

Robotics: Calibration

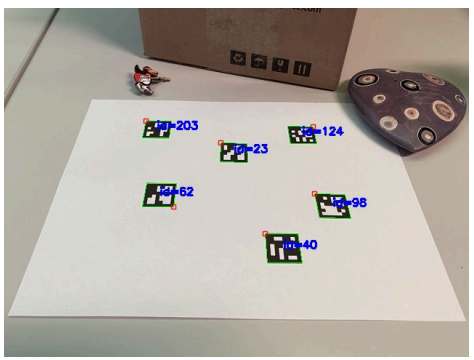
Vladimír Petrík

vladimir.petrik@cvut.cz

19 October 2026

What we know

- How to detect objects in an image
  - checkerboard
  - ArUco markers

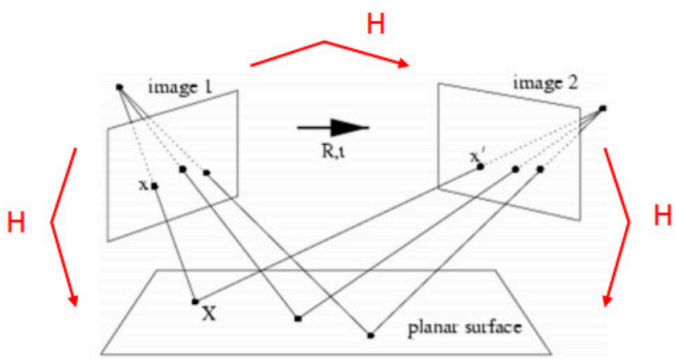




Images from docs.opencv.org

What we know

- How to detect objects in an image
  - checkerboard
  - ArUco markers
- How to estimate a homography
  - maps points between planes
  - estimated from correspondences
  - robust – no depth estimation



Images from docs.opencv.org

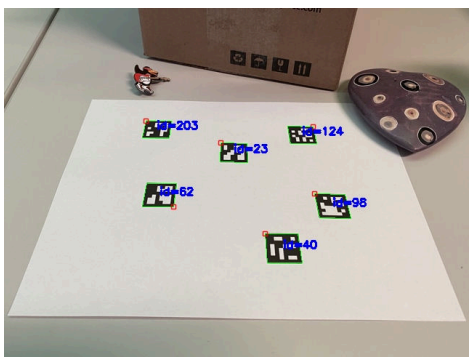
What we know

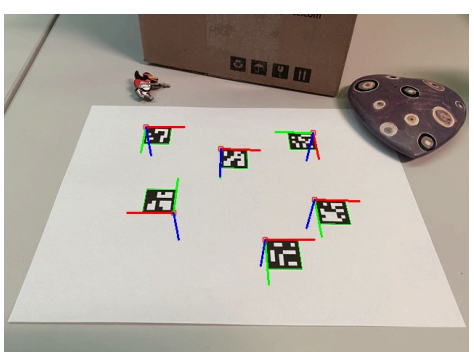
- How to detect objects in an image
  - checkerboard
  - ArUco markers
- How to estimate a homography
  - maps points between planes
  - estimated from correspondences
  - robust – no depth estimation
- Camera intrinsics calibration
  - use checkerboard images
  - estimate the camera matrix K

Images from docs.opencv.org

What we know

- How to detect objects in an image
  - checkerboard
  - ArUco markers
- How to estimate a homography
  - maps points between planes
  - estimated from correspondences
  - robust – no depth estimation
- Camera intrinsics calibration
  - use checkerboard images
  - estimate the camera matrix K
- Pose estimation
  - estimates the camera–object pose
  - uses the PnP algorithm
  - requires K and correspondences





Images from docs.opencv.org

Where is the robot?

- Consider a homography – how can we get 3D points in the robot frame?

- Limited to plane-to-plane mapping

Where is the robot?

- Consider a homography – how can we get 3D points in the robot frame?
  1. Manually measure the ArUco positions with respect to the base frame
     - place ArUco markers in the plane of interest
     - might be hard to measure accurately

- Limited to plane-to-plane mapping

Where is the robot?

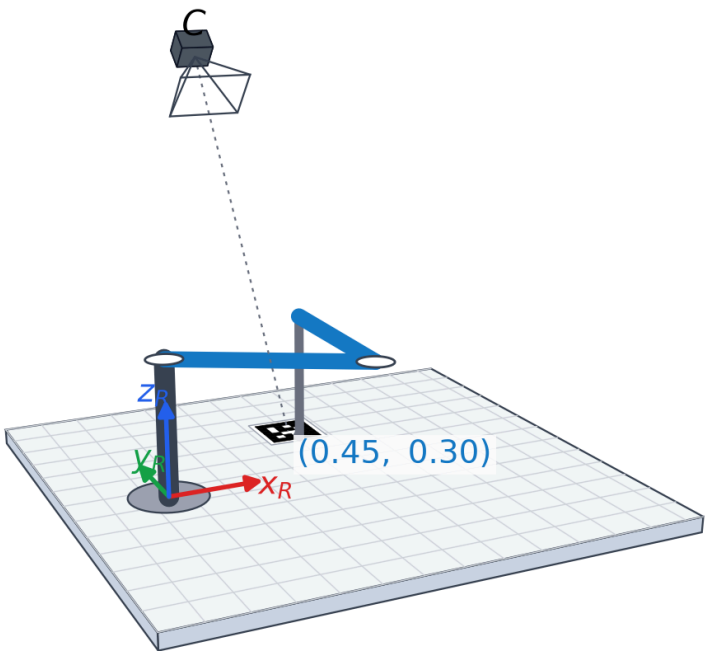
- Consider a homography – how can we get 3D points in the robot frame?
  1. Manually measure the ArUco positions with respect to the base frame
     - place ArUco markers in the plane of interest
     - might be hard to measure accurately
  2. Manually position the robot to a calibration target
     - combine the known 3D points of the target with FK

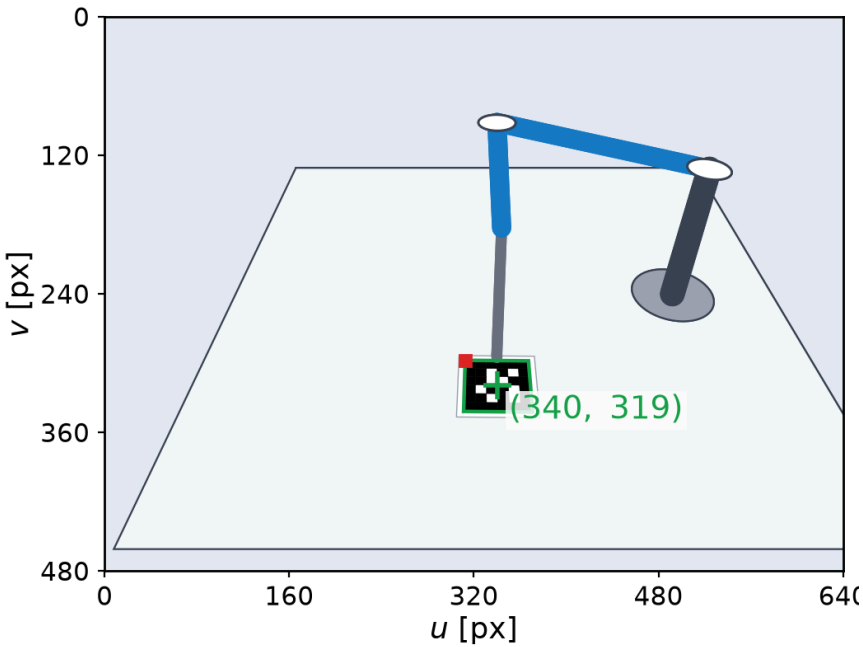
- Limited to plane-to-plane mapping

Where is the robot?

- Consider a homography – how can we get 3D points in the robot frame?
  1. Manually measure the ArUco positions with respect to the base frame
     - place ArUco markers in the plane of interest
     - might be hard to measure accurately
  2. Manually position the robot to a calibration target
     - combine the known 3D points of the target with FK
  3. Use an ArUco marker grasped by the robot
     - use FK to compute the position of the marker in the robot frame
     - we need to move in the plane of interest
     - we will not have a precise estimate of the marker w.r.t. the hand
- Limited to plane-to-plane mapping

Homography from the robot motion

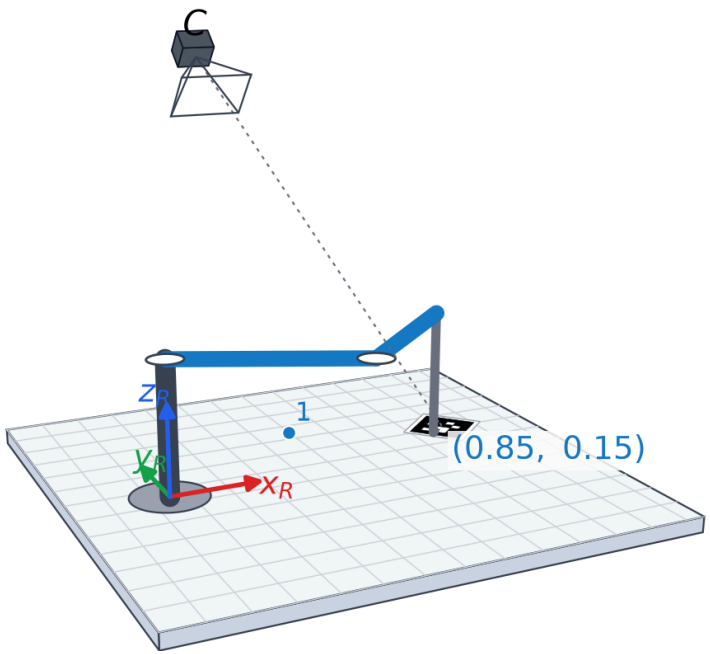


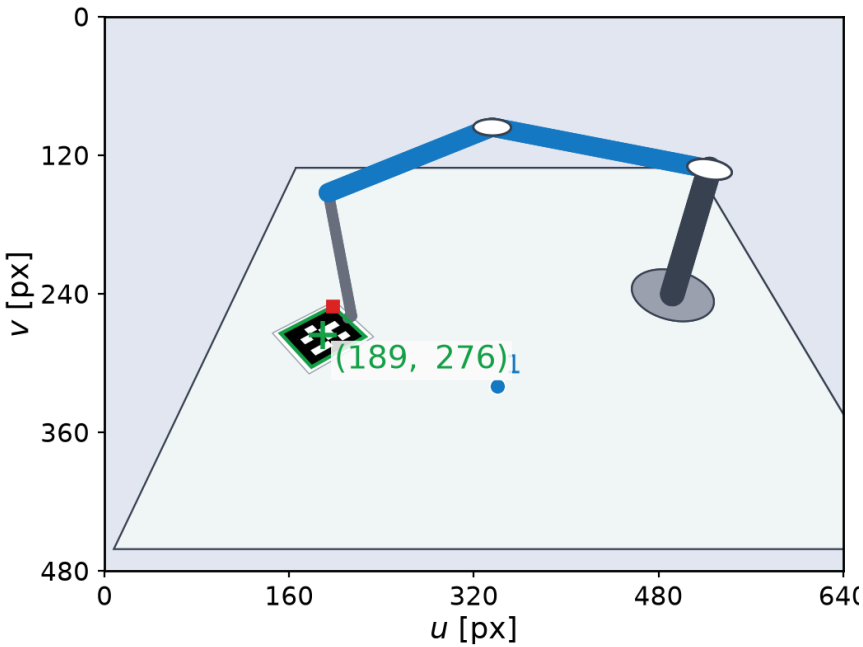


- Marker in the gripper: FK gives its position (x_R, y_R), detection gives its pixel (u, v)

robot (x_R, y_R): $(0.45, 0.30)$
image (u, v): $(340, 319)$

Homography from the robot motion

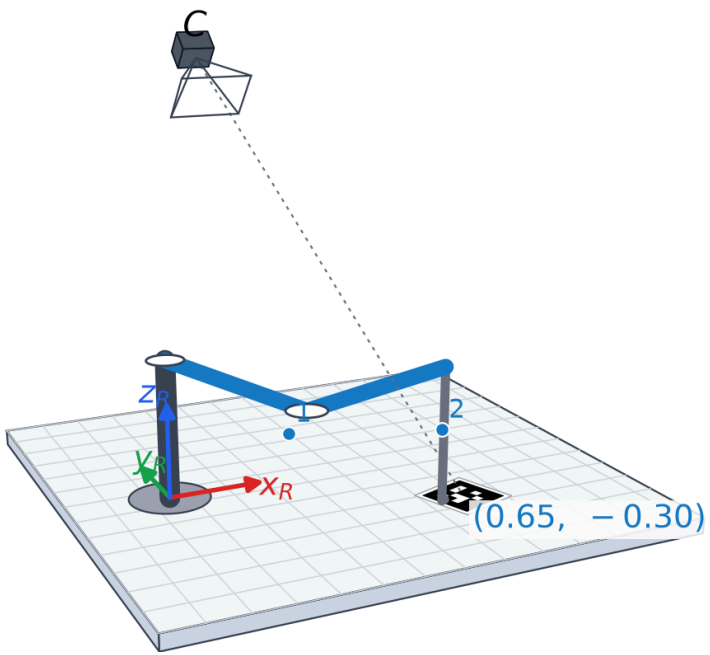


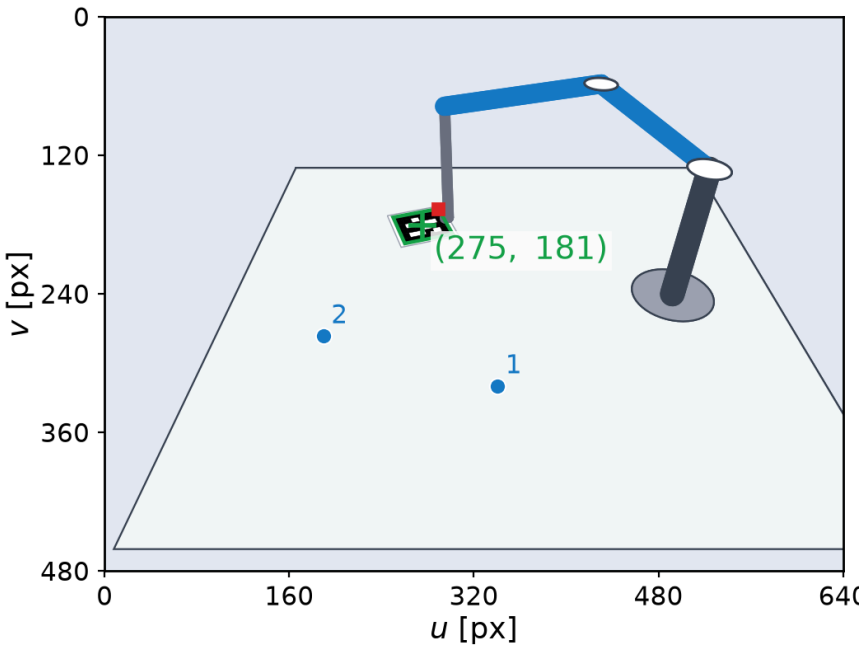


- Move the robot in the plane and detect again: one correspondence per pose

robot (x_R, y_R):	$(0.45, 0.30)$	$(0.85, 0.15)$
image (u, v):	$(340, 319)$	$(189, 276)$

Homography from the robot motion

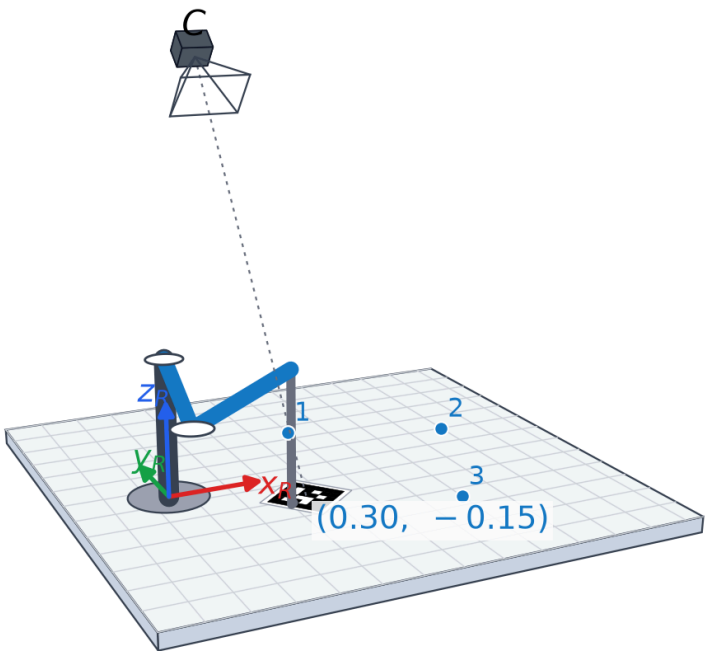


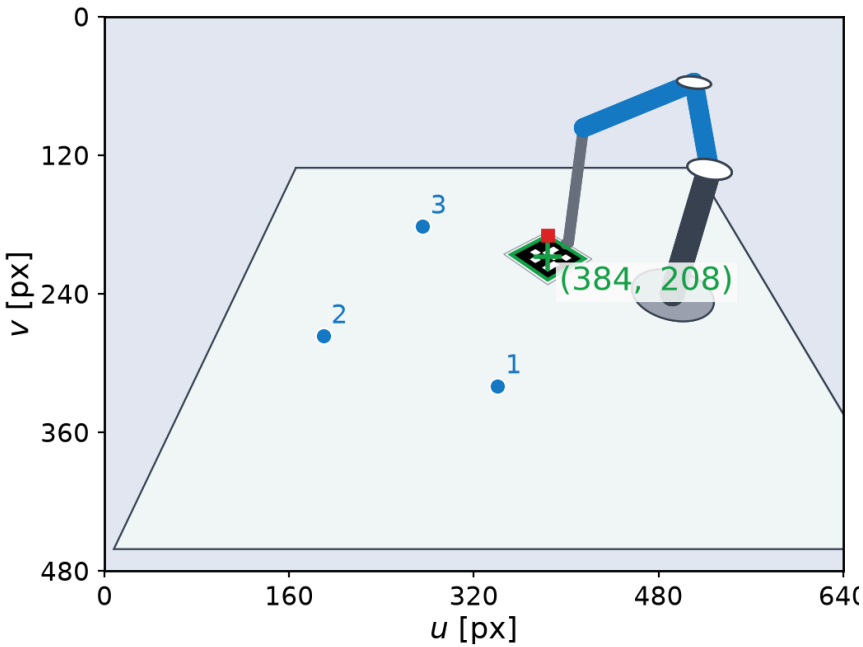


- Move the robot in the plane and detect again: one correspondence per pose

robot (x_R, y_R):	$(0.45, 0.30)$	$(0.85, 0.15)$	$(0.65, -0.30)$
image (u, v):	$(340, 319)$	$(189, 276)$	$(275, 181)$

Homography from the robot motion

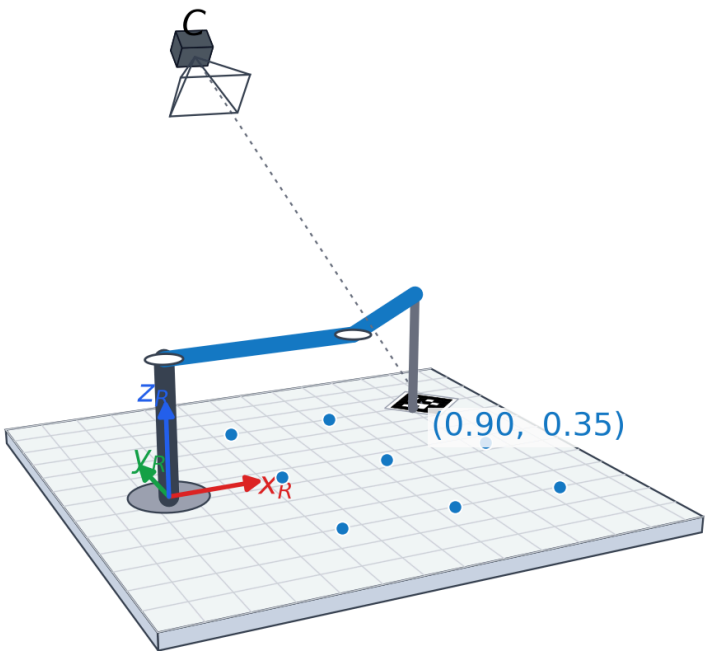


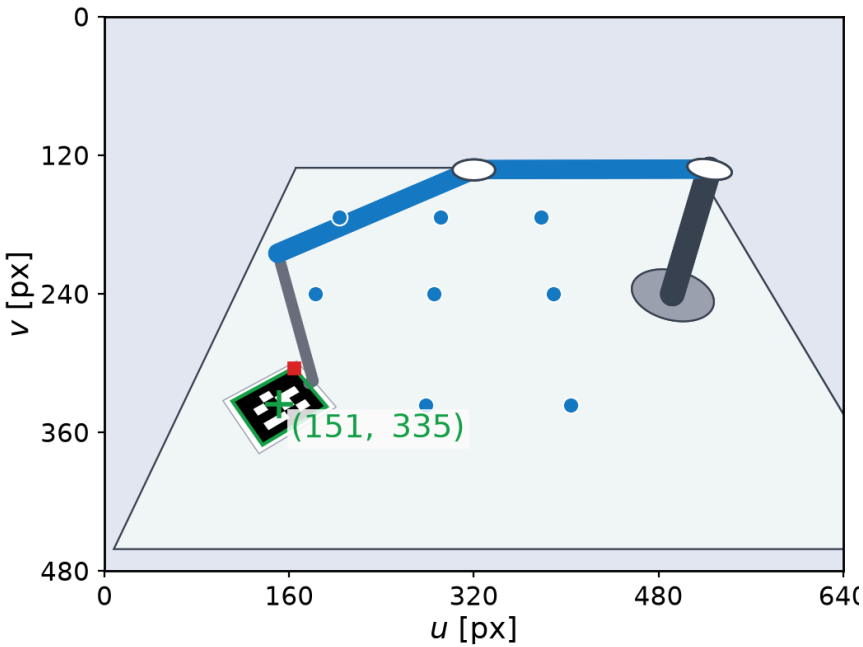


- Four correspondences determine H (8 DoF)

robot (x_R, y_R):	$(0.45, 0.30)$	$(0.85, 0.15)$	$(0.65, -0.30)$	$(0.30, -0.15)$
image (u, v):	$(340, 319)$	$(189, 276)$	$(275, 181)$	$(384, 208)$

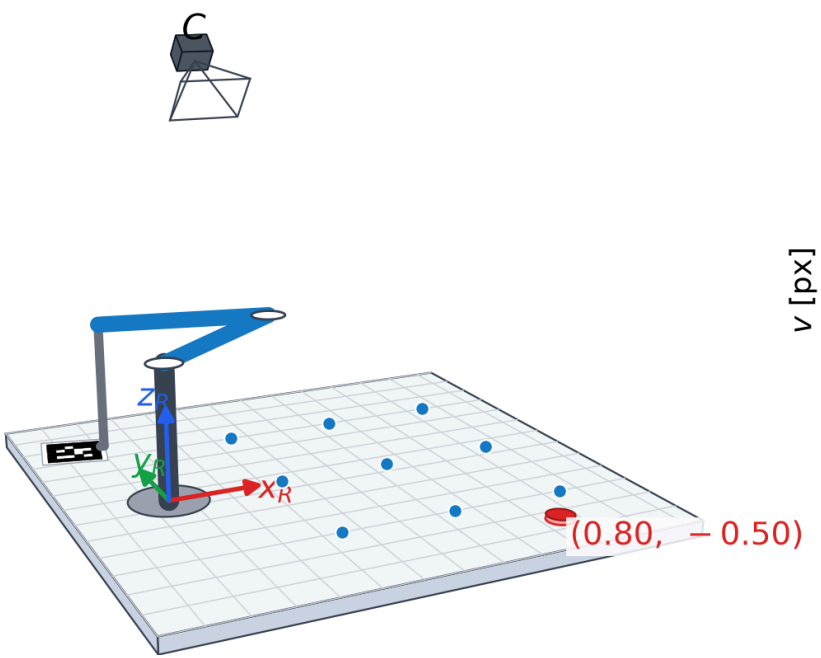
Homography from the robot motion

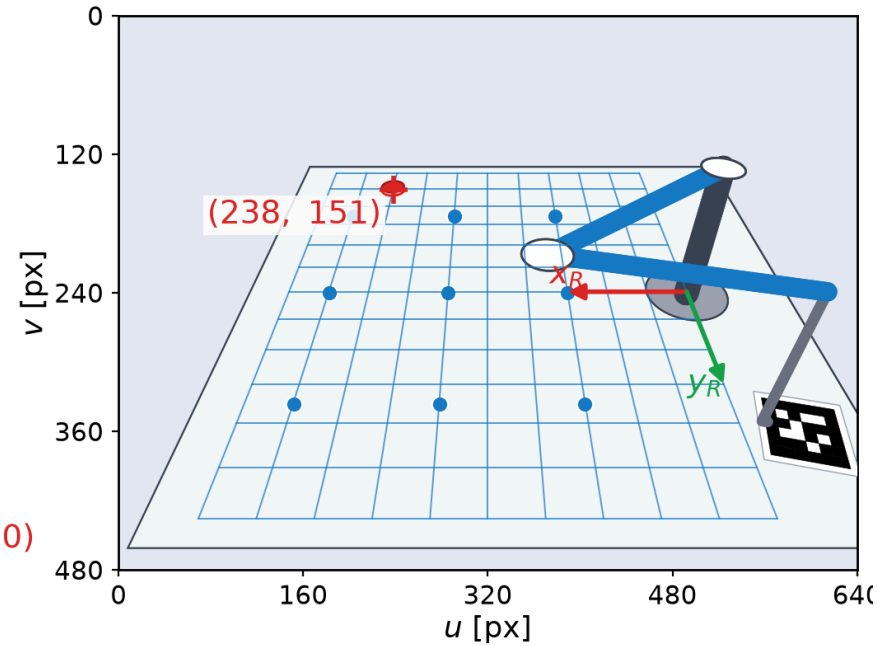




- More poses spread over the workspace: least-squares estimate, `cv2.findHomography`
- The marker has to stay in the plane of interest, here the table

Homography from the robot motion





- H^{-1} maps any pixel of the table plane to the robot frame
- Object detected at $(238, 151) \rightarrow (0.80, -0.50)$: a target for the robot

Calibrating the camera pose

- Hand-eye calibration
- Solve $A^i X = Y B^i$
  - measurements: $A^i, B^i \in SE(3)$
  - estimated parameters: $X, Y \in SE(3)$

Calibrating the camera pose

- Hand-eye calibration
- Solve $A^i X = Y B^i$
  - measurements: $A^i, B^i \in SE(3)$
  - estimated parameters: $X, Y \in SE(3)$
- The camera can be mounted on
  - the robot base frame (eye-to-hand calibration)
  - the gripper frame (eye-in-hand calibration)

Calibrating the camera pose

- Hand-eye calibration
- Solve $A^i X = Y B^i$
  - measurements: $A^i, B^i \in SE(3)$
  - estimated parameters: $X, Y \in SE(3)$
- The camera can be mounted on
  - the robot base frame (eye-to-hand calibration)
  - the gripper frame (eye-in-hand calibration)

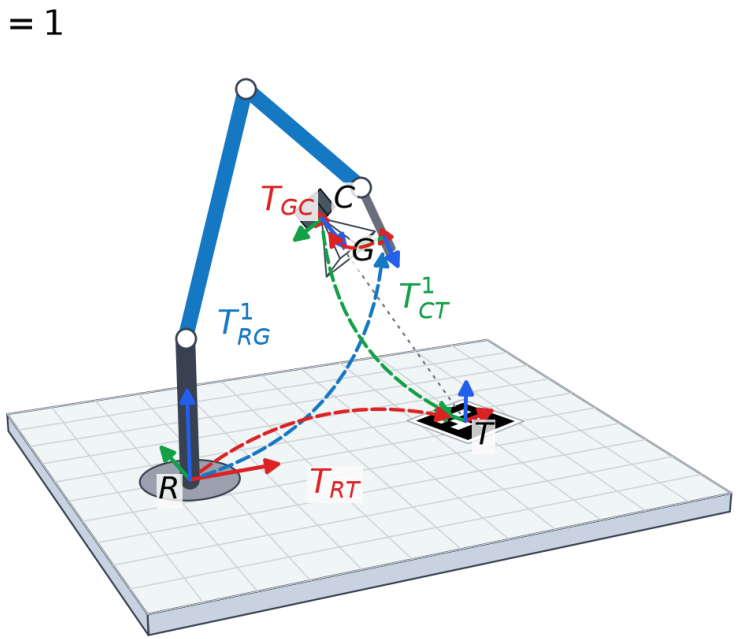
- Eye-to-hand calibration
  - $A^i = T^i_{RG}$
  - $B^i = T^i_{CT}$
  - $X = T_{GT}$
  - $Y = T_{RC}$

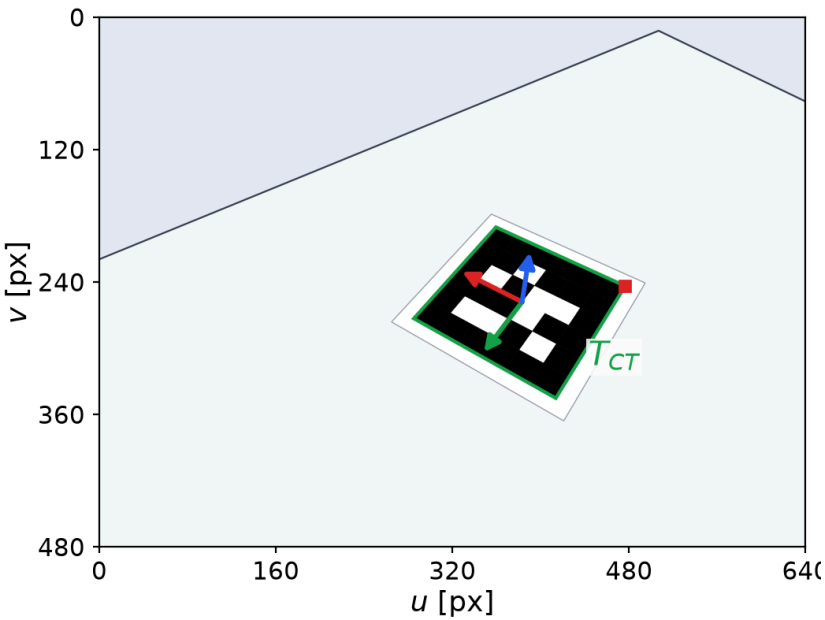
Calibrating the camera pose

- Hand-eye calibration
- Solve $A^i X = Y B^i$
  - measurements: $A^i, B^i \in SE(3)$
  - estimated parameters: $X, Y \in SE(3)$
- The camera can be mounted on
  - the robot base frame (eye-to-hand calibration)
  - the gripper frame (eye-in-hand calibration)

- Eye-to-hand calibration
  - $A^i = T^i_{RG}$
  - $B^i = T^i_{CT}$
  - $X = T_{GT}$
  - $Y = T_{RC}$
- Eye-in-hand calibration
  - $A^i = T^i_{CT}$
  - $B^i = T^i_{GR}$
  - $X = T_{TR}$
  - $Y = T_{CG}$

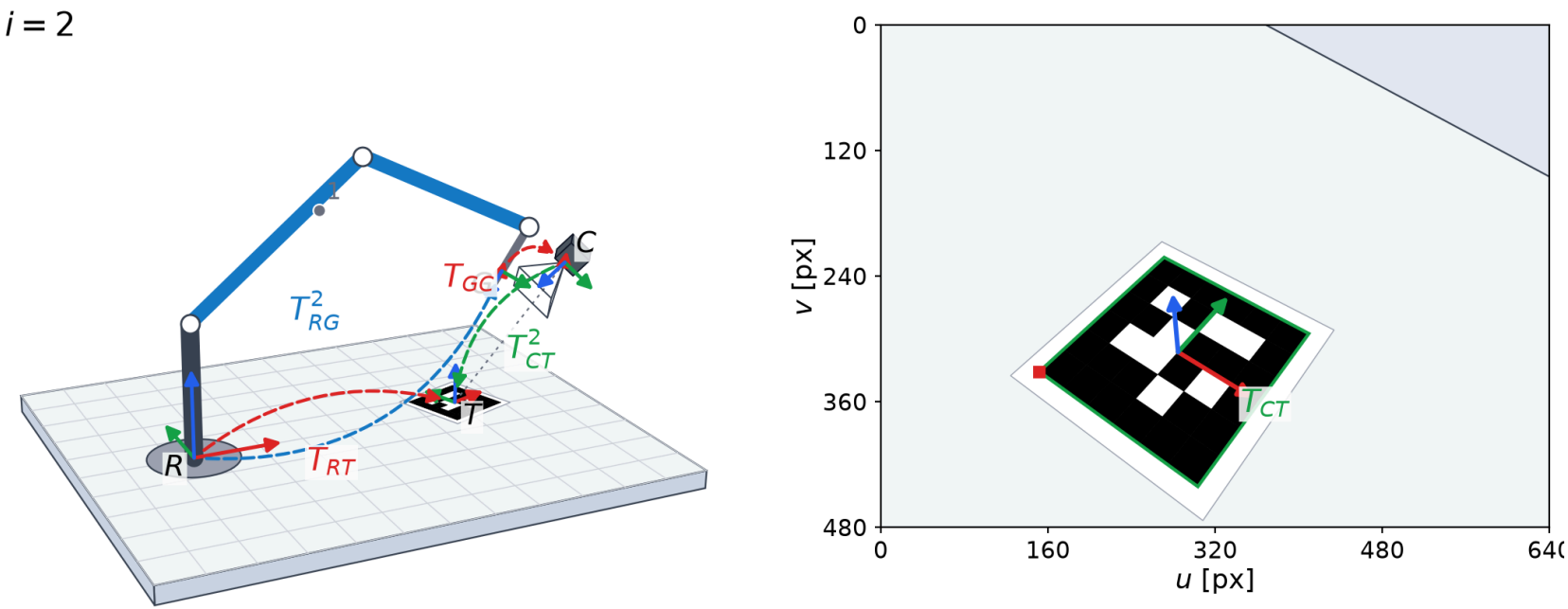
Eye-in-hand calibration





- Camera on the gripper, ArUco target fixed on the table: T^1_{RG} from FK, T^1_{CT} from pose estimation
- T_{GC} and T_{RT} are unknown, but constant

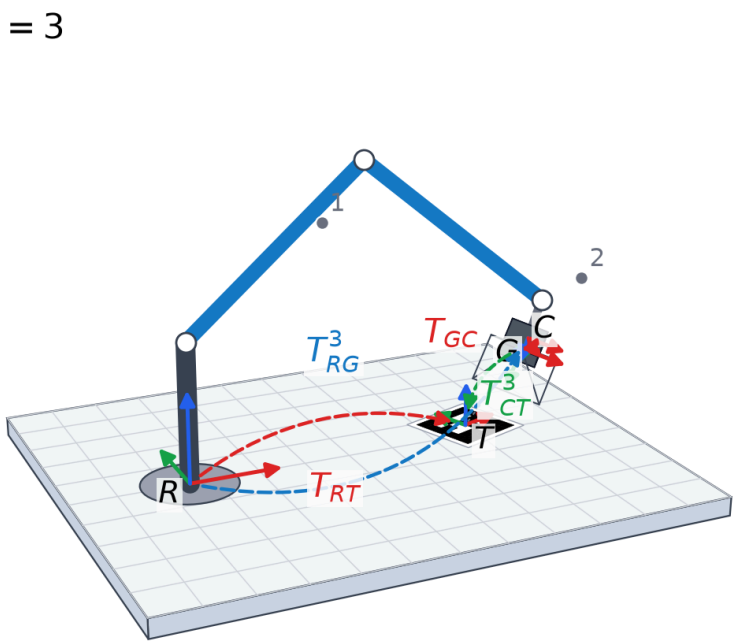
Eye-in-hand calibration

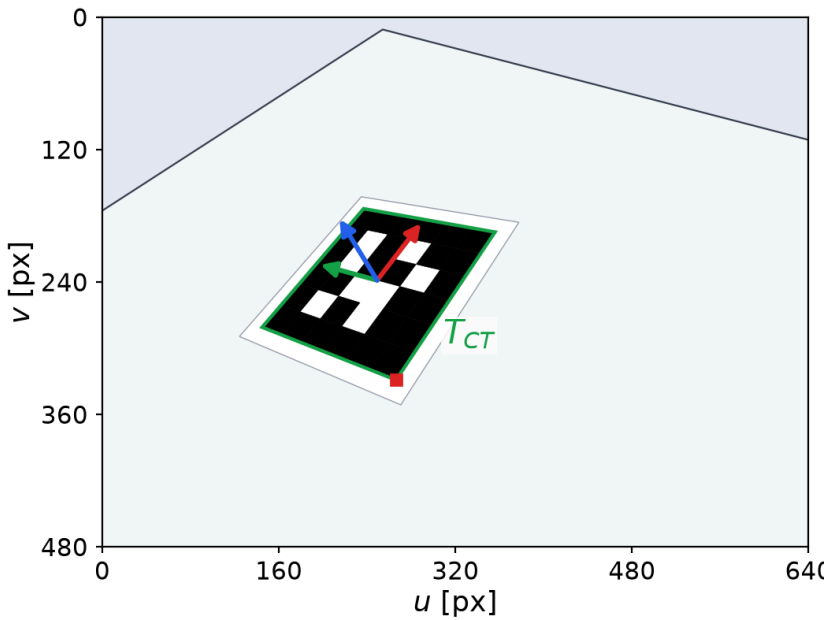


- Move the robot: new T^2_{RG} and T^2_{CT}, the loop closes for every pose i

$$T^i_{RG}\, T_{GC}\, T^i_{CT} = T_{RT}$$

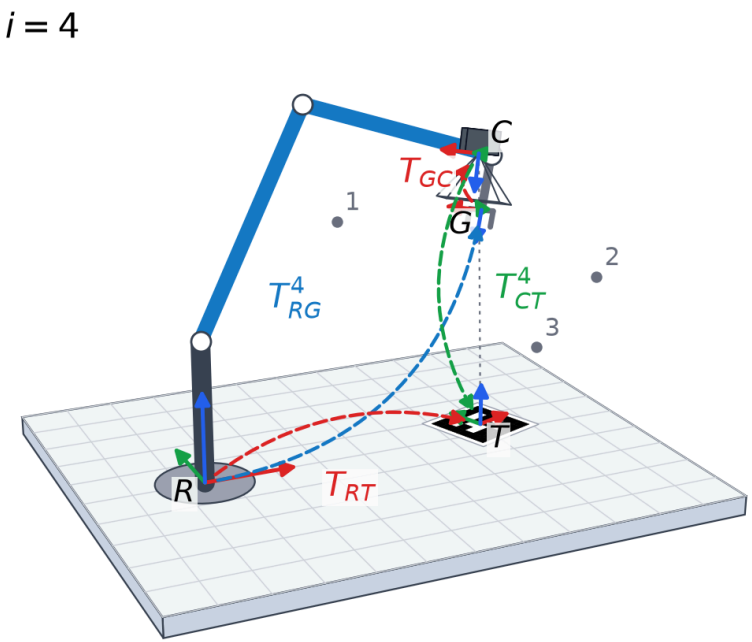
Eye-in-hand calibration

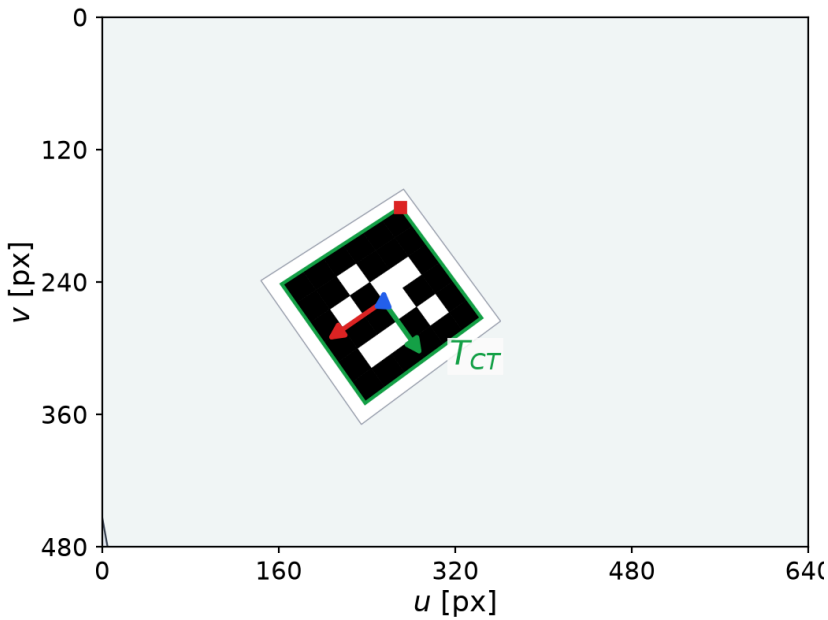




- Rotate the camera about different axes: the rotations of T^i_{RG} determine the rotation of T_{GC}
- Pure translations or rotations about a single axis are degenerate

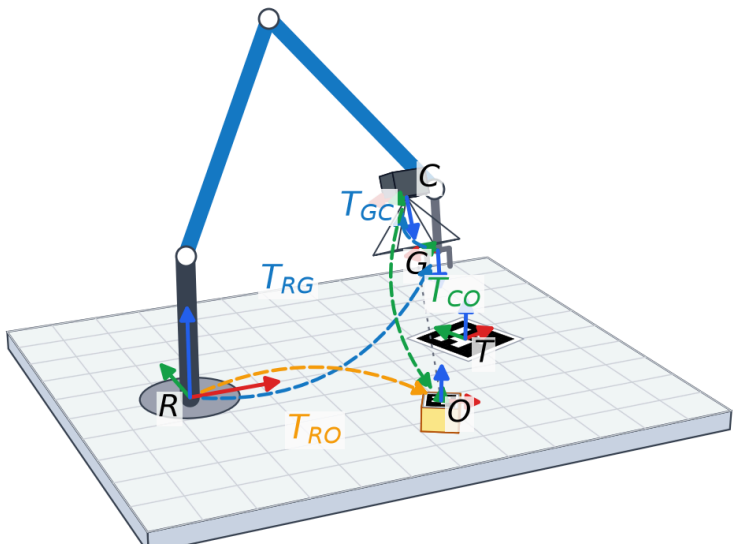
Eye-in-hand calibration

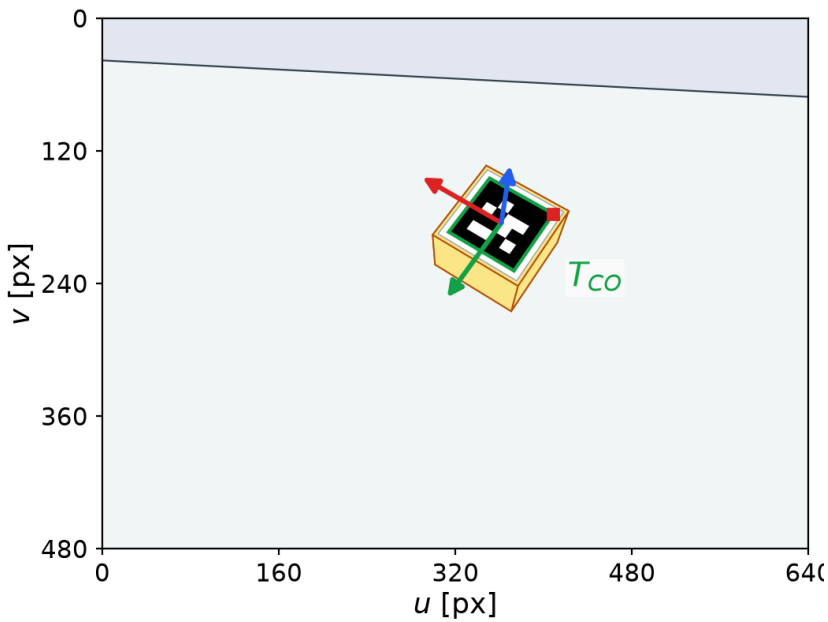




- At least 3 poses with different rotations, more poses for a least-squares estimate
- $A^i X = Y B^i$ with $A^i = T^i_{CT}$, $B^i = T^i_{GR}$, $X = T_{TR}$, $Y = T_{CG}$:
  `cv2.calibrateRobotWorldHandEye`

Eye-in-hand calibration





- Calibrated T_{GC}: any marker seen by the camera gets a pose in the robot frame

$$T_{RO} = T_{RG}\, T_{GC}\, T_{CO}$$

Code for the $AX = YB$ calibration

```python
def solve_AX_YB(a: list[SE3], b: list[SE3]) -> tuple[SE3, SE3]:
    """Solve A^i X = Y B^i, return X, Y"""

    rvec_a = [T.rotation.log() for T in a]
    tvec_a = [T.translation for T in a]
    rvec_b = [T.rotation.log() for T in b]
    tvec_b = [T.translation for T in b]

    Rx, tx, Ry, ty = cv2.calibrateRobotWorldHandEye(rvec_a, tvec_a, rvec_b, tvec_b)
    return SE3(tx[:, 0], SO3(Rx)), SE3(ty[:, 0], SO3(Ry))
```

Influence of the ArUco marker size

- Depth estimation of small ArUco markers is hard
  - large errors result in an imprecise calibration
  - large markers limit the motion

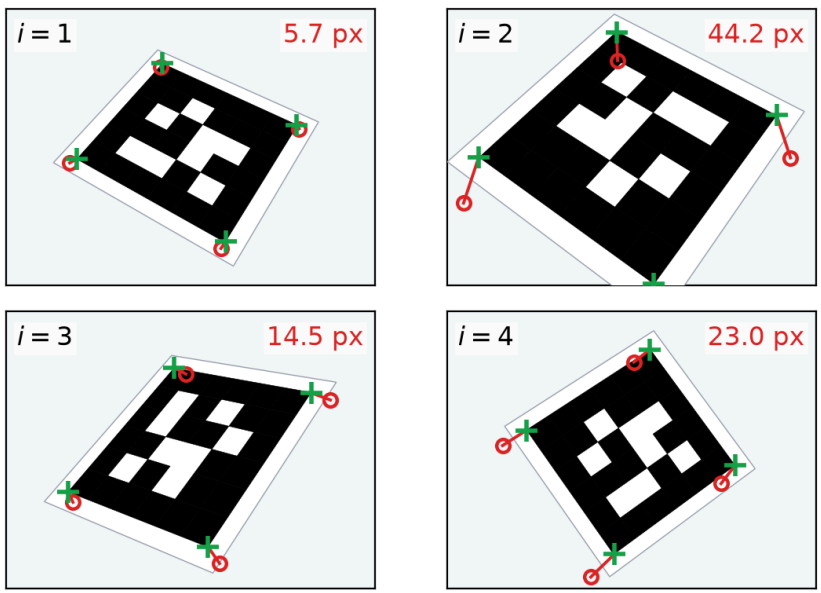
Influence of the ArUco marker size

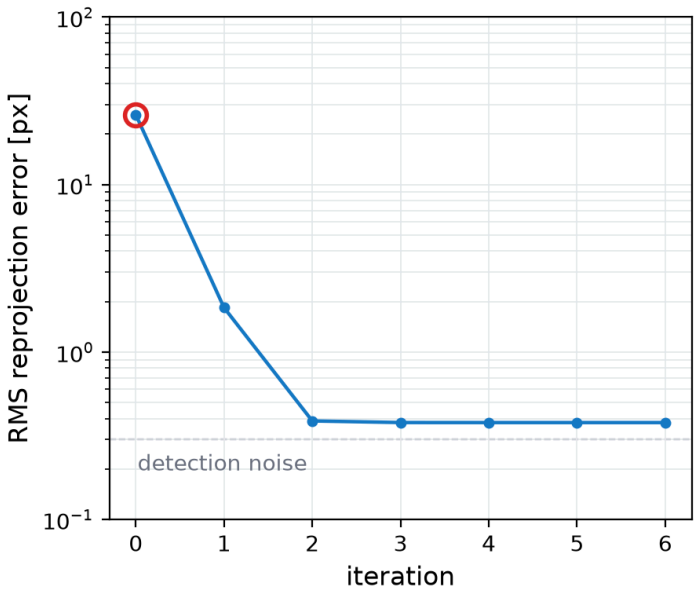
- Depth estimation of small ArUco markers is hard
  - large errors result in an imprecise calibration
  - large markers limit the motion
- Solution: minimization of the reprojection error
  - the original measurements are in pixels
  - we can perform a fine optimization in pixel space
  - reprojection error:

$$X^*, Y^* = \arg\min_{X,Y} \sum_i \left\| \boldsymbol{u}_i - \pi(\boldsymbol{x}_i(X, Y)) \right\|^2$$

    - $\boldsymbol{u}_i$ – detected 2D point, $\boldsymbol{x}_i$ – 3D point in the camera frame, π – projection
    - $\boldsymbol{x}_i$ is computed by FK and the poses X and Y
    - the initial guess comes from the hand-eye calibration

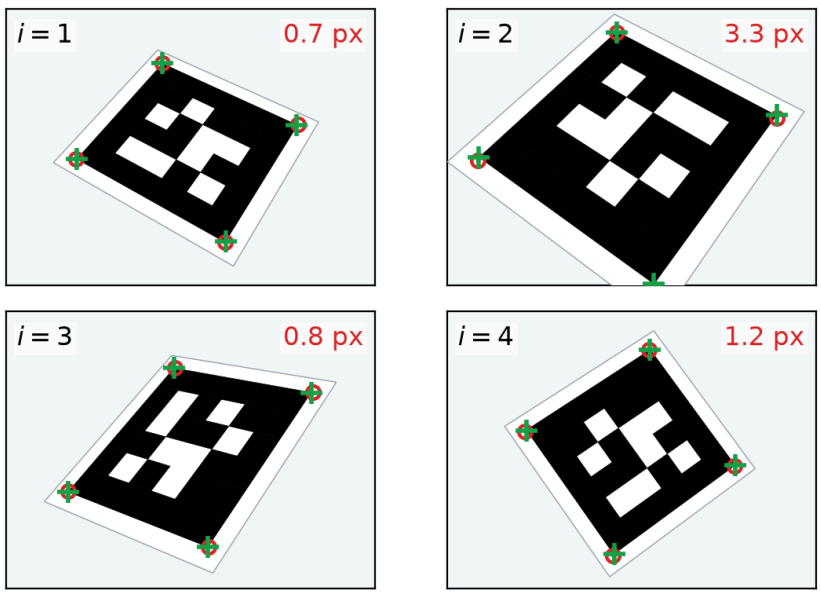
Reprojection error minimization

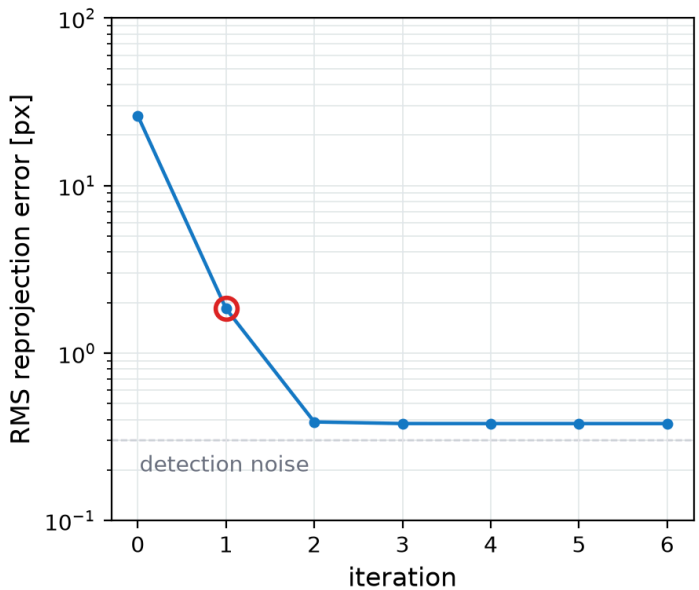




- Initial X, Y from the hand-eye calibration: detected corners u_i vs. reprojected corners $\pi(x_i(X, Y))$
- Errors of tens of pixels, different in every image

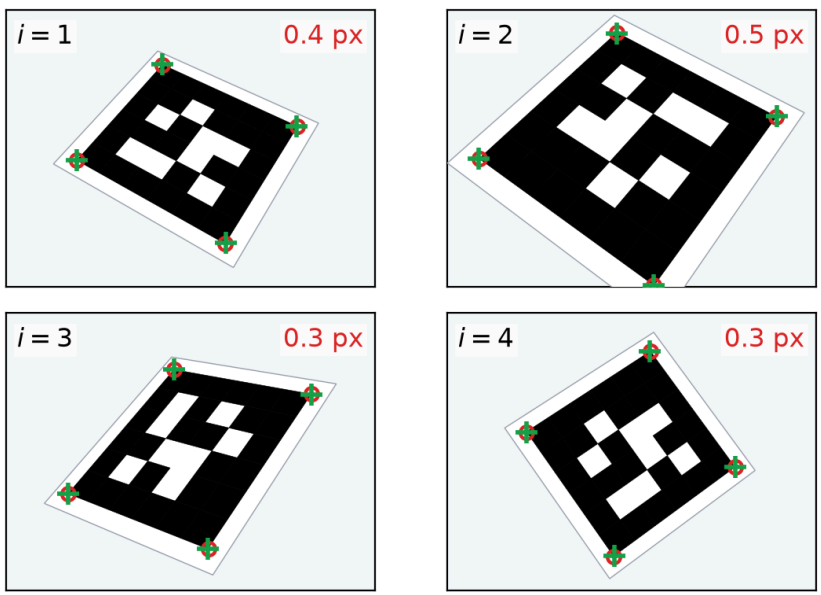
Reprojection error minimization

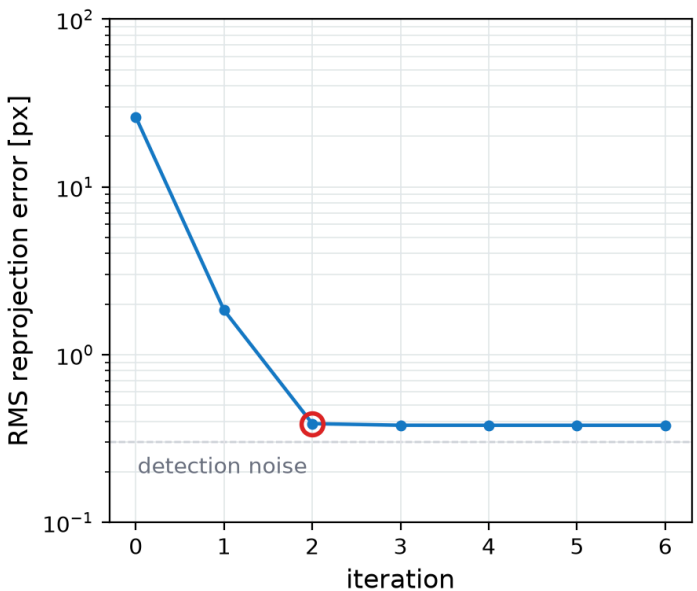




- Gauss–Newton step: linearize the residuals in the 12 parameters of X, Y, solve the linear least squares, update

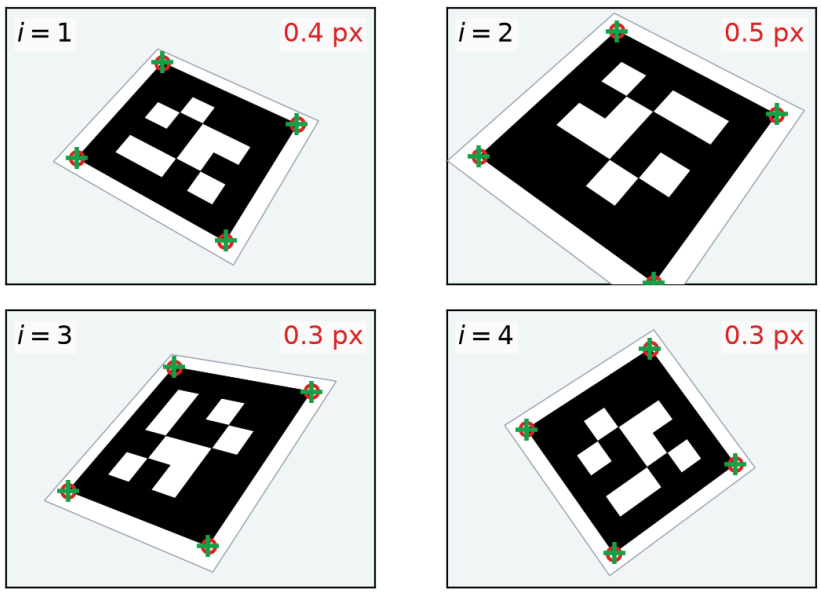
Reprojection error minimization

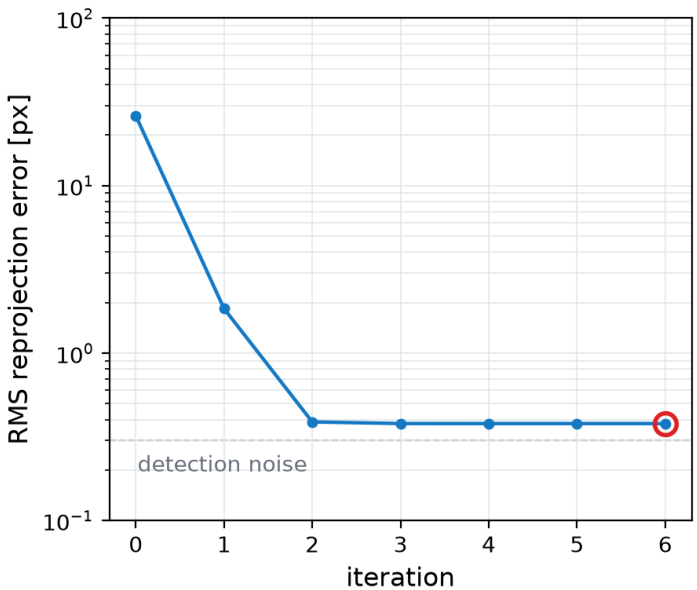




- Second iteration: the error drops below one pixel

Reprojection error minimization





- Converged after three iterations: the residual is at the level of the detection noise
- In practice: `scipy.optimize.least_squares` with the residual function

Influence of the robot kinematics

- The robot kinematics is not perfect either
  1. different configurations of the robot have different errors
  2. often errors in joint offsets

Influence of the robot kinematics

- The robot kinematics is not perfect either
  1. different configurations of the robot have different errors
  2. often errors in joint offsets
- Solution #1: restrict to a subset of configurations
  - when solving IK, use only similar configurations
  - for the calibration but also for the robot operation!

Influence of the robot kinematics

- The robot kinematics is not perfect either
  1. different configurations of the robot have different errors
  2. often errors in joint offsets
- Solution #1: restrict to a subset of configurations
  - when solving IK, use only similar configurations
  - for the calibration but also for the robot operation!
- Solution #2: calibrate the joint offsets
  - as in the reprojection error minimization

    $$X^*, Y^*, \boldsymbol{\theta}^*_{\text{offset}} = \underset{X,Y,\boldsymbol{\theta}_{\text{offset}}}{\arg\min} \sum_i \left\| \boldsymbol{u}_i - \pi(\boldsymbol{x}_i(X, Y, \boldsymbol{\theta}_{\text{offset}})) \right\|^2$$

  - FK is adjusted to include the joint offsets

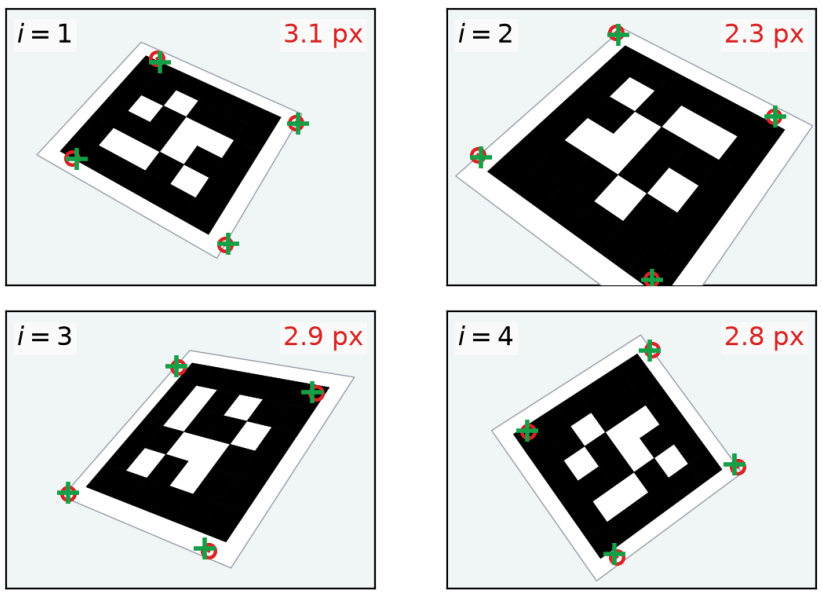
Influence of the robot kinematics

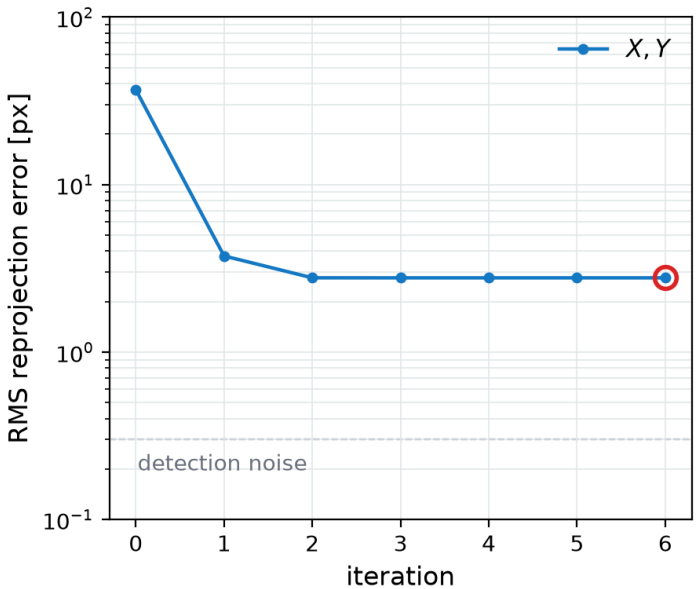
- The robot kinematics is not perfect either
  1. different configurations of the robot have different errors
  2. often errors in joint offsets
- Solution #1: restrict to a subset of configurations
  - when solving IK, use only similar configurations
  - for the calibration but also for the robot operation!
- Solution #2: calibrate the joint offsets
  - as in the reprojection error minimization

    $$X^*, Y^*, \boldsymbol{\theta}^*_{\text{offset}} = \underset{X,Y,\boldsymbol{\theta}_{\text{offset}}}{\arg\min} \sum_i \left\| \boldsymbol{u}_i - \pi(\boldsymbol{x}_i(X, Y, \boldsymbol{\theta}_{\text{offset}})) \right\|^2$$

  - FK is adjusted to include the joint offsets
- Calibration is not a one-time task
  - calibrate the robot before each important task
  - calibrate the robot after any significant change

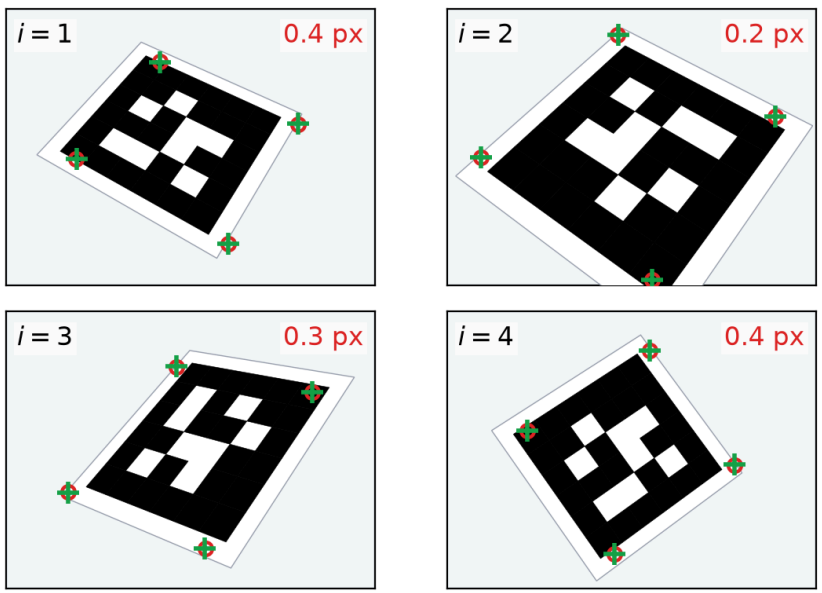
Calibrating the joint offsets

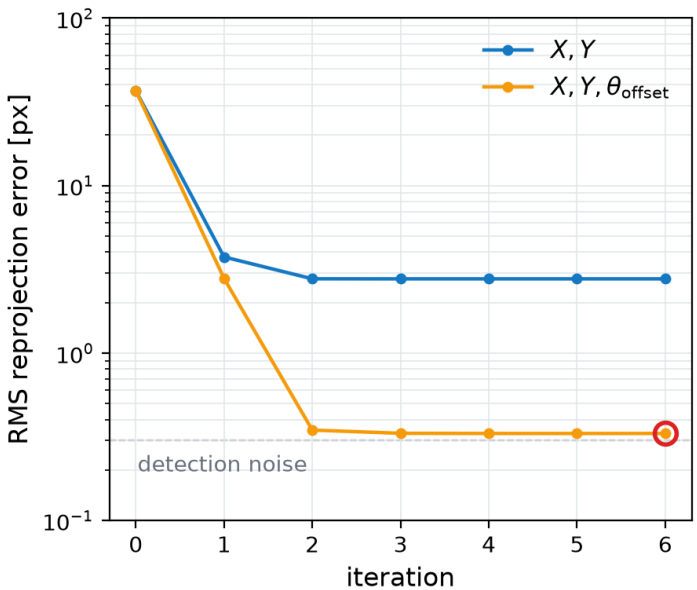




- Robot with joint offsets (here $-1.5°$ and $2.5°$ on two joints): X, Y alone cannot explain the measurements
- The residual stays above the detection noise, differently in each configuration

Calibrating the joint offsets





- Adding θ_{offset} to the parameters: the residual drops to the noise level and the offsets are recovered
- More poses and configurations make the offsets better observable

Conclusion

- Calibration depends on the task
  - homography
  - camera intrinsics
  - camera pose
  - robot kinematics
- Calibration is not a one-time task
  - good to automate it

Laboratory

- Weeks 5 and 6: individual work on the final project, consultations with the teachers
- hw03 (camera–robot transformation, homography) is due 25. 10.
- Reserve the robots in BRUTE