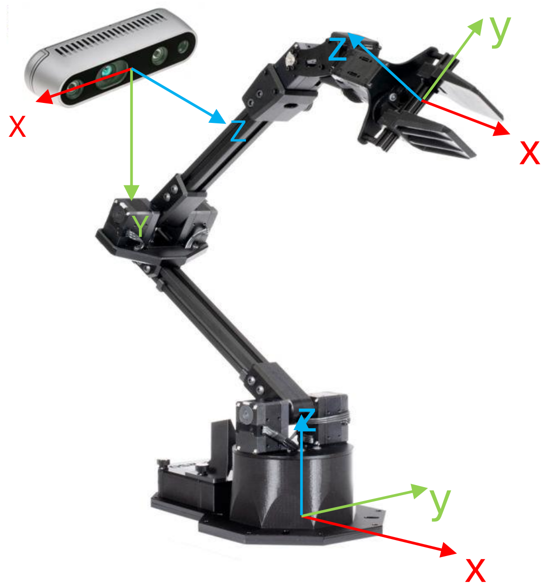
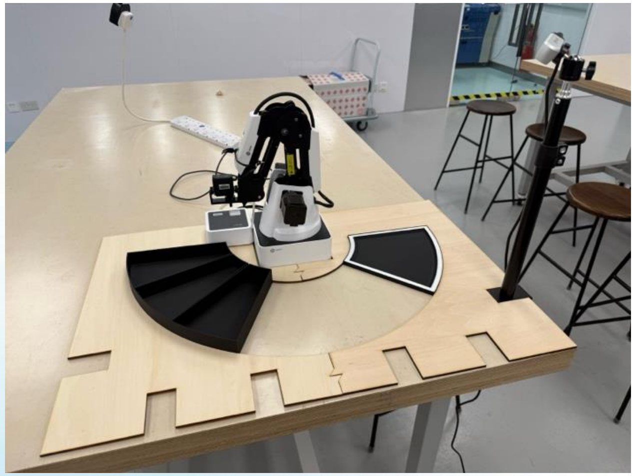
Tam Wing Fan Innovation Wing
FACULTY OF ENGINEERING

Introduction to Calibration





ArmStrong SIG

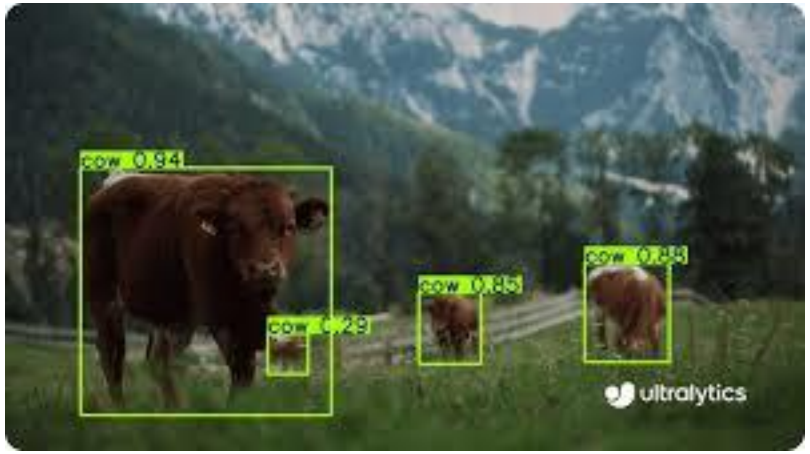
Recap of the previous workshop

Workshop 1: Robotics Arm Control

- We've learned about control the movement of the dobot magician with Python

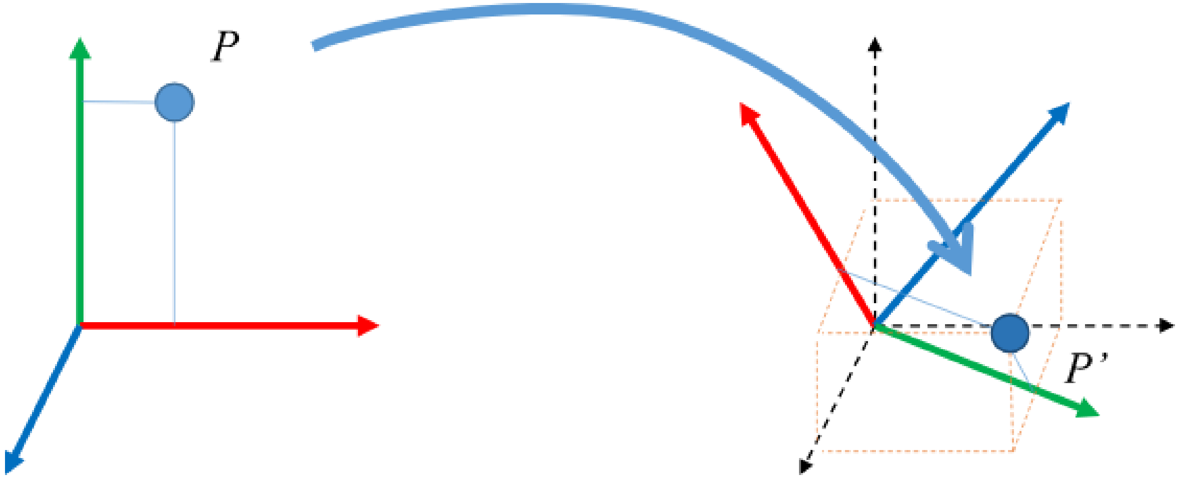
Workshop 2: Object Detection

- We've learned about training a YOLO object detection model



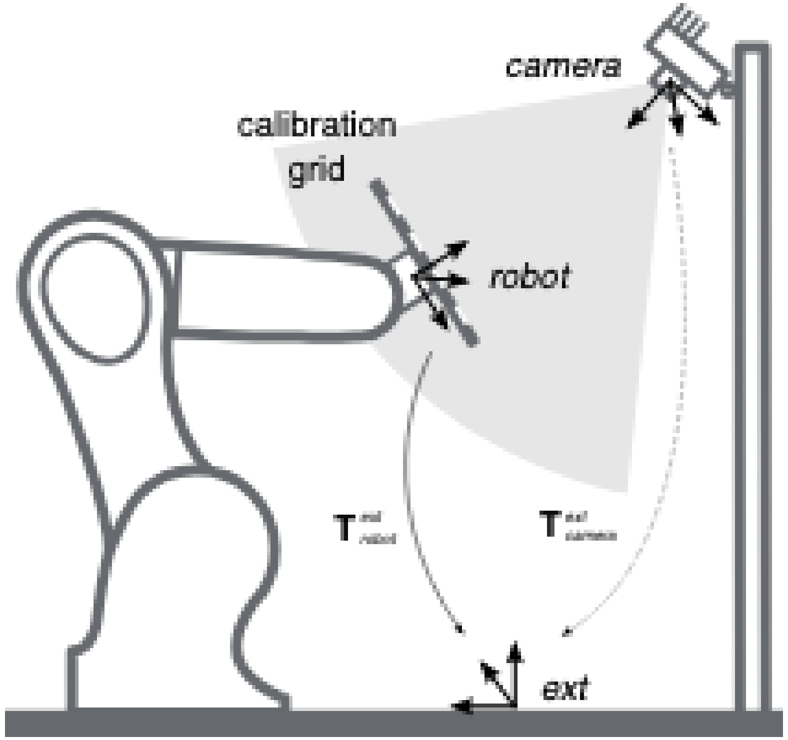
Objective of today workshop

- Understand usage of the transformation matrix
- Learn eye-to-hand calibration
- Be able to complete the click and go challenge



$$\boldsymbol{p} = (x, y, z)$$

$$\boldsymbol{p}' = R\boldsymbol{p} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$



Today's workshop

1. Introduction
2. Frame & Transformation Matrix
3. Numpy
4. AprilTag & Camera
5. Click and go challenge

Part 1
Introduction

How do I align the camera's view to the robotic arm's movement

Camera View

Output of camera:

- The screw at 2D pixel (200, 50)

Robotic Arm

Input of the arm:

- 3D Cartesian coordinate (200, 50, ??)

```
dobot.move_to(200, 50, )
```

What should be input on z?

Output of the YOLO object detection from RealSense D405

1. Introduction

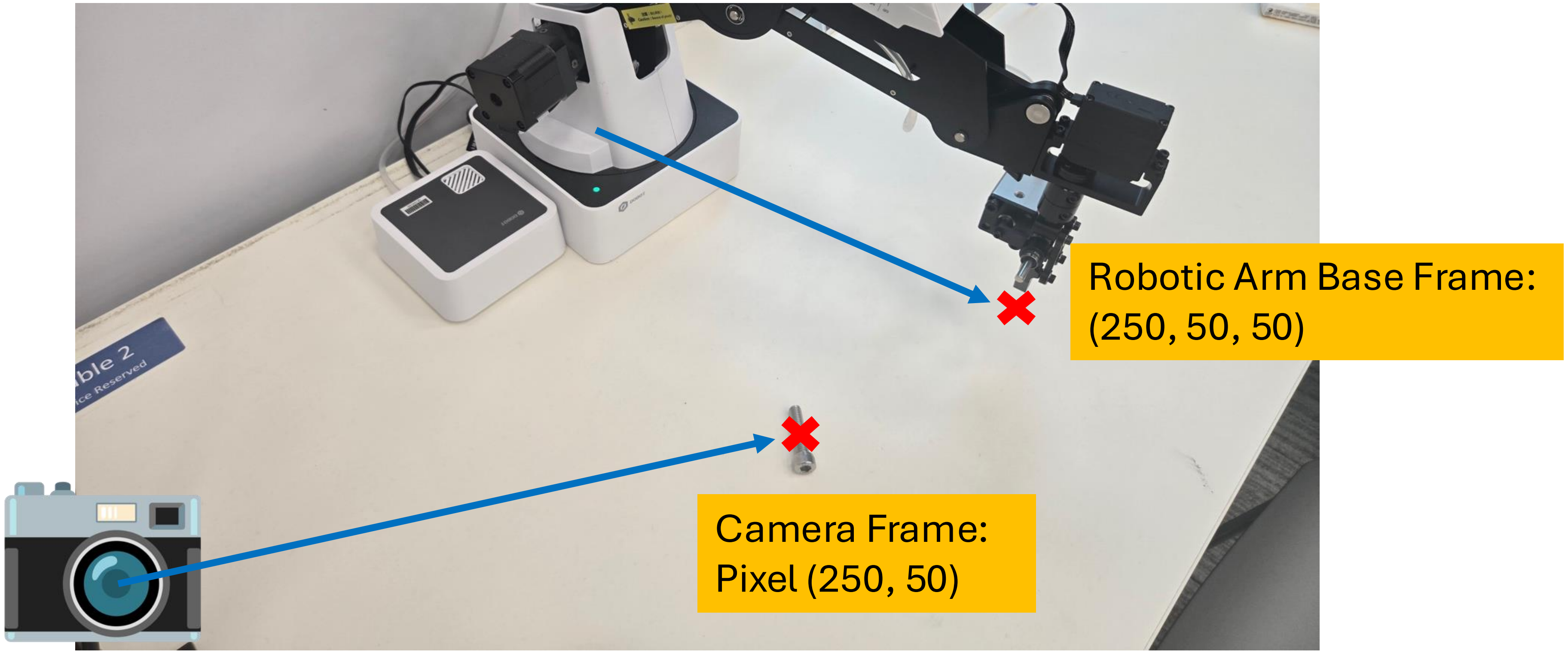
Robotic arm goes to wrong position

Why the robotic arm doesn’t reach the screw?

(If we feed the pixel coordinate from YOLO directly to the robotic arm)

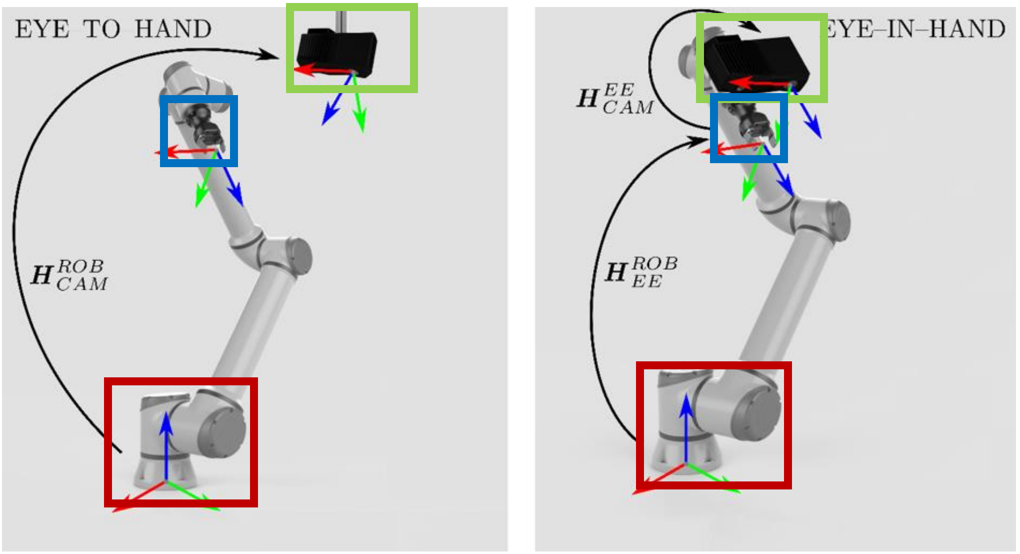
The point of view is different

- We call the point of view as “frame” in our calculation
- The coordinate (250, 50, 50) have different meaning in the camera and the robotic arm



Part 2:
Hand-eye Calibration

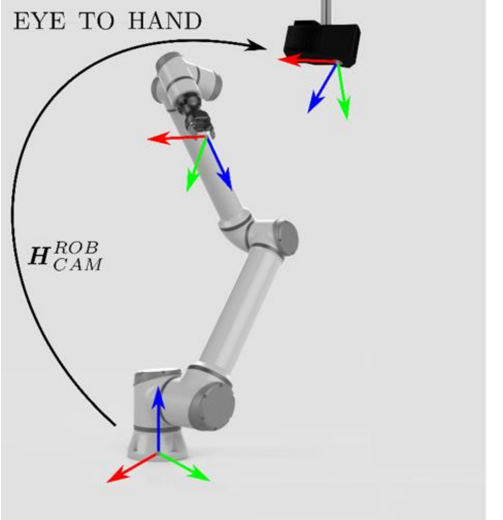
Hand-Eye Calibration



- Terms
  - The base of robotic arm: *base, robot*
  - The hand of robotic arm: *end-effector (ee), gripper, tool, hand*
  - The camera: *camera, eye, sensor*
  - The target object or the calibration board: *obj, target, cal*

- Eye-in-Hand: camera is attached on the hand
- Eye-to-Hand: camera is separate from the robot and stationary

Eye-to-Hand Calibration

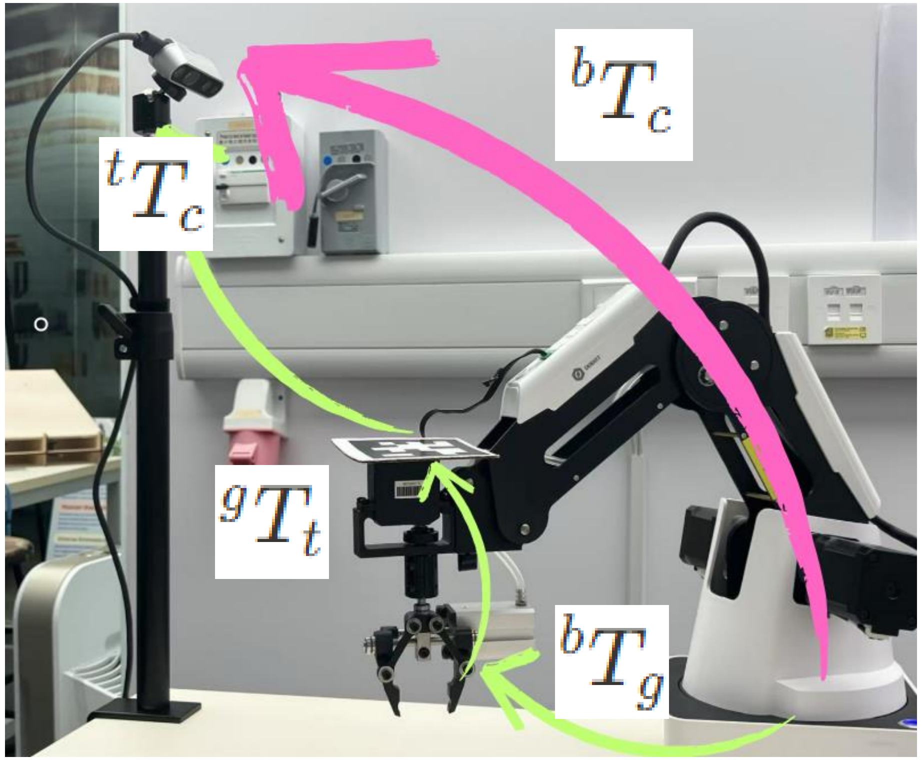


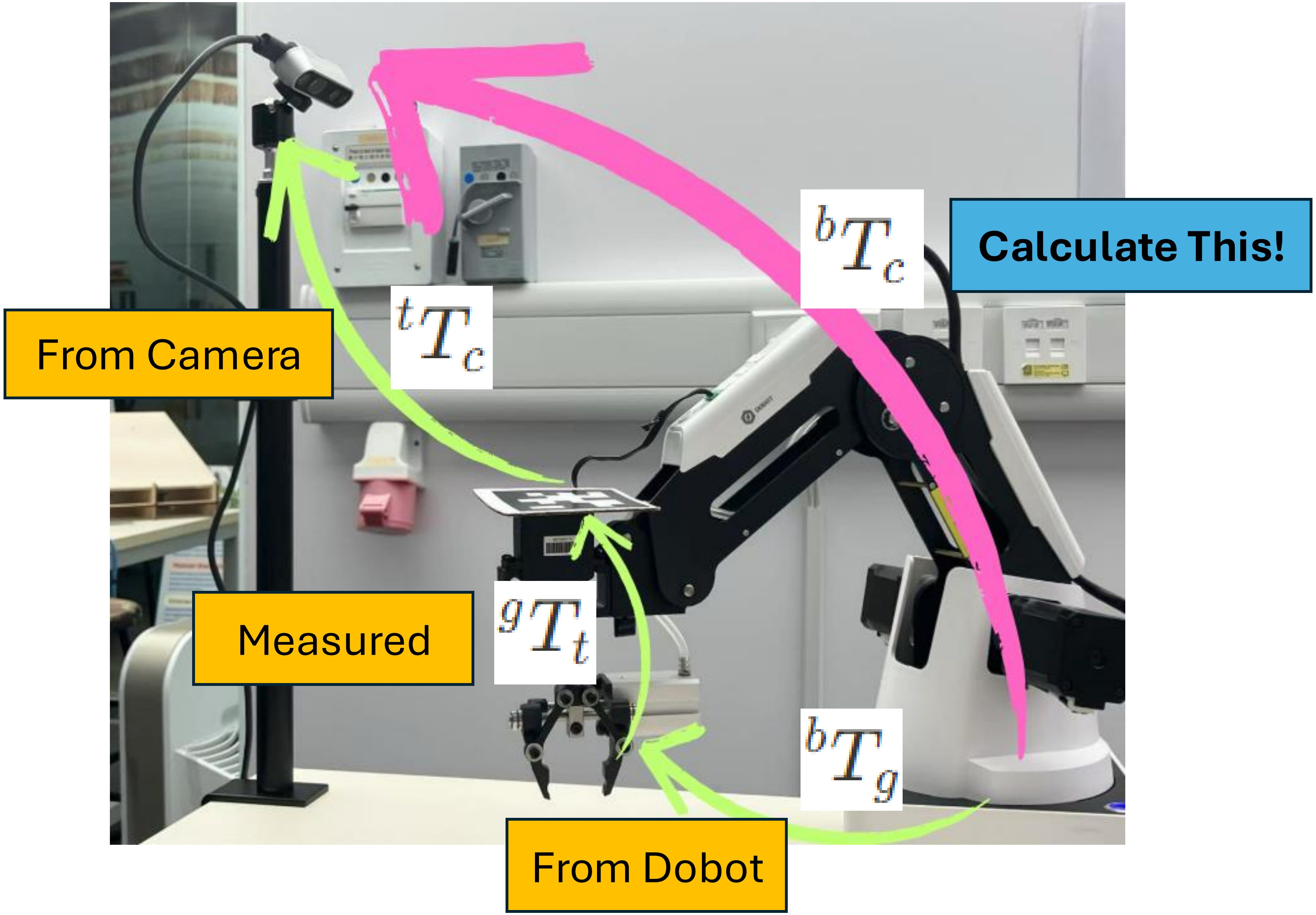
- The AprilTag will be attached on the gripper.
- There is an unknown transformation matrix from camera to the robot base T_c^b
- There is a constant transformation matrix from the calibration board / AprilTag to the gripper
  - In common practice, we don't care about this transformation matrix because it won't be involved in the calculation. But for simplification, we measure this matrix on Dobot and use a simpler method to calibrate.

$$
{}^{g}T_{t} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 140 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Eye-to-Hand Calibration

- We are to find out the transformation from camera to robot base so that we **can bring a point from camera frame to robot base frame**.
  - $T_c^b \cdot \overrightarrow{p_c} = \overrightarrow{p_a}$
- The transformation follows a loop:
  - $\boxed{\boldsymbol{T_c^b = T_g^b \cdot T_c^g \cdot T_c^t = T_g^b \cdot T_c^g \cdot T_t^{c^{-1}}}}$
- The camera obtains T_t^c and its inverse will be T_c^t





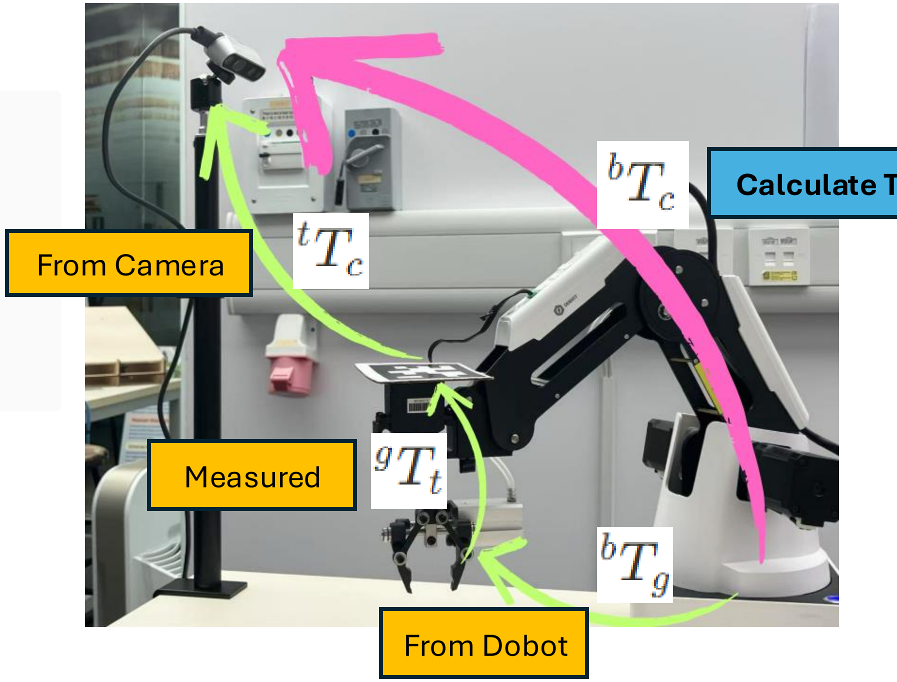
$^{b}T_{c}$
Calculate This!
$^{t}T_{c}$
From Camera
Measured
$^{g}T_{t}$
$^{b}T_{g}$
From Dobot

Today's Take-home Exercise

- Complete the base to camera transformation matrix
- Paste your implementation in calibration/utils.py
- We will explain it step by step in this workshop

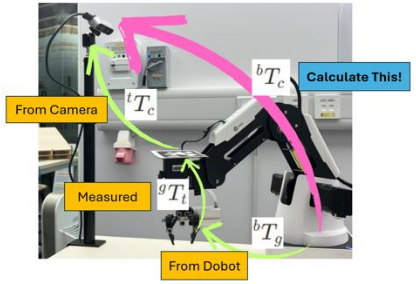
```python
def get_robot_base_to_camera(T_base_to_ee, T_ee_to_tag, T_tag_to_camera):
    # Paste your implementation here
    T_base_to_camera = np.eye(4)

    return T_base_to_camera
```



Calibration Workflow - High-level Overview (calibration_simplified.py)

1. **Data Collection:** Record transformations of T_t^c and T_g^b from various robot poses



2. **Calculation:** Get transformations of T_c^t using matrix inverse. Calculate the T_c^b transformation matrices.

3. Average: Derive the **rotation angles** and **translations** from the matrices and take the averages for them.

4. Save the final transformation in any form you like as a file for later use.

After calibration, we will get ${}^{b}T_{c}$ how do I know if it is correct?

How to validate ${}^{b}T_{c}$

Correct transformation matrix

- Convert the 3D point from camera frame to robotic arm base frames

Program to validate:

1. Get the 3D point from camera stream
2. Convert the 3D point to robotics arm base frames
3. Control the arm to the target 3D point

This program is called “Click and go”

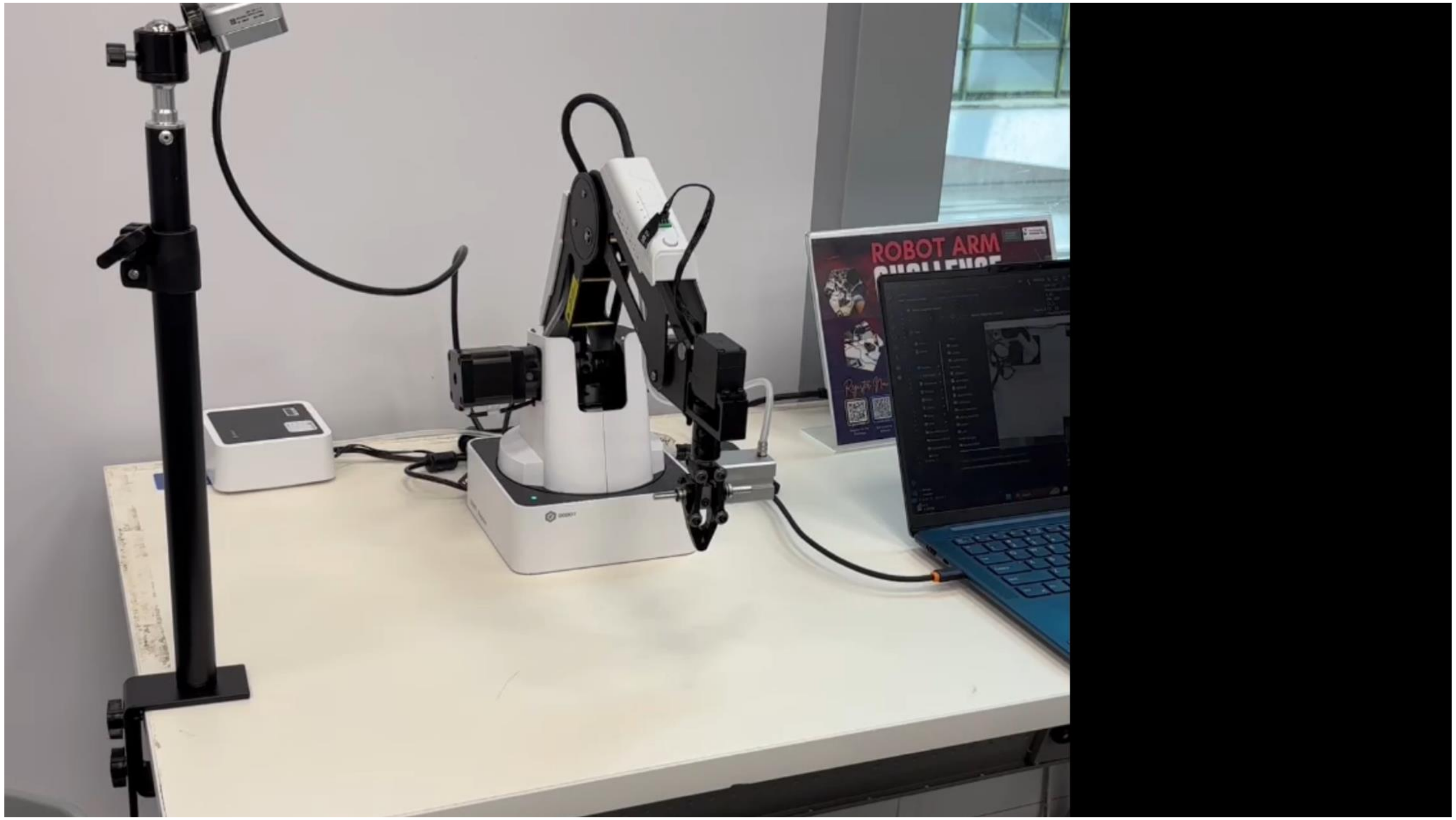
If the arm has reached the location that you clicked on screen

↓

Success

How to validate ${}^{b}T_{c}$

Click and go demonstration

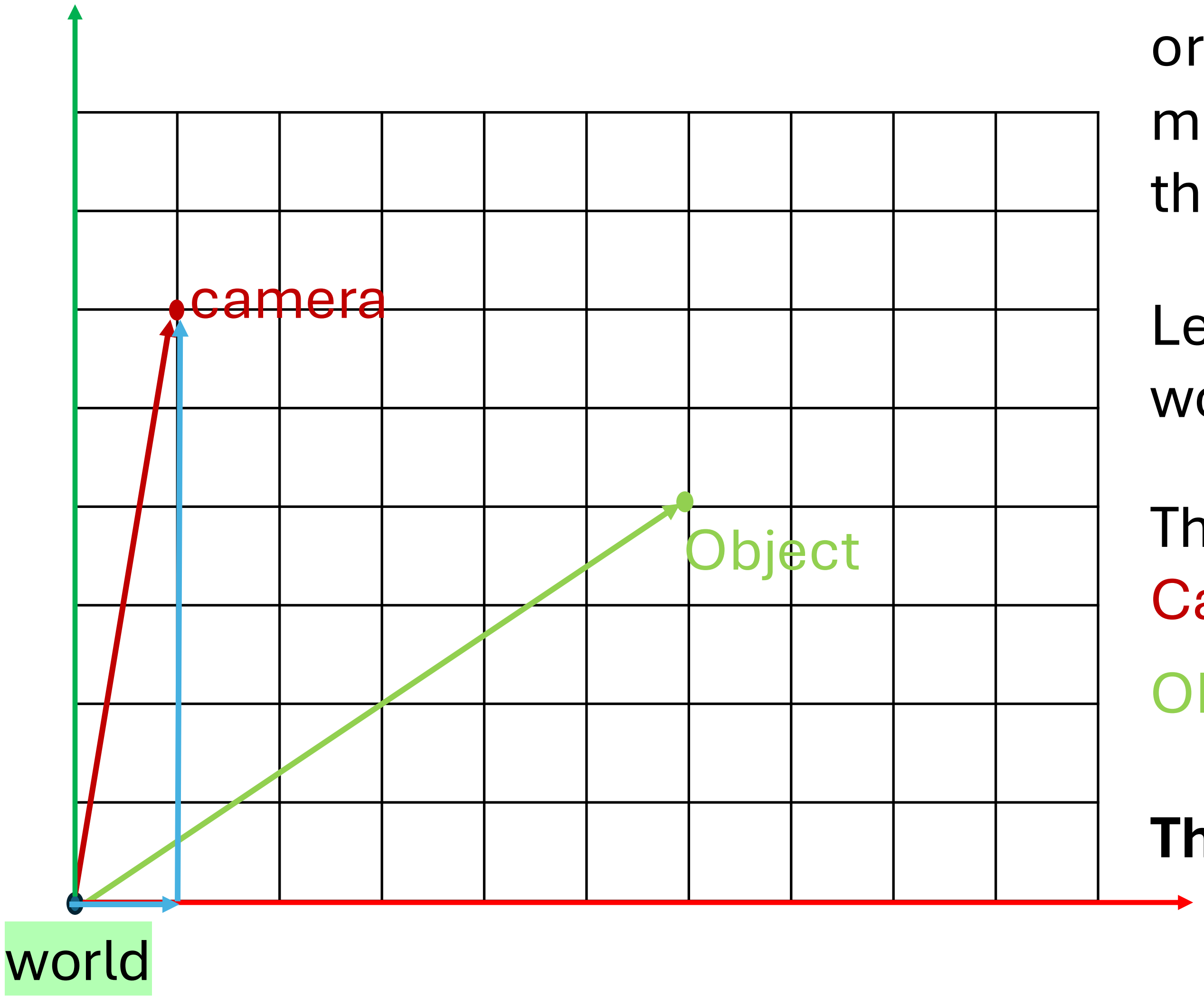


Part 3
Mathematics behind

Frame & Transformation Matrices

Frame

What is a Frame?



A frame is a 3D system consisting of an origin (position) and orientation (axes). Positions and motions are described relative to this coordinate system.

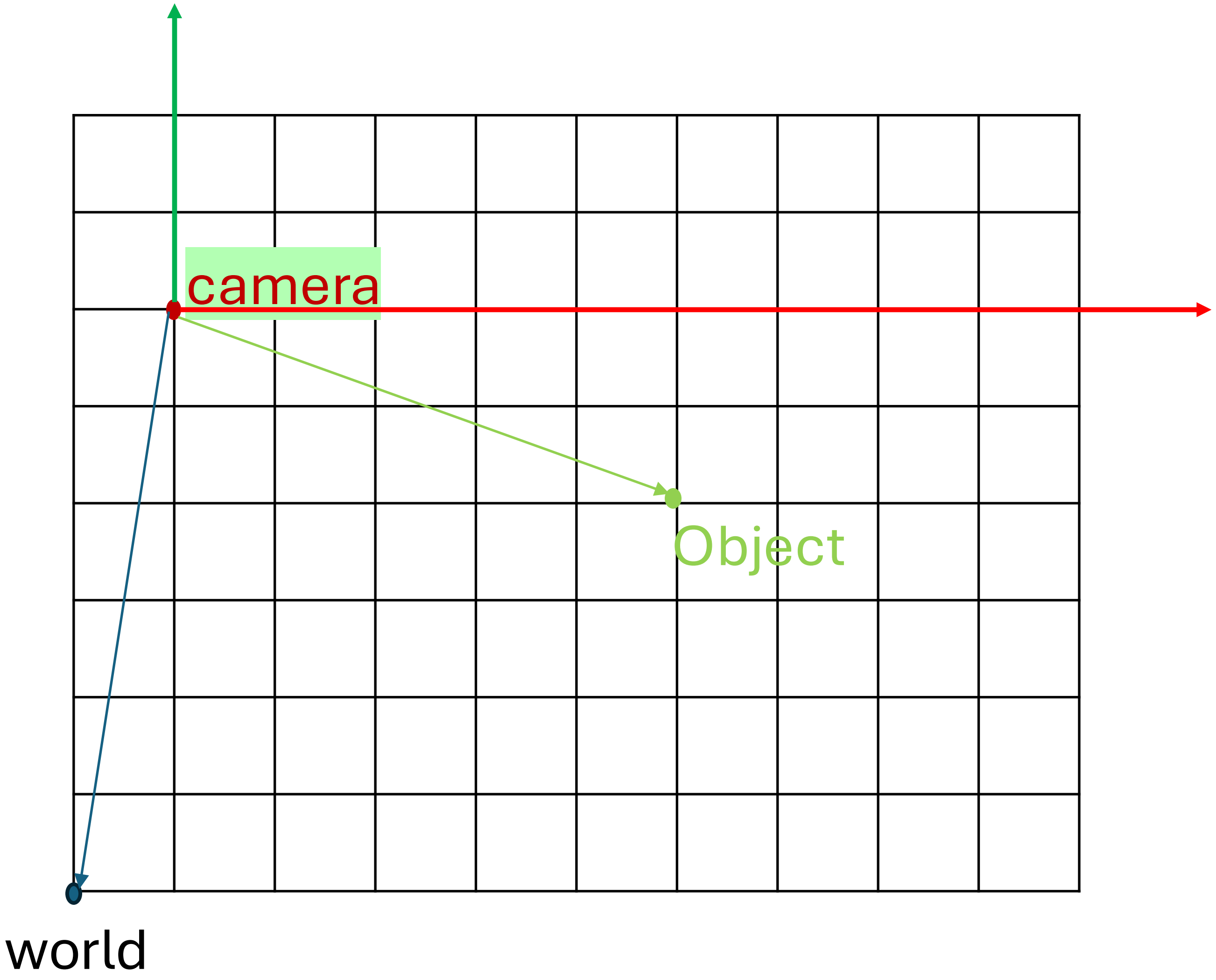
Let's define some points on the grid: world, camera, object

Then, with respect to the **world:**
Camera: (1,6) (1 units on x axis, 6 units on y)

Object: (6,4)

This is in the "world frame"

What is a Frame?



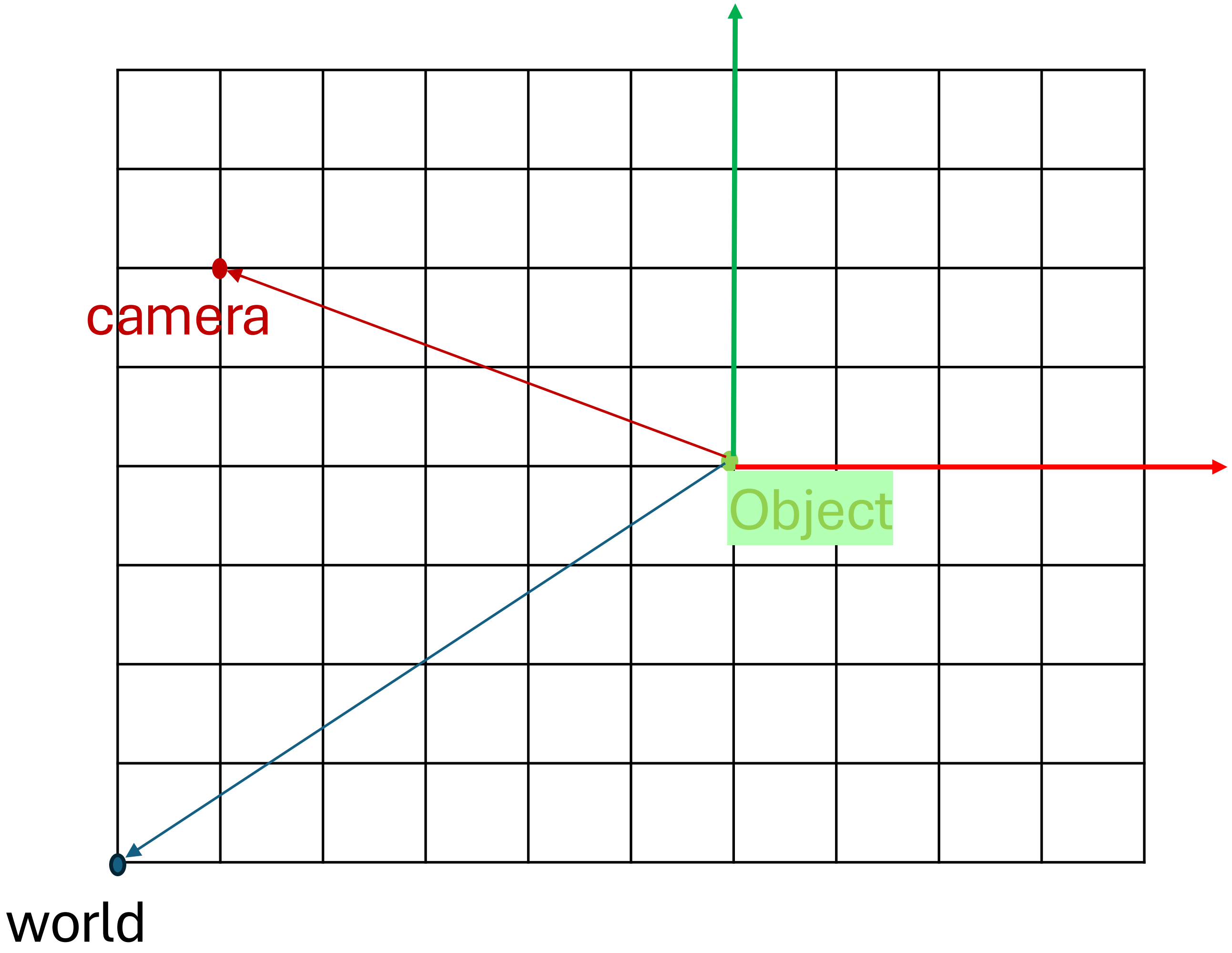
Similarly, with respect to the **camera**:

World: (-1,-6)

Object: (5,-2)

This is the "camera frame"

What is a Frame?



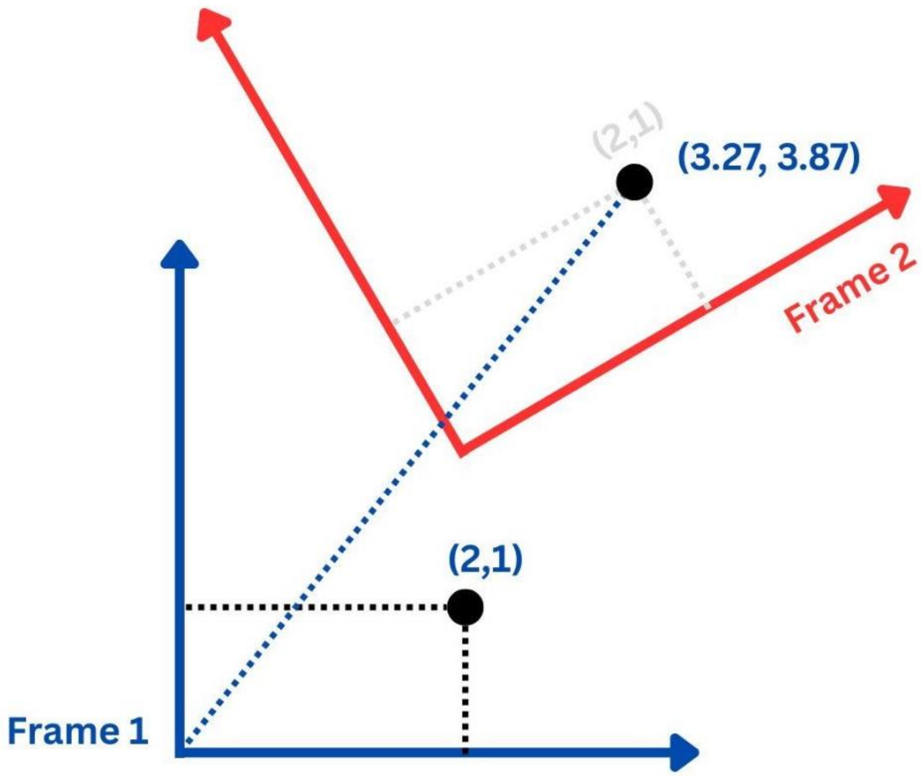
And with respect to the object:

World: (-6,-4)

Camera: (-5,2)

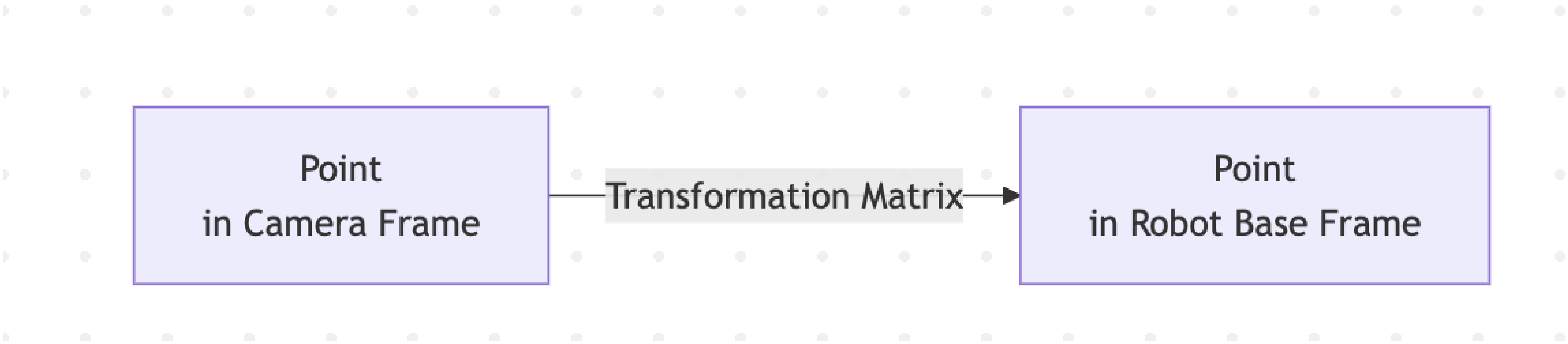
This is the "object frame"

Relating Frames via **Transformation Matrices**



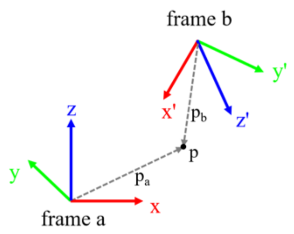
What is a transformation matrix?

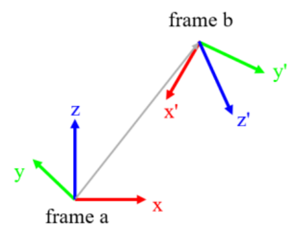
- A mathematical tool that converts coordinates between two fixed frames.



Notation:

$$H_b^a$$



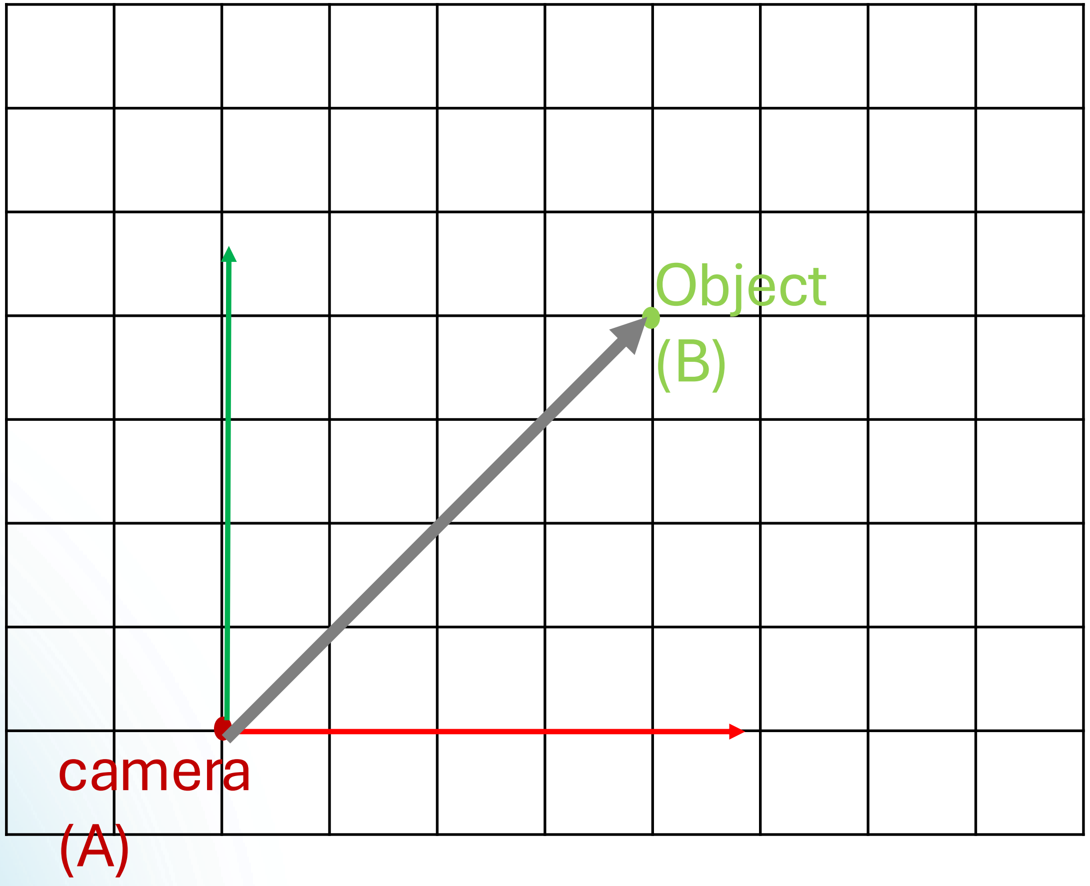


Interpretation	Meaning	Use case
Transform interpretation	$H_b^a \cdot \overrightarrow{p_b} = \overrightarrow{p_a}$	You have a point in frame **b**, and you want to express it in frame **a**
Pose interpretation	H_b^a is the pose of frame b in reference frame a	Frame **b** is located and oriented relative to frame **a**

Notation:

$$H_b^a$$

Interpretation	Meaning	Use case
Transform interpretation	$H_b^a \cdot \overrightarrow{p_b} = \overrightarrow{p_a}$	You have a point in frame **b**, and you want to express it in frame **a**
Pose interpretation	H_b^a is the pose of frame b in reference frame a	Frame **b** is located and oriented relative to frame **a**



*Q: A camera has **detected** an object and output its coordinates **relative to the camera frame**. Which matrix correctly express this relationship?*

H_B^A or H_A^B

A: H_B^A because we want to express the object's pose (frame B) in the camera's frame (frame A).

Learn more: https://support.zivid.com/en/latest/reference-articles/position-orientation-coordinate-transform.html

Transformation

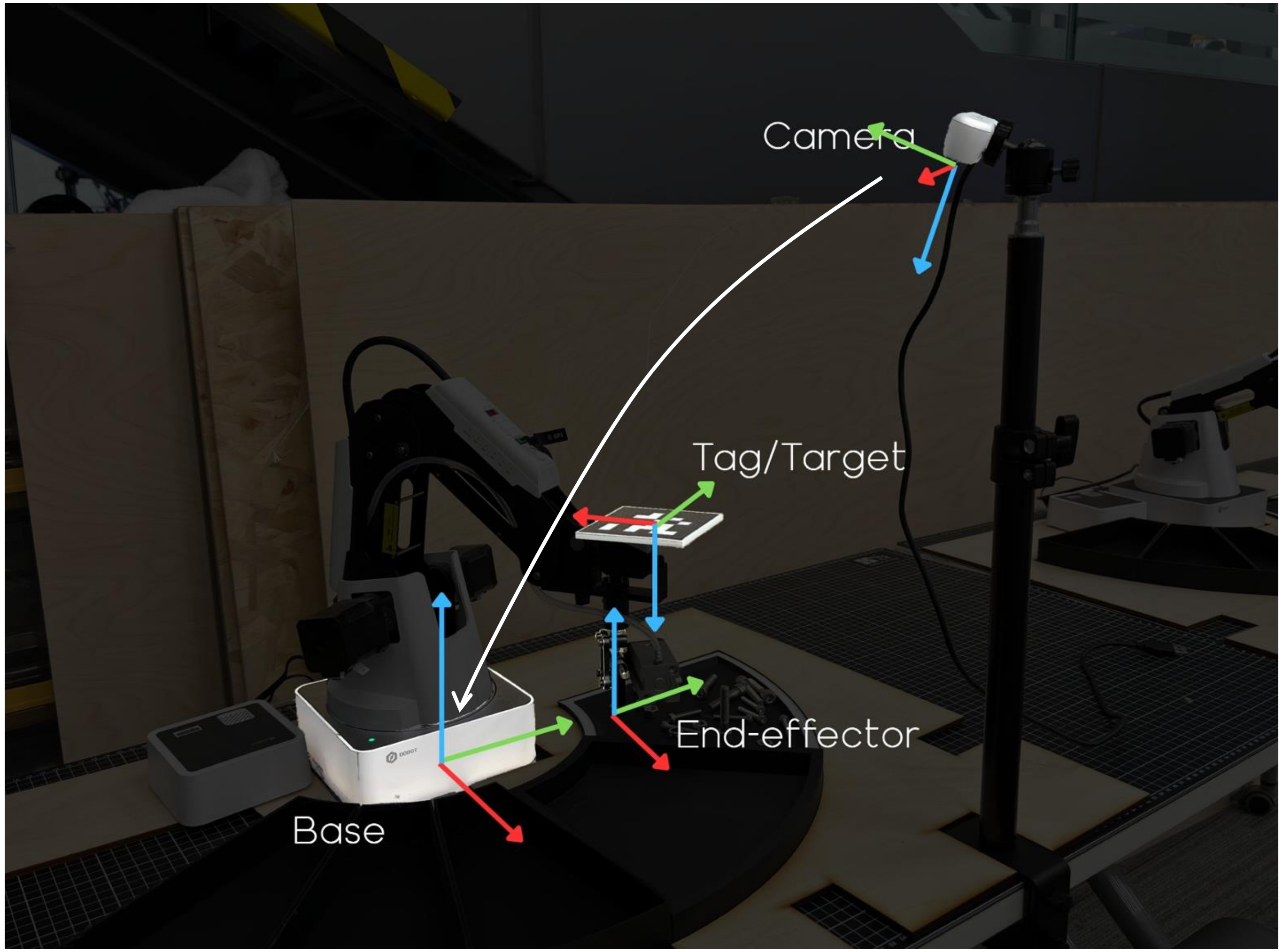
- A 3D rigid **transformation** is composed of a **rotation** and a **translation**.
- The rotation can be expressed by a 3*3 matrix, and the translation can be expressed by a 3-element vector.

$$\boldsymbol{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \qquad \boldsymbol{t} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

- The homogeneous transformation matrix is a 4*4 matrix in this form:

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x \\ r_{21} & r_{22} & r_{23} & y \\ r_{31} & r_{32} & r_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Goal of this workshop



Relate all the frames and retrieve the useful transformation matrix we want.

You have a point in **camera** frame, and you want to express it in **base** frame.

$$H_{camera}^{base}$$

Part 4
Python Library: Numpy

Useful python library with extended math feature

Useful function of numpy in this workshop

Create identity matrix

```
T_base = np.eye(3)
```

Input: size of the matrix

Output:

$$\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$$

Matrix Multiplication

$$A \times B$$

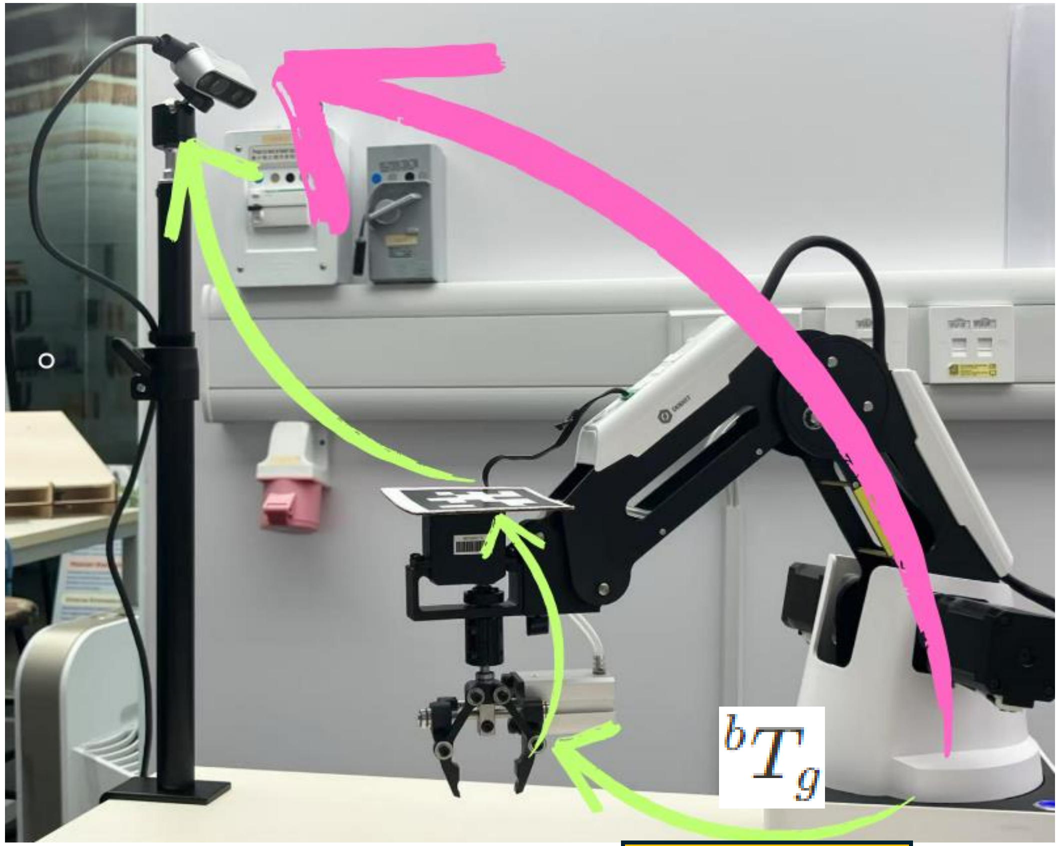
Method 1:

```
result = np.dot(A, B)
```

Method 2:

```
result = A @ B
```

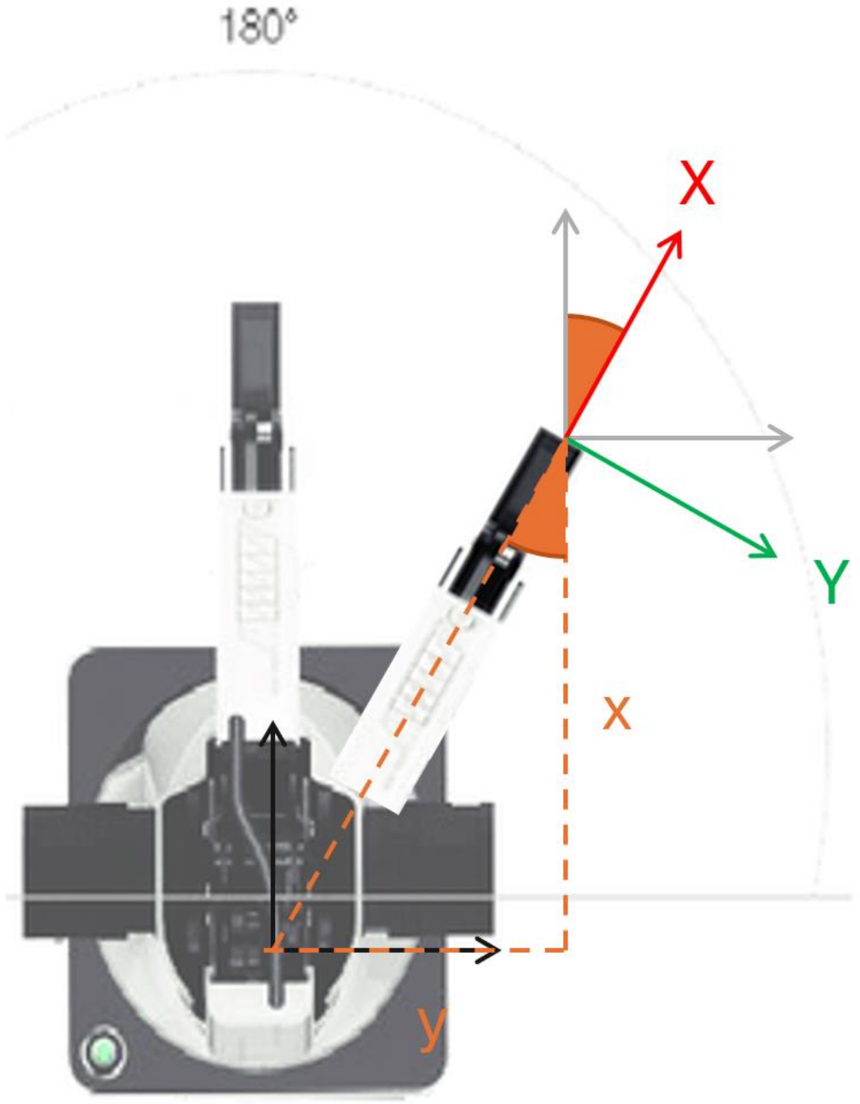
Practice 1



Build This

Build the transformation matrix of robotic arm base to robotics arm end-effector

Tips of Practice 1



What is this angle?

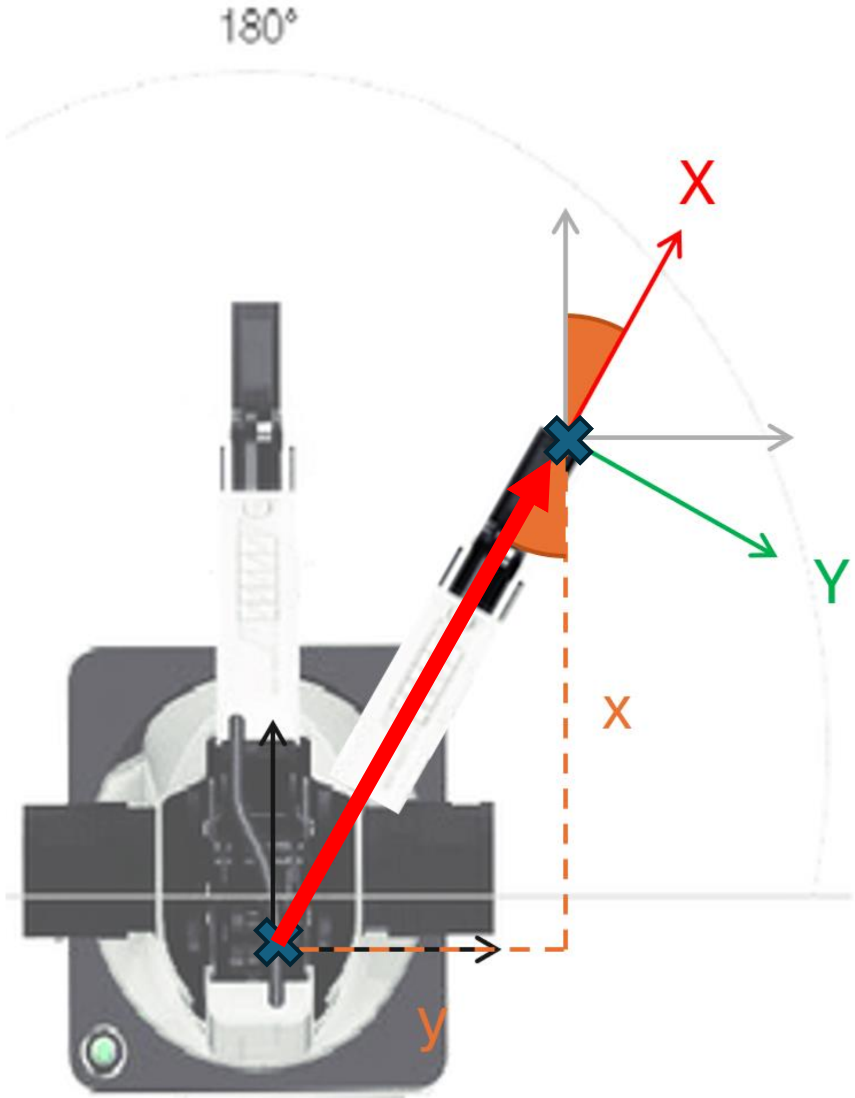
What does the rotation matrix look like?

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Note:

This robotic arm's gripper is always facing downwards, the only "freedom" to rotate is along z-axis

Tips of Practice 1



What is this translation?

$$\boldsymbol{t} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

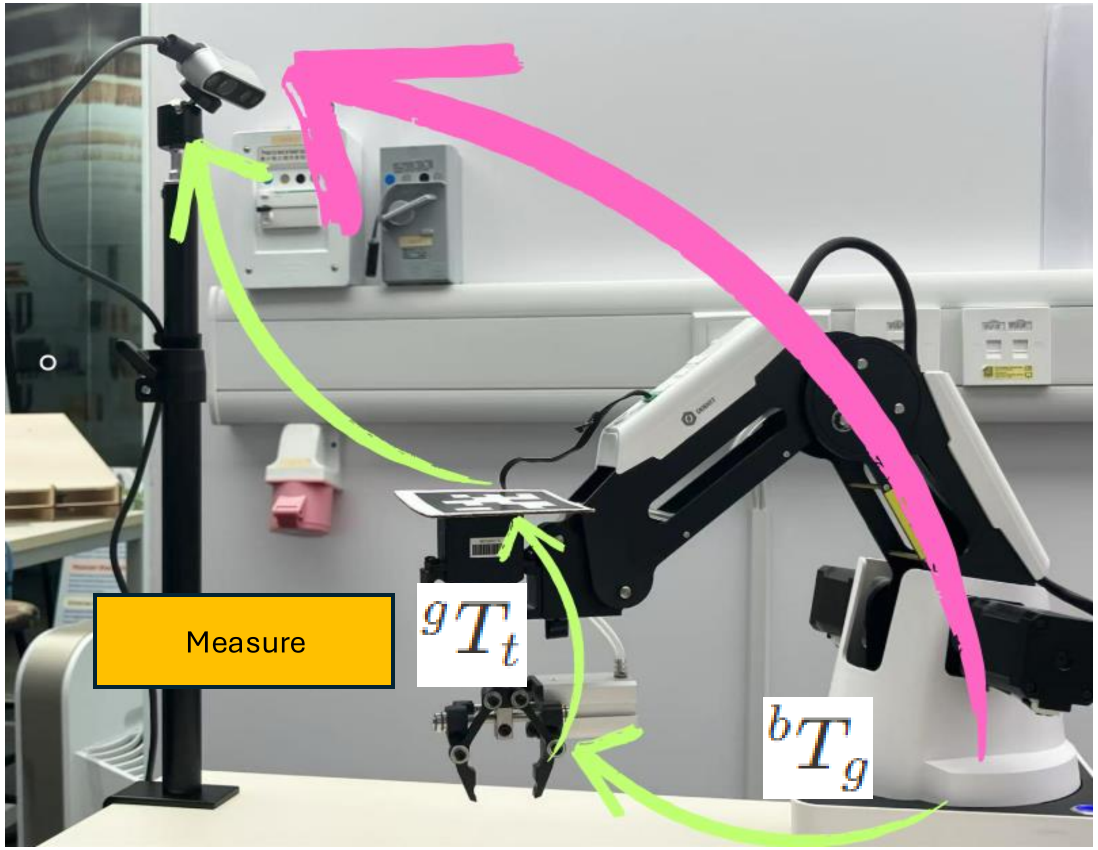
```
pose = dobot.get_pose()
print(pose.position.x, pose.position.y,
pose.position.z, pose.position.r)
```

Recall from Workshop 1: Robotic Arm Control

Save your answer to calibration/utils.py

```python
def get_robot_base_to_ee(pose):
    # Paste your implementation here

    return robot_matrix
```



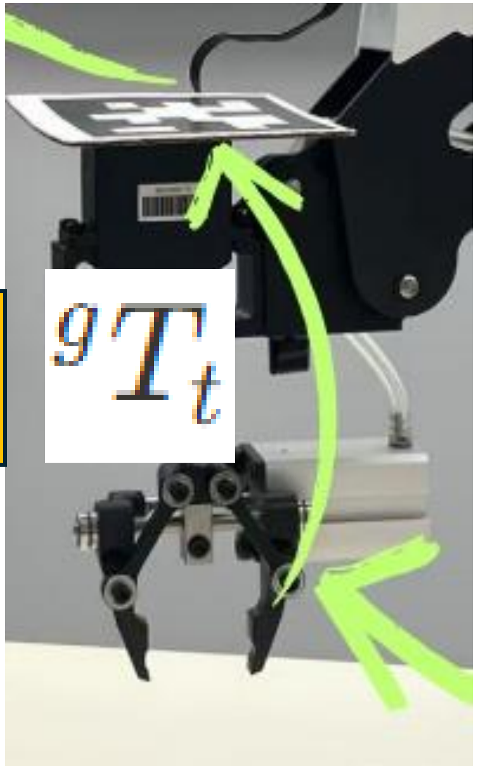
Practice 2

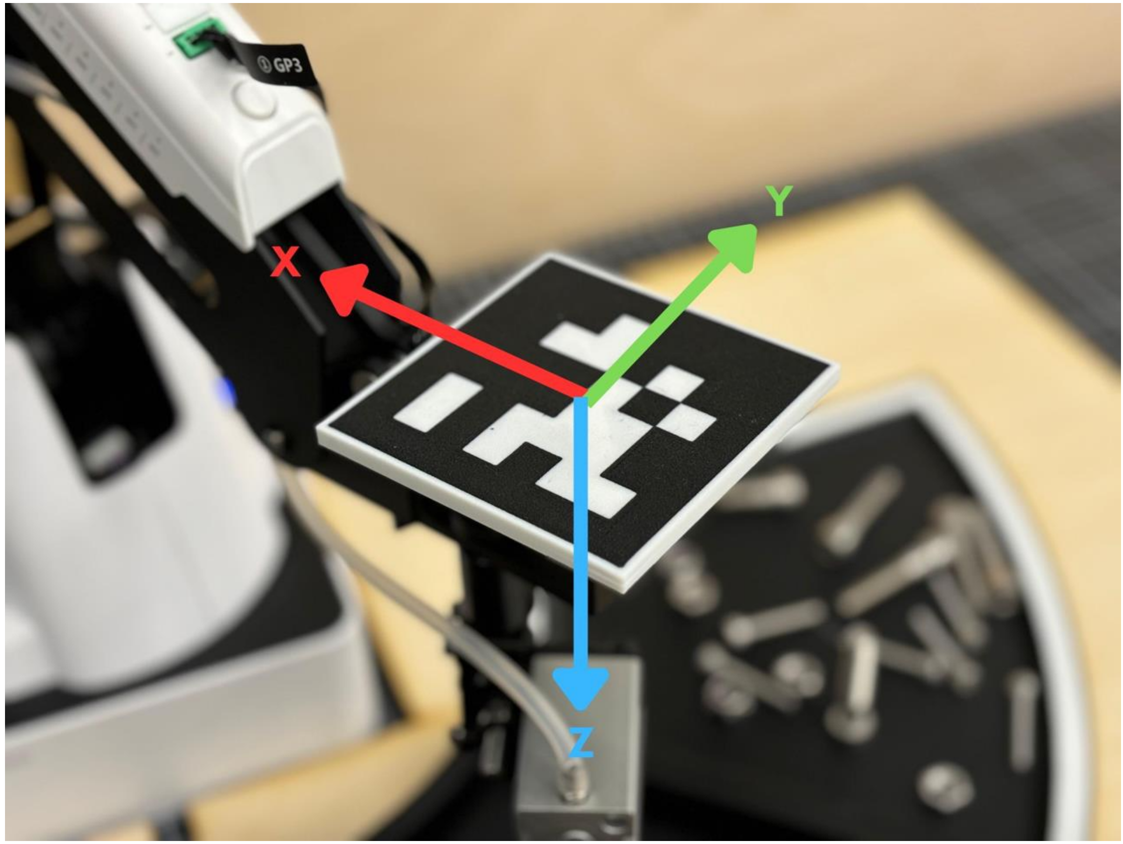
Measure and build the transformation matrix of AprilTag to End-effector (Gripper)

Practice 2

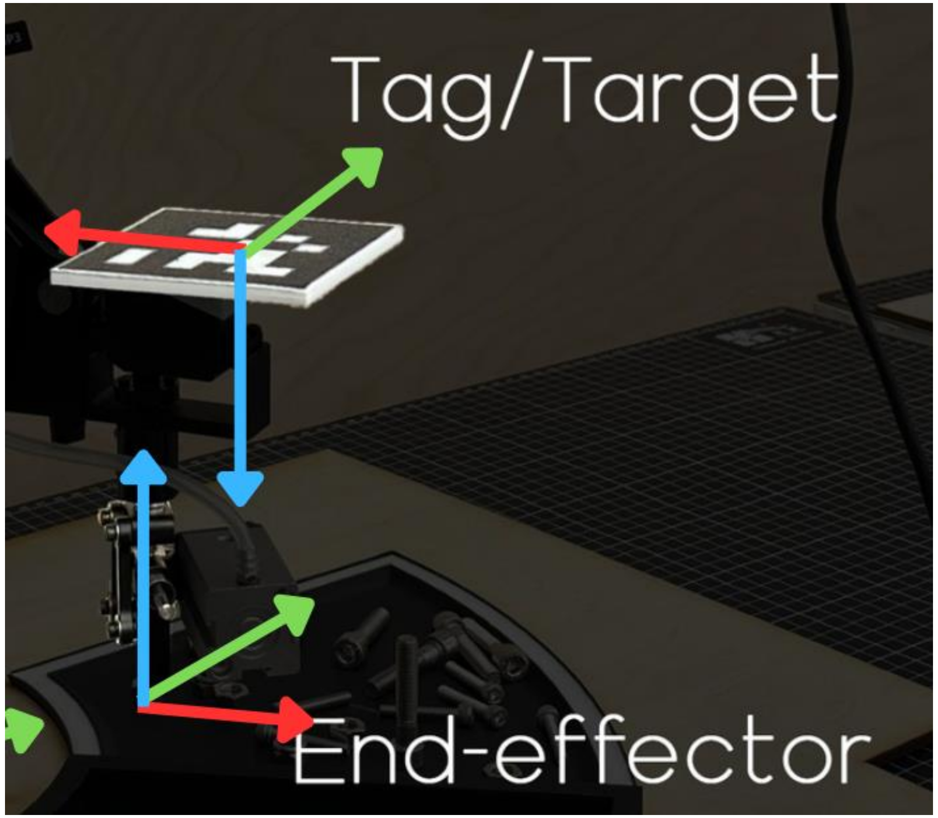
Measure and build the transformation matrix of End-effector (Gripper) to AprilTag

Measure





Tips of Practice 2



In terms of direction,

the **x-axis** and the **z-axis** are **flipped** while the **y-axis** **remains unchanged**, how can we represent this in a **rotation** matrix?

Think about the rotation matrix we did in practice 1.

Translation vector: $\begin{matrix} 30 \\ 0 \\ 153 \end{matrix}$

Save your answer to calibration/utils.py

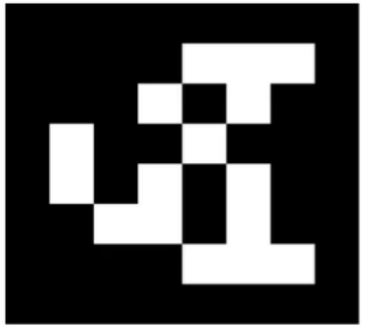
```python
def get_gripper_to_tag():
    # Paste to your measurement here

    return gripper_to_tag_matrix
```

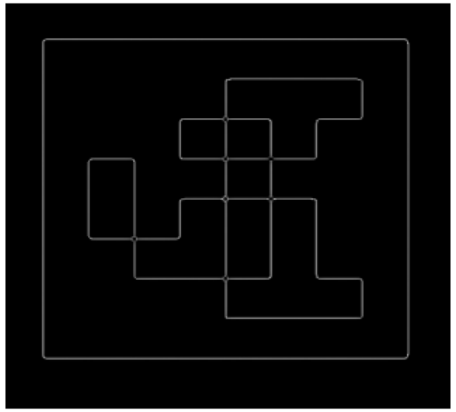
Part 5:
Camera & AprilTag

What is AprilTag?

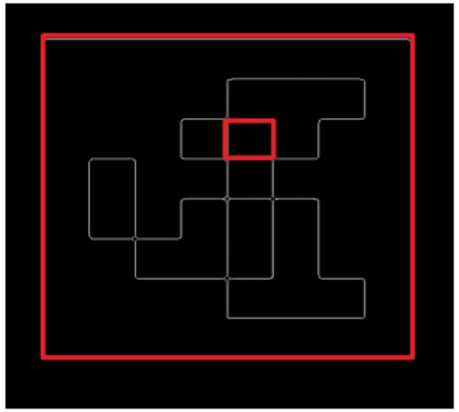
- AprilTag is used to recognize the precise **position and orientation**, relative to the camera



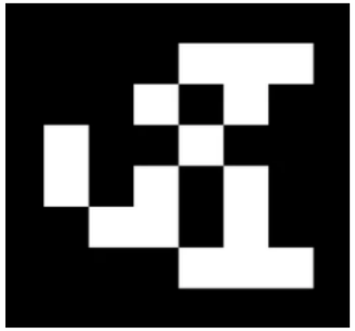
Input image: AprilTag (Class 36H10)



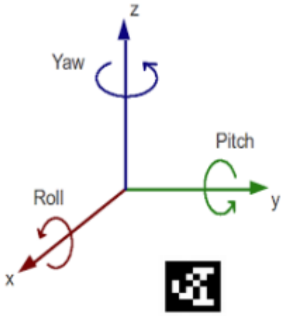
Step 1: Detection of line segments using the least square method on clusters of similar pixel gradients.



Step 2: Based upon the gradient direction, all possible quads are detected in an image.



Step 3: A quad with a valid code scheme is extracted to detect the pose.



Step 4: A pose of AprilTag in camera frame of reference is returned using homograph and intrinsic estimation.

Visit our notion page to learn more

Use AprilTag in Python

Create the AprilTag Detector

```
from pupil_apriltags import Detector
detector = Detector()
```

Recall Object-oriented Programming
From Workshop 1: Robotics Arm Control

Detect AprilTag and estimate pose

```
tags = detector.detect(
    gray_image,
    estimate_tag_pose=True,
    camera_params=camera_params,
    tag_size=tag_size
)

print(f"Tag ID: {tag.tag_id}")
print("Translation (x, y, z) in meters:", tag.pose_t)
print("Rotation matrix (3x3):\n", tag.pose_R)
```

RealSense Camera

We prepared some code snippets to simplify the camera control

Initialize the camera (Call this function once at the beginning)

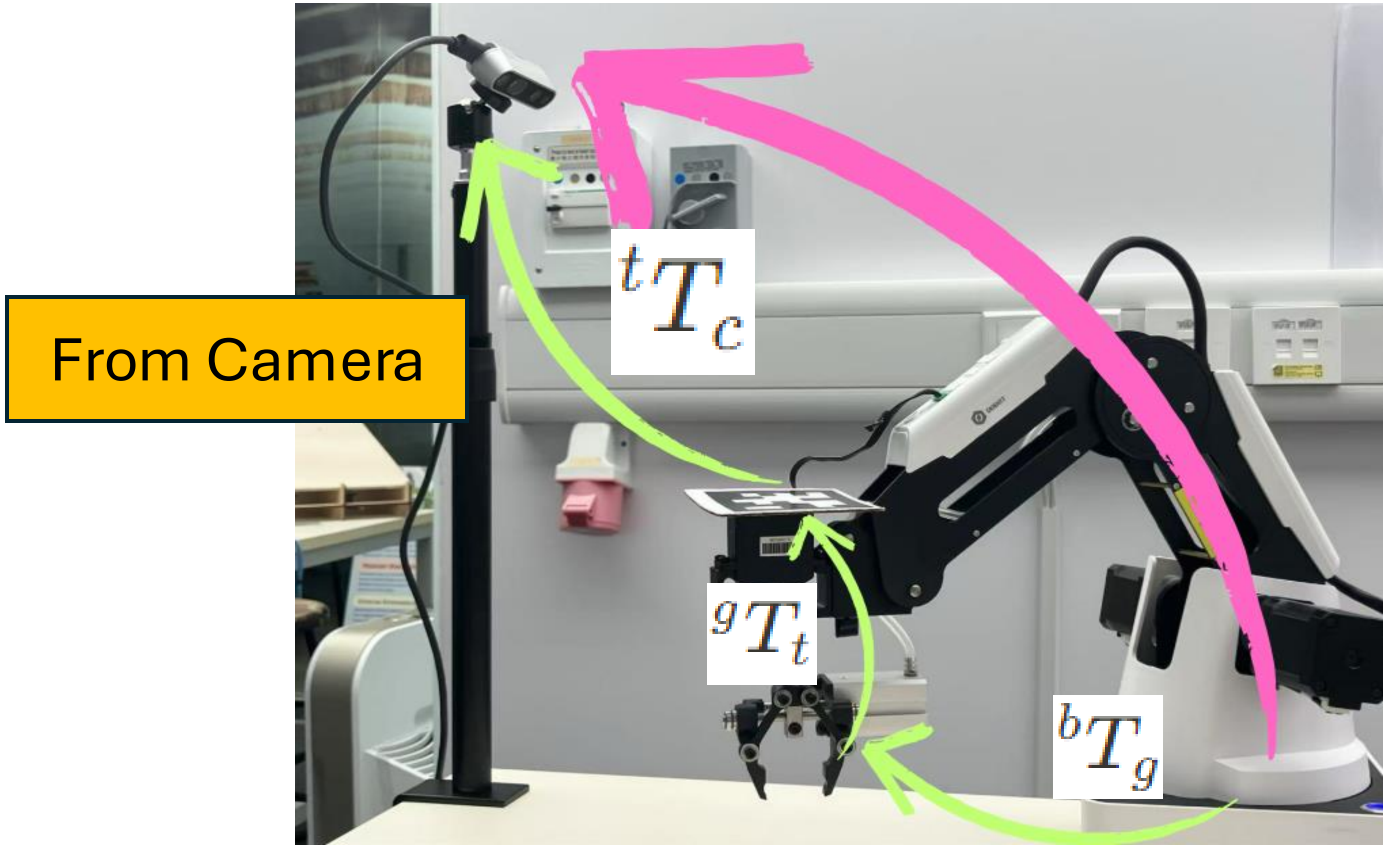
```
pipeline, profile, align = initialize_pipeline()
```

Get current **camera output**
(Call this function in the beginning of the loop)

```
color_image, depth_image, color_frame,
depth_frame = process_frames(pipeline, align)
```

Get Camera **intrinsics**

```
fx, fy, ppx, ppy = get_camera_intrinsics(profile)
```



Practice 3

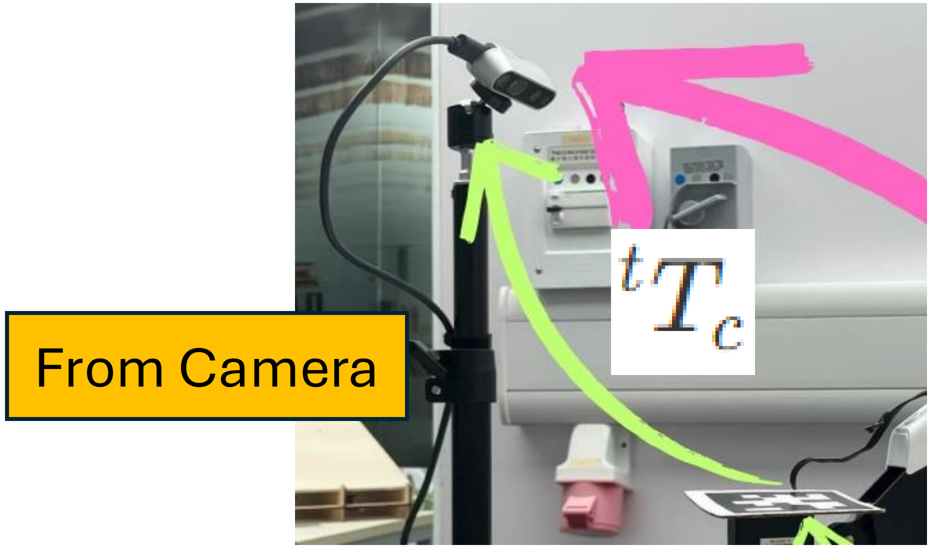
Get the transformation matrix from camera to AprilTag

Practice 3

Get the transformation matrix from camera to AprilTag

Create the AprilTag Detector

```
from pupil_apriltags import Detector
detector = Detector()
```



Detect AprilTag and estimate pose

```
tags = detector.detect(
    gray_image,
    estimate_tag_pose=True,
    camera_params=camera_params,
    tag_size=tag_size
)

print(f"Tag ID: {tag.tag_id}")
print("Translation (x, y, z) in meters:", tag.pose_t)
print("Rotation matrix (3x3):\n", tag.pose_R)
```

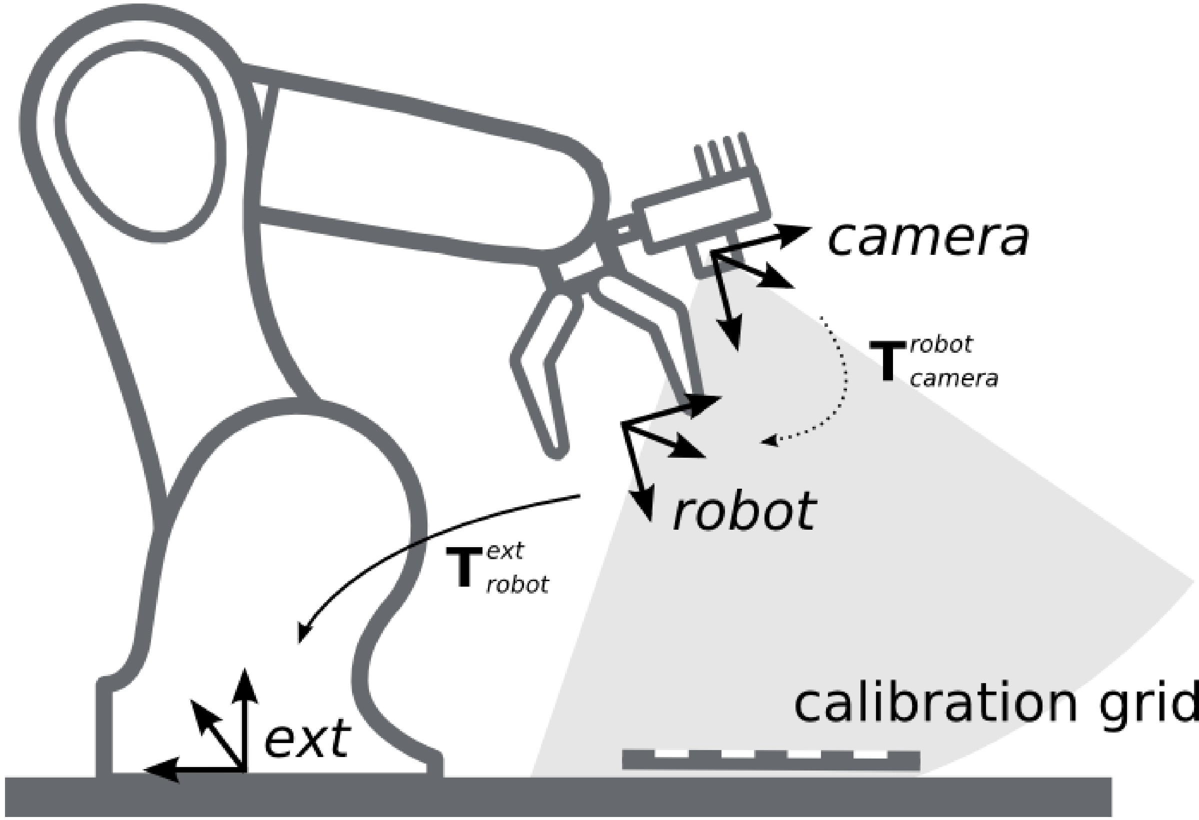
Reminder: The translation and rotation is camera to tag

Learn more: Eye-in-hand Calibration

Camera mounts on the robotic arm

Advantage:

- Close-up image of the workspace
- Lower depth error (Best accuracy of RealSense D405 at 7cm)



Learn more: Eye-in-hand Calibration

Dataset Preparation Procedure

Advanced - Image Augmentation

3 Hand-eye Calibration

Hand-Eye Calibration

Advanced - Eye-in-hand calibration

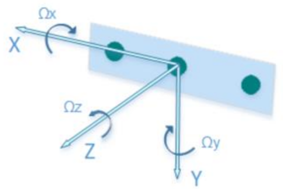
4 Reference

Go to our notion page to learn more

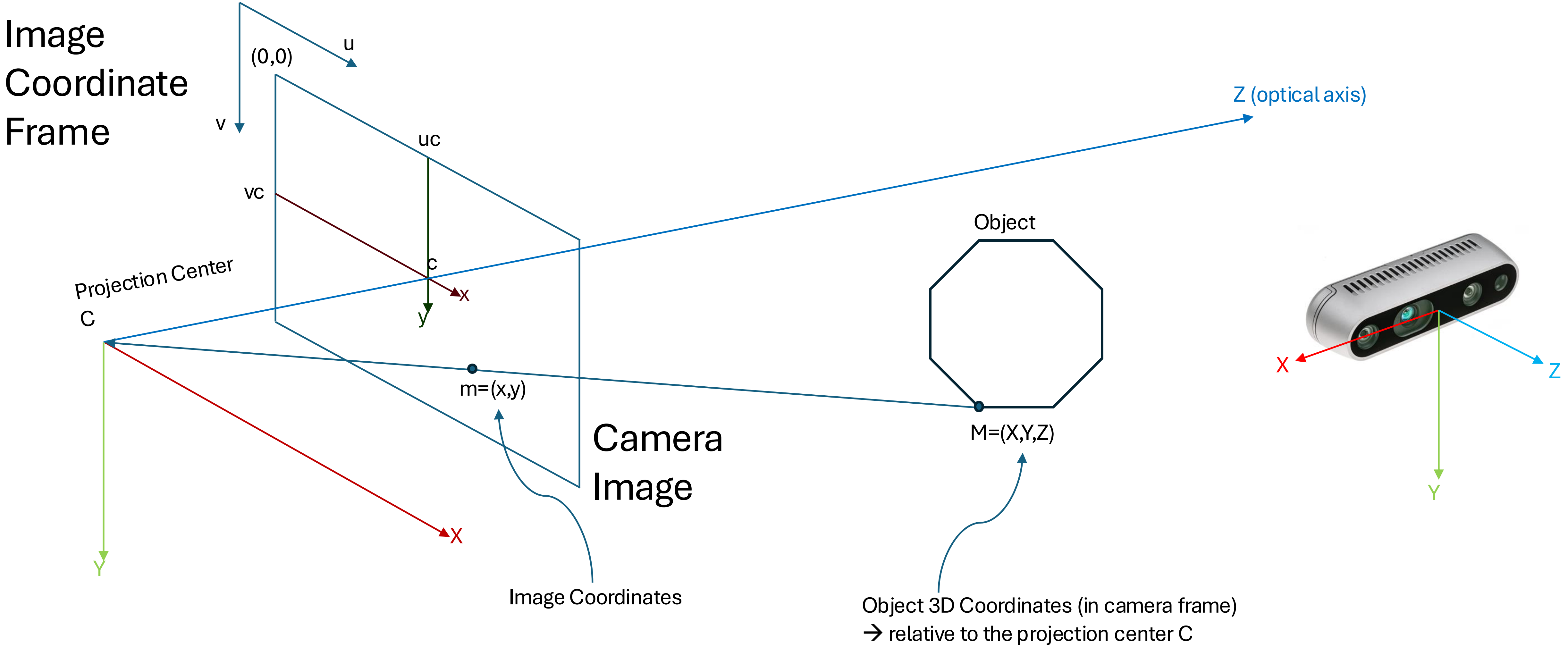
2. Hand-eye Calibration

Camera Model

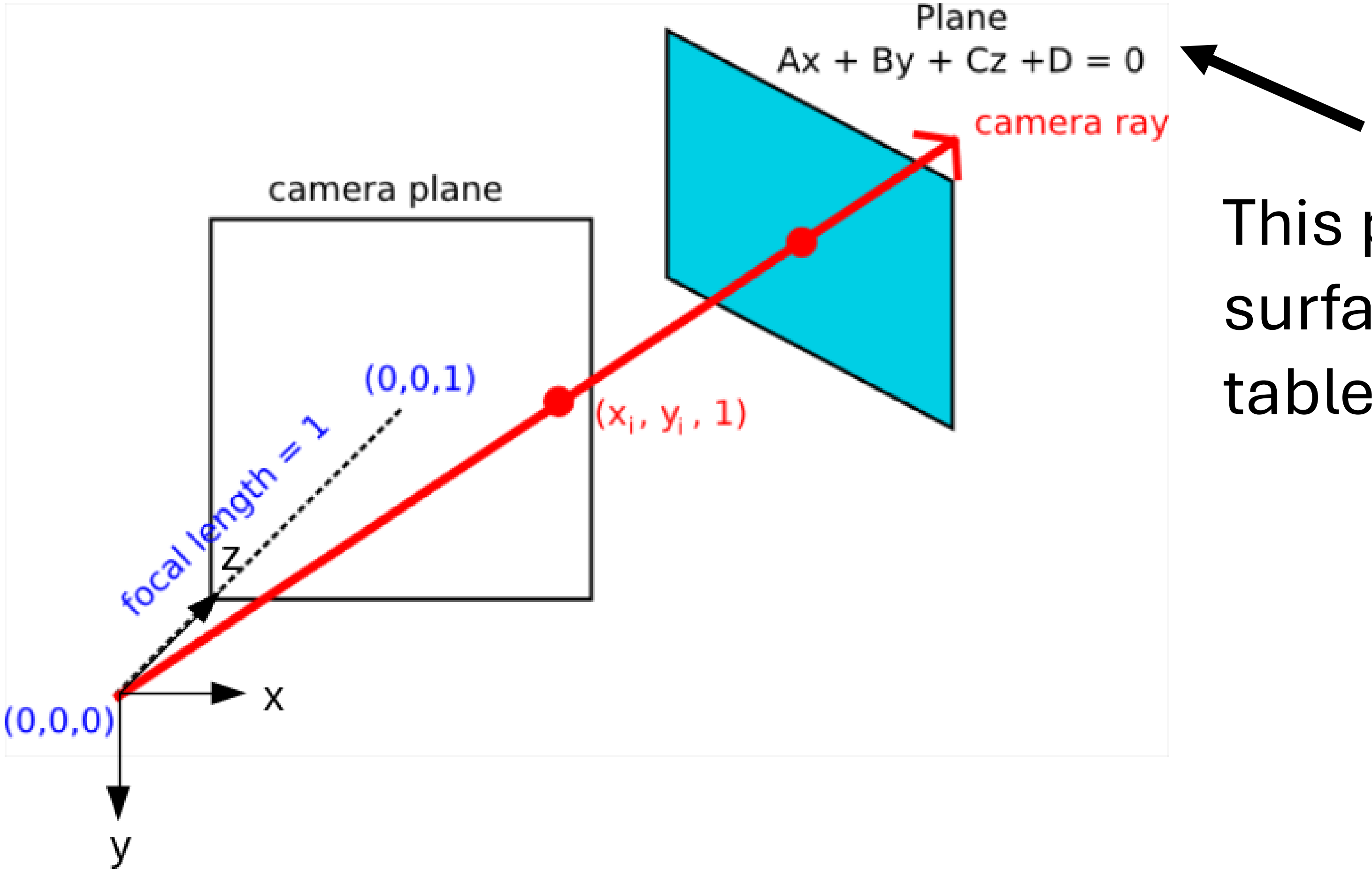
The resulted orientation angles and acceleration vectors share the coordinate system with the depth sensor.



1. The positive x-axis points to the right.
2. The positive y-axis points down.
3. The positive z-axis points forward



Learn more: Plane projection



This plane can be the surface of the working table