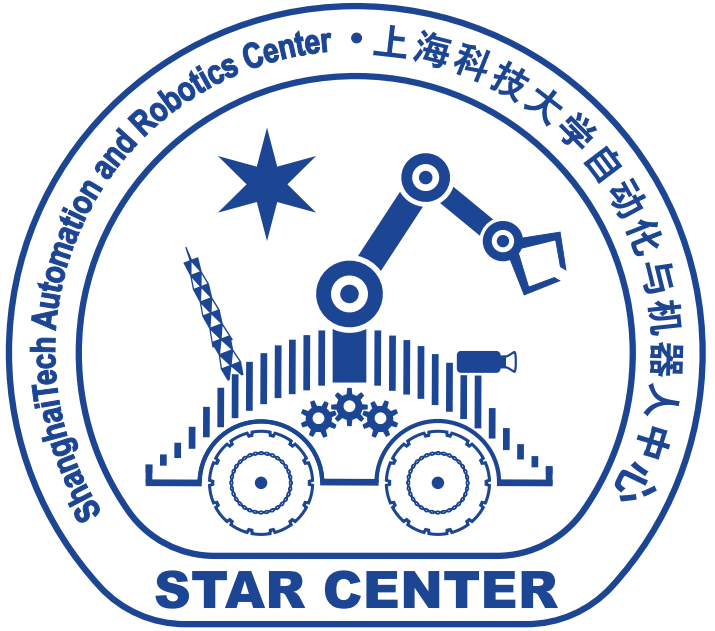
上海科技大学

ShanghaiTech University

CS289: **Mobile Manipulation Fall 2025**

Sören Schwertfeger

ShanghaiTech University



Outline

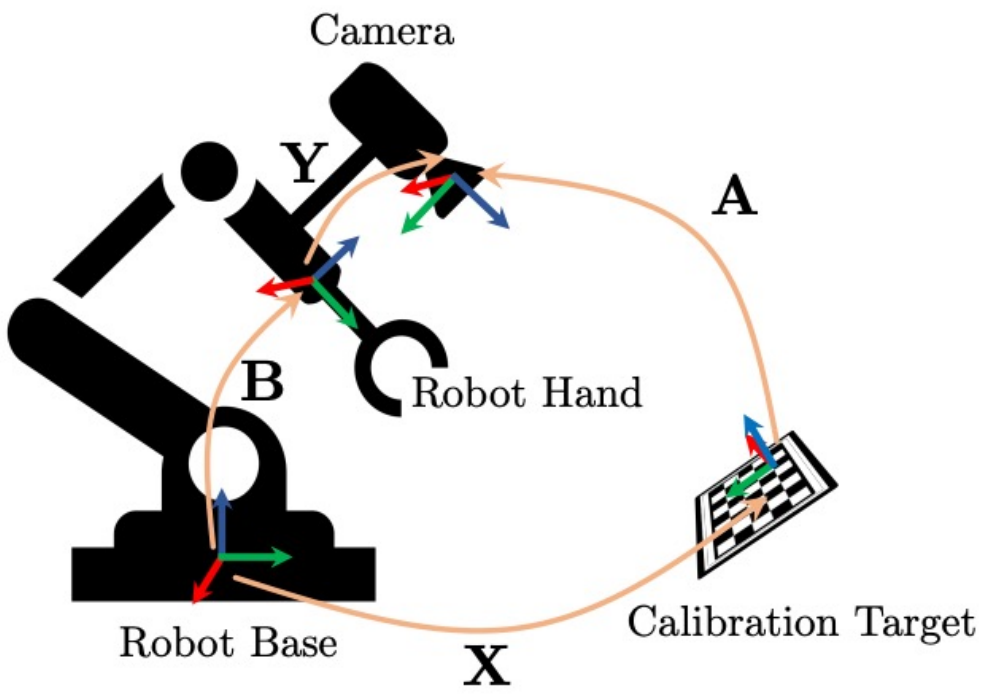
- Calibraiton
- Sensor Fusion
- Visual Servoing

CALIBRATION

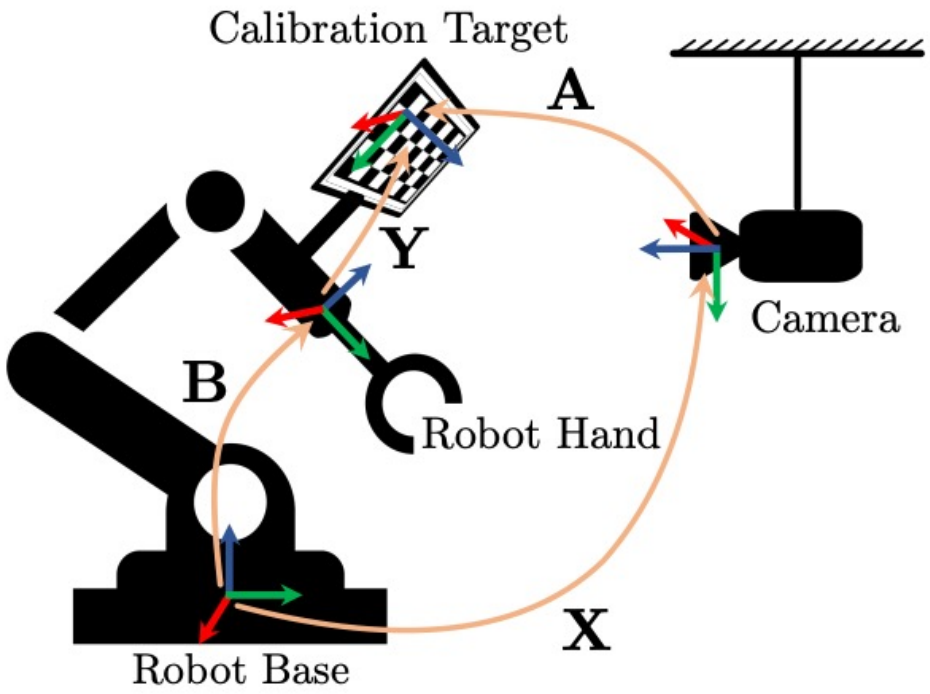
Calibration

- Intrinsic calibration:
  - Correct raw sensor data such that:
    - It adheres to certain standards
    - Reduces the error/ noise
  - Robotics/ autonomous driving:
    - Camera calibration!
    - LiDARS can also be calibrated (factory calibration typically sufficient)
    - All sensors …
- Extrinsic calibration:
  - Determine the pose (position and orientation) of sensors w.r.t. a frame of reference

Manipulation: Hand-Eye-Calibration



(a) eye-on-hand; aka: eye-in-hand



(a) eye-to-base; aka: eye-to-hand

Fig. 2. Visualization of the hand-eye/robot-world calibration problem. Both (a) *eye-on-hand* and (b) *eye-to-base* cases are constrained by $\mathbf{AX} = \mathbf{YB}$.

Calibration

Manipulator Calibration

- Move manipulator -> track “tool center point (tcp)” motion (or link to which camera is attached)
  - Use arm forward kinematics OR
  - Use tracking system. Difficulty: Calibration from tracking markers to tool needed
- Eye-on-hand:
  - Observe static target -> use Perspective-n-Point (PnP) to estimate camera transforms
- Eye-to-base
  - Static camera: observe moving target to estimate target transforms

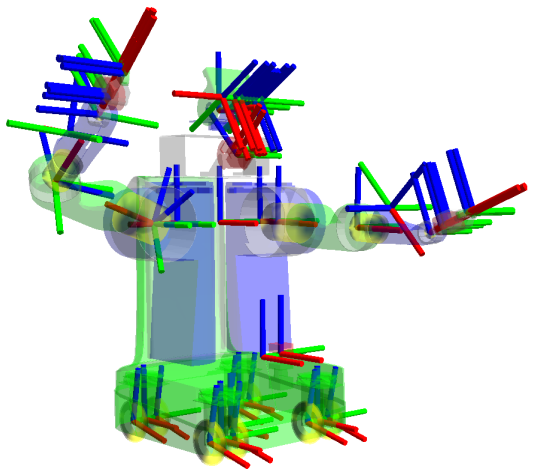
Multi-Sensor Calibration

- Mobile Robot with multiple sensors
- Option 1: Hand-eye approach:
  - Move robot (with sensors)
  - Estimate motions in all sensor frames
  - Use optimization to find transforms that explain all motions
- Option 2: Moving target approach:
  - Move a target around robot
  - Target needs to be observable by the sensors
  - Ideally: track target pose in tracking system
  - Use optimization to find transforms
  - Cannot estimate IMU pose

https://www.youtube.com/watch?v=jamH43W46Bo

How to represent extrinsic calibration in ROS?

- TF tree (Transform tree) (tf2)
  - http://wiki.ros.org/tf
  - Publish static transform (e.g. robot frame to camera 1 frame)
    - Option 1: build/ edit model of robot: URDF: http://wiki.ros.org/urdf/Tutorials
    - Option 2: publish TF by hand (once): http://wiki.ros.org/tf#static_transform_publisher
  - Publish dynamic transforms (e.g. robot arm motion; world frame to car frame)
    - tf2 broadcaster - publish tf messages – 1hz to 100hz
    - http://wiki.ros.org/tf2/Tutorials/Writing%20a%20tf2%20broadcaster%20%28C%2B%2B%29
- Visualize TF free in rviz:



RVIZ

- Rviz needs proper TF tree to visualize all the data:
  - Need to be able to transform all data into one common frame (fixed frame)
  - One application of sensor fusion
  - putting proper transforms in the tf tree is essential (also for other applications/ algorithms)
  - URDF: robot model, **calibration results**, measured distances
  - Static transform broadcaster: calibration results that may change more frequently
  - tf broadcaster: dynamic tf: e.g.:
    - Front wheel direction
    - Steering wheel orientation
    - One can publish tf for the wheel orientation!
    - Status of doors, trunk, …
    - …

Calibration of Arm Parameters

- Typically arm manufactured precisely => factory parameters excellent

- Manual calibration of (self-build) arm:
  - Attach tracking system target to tool center point (tcp)
  - Move arm, record all joint states and tcp poses
  - Minimize forward kinematics error by optimizing arm parameters …

SENSOR FUSION

Sensor Fusion

- Use data from more than one sensor:
  - reduce uncertainty and/ or
  - get more/ better data and/ or
  - get more reliability (in case one sensor doesn't provide (good) data
  - Increase sensing area
- Needs to be synchronized/ time stamped AND
- Calibrated (Intrinsic & Extrinsic)

- Examples to enhance sensor data:
  - Combine point clouds from several LiDARS to one big point cloud
  - Combine images from several cameras to a panorama image
  - Use two or more camera images for stereo processing
  - Colorize a point cloud (e.g. from Velodyne) with camera data
  - IMU: Fusing Accelerometer, Gyroscope, Compass

- Examples to reduce uncertainty:
  - Use the IMU together with localization from 3D LiDARs or cameras (and GPS)
  - Object detection/ tracking/ prediction from multiple sensors

Algorithms often used:

Averaging
Kalman Filter
Bayesian networks
Neural Networks
…

Hardware Time Synchronization

- Can be triggered:
  - Cameras
  - IMUs
- Need Time Stamps:
  - LiDAR
  - GPS (provides time stamps)
  - PPS: pule per second

Mapping Robot Point Cloud Colorization Example

Mapping robot of Prof. Schwertfeger

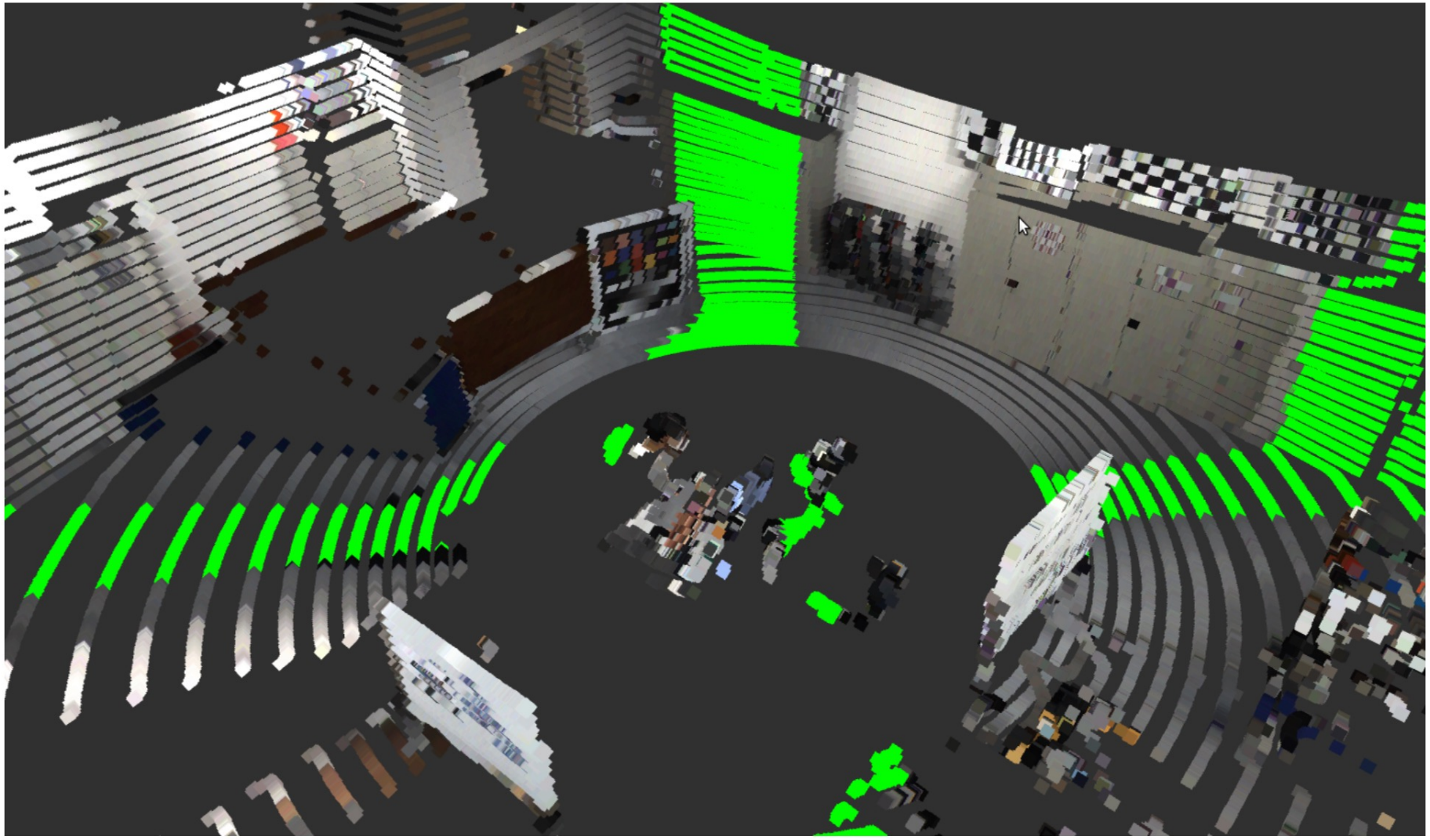
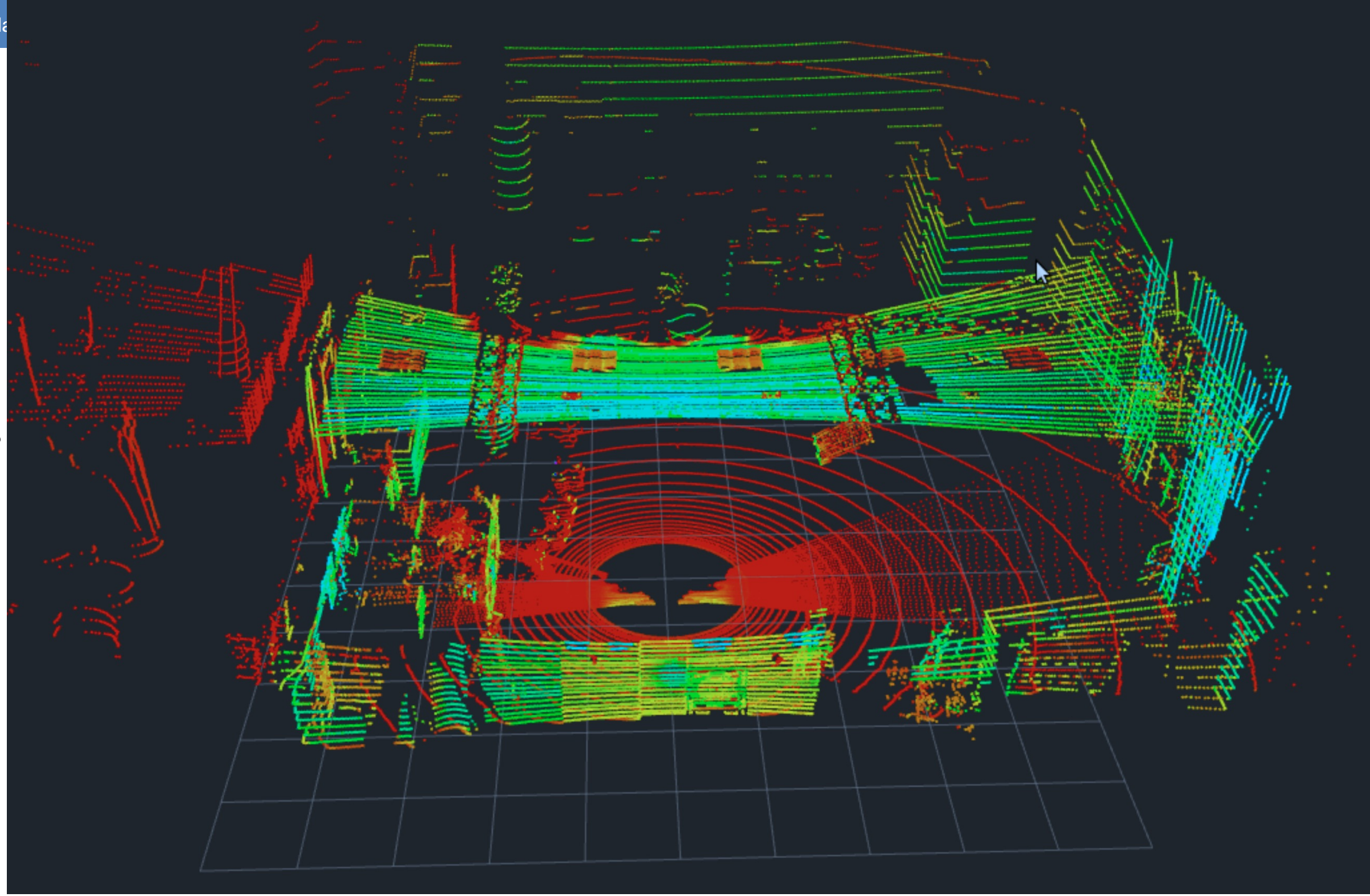
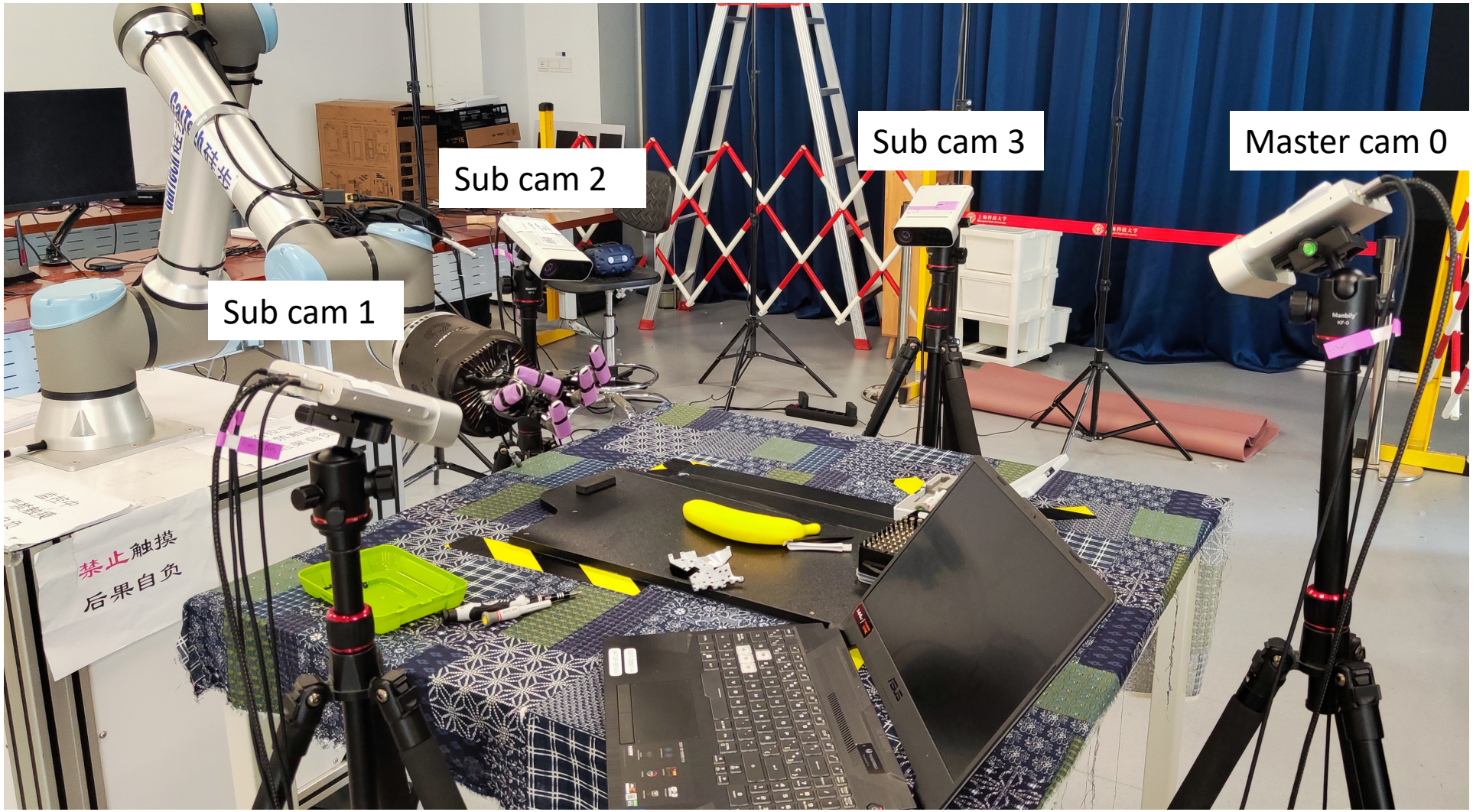


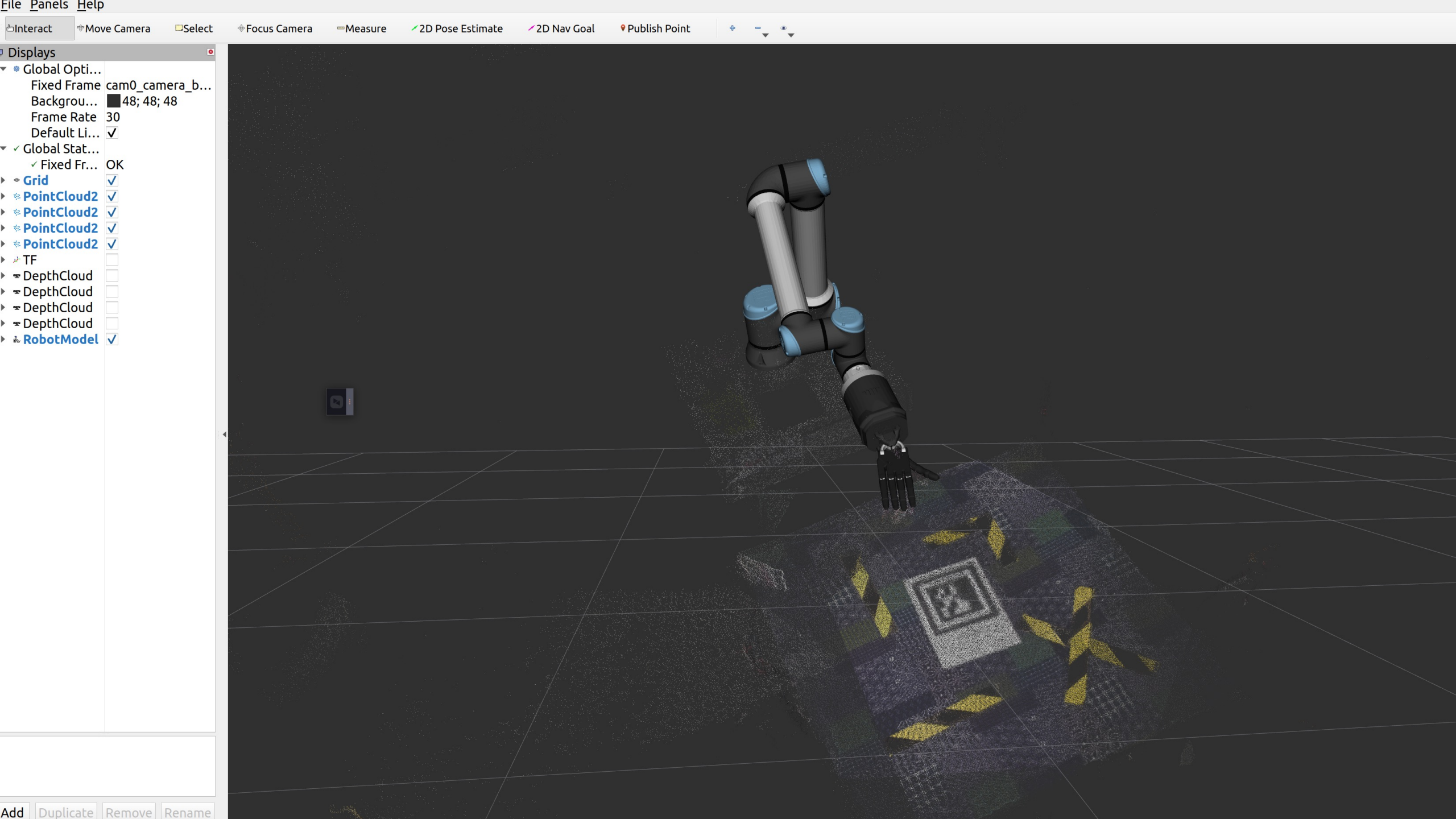
Fig. 7. A 3D Lidar scan colored by all the 7 horizontal cameras. The transformations between each camera and 3D Lidar are acquired from global optimization result. All the green points represent areas where no camera is overlapping with the pointcloud.

Mapping Robot:
2 LiDAR sensor's point clouds in one view



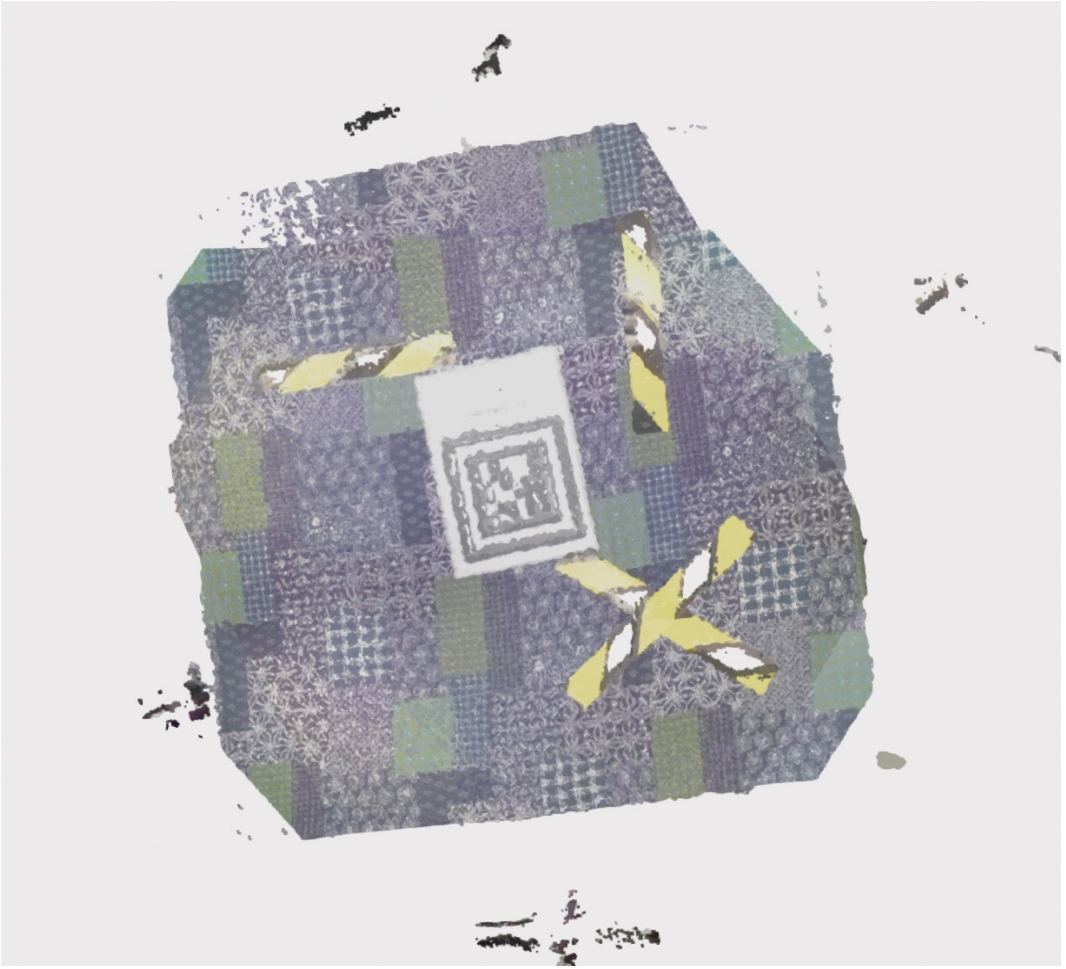
Shadowhand Experiment Example




File Panels Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Displays
Global Opti...
Fixed Frame cam0_camera_b...
Backgrou... 48; 48; 48
Frame Rate 30
Default Li...
Global Stat...
Fixed Fr... OK
Grid
PointCloud2
PointCloud2
PointCloud2
PointCloud2
TF
DepthCloud
DepthCloud
DepthCloud
DepthCloud
RobotModel
Add
Duplicate
Remove
Rename

Calibration

- Multiple camera calibration
  - AprilTag for rough transformation
  - Corp, down-sample
  - Colored ICP to refine



Hand eye calibration

Goal: get the transformation between master camera and robot base_link

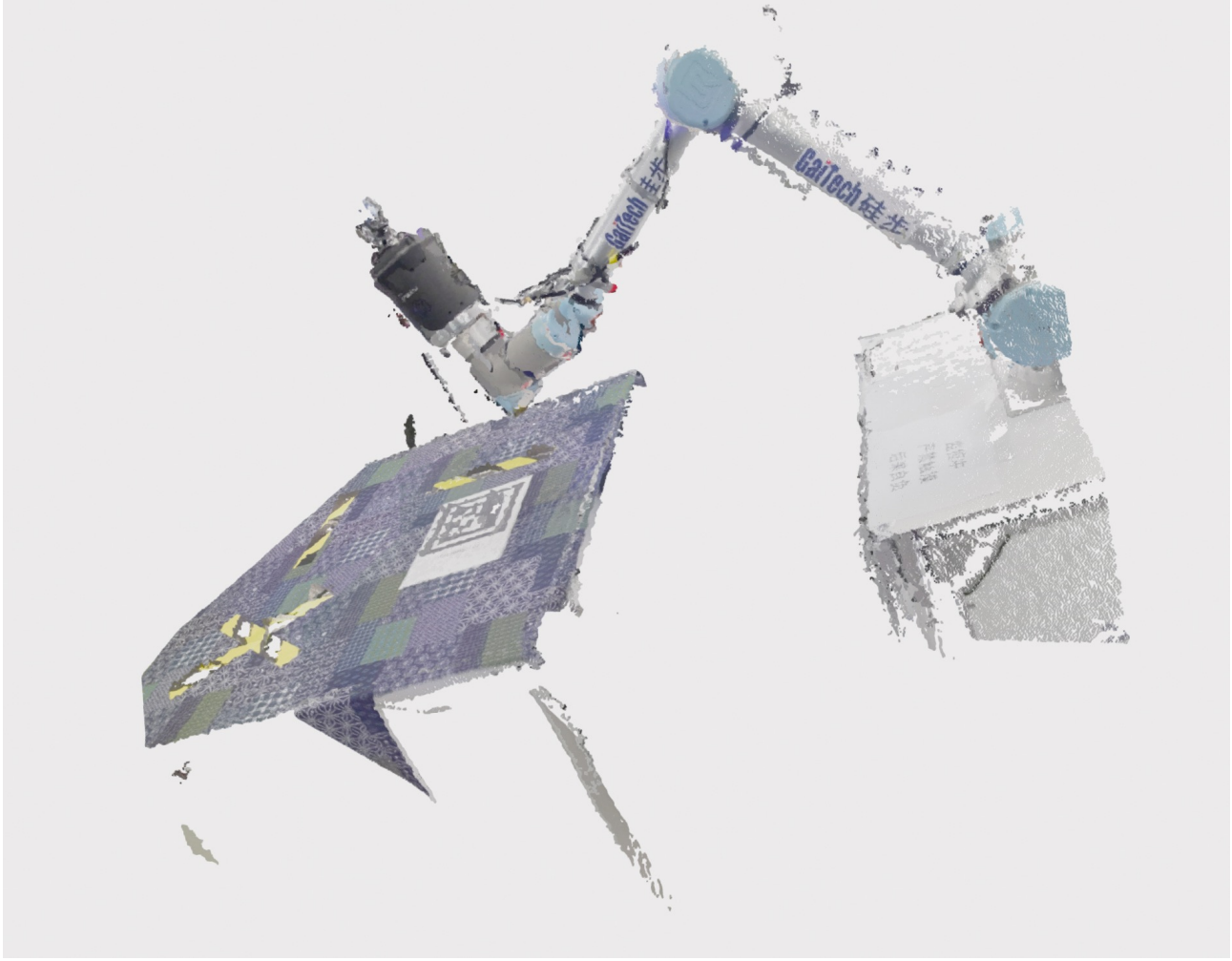
Rough calibration: easy_handeye

- Use robot hand hold the apriltag
- collect apriltag poses while moving the arm to different poses (do not move the hand)
- get handeye transformation by solving equation

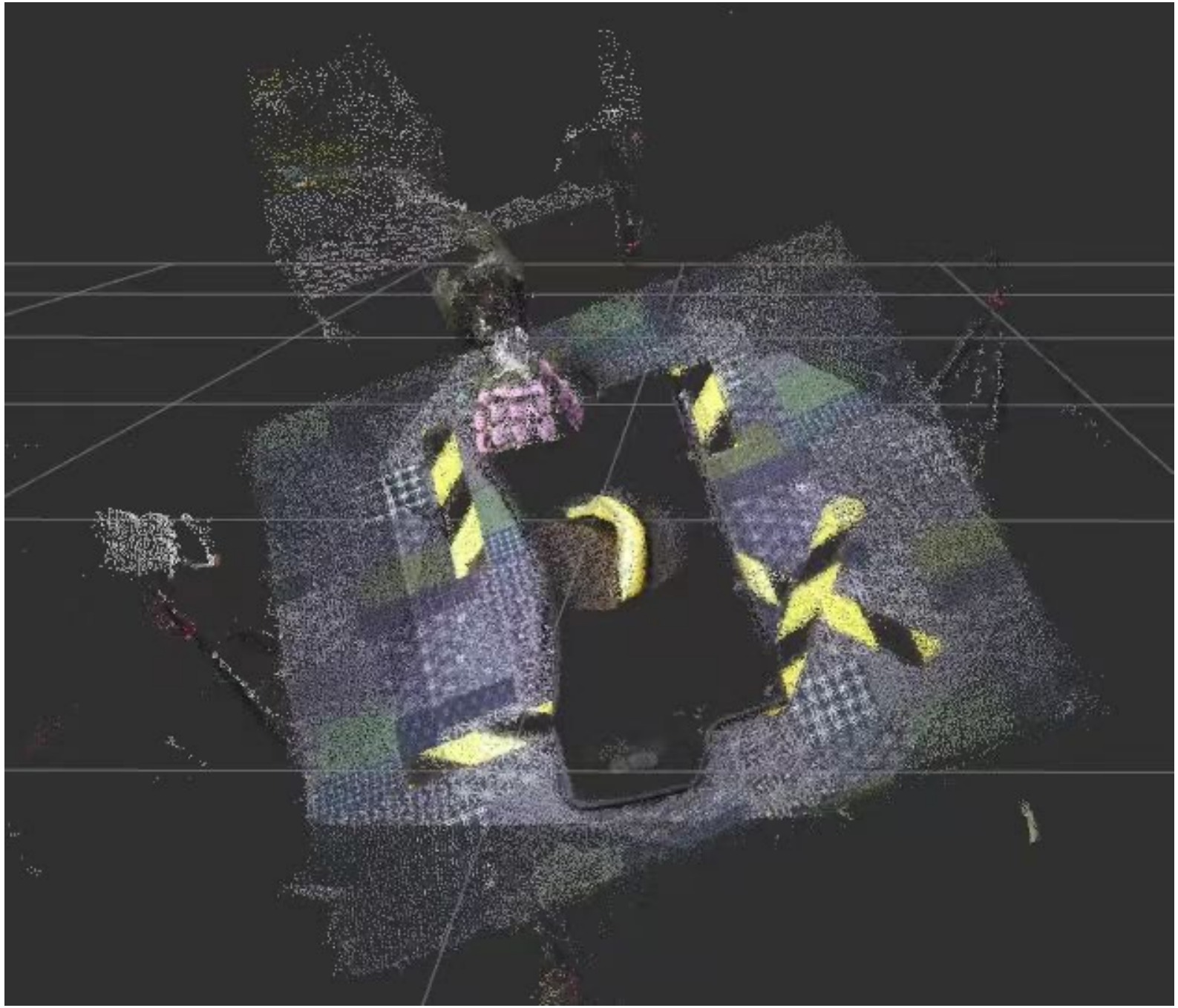
Hand eye calibration:

Refinement: icp

- turn one of the sub-camera to see most of the arm ,
- sample a pointcloud from corresponding robot mesh
- align the above two pointcloud, and get the transformation between the sub-camera and robot base_link
- get the goal transformation by multipyling sub-camera to robot transformation and sub_camera to master camera transformation



Banana from 4 fused point clouds



VISUAL SERVOING

Definition of Visual Servoing (VS)

“ VS is the use of computer *vision* data in the servo loop that *controls* the motion of a robot ”

“ VS is the action taken by a *vision-based control* ”

“ VS is the way to provide a control algorithm with *visual feedback* to reach a desired target ”

Material derived from Antonio Paolillo
https://totopaolillo.github.io/documents/20210825_summerschool_innsbruck.pdf
and Seth Hutchinson
https://dellaert.github.io/21S-8803MM/Readings/L7%20Visual%20Servo%20Control.pdf

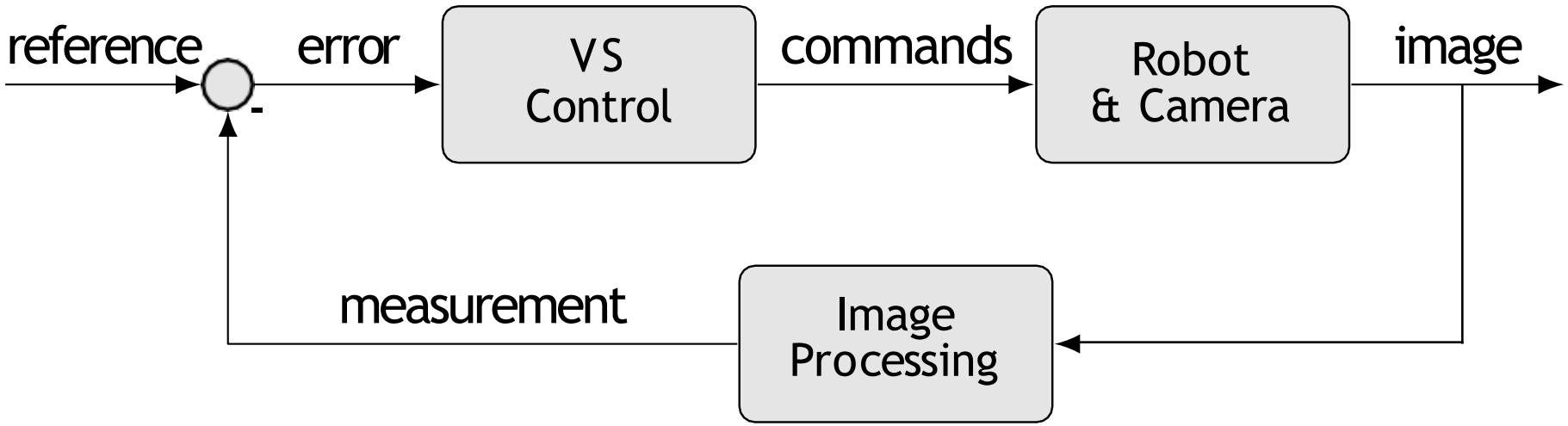
Visual Servoing: The Basic Problem

- A camera views the scene from an initial pose, yielding the current image.
- The desired image corresponds to the scene as viewed from the desired camera pose.
- Determine a camera motion to move from initial to desired camera pose, using the time-varying image as input.

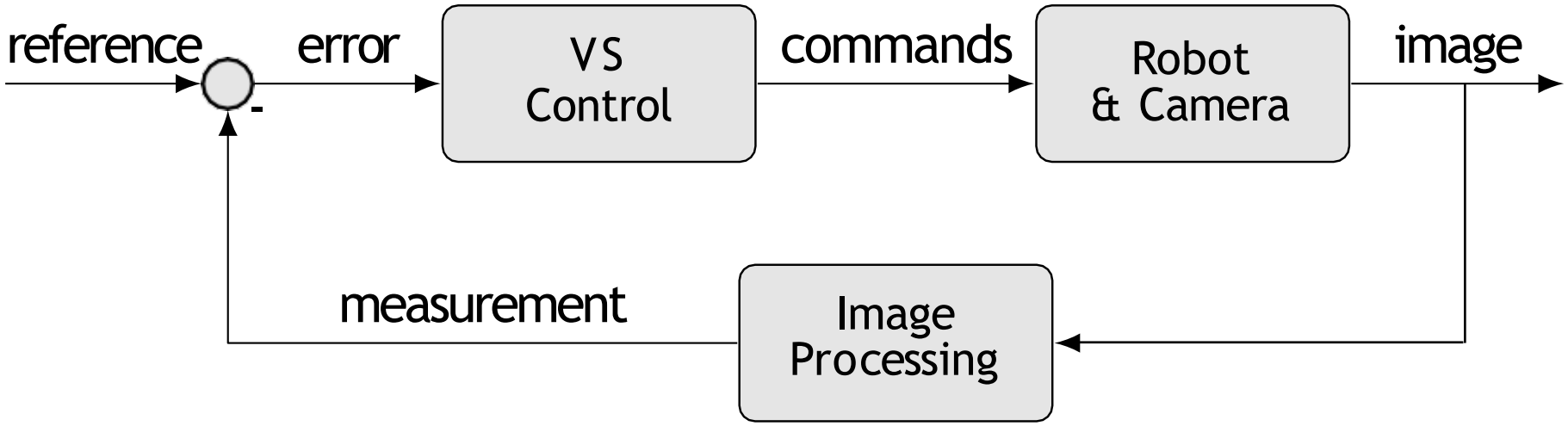
There are many variations on the problem:

- Eye-in-hand vs. fixed camera
- Which image features to use
- How to specify desired images for specified tasks
- Etc…

Block diagram & scheme classification



Block diagram & scheme classification



Two main VS schemes:

1. Position-based visual servoing (*PBVS*)
   - — More complicated image processing (need to reconstruct a pose)
   - \+ Relatively easier control law
2. Image-based visual servoing (*IBVS*)
   - \+ Easier image processing (it is a features extraction)
   - — More complicated control law

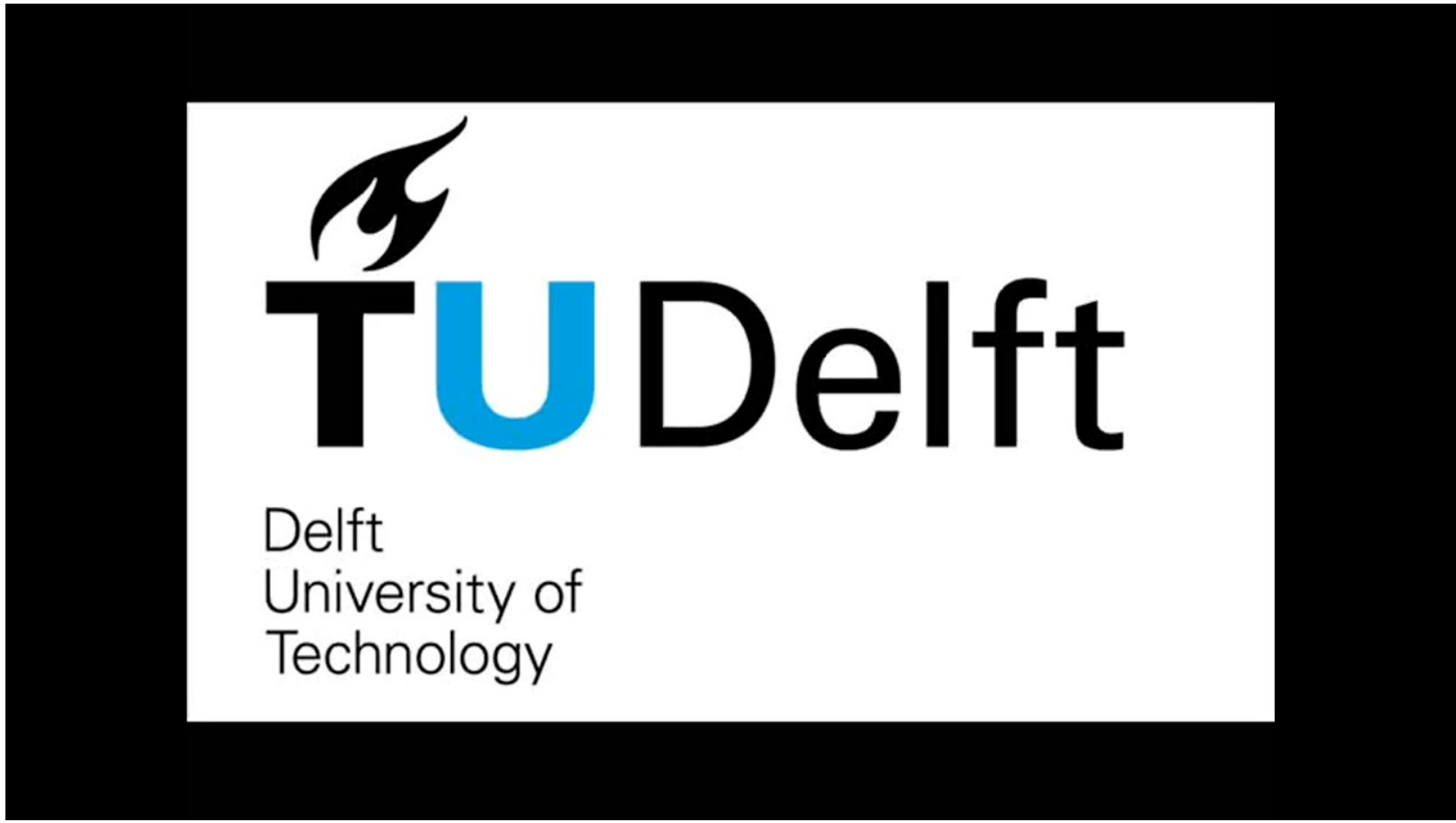
► Other options are also possible, such as *2.5D VS*

Simple example: 2D image based visual servoing

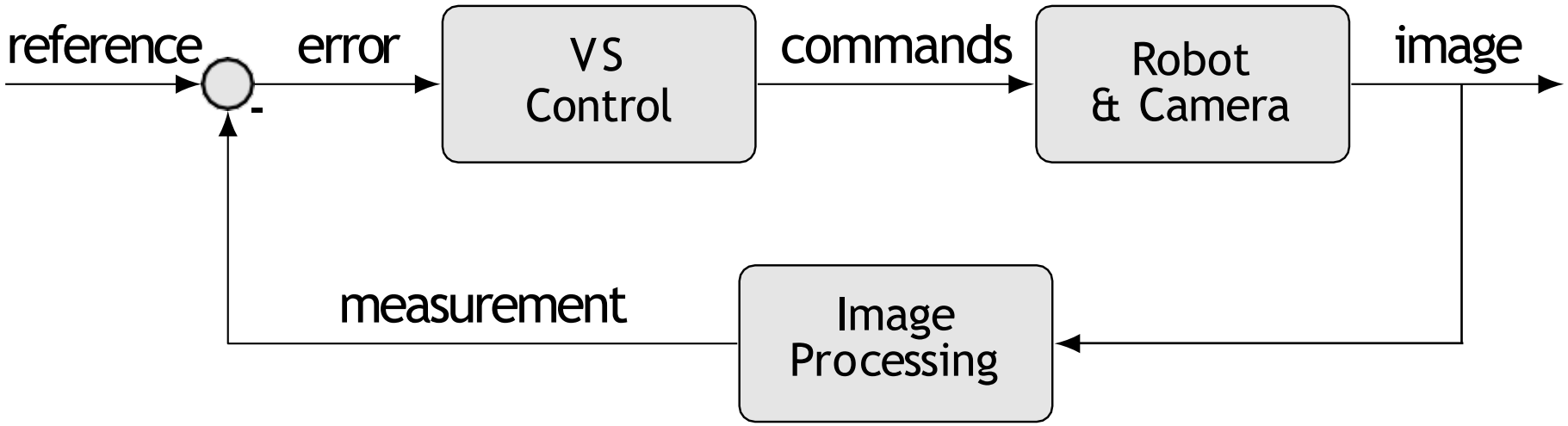
- If goal is above center, go up;
- if goal below center, go down,
- if goal is right, go right,
- if goal is left, go left

- Example (working not so well):
  - land drone on red dot =>
  - if goal near center, go down

- Need more complex VS for:
  - More DoF actuators (e.g. 6 DoF arm)
  - 3D rotated visual features (perspective change!)



Position Based Visual Servoing (PBVS): block diagram



reference	desired camera pose	$\mathbf{s}^*$
measurement	current camera pose	$\mathbf{s}$
error	Cartesian error	$\mathbf{e} = \mathbf{s} - \mathbf{s}^*$
commands	velocities command	$\mathbf{v}$

PBVS and the camera/ object pose reconstruction problem

- PBVS implies the reconstruction of the camera goal pose, which is normally a complex task
  - Often: camera goal pose w.r.t. detected object pose
  - Or: camera goal pose w.r.t. world coordinate frame => localization problem

- A number of modules can be used
  - Model-based pose reconstruction modules
  - Fiducial marker detectors (e.g., April tag)
  - Visual Odometry
  - Visual Simultaneous Localization and Mapping (V-SLAM)
  - Machine learning-based approaches, such as self-supervised learning

PBVS: Basic idea

- Camera goal coordinates to joint configuration space: Inverse Kinematics
- Control from current joint configuration to goal configuration:
  - Differential equations on the errors => control theory OR
  - Interpolate (but: be aware of colisions!) OR
  - Plan a path (e.g. RRT) (not really “servoing” anymore)

https://www.youtube.com/watch?v=miYfyCduVzQ

https://www.youtube.com/watch?v=veAIuLdbgJU

Image Based Visual Servoing (IBVS)

- You do not have a 3D goal pose/ position of the camera
- Instead: goal image coordinates of certain image features
- Goal: move camera such that the features in camera image are at the goal positions

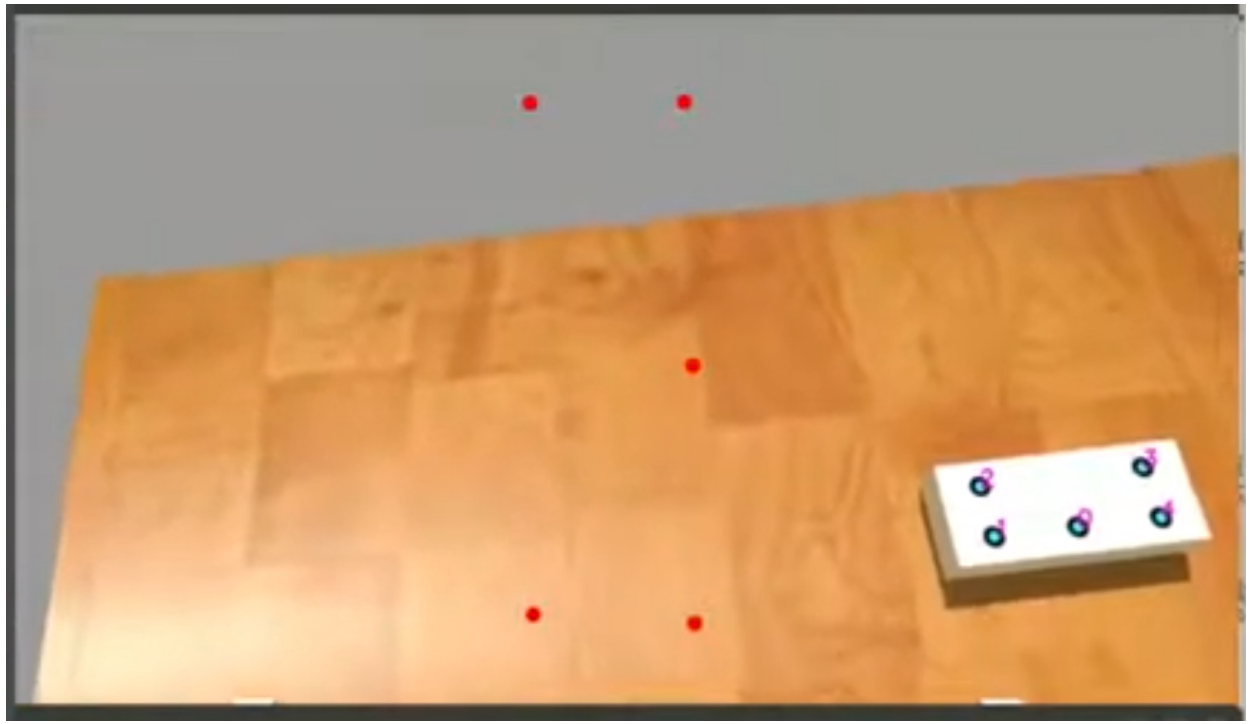
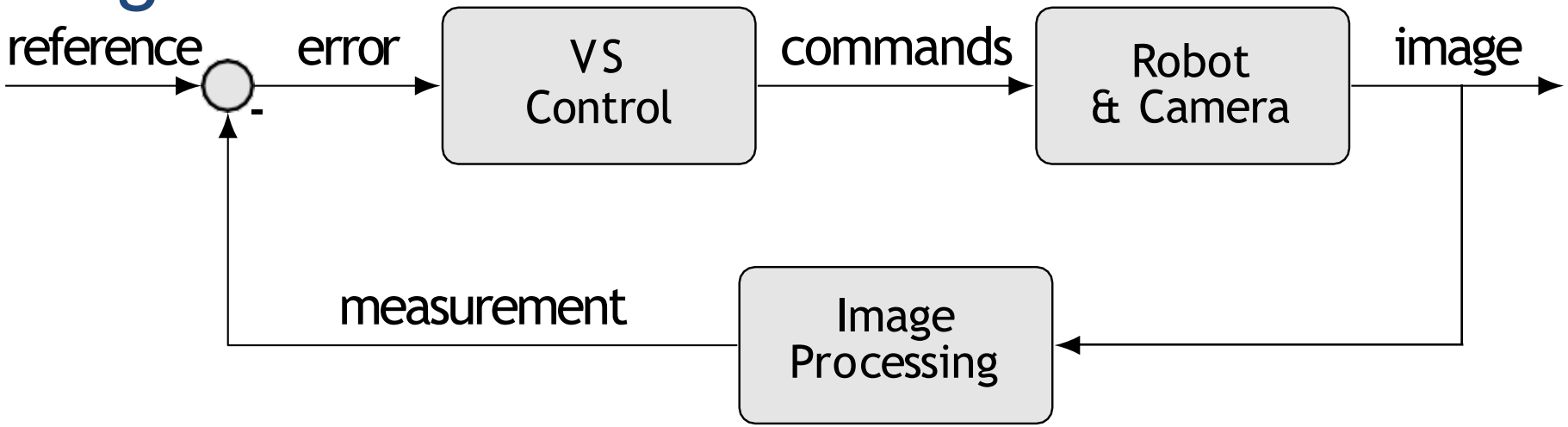


Image Based Visual Servoing (IBVS): block diagram



Most of the lecture focuses on *IBVS*

reference	desired visual features	$\mathbf{s}^*$
measurement	current visual features	$\mathbf{s}$
error	visual error	$\mathbf{e} = \mathbf{s} - \mathbf{s}^*$
commands	velocities command	$\mathbf{v}$

Definition of visual feature

- In computer vision, it is the set of pixels for which the *link* between photometric measurement and geometric primitives can be established
- It is the attempt to *summarize* the richness of data coming from the camera video stream
  - Be aware of the *information loss* that this "summary" involves
- It is the *gist* of the scene needed to control the robot
- It is the summary information got from the captured image, needed to close the VS loop and achieve a desired robotic behavior

Some Basic Assumptions

Numerous considerations when designing a visual servo system – consider only systems that satisfy the following basic assumptions:

- Eye-in-hand systems — the camera is mounted on the end effector of a robot and treated as a free-flying object with configuration space Q = SE(3).
- Static (i.e., motionless) targets.
- Purely kinematic systems — we do not consider the dynamics of camera motion, but assume that the camera can execute accurately the applied velocity control.
- Perspective projection — the imaging geometry can be modeled as a pinhole camera.

Some or all of these may be relaxed in more advanced applications.

Why visual features?

Consider the task of looking at the red object

captured image
240 × 320 *pixels*

Why visual features?

Consider the task of looking at the red object

captured image
240 × 320 *pixels*

$$\begin{bmatrix} \begin{bmatrix}157\\182\\202\end{bmatrix} & \begin{bmatrix}157\\182\\202\end{bmatrix} & \begin{bmatrix}157\\182\\202\end{bmatrix} & \cdots & \begin{bmatrix}82\\106\\130\end{bmatrix} \\ \begin{bmatrix}157\\182\\202\end{bmatrix} & \begin{bmatrix}157\\182\\202\end{bmatrix} & \begin{bmatrix}157\\182\\202\end{bmatrix} & \cdots & \begin{bmatrix}82\\106\\130\end{bmatrix} \\ \vdots & & & \ddots & \vdots \\ \begin{bmatrix}51\\61\\71\end{bmatrix} & \begin{bmatrix}51\\61\\71\end{bmatrix} & \begin{bmatrix}50\\60\\70\end{bmatrix} & \cdots & \begin{bmatrix}29\\41\\53\end{bmatrix} \end{bmatrix}$$

how it looks like in the PC
240 × 320 × 3 *matrix of numbers*

Why visual features?

Consider the task of looking at the red object

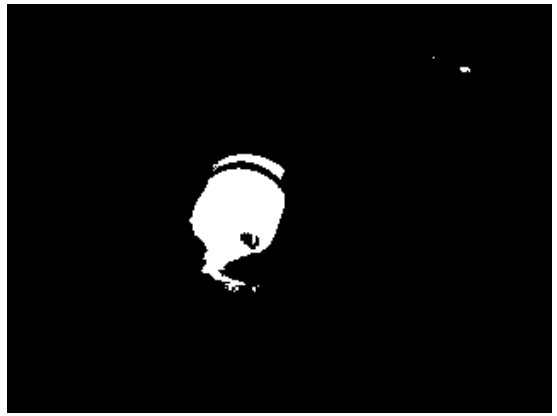
captured image
240 × 320 *pixels*

coordinates of the object centroid
2 *scalar numbers*

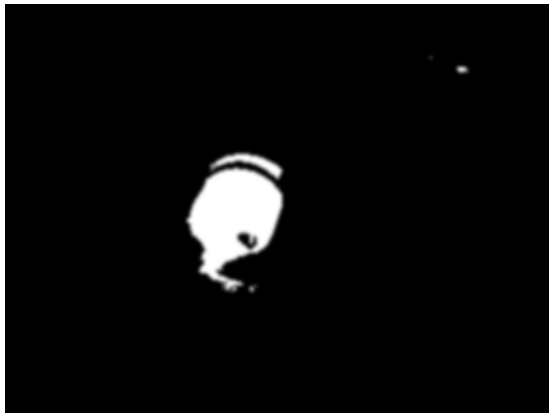
An example of image processing algorithm

- Computer vision community provides many *ready-to-use* tools

original image

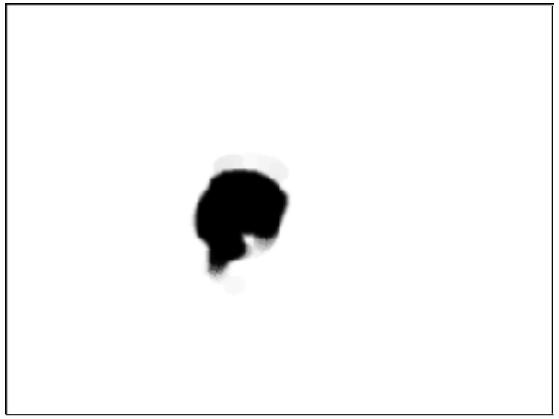


color filtering



smoothing

erode & dilate

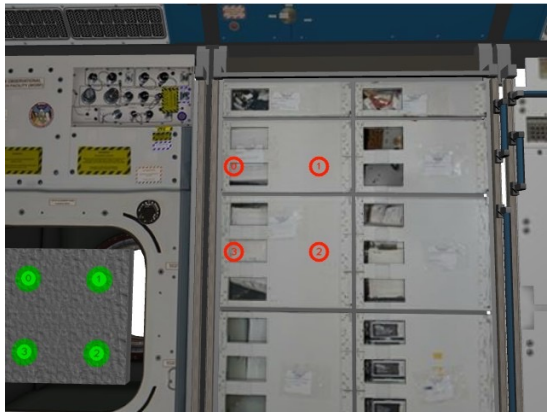


inversion

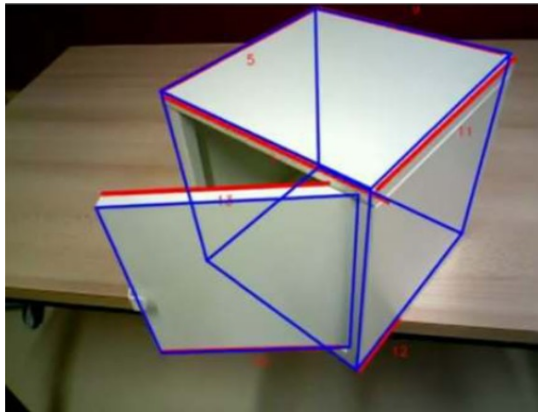
blob detection

- All these operations are available in the *opencv library*, for example

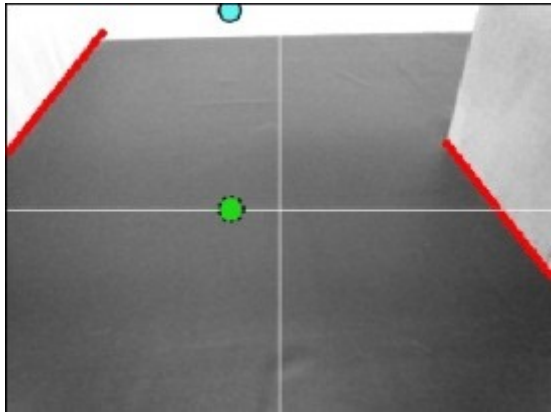
Examples of visual features



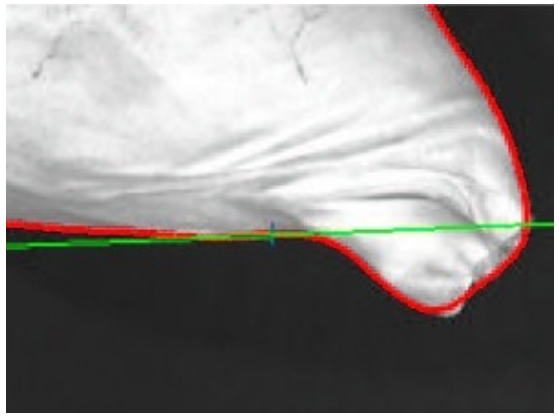
points



lines



reconstructed points



countours

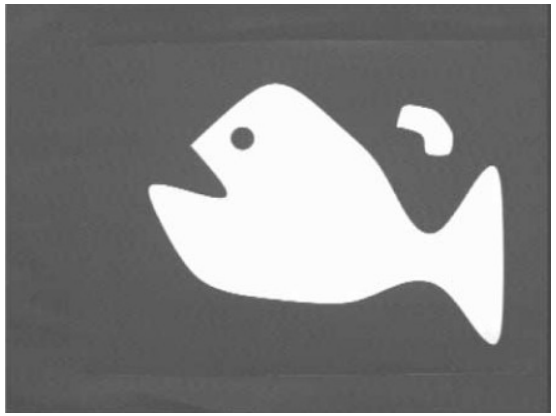
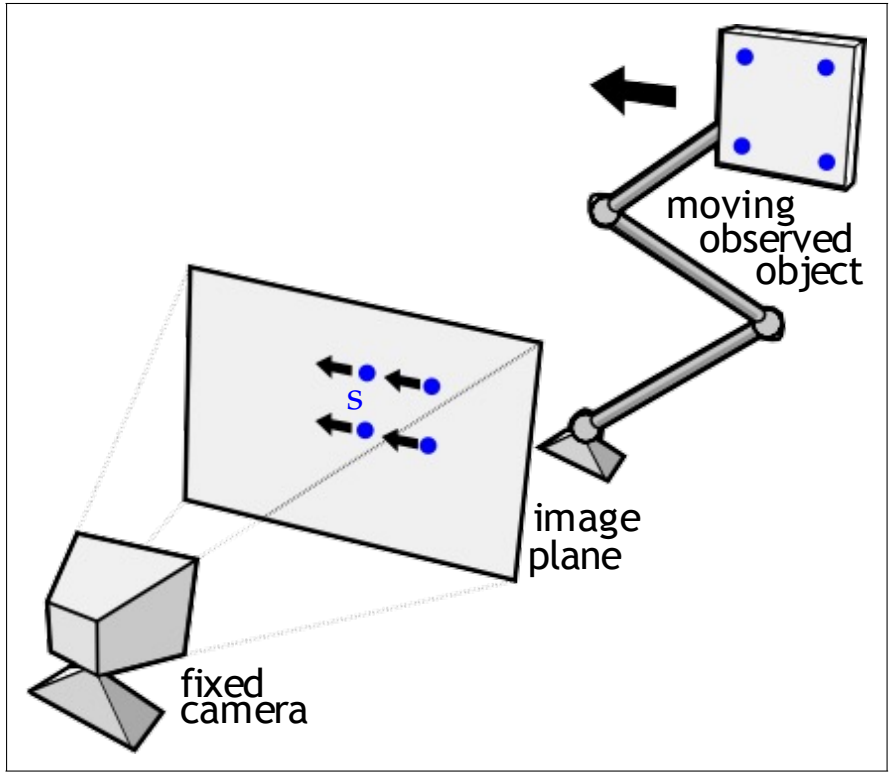


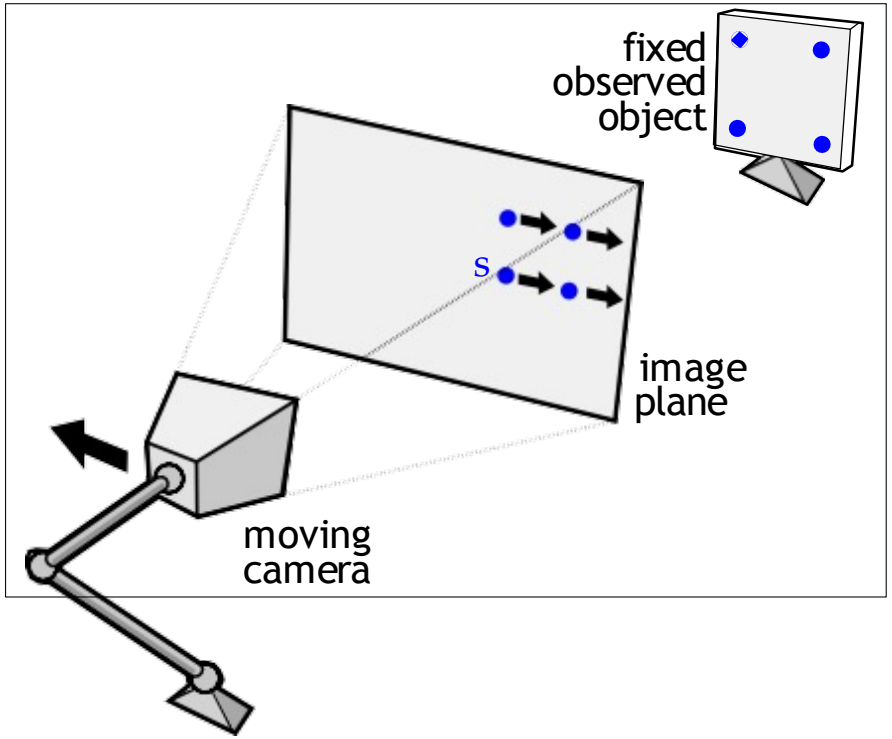
image moments

pixel luminance

In this lecture we focus on *point visual features,* e.g. SIFT feature points

Eye-to-hand & eye-in-hand configuration

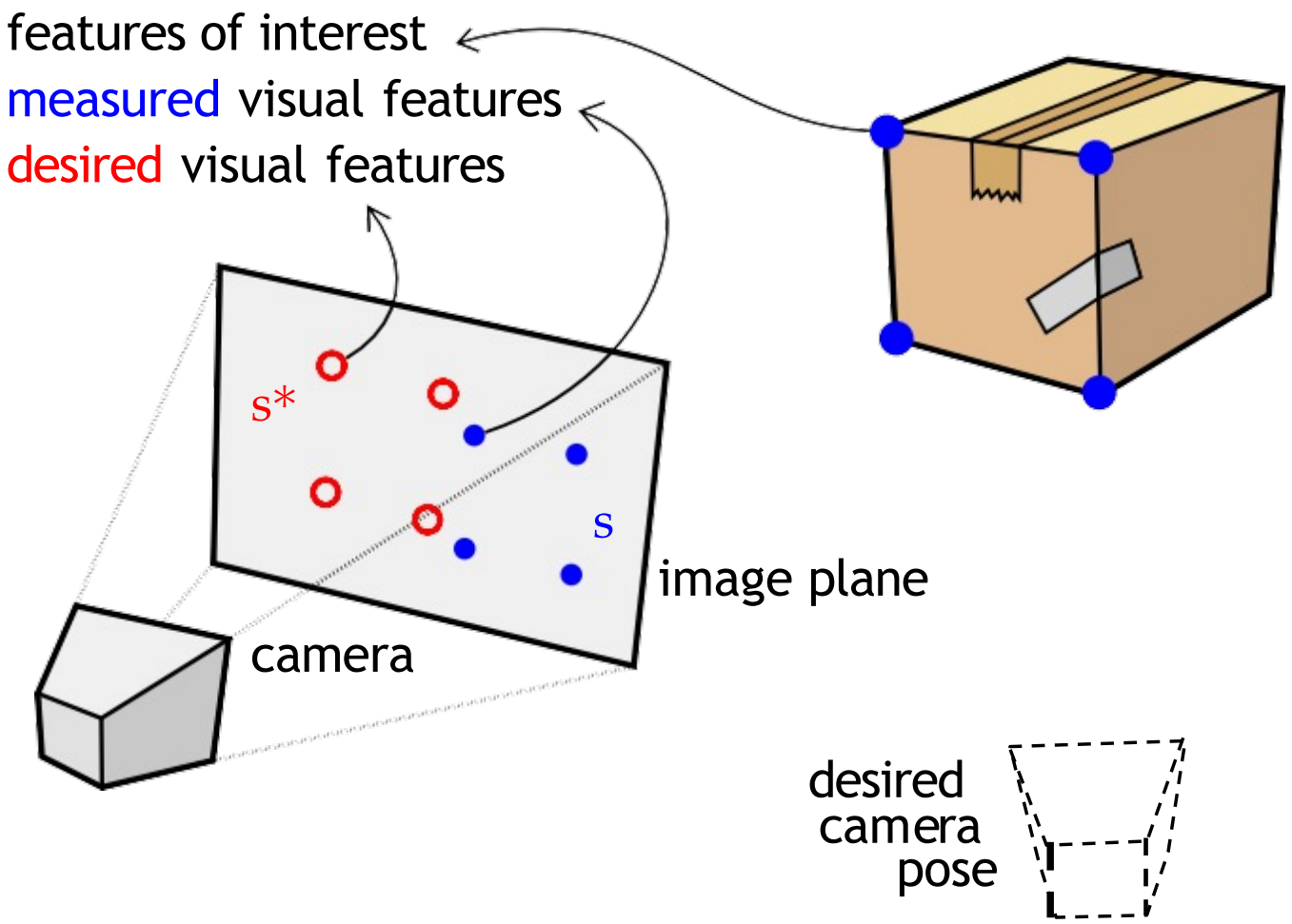




- *Eye-to-hand*: actuated target observed by a camera (left)
- *Eye-in-hand*: actuated camera observing a target (right)

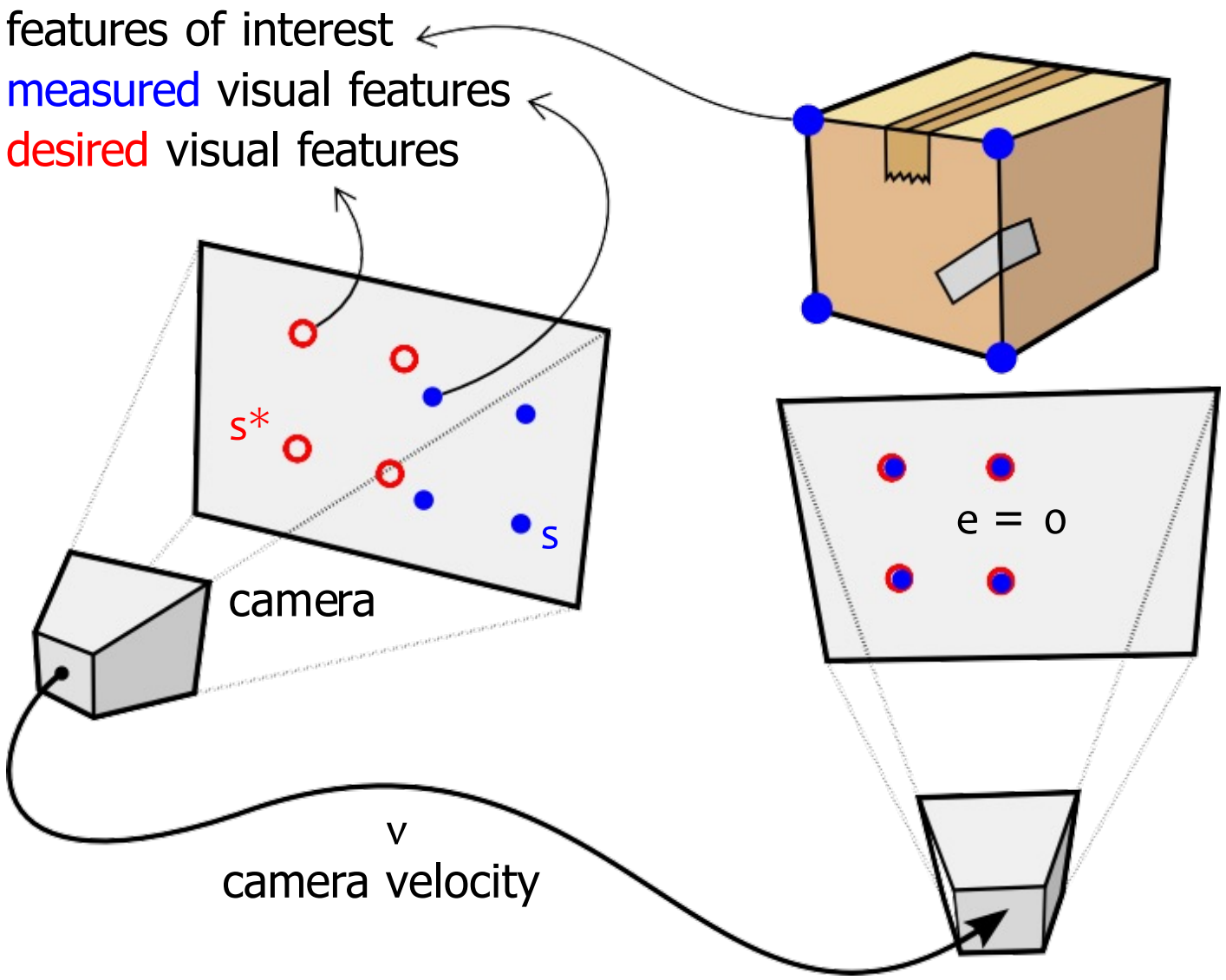
In this lecture we focus on *eye-in-hand* configurations

Working principle (with a hand-held camera)



The high-level task consists in moving the camera to a desired pose

Working principle (with a hand-held camera)



The cartesian task is actually translated in a *visual task*

Designing the Control Law --- The Basic Idea

- Given s, control design can be quite simple:

- E.g. velocity controller with relationship between time variation s and camera velocity v = (ν, ω)
  - ν: instantaneous linear velocity of the origin of the camera frame
  - ω: instantaneous angular velocity of the camera frame

Computing the VS control law

The VS control law is obtained in three steps

1. *Model design*: the features motion is related to the camera motion as

$$\dot{\mathbf{s}} = \mathbf{L}\mathbf{v} \tag{1}$$

where **L** is the *interaction matrix (aka. image Jacobian)*

2. *Stable error dynamics*: we want $\mathbf{s} \rightarrow \mathbf{s}^*$, that is $\mathbf{e} = (\mathbf{s} - \mathbf{s}^*) \rightarrow \mathbf{0}$

$$\dot{\mathbf{e}} = \dot{\mathbf{s}} - \dot{\mathbf{s}}^* = -\lambda \mathbf{e}, \quad \lambda > 0 \tag{2}$$

where λ is the *control gain*

3. *Controller computation*: (1) in (2) with a constant target ($\dot{\mathbf{s}}^* = \mathbf{0}$)

$$\dot{\mathbf{e}} = -\lambda \mathbf{e} = \dot{\mathbf{s}} = \mathbf{L}\mathbf{v} \Longrightarrow \boxed{\mathbf{v} = -\lambda \mathbf{L}^+ \mathbf{e}}$$

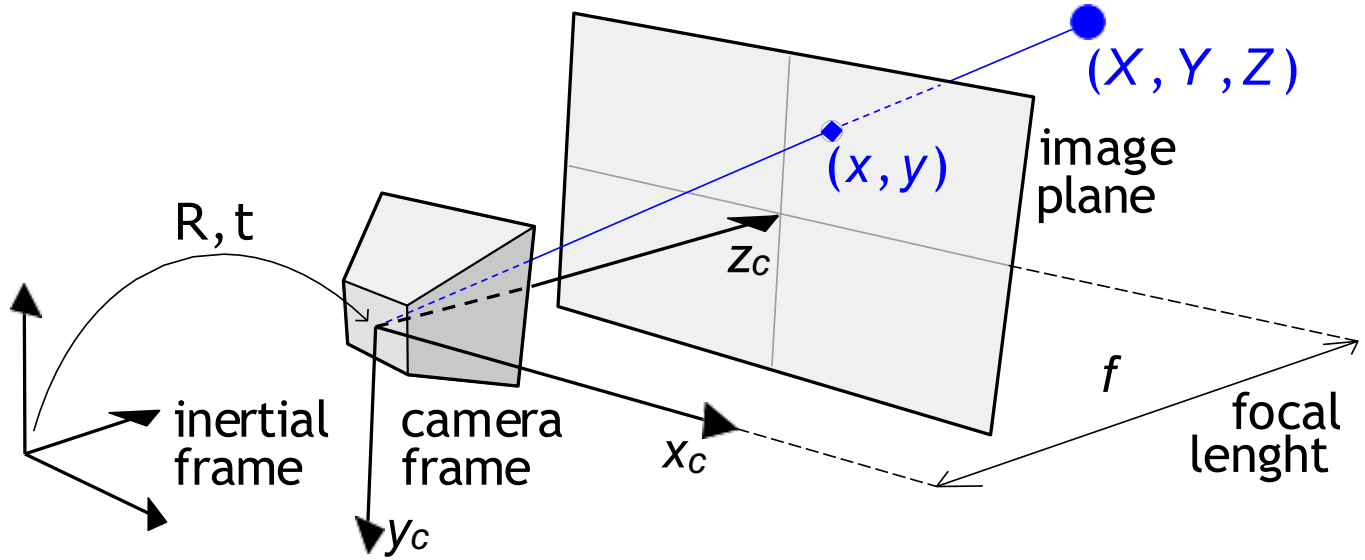
$\mathbf{L}^+$: Moore-Penrose pseudo-inverse: $\mathbf{L}^+ = (\mathbf{L}^T \mathbf{L})^{-1} \mathbf{L}^T$

Visual Servoing Bigger Picture

- In practice, it is impossible to know the exact value of L or of L + , since these depend on measured data.
- Learning, planning, perception and action are often tightly coupled activities.
- Visual servo control is the coupling of perception and action — hand-eye coordination.
- Basic visual servo controllers can serve as primitives for planning algorithms.
- There are a number of analogies between human hand-eye coordination and visual servo control.
- A rigorous understanding of the performance of visual servo control systems provides a foundation for sensor-based robotics.

Camera projection model (1/5)

- *Frontal pin-hole* camera model



- *Perspective projection*

$$x = f\frac{X}{Z}, \quad y = f\frac{Y}{Z}$$

- Sometimes *normalized coordinates* are used, considering $f = 1$
- Also used to computed the interaction matrix

Camera projection model (2/5)

- In a more compatc way, using *homogeneous coordinates*:

$$Z \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

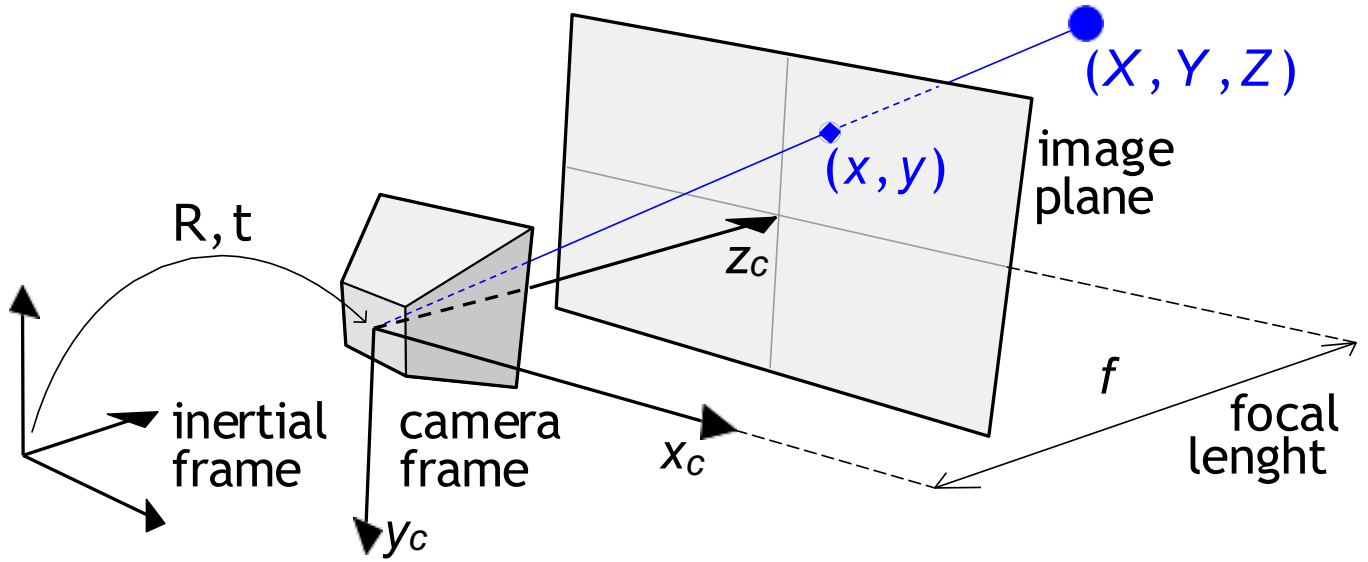
- The *depth* Z is *unkown* (remember the lost of information): we call it as *parameter* ζ in the left-hand side of the equation
- For convenience, we write the matrix as

$$\begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

- In general, the Cartesian point can be expressed in the *inertial frame*

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0\,0\,0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \\ 1 \end{pmatrix}$$

Camera projection model (3/5)



- The *camera ideal model* results to be

$$\zeta \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0\,0\,0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \\ 1 \end{pmatrix}$$

Camera projection model (4/5)

- However, the features are *measured in pixels*, with coordinates (u, v), which are related to (x, y) through the following relationship

$$u = u_0 + \frac{x}{\rho}, \quad v = v_0 + \frac{y}{\rho}$$

where (ρ_w, ρ_h) is the size of the pixel and (u_0, v_0) is the central point

- Using homogenous coordinates and writing in compact form:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} 1/\rho_w & 0 & u_0 \\ 0 & 1/\rho_h & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- Used to compute the interaction matrix

Camera projection model (5/5)

- Putting all together

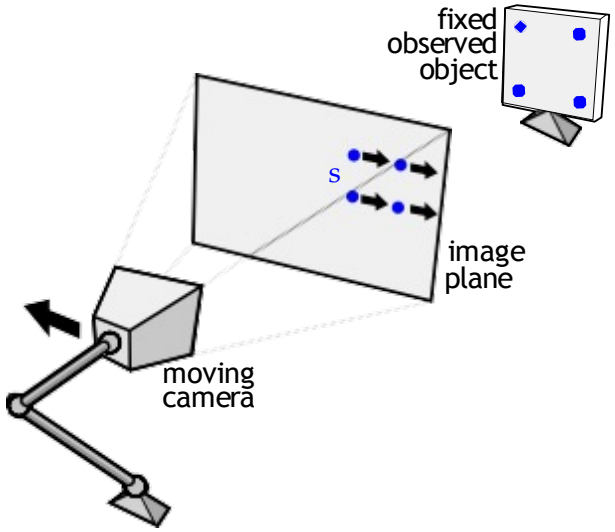
$$\underbrace{\zeta \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}}_{\tilde{\mathbf{p}}} = \overbrace{\begin{pmatrix} f/\rho_w & 0 & u_0 \\ 0 & f/\rho_h & v_0 \\ 0 & 0 & 1 \end{pmatrix}}^{\text{intrinsic}} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\mathbf{P}} \overbrace{\begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0\,0\,0 & 1 \end{pmatrix}^{-1}}^{\text{extrinsic}} \underbrace{\begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \\ 1 \end{pmatrix}}_{\tilde{\mathbf{P}}}$$

$$\underbrace{\mathbf{K}}_{\mathbf{K}} \underbrace{\mathbf{P}}_{\mathbf{P}} \underbrace{\left({}^0\mathbf{T}_c\right)^{-1}}_{{}^0\mathbf{T}_c^{-1}}$$

$$\tilde{\mathbf{p}} = \underbrace{\mathbf{K}\,\mathbf{P}\,\left({}^0\mathbf{T}_c\right)^{-1}}_{\mathbf{C}}\,\tilde{\mathbf{P}}$$

- **K** is called *intrinsic parameter matrix* or *calibration matrix*
- **P** is called *standard projection matrix*
- ${}^0\mathbf{T}_c$ is obtained with a *extrinsic calibration*
- **C** is called *camera matrix*
- f/ρ_w and f/ρ_h are the *focal lenght expressed in units of pixels*
- From $\tilde{\mathbf{p}}$ we obtain the model in pixels of our visual feature: $\mathbf{s} = (u, v)^T$

Computation of the interaction matrix (1/3)



- The interaction matrix relates the *velocity of the feature* to the *velocity of the camera*

$$\dot{\mathbf{s}} = \begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} = \mathbf{L}\mathbf{v} = \mathbf{L} \begin{pmatrix} \nu \\ \omega \end{pmatrix}$$

  - Remember: eye-in-hand configuration

- From the perspective equation we have

$$\dot{x} = f\frac{\dot{X}Z - X\dot{Z}}{Z^2} = \frac{f}{Z}\dot{X} - \frac{x}{Z}\dot{Z}, \quad \dot{y} = f\frac{\dot{Y}Z - Y\dot{Z}}{Z^2} = \frac{f}{Z}\dot{Y} - \frac{y}{Z}\dot{Z}$$

- In compact form:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} \frac{f}{Z} & 0 & -\frac{x}{Z} \\ 0 & \frac{f}{Z} & -\frac{y}{Z} \end{pmatrix} \begin{pmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{pmatrix}$$

Computation of the interaction matrix (2/3)

- The time derivative of the point expressed in Camera frame is related to the velocity of the camera:

$$\begin{pmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{pmatrix} = -\boldsymbol{\nu} - \boldsymbol{\omega} \times \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}$$

that is

$$\begin{pmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{pmatrix} = \begin{pmatrix} -1 & 0 & 0 & 0 & -Z & Y \\ 0 & -1 & 0 & Z & 0 & -X \\ 0 & 0 & -1 & -Y & X & 0 \end{pmatrix} \begin{pmatrix} \boldsymbol{\nu} \\ \boldsymbol{\omega} \end{pmatrix}$$

- Substituting:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} -\frac{f}{Z} & 0 & \frac{x}{Z} & \frac{xY}{Z} & -f - \frac{xX}{Z} & \frac{fY}{Z} \\ 0 & -\frac{f}{Z} & \frac{y}{Z} & f + \frac{yY}{Z} & -\frac{yX}{Z} & -\frac{fX}{Z} \end{pmatrix} \begin{pmatrix} \boldsymbol{\nu} \\ \boldsymbol{\omega} \end{pmatrix}$$

Computation of the interaction matrix (3/3)

- Considering that $X = xZ/f$ and $Y = yZ/f$:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} -\frac{f}{Z} & 0 & \frac{x}{Z} & \frac{xy}{f} & -f - \frac{x^2}{f} & y \\ 0 & -\frac{f}{Z} & \frac{y}{Z} & f + \frac{y^2}{f} & -\frac{xy}{f} & -x \end{pmatrix} \begin{pmatrix} \nu \\ \omega \end{pmatrix}$$

- From the metric-pixel conversion we have that

$$\dot{u} = \dot{x}/\rho_w, \quad \dot{v} = \dot{y}/\rho_h$$

$$x = (u - u_0)\rho_w = \bar{u}\rho_w, \quad y = (v - v_0)\rho_h = \bar{v}\rho_h$$

- Substituting:

$$\begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} = \underbrace{\begin{pmatrix} -\frac{f}{\rho_w Z} & 0 & \frac{\bar{u}}{Z} & \frac{\bar{u}\bar{v}\rho_h}{f} & -f - \frac{\bar{u}^2\rho_w}{f} & \bar{v} \\ 0 & -\frac{f}{\rho_h Z} & \frac{\bar{v}}{Z} & f + \frac{\bar{v}^2\rho_h}{f} & -\frac{\bar{u}\bar{v}\rho_h}{f} & -\bar{u} \end{pmatrix}}_{\mathbf{L}} \begin{pmatrix} \nu \\ \omega \end{pmatrix}$$

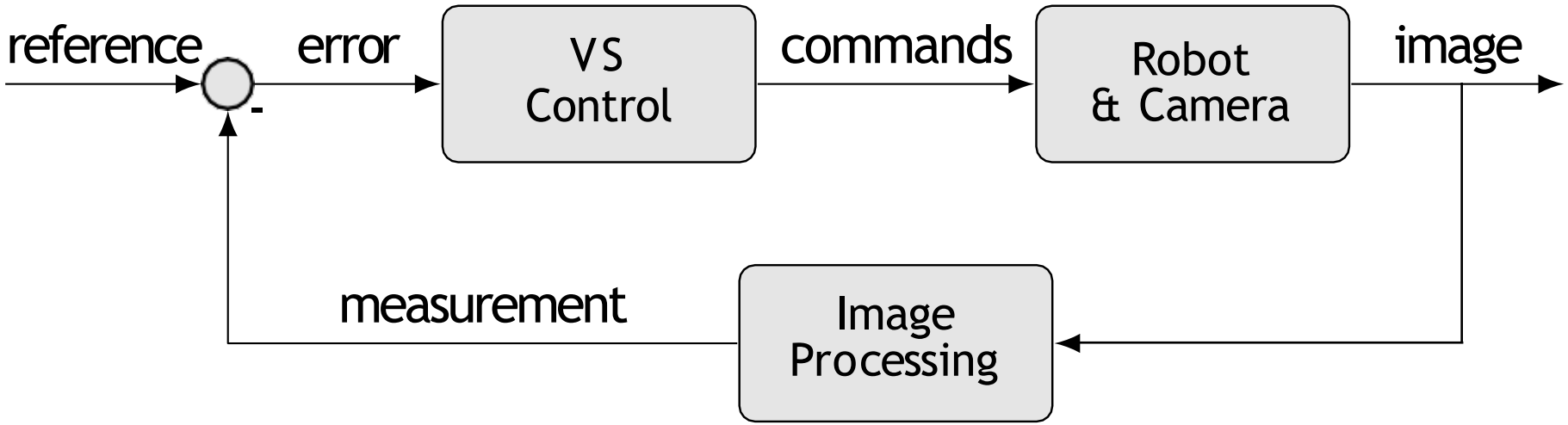
(Some) practical aspects of VS

- One point is not enough to uniquely determine the pose of the camera at convergence
- For example, if we want to control the motion of the camera in the 3D space, at least three points have to be used
- This means that the information used in the control law is the stack of three sets:

$$\mathbf{s} = \begin{bmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \mathbf{s}_3 \end{bmatrix}, \quad \mathbf{s}^* = \begin{bmatrix} \mathbf{s}_1^* \\ \mathbf{s}_2^* \\ \mathbf{s}_3^* \end{bmatrix}, \quad \mathbf{L} = \begin{bmatrix} \mathbf{L}_1 \\ \mathbf{L}_2 \\ \mathbf{L}_3 \end{bmatrix}$$

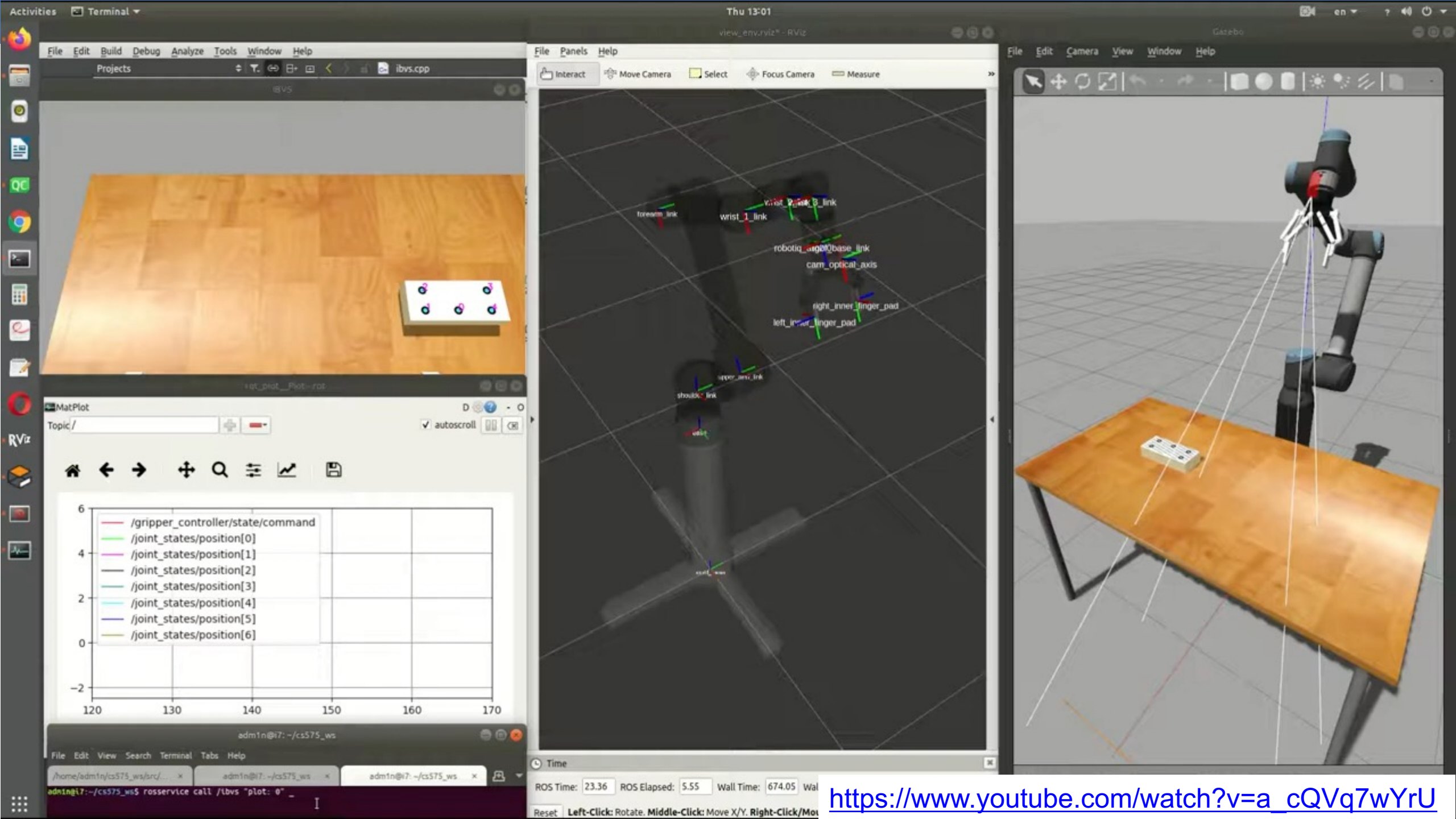
- The choise of the visual features, their number, and their desired value is part of the algorithm design

IBVS block diagram

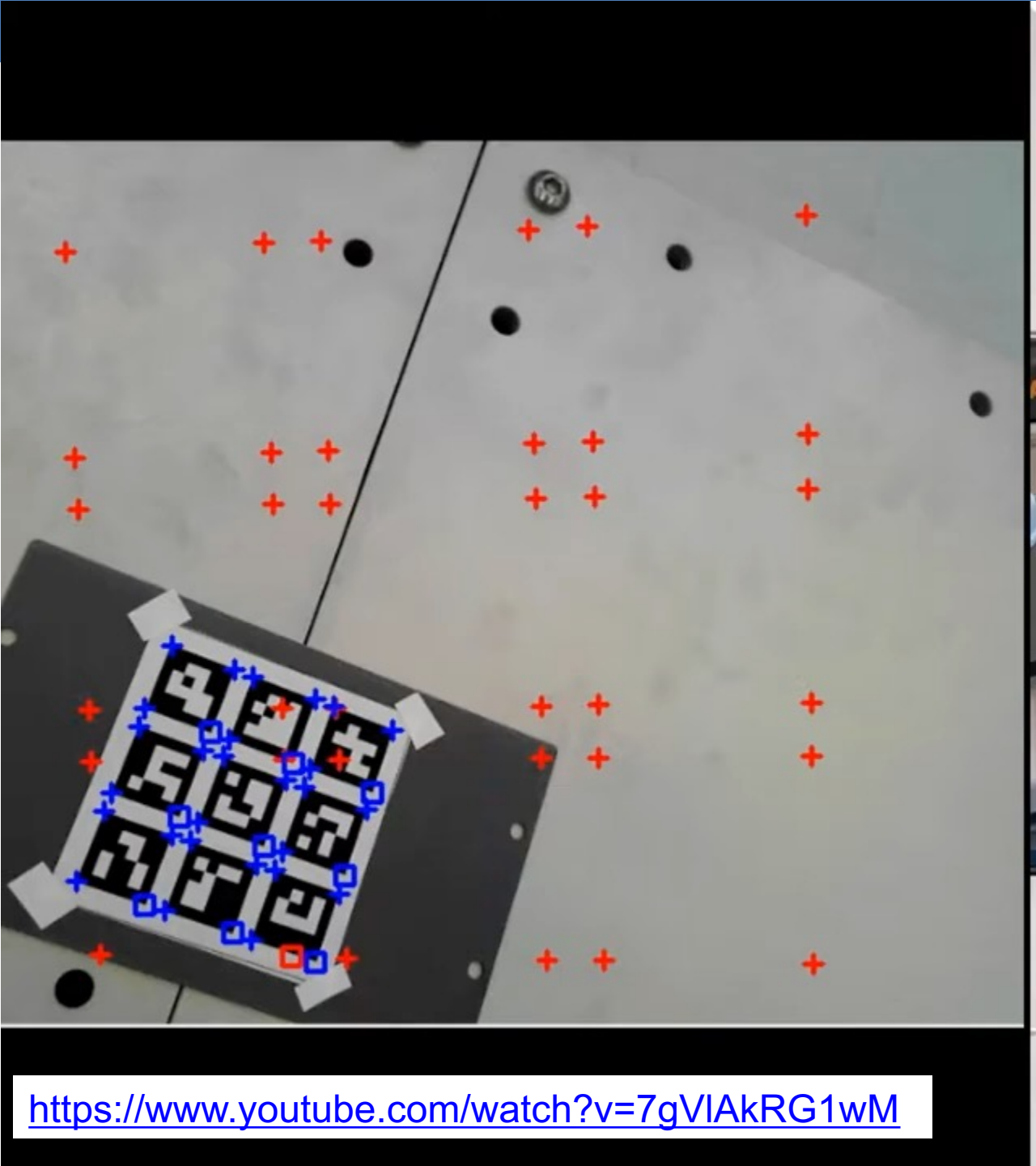


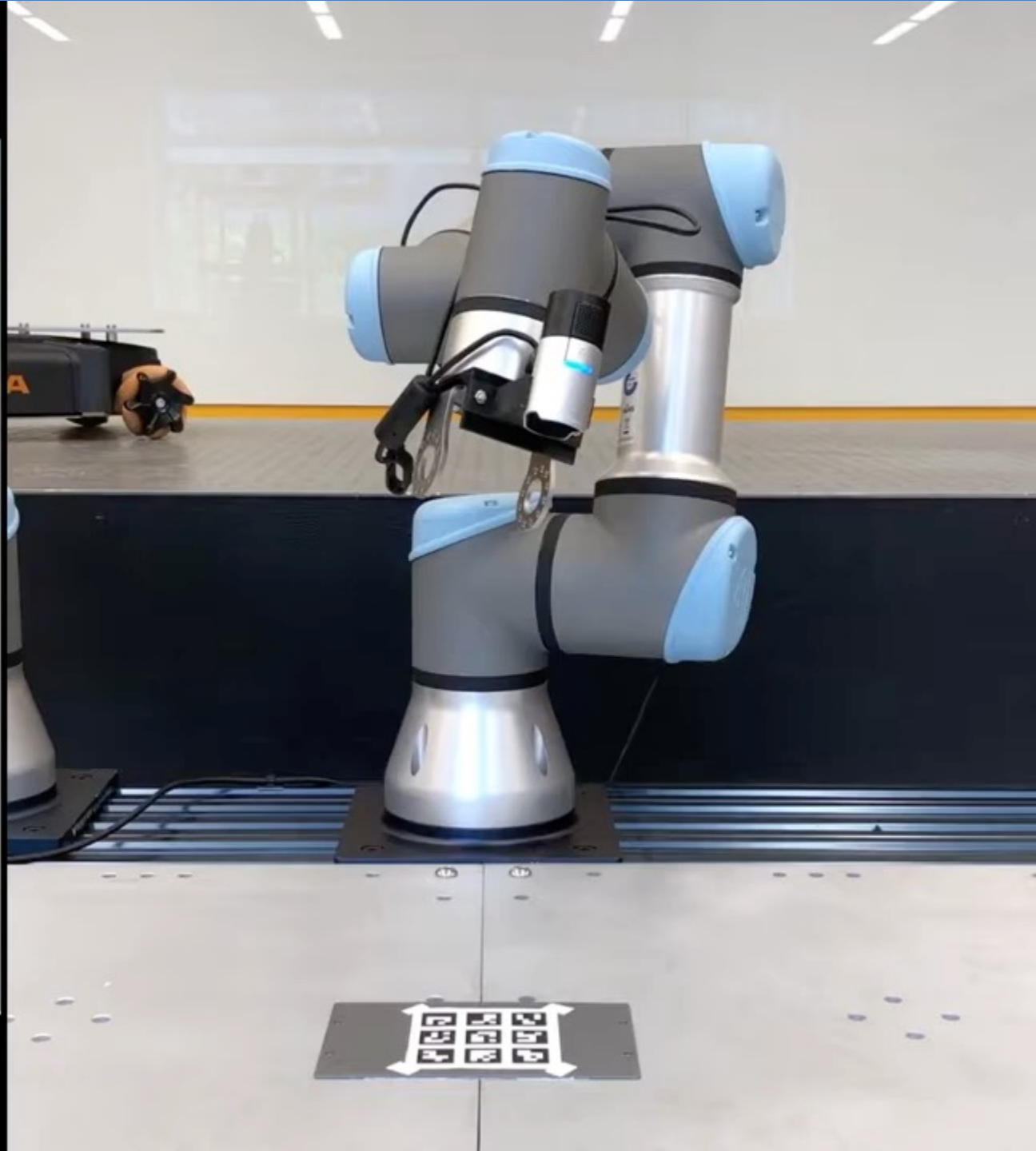
reference	desired visual features	$\mathbf{s}^*$
measurement	current visual features	$\mathbf{s}$
error	visual error	$\mathbf{e} = \mathbf{s} - \mathbf{s}^*$
commands	velocities command	$\mathbf{v} = -\lambda \mathbf{L}^+ \mathbf{e}$

λ is the control gain; L^+: Moore-Penrose pseudo-inverse: $L^+ = (L^T L)^{-1} L^T$



https://www.youtube.com/watch?v=a_cQVq7wYrU





https://www.youtube.com/watch?v=7gVlAkRG1wM