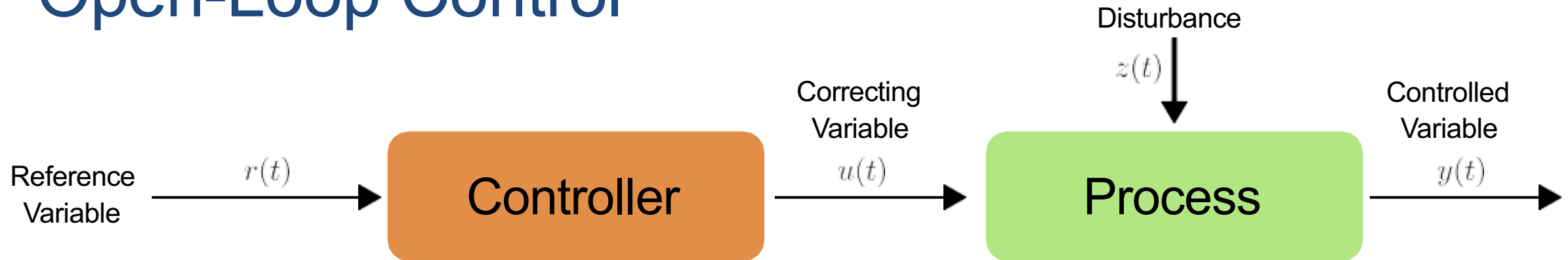
Control



Control

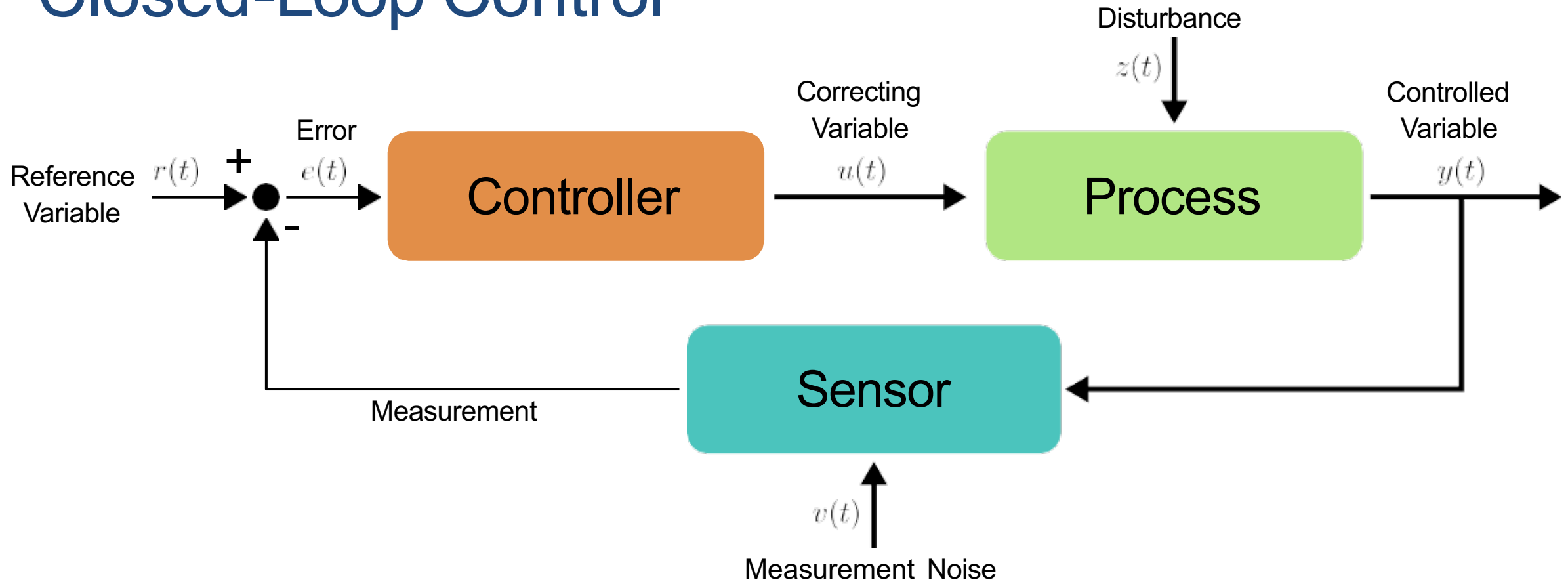


Open-Loop Control



- ▶ Requires **precise knowledge** of the plant and the influence factors
- ▶ **No feedback** about the controlled variable
- ▶ Cannot handle unknown disturbances, resulting in **drift**

Closed-Loop Control



- Exploit feedback loop to minimize error between reference and measurement

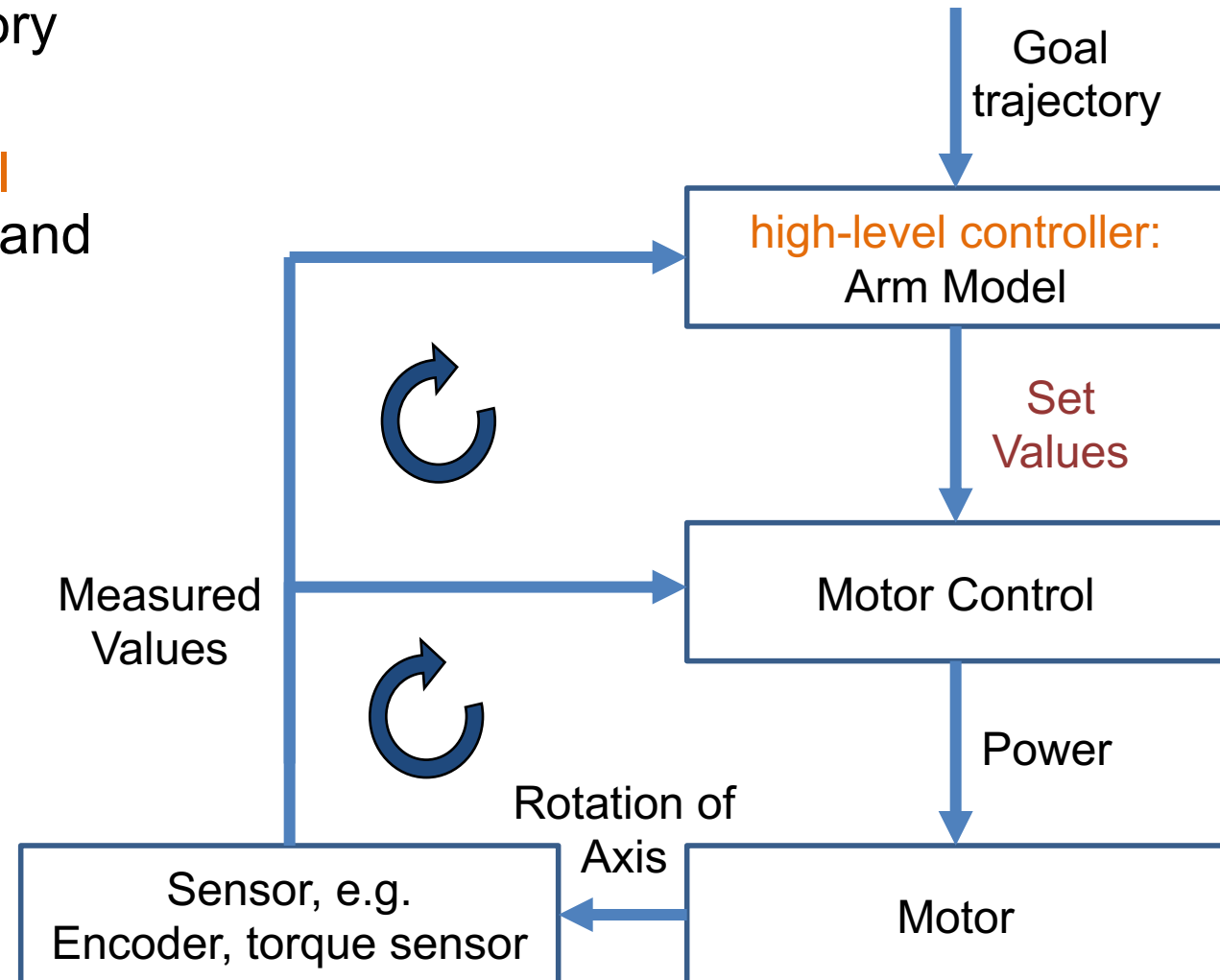
Centrifugal Governor



<https://www.youtube.com/watch?v=B01LgS8S5C8>

Control Hierarchy

- Assume we have a goal trajectory
- What values does the **high-level controller** set, using arm model and goal trajectory?
 - Position
 - Speed, Acceleration
 - Torque
 - Force
- Different control loops with different speeds
 - e.g. high-level controller 50Hz;
 - e.g. Motor controller 1000Hz;



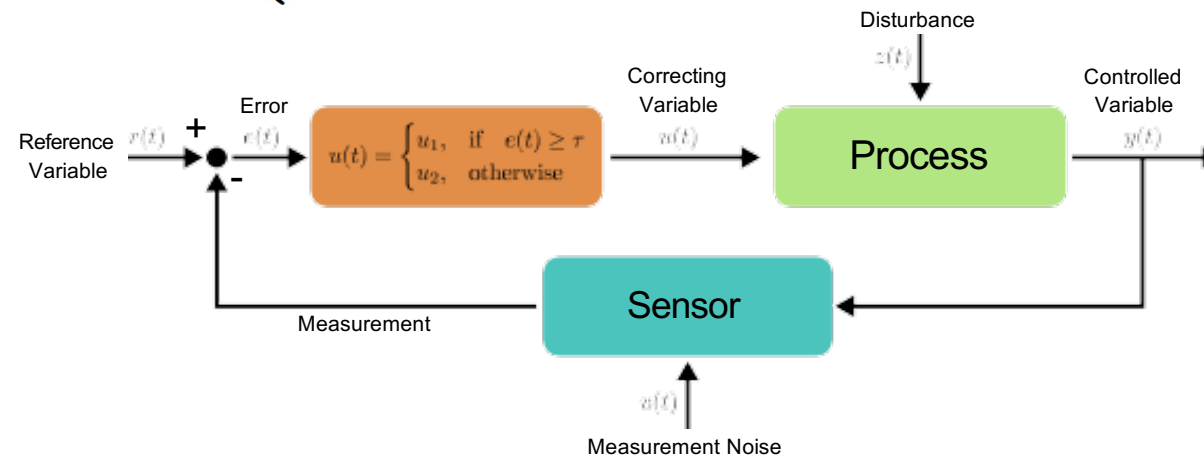
BLACK BOX CONTROL

Bang-Bang Control

Bang-Bang Control

- ▶ Also called: hysteresis controller
- ▶ Often applied, e.g. in household thermostats
- ▶ Switches abruptly between two states
- ▶ Mathematical formulation:

$$u(t) = \begin{cases} u_1, & \text{if } e(t) \geq \tau \\ u_2, & \text{otherwise} \end{cases}$$



PID Control

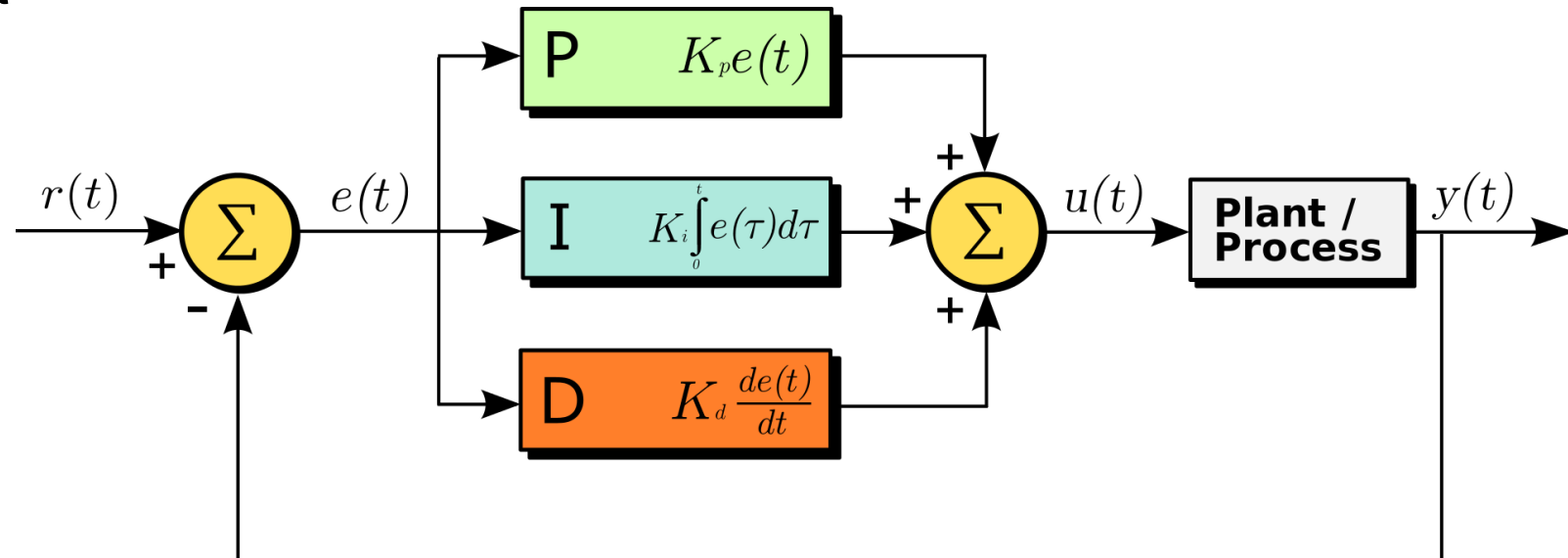
PID: Proportional-Integral-Derivative Controller

- Input: Desired Speed (of wheel/ motor)
 - Actually: Error of the current speed (process variable) to the desired speed (setpoint)
- Output: Amount of power to the motor
- Not needed: Model of the plant process (e.g. motor, robot & terrain parameters)
- Parameters:
 - K_p proportional gain constant
 - K_i integral gain
 - K_d derivative gain

- Discrete Version:

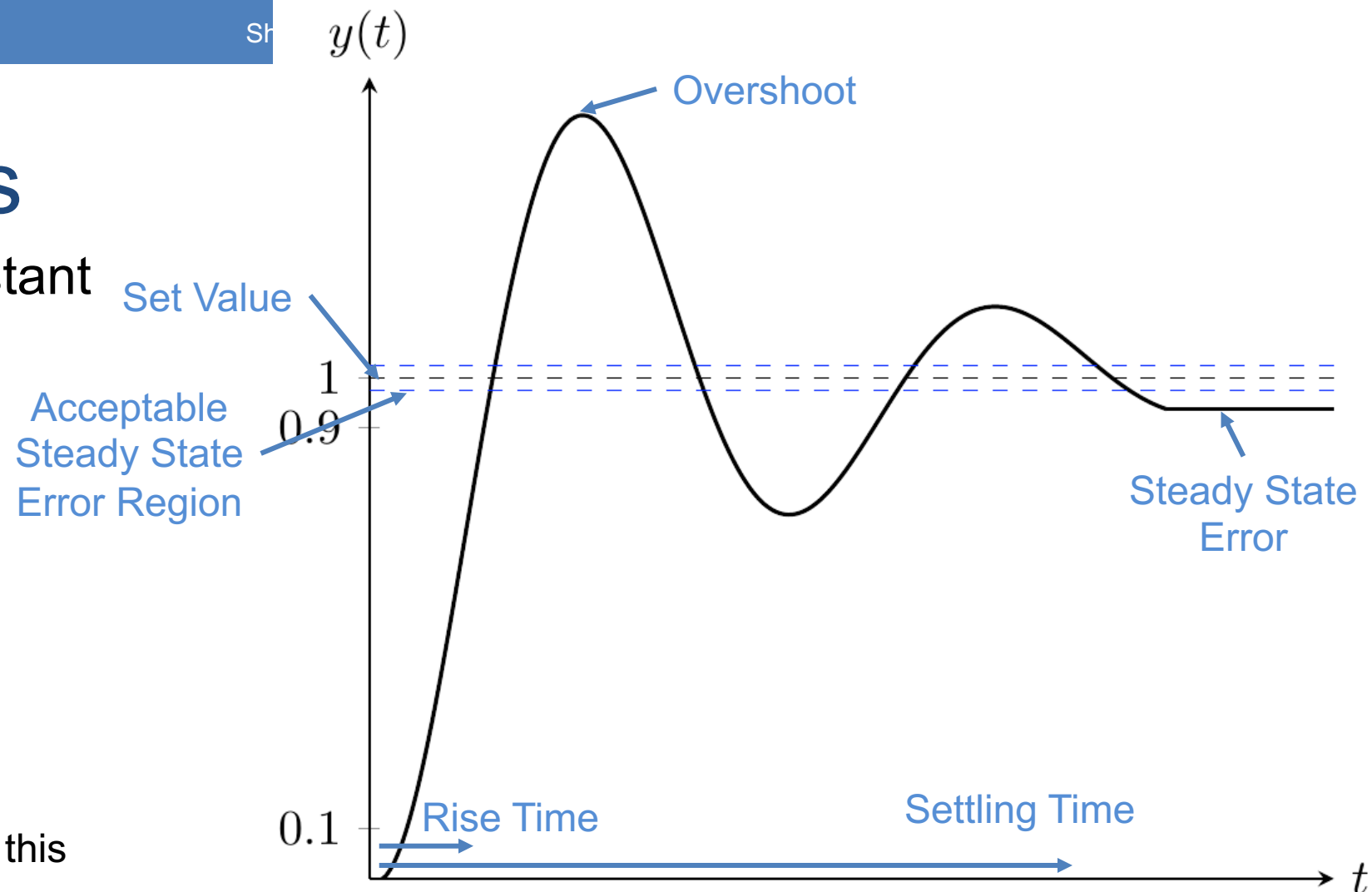
$$\int_0^{t_k} e(\tau) d\tau = \sum_{i=1}^k e(t_i) \Delta t$$

$$\frac{de(t_k)}{dt} = \frac{e(t_k) - e(t_{k-1})}{\Delta t}$$



Tune Parameters

- K_p proportional gain constant
 - Too small: long rise time
 - Too big: big overshoot or even unstable control
 - Should contribute most of the output change
- K_i integral gain
 - Reduces steady state error
 - May cause overshoot
 - Leaky integration may solve this
- K_d derivative gain
 - Predicts error by taking slope into account
 - May reduce settling time and overshoot



Parameter Increase	Rise time	Overshoot	Settling Time	Steady-state error
K_p	↓	↑	Small Change	↓
K_i	↓	↑	↑	Great reduce
K_d	Small Change	↓	↓	Small Change

Table (2) PID controller parameter characteristics on a fan's response

Control Theory

- Other controllers used

- P Controller
- PD Controller
- PI Controller

```
1 previous_error := 0
2 integral := 0
3
4 loop:
5     error := setpoint - measured_value
6     integral := integral + error × dt
7     derivative := (error - previous_error) / dt
8     output := Kp × error + Ki × integral + Kd × derivative
9     previous_error := error
10    wait(dt)
11    goto loop
```

Pseudo Code PID Controller

- PID sufficient for most control problems
- PID works well if:
 - Dynamics of system is small
 - System is linear (or close to)
- Lots of Control Theory courses at ShanghaiTech University...
- Popular alternative: Model Predictive Control (MPC)
 - Optimal Control Technique: satisfy a set of constraints
 - Finite time horizon to look into the future (“plan”)
 - Used when PID is not sufficient; e.g.:
 - Very dynamic system
 - Second order system (oscillating system)
 - Multi-variable control
 - Use Cases: Chemical plants; planes; robot arms; legged robots; ...

Controlling Self-Driving Cars



Aerospace Controls Laboratory
Massachusetts Institute of Technology



Closed-Loop Arm Control

- Independent Joint Control

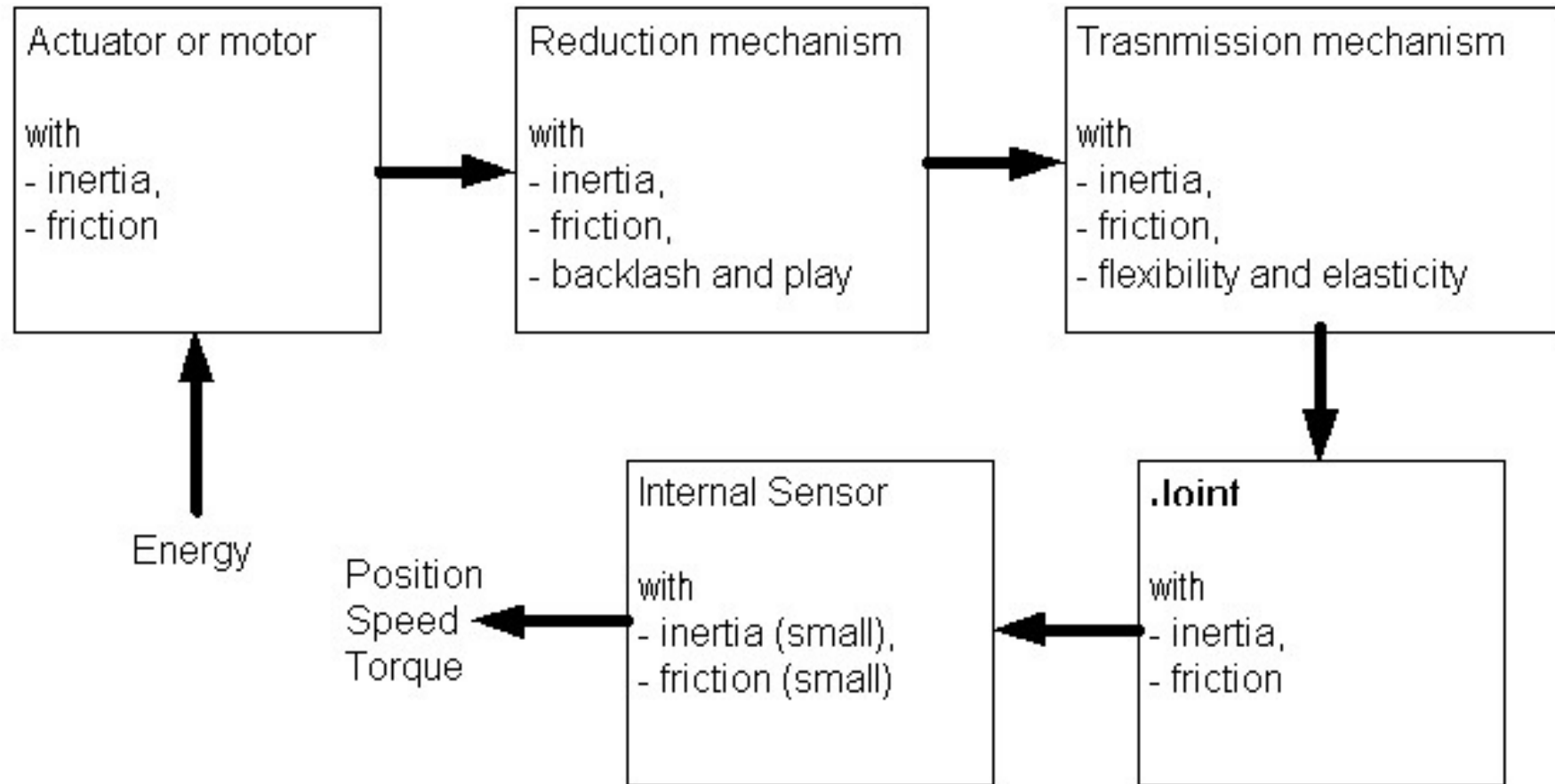
- Use computed reference points (setpoints) for each joint
- Control each joint “independently”
 - Ignore dynamic effects
 - Treat each joint as a stand alone “motor”

- Dynamics Based control

- Use dynamics model to facilitate control
 - Compute torque feedforward
 - Inverse Dynamic Control
 - Operation Space control
 - and Compliance, Impedance, Force....

Control:
Position
Speed, Acceleration
Torque
Force

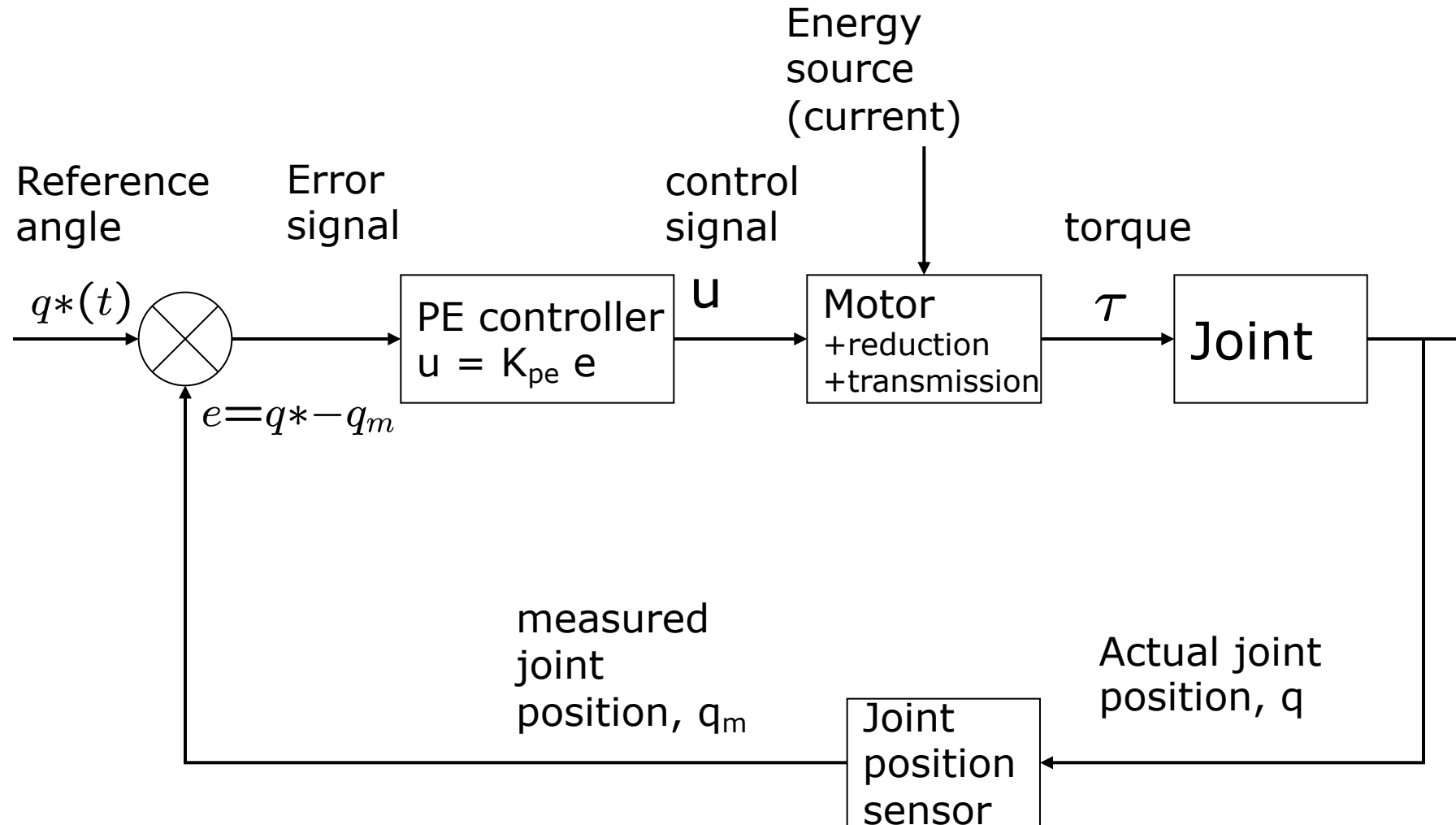
Jointed system components



Independent Joint Control

- Use computed reference points (setpoints) for each joint
- Control each joint “independently”
 - Ignore dynamic effects
 - Treat each joint as a stand alone “motor”
- Simplifies control
- Block Diagram (next slide)

Block Diagram of PE controller for a single joint



Independent Joint Control

- Control each joint independently without “communication” between actuators
- Basic Steps:
 - Model actuator
 - Use kinematics to obtain set-points for each joint
 - Develop a controller for each joint
 - Error for joint i :

$$e_i = (q_i^* - q_m)$$

$$\begin{aligned} q_i^* &= \text{desired joint position} \\ q_m &= \text{measured joint position} \end{aligned}$$

Actuator Model

- Need to model relationships:
 - between actuator input (current) and output (torque)
 - Torque is approximately linear with applied current

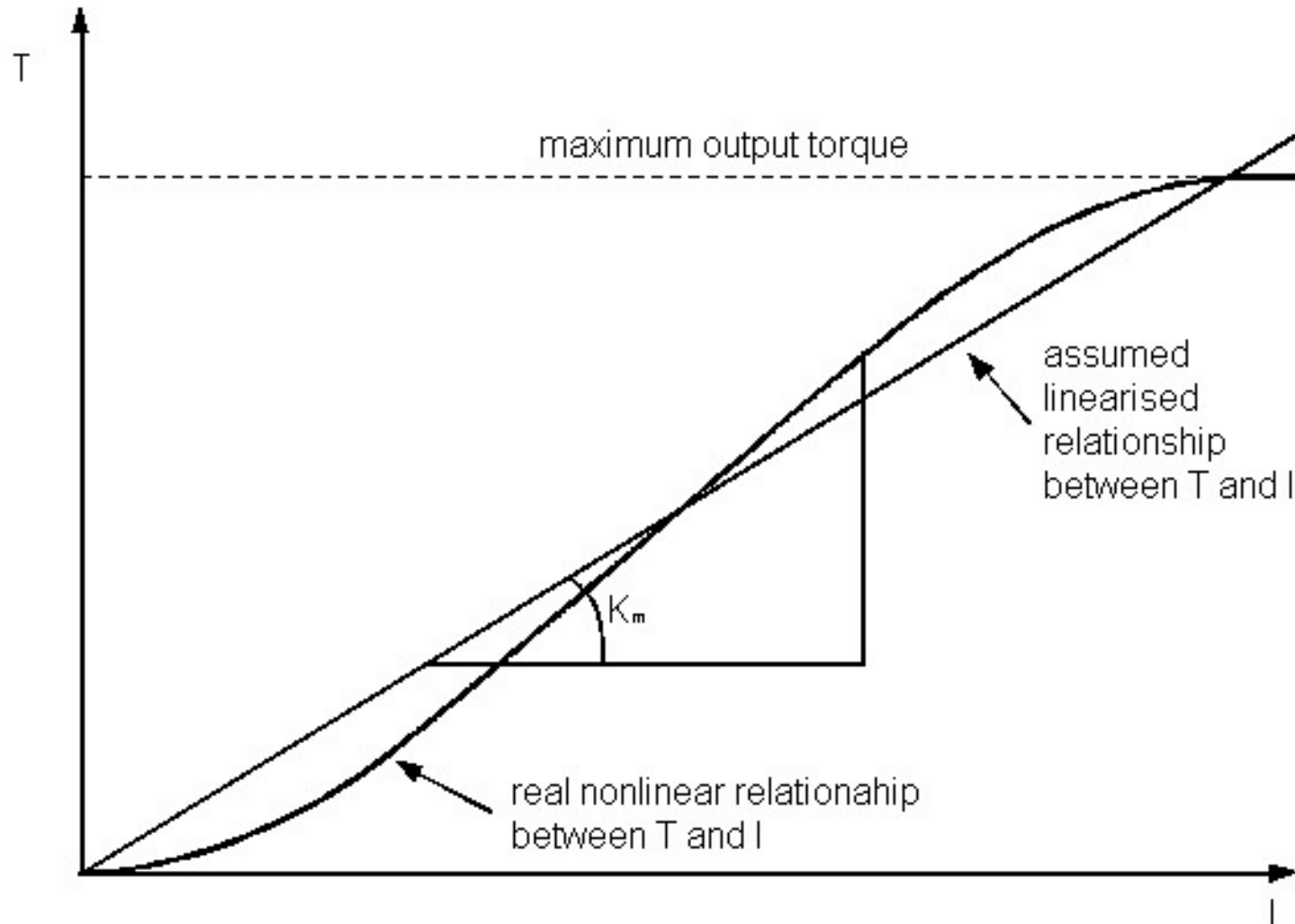
$$\tau_m = K_m i_a$$

Motor torque, Nm

Applied current, amp

Motor constant, Nm/amp

actuator current vs torque



Actuator Model

- Need to model relationships:
 - between actuator torque and motor angle (q)
 - Second order ode

$$J\ddot{q}(t) + B\dot{q}(t) = u(t) - d$$

Rotational inertia of
joint, kg m^2

disturbance

control input

Effective damping (friction,
back emf), Nm/amp

Independent Joint Control

- Control each joint independently without “communication” between actuators
- Basic Steps:
 - ✓ Model actuator
 - ✓ Use kinematics to obtain setpoints for each joint (IK)
 - **Develop a controller for each joint**
 - Error for joint i:

$$e_i = (q_i^* - q_m)$$

$$q_i^* = \text{desired joint position}$$

$$q_m = \text{measured joint position}$$