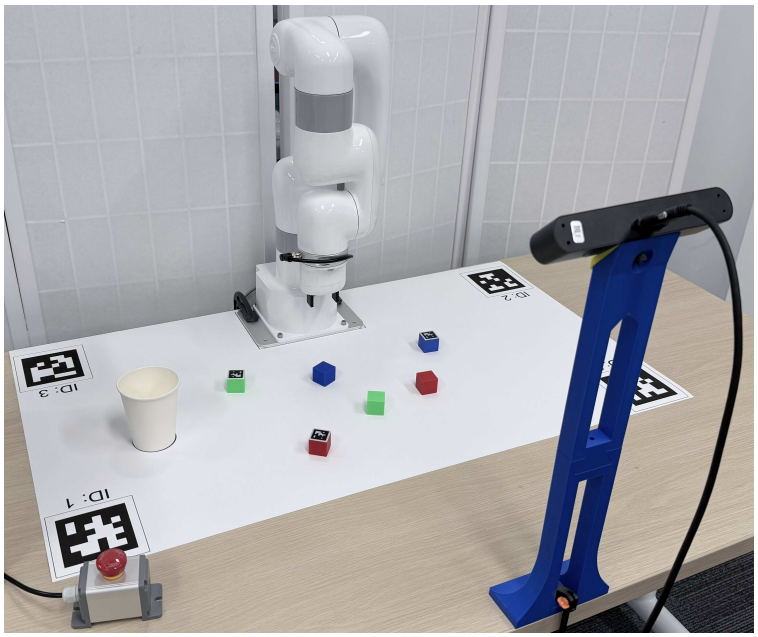
CSCI 5551: Real Robot Challenge

1 Overview



- This activity is designed to introduce fundamental concepts in robotic perception and manipulation through a structured, progressive challenge.
- The arena consists of a Lite6 robot, a ZED 2i stereo camera, an arena poster, and multiple colored cubes (with and without AprilTags). Each team will use the robot to manipulate objects and the camera to perceive the environment.
- The activity consists of two parts: **Checkpoints** and **Challenge Day**.
- **Checkpoints** are guided tasks designed to help you build a perception and manipulation pipeline for a stacking task. They are divided into two stages:
  - **AprilTag-Based Manipulation (Checkpoints 1–5)**, where object pose estimation is assisted by AprilTags.
  - **Pure Vision Manipulation (Checkpoints 6–10)**, where similar tasks must be completed without using AprilTags.
- **Checkpoint Evaluation:** After completing each checkpoint, your team must record a short demonstration using an external device (e.g., a phone) showing the robot successfully completing the task. This video will be used for checkpoint verification.
- **Challenge Day** consists of two competitive stacking tasks where teams apply and refine the system they developed during the checkpoints.

2 Lite 6 Robot

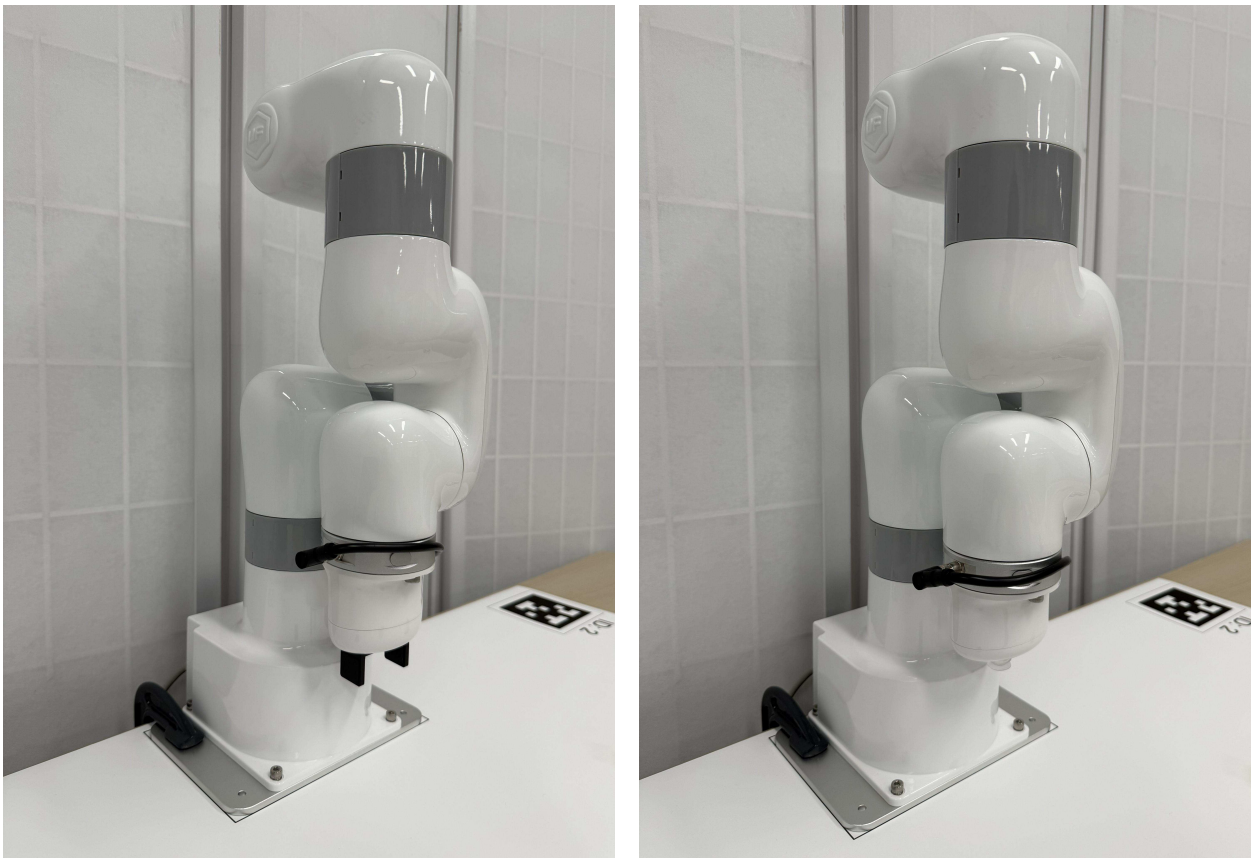


Figure 1: Lite 6 robot with parallel gripper and vacuum gripper

The Lite6 is a 6-DoF robotic arm developed by UFactory. In this challenge, you will use this robot to manipulate cubes on the table. The robot is controlled through the `xArm Python SDK`, which allows motion commands to be sent to the robot using Python.

The robot can also be controlled through a web-based controller provided by UFactory. This interface allows users to monitor the robot, move joints manually, and adjust basic settings. The web controller will be demonstrated during the lab session.

Gripper Types

Two types of grippers are available:

- **Parallel Gripper:** a gripper that grasps objects by closing two parallel fingers.
- **Vacuum Gripper:** a suction gripper that lifts objects using vacuum pressure.

Teams may choose which gripper to use depending on their strategy and task design.

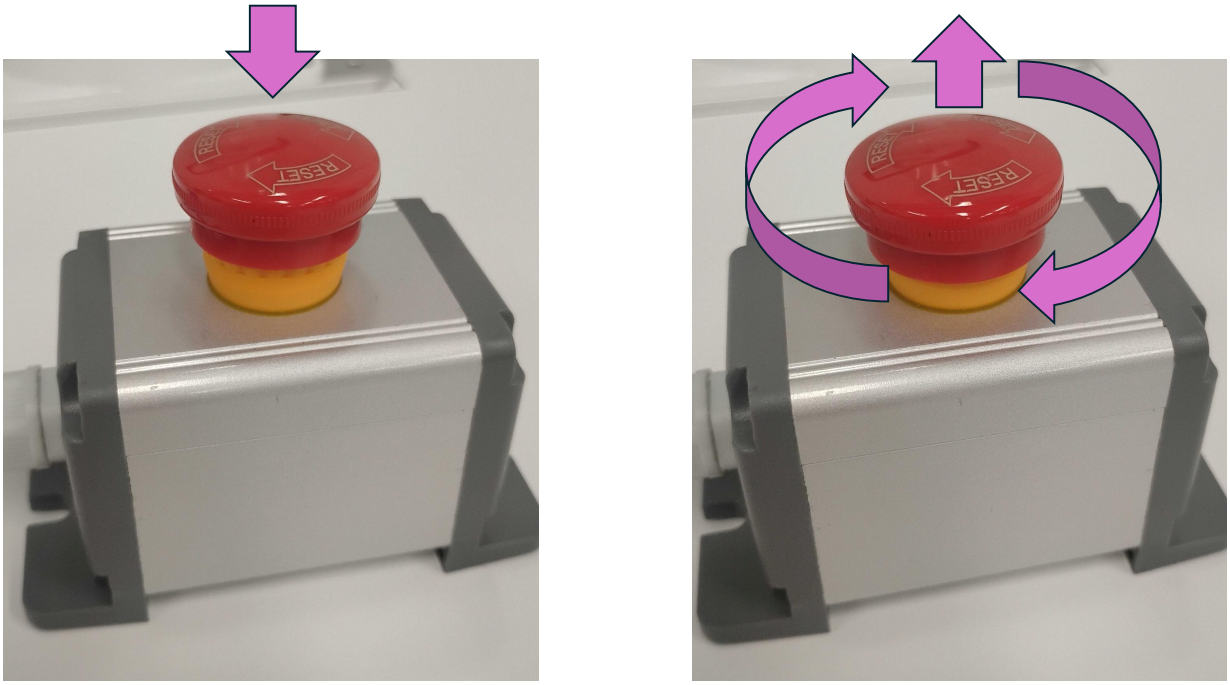
Free Drive Mode

The robot also supports a Free Drive Mode. In Free Drive Mode, the motors are relaxed and the robot arm can be moved manually by hand. This is useful for quickly positioning the robot or data collection. Free Drive Mode can be enabled from the web-based controller.

E-stop Button

In an emergency, the red e-stop (emergency stop) button can be used to instantly cut power to the robot. As shown in Figure 2, pressing the button down halts all motion. To release the e-stop and restore power, rotate the button clockwise following the printed arrows.

Note that e-stops can also be triggered automatically by collisions with the environment or people. After any e-stop event, the robot must be safely re-enabled and initialized through the GUI or API.



(a) Press down to stop the robot (b) Rotate clockwise to reset

Figure 2: The emergency stop (e-stop) button.

Robot Initialization & Shutdown

Before sending any motion commands, the robot must be initialized. The following code snippet shows the initialization procedure used in the provided template script.

```
arm = XArmAPI(robot_ip)
arm.connect()
arm.motion_enable(enable=True)
arm.set_tcp_offset([0, 0, GRIPPER_LENGTH, 0, 0, 0])
arm.set_mode(0)
arm.set_state(0)
arm.move_gohome(wait=True)
```

At the end of the program, the robot should be safely returned to its home position before disconnecting from the controller.

```
arm.move_gohome(wait=True)
arm.disconnect()
```

Moving the Robot

The following API command moves the end-effector to a target pose in the robot base frame.

`arm.set_position(x, y, z, roll, pitch, yaw, is_radian=None, wait=False)`

Parameters:

- `x, y, z`: Position of the end-effector in the robot base frame (mm).
- `roll, pitch, yaw`: Orientation of the end-effector (fixed-axis RPY order).
- `is_radian`: Indicates whether the orientation values are given in radians.
- `wait`: If `True`, the command blocks until the motion is completed.

Parallel Gripper

Use the following commands to open or close the parallel gripper and to stop the gripper motor. After opening or closing the gripper, always call `stop_lite6_gripper()` to prevent continuous motor operation and unnecessary noise.

```
arm.open_lite6_gripper()
arm.close_lite6_gripper()
arm.stop_lite6_gripper()
```

Vacuum Gripper

Use the following command to control the vacuum suction used for grasping.

```
arm.set_vacuum_gripper(on, wait=False)
```

Parameters:

- `on`: Enables or disables the vacuum suction.
- `wait`: If `True`, waits until the command is executed.

Additional API Documentation

The full Lite6 Python SDK is available online:

- SDK Repository
  `https://github.com/xArm-Developer/xArm-Python-SDK`
- API Source File
  `https://github.com/xArm-Developer/xArm-Python-SDK/blob/master/xarm/wrapper/xarm_api.py`

You are encouraged to explore the SDK if you want to use additional robot functionality.

3 ZED 2i Camera

ZED 2i stereo camera. Image source: Stereolabs.

The ZED 2i is a stereo RGB camera developed by Stereolabs. It provides high-resolution images and depth information using stereo vision and is a **passive stereo sensor**, meaning it estimates depth by comparing images captured by two cameras rather than emitting light or using an active depth projector.

In this challenge, the ZED camera is used to capture RGB images and generate a point cloud of the scene. These outputs will be used for object detection and pose estimation.

To simplify the camera interface, a wrapper class `ZedCamera` is provided. This wrapper handles camera initialization, background image capture, and data retrieval so that camera data can be accessed using simple function calls.

You are free to modify the wrapper class if additional functionality is needed.

Camera Initialization & Closing

The camera is initialized by creating a `ZedCamera` object, which starts the camera and launches a background thread that continuously captures images and point cloud data.

```
zed = ZedCamera()
```

When the program finishes, the camera should be closed to release hardware resources.

```
zed.close()
```

Retrieving Camera Data

The wrapper provides convenient properties for accessing camera data.

```
image = zed.image
point_cloud = zed.point_cloud
camera_intrinsic = zed.camera_intrinsic
```

- `image`: The BGRA image captured from the left camera.
- `point_cloud`: A dense point cloud where each pixel contains four values (X, Y, Z, unused).
- `camera_intrinsic`: The camera intrinsic matrix.

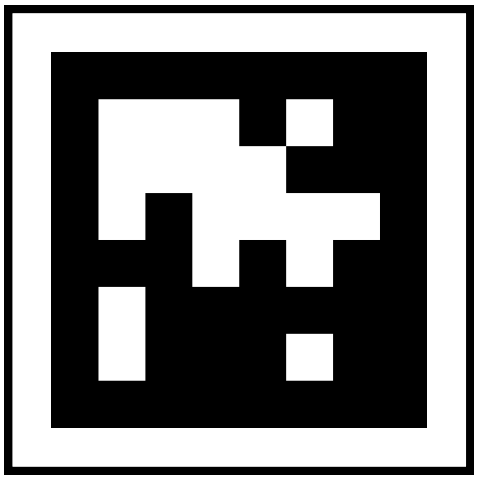
Additional API Documentation

The full ZED SDK documentation is available online:

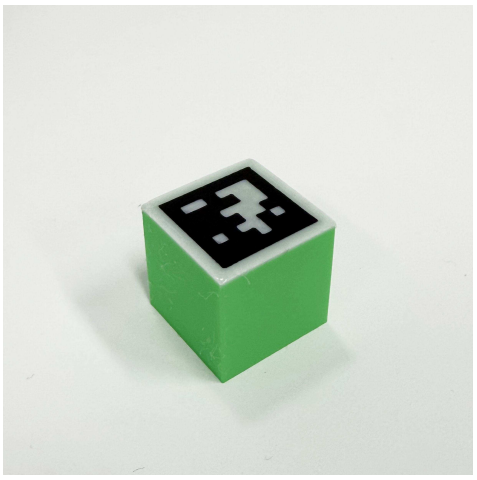
- Python API Documentation
  `https://www.stereolabs.com/docs/api/python`
- ZED SDK Repository
  `https://github.com/stereolabs/zed-sdk`

You are encouraged to explore the SDK if you wish to use additional camera functionality.

4 AprilTag Detection



(a) AprilTag



(b) A cube with AprilTag

AprilTags are square fiducial markers that can be detected reliably in an image and used for pose estimation. In the first stage of the checkpoints, you will use AprilTags attached to the cubes to simplify the perception pipeline and easily determine the cube's position and orientation.

The provided template script uses the `pupil_apriltags` package to detect tags and estimate their poses. You are encouraged to explore other packages if additional functionality is needed.

Tag Configuration

The following AprilTag settings are used for the cube in this challenge:

- **Tag Family:** tag36h11
- **Cube Tag ID:** 4
- **Tag Size:** 0.0205 m

API Documentation

The full API documentation for the `pupil_apriltags` detector is available online:
`https://pupil-apriltags.readthedocs.io/en/stable/api.html`

Note: The `pupil_apriltags` detector operates on grayscale images.

Checkpoint 0: Workspace Registration [Provided]

Figure 4: Visualization of the estimated robot base frame origin.

Goal

The goal of this preliminary checkpoint is to establish the coordinate transformation between the ZED camera frame and the Lite6 robot's base frame.

Description

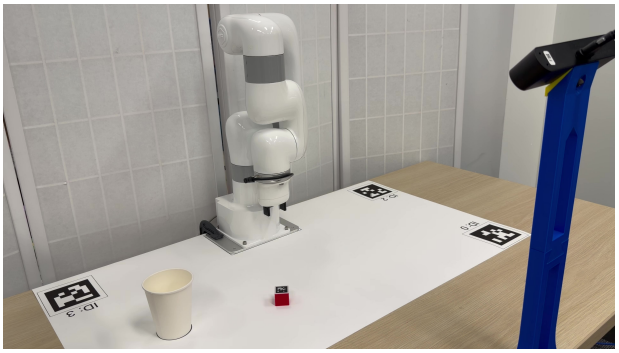
To successfully manipulate an object, the robot must know exactly where the object is relative to its own base. However, the ZED camera detects objects relative to its own lens (the camera frame). Therefore, we must calculate a transformation matrix (`t_cam_robot`) that bridges these two spatial coordinate systems.

This script is fully provided for you in `checkpoint0.py`. It uses four reference AprilTags printed on the arena poster at precise, known 3D coordinates to automatically calculate the exact position and orientation of the camera relative to the robot's base.

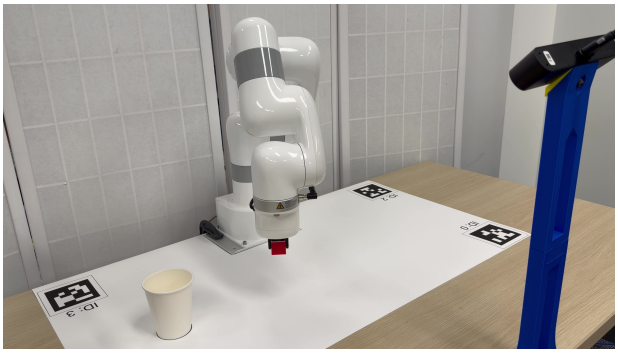
You do not need to write any code for this checkpoint. However, you will import and call the `get_transform_camera_robot()` function from this script in every subsequent checkpoint. To verify the calibration, run the script. A window will pop up showing the 3D axes drawn directly over the robot's base origin.

CSCI 5551: Real Robot Challenge

Checkpoint 1: Basic Grasp (AprilTag) [1 pt]



(a) Initial State



(b) Target State

Goal

The goal of this checkpoint is to estimate a single cube's pose using its attached AprilTag, grasp it, and safely put it back down.

Description

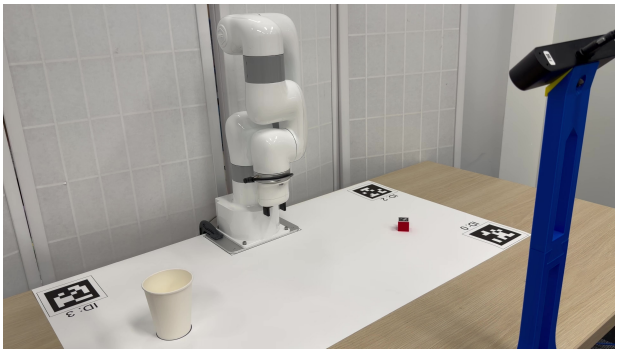
In this checkpoint, there will be only one cube in the arena. You are provided with a template script (`checkpoint1.py`) that contains the necessary skeleton code. To complete this task, you need to fill in the blank `TODO` sections within the `main()` function to sequence the robot's actions, as well as implement the following core functions:

- `get_transform_cube()`: Estimate the pose of the cube.
- `grasp_cube()`: Command the robot to pick up the cube.
- `place_cube()`: Command the robot to place the cube.

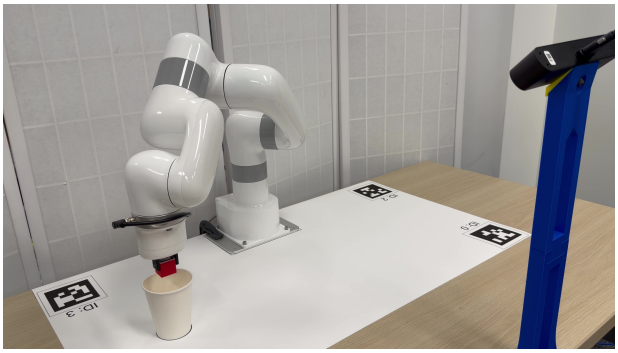
Hints

- The template script opens a visualization window showing the estimated 3D axes on the target cube. If the pose is accurate, press the **'k'** key to execute the robot's motion.
- Always ensure the robot moves to a safe pre-grasp height before descending to pick the object.

Checkpoint 2: Pick and Place (AprilTag) [1 pt]



(a) Initial State



(b) Target State

Goal

The goal of this checkpoint is to grasp a single tagged cube and drop it into a paper cup located on the side of the arena.

Description

In this checkpoint, there will be only one cube in the arena and a paper cup serving as the drop-off location. You are provided with a template script (`checkpoint2.py`). Use the robot's Free Drive Mode to determine the cup's static location and update the `BASKET_POSE` constant. To complete this task, fill in the `TODO` sections within the `main()` function to sequence the robot's actions, and implement the following core function:

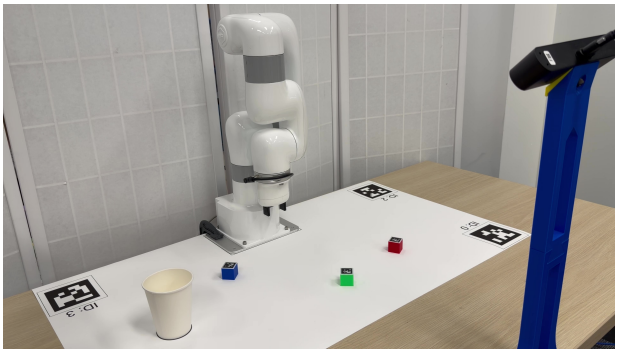
- `place_in_basket()`: Command the robot arm to move to the basket location and release the grasped object.

Hints

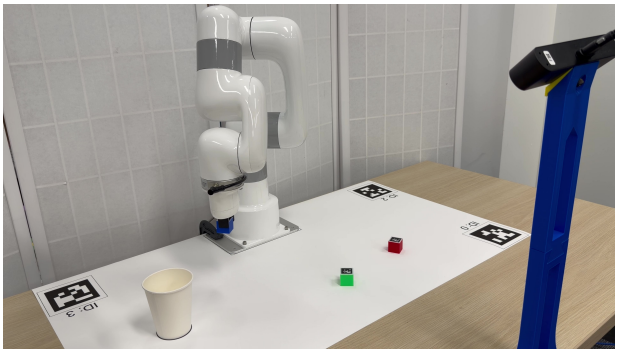
- Use the UFactory web controller to enable Free Drive Mode, manually move the robot's end-effector over the paper cup, and read the pose coordinates to set your `BASKET_POSE`.
- Ensure the robot moves high enough to clear the lip of the paper cup before dropping the cube.

CSCI 5551: Real Robot Challenge

Checkpoint 3: Target Selection (AprilTag) [1 pt]



(a) Initial State



(b) Target State

Goal

The goal of this checkpoint is to identify and grasp a specific cube among multiple tagged cubes based on a semantic text prompt (e.g., 'blue cube'), and safely put it back down.

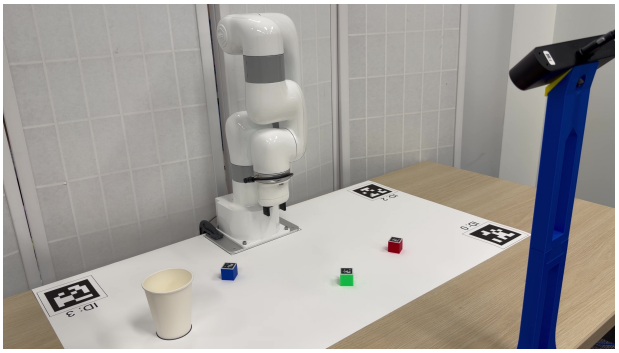
Description

In this checkpoint, there will be three cubes in the arena (red, blue, and green). All three cubes have an AprilTag attached. You are provided with a template script (`checkpoint3.py`). To complete this task, you need to fill in the `TODO` sections within the `main()` function to sequence the robot's actions, and implement the `CubePoseDetector` class.

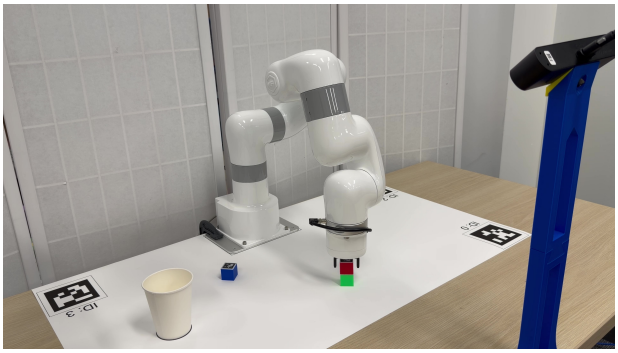
Hints

- You have the freedom to choose your detection method. You can use traditional computer vision techniques (like HSV color thresholding) or leverage more advanced zero-shot segmentation and detection models.
- You can reuse your `grasp_cube()` and `place_cube()` functions from Checkpoint 1.

Checkpoint 4: Stacking (AprilTag) [2 pt]



(a) Initial State



(b) Target State

Goal

The goal of this checkpoint is to pick up the red cube and successfully stack it directly on top of the green cube.

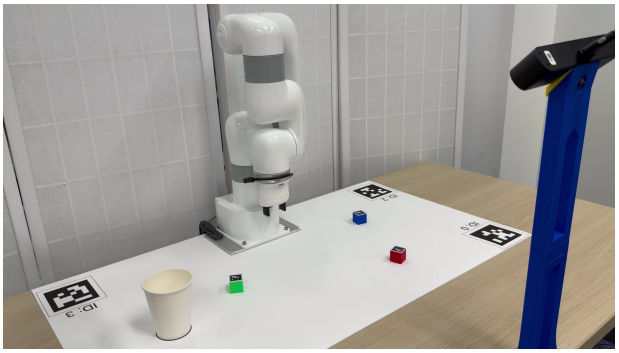
Description

In this checkpoint, there will be three cubes in the arena (red, blue, and green). All three cubes have an AprilTag attached. You are provided with a template script (`checkpoint4.py`). To complete this task, you need to reuse your `CubePoseDetector` from Checkpoint 3 to locate both the red and green cubes. Then, determine a suitable `STACK_HEIGHT` constant, and fill in the `TODO` sections within the `main()` function to sequence the robot's actions.

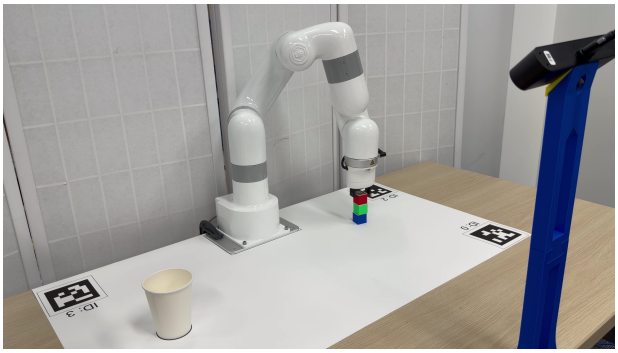
Hints

- Determine `STACK_HEIGHT` carefully. It should be large enough to clear the green cube, but small enough to ensure a stable drop without bouncing or tipping.
- You can reuse your `grasp_cube()` and `place_cube()` functions from Checkpoint 1.

Checkpoint 5: Sequential Stacking (AprilTag) [1 pt]



(a) Initial State



(b) Target State

Goal

The goal of this checkpoint is to identify all three tagged cubes and stack them into a single tower according to a specific order.

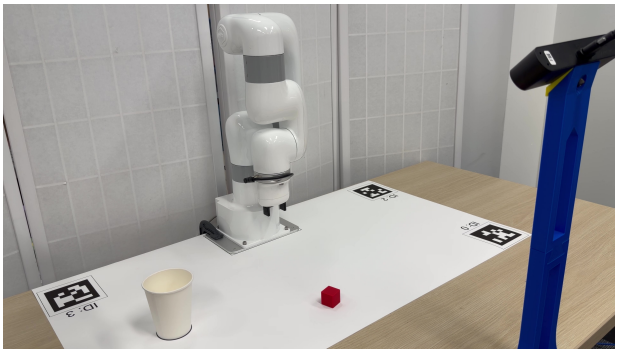
Description

In this checkpoint, the red, green, and blue cubes will all be in the arena. You are provided with a template script (`checkpoint5.py`) that includes a `stacking_order` list defining the tower from top to bottom. To complete this task, fill in the `TODO` sections within the `main()` function to locate all three cubes and sequence the robot's actions to build the stack.

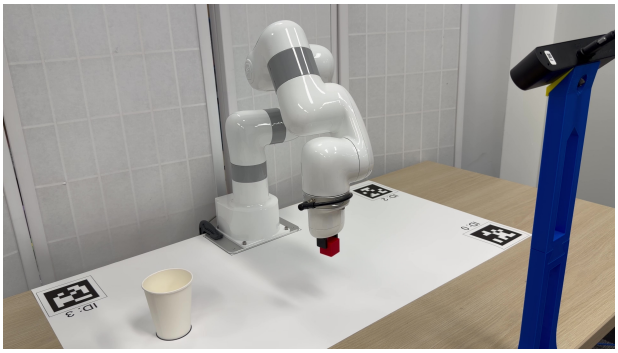
Hints

- You will reuse your `CubePoseDetector`, `STACK_HEIGHT`, and pick-and-place functions from the previous checkpoints.

Checkpoint 6: Basic Grasp (Pure Vision) [2 pt]



(a) Initial State



(b) Target State

Goal

The goal of this checkpoint is to estimate a single cube's pose using only raw camera data (image and point cloud), grasp it, and safely put it back down.

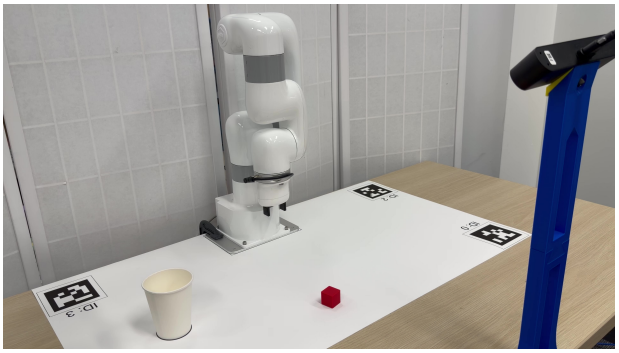
Description

In this checkpoint, the AprilTag is removed. You are provided with a template script (`checkpoint6.py`) that passes both the 2D image and the 3D point cloud into the perception function. Use the data to determine the object's pose. To complete this task, fill in the `TODO` sections within the `main()` function to sequence the robot's actions, and implement the function `get_transform_cube()`.

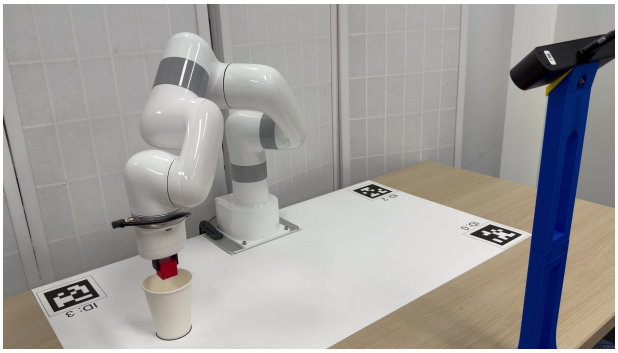
Hints

- Raw point clouds often contain invalid data. Remember to filter out `NaN` values before processing your 3D points.
- You can use libraries like Open3D to convert your filtered numpy points into a `PointCloud` object and extract an oriented bounding box.
- If you prefer to use RGB-D data (color + depth map) instead of the raw point cloud array, you are free to do so, but you will need to modify the `ZedCamera` wrapper class yourself to retrieve the depth map.

Checkpoint 7: Pick and Place (Pure Vision) [1 pt]



(a) Initial State



(b) Target State

Goal

The goal of this checkpoint is to grasp a single cube using pure vision pose estimation and drop it into the paper cup located on the side of the arena.

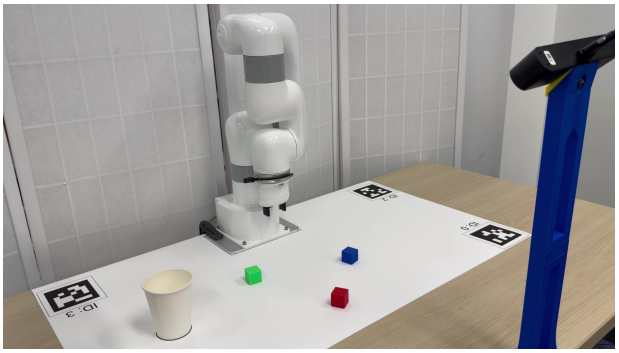
Description

In this checkpoint, the AprilTag is removed. There will be one cube in the arena and a paper cup serving as the drop-off location. You are provided with a template script (`checkpoint7.py`). To complete this task, fill in the `TODO` sections within the `main()` function to sequence the robot's actions.

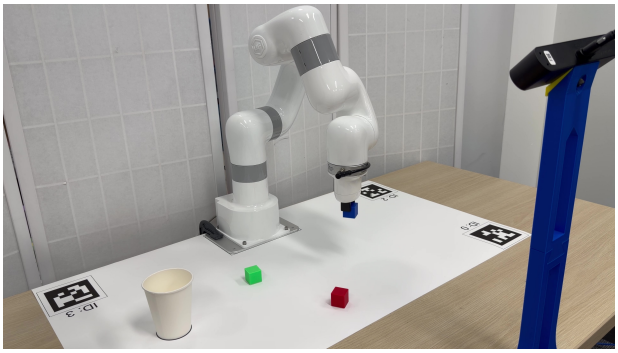
Hints

- Make sure all your imported functions are working reliably, as this checkpoint relies entirely on the accuracy of your pure vision perception pipeline from Checkpoint 6.

Checkpoint 8: Target Selection (Pure Vision) [2 pt]



(a) Initial State



(b) Target State

Goal

The goal of this checkpoint is to identify and grasp a specific cube based on a semantic text prompt (e.g., 'blue cube'), and safely put it back down.

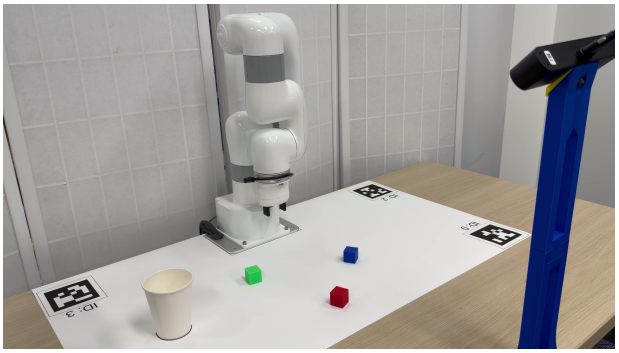
Description

In this checkpoint, the AprilTags are removed. There will be three cubes in the arena (red, blue, and green). You are provided with a template script (`checkpoint8.py`). To complete this task, you need to fill in the `TODO` sections within the `main()` function to sequence the robot's actions, and implement the `CubePoseDetector` class using raw camera data (image and point cloud).

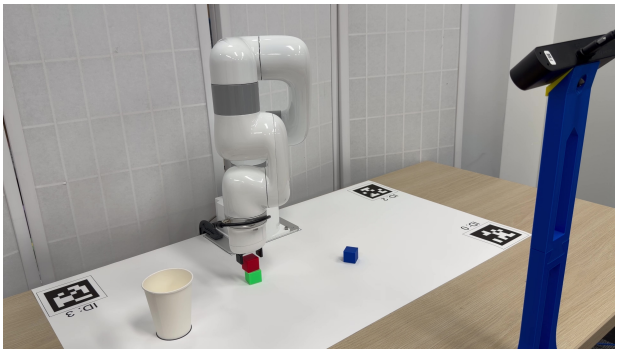
Hints

- This task requires you to merge the concepts from Checkpoint 3 and Checkpoint 6. You can use your color thresholding or segmentation method to find the target cube in the 2D image, and then use that mask to isolate the target's 3D points from the point cloud.
- Remember to filter out `NaN` values from the isolated point cloud.
- You can reuse your `grasp_cube()` and `place_cube()` functions from Checkpoint 1.

Checkpoint 9: Stacking (Pure Vision) [2 pt]



(a) Initial State



(b) Target State

Goal

The goal of this checkpoint is to pick up the red cube and successfully stack it directly on top of the green cube without using AprilTags.

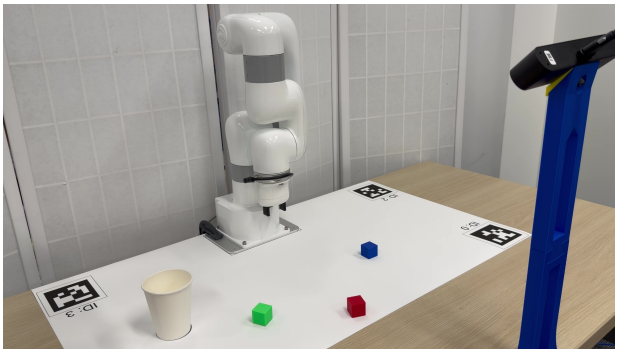
Description

In this checkpoint, the AprilTags are removed. There will be three cubes in the arena (red, blue, and green). You are provided with a template script (`checkpoint9.py`). To complete this task, you need to reuse your `CubePoseDetector` from Checkpoint 8 to locate both the red and green cubes. Then, use your previously determined `STACK_HEIGHT` and fill in the `TODO` sections within the `main()` function to sequence the robot's actions.

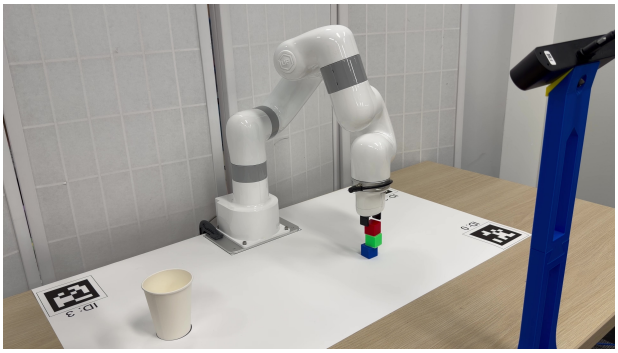
Hints

- Determine `STACK_HEIGHT` carefully. It should be large enough to clear the green cube, but small enough to ensure a stable drop without bouncing or tipping.
- You can reuse your `grasp_cube()` and `place_cube()` functions from Checkpoint 1.

Checkpoint 10: Sequential Stacking (Pure Vision) [2 pt]



(a) Initial State



(b) Target State

Goal

The goal of this checkpoint is to identify all three tagless cubes and stack them into a single tower according to a specific order using only pure vision.

Description

In this checkpoint, the red, green, and blue cubes will all be in the arena without AprilTags. You are provided with a template script (`checkpoint10.py`) that includes a `stacking_order` list defining the tower from top to bottom. To complete this task, fill in the `TODO` sections within the `main()` function to locate all three cubes via your `CubePoseDetector` and sequence the robot's actions to build the stack.

Hints

- You will heavily reuse your `CubePoseDetector`, `STACK_HEIGHT`, and pick-and-place functions from previous checkpoints.

5 Challenge

Challenge 1: The Standard Tower

Goal

Stack as many standard-sized cubes as possible into a single, stable vertical tower.

Description & Rules

In this challenge, your team will use the pure vision pipeline developed in Stage 2 to stack identical cubes. The environment will be completely devoid of AprilTags.

- You have a limited time window to execute your stacking sequence.
- The tower must stand completely unsupported for at least 3 seconds.
- The team that successfully stacks the highest number of cubes wins this challenge.

Challenge 2: The Irregular Skyscraper

Goal

Stack cubes of varied, unknown sizes into a single, stable vertical tower.

Description & Rules

This challenge introduces a new variable: the cubes are no longer a uniform size. You will be provided with cubes of three different sizes ranging from 15 mm to 35 mm.

- You have a limited time window to execute your stacking sequence.
- The tower must stand completely unsupported for at least 3 seconds.
- The team that achieves the highest tower (measured from the table to the top of the highest cube) wins this challenge.

Prizes: The winning teams for Challenge 1 and Challenge 2 will each receive an Amazon Gift Card! In the event of a tie, the team that completes their final stable tower in the shortest time wins.