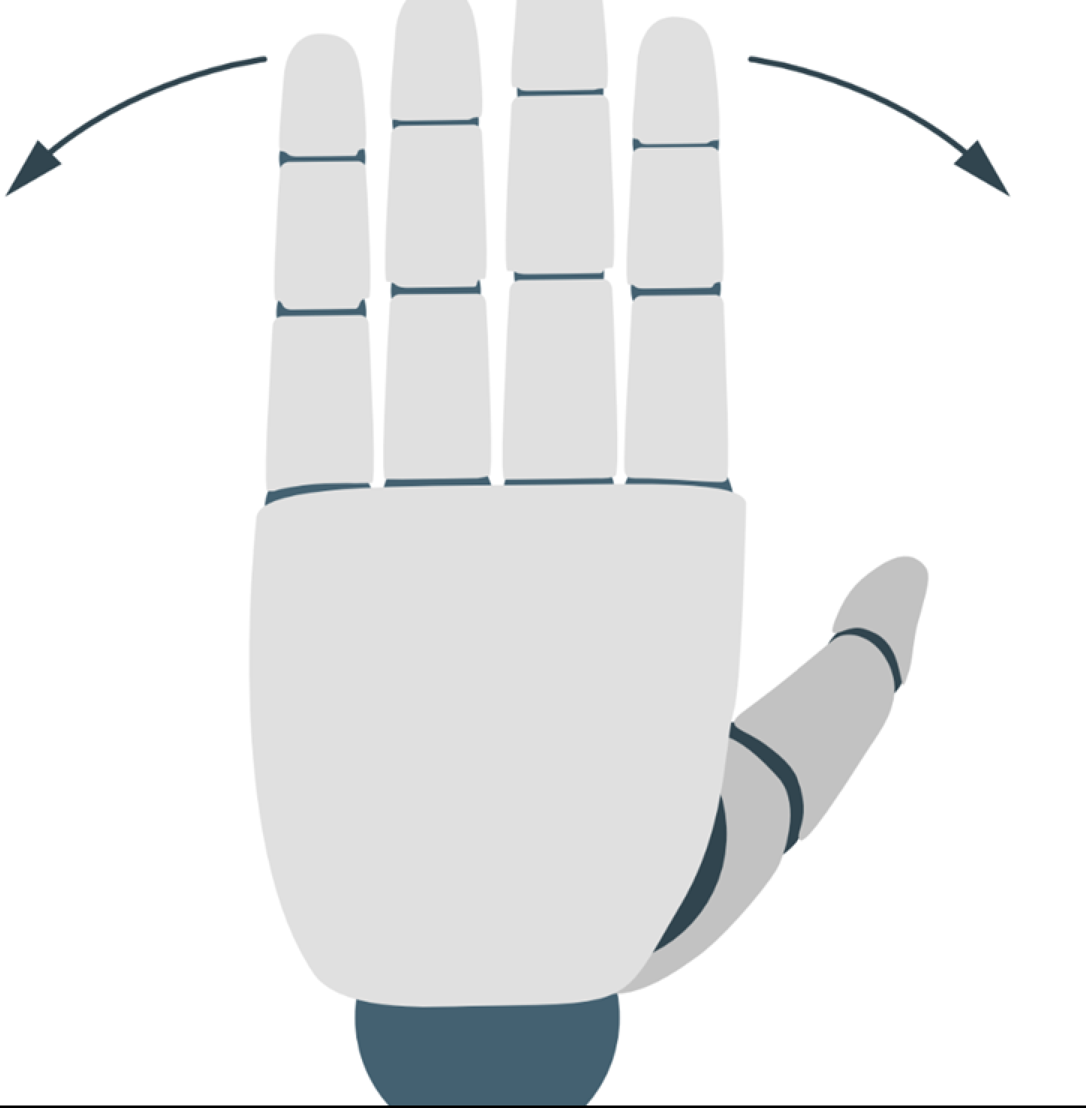
Workshop Unit 7

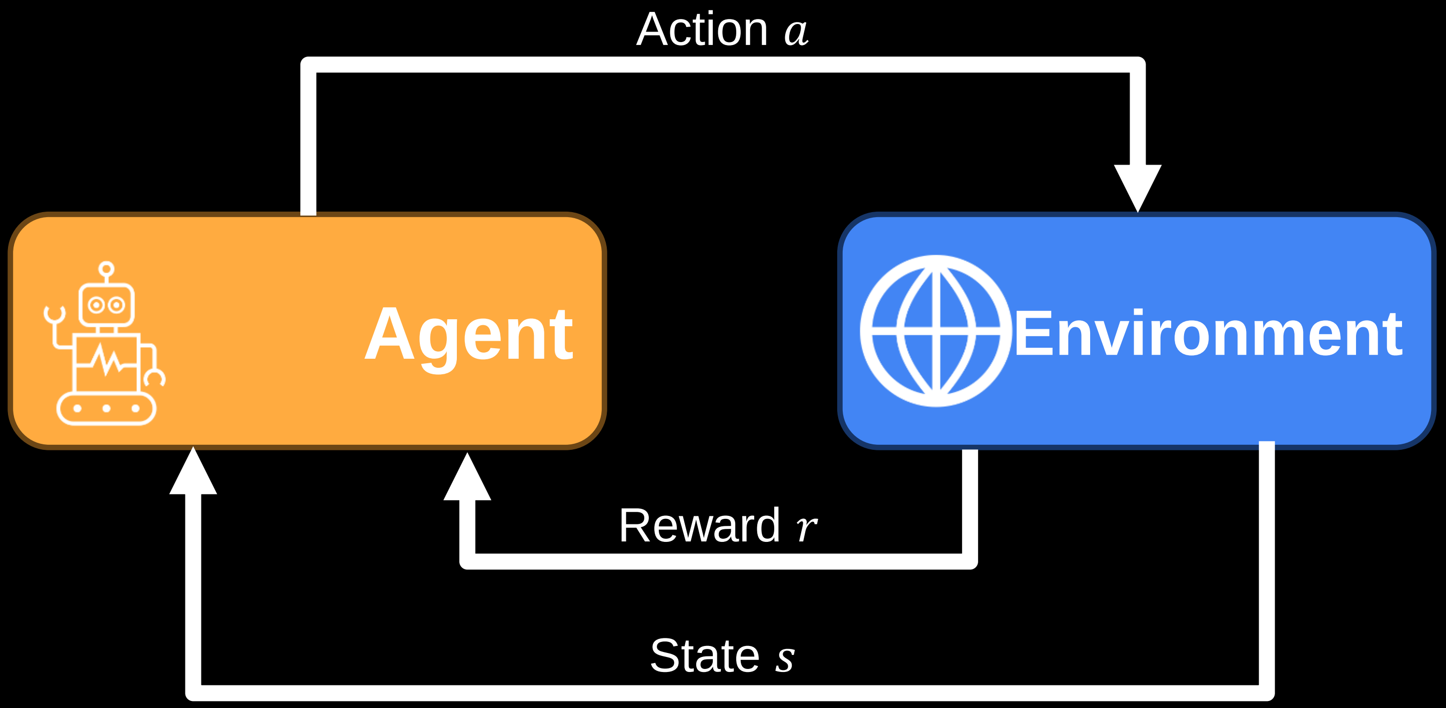
Object Pose Estimation for Real World Policies



Davide Liconti

27 October 2025

States to Actions



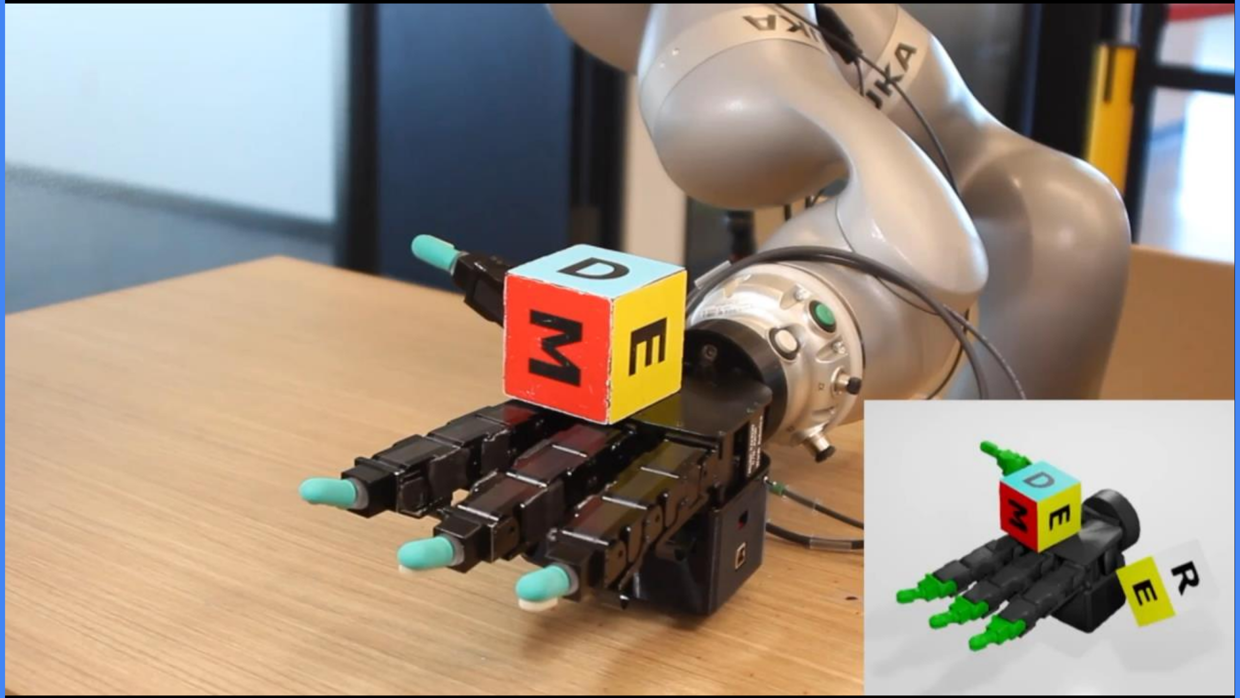
For robotic manipulation:

- Manipulator States "proprioception"
- Environment State "perception"

Explicit vs Implicit Perception

Explicit

Object Pose + Proprioception → Actions

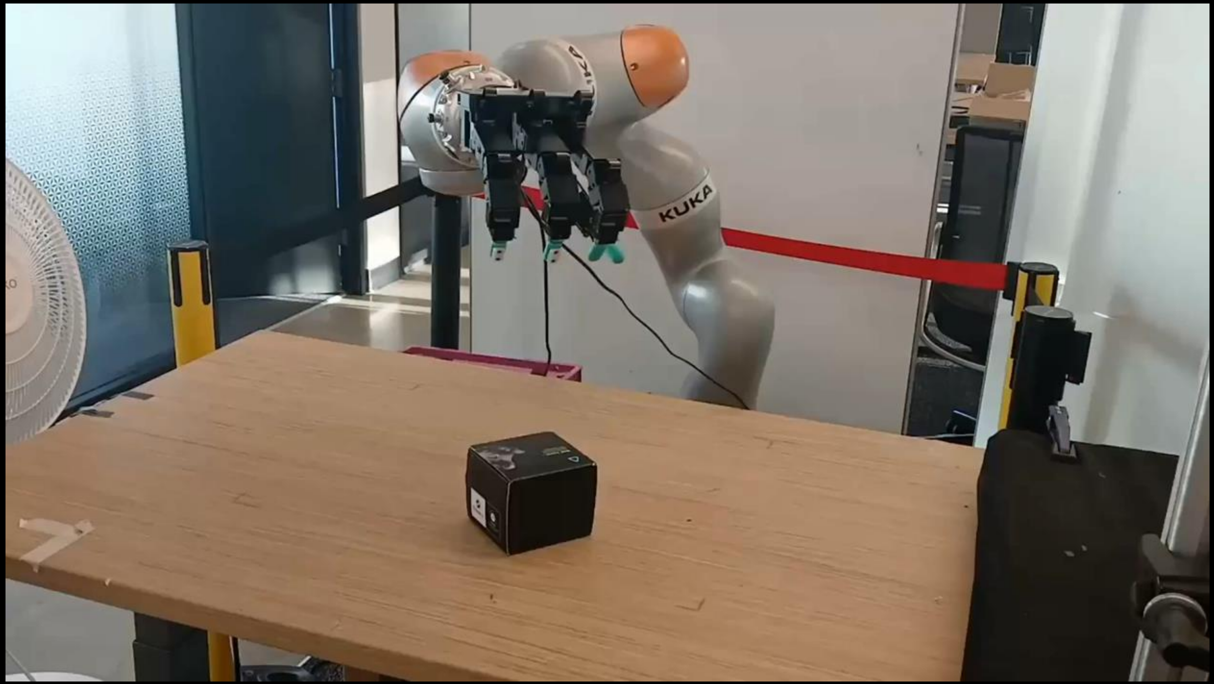


Easier to train
- Lower dim
- Clearly defined reward and goal

Need real world estimation

Implicit

RGB image+ Proprioception → Actions



Can directly deploy from cameras

Very difficult to train

High dimensional input -> need feature extraction

Big sim-to-real gap for not photorealistic simulators

6D Pose Estimation

Direct Methods

RGB (+D) (+CAD)

↓

6D Pose : [R | t] ∈ SE(3).

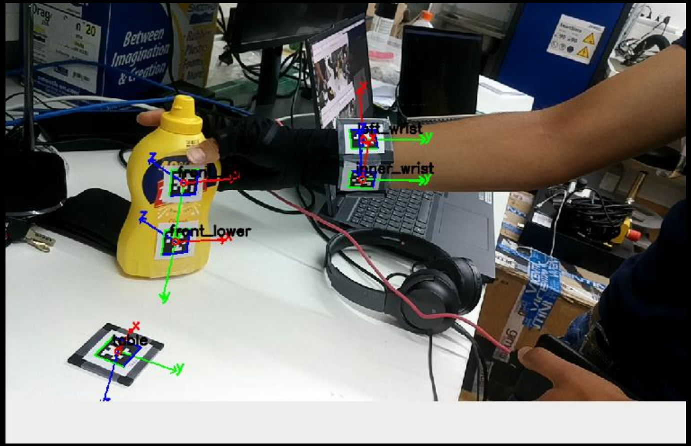
[FoundationPose]

• Often built on **transformers** or **CNN backbones** (e.g., *FoundationPose, PoseCNN*).
• Can regress to rotation (quaternion or 6D representation) and translation vectors.
• May use **synthetic data pretraining** and **domain adaptation** for robustness.

Pros:
✔️ Can generalize across unseen instances (with proper training).
Cons:
⚠️ High data requirements.
⚠️ Harder to debug or interpret errors.
⚠️ May fail on occlusion or unseen viewpoints.
⚠️ Computationally heavier at runtime.

6D Pose Estimation

Marker-based Methods



•Use **known patterns** (AprilTags, ArUco) with precisely defined 3D geometry.
•Steps:

- Detect corner points in 2D image (from tag pattern)
- Compute pose using PnP with tag's known 3D coordinates.
- Known tag pattern -> 6D pose

Pros:
✓ High accuracy and low computational cost.
✓ Great for camera calibration and ground truth validation.
Cons:
⚠️ Requires visible tags (not for real objects)
⚠️ Lighting and motion blur

6D Pose Estimation

Indirect Methods

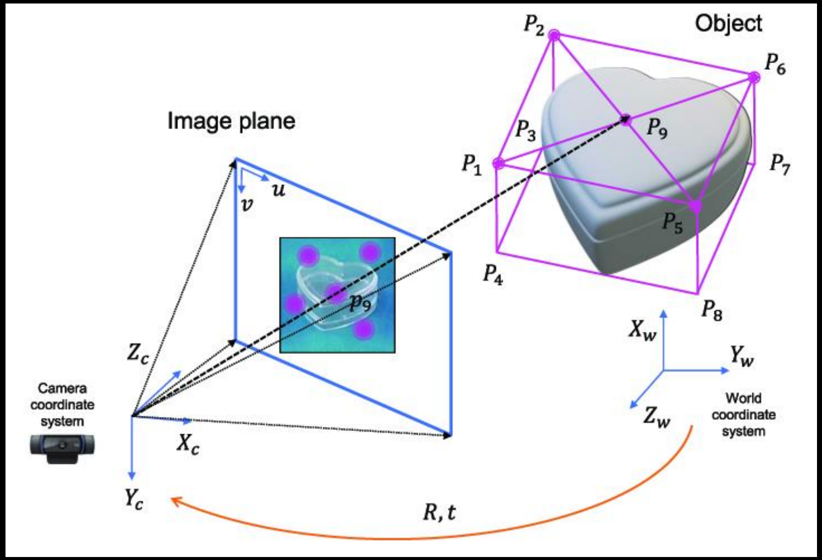
RGB (+D) + model

↓ *CNN*

2D keypoints

↓ *PnP*

6D Pose : [R | t] ∈ SE(3).



1. Detect **2D image keypoints** corresponding to **known** 3D model points.
2. Step 2: Use **PnP (Perspective-n-Point)** to solve for camera–object pose

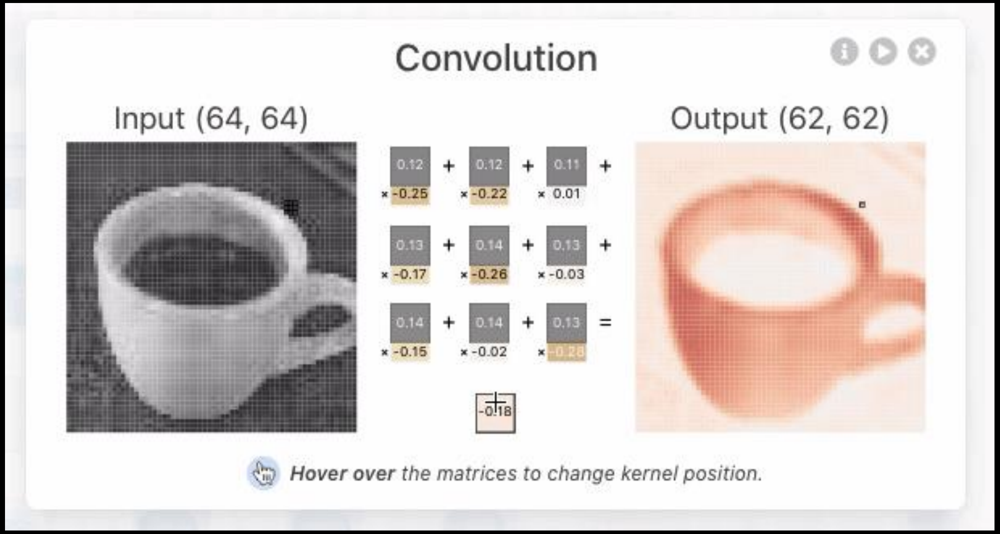
Pros:

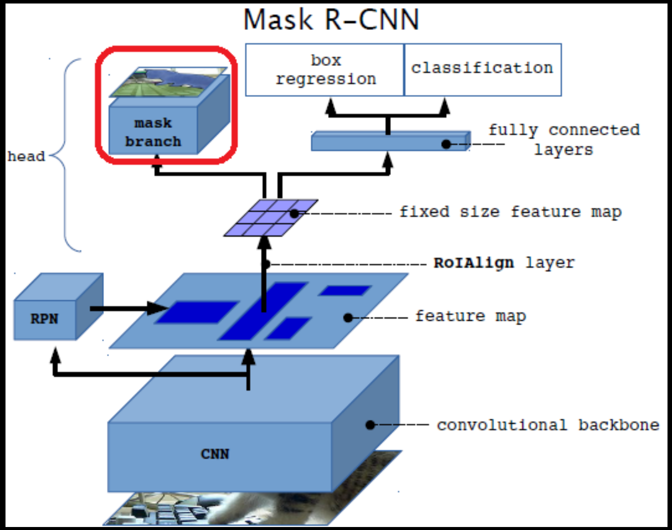
✅ Uses strong geometric constraints (interpretable).
✅ Easy to integrate with known CAD models.
✅ Works well even with relatively small datasets.

Cons:

⚠️ Sensitive to keypoint detection noise.
⚠️ Requires known 3D model.
⚠️ Computationally heavier at runtime.

2D keypoints detection with CNNs





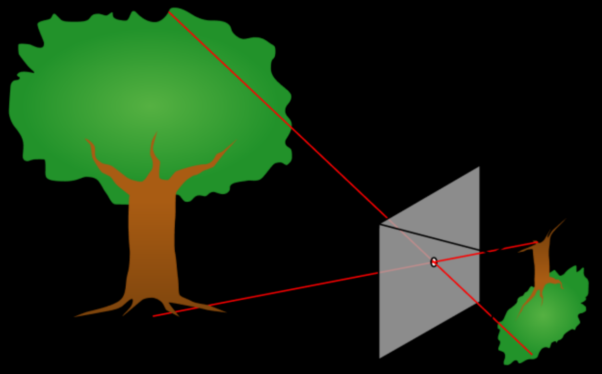
Estimate the **2D image locations** of keypoints corresponding to known 3D points on the object.

Why CNNs:

• Convolutional layers capture **spatial patterns** (edges, corners, textures).

• **Heatmap Regression:** Predicts a probability map per keypoint → take argmax

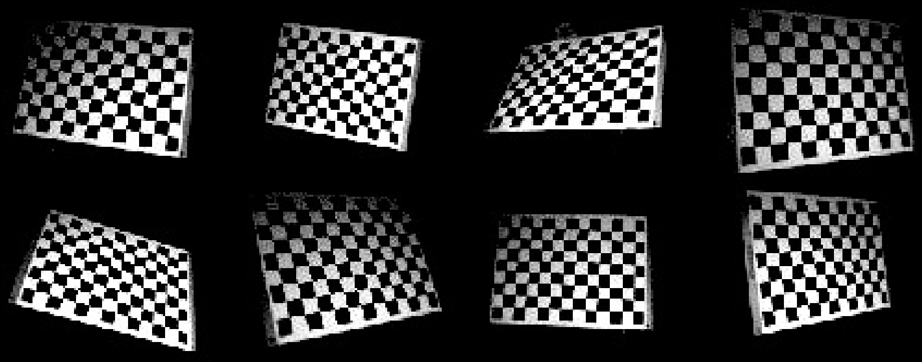
Camera calibration



Camera pinhole model:

$$p_c = K\,[R|t]\,p_w$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$



- Can calibrate the camera intrinsics with checkerboard
- Usually can ignore distortion parameters (unless fisheye camera)
- For camera extrinsics (world → camera) transformation, you can use a marker **(and then invert the transformation!!)**

PnP for pose estimation

Perspective-*n*-Point is the problem of **estimating the pose of a calibrated camera** given a set of *n* 3D points in the world and their corresponding 2D projections in the image.

In our case, we want to estimate the pose of the object, in camera frame

unknown

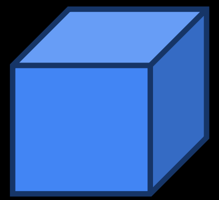
$$p_c = K\ [R|t]\ p_w$$

known

p_c : 2D coordinates of the keypoints

[(234, 531)
 (123, 232)
…
]

p_w: known 3D coordinated of the keypoints



K : intrinsics matrix (from camera calibration)

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 0 \end{bmatrix}$$

Solve for $[R|t]$
Needs at least 4 non-coplanar points