

# Object Pose Estimation for Real-World Policies

**Max Firkowski**

Teaching Assistant, Real World Robotics

26.10.2026, Zürich

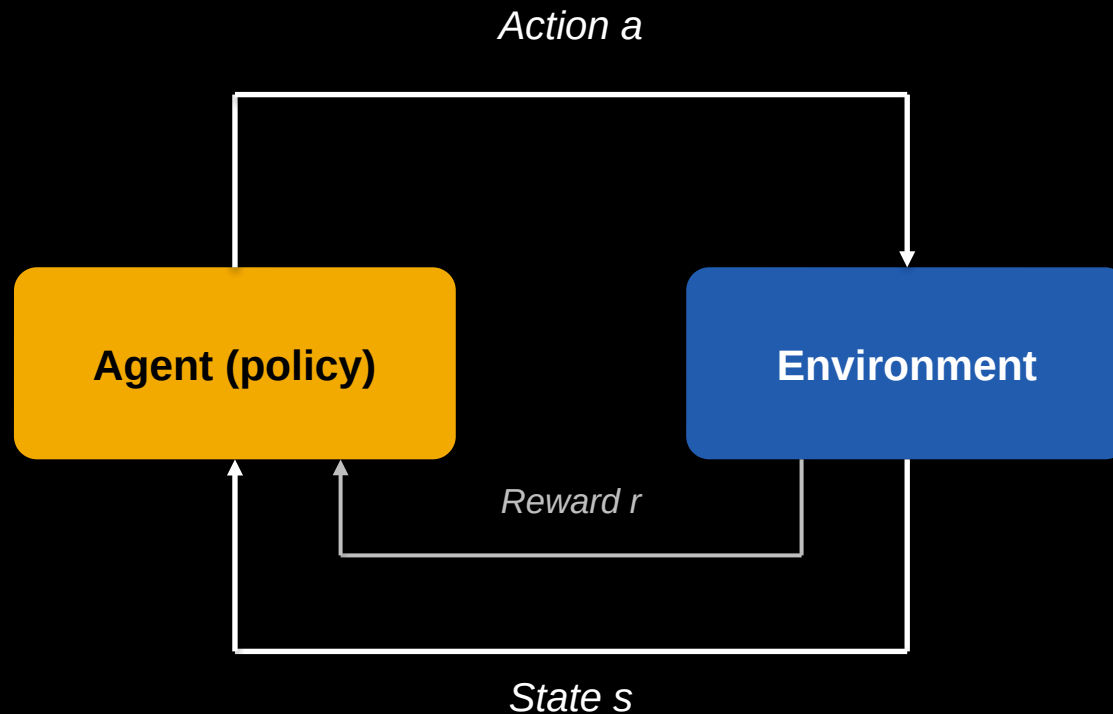


# Agenda

1. Why the policy needs the cube pose
2. Three ways to estimate a 6D pose: direct, marker-based, keypoints + PnP
3. Keypoint detection, camera calibration and PnP
4. Our cube tracker and the frames in Task 1

# From states to actions

Motivation | Methods | Task 1 setup



For robotic manipulation, the state has two parts:

## Manipulator state – “proprioception”

- What the robot knows about itself
- Task 1: 17 joint positions and velocities

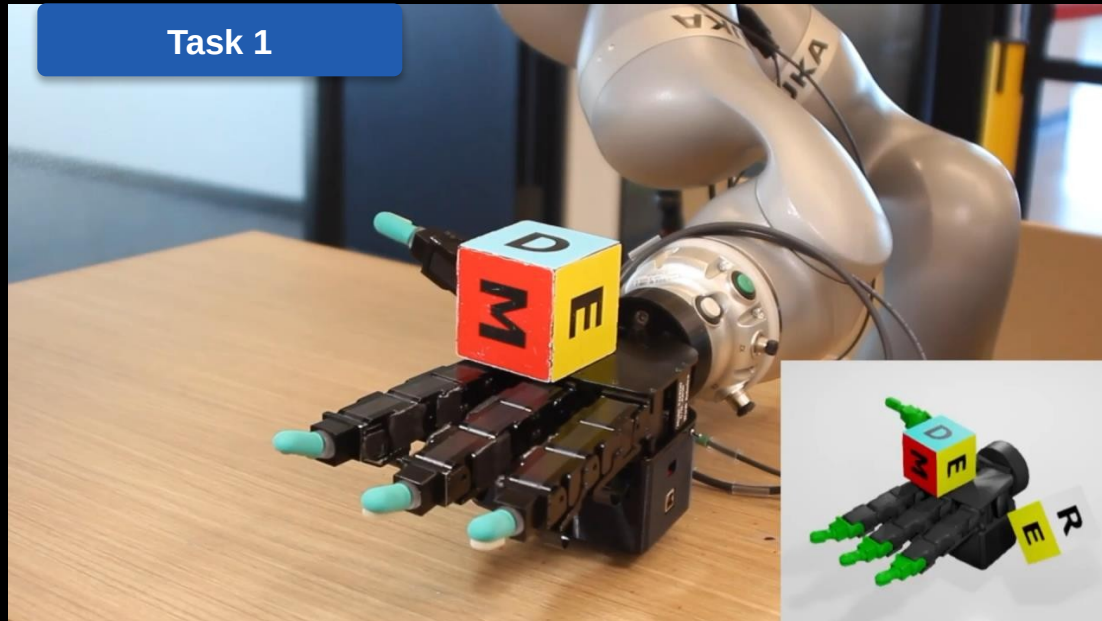
## Environment state – “perception”

- What the robot knows about the world
- Task 1: position and orientation of the cube

Subtask 2 adds the cube pose → today's topic

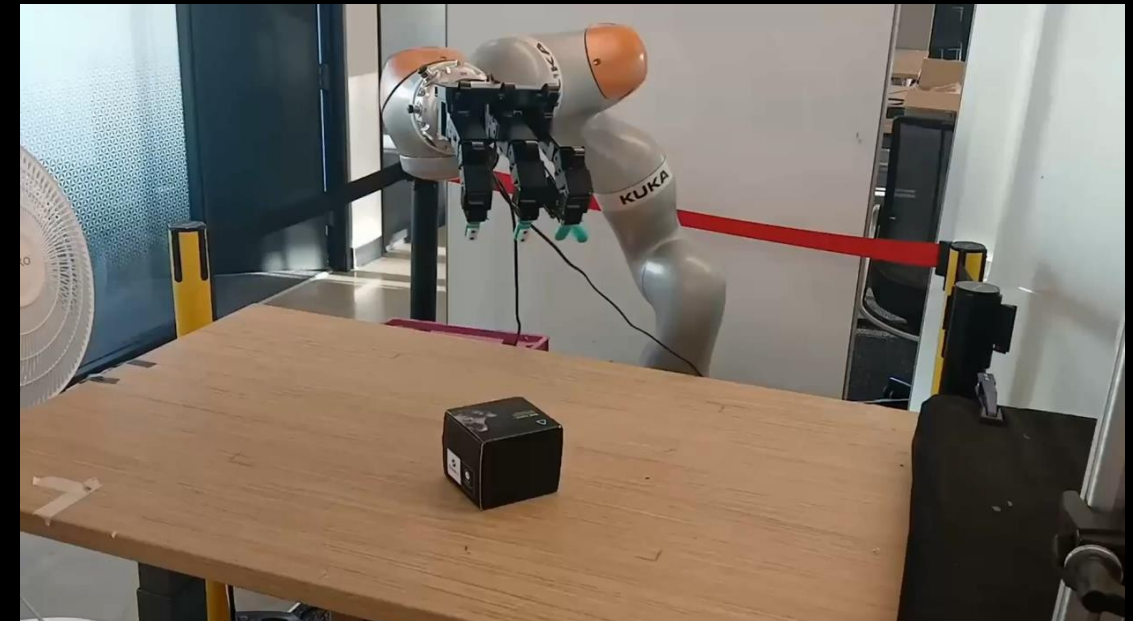
# Explicit vs implicit perception

Motivation | Methods | Task 1 setup



**Explicit: object pose + proprioception → actions**

- + Easier to train: low-dimensional input, clear reward and goal
- Needs a pose estimator that works in the real world



**Implicit: RGB image + proprioception → actions**

- + Deploy directly from the camera images
- Very hard to train: high-dimensional input, the network must learn its own features
- Big sim-to-real gap: simulated images are not photorealistic

# 6D pose estimation – direct methods

Motivation | Methods | Task 1 setup



FoundationPose (Wen et al., CVPR 2024): input image and estimated object poses

RGB(-D) (+ CAD)  $\rightarrow$  network  $\rightarrow$   $[R \mid t] \in SE(3)$

6D pose = 3D rotation  $R$  + 3D translation  $t$

Examples: PoseCNN (2018), FoundationPose (2024)

## How

- CNN or transformer backbones
- Regress rotation (quaternion or 6D representation) and translation, or refine the pose by render-and-compare
- Pretrain on synthetic data, adapt to the real domain

## Pros

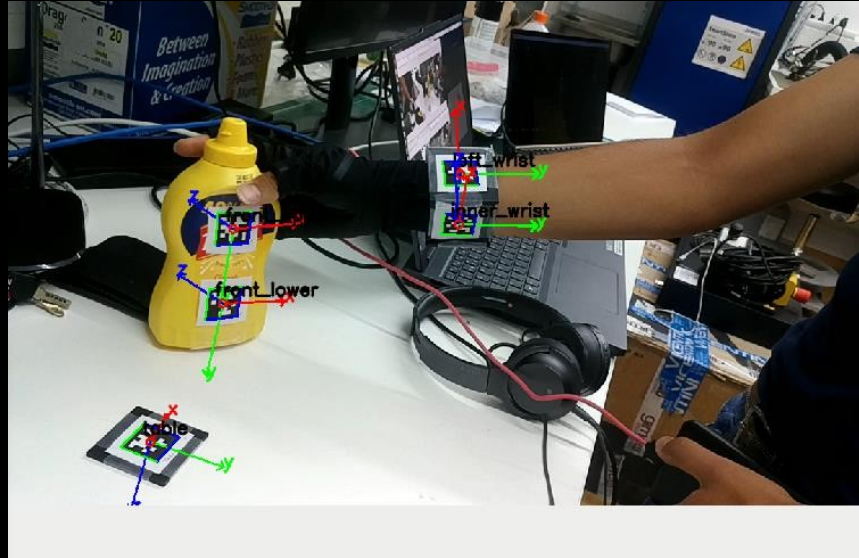
- + Can generalize to new objects (with enough training)

## Cons

- Large data requirements
- Errors are hard to debug or interpret
- Can fail with occlusion or unseen viewpoints
- Heavy at runtime

# 6D pose estimation – marker-based methods

Motivation | Methods | Task 1 setup



Use a known printed pattern (AprilTag, ArUco) with an exact 3D geometry

## Steps

1. Detect the 4 tag corners in the 2D image
2. Solve PnP with the known 3D corner positions
3. Known tag pattern → 6D pose of the tag

## Pros

- + High accuracy, low computational cost
- + Very good for camera calibration and ground truth

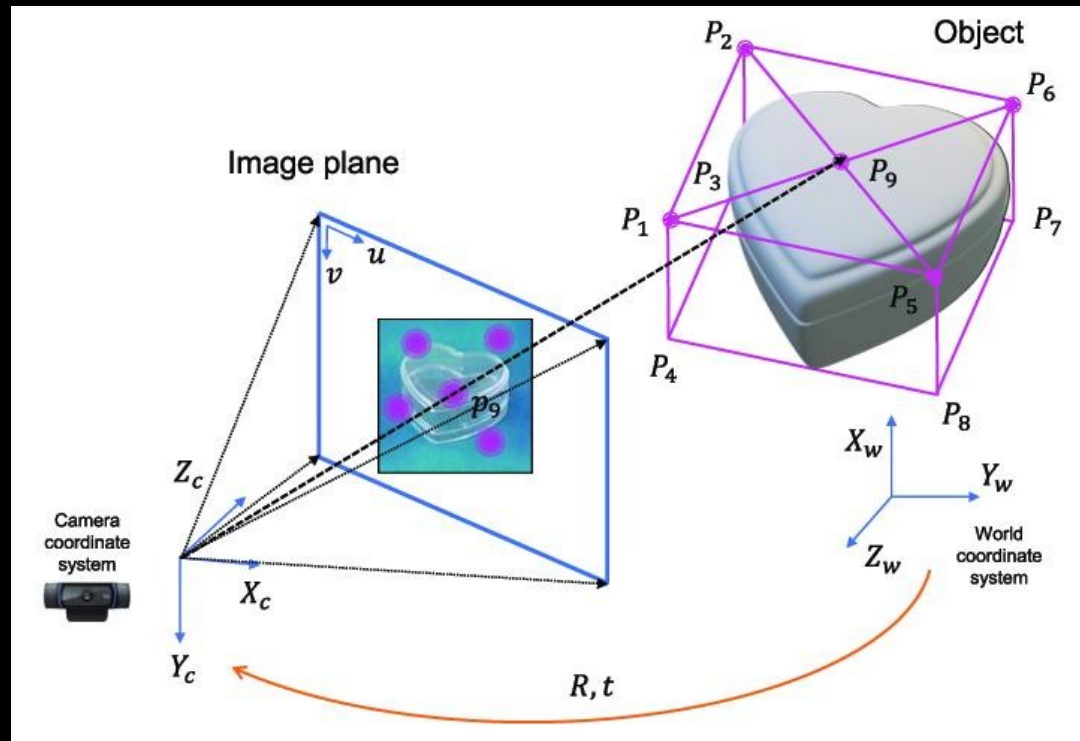
## Cons

- The tag must be visible – not for real objects
- Lighting and motion blur

Task 1: AprilTag 36h11, ID 25, 8 cm → camera calibration

# 6D pose estimation – indirect (keypoint) methods

Motivation | Methods | Task 1 setup



## Two steps

1. A CNN detects 2D image keypoints that belong to known 3D model points
2. PnP (Perspective-n-Point) solves for the object pose in the camera frame

## Pros

- + Strong geometric constraints, easy to interpret
- + Easy to integrate with a known CAD model
- + Works with relatively small datasets

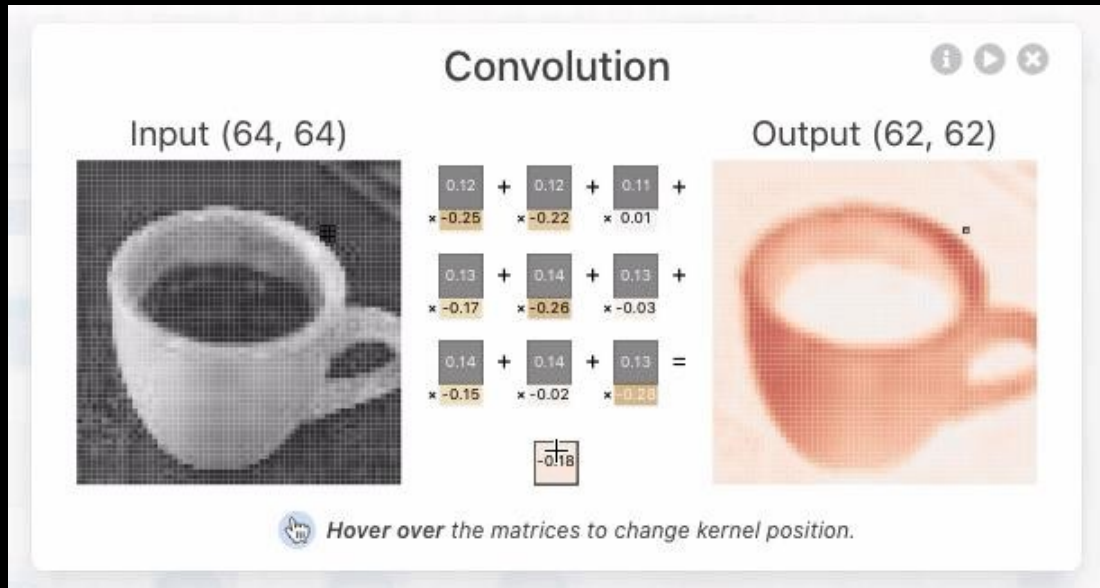
## Cons

- Sensitive to keypoint noise
- Needs a known 3D model
- Symmetric objects are ambiguous (a cube!)

Task 1: our cube tracker = 8 corner keypoints + PnP

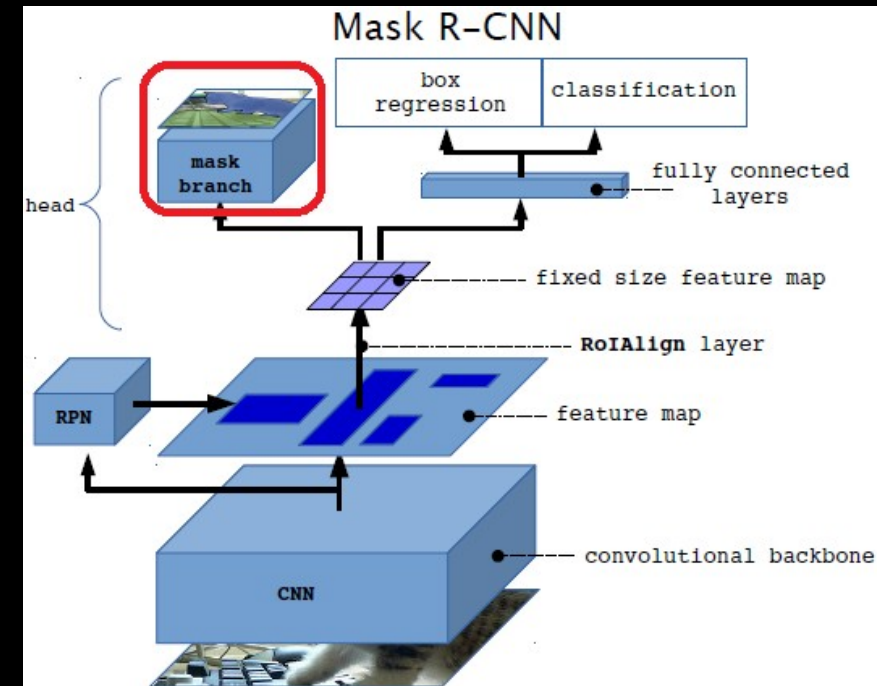
# 2D keypoint detection with CNNs

Motivation | Methods | Task 1 setup



**Goal: find the 2D image positions of known 3D object points**

- Convolutions detect local patterns (edges, corners, texture) anywhere in the image
- Heatmap regression: one probability map per keypoint → take the peak (argmax)

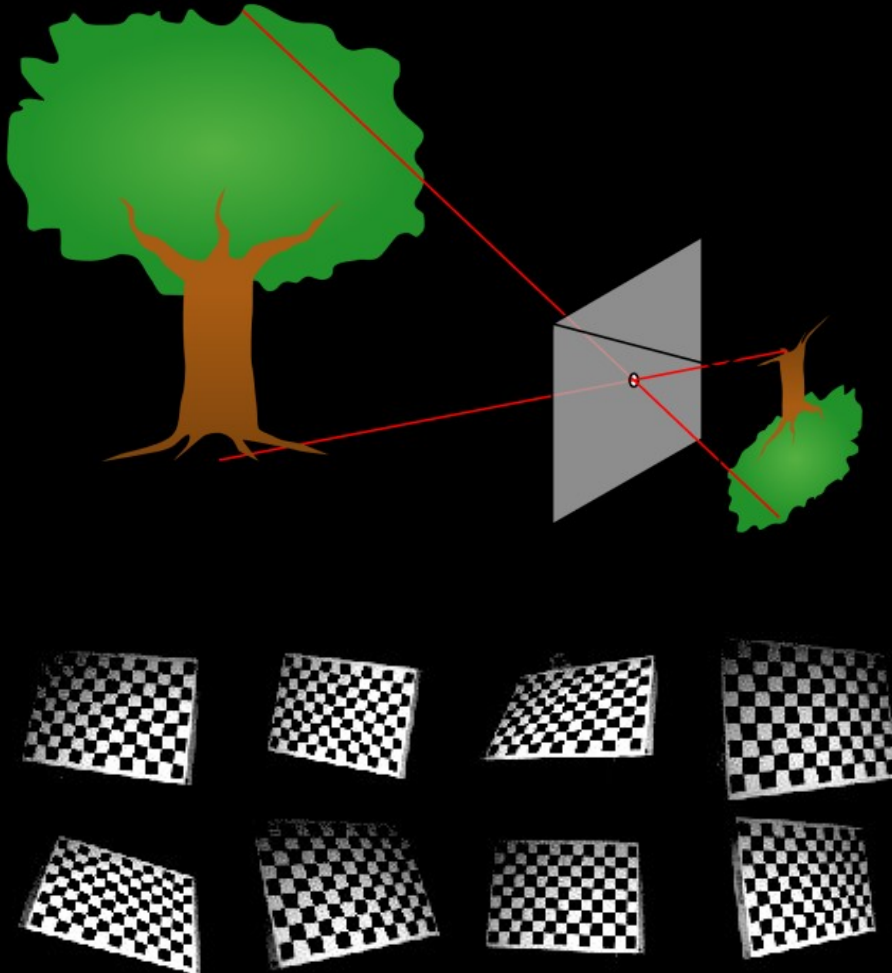


**Mask R-CNN (He et al., ICCV 2017)**

- Detect the object: box + mask
- Keypoint head: one heatmap per keypoint inside the box
- Ours: 8 cube-corner keypoints

# Camera calibration

Motivation | Methods | Task 1 setup



## Pinhole camera model

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{K \text{ (intrinsic)}} \underbrace{\begin{bmatrix} R & t \end{bmatrix}}_{\text{extrinsics}} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

- Intrinsic K: calibrate with a checkerboard
  - Task 1: the OAK-D comes factory-calibrated
- Extrinsic (world → camera): use a marker – **and invert the transform!**

# PnP for pose estimation

Motivation | Methods | Task 1 setup

**Perspective-n-Point:** estimate the pose of a calibrated camera from n known 3D points and their 2D projections. We use it the other way round: the pose of the cube in the camera frame.

$$\lambda p_c = K \begin{bmatrix} R & | & t \end{bmatrix} p_w \quad K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

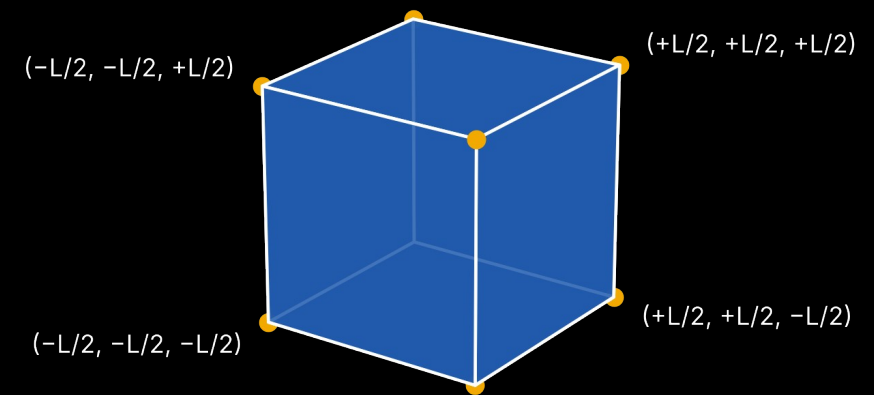
$p_c$  2D keypoints from the CNN (pixels) **known**

$p_w$  3D cube corners ( $\pm L/2, \pm L/2, \pm L/2$ ) **known**

$K$  intrinsics from the camera calibration **known**

$R, t$  cube pose in the camera frame **unknown**

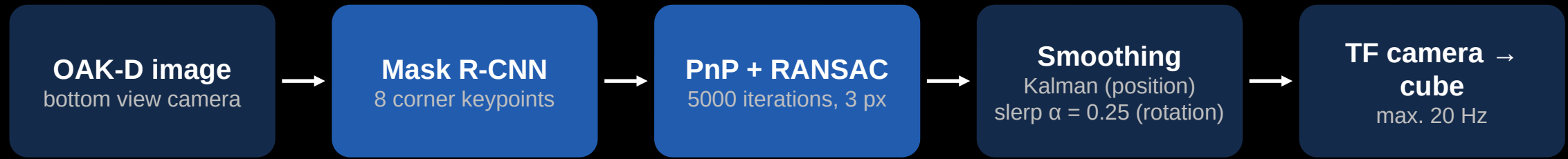
- At least 4 points for a unique pose (3 points: up to 4 solutions); coplanar points are fine
- RANSAC rejects wrong keypoints; the solver minimizes the reprojection error



**L = 45 mm, 8 corner keypoints**

# Our cube tracker

Motivation | Methods | Task 1 setup



## Detector

- Mask R-CNN, ResNet-50-FPN backbone
- Starts from a CosyPose detector (YCB-Video)
- Fine-tuned on synthetic images rendered in faive\_lab
- Input image  $240 \times 320$

## Geometry

- Cube edge length 45 mm
- Intrinsic K from the OAK-D
- RANSAC: 3 px threshold, at least 3 inliers
- Smoothing removes noise but adds lag

## Sim-to-real for Subtask 2

- Add tracker noise to the simulated cube observation
- Model the lag (camera + smoothing)
- Decide what the policy gets when the cube is lost
- Symmetry: the orientation can jump

# Questions?

image → keypoints (CNN) → pose (PnP + RANSAC) → smoothing → frame transforms → policy



# System Identification

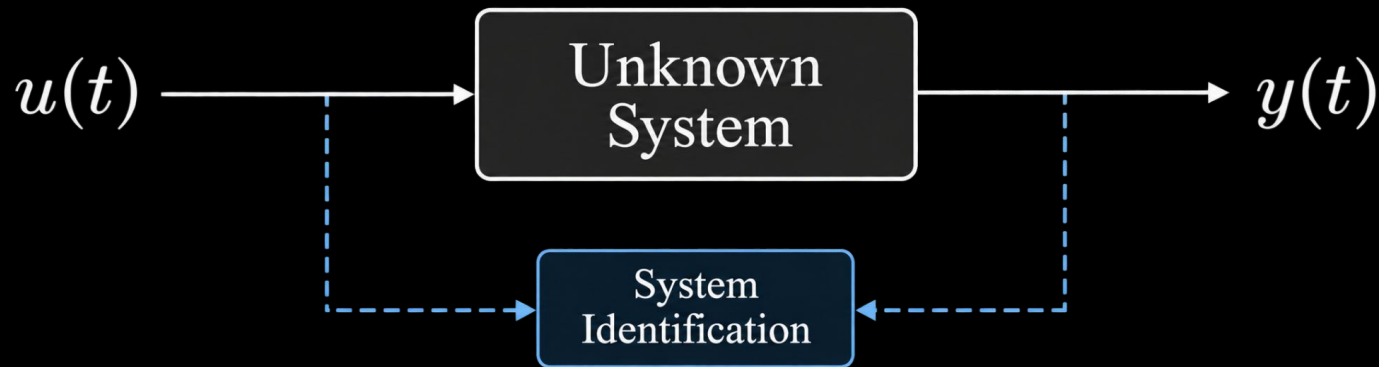
Yucheng Xia



# Why do we need System Identification (SysID) ?



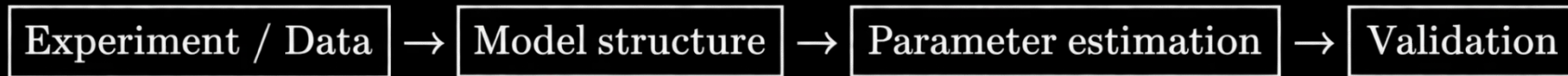
$$x_{t+1} = f(x_t, u_t)$$



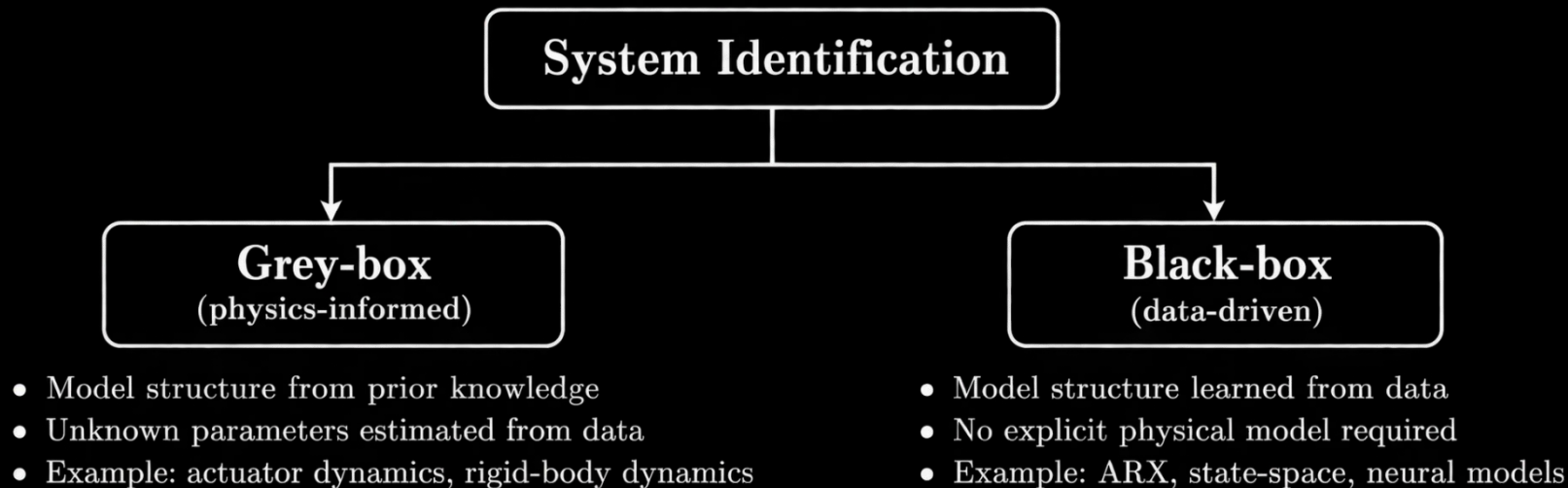
$$\hat{y}(t) = \hat{f}(u(t), \text{history}; \theta)$$

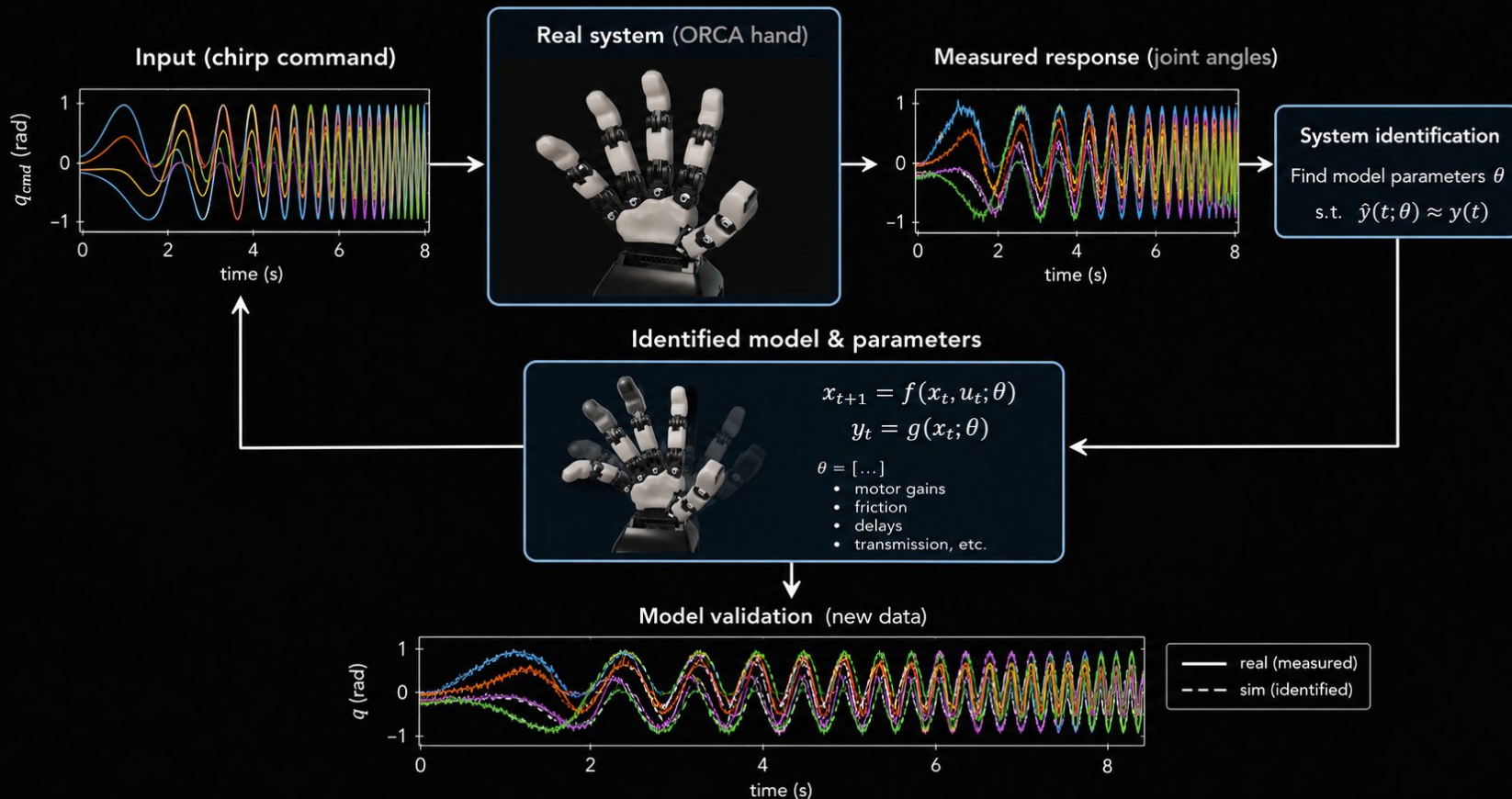
$$\hat{y}(t) \approx y(t)$$

# Principles of System Identification



# Two different paradigms of SysID





# Preliminaries for Orca Hand v2 SysID Practice (following slides)



- **Bayesian Optimization**

Uses past observations to balance between exploring and exploiting, seeking the best parameters with fewer experiments.

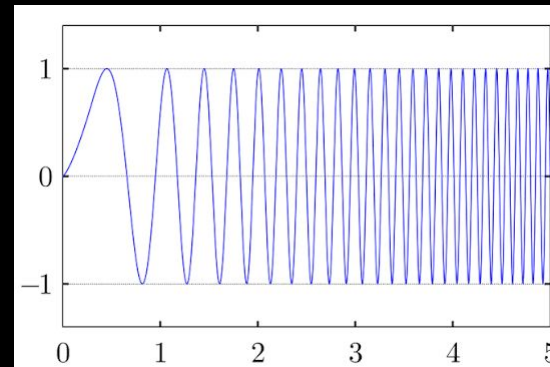
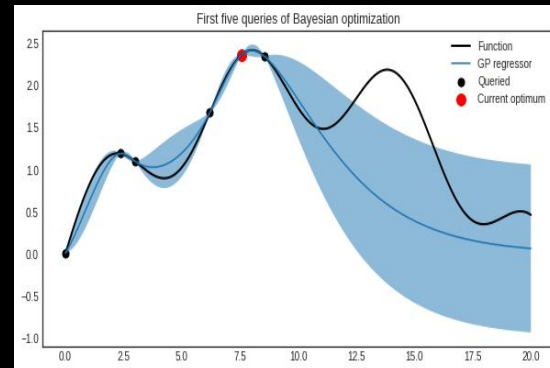
- **Transfer Function**

$$G(s) = G_2(s) \underbrace{\frac{1 + s/\omega_z}{1 + s/\omega_p}}_{\text{Additional dynamic correction}}$$

$G_2(s)$ : Original actuator model in Isaac Sim

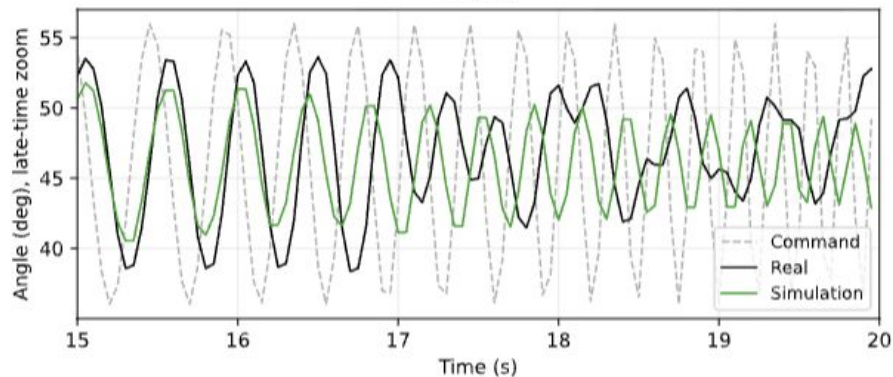
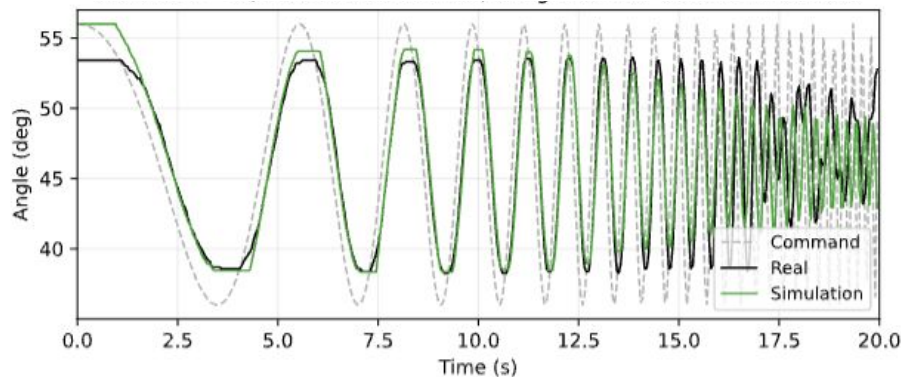
- **Chirp signal**

A wave where the frequency increases (up-chirp) or decreases (down-chirp) continuously over time.

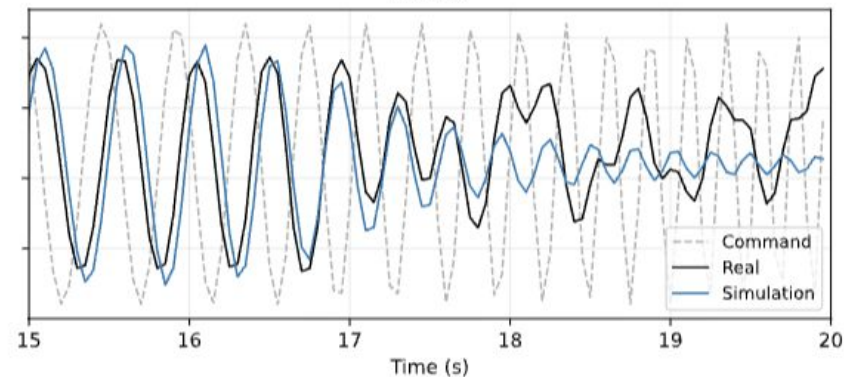
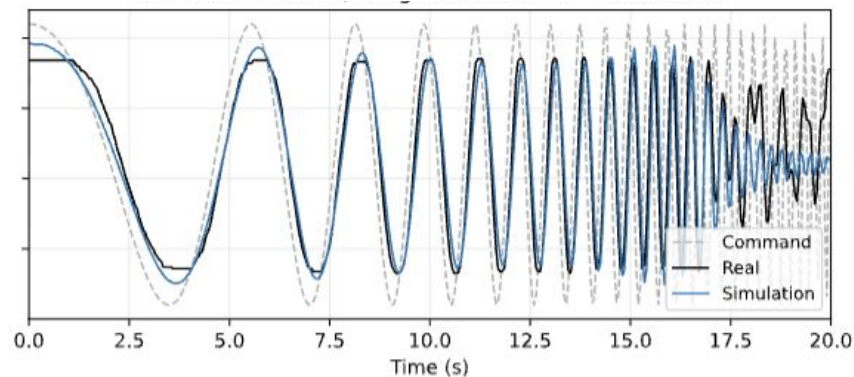


# Compare Model Structure

## Default Actuator

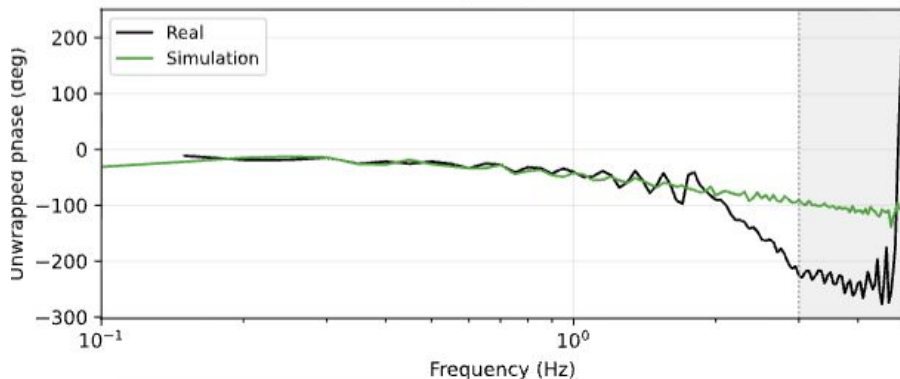
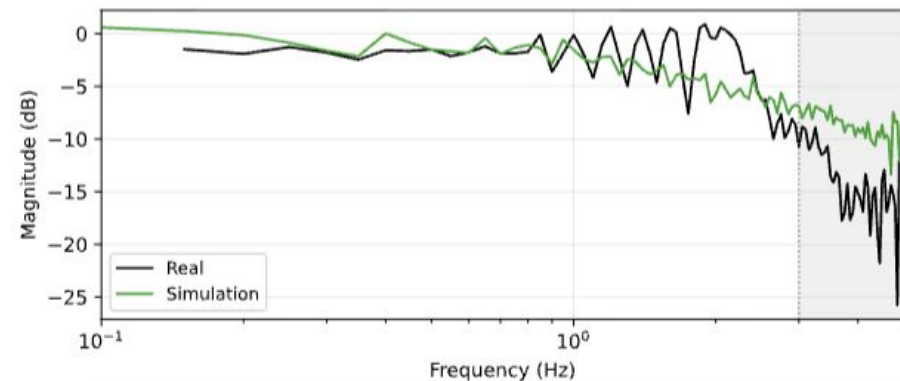


## Updated Actuator

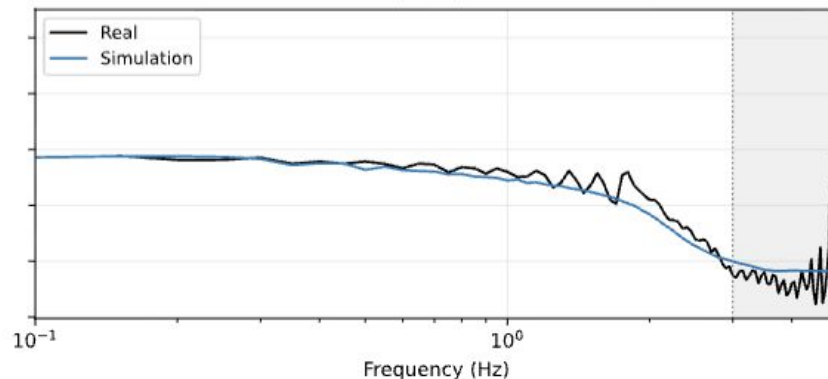
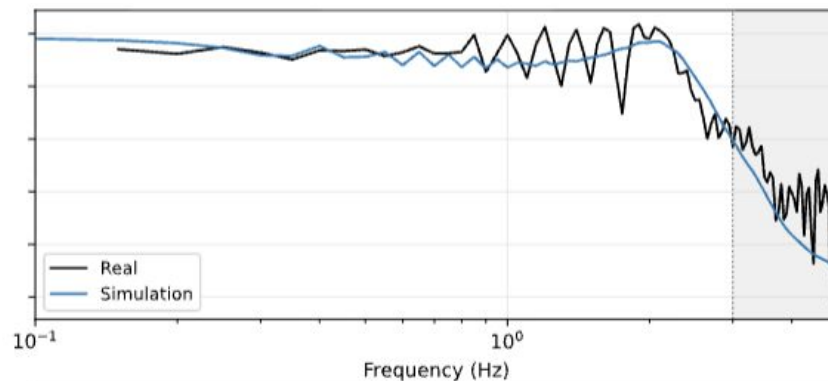


# Compare Model Structure

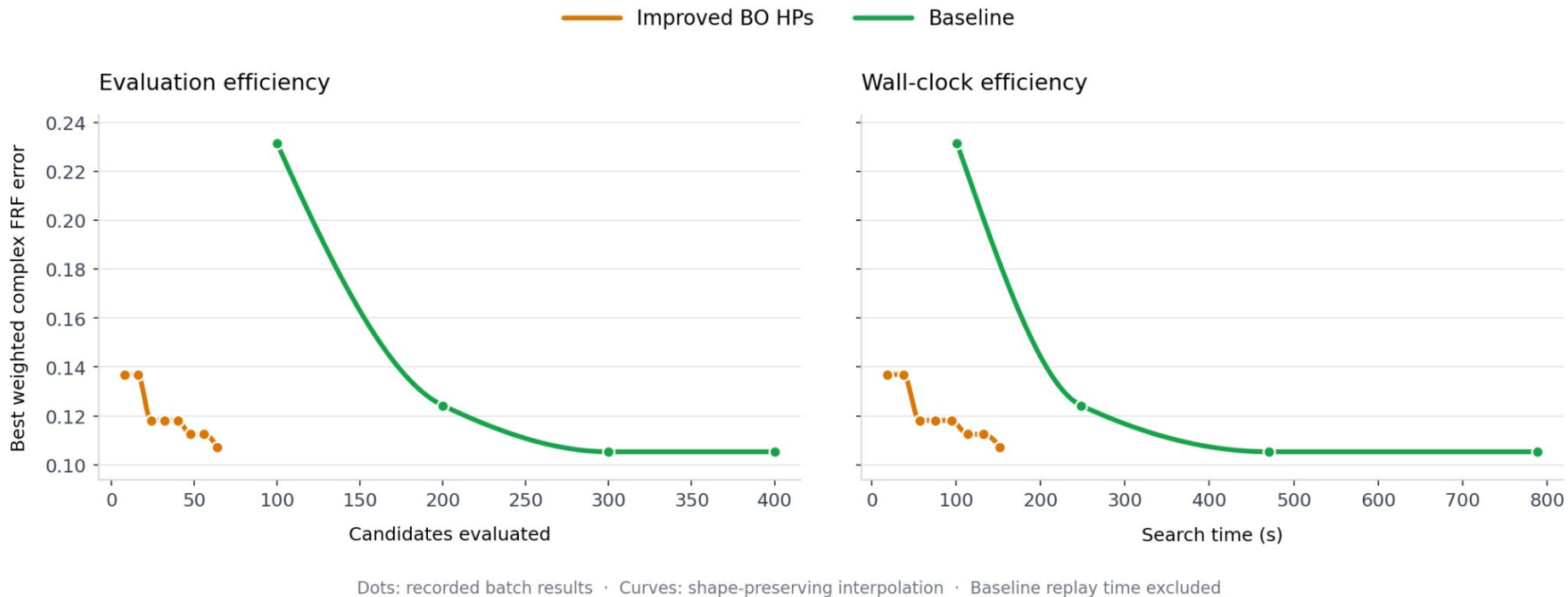
## Default Actuator



## Updated Actuator



# Compare Params Search Method



## Suggestions for Task 1

1. **The default, unchanged params work quite well** in the real world , but might exist more sim2real gap compared to below methods.
2. **Tune the Bayesian Optimization algo** with current actuator model to find the best-fit params to minimize sim2real gap.
3. **Add a zero-pole filter** to the actuator model would result in a more fine-grained fitting of motor characteristics, leading to smaller sim2real gap.