

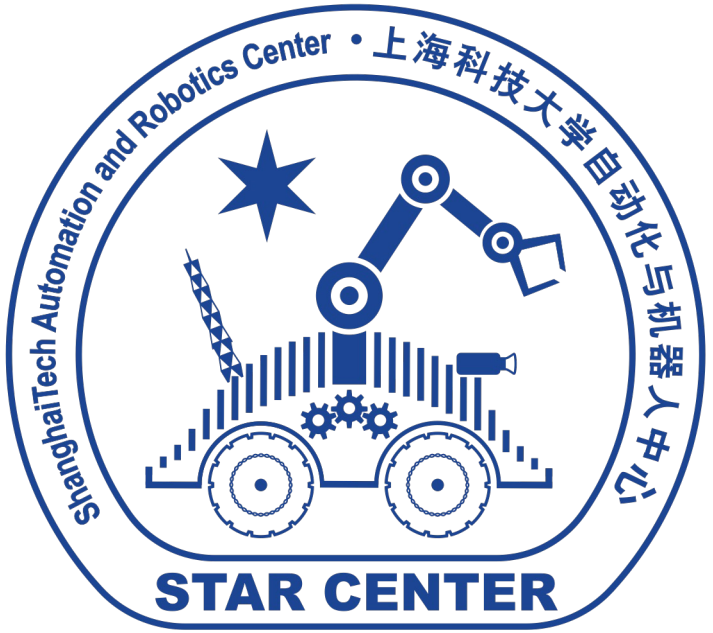
上海科技大学

ShanghaiTech University

CS289: **Mobile Manipulation Fall 2026**

Sören Schwertfeger

ShanghaiTech University



Outline

- Mission Planning
  - Combined Task & Motion Planning
  - LLMs
- Calibraiton
- Sensor Fusion

COMBINED TASK & MOTION PLANNING

Differentiable Task Assignment and Motion Planning

Jimmy Envall, Roi Poranne, Stelian Coros

ETH zürich

CRL Computational Robotics Lab

אוניברסיטת חיפה
University of Haifa
جامعة حيفا

Envall, J., Poranne, R., & Coros, S. (2023, October). Differentiable task assignment and motion planning. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 2049-2056). IEEE. https://arxiv.org/pdf/2304.09734

https://www.youtube.com/watch?v=zq3WyQFE5mA

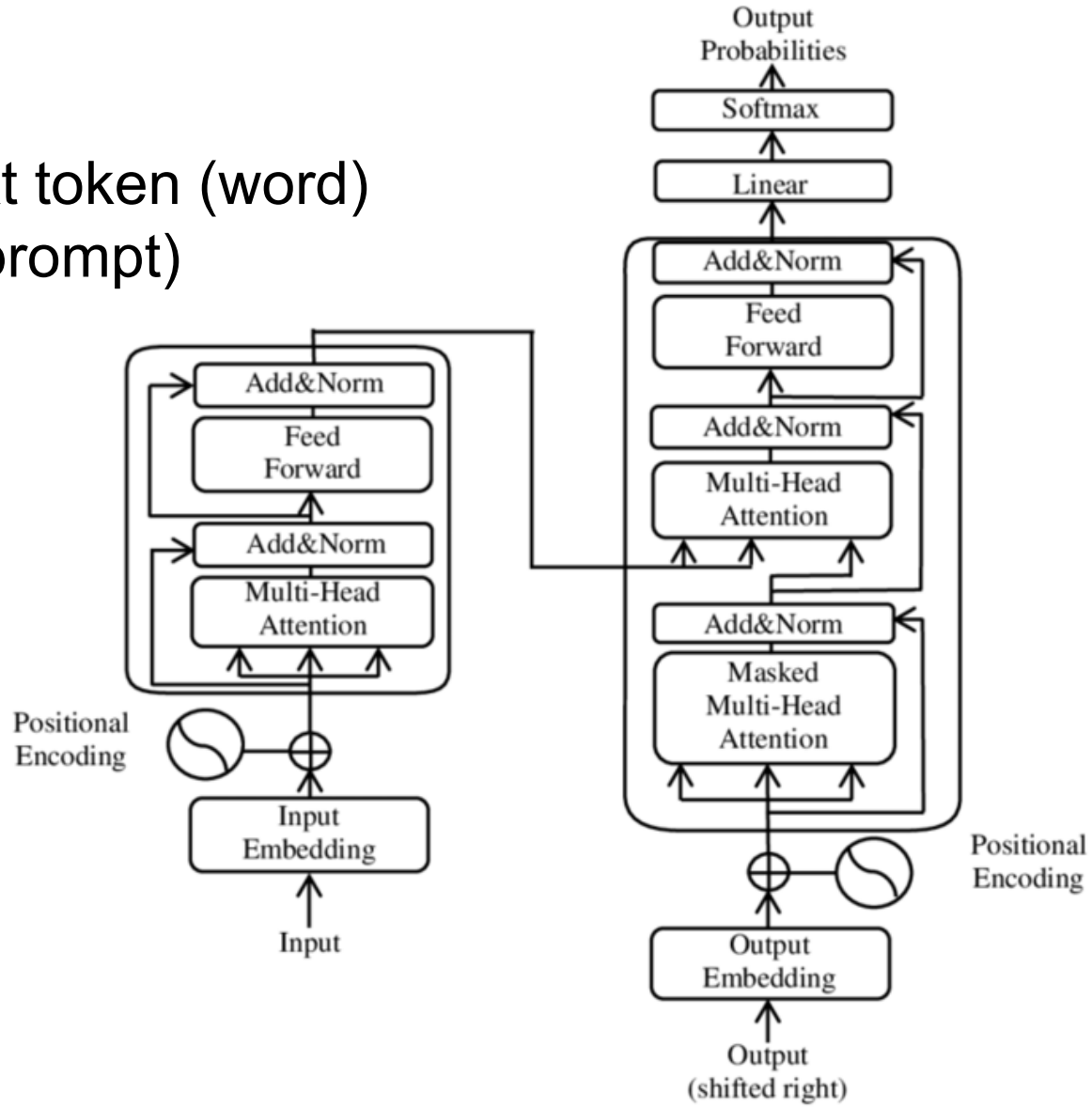
LLM

Large Language Model

- Transformer-based model outputting next token (word) with highest probability based on input (prompt)

- Powerful AI tool because:
  - “Understands” natural language (and other structured text) input
  - Through training has good world knowledge
  - Some form of (limited) reasoning

  - => useful in robotics, especially
    - Human Robot Interaction
    - All kinds of planning tasks
    - Various other purposes



LLM for Robotics

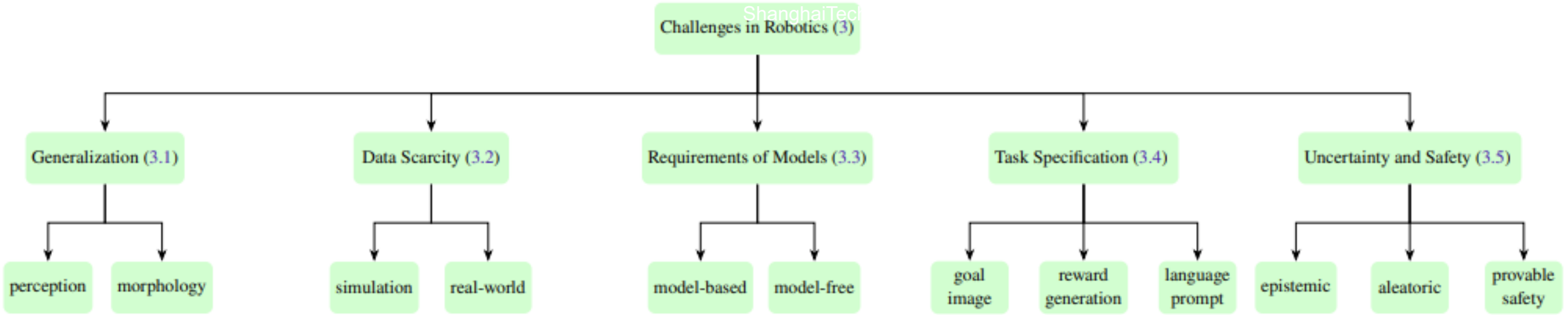
- Do As I Can, Not As I Say: Grounding Language in Robotic Affordances
  —4 Apr 2022: Robotics at Google, Everyday Robots
- ChatGPT for Robotics: Design Principles and Model Abilities
  —20 Feb 2023: Microsoft Autonomous Systems and Robotics Research
- Palm-E: An embodied multimodal language model
  —6 Mar 2023: Robotics at Google, TU Berlin, Google Research
- RT-2: New model translates vision and language into action
  —28 Jul. 2023: Google DeepMind
- RobotGPT: Robot Manipulation Learning from ChatGPT
  —3 Dec 2023: Samsung Research China, Tsinghua University
- Large Language Models for Robotics:Opportunities, Challenges, and Perspectives
  —9 Jan 2024: Northwestern Polytechnical University, Univ. Georgia, etc.
- Large Language Models for Robotics: A Survey
  —13 Nov 2023: Jinan University, University of Illinois Chicago
- Toward General-Purpose Robots via Foundation Models: A Survey and Meta-Analysis
  —14 Dec 2023: CMU, FAIR at Meta, Google DeepMind
- A Survey of Language-Based Communication in Robotics
  —6 Jun 2024: University of Southampton

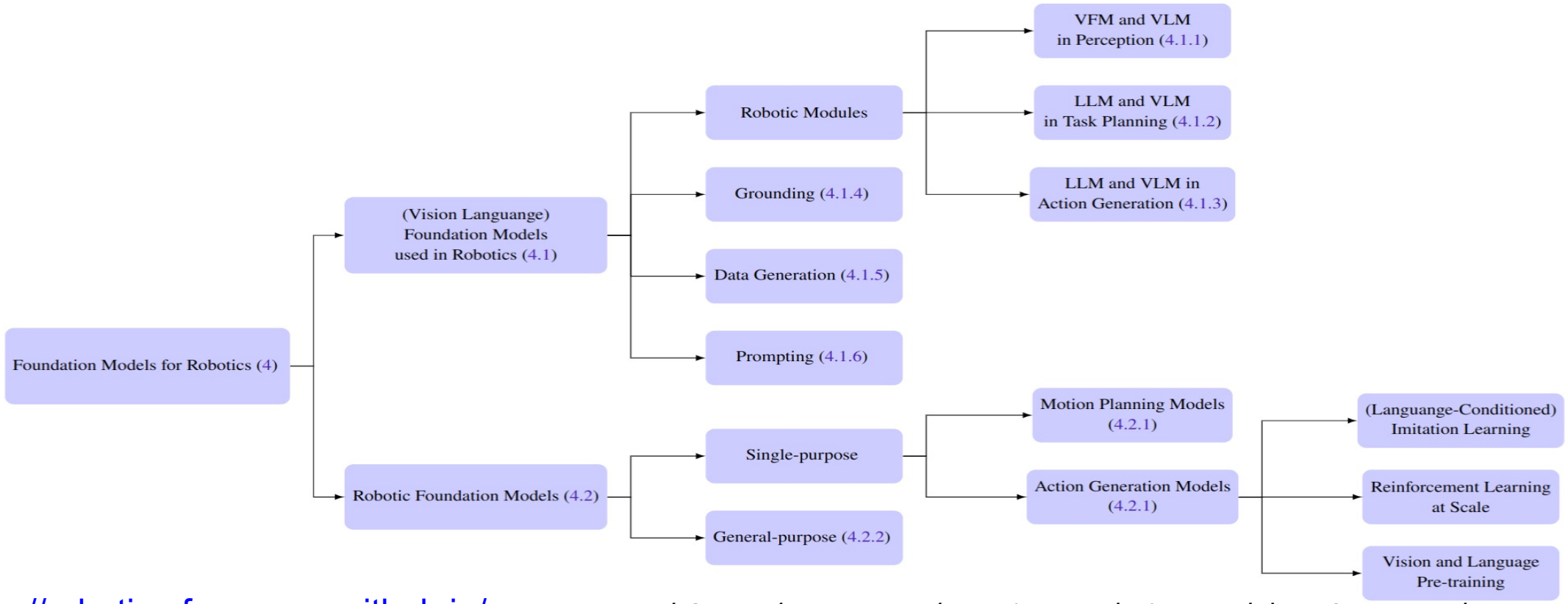
LLM for Robotics https://github.com/GT-RIPL/Awesome-LLM-Robotics

Surveys

- "No Free Checker: A Survey of Verifiers for Robot Policies", *arXiv, Sep 2026*, [Paper] [Paper List] [Website]
- "Safety in Embodied AI: A Survey of Risks, Attacks, and Defenses", *arXiv, May 2026*, [Paper] [Website]
- "A Survey of Robotic Language Grounding: Tradeoffs between Symbols and Embeddings", *IJCAI, Aug 2024*, [Paper]
- "A Superalignment Framework in Autonomous Driving with Large Language Models", *arXiv, Jun 2024*, [Paper]
- "Neural Scaling Laws for Embodied AI", *arXiv, May 2024*. [Paper]
- "On the Prospects of Incorporating Large Language Models (LLMs) in Automated Planning and Scheduling (APS)", *ICAPS, May 2024*, [Paper] [Website]
- "Toward General-Purpose Robots via Foundation Models: A Survey and Meta-Analysis", *arXiv, Dec 2023*. [Paper] [Paper List] [Website]
- "Language-conditioned Learning for Robotic Manipulation: A Survey", *arXiv, Dec 2023*, [Paper]
- "Foundation Models in Robotics: Applications, Challenges, and the Future", *arXiv, Dec 2023*, [Paper] [Paper List]
- "Robot Learning in the Era of Foundation Models: A Survey", *arXiv, Nov 2023*, [Paper]
- "The Development of LLMs for Embodied Navigation", *arXiv, Nov 2023*, [Paper]

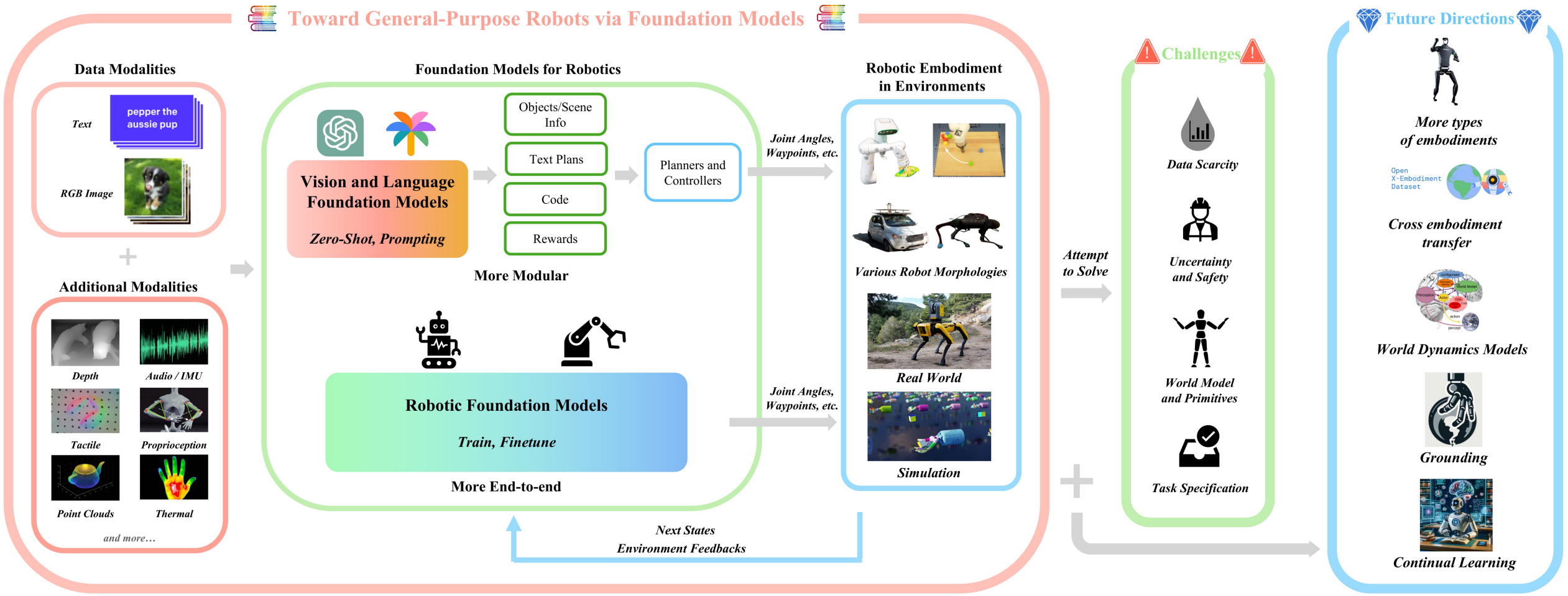
ShanghaiTech





https://robotics-fm-survey.github.io/ - *Toward General-Purpose Robots via Foundation Models: A Survey and Meta-Analysis*

Toward General-Purpose Robots via Foundation Models: A Survey and Meta-Analysis



ChatGPT for Robotics: Design Principles and Model Abilities

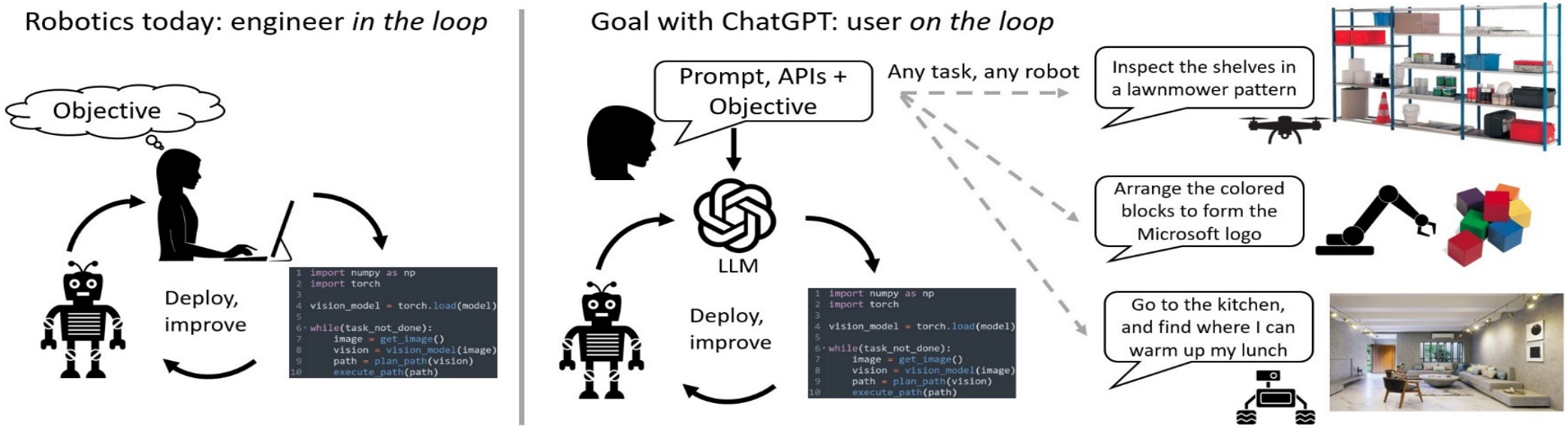


Figure 1: Current robotics pipelines require a specialized engineer in the loop to write code to improve the process. Our goal with ChatGPT is to have a (potentially non-technical) user on the loop, interacting with the language model through high-level language commands, and able to seamlessly deploy various platforms and tasks.

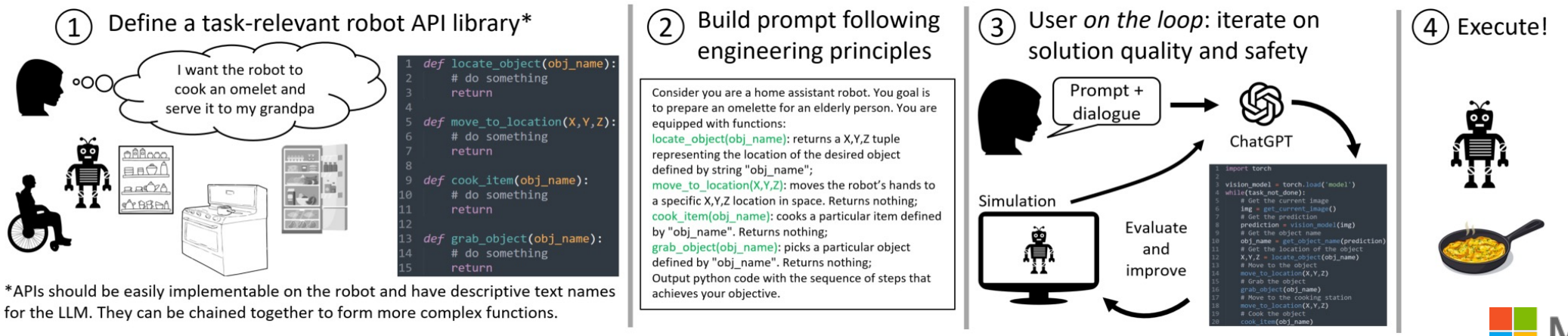
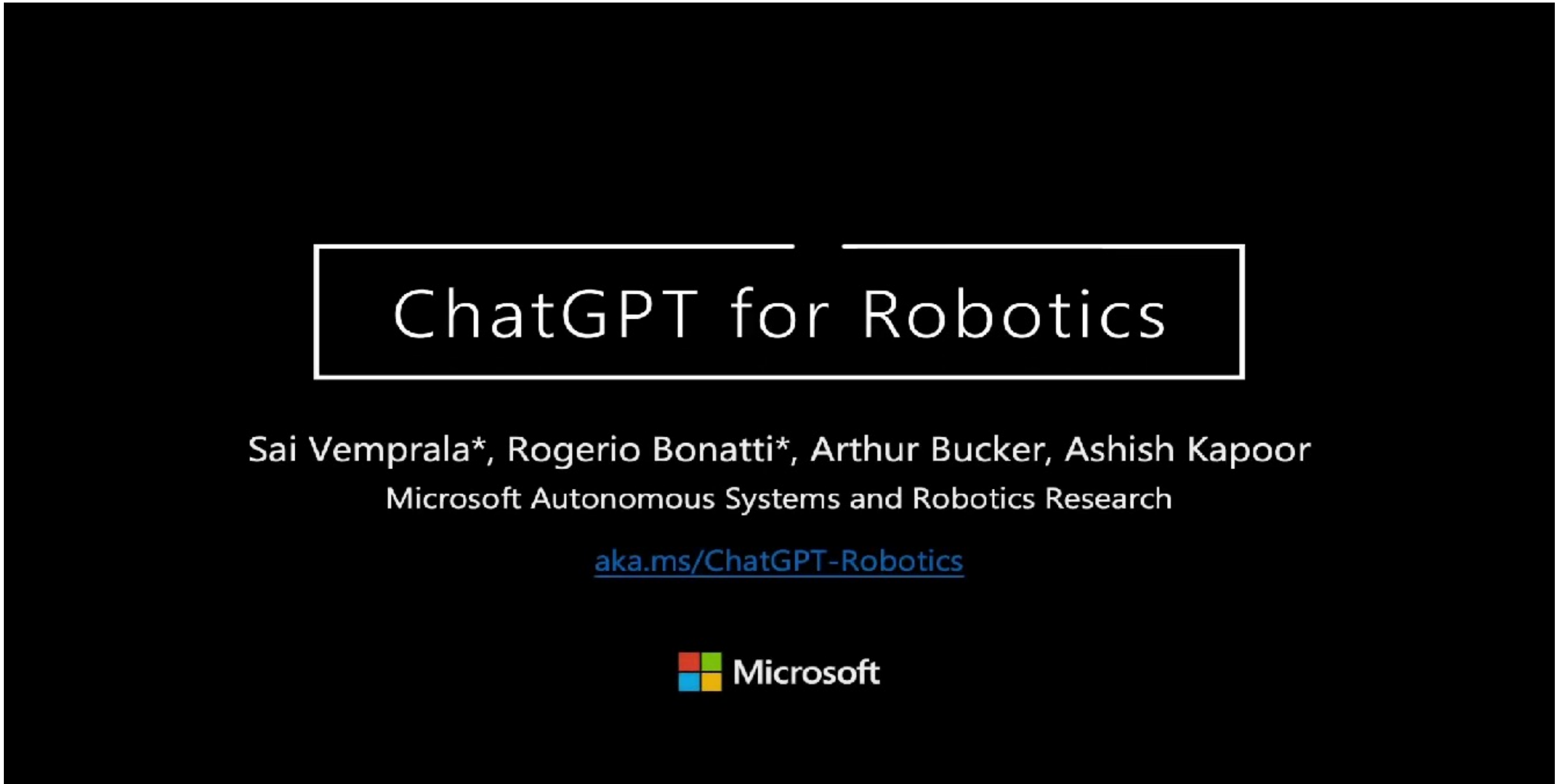


Figure 2: Robotics pipeline employing ChatGPT with the user on the loop to evaluate the output's quality and safety.

Microsoft

ChatGPT for Robotics: Design Principles and Model Abilities



PaLM-E: An Embodied Multimodal Language Model

Robotics at Google

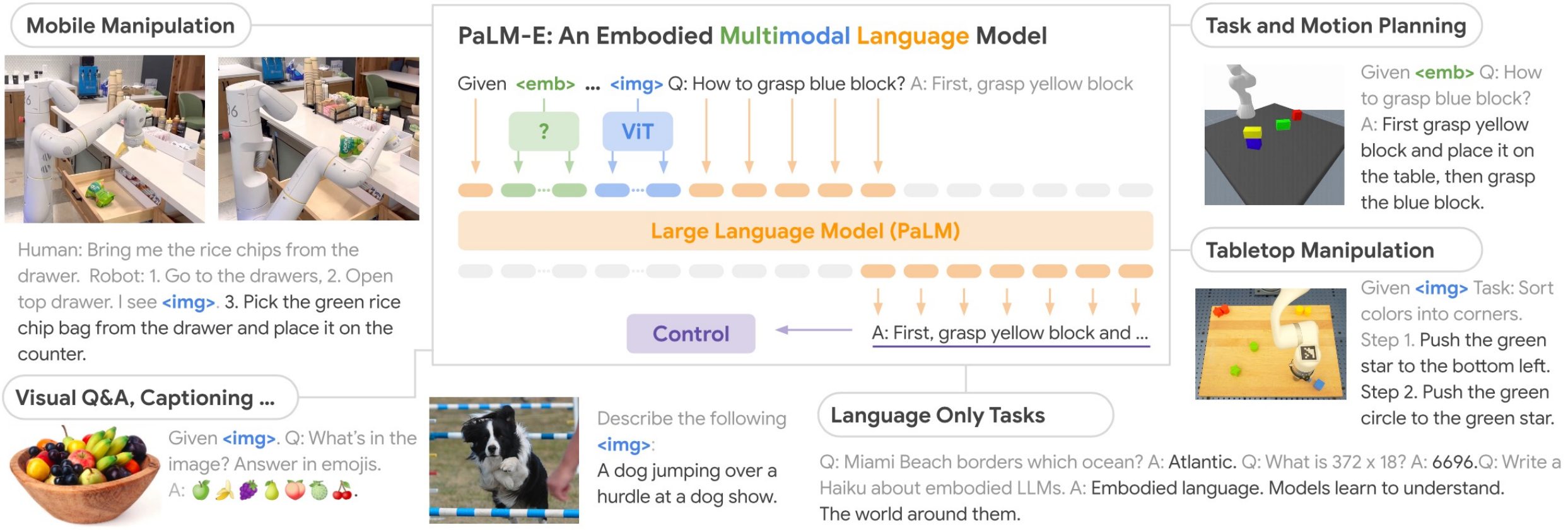
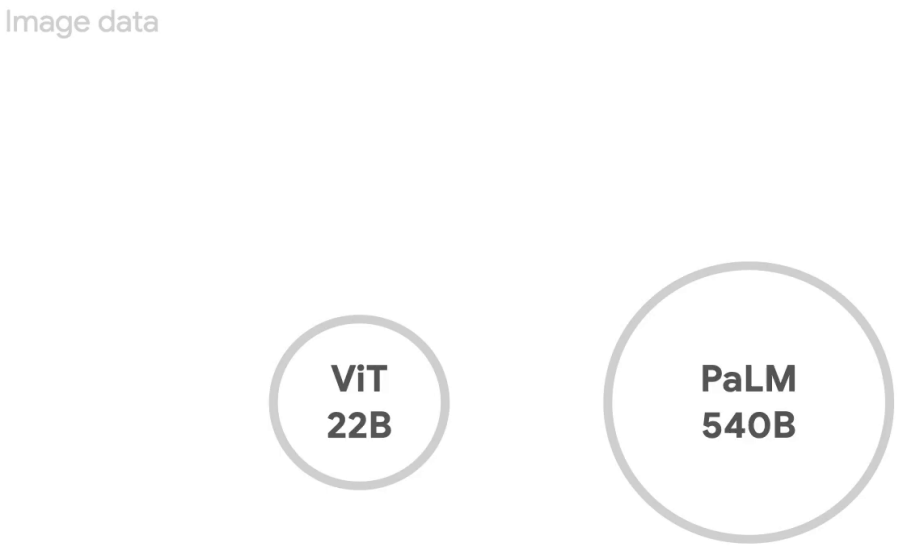


Figure 1: PaLM-E is a single general-purpose multimodal language model for embodied reasoning tasks, visual-language tasks, and language tasks. PaLM-E transfers knowledge from visual-language domains into embodied reasoning – from robot planning in environments with complex dynamics and physical constraints, to answering questions about the observable world. PaLM-E operates on multimodal sentences, i.e. sequences of tokens where inputs from arbitrary modalities (e.g. images, neural 3D representations, or states, in green and blue) are inserted alongside text tokens (in orange) as input to an LLM, trained end-to-end.

PaLM-E: An Embodied Multimodal Language Model





Abstract

Large language models have been demonstrated to perform complex tasks. However, enabling general inference in the real world, e.g. for robotics problems, raises the challenge of grounding. We propose embodied language models to directly incorporate real-world continuous sensor modalities into language models and thereby establish the link between words and percepts. Input to our embodied language model are multi-modal sentences that interleave visual, continuous state estimation, and textual input encodings.

Robotics at Google

This video has audio

Language Models as Zero-Shot Planners:

Extracting Actionable Knowledge for Embodied Agents

Wenlong Huang
UC Berkeley

Pieter Abbeel
UC Berkeley

Deepak Pathak*
CMU

Igor Mordatch*
Google



- Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents, ICML 2022

RT-2: Vision-Language-Action Models, Transfer Web Knowledge to Robotic Control

Google DeepMind

RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control

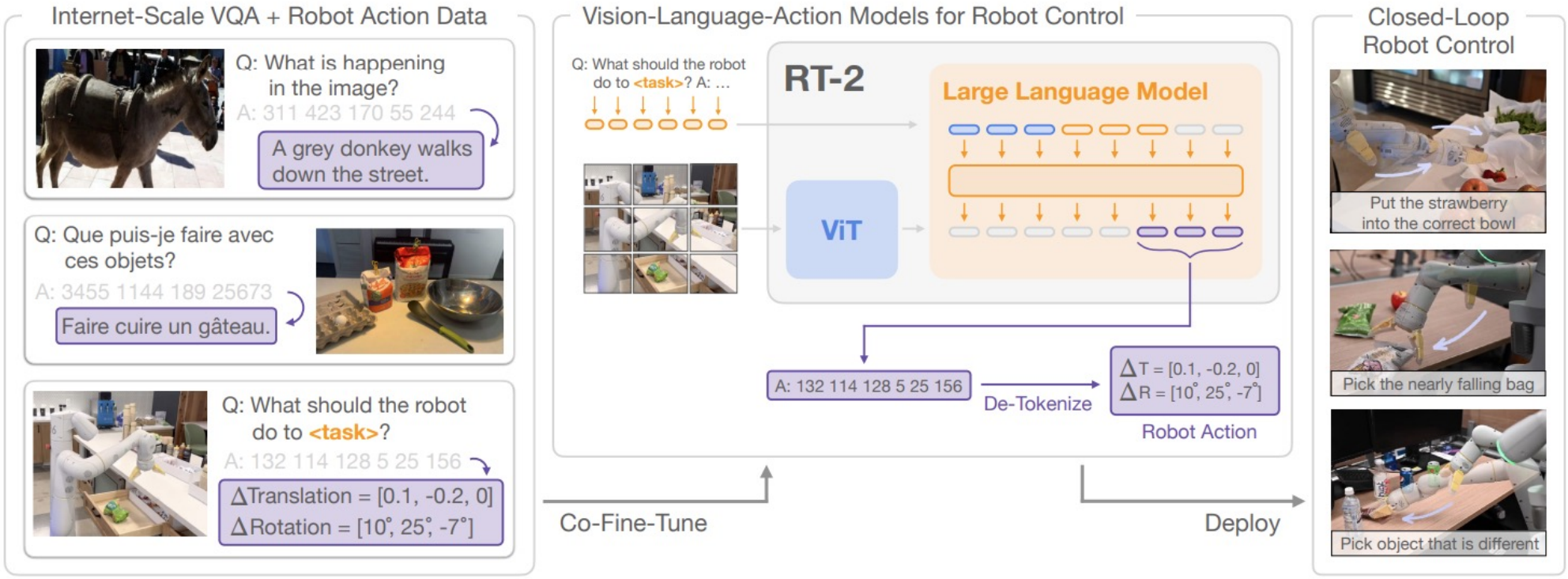
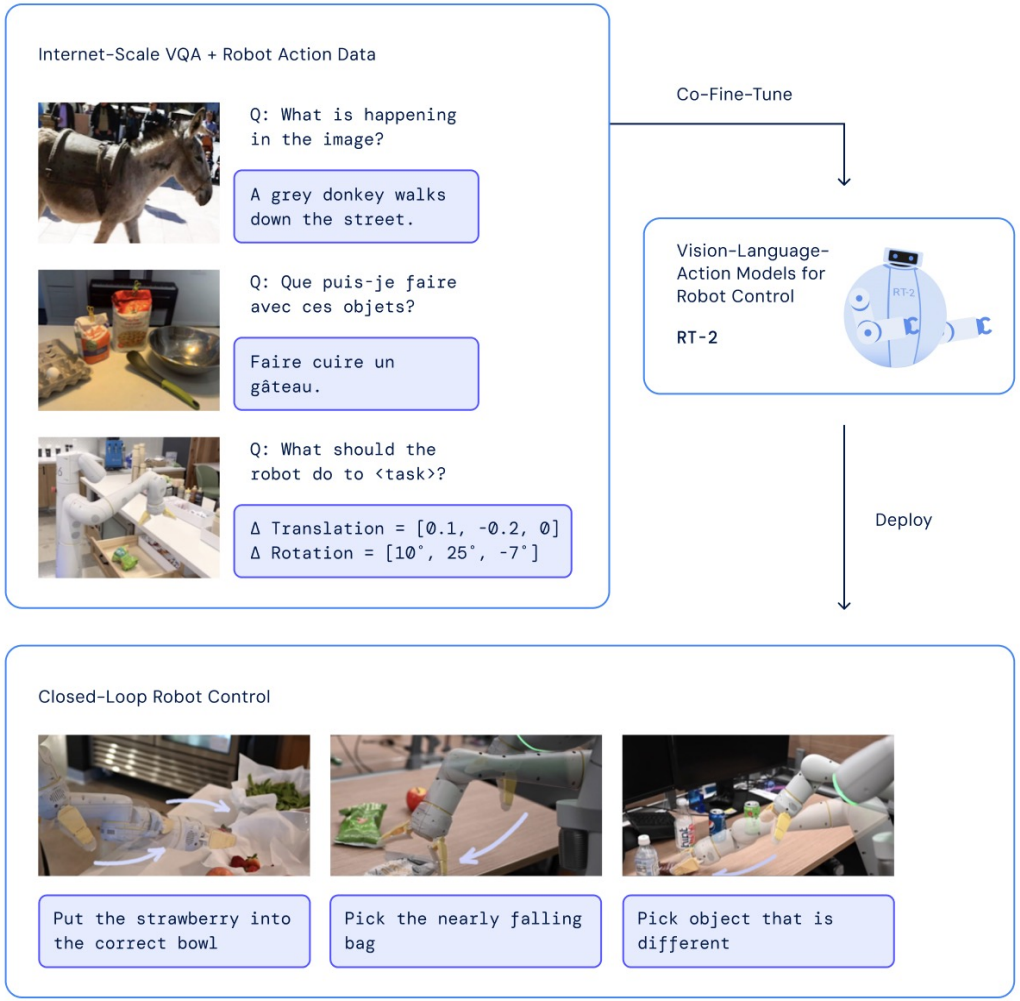


Figure 1 | RT-2 overview: we represent robot actions as another language, which can be cast into text tokens and trained together with Internet-scale vision-language datasets. During inference, the text tokens are de-tokenized into robot actions, enabling closed loop control. This allows us to leverage the backbone and pretraining of vision-language models in learning robotic policies, transferring some of their generalization, semantic understanding, and reasoning to robotic control. We demonstrate examples of RT-2 execution on the project website: robotics-transformer2.github.io.

RT-2: Vision-Language-Action Models, Transfer Web Knowledge to Robotic Control

Google DeepMind



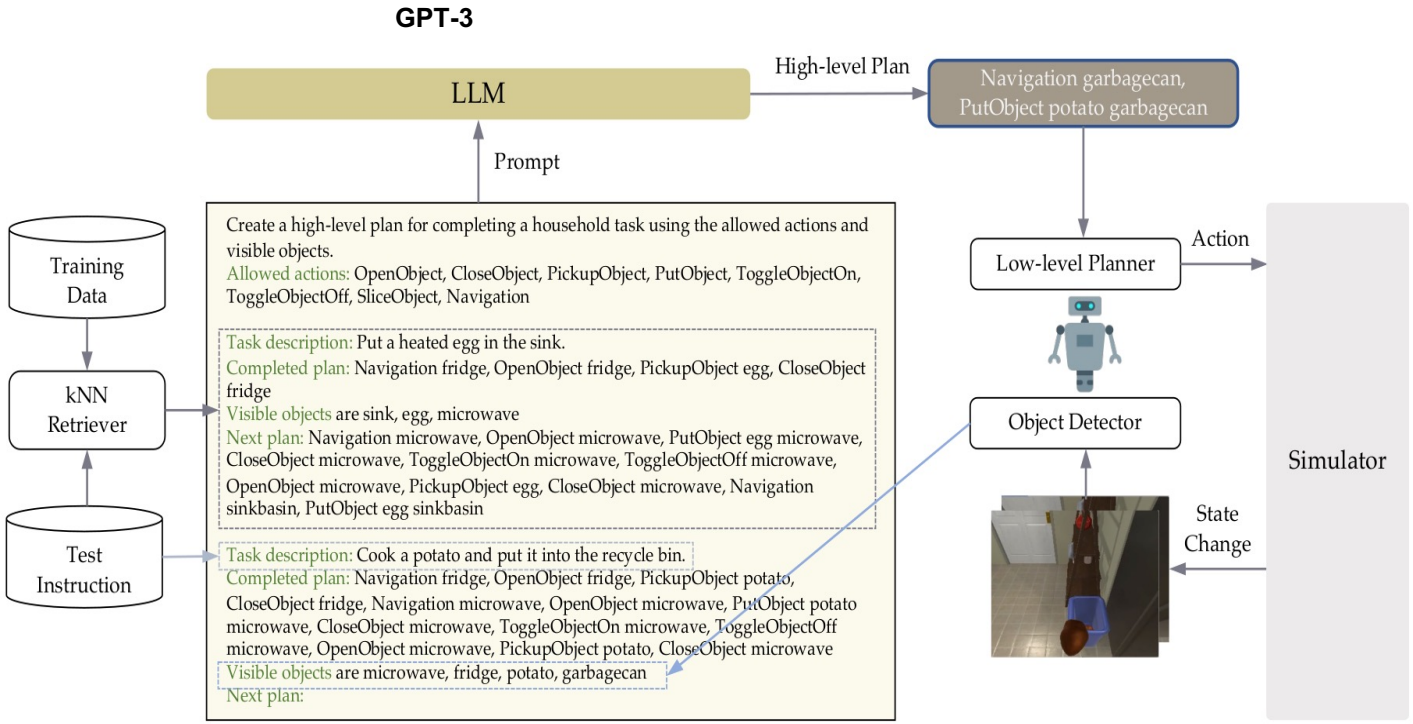
RT-2 architecture and training



- Song C H, Wu J, Washington C, et al.

LLM-planner: Few-shot grounded planning for embodied agents with large language models

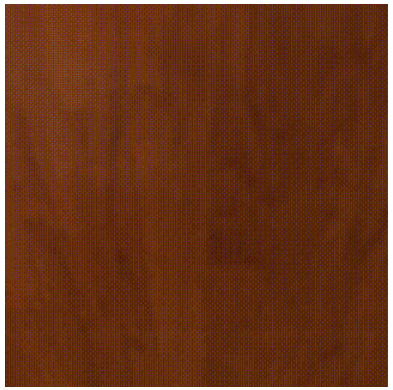
Proceedings of the IEEE/CVF International Conference on Computer Vision. 2023: 2998-3009.



- Dataset/ Simulator:

Alfred: vision-and-language navigation dataset: https://askforalfred.com/





- Another Embodied AI Simulator: Habitat: https://research.facebook.com/publications/habitat-a-platform-for-embodied-ai-research/

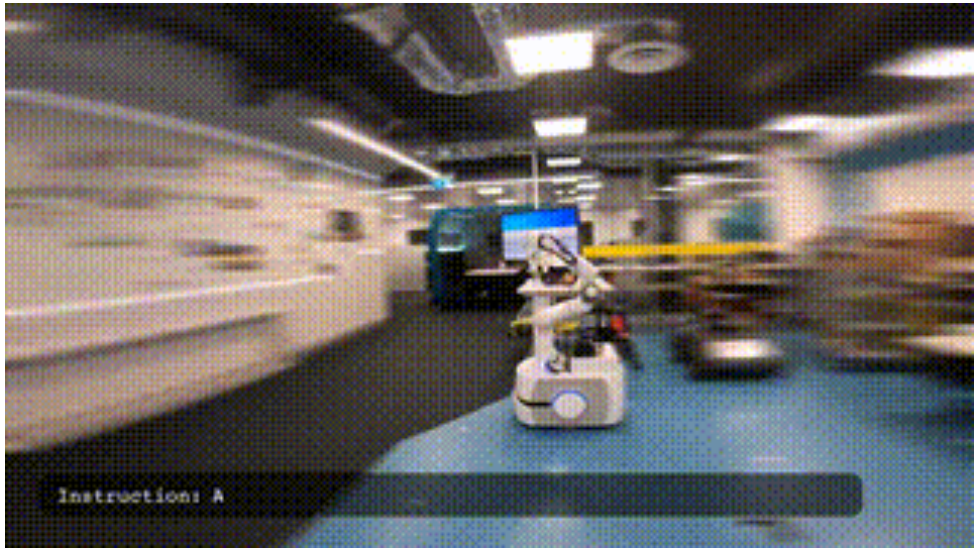
Table V: Robot datasets collected by recent VLAs. VIMA skills refer to "meta-tasks" in their original paper. We use the newer BridgeData V2. PC: point cloud.

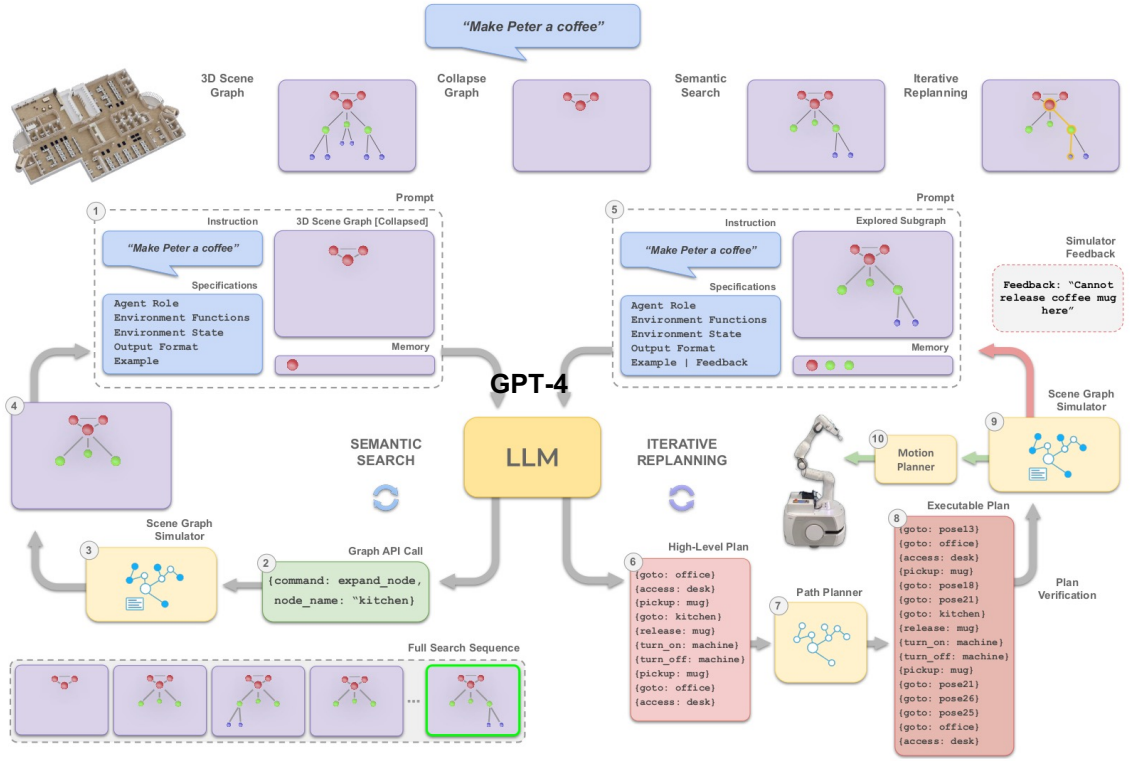
Name	Type	Instruction	Observation	Robot	Skills	Tasks	Objects	Episodes
RoboNet [165]	Real-world	Goal image	RGB	7 embodiments	-	-	-	162K
MT-Opt [71]	Real-world	Lang	RGB	7 embodiments	2	12	-	800K
BC-Z [24]	Real-world	Lang, Demo	RGB	Everyday	9	100	-	25.9K
RT-1 [33]	Real-world	Lang	RGB	Everyday	12	700+	16	130K
MOO [42]	Real-world	Multi-modal	RGB	Everyday	5	-	106	59.1K
VIMA [41]	Simulator	Multi-modal	RGB	UR5	17	-	29	650K
RoboSet [166]	Real-world	Lang	RGB, D	Franka	12	38	-	98.5K
BridgeData [167]	Real-world	Lang	RGB, D	WidowX 250	13	-	100+	60.1K
OXE [37]	Real-world	Lang	RGB, D, PC	22 embodiments	527	160,266	-	1M+

Ma Y, Song Z, Zhuang Y, et al. A Survey on Vision-Language-Action Models for Embodied AI[J]. arXiv preprint arXiv:2405.14093, 2024.

Planning: Grounding large language models using 3d scene graphs for scalable task planning

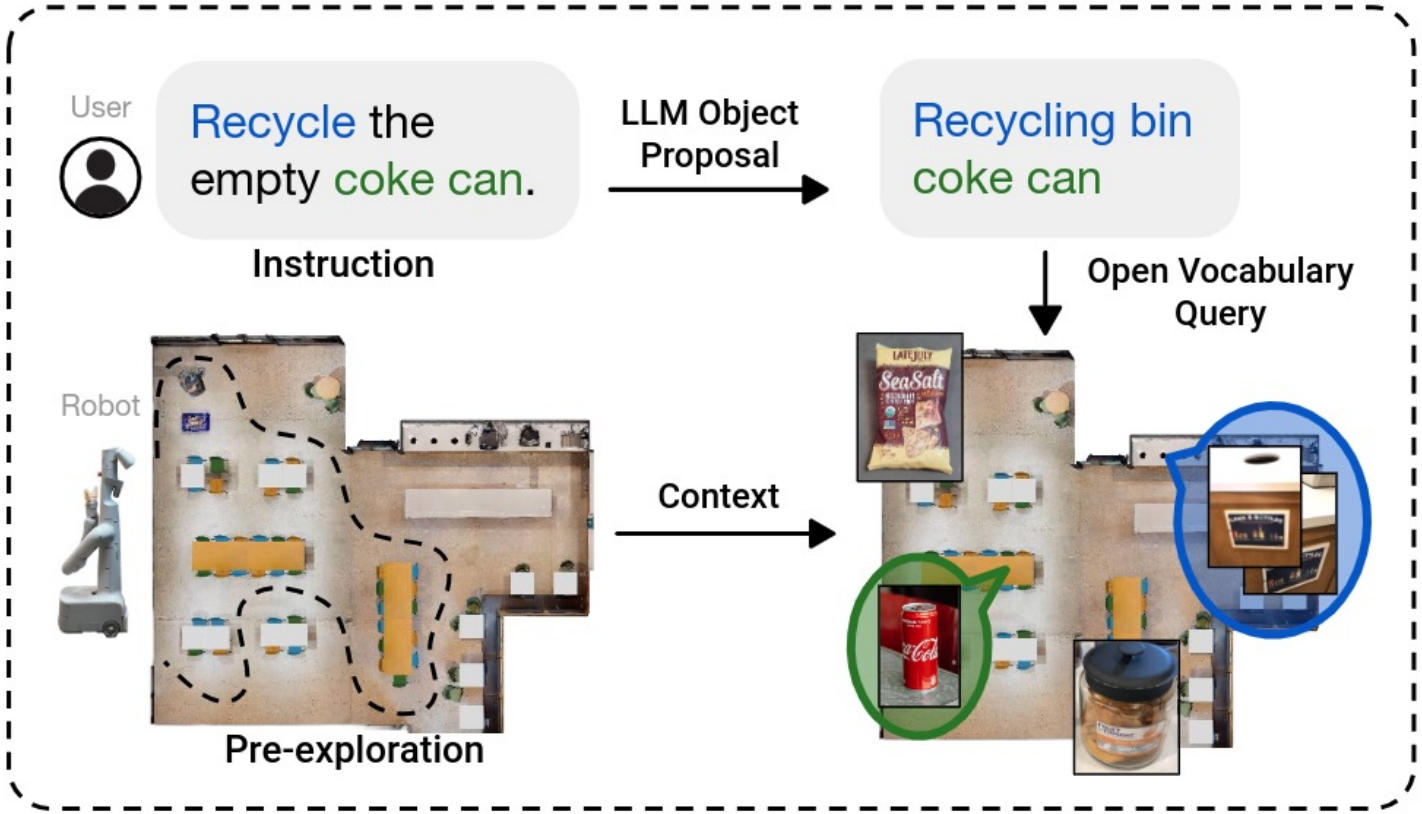
- Rana K, Haviland J, Garg S, et al. Sayplan: Grounding large language models using 3d scene graphs for scalable task planning[J]. arXiv preprint arXiv:2307.06135, 2023.





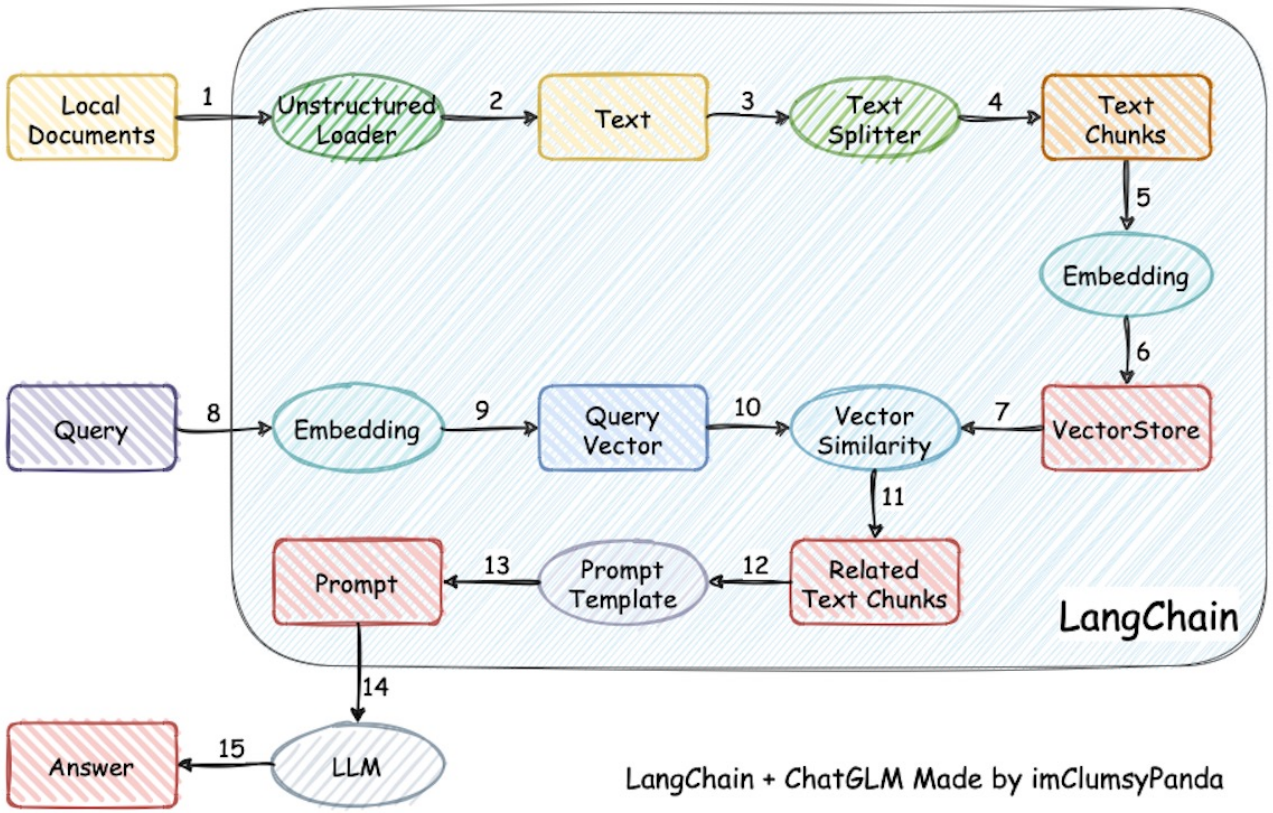
Mission Planning :

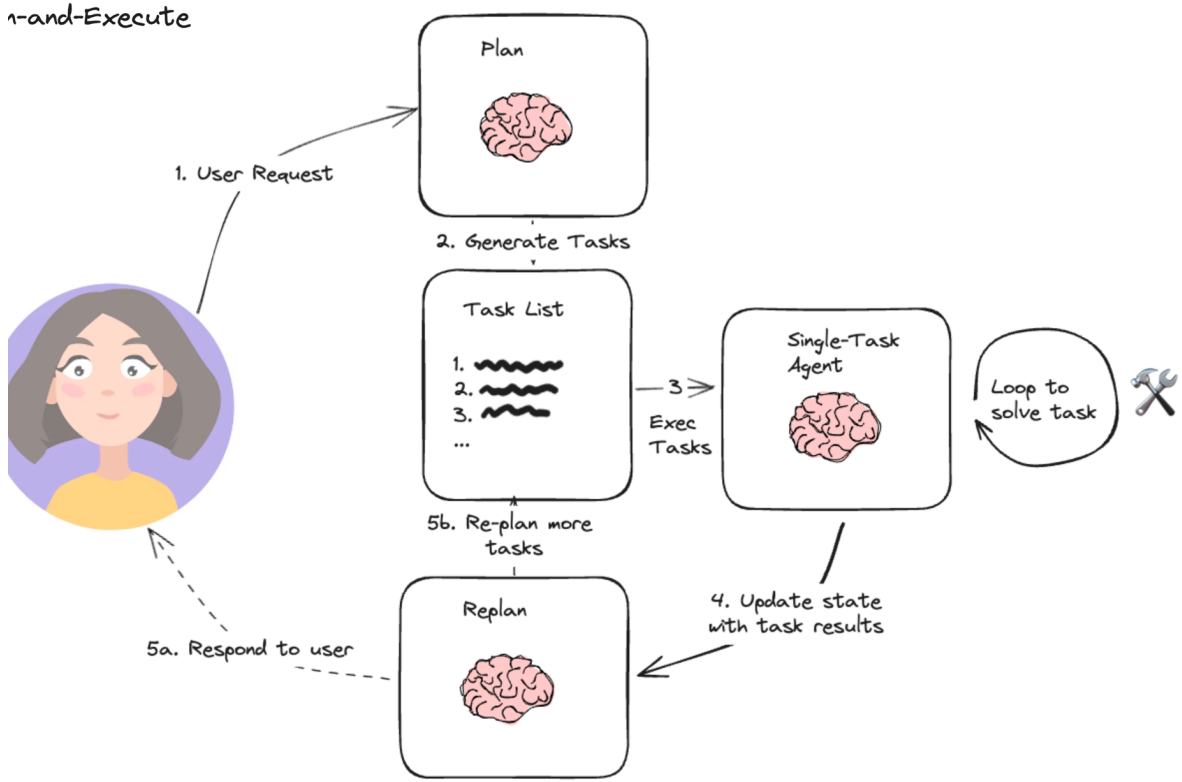
Open-vocabulary queryable scene representations for real world planning



Chen, B., Xia, F., Ichter, B., Rao, K., Gopalakrishnan, K., Ryoo, M. S., ... & Kappler, D. (2023, May). Open-vocabulary queryable scene representations for real world planning. In *2023 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 11509-11522). IEEE.

Retrieval Augmented Generation: LangGraph





https://langchain-ai.github.io/langgraph/

SPINE:
Online Semantic Planning for Missions with Incomplete Natural Language Specifications in Unstructured Environments

Zachary Ravichandran, Varun Murali, Mariliza Tzes,
George J. Pappas, and Vijay Kumar

Penn Engineering | GRASP Laboratory
General Robotics, Automation, Sensing & Perception Lab

https://zacravichandran.github.io/SPINE/

Prompt engineering

- Ask the LLM to provide uncertainty of the prompts:
  - Ren A Z, Dixit A, Bodrova A, et al. Robots that ask for help: Uncertainty alignment for large language model planners[J]. arXiv preprint arXiv:2307.01928, 2023.
  - SaySelf: Teaching LLMs to Express Confidence with Self-Reflective Rationales

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

Camera Calibration Parameters

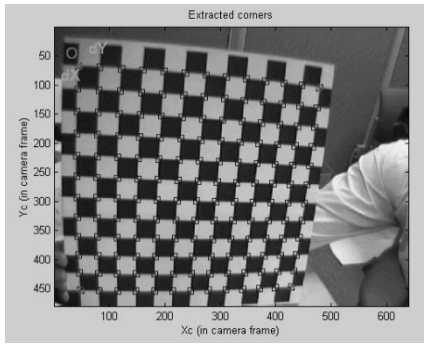
- Different models possible – typical one:
- α_u, α_v: focal length and size of pixel
- u_0, v_0: position of the sensor w.r.t optical center
- k_1: radial distortion

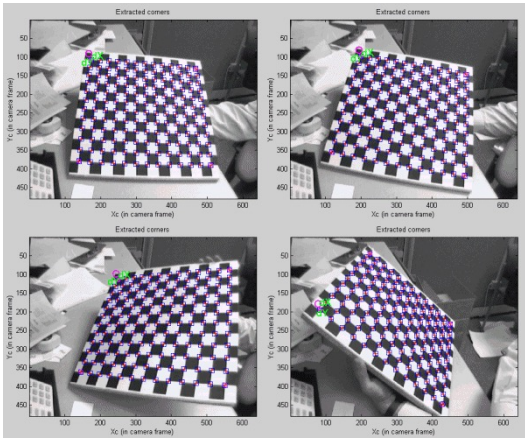
$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot R \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + T \qquad \begin{bmatrix} u_d \\ v_d \end{bmatrix} = (1 + k_1 \rho^2) \cdot \begin{bmatrix} u \\ v \end{bmatrix}$$

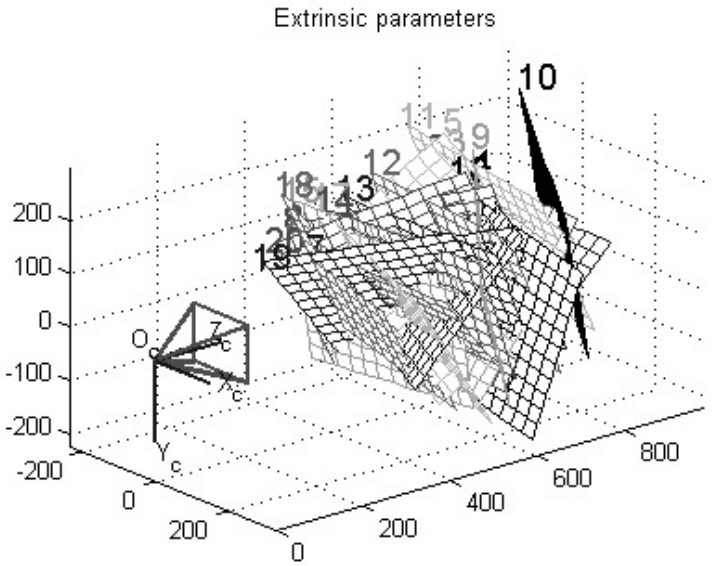
Camera Calibration

- Parameters: govern mapping from scene points to image points
- Idea: known:
  - pixel coordinates of image points p
  - 3D coordinates of the corresponding scene points P
  - => compute the unknown parameters A, R, T by solving the perspective projection equation

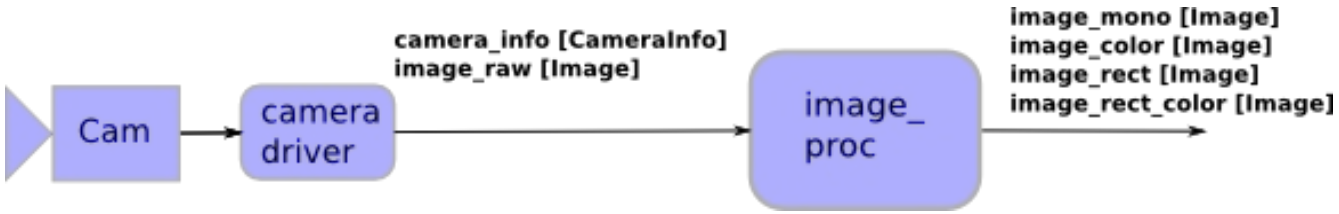
$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot R \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + T$$







ROS Image Processing



sensor_msgs/CameraInfo

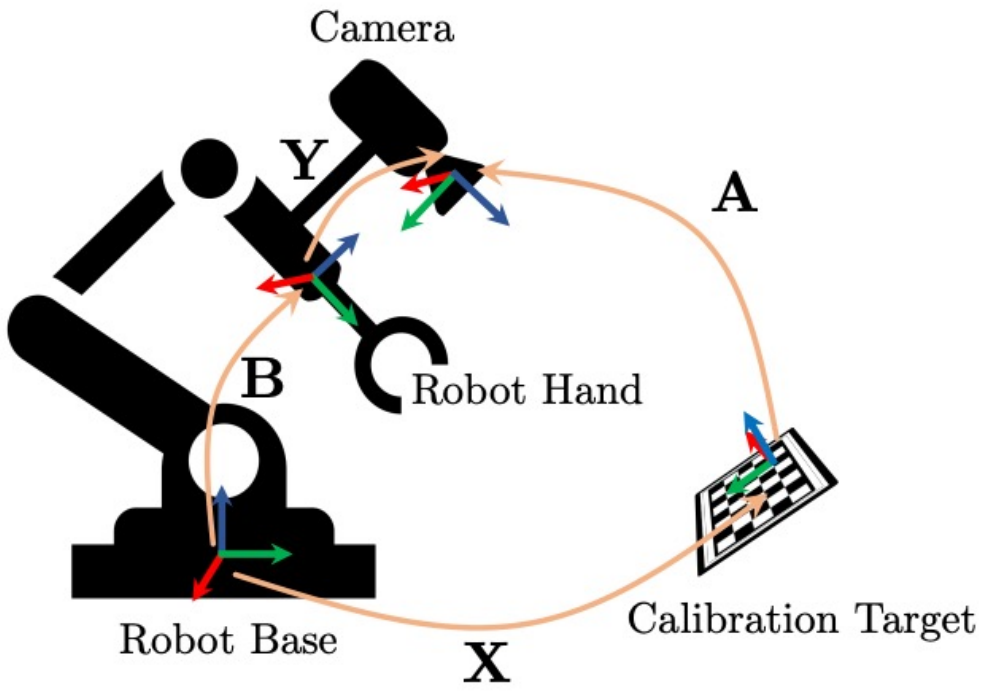
std_msgs/Header header
uint32 height
uint32 width
string distortion_model
float64[] D
float64[9] K
float64[9] R
float64[12] P
uint32 binning_x
uint32 binning_y
sensor_msgs/RegionOfInterest roi

- image_proc : Image processing node
  - Take calibration data from sensor_msgs/CameraInfo to calibrate/ rectify images
  - http://wiki.ros.org/image_proc
- Calibrate cameras:
  - http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration
  - http://wiki.ros.org/camera_calibration/Tutorials/StereoCalibration

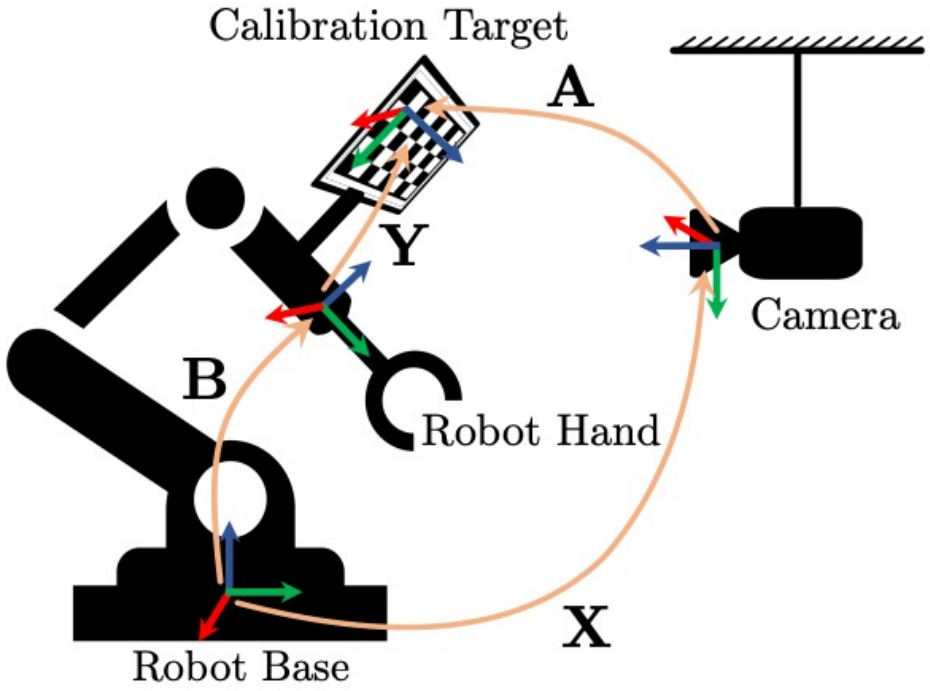
Extrinsic Calibration

- Find pose of sensor w.r.t:
  - another sensor of same type
    - registration/ scan matching
  - another sensor of a different type
    - Heterogeneous calibration (e.g. LiDAR, camera, IMU)
  - a robot/ the arm frame – arm coordinate system
    - Hand-eye calibration
  - many other sensors:
    - Multi-sensor calibration

Manipulation: Hand-Eye-Calibration



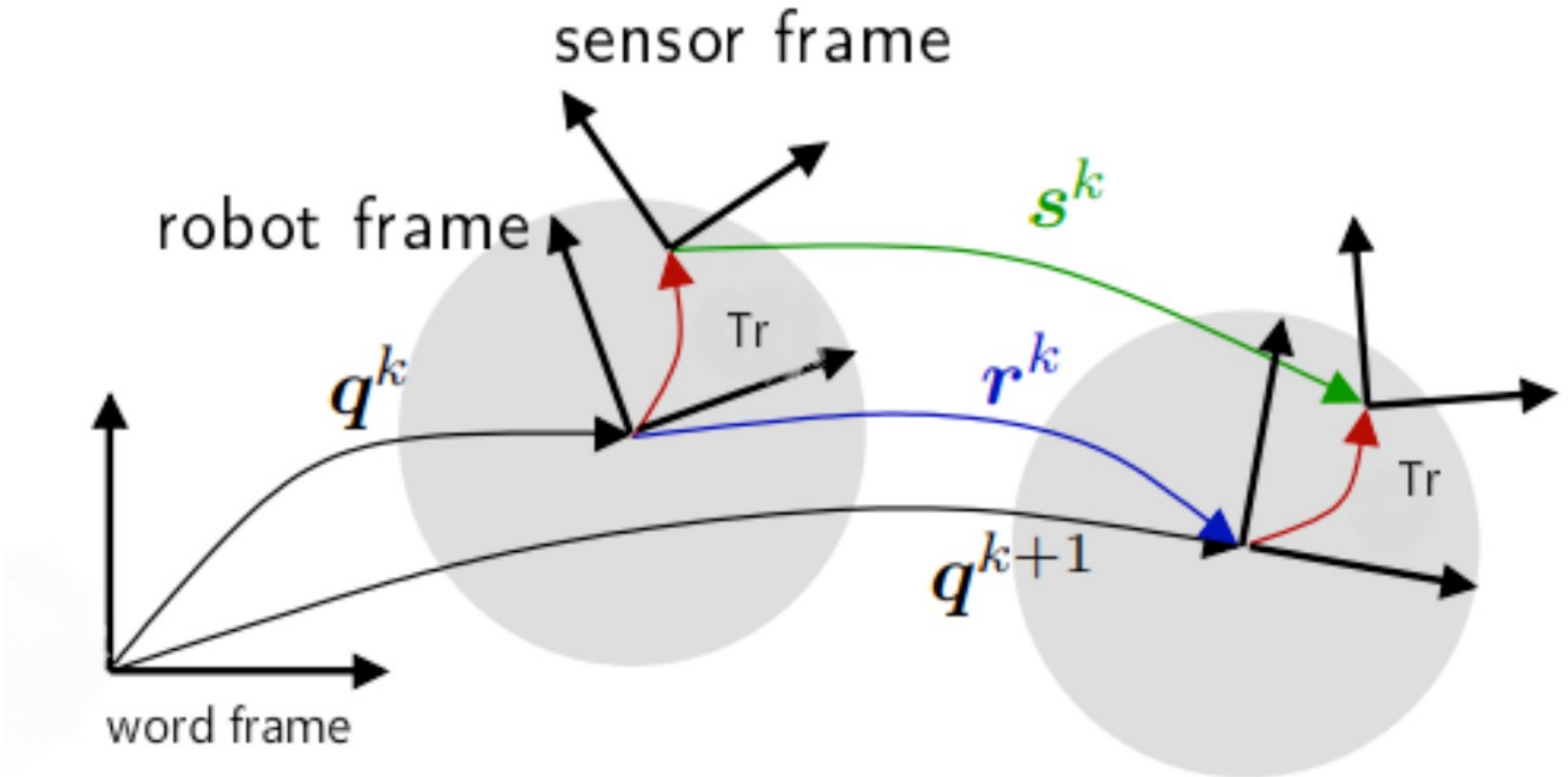
(a) eye-on-hand; aka: eye-in-hand



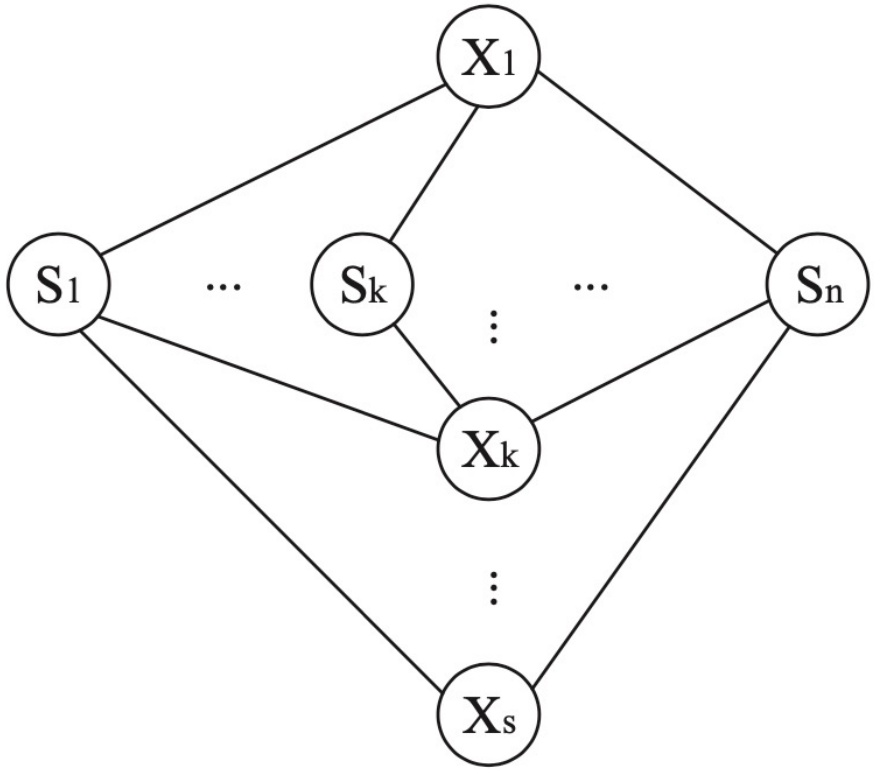
(a) eye-to-base; aka: eye-to-hand

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 $\mathbf{AX} = \mathbf{YB}$.

Hand-Eye calibration: robot to sensor transform



Heterogeneous Multi-sensor Calibration based on Graph Optimization



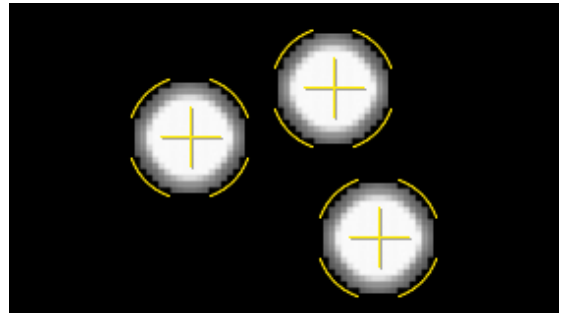
5.6 The graph representing the relationships between different sensors. $X_1 ... X_s$ and $S_1 ... S_n$ are different types of sensors such as camera and Velodyne. An edge between two nodes represents a direct sensor-to-sensor calibration between these two devices.

Segway: Tracking Systems

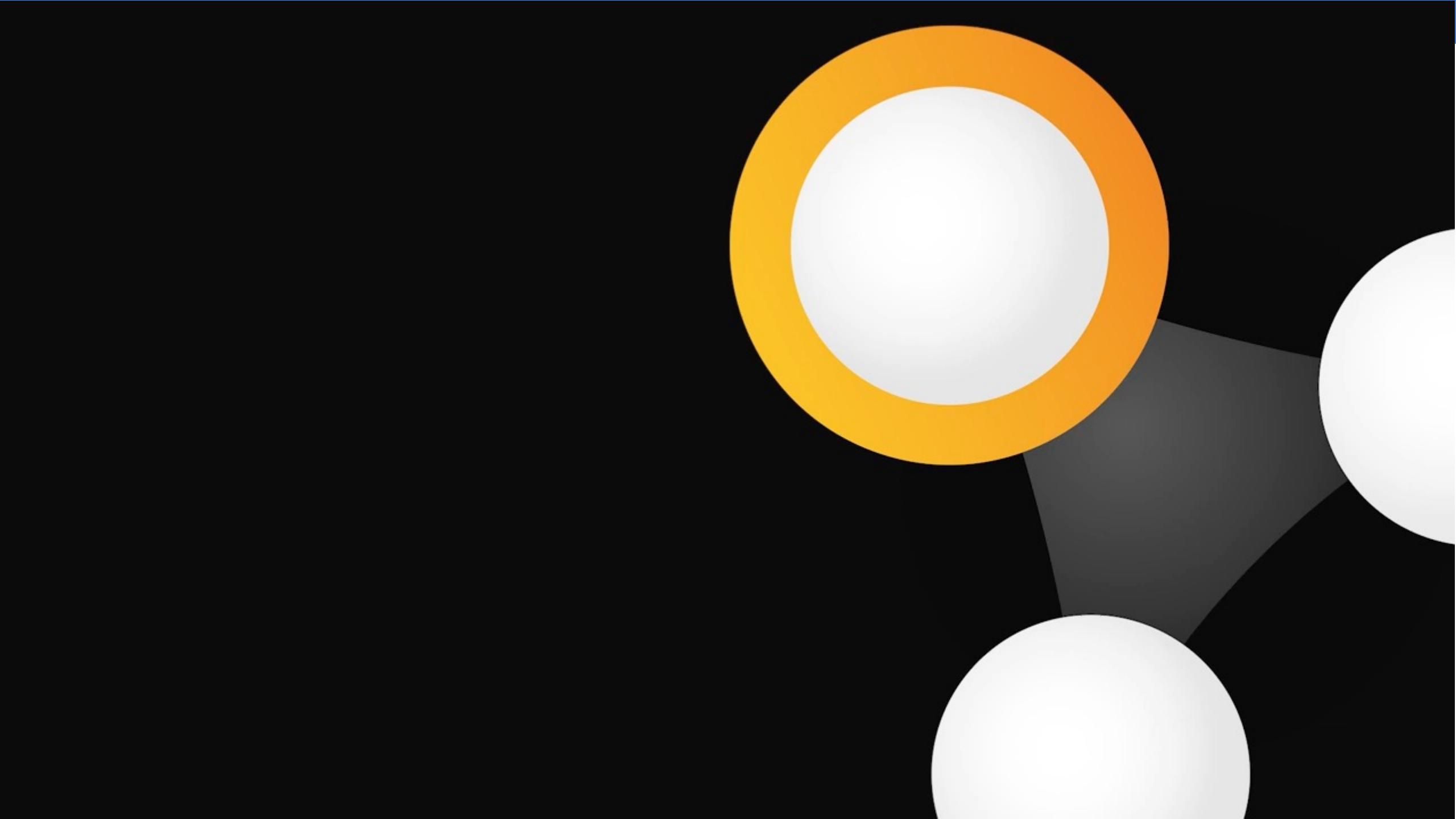


- Use cameras to detect infrared markers
- Triangulate 3D marker position from multiple cameras
- Create groups of markers to estimate 3D poses of groups
- Cameras need to be intrinsically & extrinsically calibrated
- Cameras report only marker (image) positions to system => one system can handle many cameras
- Famous vendors: Vicon, Optitrack
- E.g. Optitrack system at STAR Center:
  - 240Hz
  - Error: +-0.20 mm
  - 21 cameras





Video next slide: https://www.youtube.com/watch?v=o2xufLT8hUs
Video next next slide: https://www.youtube.com/watch?v=m0szHx3St-M



Calibration

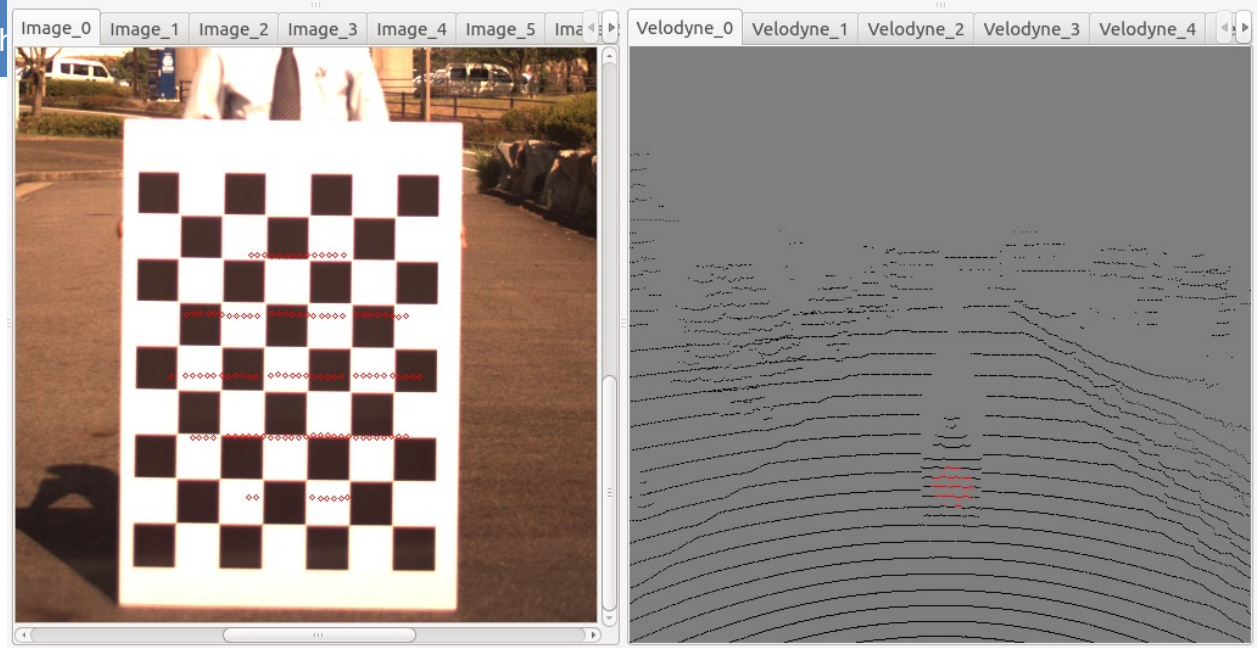
Manipulator Calibration

- 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

- 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

Tools & Tutorials I

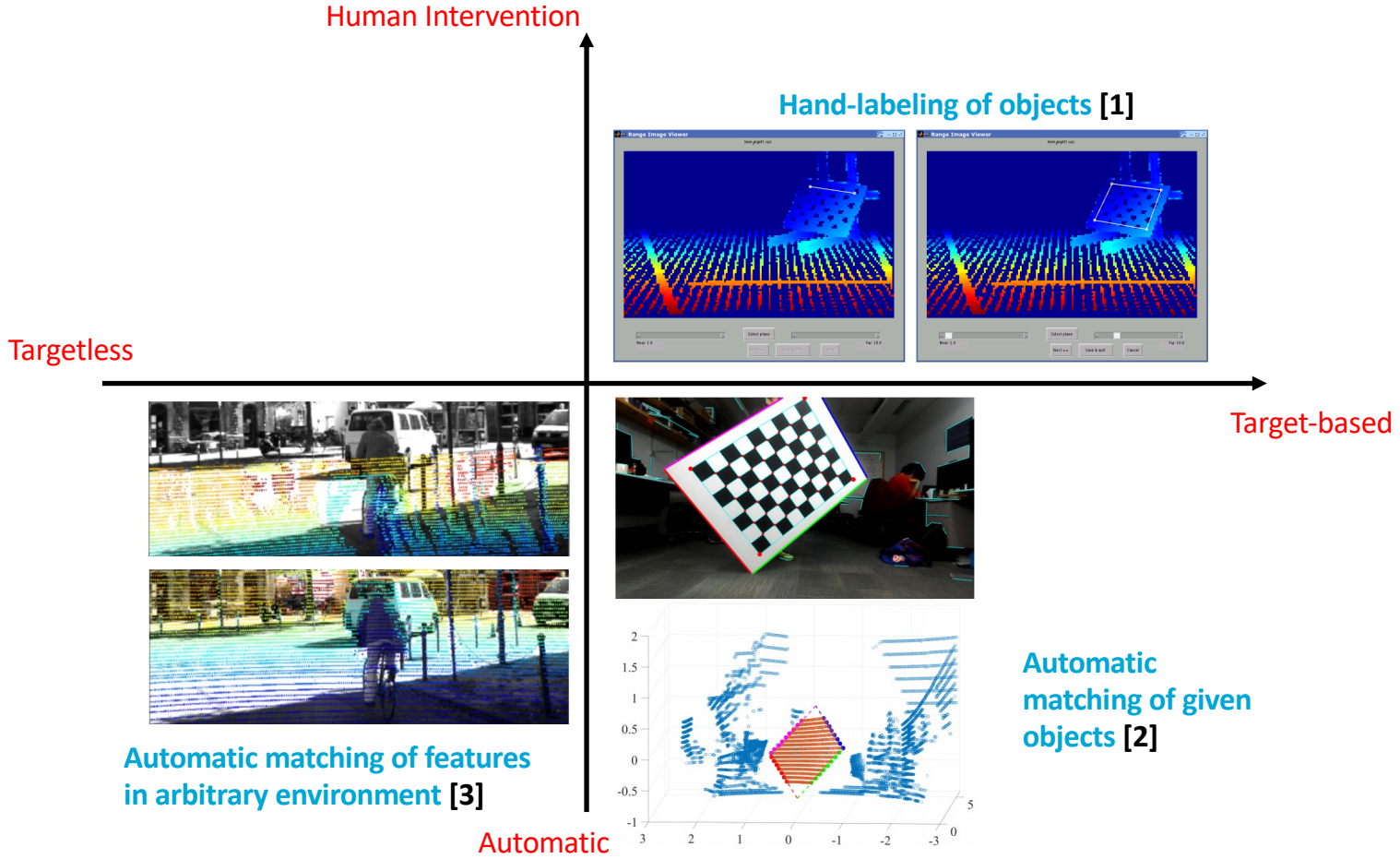


- Camera to Camera:
  - Stereo calibration (see previous)
- Camera to LiDAR:
  - https://github.com/CPFL/Autoware/wiki/Calibration
  - https://www.youtube.com/watch?v=pfBmfgHf6zg
  - Needs Autoware: https://github.com/cpfl/Autoware
- Point Cloud to Point Cloud (3D LiDAR):
  - Any point cloud registration, e.g. ICP
  - Example: Normal Distributions Transform https://pointclouds.org/documentation/tutorials/normal_distributions_transform.html

Tools & Tutorials II

- Camera to IMU:
  - https://github.com/ethz-asl/kalibr
  - Offers multi-camera calibration; camera IMU calibration
  - Tutorial: https://github.com/ethz-asl/kalibr/wiki/Camera-IMU-calibration
  - Install: https://github.com/ethz-asl/kalibr/wiki/installation
- Hand-Eye Calibration:
  - E.g. Simultaneous Hand-Eye Calibration and Reconstruction
    - Doesn't use calibration targets
    - https://github.com/STAR-Center/shecar
  - Or: https://github.com/IFL-CAMP/easy_handeye
- Multi-sensor Calibration:
  - Optimize the results of a graph of individual calibrations…
  - "Heterogeneous Multi-sensor Calibration based on Graph Optimization" https://arxiv.org/pdf/1905.11167.pdf
  - **Multical: Spatiotemporal Calibration for Multiple IMUs, Cameras and LiDARs** https://github.com/zhixy/multical

Camer <-> LiDAR Calibration



[1] R. Unnikrishnan and M. Hebert, “Fast Extrinsic Calibration of a Laser Rangefinder to a Camera,” Pittsburgh, PA, techreport CMU-RI-TR-05-09, Jul. 2005.
[2] L. Zhou, Z. Li and M. Kaess, "Automatic Extrinsic Calibration of a Camera and a 3D LiDAR Using Line and Plane Correspondences," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, 2018, pp. 5562-5569.

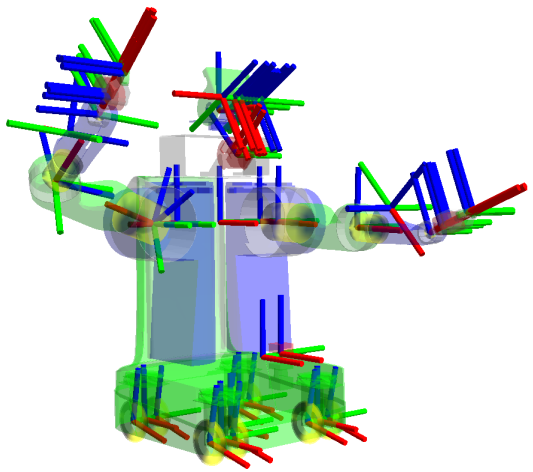
[3] J. Castorena, U. S. Kamilov, and P. T. Boufounos. “Autocalibration of lidar and optical cameras via edge alignment”. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2862–2866, 2016.



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
…

Mapping Robot Point Cloud Colorization Example



Mapping robot of Prof. Schwertfeger

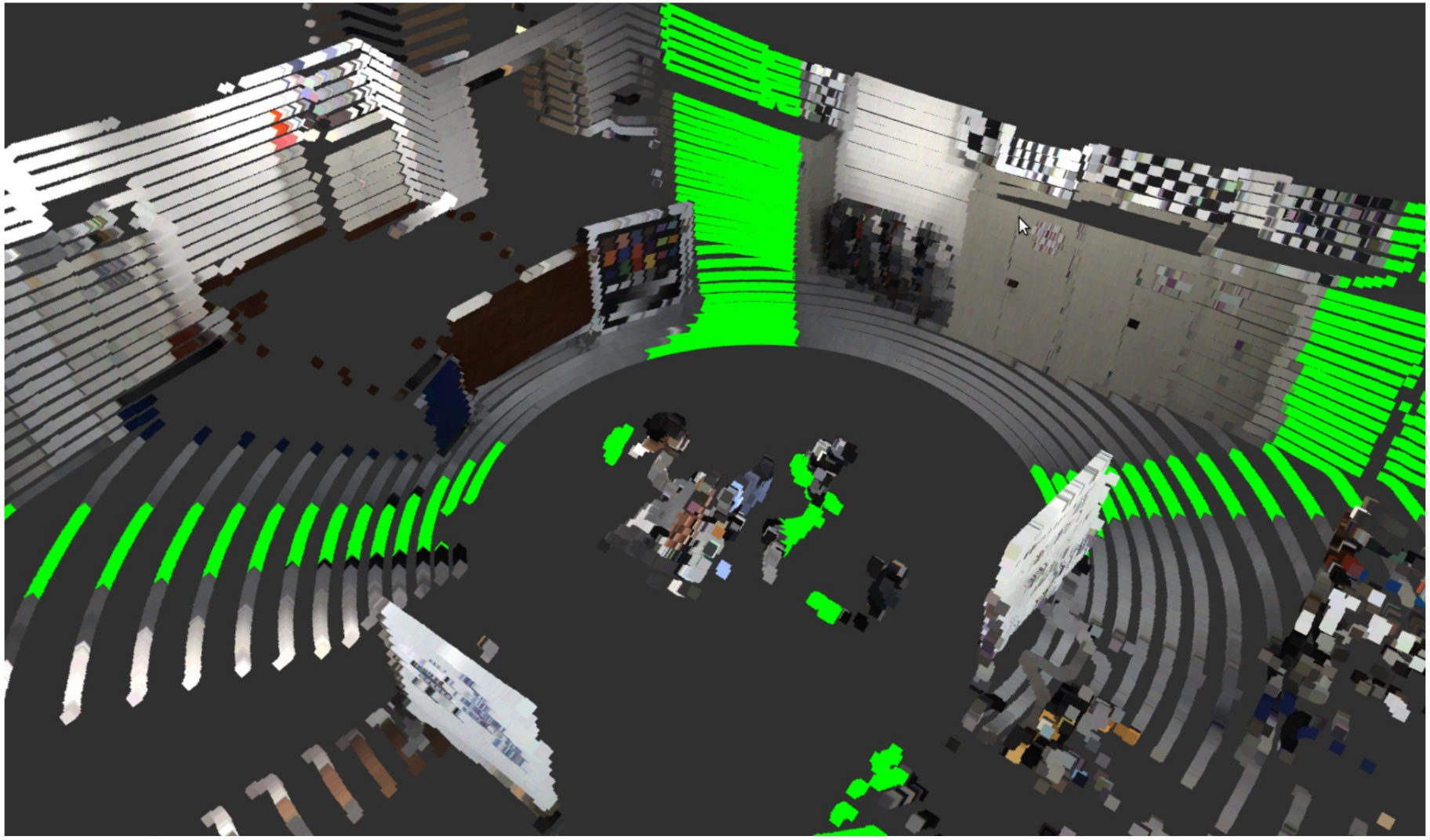
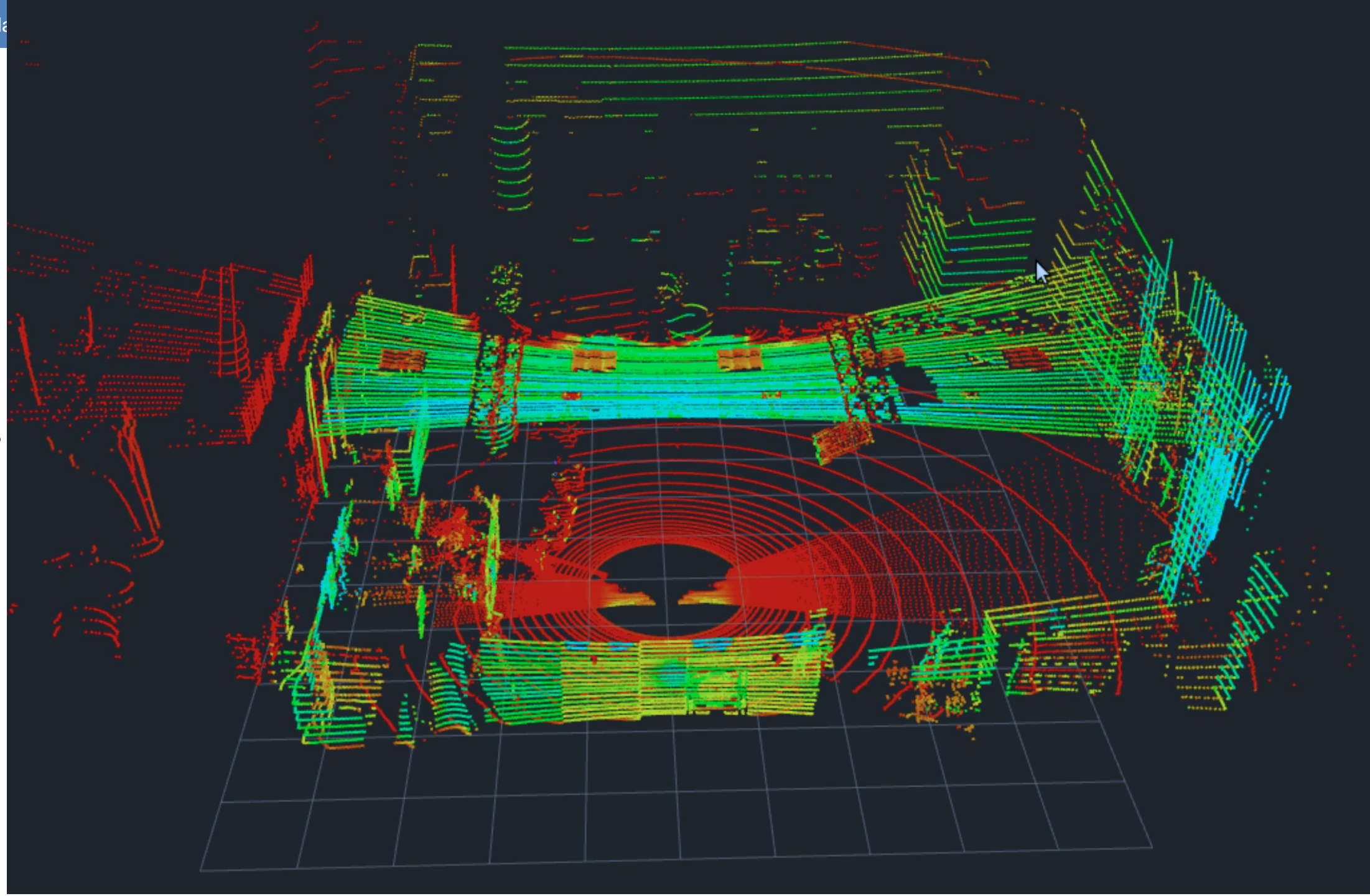
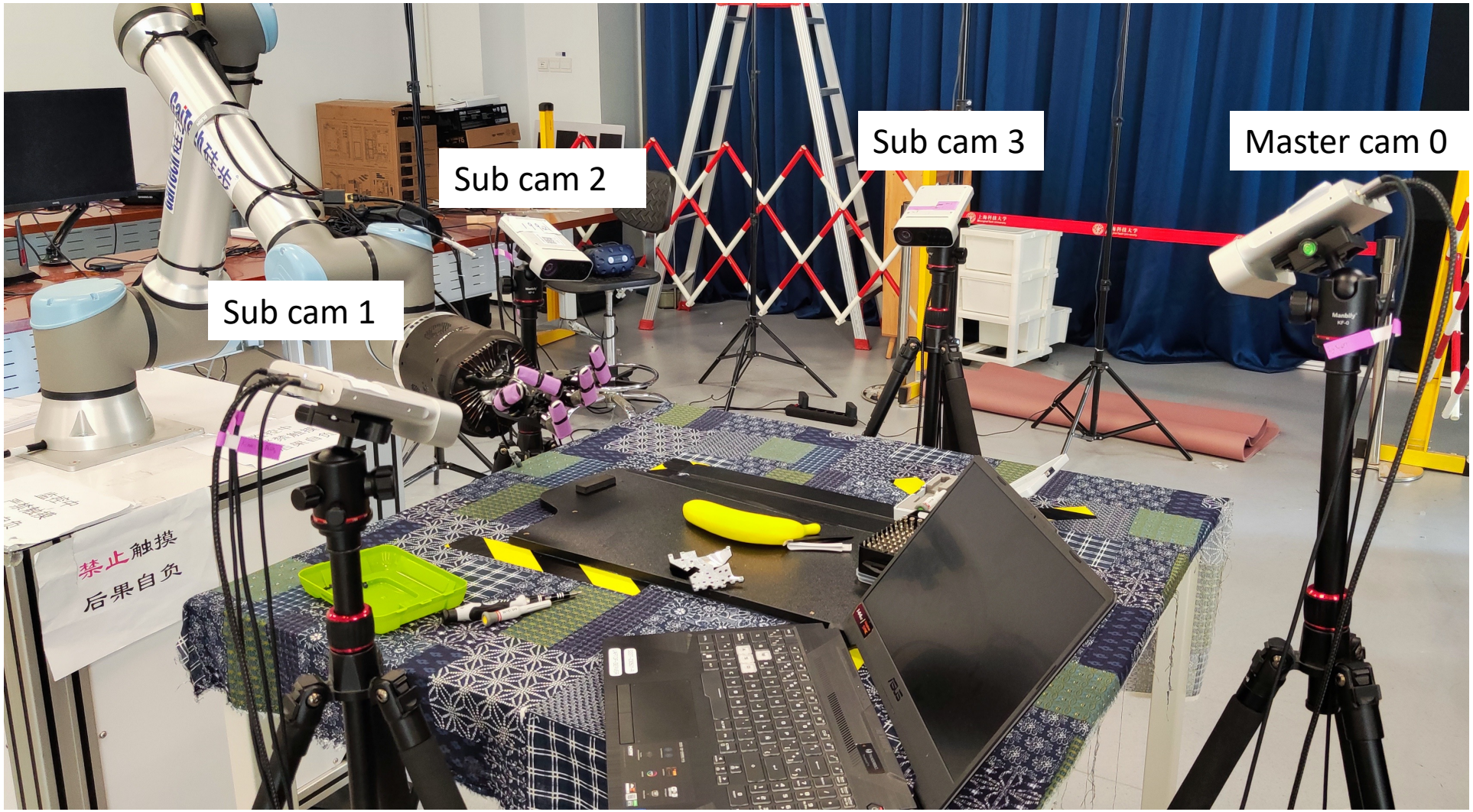


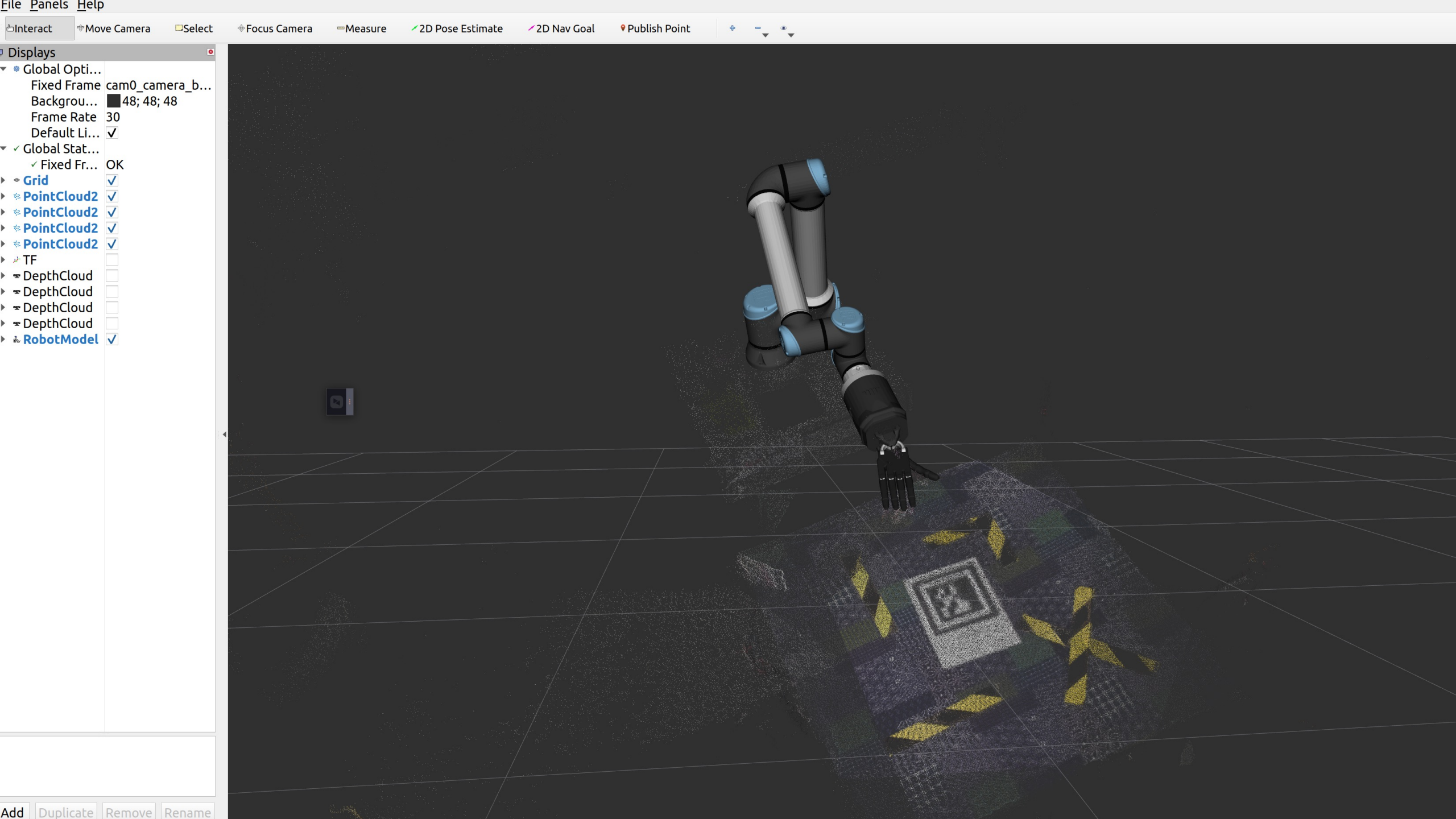
Fig. 7. A 3D Lidar scan colored by all the 7 horizontal cameras. The transformations between each camera and 3D Lidar are acquired from global optimization result. All the green points represent areas where no camera is overlapping with the pointcloud.

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



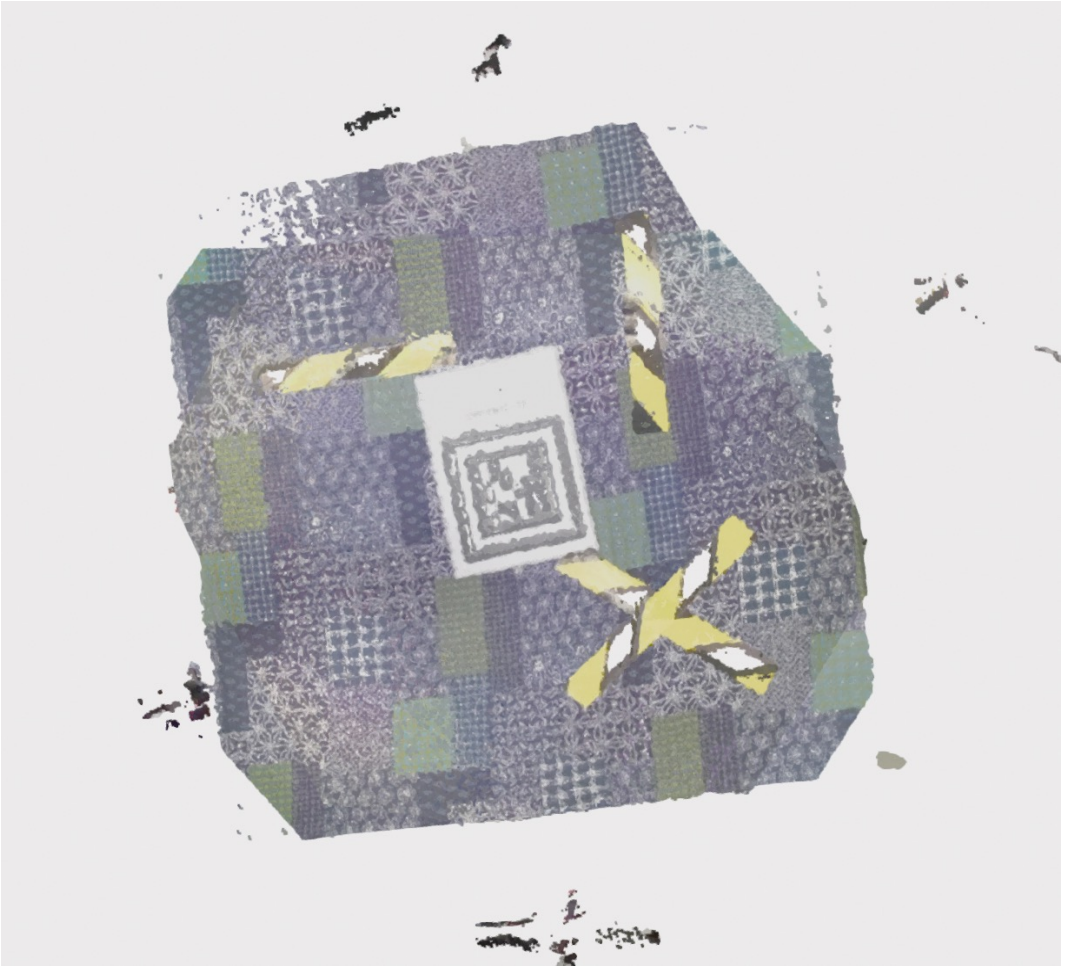
Shadowhand Experiment Example




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

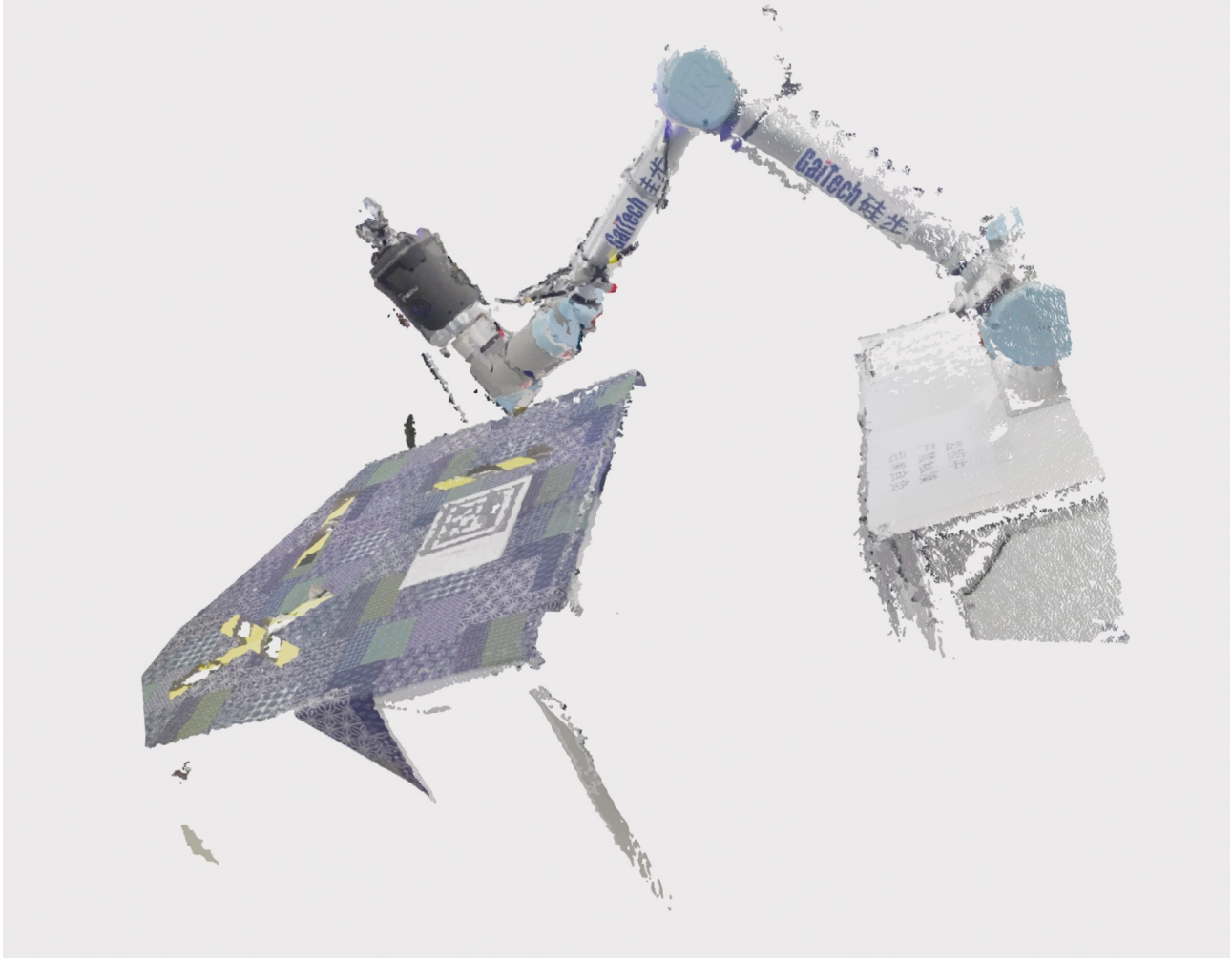
- Use robot hand hold the apriltag
- collect apriltag poses while moving the arm to different poses (do not move the hand)
- get handeye transformation by solving equation



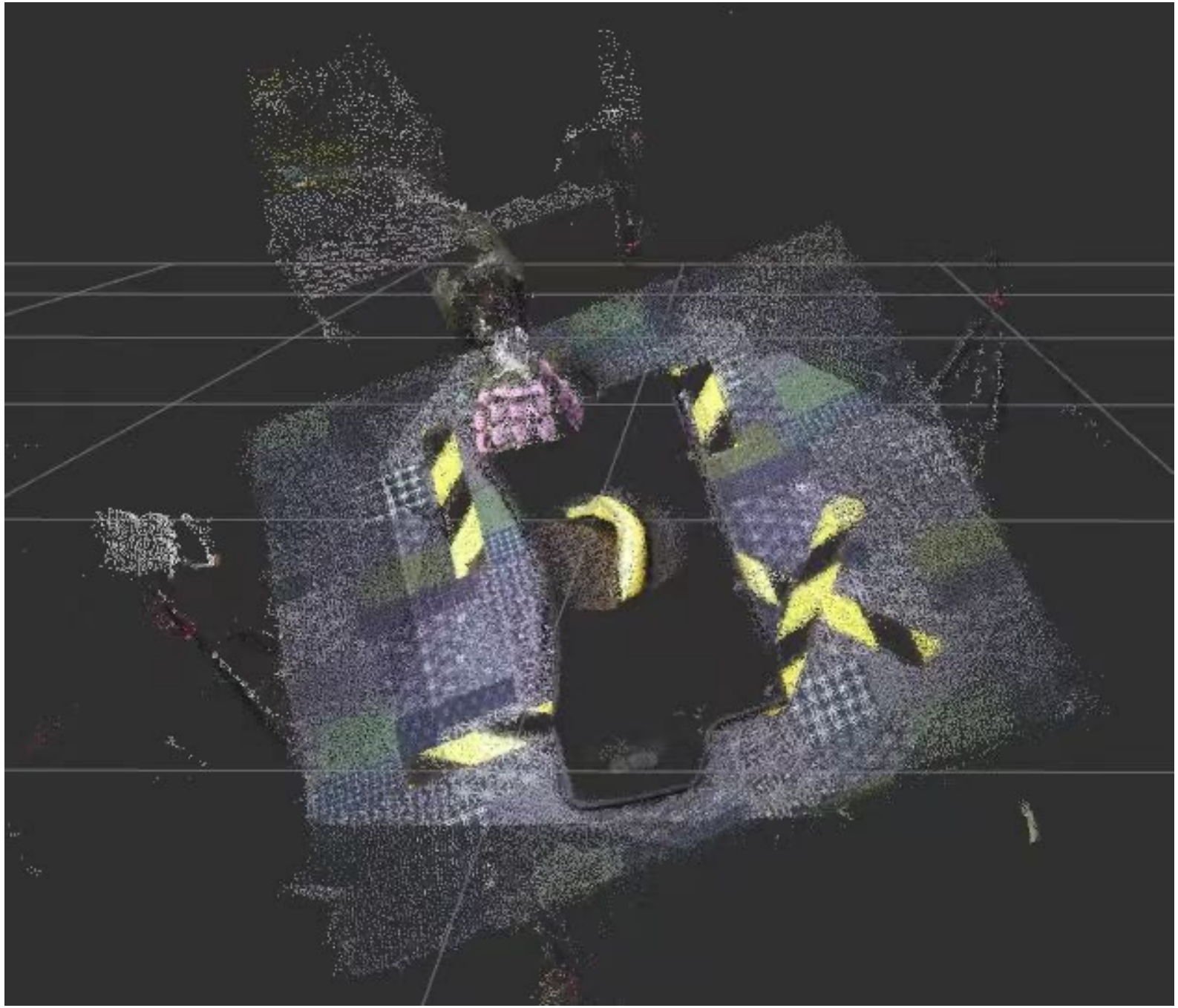
Hand eye calibration:

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

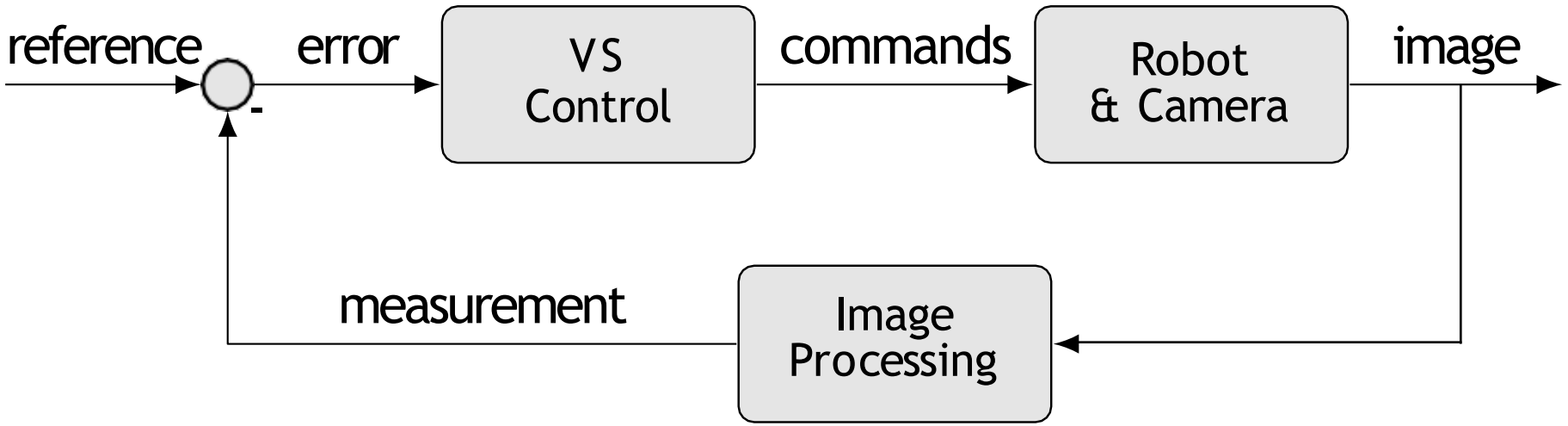
Visual Servoing: The Basic Problem

- 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.

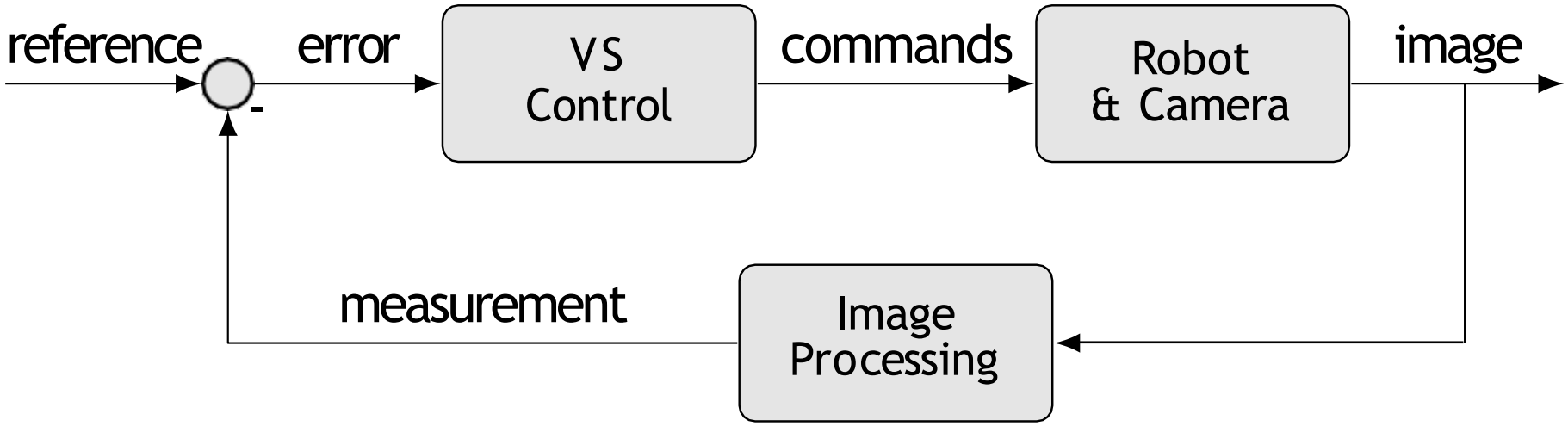
There are many variations on the problem:

- 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

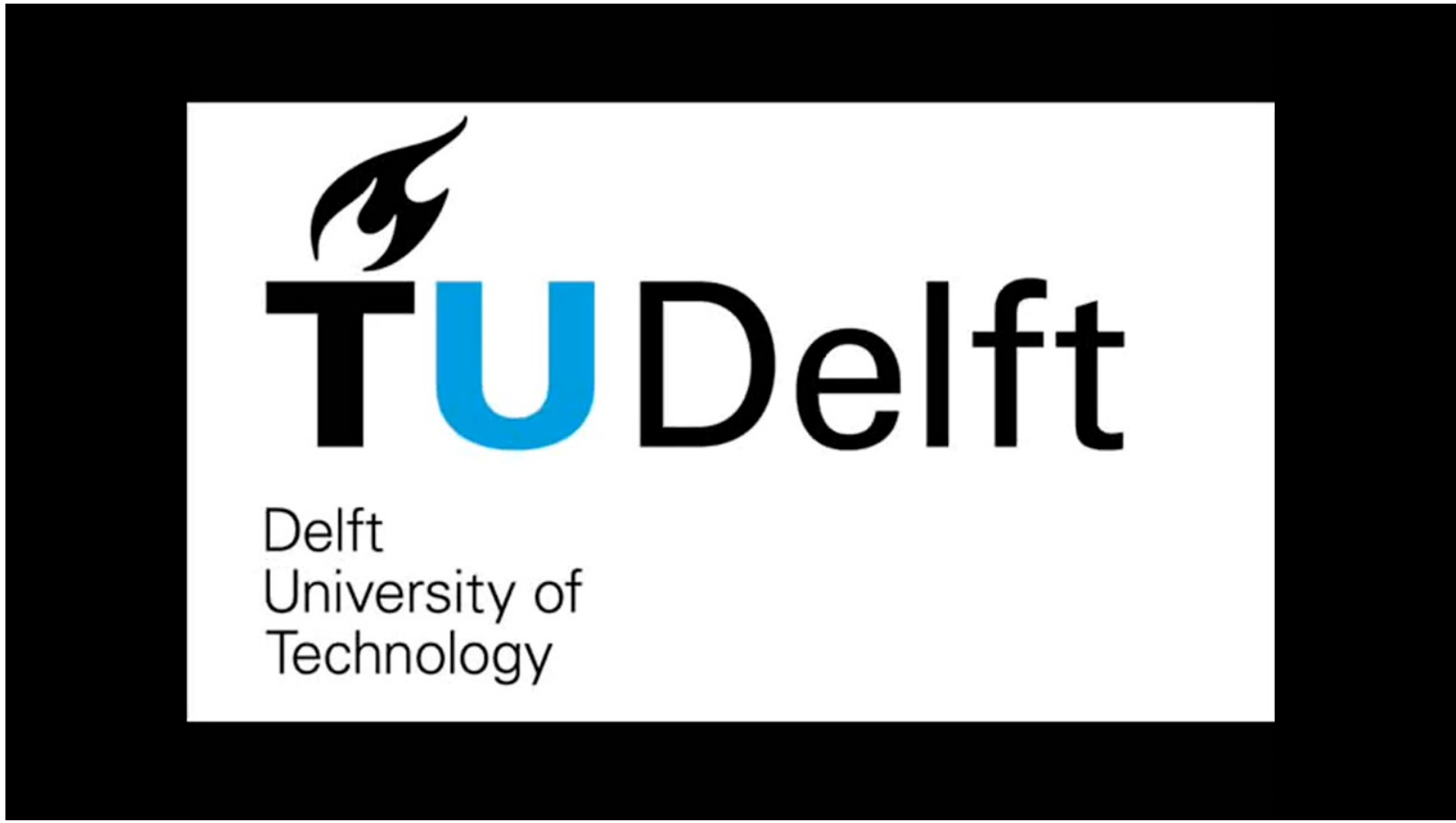
► Other options are also possible, such as *2.5D VS*

Simple example: 2D image based visual servoing

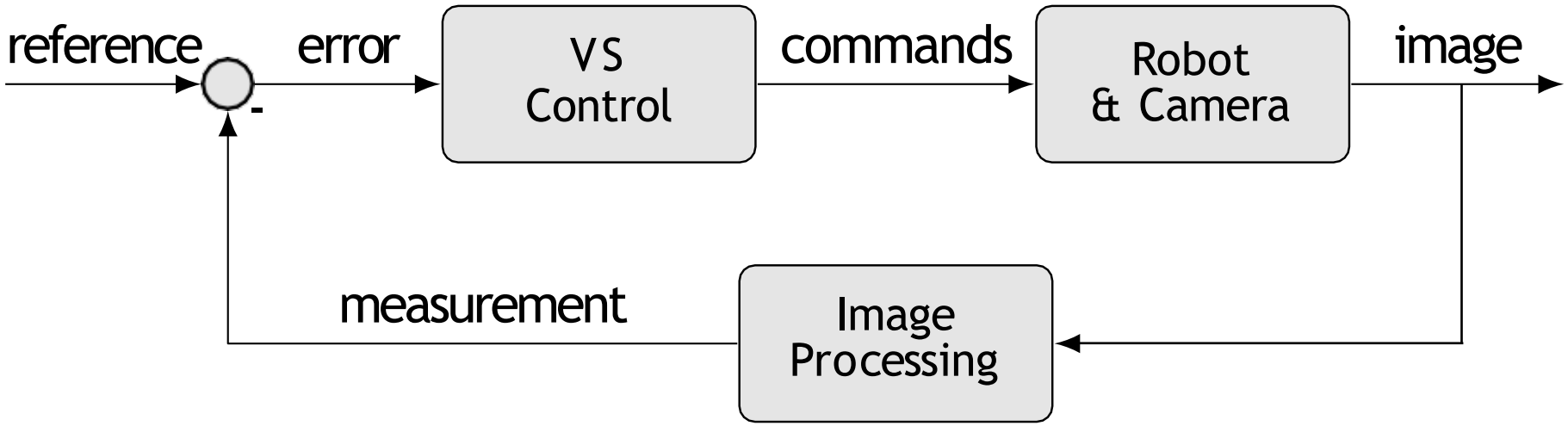
- If goal is above center, go up;
- if goal below center, go down,
- if goal is right, go right,
- if goal is left, go left

- 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

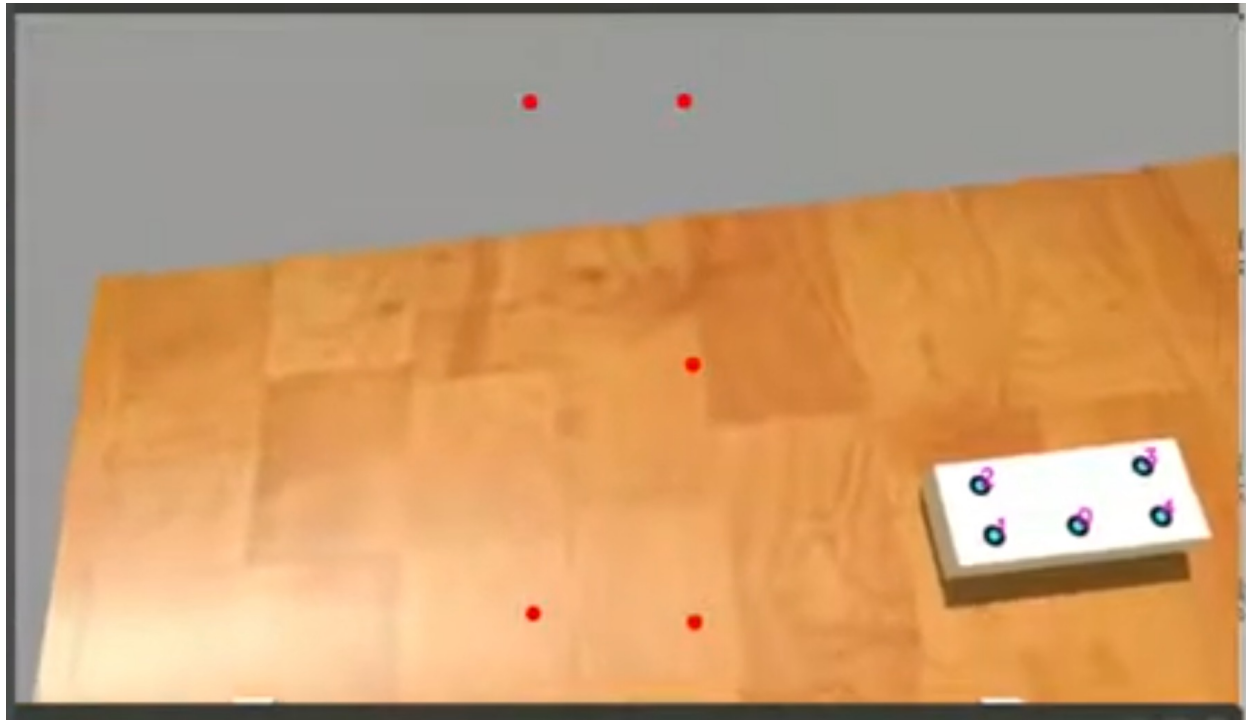
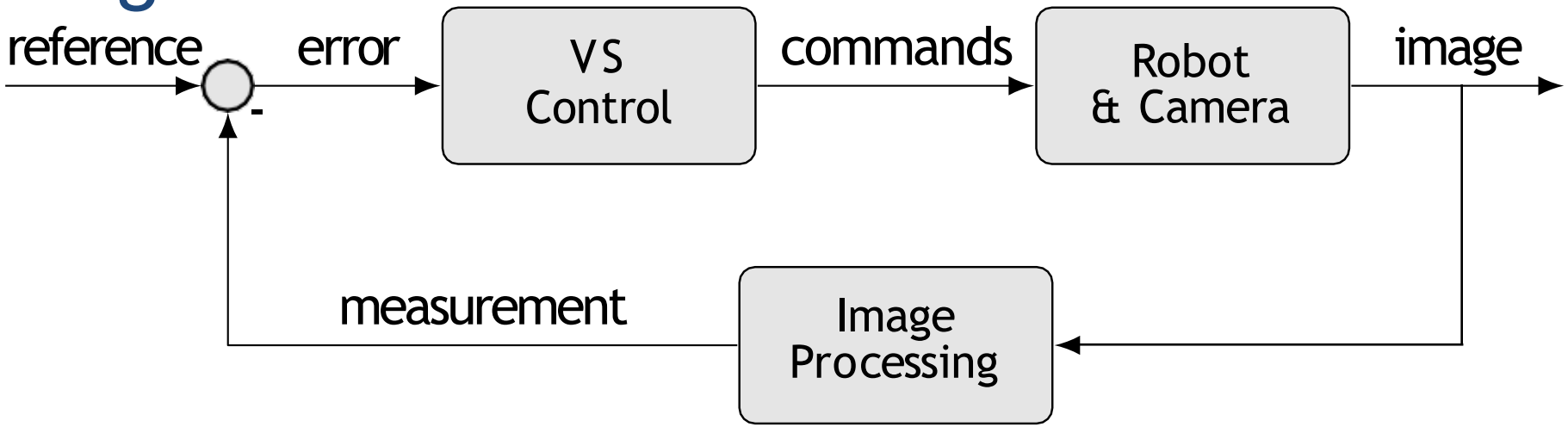


Image Based Visual Servoing (IBVS): block diagram



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*



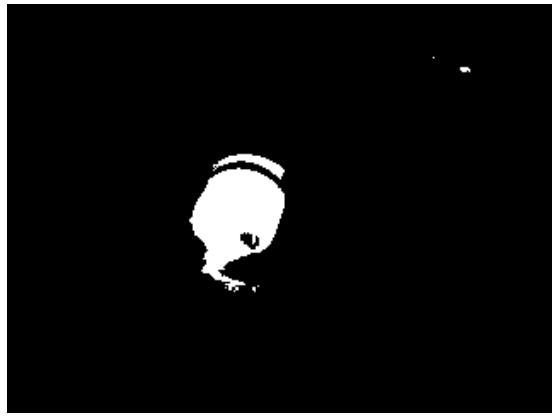
coordinates of the object centroid
2 *scalar numbers*

An example of image processing algorithm

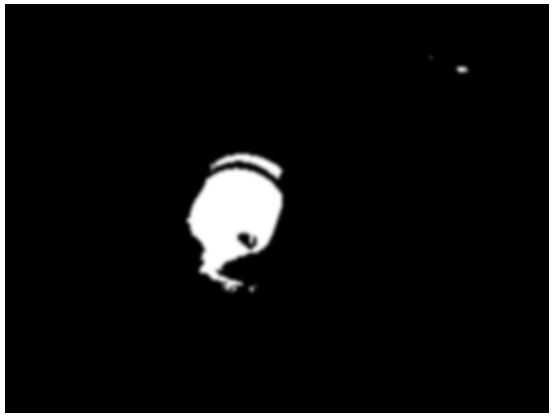
- Computer vision community provides many *ready-to-use* tools



original image



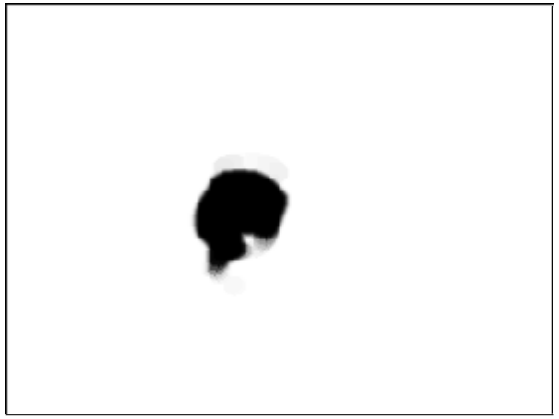
color filtering



smoothing



erode & dilate



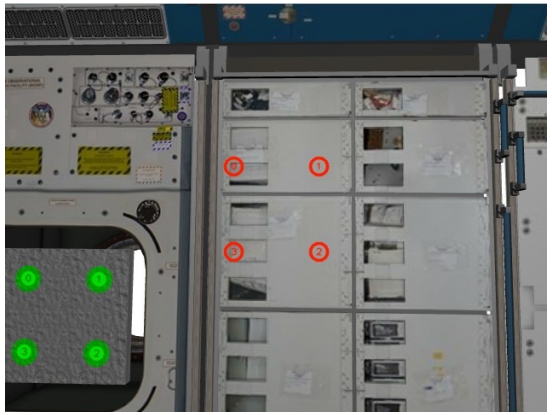
inversion



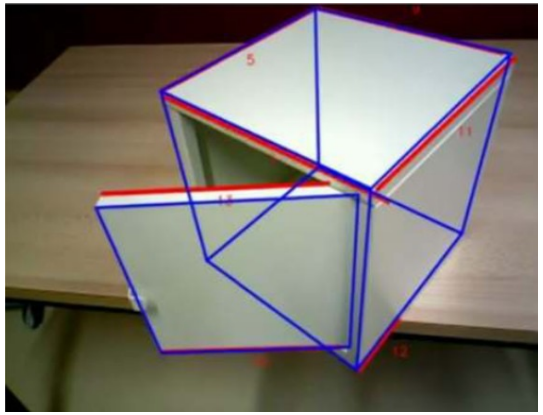
blob detection

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

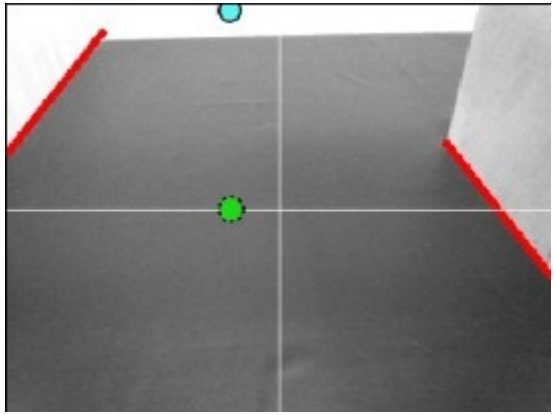
Examples of visual features



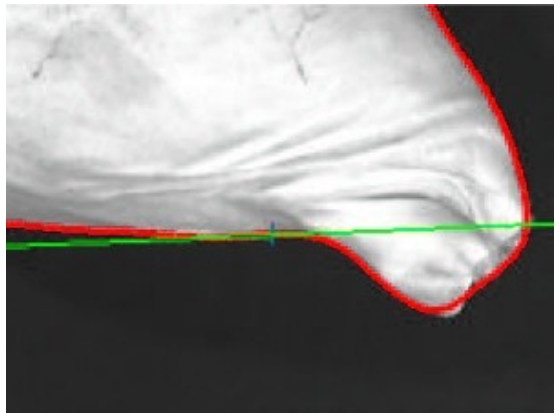
points



lines



reconstructed points



countours

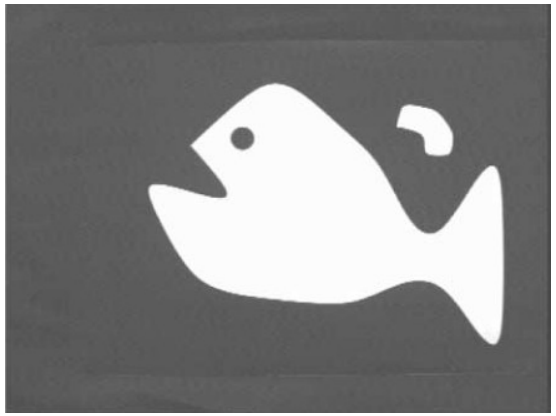


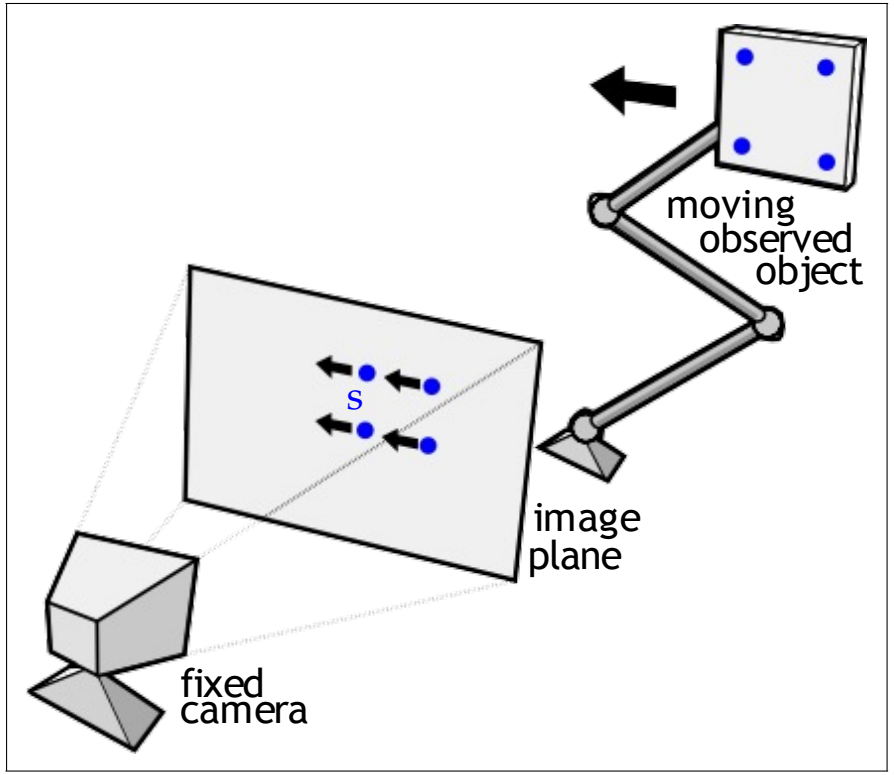
image moments

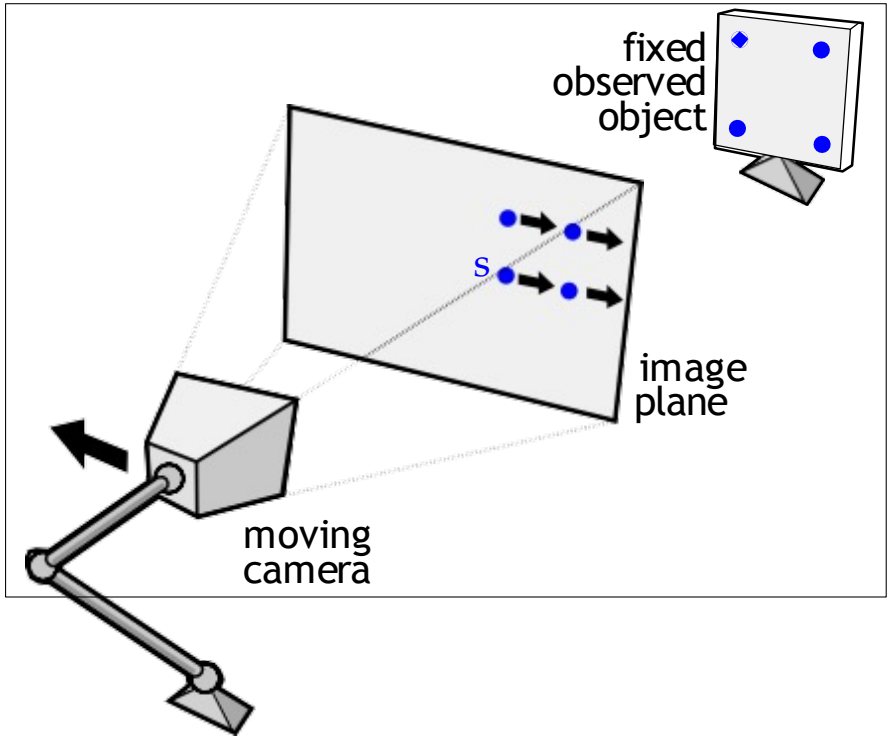


pixel luminance

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

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

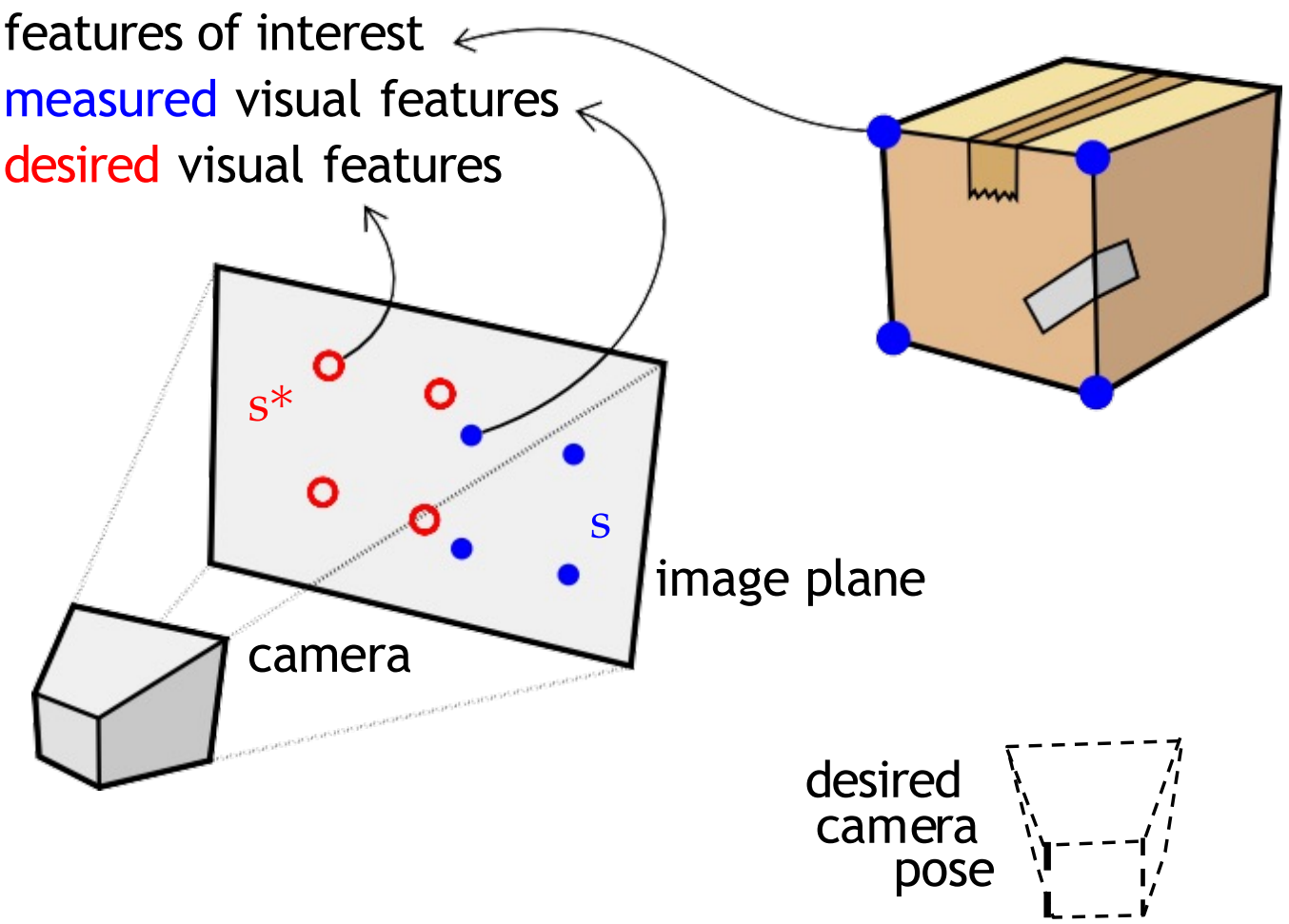




- *Eye-to-hand*: actuated target observed by a camera (left)
- *Eye-in-hand*: actuated camera observing a target (right)

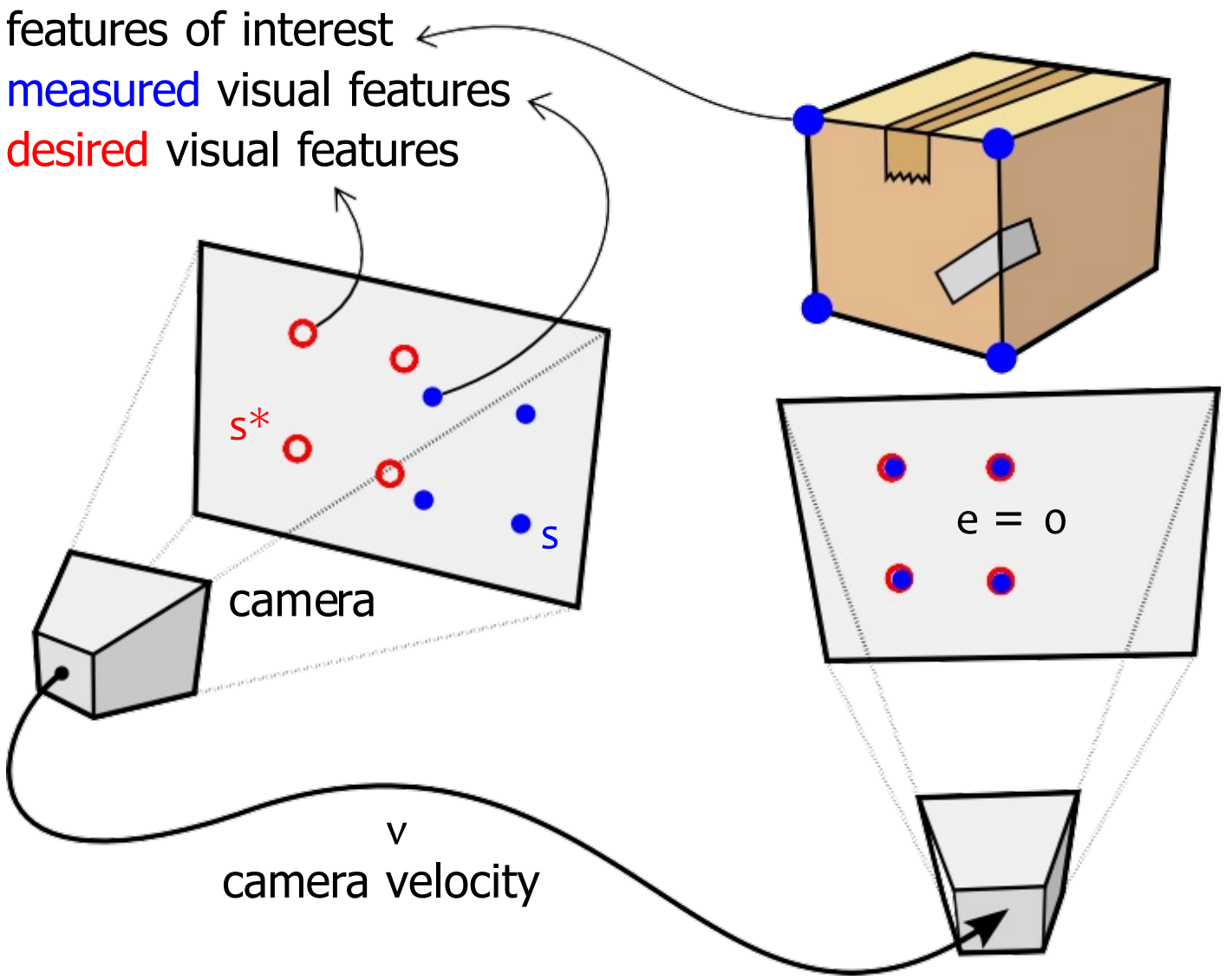
In this lecture we focus on *eye-in-hand* configurations

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 $\mathbf{e} = (\mathbf{s} - \mathbf{s}^*) \rightarrow \mathbf{0}$

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

where λ is the *control gain*

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

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

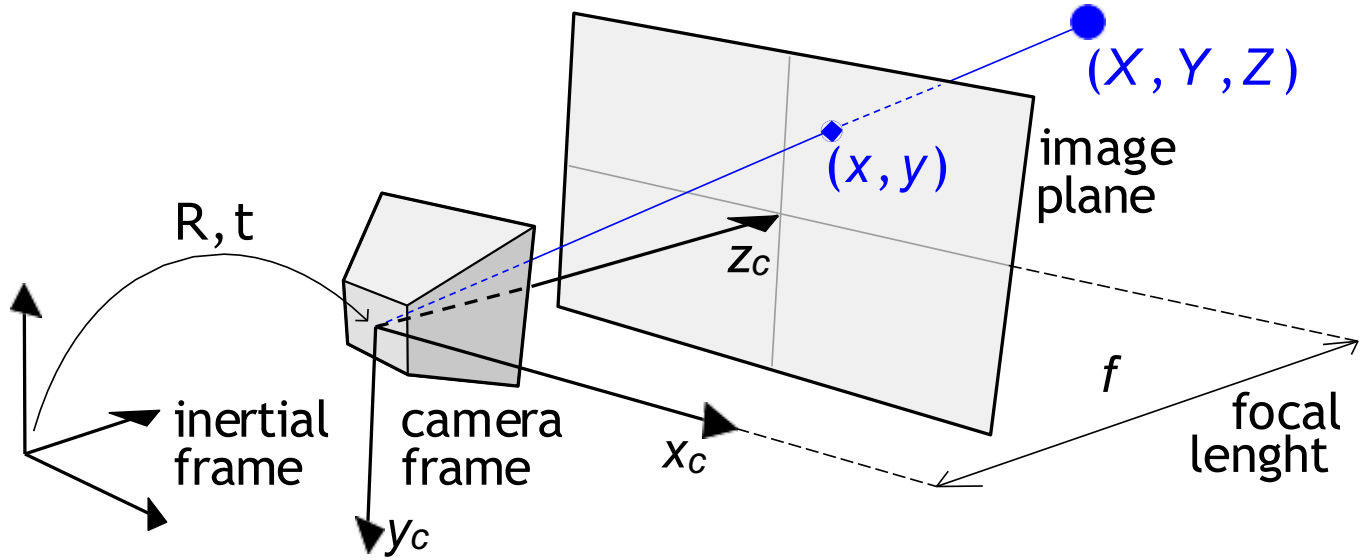
$\mathbf{L}^+$: Moore-Penrose pseudo-inverse: $\mathbf{L}^+ = (\mathbf{L}^T \mathbf{L})^{-1} \mathbf{L}^T$

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*:

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

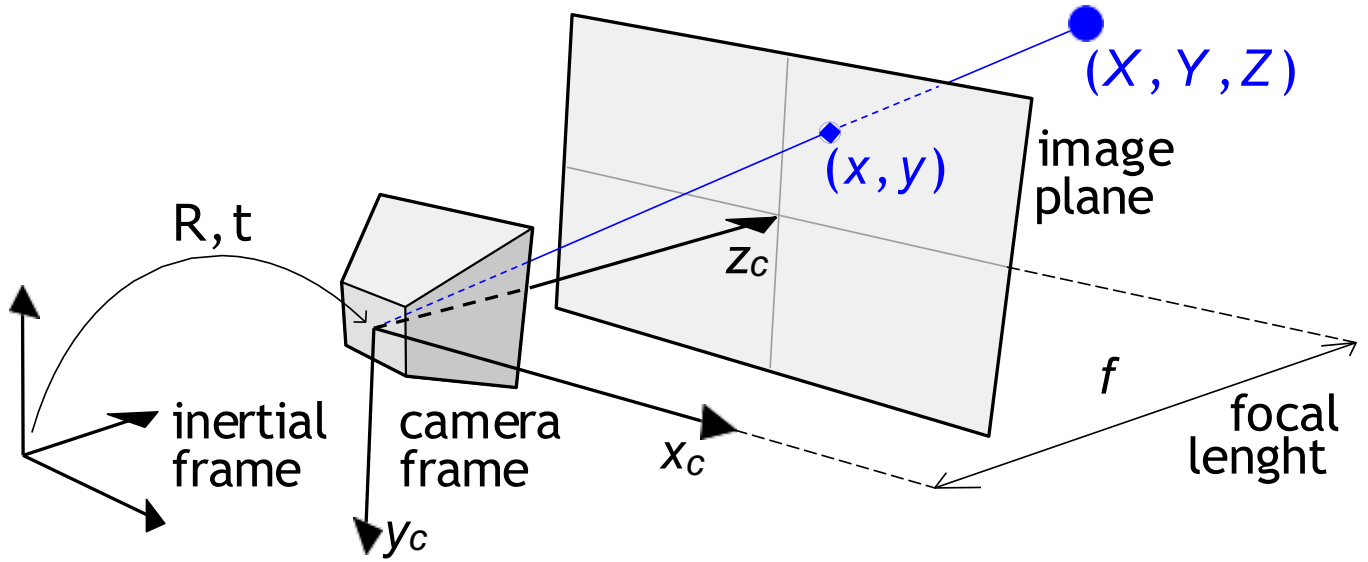
- The *depth* Z is *unkown* (remember the lost of information): we call it as *parameter* ζ in the left-hand side of the equation
- For convenience, we write the matrix as

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

- 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)

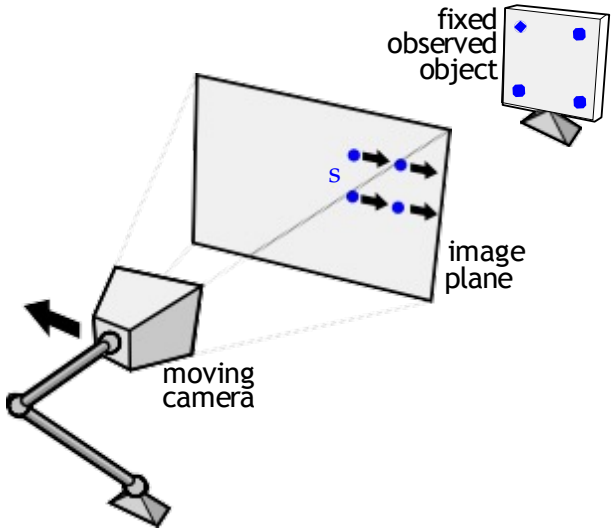
- Putting all together

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

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

- **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:

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

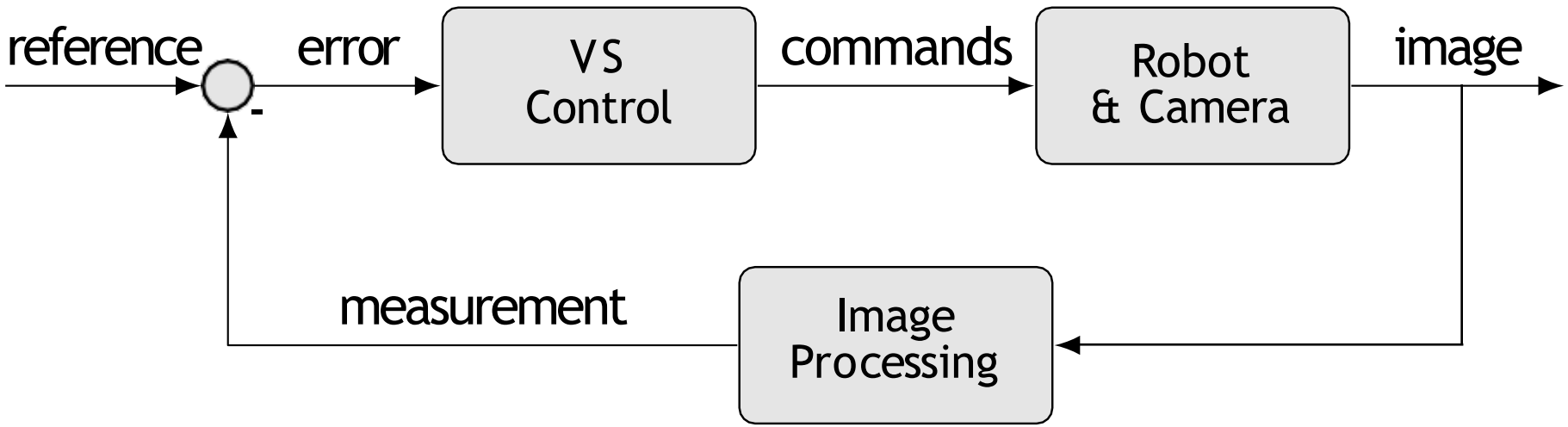
(Some) practical aspects of VS

- 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:

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

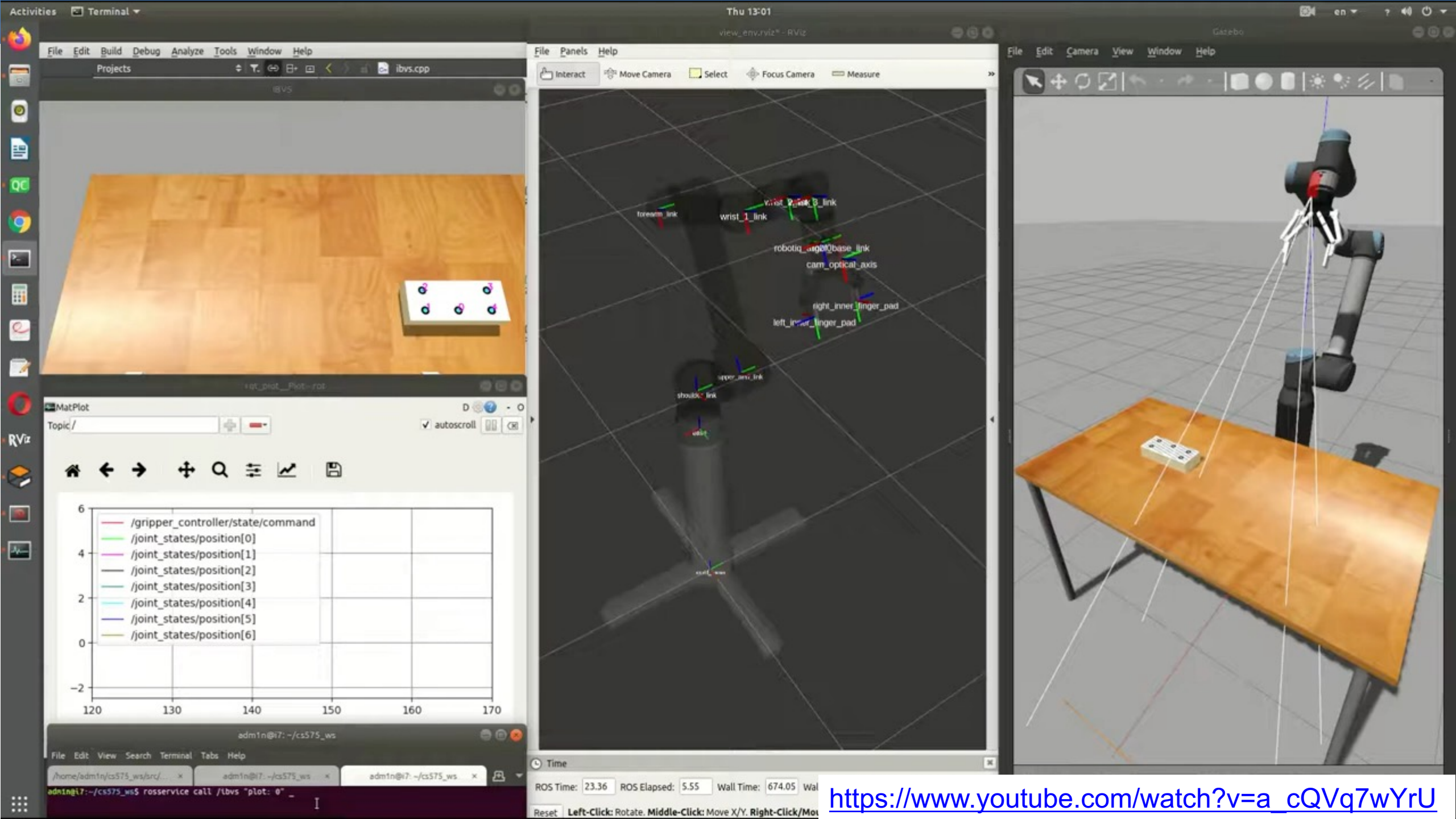
- The choise of the visual features, their number, and their desired value is part of the algorithm design

IBVS block diagram

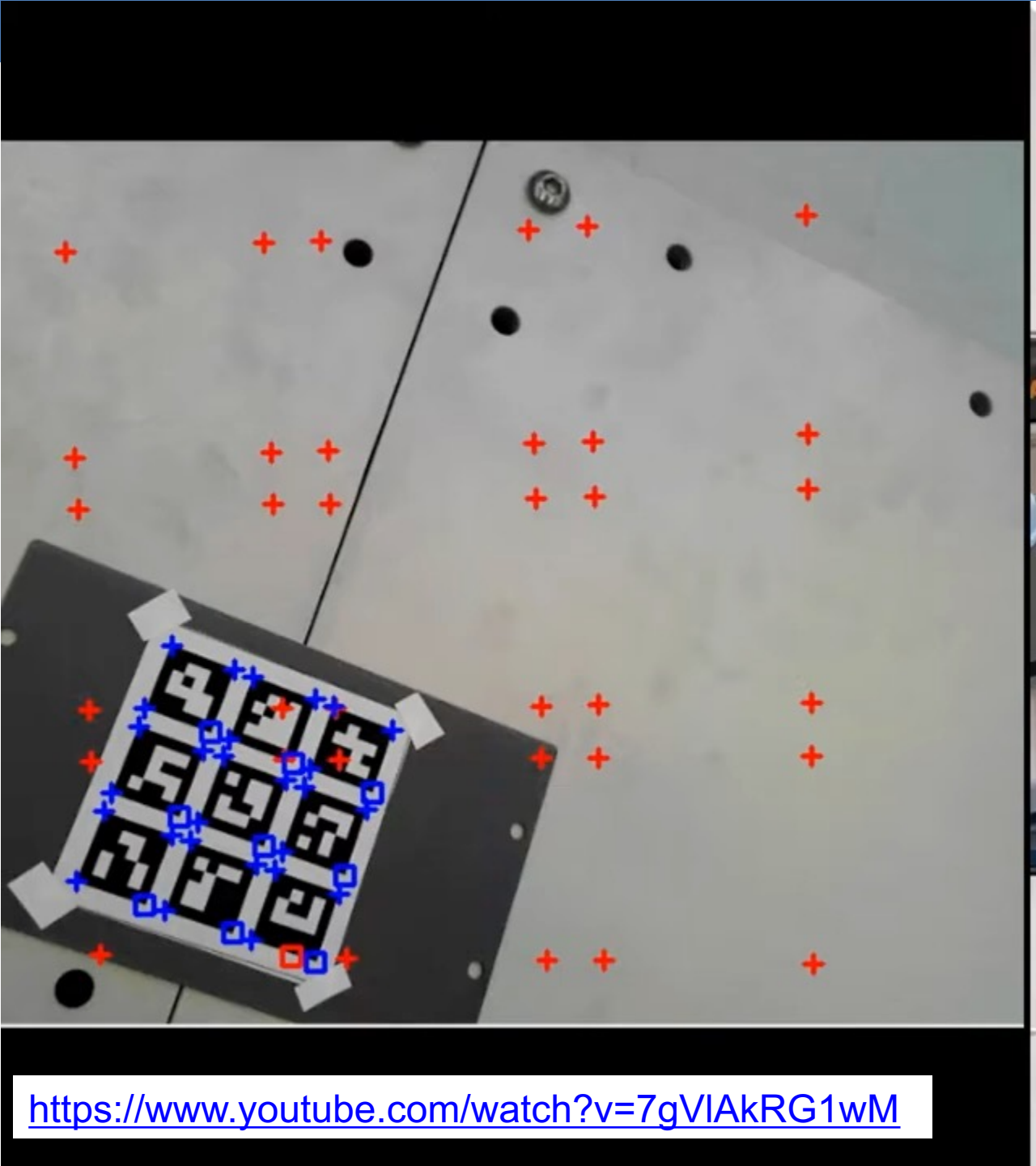


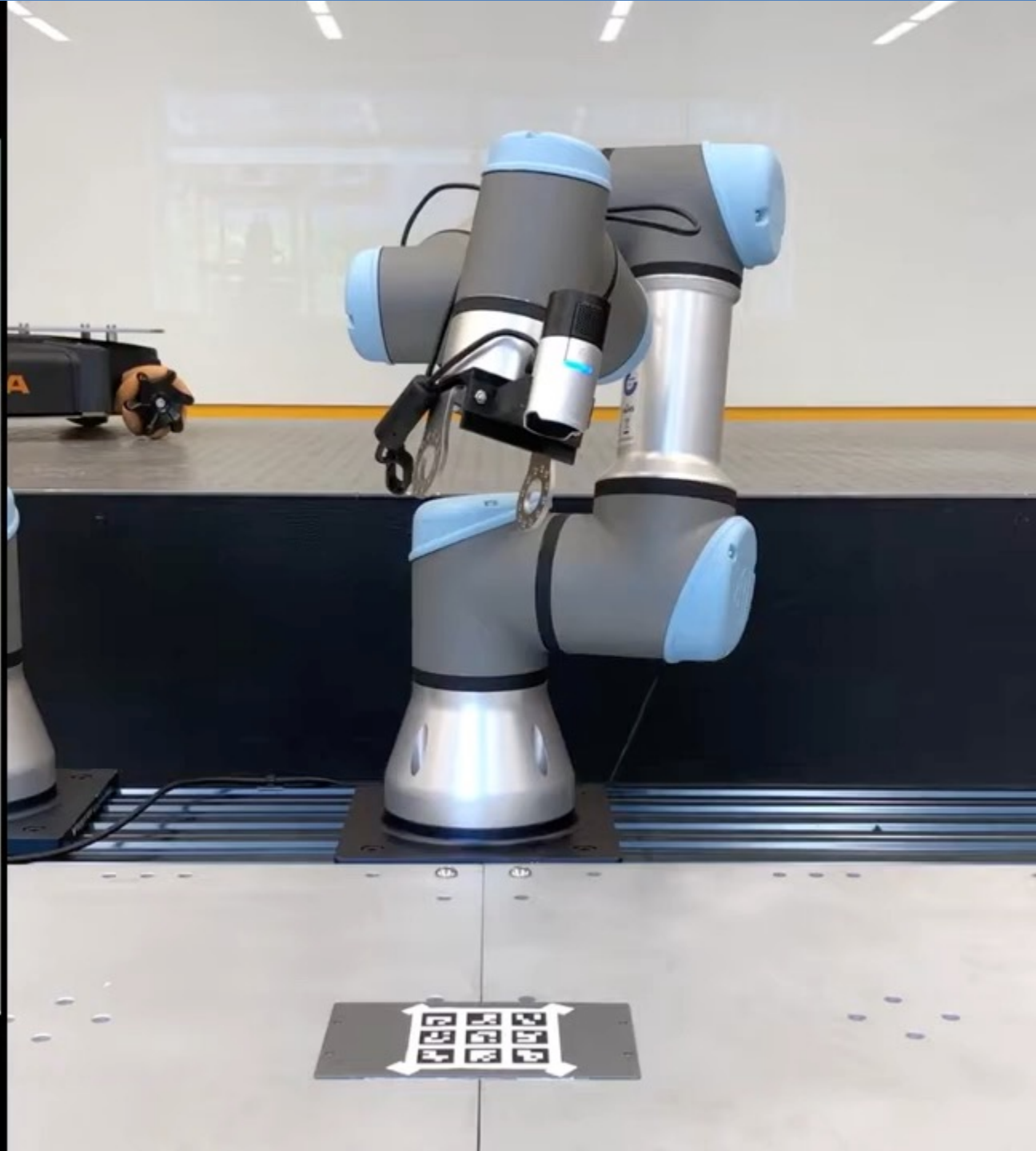
reference	desired visual features	$\mathbf{s}^*$
measurement	current visual features	$\mathbf{s}$
error	visual error	$\mathbf{e} = \mathbf{s} - \mathbf{s}^*$
commands	velocities command	$\mathbf{v} = -\lambda \mathbf{L}^+ \mathbf{e}$

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