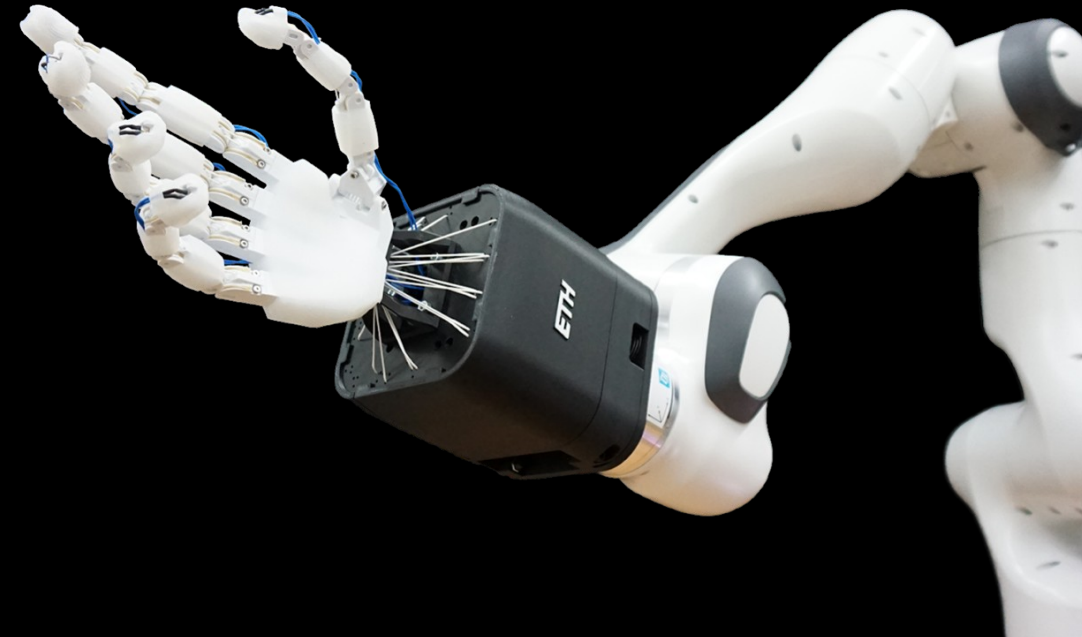




Kinematics, Dynamics and Control of Robotic Hands

Robert Katzschmann

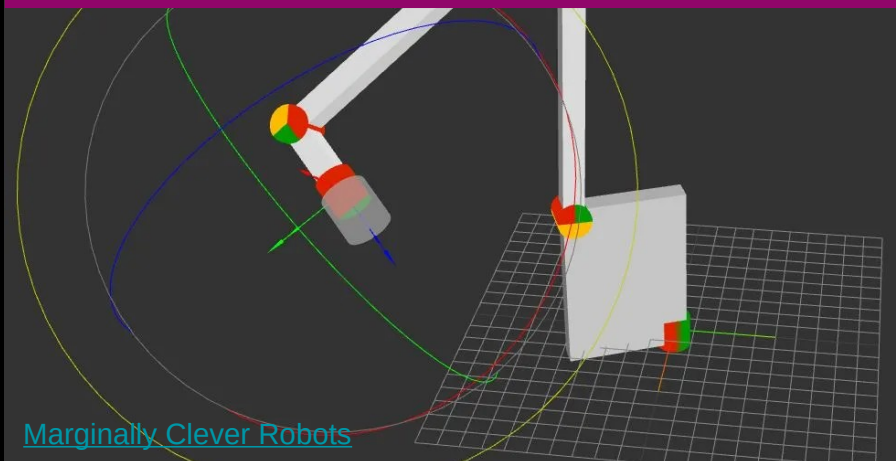
Assistant Professor of Robotics, Soft Robotics Lab



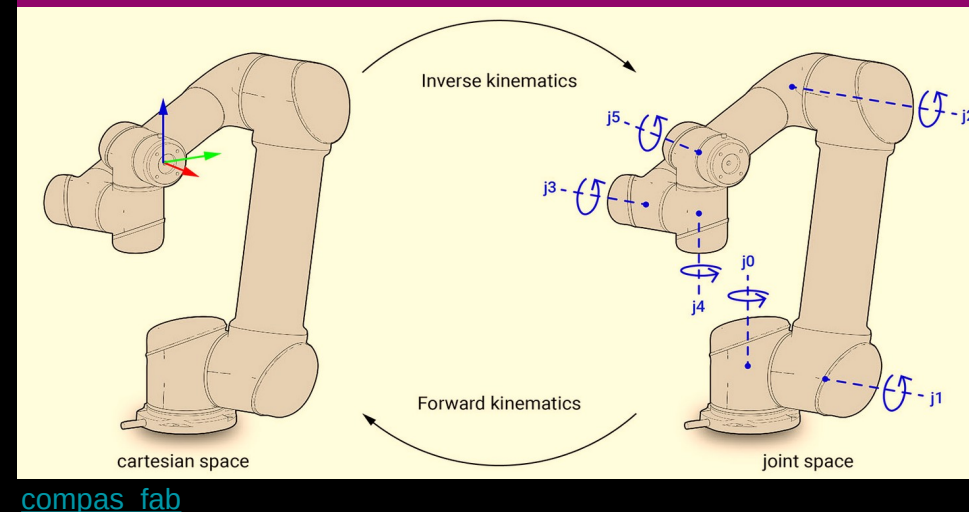
Focus Topics for Today



1. Intro to Robot Kinematics and Dynamics



2. Forward and Inverse Kinematics

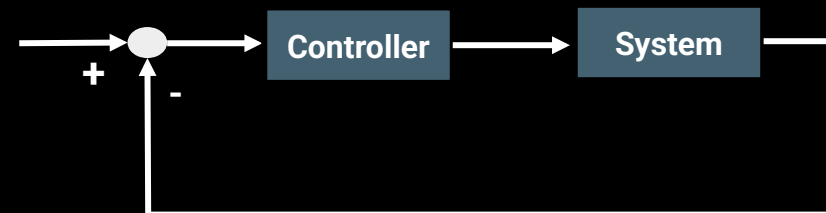


3. Kinematics and Dynamics for hand joints

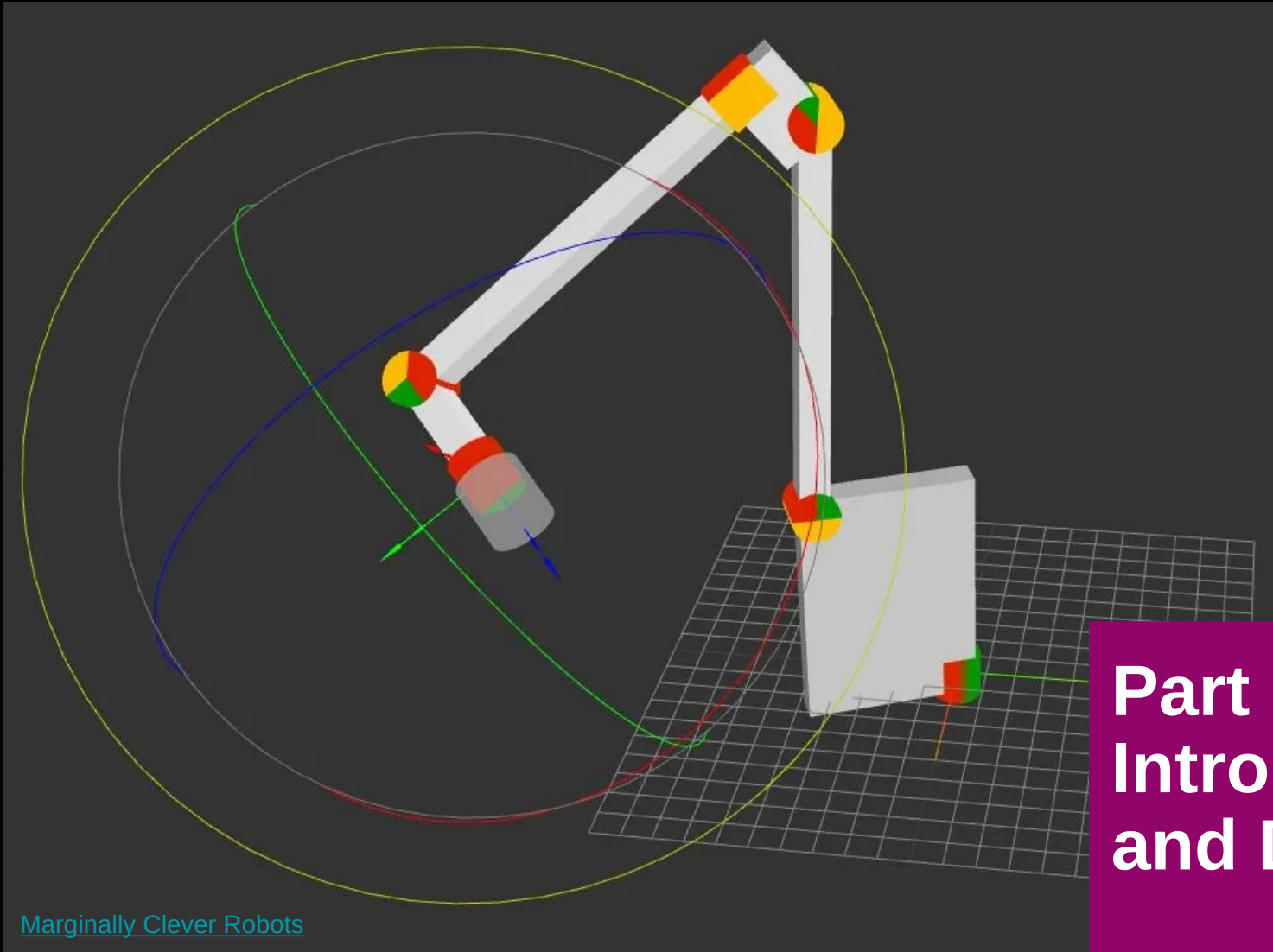


[Mimic Robotics](#)

4. Control

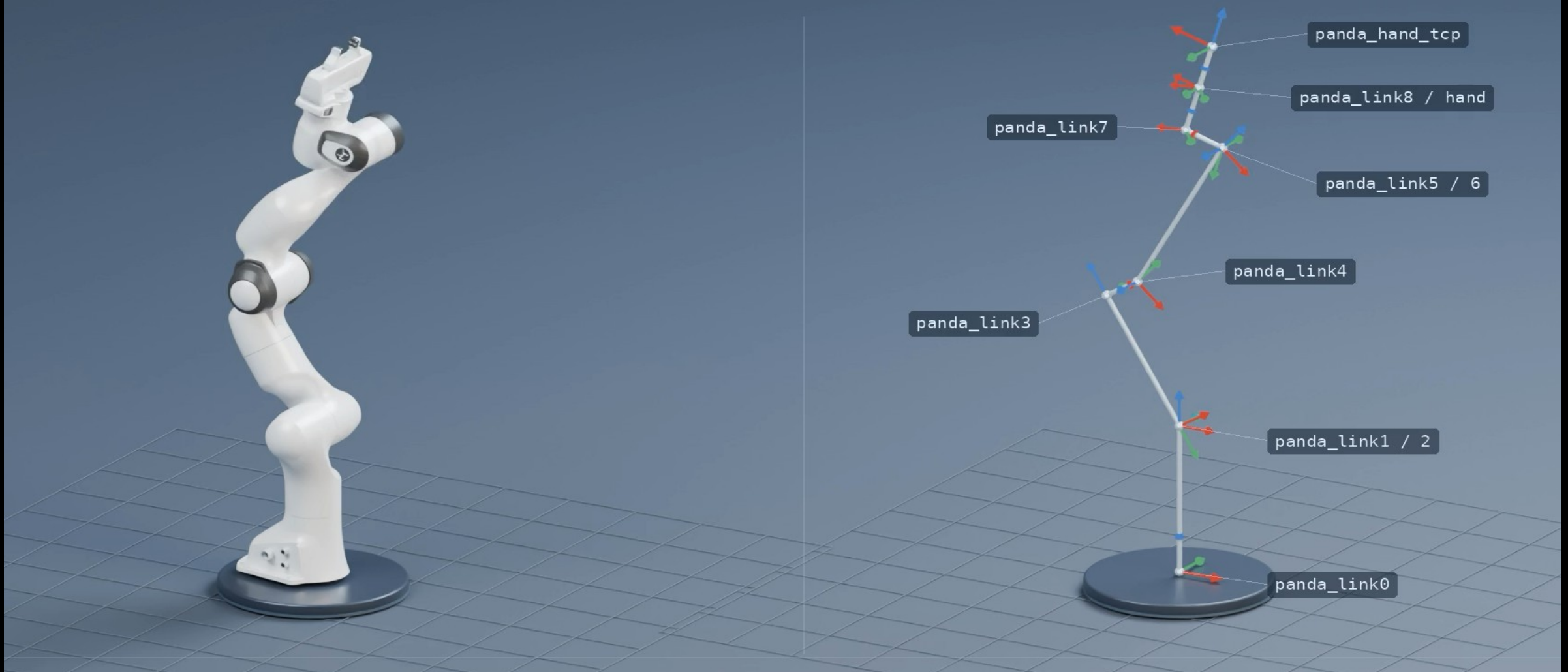


5. Challenges

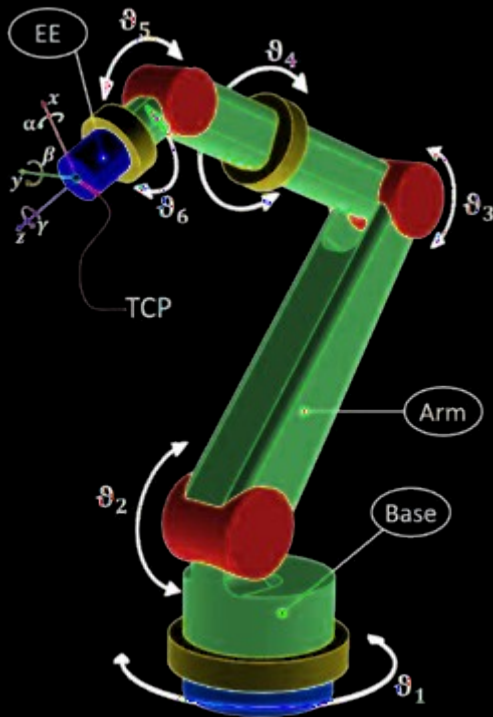


[Marginally Clever Robots](#)

Part 1: Intro to Robot Kinematics and Dynamics

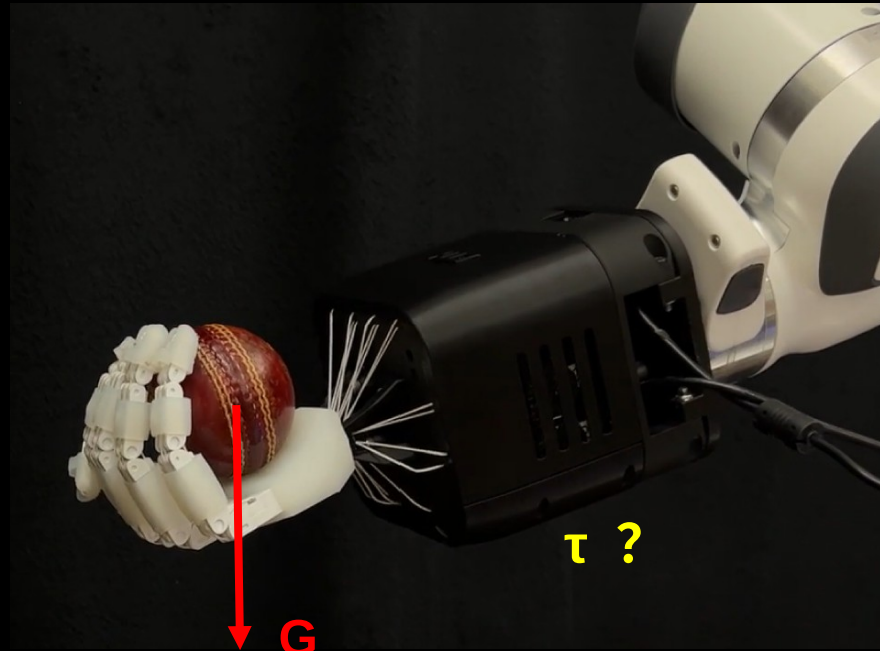


Robot Kinematics and Dynamics



researchgate.net

Kinematics



Toshimitsu et al. (2023) <https://srl-ethz.github.io/get-ball-rolling/>

Dynamics

Simulation

reaction to certain actuator commands

Control

invert of simulation, want to get somewhere, what to command?

Design

how are the loads distributed?

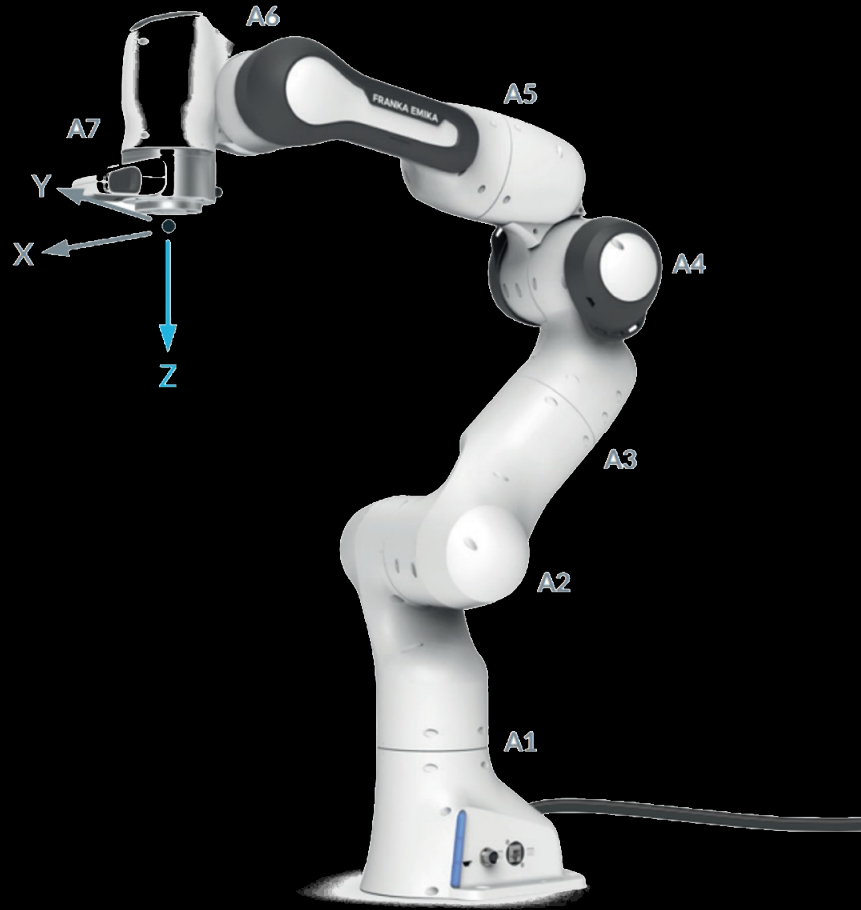
Optimization

what dimension should I have?

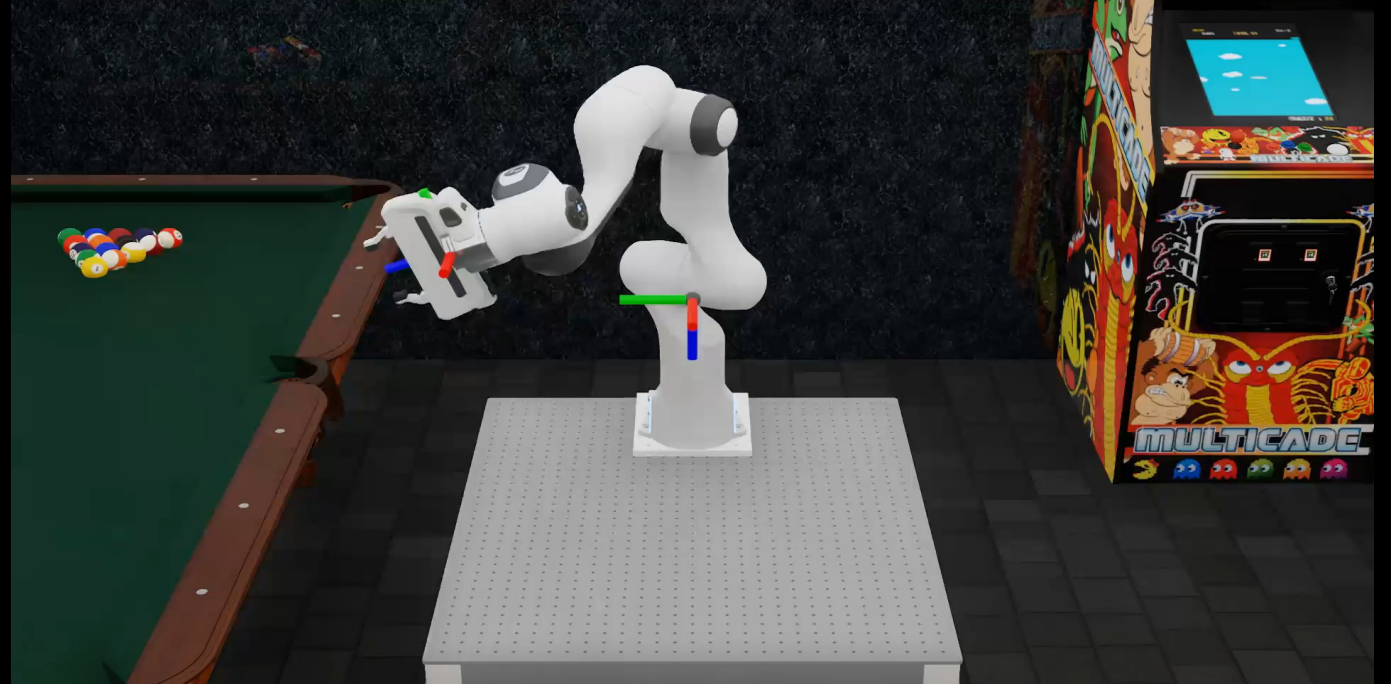
Actuation

torque, speed, powder etc.

Robotic Arm

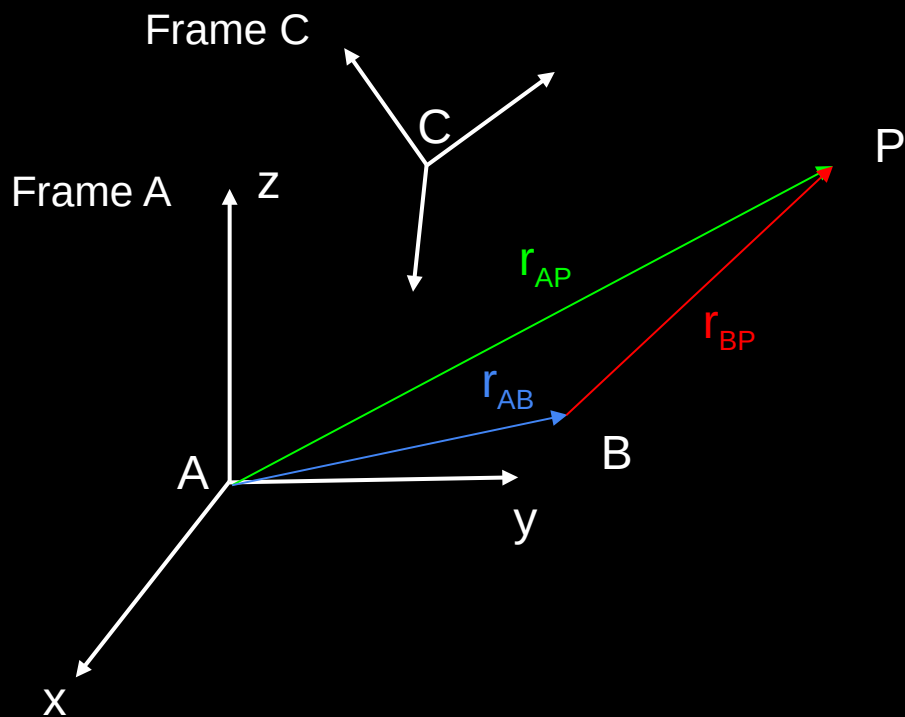


franka.de



Videos from Orbit

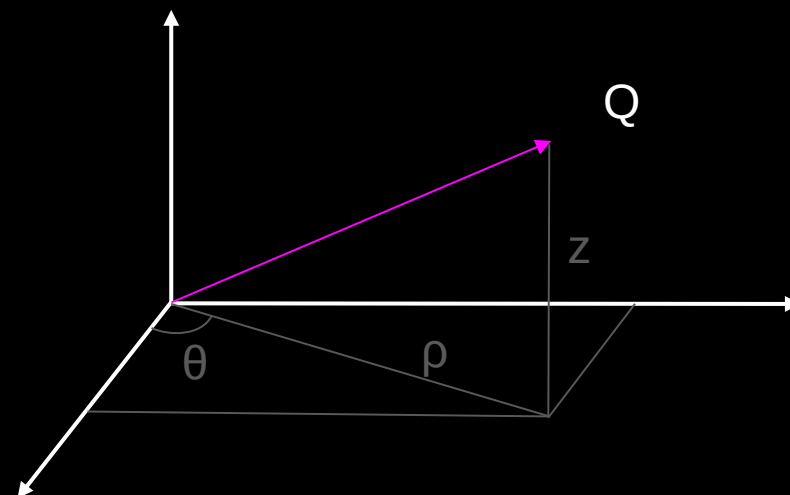
Points, Lines, and Coordinates



Point P in Cartesian Coordinates Frame A: ${}_A\mathbf{X}_P = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$

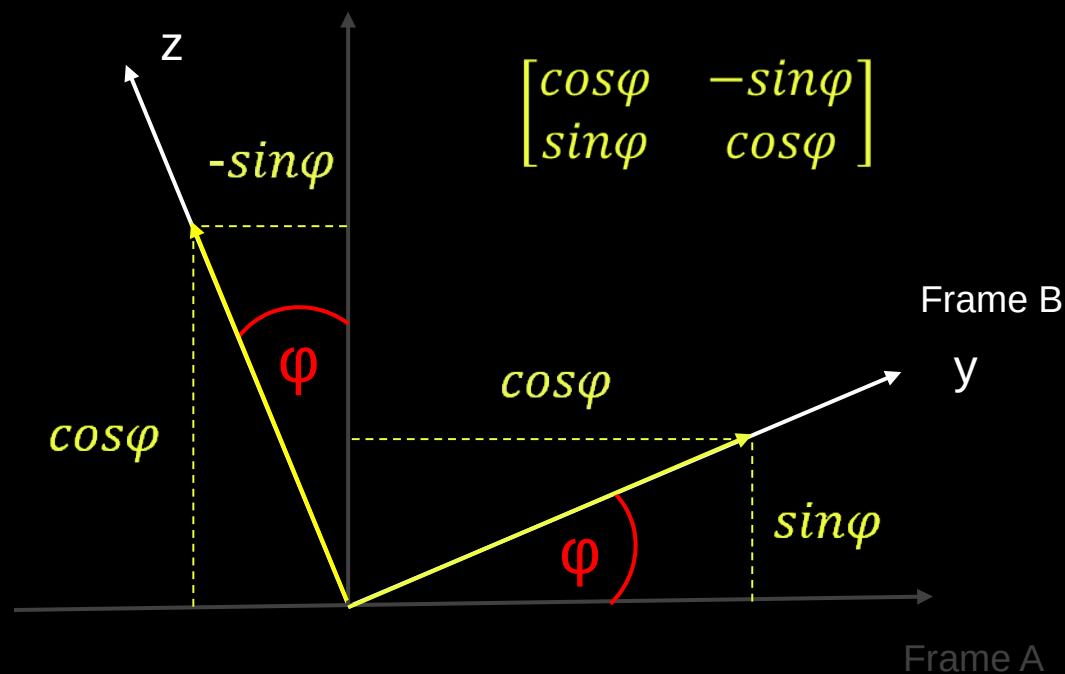
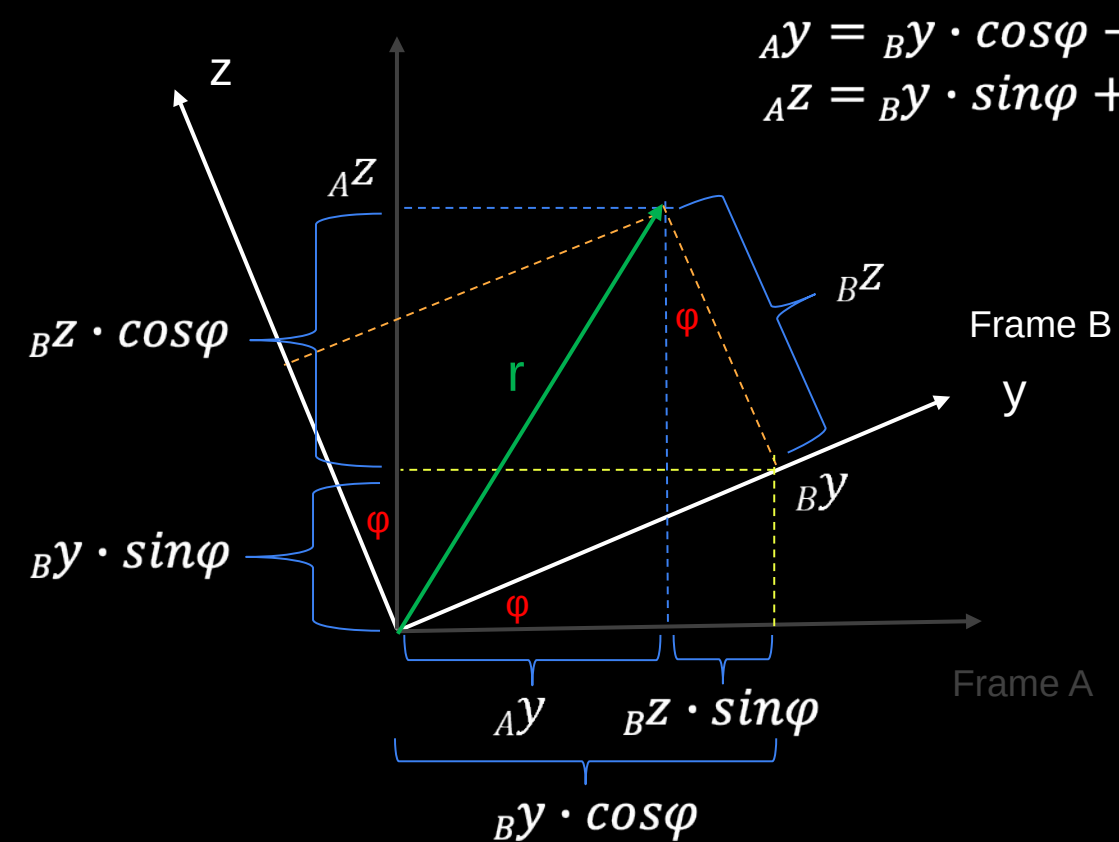
$${}_A\mathbf{r}_{AP} = {}_A\mathbf{r}_{AB} + {}_A\mathbf{r}_{BP}$$

$${}_A\mathbf{r}_{AP} \neq {}_A\mathbf{r}_{AB} + {}_C\mathbf{r}_{BP}$$



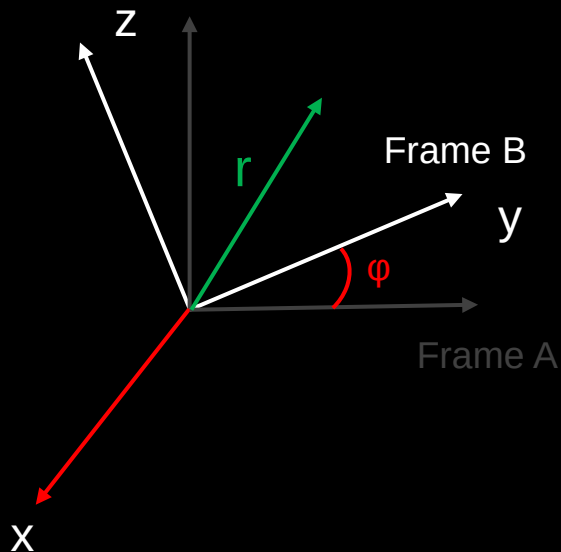
Point Q in Cylindrical Coordinate: $\mathbf{X}_Q = \begin{pmatrix} \rho \\ \theta \\ z \end{pmatrix}$

Rotation

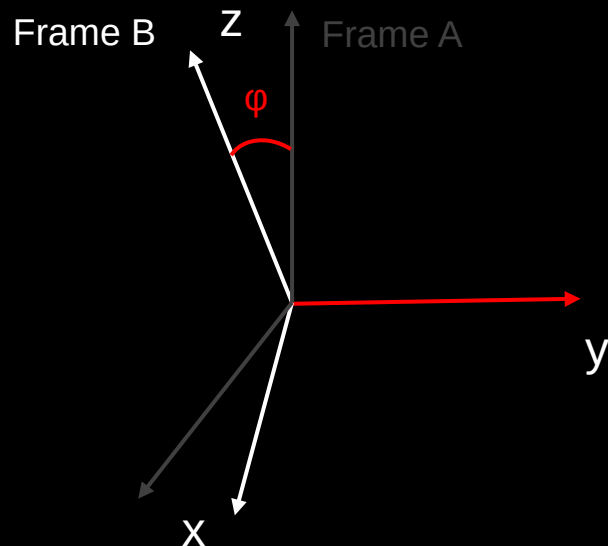


(only looking at y & z here)

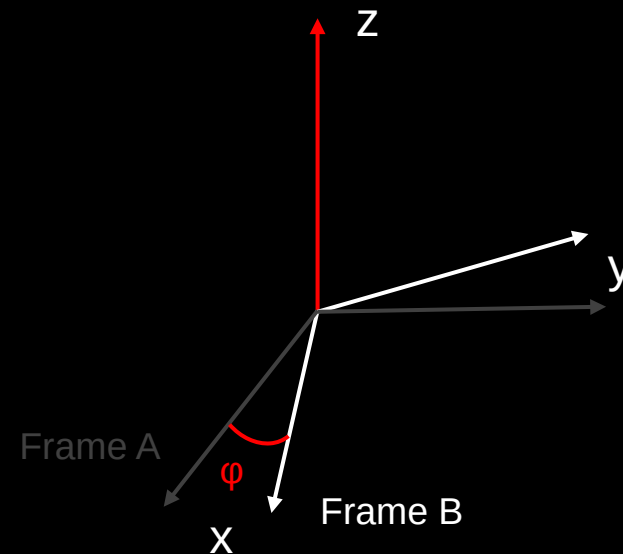
Rotation



$$C_x(\varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{bmatrix}$$



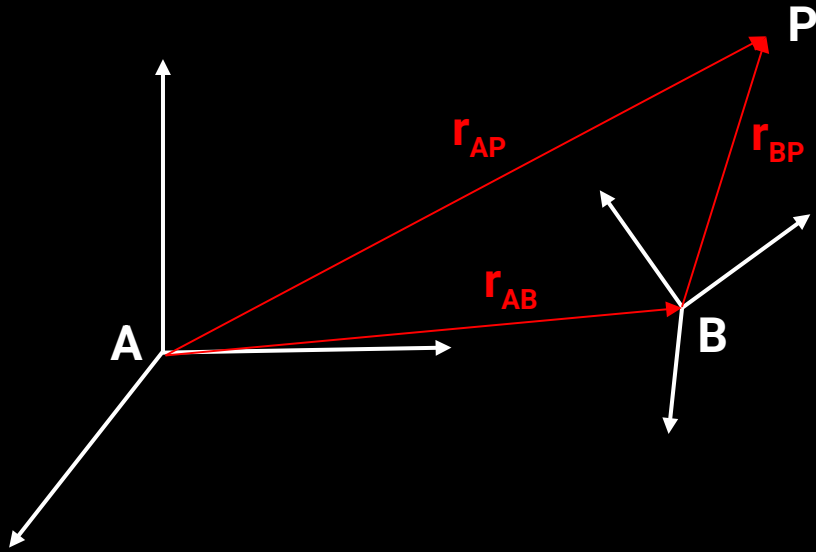
$$C_y(\varphi) = \begin{bmatrix} \cos\varphi & 0 & \sin\varphi \\ 0 & 1 & 0 \\ -\sin\varphi & 0 & \cos\varphi \end{bmatrix}$$



$$C_z(\varphi) = \begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}_A\mathbf{r} = \mathbf{C}_{AB} \cdot {}_B\mathbf{r} \rightarrow \begin{pmatrix} {}_A x \\ {}_A y \\ {}_A z \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{bmatrix} \begin{pmatrix} {}_B x \\ {}_B y \\ {}_B z \end{pmatrix} = \begin{pmatrix} {}_B x \\ {}_B y \cdot \cos\varphi - {}_B z \cdot \sin\varphi \\ {}_B y \cdot \sin\varphi + {}_B z \cdot \cos\varphi \end{pmatrix}$$

Homogeneous Transformation Matrix

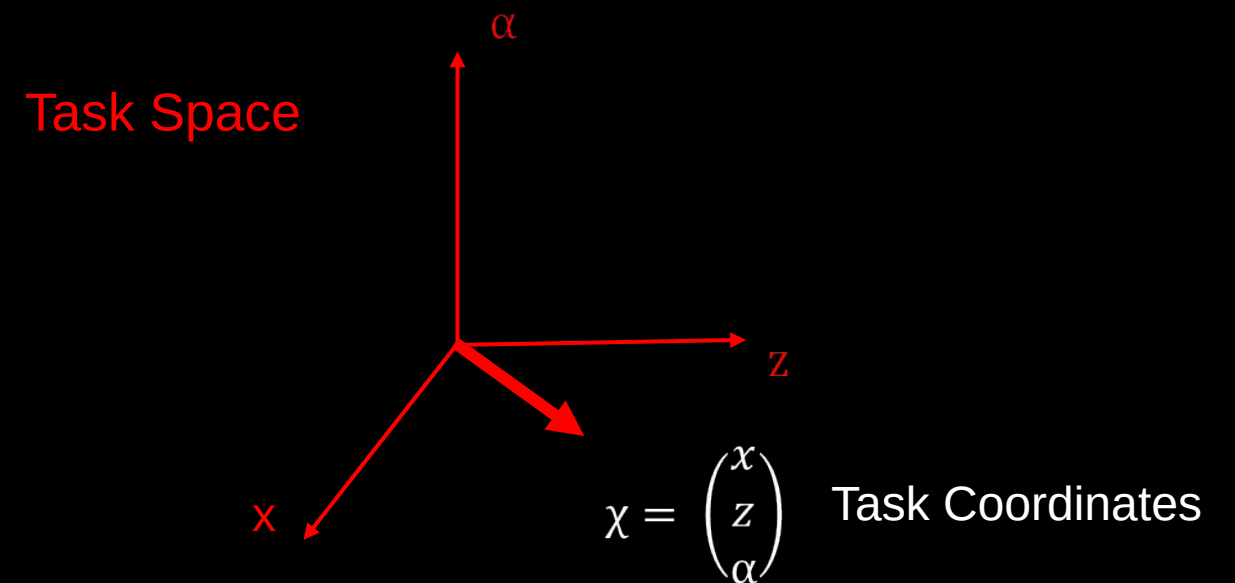
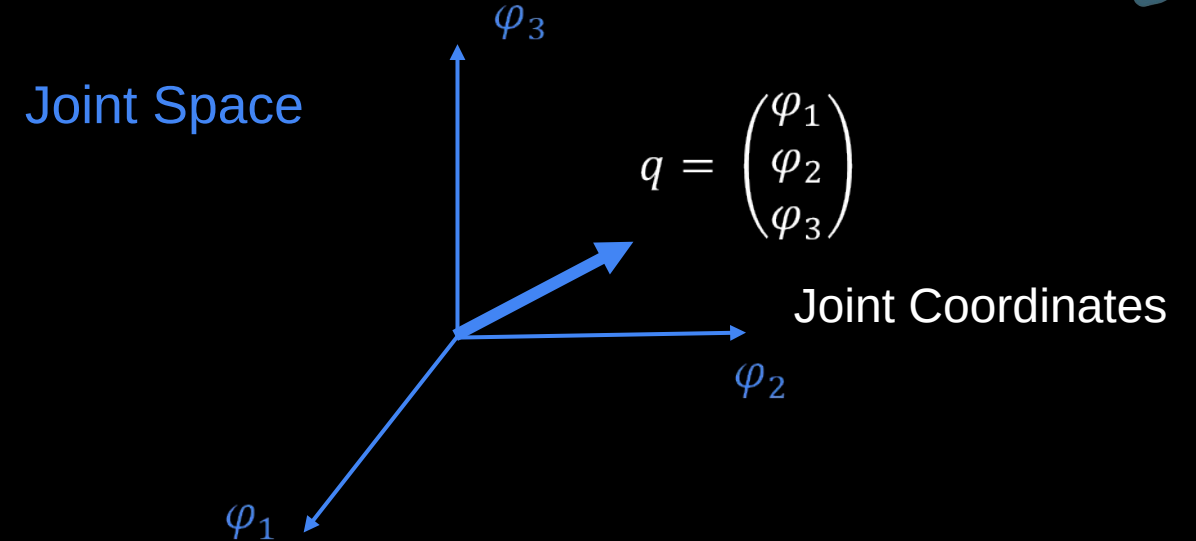
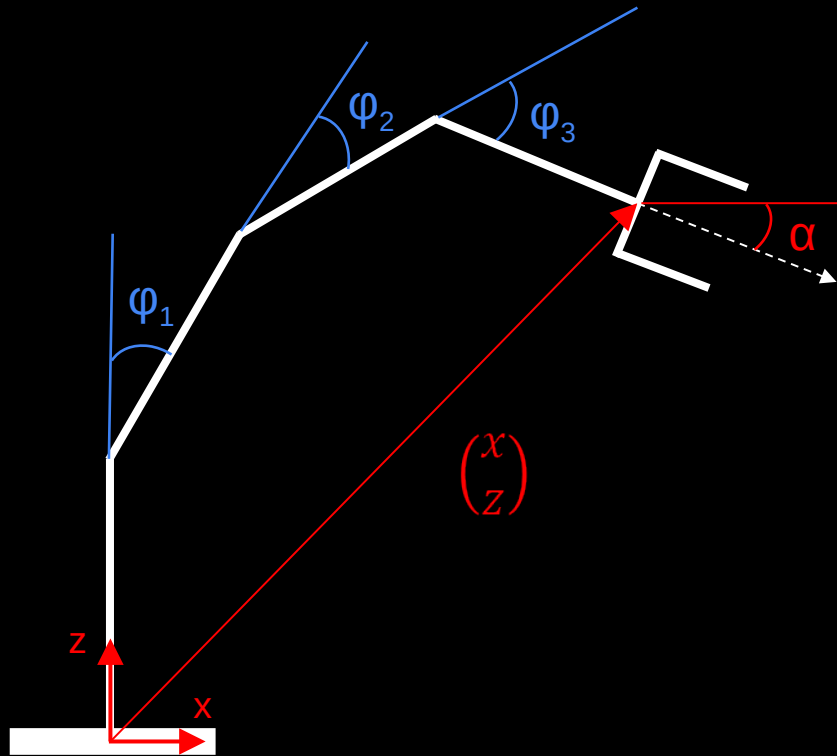


$$\mathbf{r}_{AP} = \mathbf{r}_{AB} + \mathbf{r}_{BP}$$

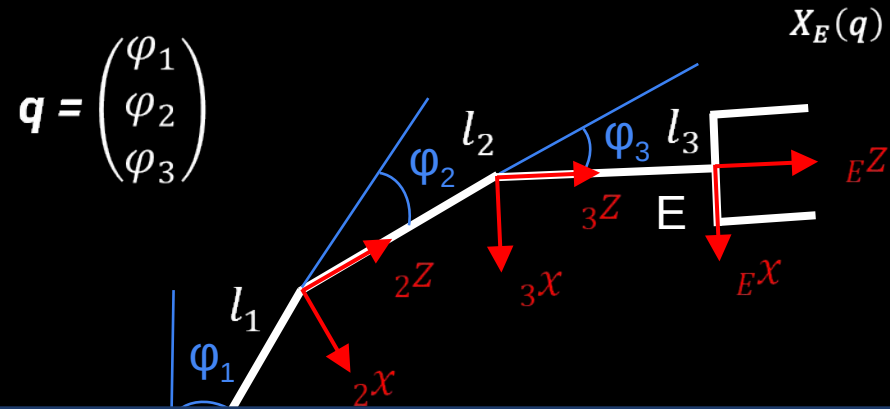
$${}^A\mathbf{r}_{AP} = {}^A\mathbf{r}_{AB} + {}^A\mathbf{r}_{BP} = {}^A\mathbf{r}_{AB} + \mathbf{C}_{AB} {}^B\mathbf{r}_{BP}$$

$$\begin{pmatrix} {}^A\mathbf{r}_{AP} \\ 1 \end{pmatrix} = \underbrace{\begin{bmatrix} \mathbf{C}_{AB} & {}^A\mathbf{r}_{AB} \\ \mathbf{0}_{1 \times 3} & 1 \end{bmatrix}}_{\mathbf{T}_{AB}} \begin{pmatrix} {}^B\mathbf{r}_{BP} \\ 1 \end{pmatrix}$$

Joint Space and Task Space

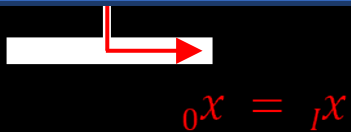


Joint Space and Task Space



$$T_{IE} = T_{I0} \cdot T_{01} \cdot T_{12} \cdot T_{23} \cdot T_{3E}$$

How about velocity



Forward Differential Kinematics and Jacobian



$$\delta X_E \approx \frac{\delta X_E(q)}{\delta q} \delta q = J_{EA}(q) \delta q \quad \text{with } J_{EA} = \frac{\delta X_E}{\delta q} = \begin{bmatrix} \frac{\delta X_1}{\delta q_1} & \dots & \frac{\delta X_1}{\delta q_n} \\ \vdots & \ddots & \vdots \\ \frac{\delta X_m}{\delta q_1} & \dots & \frac{\delta X_m}{\delta q_n} \end{bmatrix}$$

$$\dot{X}_E = J_{EA}(q) \dot{q} \quad \text{with } J_{EA}(q) \in \mathbb{R}^{m \times n}$$



- Previously we showed that:

$$J(q)\dot{q} = \dot{\chi}_e = \begin{bmatrix} \dot{p}_e \\ \dot{w}_e \end{bmatrix}$$

- If we invert it we obtain:

$$\dot{q} = J^+ \dot{\chi}_e \text{ with } J^+ = J^T (J J^T)^{-1}$$

- And in a differential form:

$$\Delta \chi_e = J^+ \Delta q$$

Algorithm 1 Numerical Inverse Kinematics

```
1:  $\mathbf{q} \leftarrow \mathbf{q}^0$  ▷ Start configuration
2: while  $\|\chi_e^* - \chi_e(\mathbf{q})\| > tol$  do ▷ While the solution is not reached
3:    $\mathbf{J}_{eA} \leftarrow \mathbf{J}_{eA}(\mathbf{q}) = \frac{\partial \chi_e}{\partial \mathbf{q}}(\mathbf{q})$  ▷ Evaluate Jacobian for  $\mathbf{q}$ 
4:    $\mathbf{J}_{eA}^+ \leftarrow (\mathbf{J}_{eA})^+$  ▷ Calculate the pseudo inverse
5:    $\Delta \chi_e \leftarrow \chi_e^* - \chi_e(\mathbf{q})$  ▷ Find the end-effector configuration error vector
6:    $\mathbf{q} \leftarrow \mathbf{q} + \mathbf{J}_{eA}^+ \Delta \chi_e$  ▷ Update the generalized coordinates
7: end while
```

A possible inverse kinematics algorithm

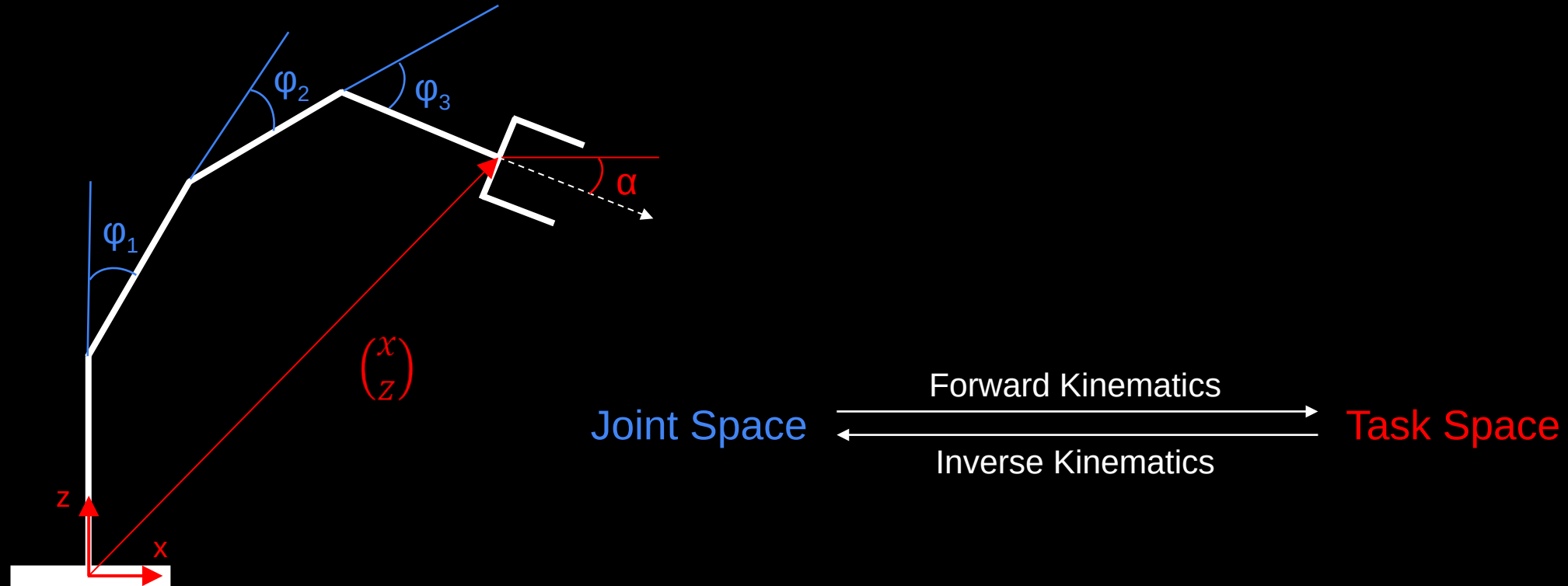
Robot Dynamics Class @ ETH Zurich

To overcome stability issues, the update can be scaled by a factor k

-> slower convergence

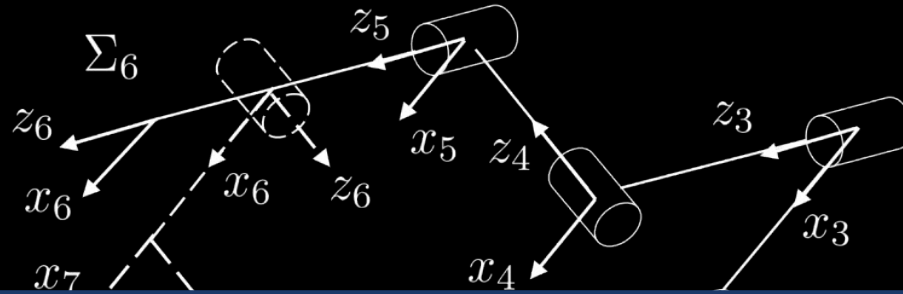
$$\mathbf{q} \leftarrow \mathbf{q} + k \mathbf{J}_{eA}^+ \Delta \chi_e \text{ with } k \in (0, 1)$$

Forward and Inverse Kinematics



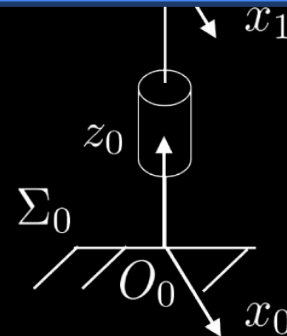
From greek *kinema* = motion

Underactuation and Overactuation

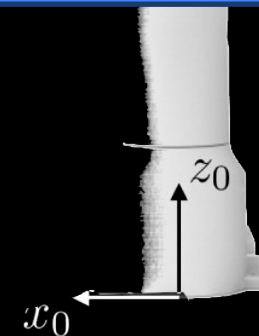


What's the number of rows and columns of $J(q)$

Filippeschi et al. *Kinematic Optimization for the Design of a Collaborative Robot End-Effector for Tele-Echography* (2021)



(a)

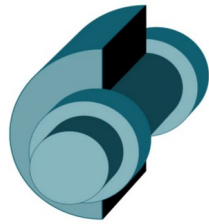


(b)

Recap: Different Types of Joints



SOFT ROBOTICS - JOINT TYPES



PIN



FLEXURE



SYNOVIAL

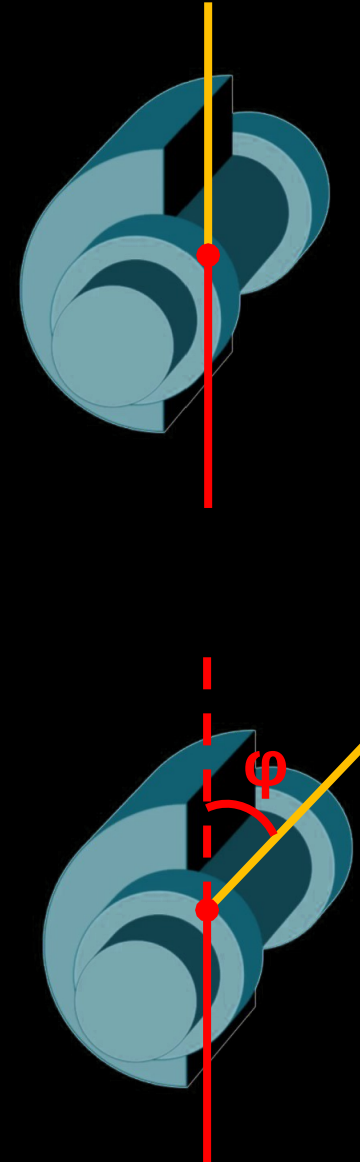
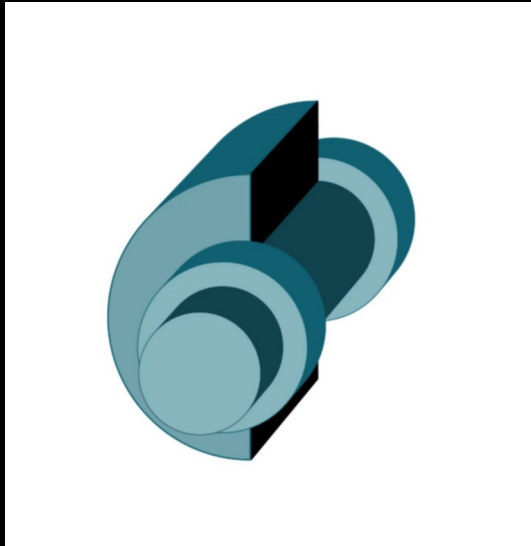


ROLLING
CONTACT

Pin Joint



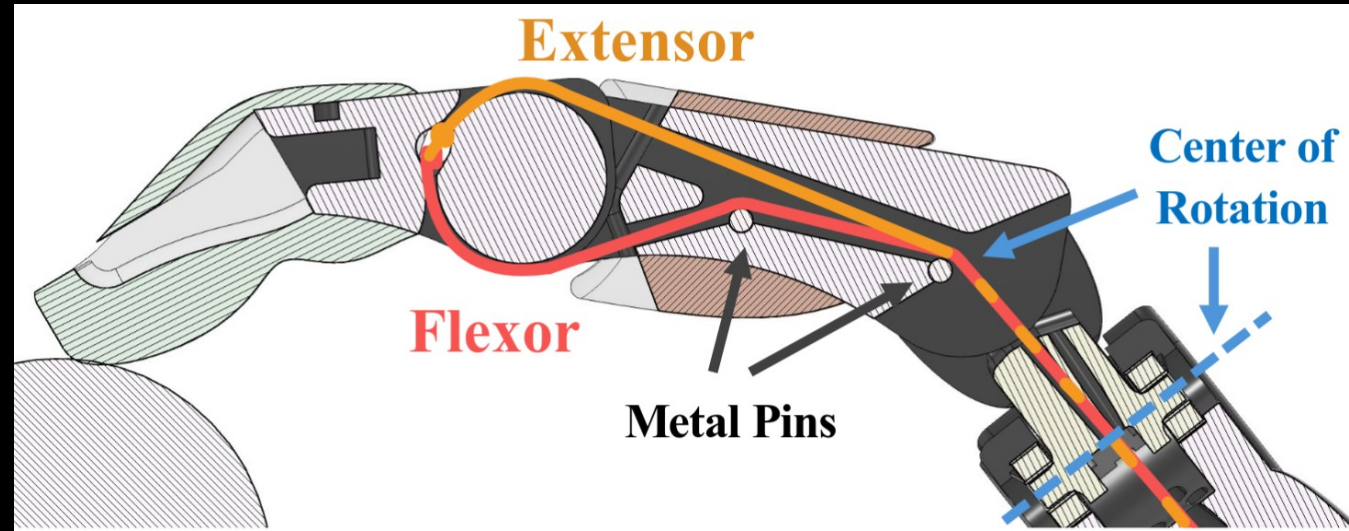
What the ORCA Hand uses



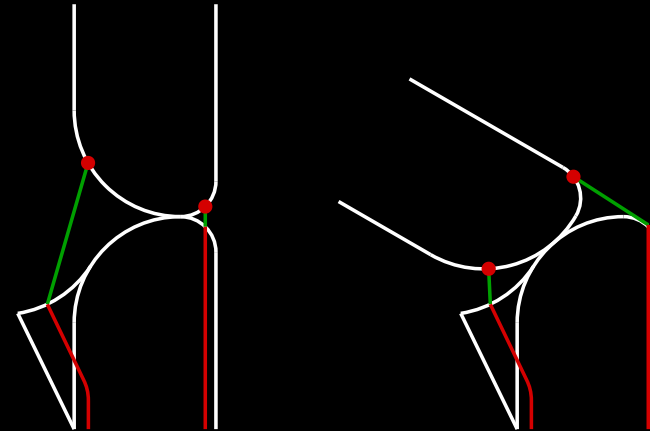
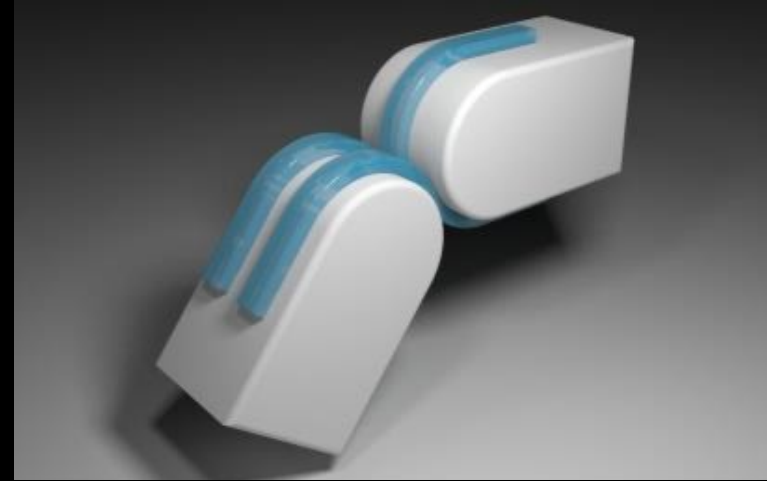
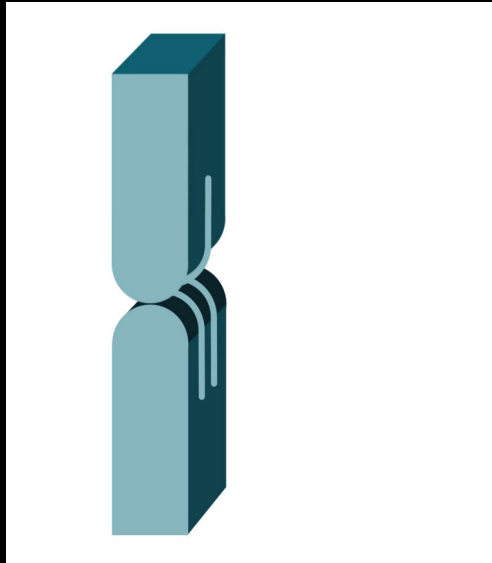
Pin Joint



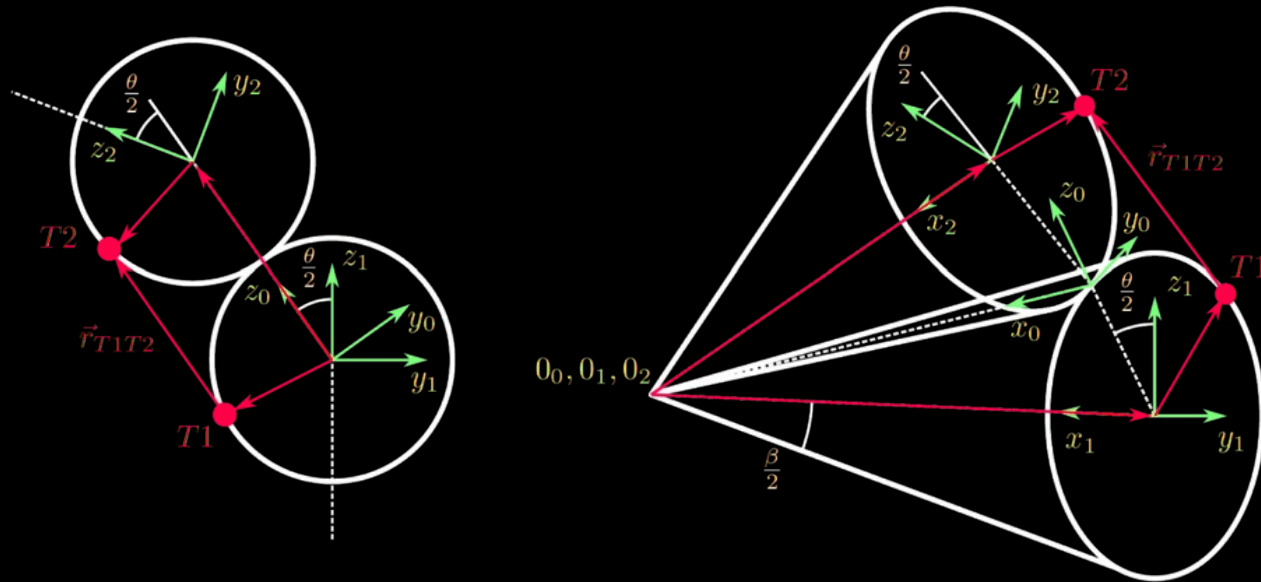
What the ORCA Hand uses

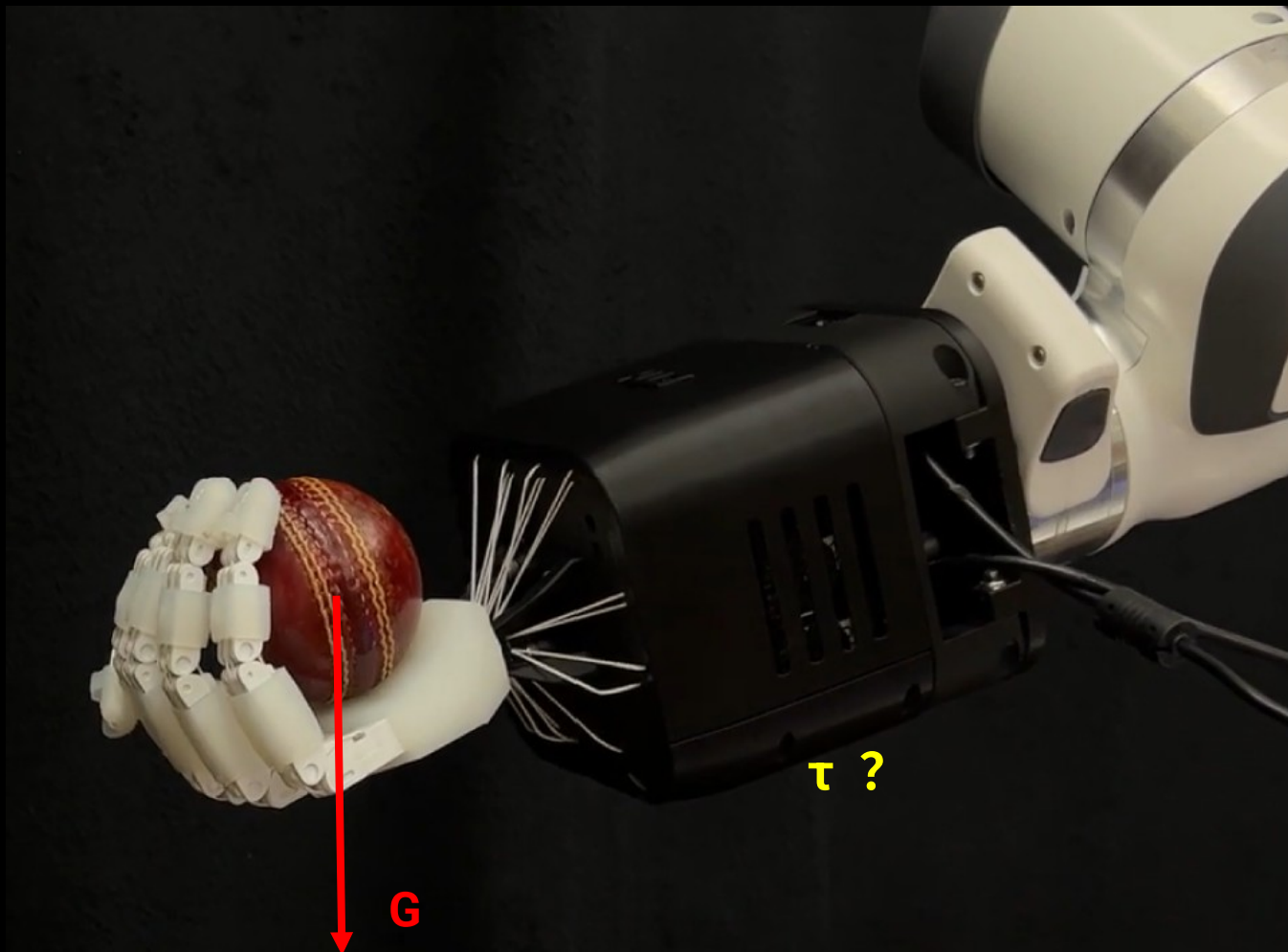


Rolling Contact Joint – Joints Used on Faive Hand



Kinematics for Rolling Contact Joint





Dynamics: End-Effector Force and Joint Torque



Task space
velocity

Joint space
velocity

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

$$\tau^T \dot{q} = F^T \dot{x}$$

$$\left. \begin{array}{l} \dot{x} = J \dot{q} \\ \tau^T \dot{q} = F^T \dot{x} \end{array} \right\} \tau^T \dot{q} = F^T J \dot{q}$$

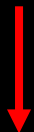
Conservation of Power

$$\tau = J^T F$$

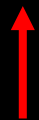
Dynamics



Velocity of the
finger joints



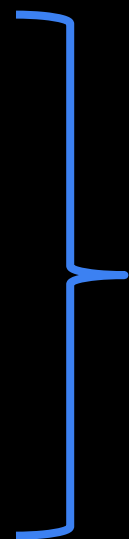
$$\dot{p} = J_m \cdot \dot{q}$$



Velocity of
the motors

$$\tau^T \cdot \dot{q} = T^T \cdot \dot{p}$$

Conservation of Power

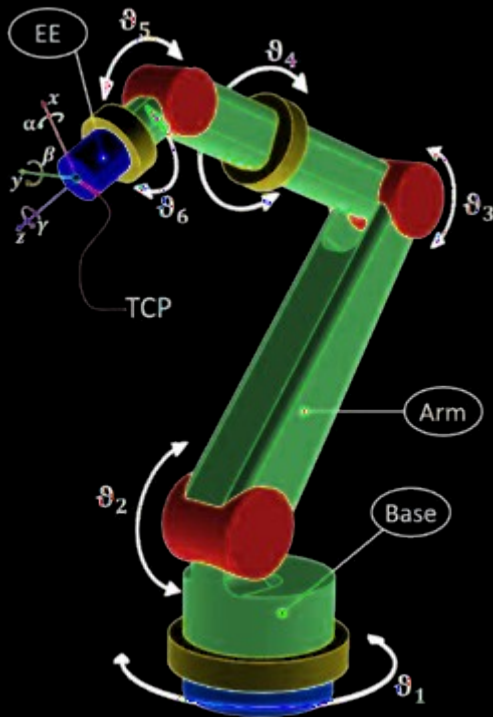


$$\tau^T \cdot \dot{q} = T^T \cdot J_m \cdot \dot{q}$$



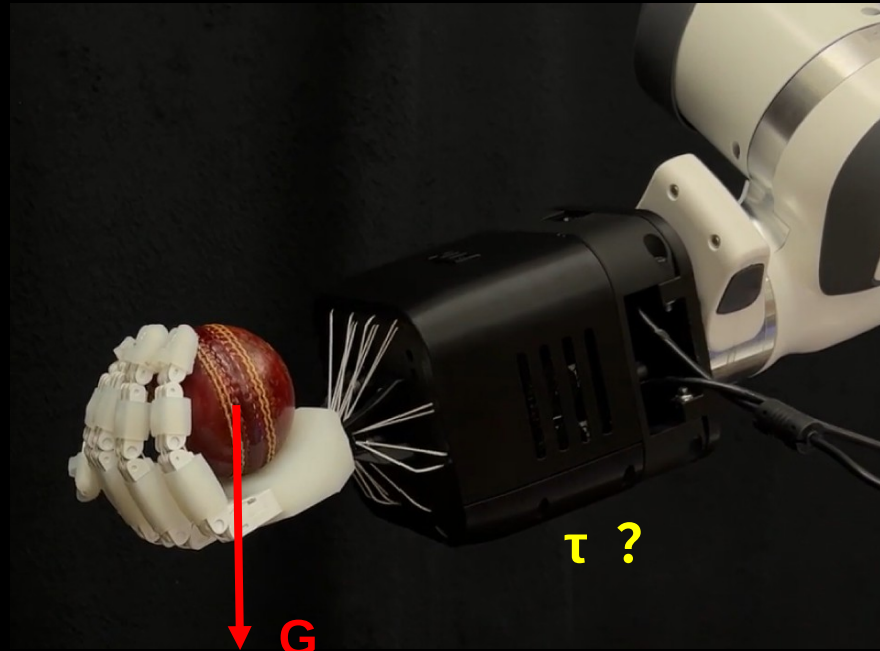
$$\tau = J_m^T \cdot T$$

Robot Kinematics and Dynamics



researchgate.net

Kinematics



Toshimitsu et al. (2023) <https://srl-ethz.github.io/get-ball-rolling/>

Dynamics

Simulation

reaction to certain actuator commands

Control

invert of simulation, want to get somewhere, what to command?

Design

how are the loads distributed?

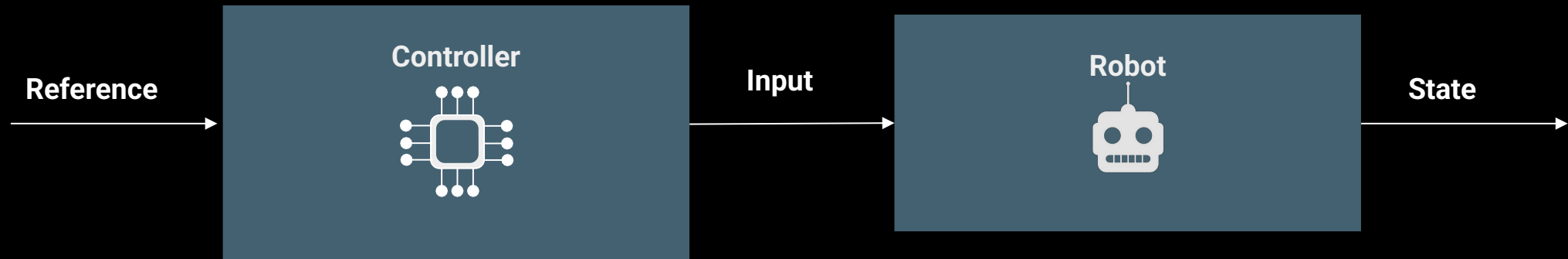
Optimization

what dimension should I have?

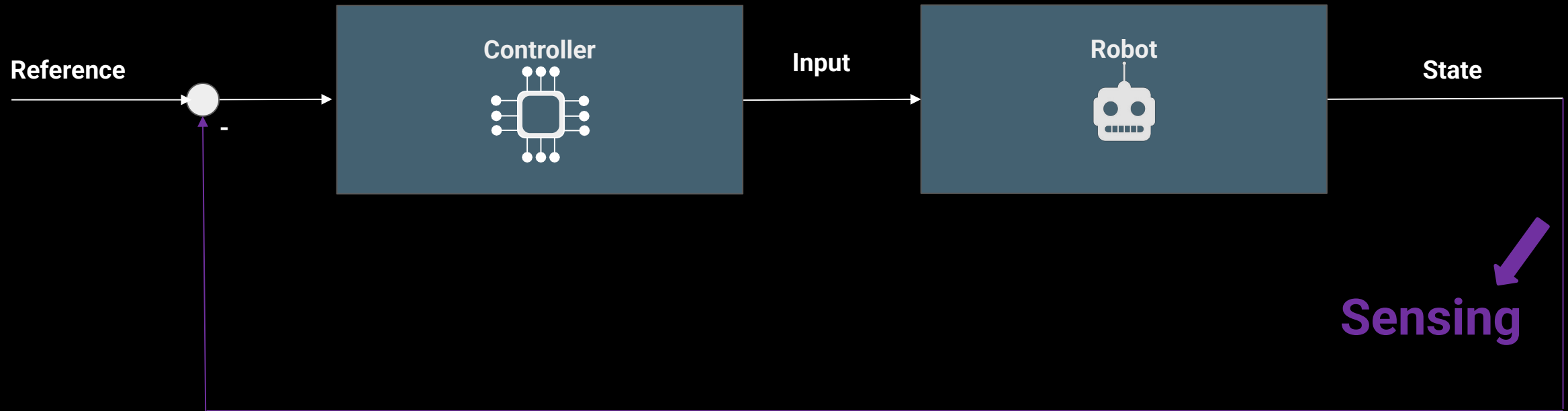
Actuation

torque, speed, powder etc.

Simplest controller possible: Open loop



Closed Loop Controller



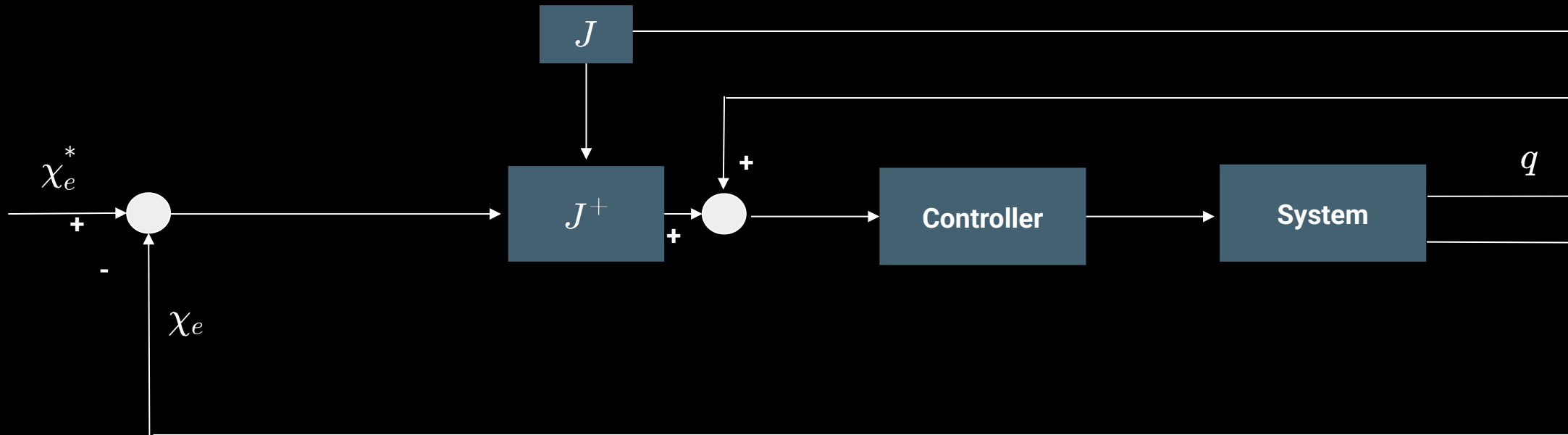
Remember - Inverse Kinematics



Numerical Inverse Kinematics (iterative approach):

$$q \leftarrow q + kJ_{eA}^+ \Delta \chi_e \text{ with } k \in (0, 1)$$

Inverse Kinematics Control



Trajectory/Task space control → PID Control

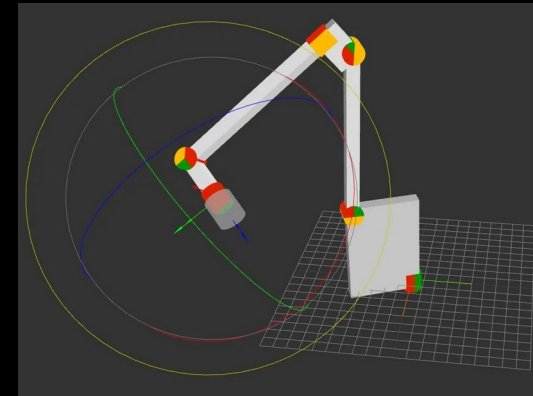


- Idea: Compare desired vs actual output → compute error → apply correction (Proportional, Integral, Derivative).
- **Strengths:** Simple, cheap, widely used, doesn't require a full model.
- **Weaknesses:** Limited with nonlinear or high-dimensional systems; needs tuning; not predictive.

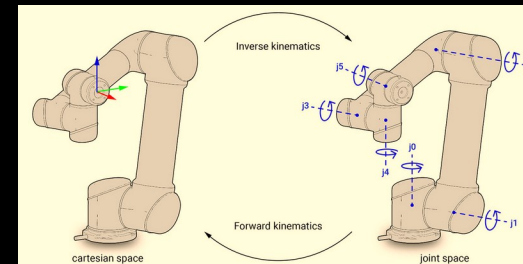
Summary of Kinematics and Dynamics



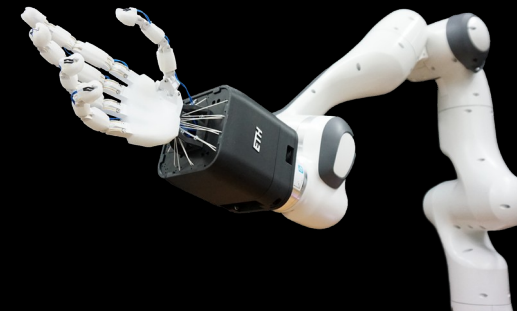
- **Intro to Robot Kinematics and Dynamics**
 - Representing points and lines in different coordinates and frames
 - Rotational matrix
 - Joint space and task space
- **Forward and Inverse Kinematics**
 - Homogeneous transformation matrix
 - Forward differential kinematics and Jacobian
 - Inverse kinematics
- **Kinematics and Dynamics for hand joints**
 - Hand Joints
 - Kinematics for rolling joints
 - Dynamics for rolling joints



[Marginally Clever Robots](#)



[compas_fab](#)



[Faive Robotics](#)

Trajectory Control



We can use a closed loop controller, but we need to add a component for the desired velocities

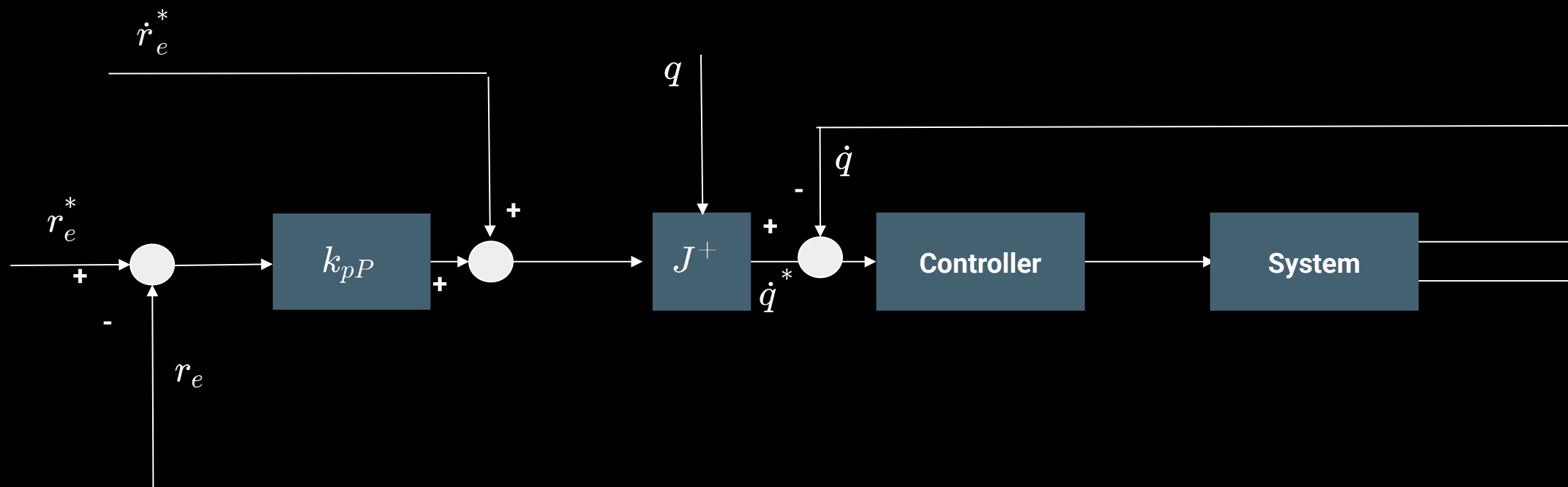
We define $\Delta r_e^t = r_e^*(t) - r_e(q^t)$

And the desired joint velocity $\dot{q}^*$ $= J_{e0_P}^+(q^t) \cdot (\dot{r}_e^*(t) + k_{pP} \Delta r_e^t)$

If we have a desired rotation rate we write $\dot{q}^*$ $= J_{e0_R}^+(q^t) \cdot (\omega_e^*(t) + k_{pR} \Delta \phi)$

Where ϕ are the angles used to represent the orientation of the end effector.

Trajectory Control



Dynamic control



The dynamic model is

$$M(q)\ddot{q} + b(q, \dot{q}) + g(q) = \tau + J_c(q)^T F_c$$

With:

$M(q)$: Generalized mass matrix

$q, \dot{q}, \ddot{q}$: Generalized position, velocity and acceleration vector

$b(q, \dot{q})$: Coriolis and centrifugal terms

$g(q)$: Gravitational terms

τ : External generalized forces

F_c : External Cartesian forces

$J_c(q)$: Geometric Jacobian corresponding to the external forces



Dynamic control

The dynamic model is

$$M(q)\ddot{q} + b(q, \dot{q}) + g(q) = \tau + J_c(q)^T F_c$$

If we know the desired generalized accelerations, velocities and poses we can write

$$\ddot{q}^* = k_p(q^* - q) + k_d(\dot{q}^* - \dot{q})$$

Thus the joint torques will be

$$\tau^* = M(q)\ddot{q}^* + b(q, \dot{q}) + g(q)$$



Task-space control

Remember that $J(q)\dot{q} = \chi_e = \begin{bmatrix} \dot{p}_e \\ w_e \end{bmatrix}$

If you derive that with respect to time: $\dot{\chi}_e = J(q)\ddot{q} + \dot{J}(q)\dot{q}$

And if we solve the dynamics equation for the joint acceleration and substitute in the equation above we get:

$$\dot{\chi}_e = JM^{-1}(\tau - b - g) + \dot{J}\dot{q}$$

Finally, remembering that $\tau = J_e^T F_e$

$$\Lambda_e \dot{\chi}_e + \mu + p = F_e$$

We can write

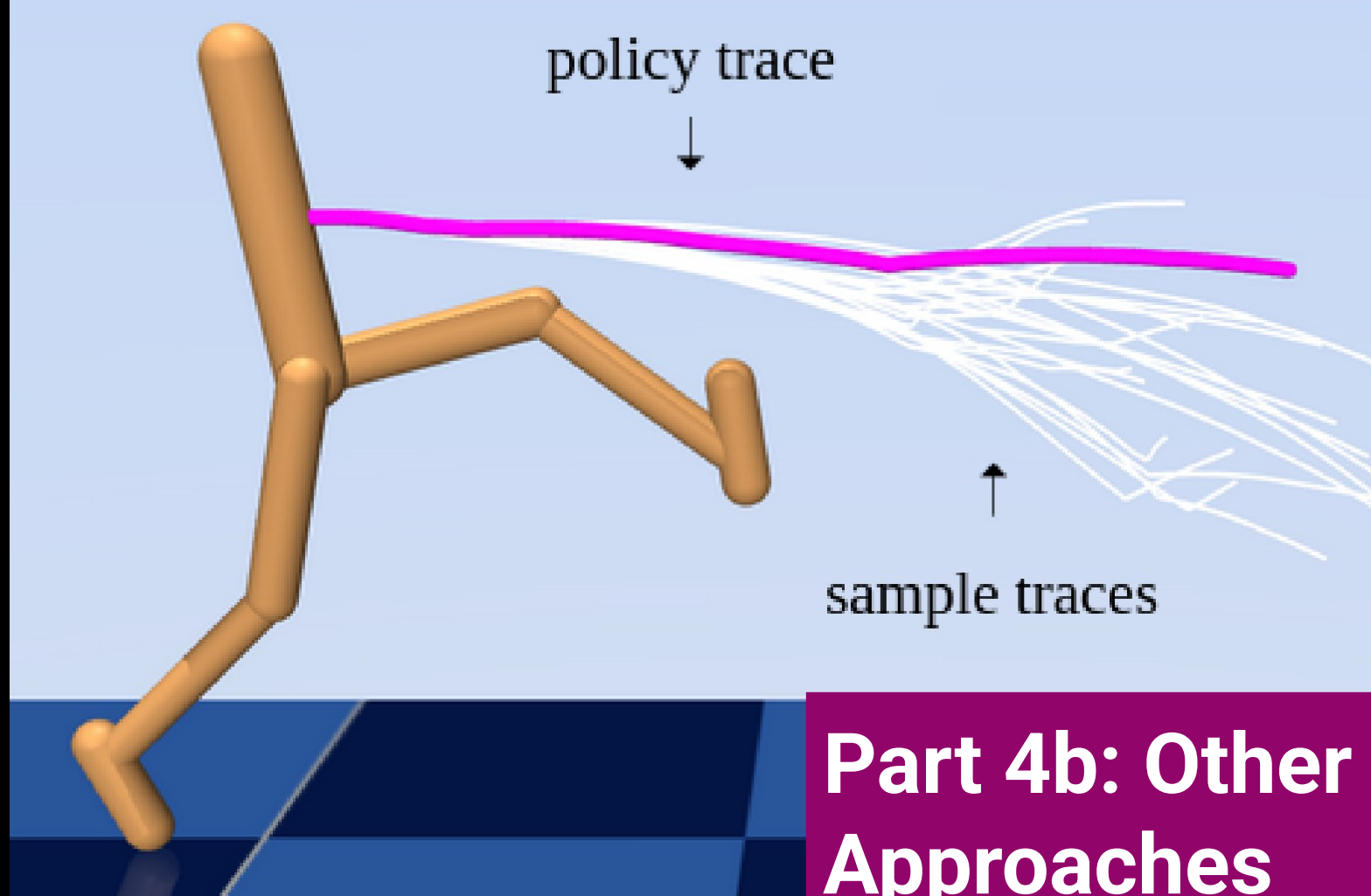
$$\begin{aligned} \Lambda_e &= (J_e M^{-1} J_e^T)^{-1} \\ \mu &= \Lambda_e J_e M^{-1} b - \Lambda_e \dot{J}_e \dot{q} \\ p &= \Lambda_e J_e M^{-1} g \end{aligned}$$

Task-space control



Defining the dynamics uniquely depending on the state of the end effector allows us to design a control loop

$$\boxed{\dot{\chi}_e^*} = \begin{pmatrix} r_e^* - r_e \\ \Delta\phi_e \end{pmatrix} + k_d(\chi_e^* - \chi_e)$$



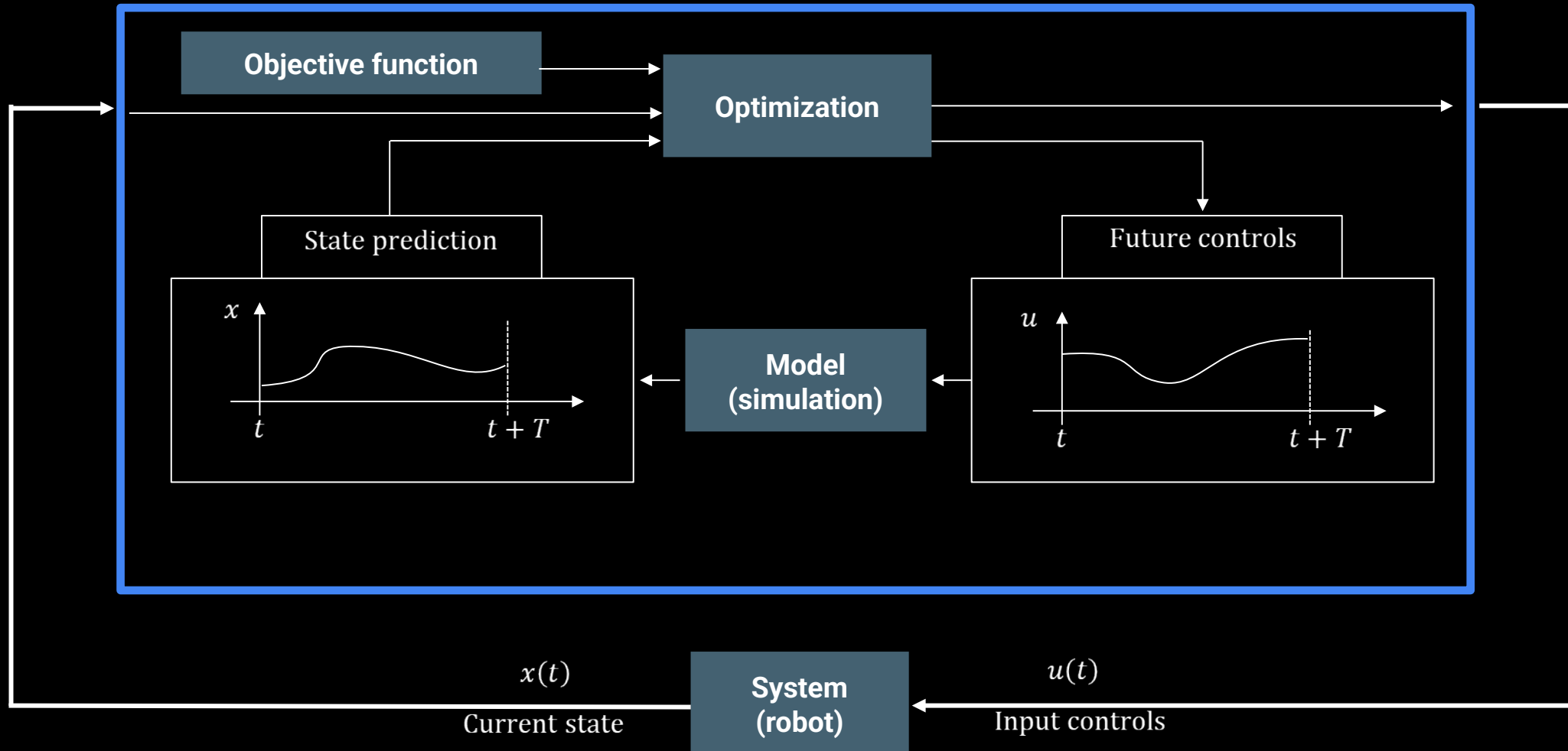
Mujoco PC

Part 4b: Other Control Approaches

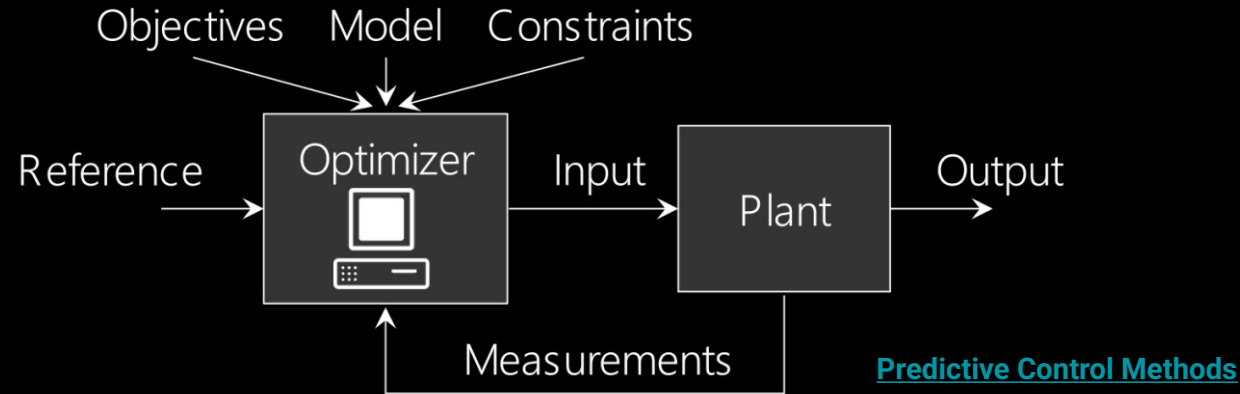
Model predictive control



Model predictive controller



Model Predictive Control (MPC)

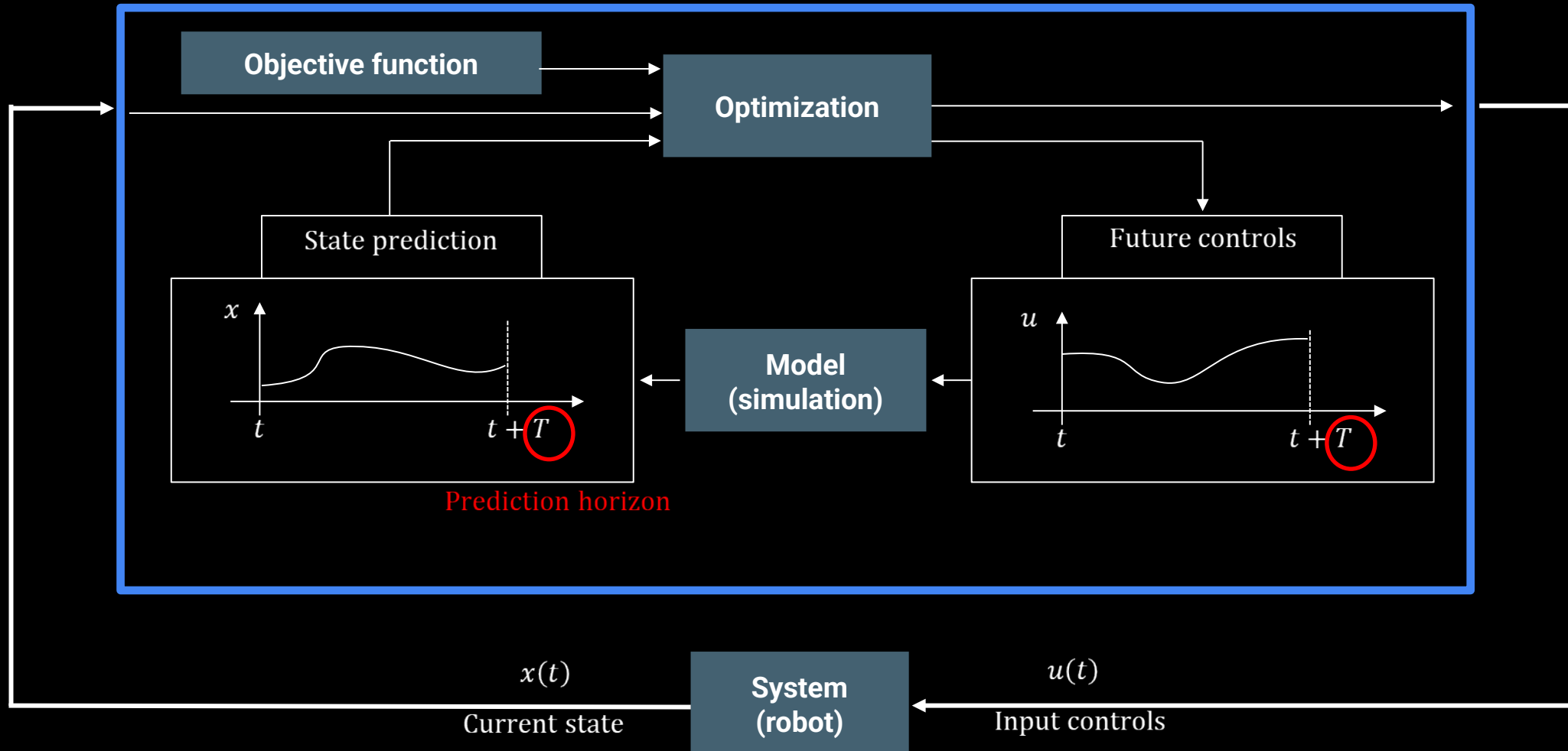


- **Idea:** Dynamics model to *predict future states over a horizon* → optimize control inputs to minimize a cost.
- **Strengths:** Handles constraints, anticipates the future.
- **Weaknesses:** Computationally expensive, requires an accurate model (sim-to-real gap).

Model predictive control



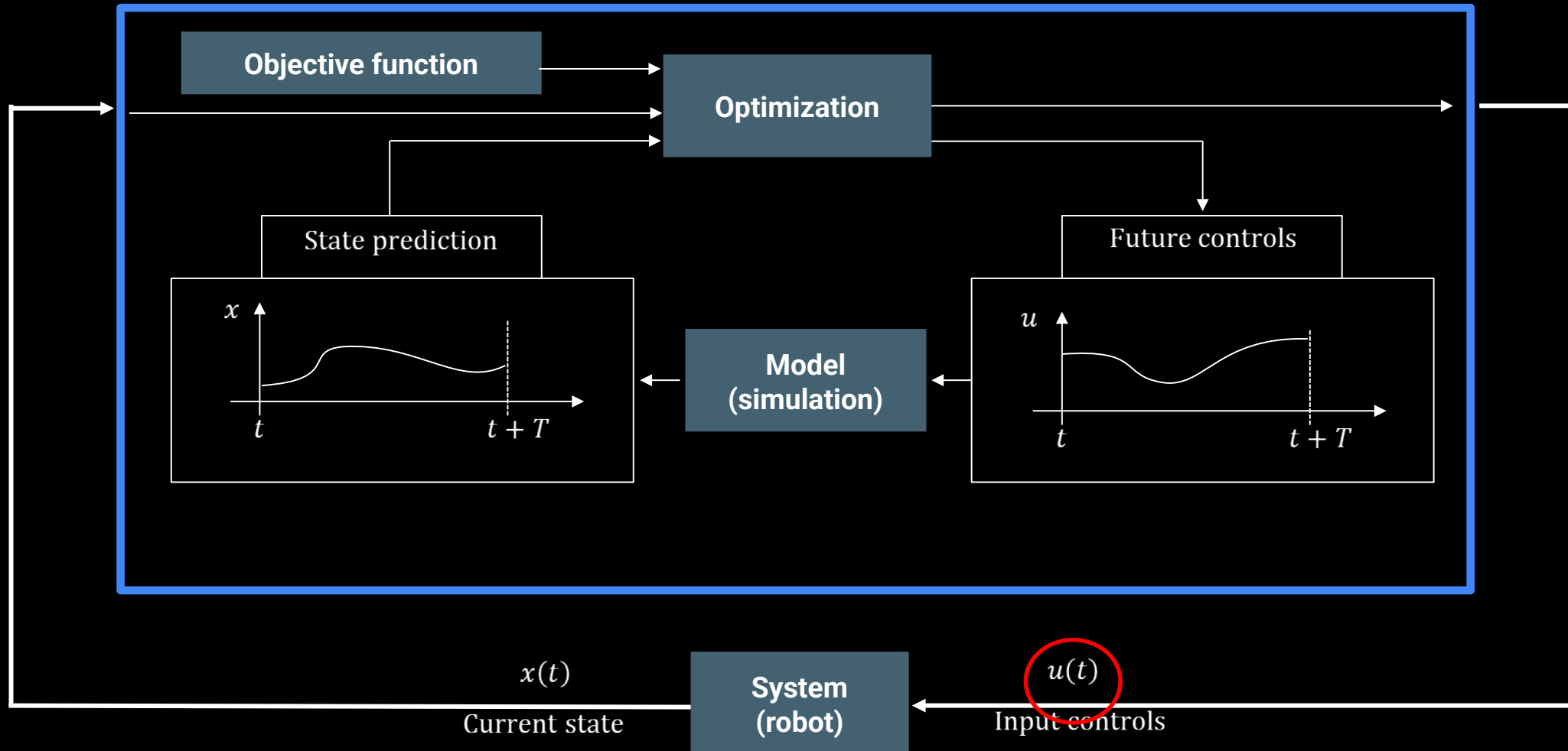
Model predictive controller



Model predictive control



Model predictive controller

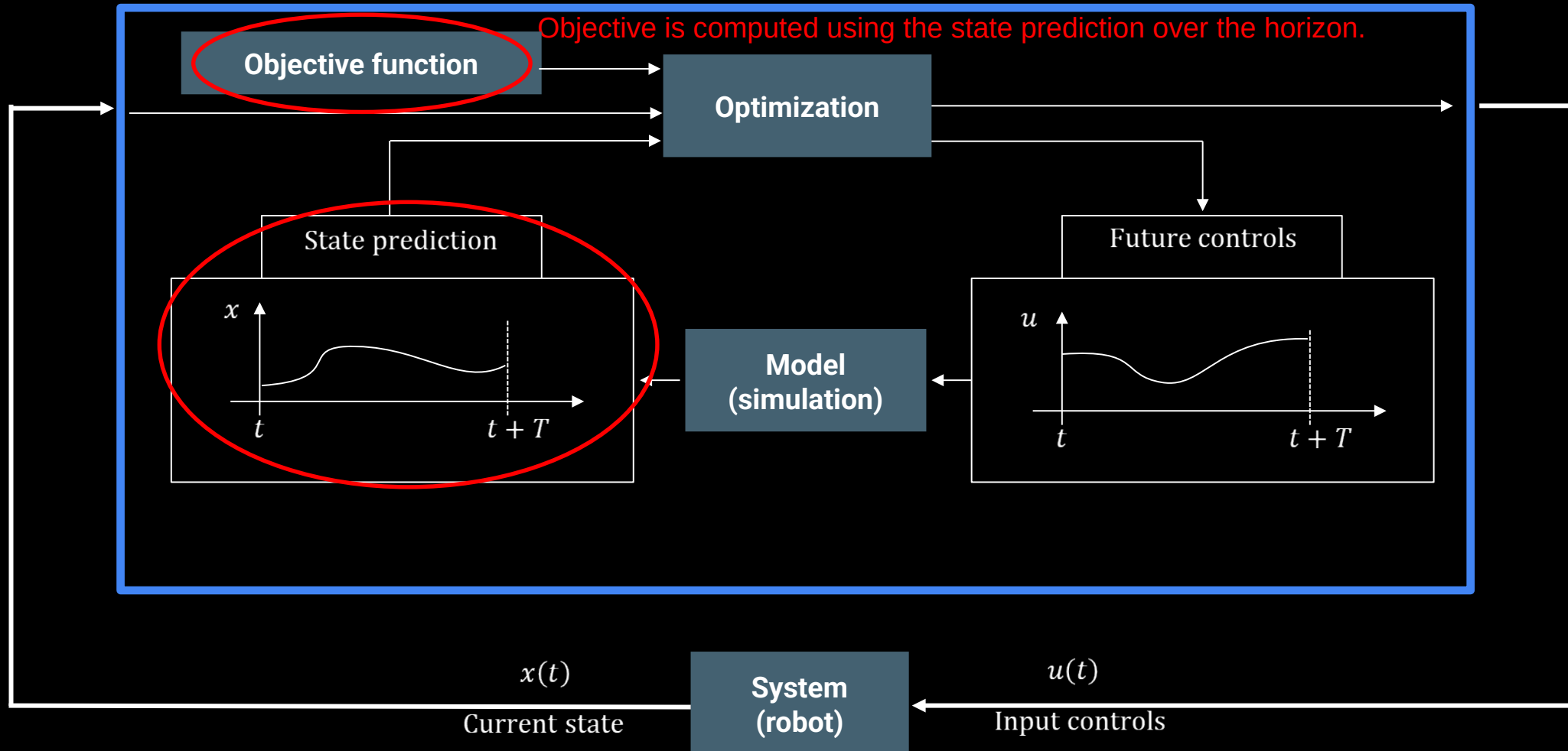


Execute only the “next step”,
not the whole prediction horizon.

Model predictive control



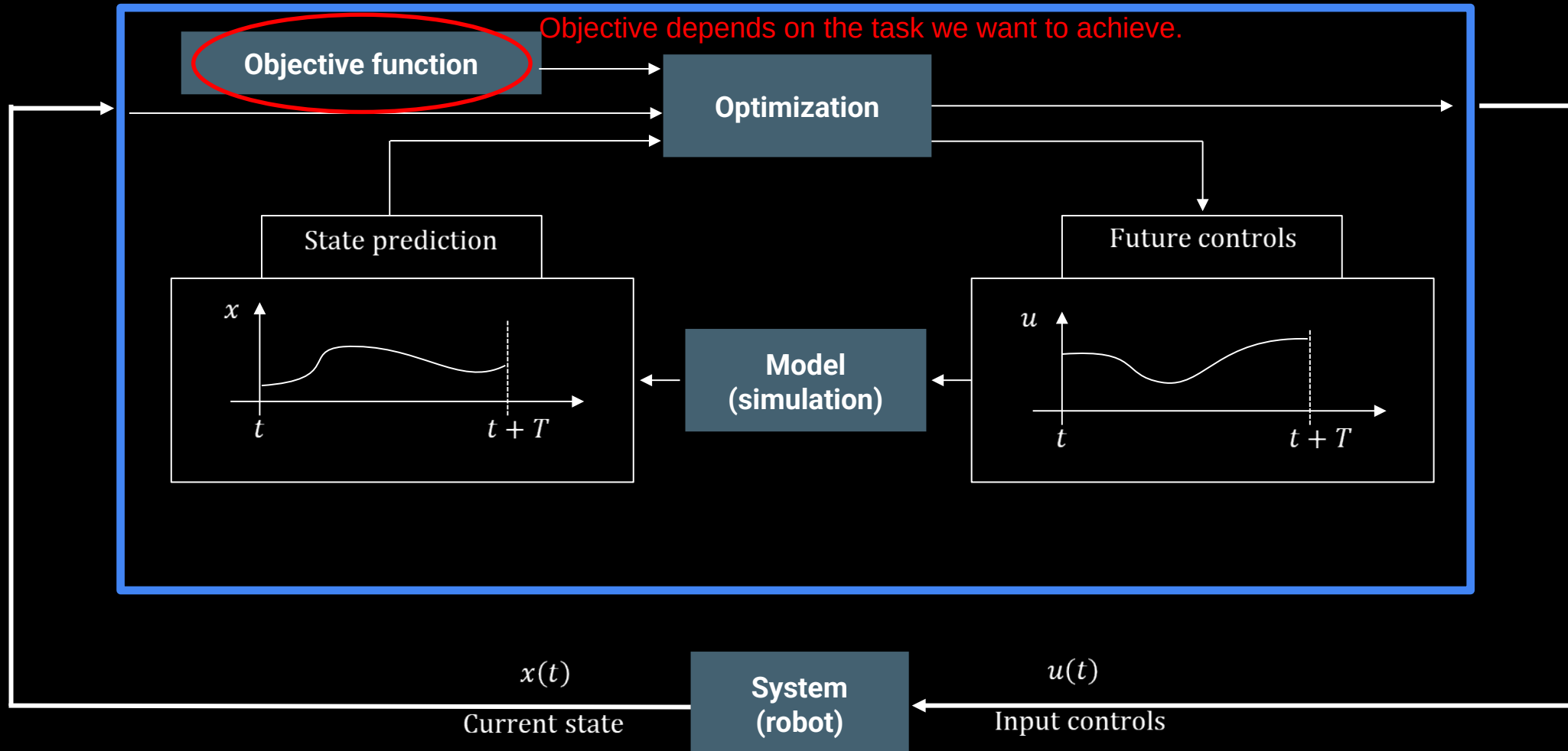
Model predictive controller



Model predictive control

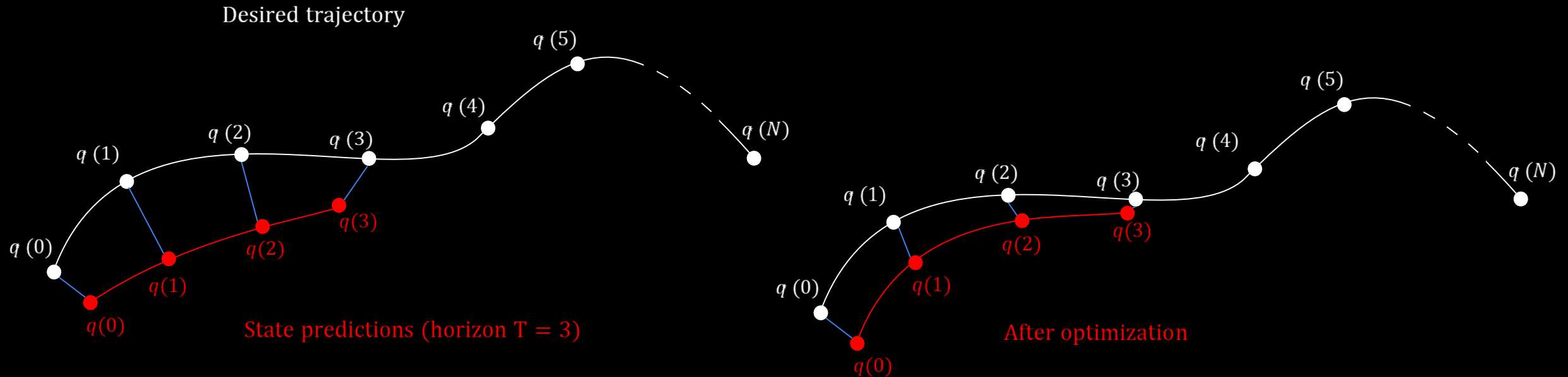


Model predictive controller





Trajectory following with MPC



$$\text{Objective } J = \sum_{t=0}^T c(t)$$

where each step cost $c(t) = \|q^*(t) - q(t)\|_2$

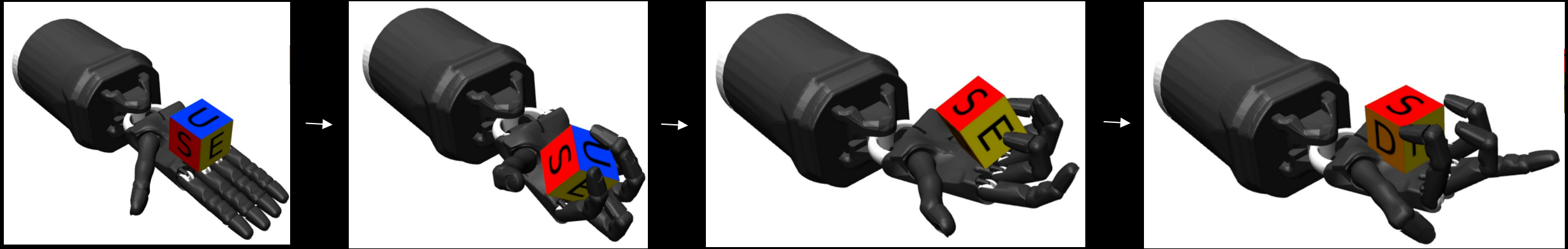
Cube reorientation with MPC



Goal orientation:



Howell et al. 2022, "Predictive Sampling: Real-time Behaviour Synthesis with MuJoCo"



System state $x(t)$ includes robot state $q(t)$, but also the object state.

$$\text{Objective } J = \sum_{t=0}^T c(t)$$

where $c(t) = ||\text{cube orientation}(t) - \text{goal orientation}||_2 + ||\text{cube position}(t) - \text{palm center}||_2$



Reinforcement Learning (RL) – Overview

Idea: Learn a policy (state $\rightarrow$ action mapping) by maximizing long-term reward through interaction.

- **Key characteristics:**

- No explicit model required (model-free RL).
- Works with nonlinear, high-dimensional dynamics.

- **Strengths:**

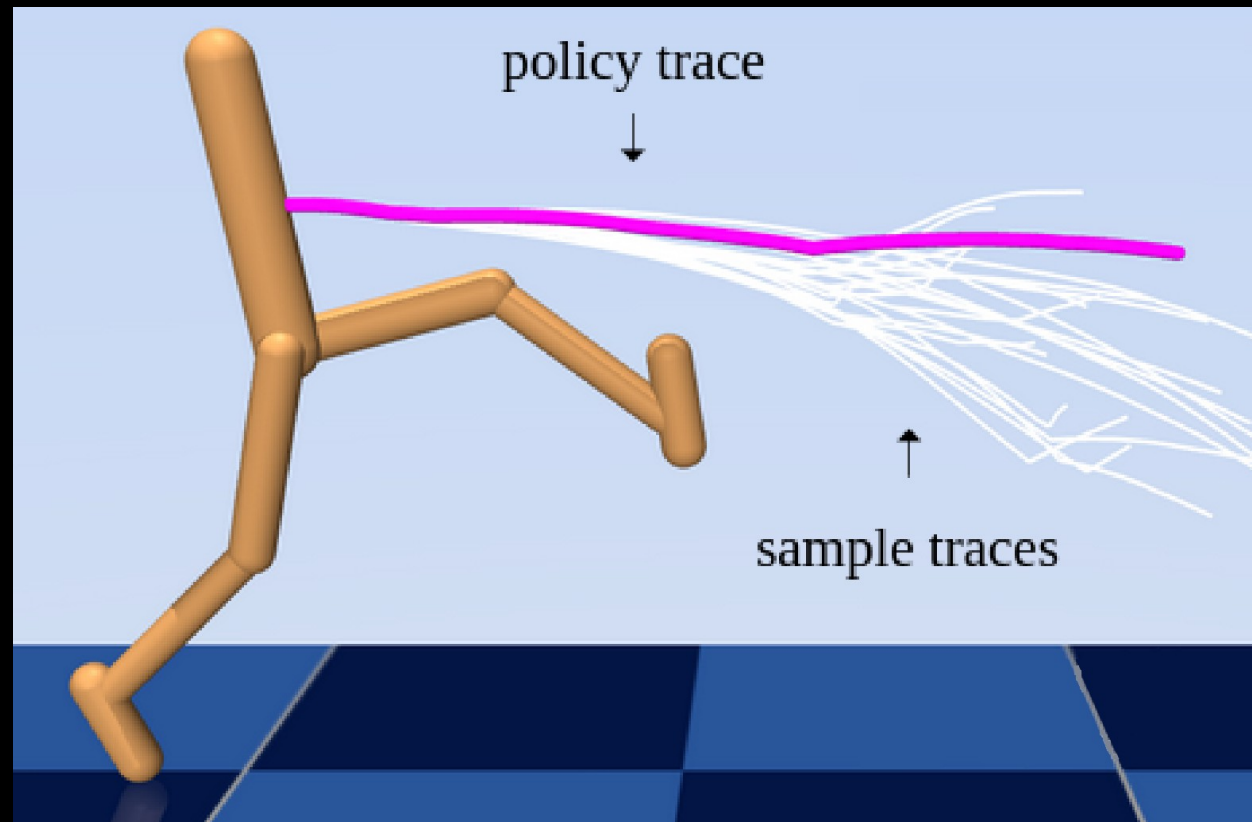
- Captures “intelligent” behaviors.
- Generalizes beyond what is explicitly modeled.

- **Weaknesses:**

- Data hungry $\rightarrow$ needs simulation or many real-world trials (hard and expensive).
- Transfer from sim to real can be hard (sim-to-real gap).

RL will be covered in two weeks.

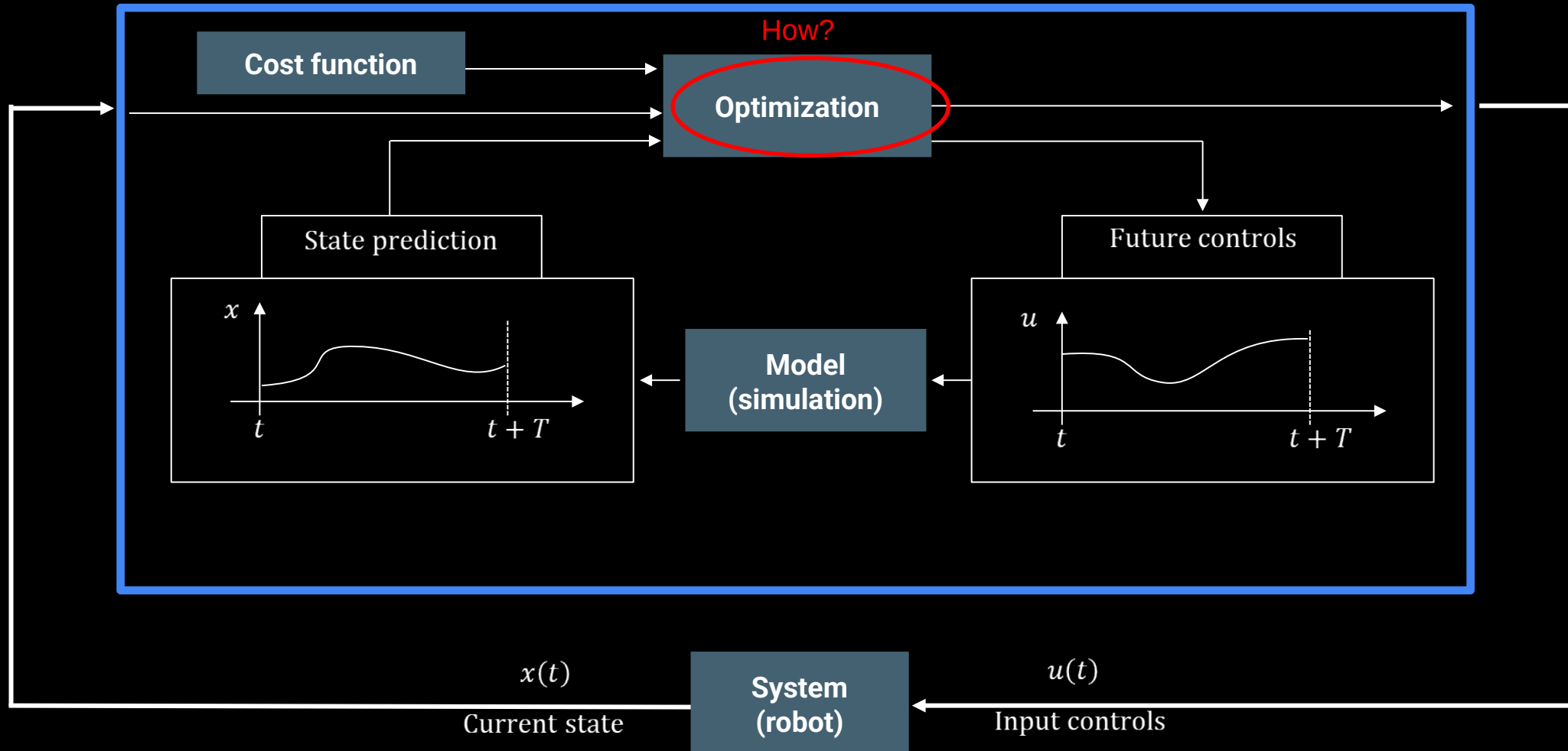
Mujoco Predictive Control (MJPC) Demo



Model predictive control



Model predictive controller



Trajectory optimization



Given current state $x(0)$

$$\min_{u(0), u(1), \dots, u(T-1)} J$$

such that $x(t+1) = f(x(t), u(t))$



LQR - The easy case

Given current state $x(0)$

$$\min_{u(0), u(1), \dots, u(T-1)} J$$

such that $x(t+1) = f(x(t), u(t))$

If the dynamics, f , is linear, i.e., can be written as $x(t+1) = A x(t) + B u(t)$

And if the objective is quadratic, i.e., $J = \sum c(t)$ and $c(t) = x(t)^T Q x(t) + u(t)^T R u(t)$ where Q and R are symmetric positive semi-definite matrices.

Then this is called a **linear-quadratic regulator (LQR)** problem, and the optimal solution can be written as a feedback control law.



Trajectory optimization with non-linear dynamics

Given current state $x(0)$

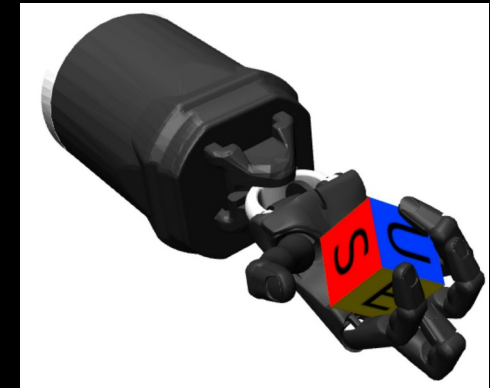
$$\min_{u(0), u(1), \dots, u(T-1)} J$$

such that $x(t+1) = f(x(t), u(t))$

For robotics tasks, the system dynamics, f , is often non-linear.

Trajectory optimization with non-linear dynamics, particularly applied to robotics, is an active research area.

Highly non-linear!





Trajectory optimizers in MJPC

- **Derivative-based methods**

iLQR: Makes local LQR approximation to the non-linear problem, solves the LQR problem, updates the trajectory, and iterates (i.e., makes a local LQR approximation again) until convergence.

Requires computing first and second-order derivatives, e.g., $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial u}, \frac{\partial c}{\partial x}, \frac{\partial c}{\partial u}, \frac{\partial^2 c}{\partial x^2}, \frac{\partial^2 c}{\partial u^2}, \frac{\partial^2 c}{\partial x \partial u}$

Gradient descent: Computes the gradient of the objective with respect to the controls, $\frac{\partial J}{\partial u}$ and takes a step in that direction.

Analytical derivatives are often not available. (f is the simulator!)

Derivatives are usually computed by “finite-differencing”, which requires stepping the simulator at slightly different initial conditions, and therefore computationally expensive.