Tam Wing Fan Innovation Wing
FACULTY OF ENGINEERING

Innovation Academy
Faculty of Engineering, HKU

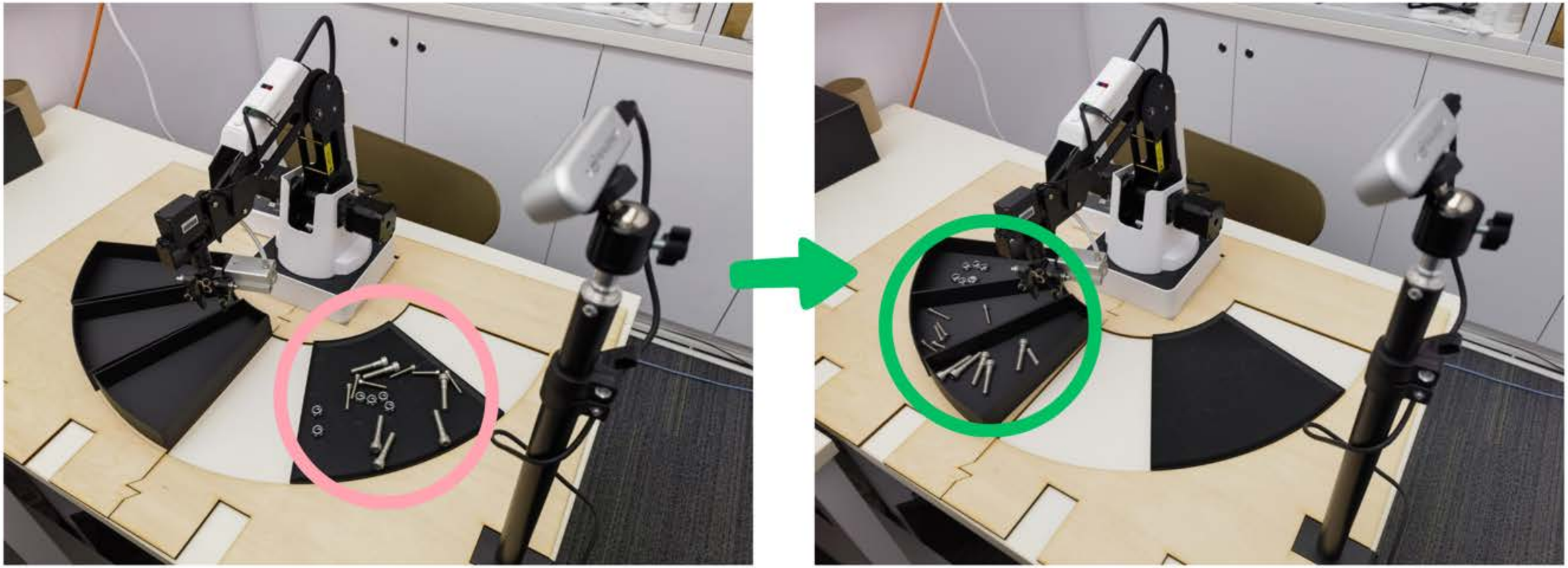
Robot Arm Challenge

Robot Arm Programming

Presented by HKU ArmStrong SIG

What is the challenge?

What is the challenge?



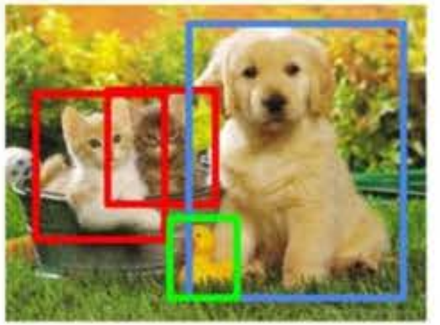
Autonomous
random-bin picking

What skill sets are required?

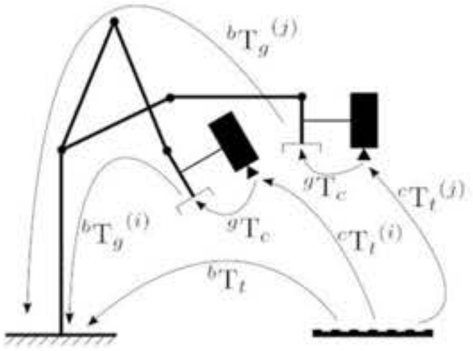
To Move
To See
To Think

Robot Arm Challenge

10th Sep	Workshop 1 Robotic Arm Control 🦾
24th Sep	Workshop 2 Object Detection 👀
8th Oct	Workshop 3 Hand-eye Calibration 🤝
5th Nov	Challenge Day 🏆
And a lot of help desk session to be announced!	



CAT, DOG, DUCK



Today’s Workshop

1. Learn basic programming practice
2. Learn to control robotic arm with Python
3. Understand the basic kinematics behind the robotic arm

Hardware in Robotics Arm Challenge

DOBOT Magician

Realsense D405
Depth Camera

Part 1.
How to do programming

How to describe a thing

Sleeping

Barking

Searching and locating the ball

Chasing a ball

How to describe a thing

Sleeping

Barking

Searching and locating the ball

Chasing a ball

This case is highly related to our challenge

How to describe a thing

Type
Dog

Actions
Sleep
Bark
See
Chase ball

Features
Name: Max
Gender: Male
Age: 7
Weight: 20kg

What if we define dog as a common object

```
1  my_dog = Dog("Max", "Male", 7, 20)
```

Type
Dog

Features
Name: Max
Gender: Male
Age: 7
Weight: 20kg

Actions
Sleep
Bark
See
Chase ball

```
class Dog:
 def __init__(self, name, gender, age, weight):
   self.name = name
   self.gender = gender
   self.age = age
   self.weight = weight

 def sleep(self):
   print(f"{self.name} is sleeping.")

 def bark(self):
   print(f"{self.name} says Woof!")

 def see(self):
   print(f"{self.name} is looking around.")

 def chase_ball(self):
   print(f"{self.name} chases the ball.")
```

What if we define dog as a common object

Class
A blueprint for creating objects

Attribute
Data stored in the object.

Method
Function that belongs to the object.

Type
Dog

Features
Name: Max
Gender: Male
Age: 7
Weight: 20kg

Actions
Sleep
Bark
See
Chase ball

```python
class Dog:
 def __init__(self, name, gender, age, weight):
   self.name = name
   self.gender = gender
   self.age = age
   self.weight = weight

 def sleep(self):
   print(f"{self.name} is sleeping.")

 def bark(self):
   print(f"{self.name} says Woof!")

 def see(self):
   print(f"{self.name} is looking around.")

 def chase_ball(self):
   print(f"{self.name} chases the ball.")
```

Try to make 3 dog in programming language

dog1

“Charlie”

dog2

“Bella”

dog3

“Max”

```python
# Dog1
dog1_name = "Charlie"
dog1_gender = "Male"
dog1_age = 7
dog1_weight = 20

def dog1_sleep():
 print("Charlie is sleeping.")

def dog1_bark():
 print("Charlie says Woof!")

def dog1_see():
 print("Charlie is looking around.")

def dog1_chase_ball():
 print("Charlie chases the ball.")
```

dog1

“Charlie”

```python
# Dog2
dog2_name = "Bella"
dog2_gender = "Female"
dog2_age = 5
dog2_weight = 18

def dog2_sleep():
 print("Bella is sleeping.")

def dog2_bark():
 print("Bella says Woof!")

def dog2_see():
 print("Bella is looking around.")

def dog2_chase_ball():
 print("Bella chases the ball.")
```

dog2

“Bella”

```python
# Dog3
dog3_name = "Max"
dog3_gender = "Male"
dog3_age = 3
dog3_weight = 15

def dog3_sleep():
 print("Max is sleeping.")

def dog3_bark():
 print("Max says Woof!")

def dog3_see():
 print("Max is looking around.")

def dog3_chase_ball():
 print("Max chases the ball.")
```

dog3

“Max”

Very messy

The messy code gets harder to manage as you add more dogs.

```
# Using the dogs
print(dog1_name, dog1_gender, dog1_age, dog1_weight)
dog1_sleep()
dog1_bark()
dog1_see()
dog1_chase_ball()

print(dog2_name, dog2_gender, dog2_age, dog2_weight)
dog2_sleep()
dog2_bark()
dog2_see()
dog2_chase_ball()

print(dog3_name, dog3_gender, dog3_age, dog3_weight)
dog3_sleep()
dog3_bark()
dog3_see()
dog3_chase_ball()
```

Very messy

The messy code gets harder to manage as you add more dogs.

Try to make 3 dog-objects

Object
An actual thing created using the class

```
dog1 = Dog("Charlie", "Male",7,20)
dog2 = Dog("Bella", "Female",5,18)
dog3 = Dog("Max", "Male",3,15)
```

Here, we have 3 objects (dog1, dog2, dog3) with the same class -- “Dog”

Each object has different attributes

dog1
“Charlie”

dog2
“Bella”

dog3
“Max”

```python
# Dog1
dog1_name = "Charlie"
dog1_gender = "Male"
dog1_age = 7
dog1_weight = 20

def dog1_sleep():
 print("Charlie is sleeping.")

def dog1_bark():
 print("Charlie says Woof!")

def dog1_see():
 print("Charlie is looking around.")

def dog1_chase_ball():
 print("Charlie chases the ball.")

# Dog2
dog2_name = "Bella"
dog2_gender = "Female"
dog2_age = 5
dog2_weight = 18

def dog2_sleep():
 print("Bella is sleeping.")

def dog2_bark():
 print("Bella says Woof!")

def dog2_see():
 print("Bella is looking around.")

def dog2_chase_ball():
 print("Bella chases the ball.")

# Dog3
dog3_name = "Max"
dog3_gender = "Male"
dog3_age = 3
dog3_weight = 15

def dog3_sleep():
 print("Max is sleeping.")

def dog3_bark():
 print("Max says Woof!")

def dog3_see():
 print("Max is looking around.")

def dog3_chase_ball():
 print("Max chases the ball.")

# Using the dogs
print(dog1_name, dog1_gender, dog1_age, dog1_weight)
dog1_sleep()
dog1_bark()
dog1_see()
dog1_chase_ball()

print(dog2_name, dog2_gender, dog2_age, dog2_weight)
dog2_sleep()
dog2_bark()
dog2_see()
dog2_chase_ball()

print(dog3_name, dog3_gender, dog3_age, dog3_weight)
dog3_sleep()
dog3_bark()
dog3_see()
dog3_chase_ball()
```

```python
class Dog:
 def __init__(self, name, gender, age, weight):
   self.name = name
   self.gender = gender
   self.age = age
   self.weight = weight

 def sleep(self):
   print(f"{self.name} is sleeping.")

 def bark(self):
   print(f"{self.name} says Woof!")

 def see(self):
   print(f"{self.name} is looking around.")

 def chase_ball(self):
   print(f"{self.name} chases the ball.")

# Create multiple Dog objects
dog1 = Dog("Charlie", "Male",7,20)
dog2 = Dog("Bella", "Female",5,18)
dog3 = Dog("Max", "Male",3,15)

# Use the dogs
for dog in [dog1, dog2, dog3]:
 print(dog.name, dog.gender, dog.age, dog.weight)
 dog.sleep()
 dog.bark()
 dog.see()
 dog.chase_ball()
```

The code become more **clean** and **less redundant**

Possible description for Dobot

Type

Dobot

Actions

move_to
get_pose
locate_objects
grip

Features

Port: COM1
ID: 101

Part 2.
Robotic Arm Control

Flow

2.1. Basic Information and Operations

2.2. Arm control via Computer

2.3. Programming and Coding

2.1. Basic Information and Operation

Basic Information and Operation

Turn on Dobot Magician

Check status of Robotic Arm

Green

- Robotic arm ready to use

Red

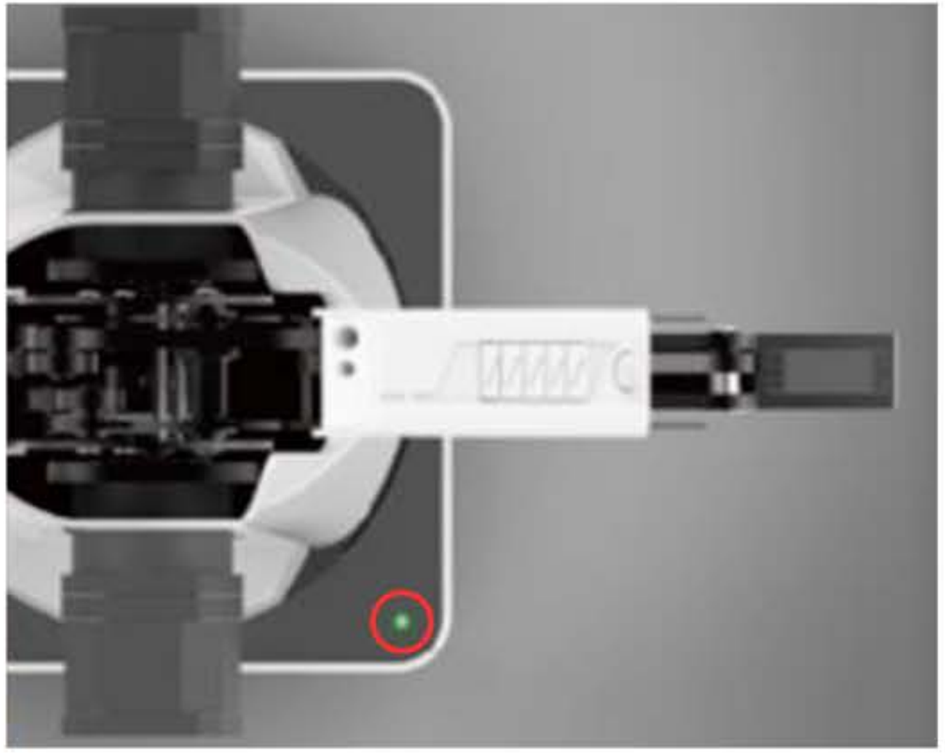
- Robotic arm at limited position

Orange

- Robotic arm is powered on, and all the stepper motors lock (Not usable)

Blinking blue

- Robotic arm is homing



ONLY USE THE ARM WHEN THE LIGHT IS GREEN

Unlock and Drag the robotic arm

Press and hold the unlock button
to drag the robotic arm around

2.2. Arm Control via Computer
2.3. Programming and Coding

Arm Control via Computer

Open the python notebook (dobot.ipynb)

Dobot Magician Robot Arm Control Tutorial

This is the tutorial for the dobot magician robot arm. The code below demonstrates how to control the robot arm using Python.

Learning Objectives

- Understand how to connect to the Dobot Magician robot arm.
- Learn how to move the robot arm to specific positions.
- Explore how to control the robot arm's gripper.

Prerequisites

- Basic knowledge of Python programming.

Outcomes

- A program that can control the Dobot Magician robot arm to move to specific positions and pick up objects.

[1]:

```
# Find the port of the Dobot Magician
from serial.tools import list_ports
from pydobotplus import Dobot, CustomPosition
import time

available_ports = list_ports.comports()
port = available_ports[2].device
def find_robot_arm_port():
    print("Step 1: Please ensure the Dobot Magician is plugged in, then press Enter ")
```

Try to connect your robotic arm

- Run **Practice 1** to find the USB port of robotic arm

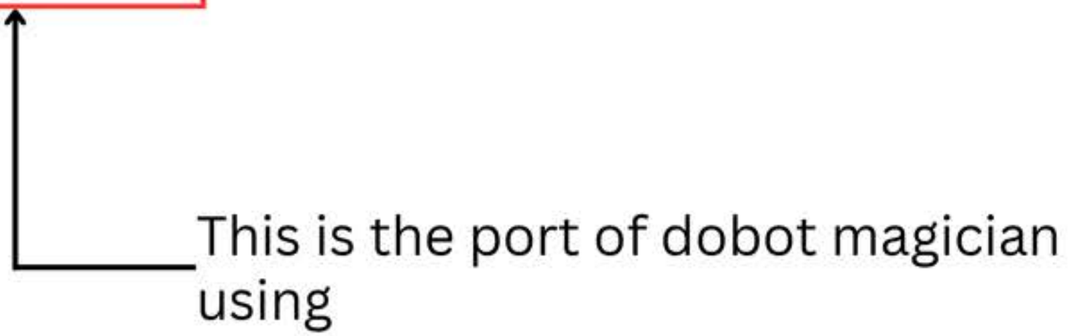
Practice 1

```python
# Find the port of the Dobot Magician
from serial.tools import list_ports
import time

available_ports = list_ports.comports()
port = available_ports[0].device
def find_robot_arm_port():
    print("Step 1: Please ensure the Dobot Magician is plugged in, then press Enter.")
    input()
    ports_before = {port.device for port in list_ports.comports()}

    print("Step 2: Now unplug the Dobot Magician, then press Enter.")
    input()
    ports_after = {port.device for port in list_ports.comports()}

    diff_ports = ports_before - ports_after
    if not diff_ports:
        print("No port difference detected. Please try again.")
        return None
    robot_port = diff_ports.pop()
    print(f"Dobot Magician detected on port: {robot_port}")
    return robot_port

port = find_robot_arm_port()
print(f"Using port: {port}")
```

Try to connect your robotic arm

Expected output

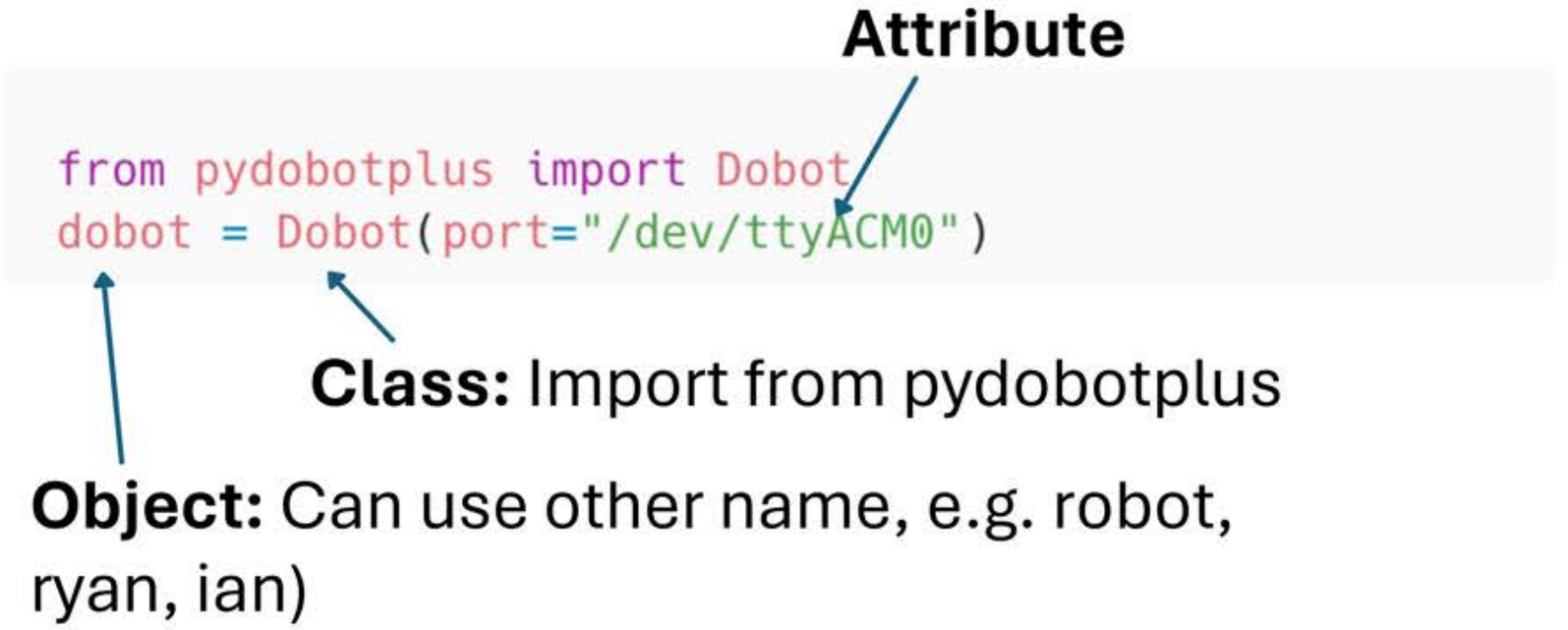
```
Step 1: Please ensure the Dobot Magician is plugged in, then press Enter.
Step 2: Now unplug the Dobot Magician, then press Enter.
Dobot Magician detected on port: /dev/ttyACM0
Using port: /dev/ttyACM0
```



Try to connect your robotic arm

Connect to the robotic arm

- Create an **object** of Dobot, named dobot
- Remember to add **attribute** to specify which port we are connecting to



Try to connect your robotic arm

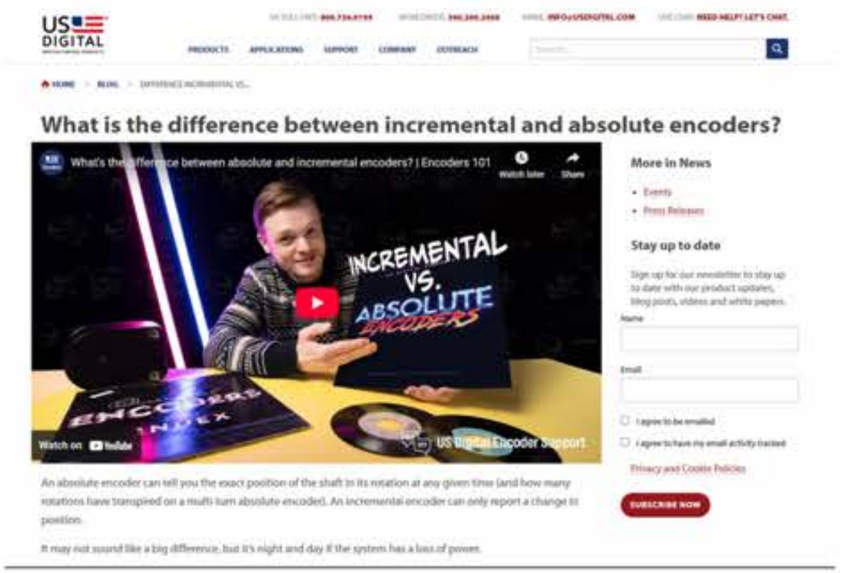
- Run Practise 2 to **verify the connection**
- If connected correctly, the robotic arm will run “home” command

Practice 2

```python
from pydobotplus import Dobot, CustomPosition

# Home the Dobot Magician
my_dobot = Dobot(port="COM4")
my_dobot.home()

print("Dobot Magician is homing.")
```

What is .home()

- A command to **re-calibrate** the robotic arm position (Encoder* parameter)
- The robotic arm will rotate clockwise to the limited position and then return to the homing point automatically
- **Run it before each task for precision!**



* You may learn more about Encoder here in this blog

Controlling the robotic arm

- Run **Practice 3a**

Practice 3a

```
import time

# After running this code, the robot arm should move forward for 50mm
print("Original position of the robot arm is:", my_dobot.get_pose().position)
my_dobot.move_to(300, 0, 50, 0, wait=True)
print("Moved robot arm to position:", my_dobot.get_pose().position)
```

Get the position of the robot arm — (x, y, z)

Move to (300, 0, 50, 0)

Programming and Coding

Basic robot control

Recall Practice 3a:

.move_to(x, y, z, r, wait=True)

It is a function to move the arm to given coordinate

```
dobot.move_to(x=100, y=100, z=50, wait=True)
```

Basic robot control

.get_pose()

It is a function to get the current coordinate of the **gripper**

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

Controlling the robotic arm

- Run **Practice 3b**

```
# Close the gripper
dobot.grip(True)
print("Gripper is closed.")
time.sleep(2)

# Open the gripper
dobot.grip(False)
print("Gripper is opened.")
```

Open the gripper

Close the gripper

Exercise 1

Move robotic arm to (100, -200, 50, 0)

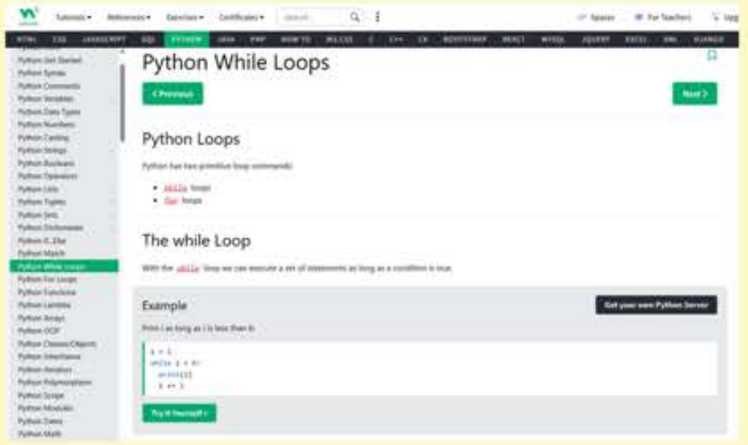
Exercise 2

Move robotic arm with the user interface and copy down the final coordinate.

Exercise 3

Move the robot arm repeatedly between (100, -200, 50, 0) and (-100, 200, 70, 0)

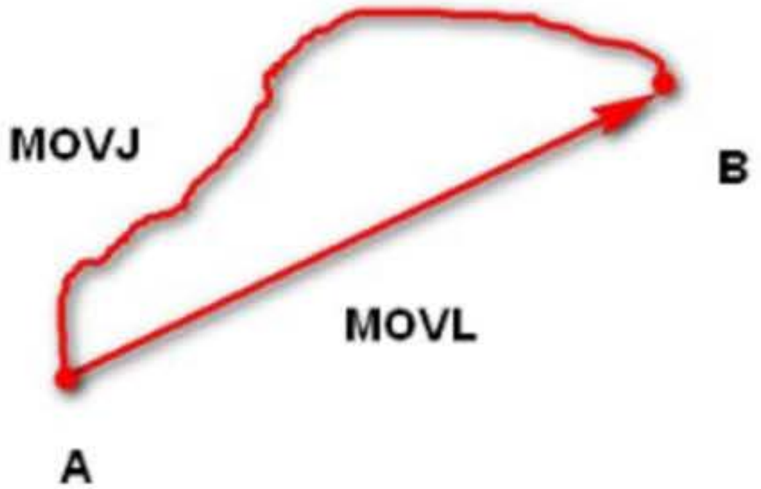
- To achieve this, you may make use of a while-loop* sequence



* You may learn more about while-loop here

Different movement method

- **MOVJ**
  - From point A to point B, each joint will run from initial angle to its target angle, regardless of the trajectory.
- **MOVL**: Rectilinear movement
  - The joints will perform a straight-line trajectory from point A to point B



Exercise 4

Move the robot arm between (100, 200, 50, 0) and (100, -200, 50, 0) with **movl** and **movj** respectively.

```
my_dobot.move_to(100, -200, 50, 0,
mode=dobot.MODE_PTP.MOVL_XYZ)
```

MOVL

```
my_dobot.move_to(100, -200, 50, 0,
mode=dobot.MODE_PTP.MOVJ_XYZ)
```

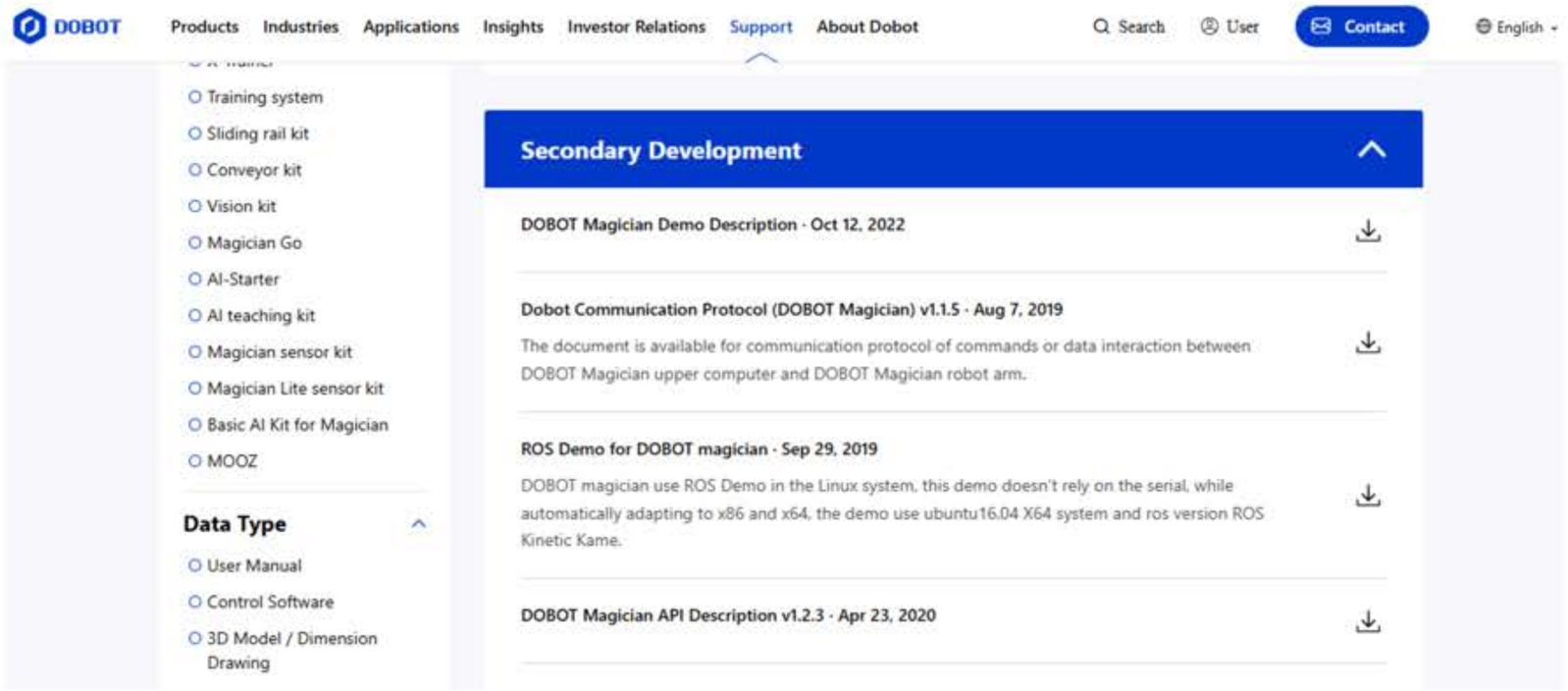
MOVJ

- Observe the difference!

Extra Information

Understand how the program works

- Start with the source: the **official documentation** is your best friend.



Keywords to search online: documentation, user menu, protocol, API

https://www.dobot-robots.com/service/download-center

Extra Information

From Python Code → Protocol Docs

- Explore how the code maps to the robot's protocol.

Source: pydobotplus.py

```
834     def move_to(self, x=None, y=None, z=None, r=0, wait = True, mode=None, position=None):
835         if position is not None:
836             x, y, z, r = position.x, position.y, position.z, position.r
837         elif x is None and y is None and z is None:
838             raise ValueError("Either a Position object or x, y, z coordinates must be provided")
839
840         current_pose = self.get_pose().position
841         if x is None: x = current_pose.x
842         if y is None: y = current_pose.y
843         if z is None: z = current_pose.z
844         if r is None: r = current_pose.r
845         print(current_pose)
846
847         if mode is None:
848             mode = MODE_PTP.MOVJ_XYZ  # Use default mode if not provided
849
850         return self._extract_cmd_index(self._set_ptp_cmd(x, y, z, r, mode, wait = wait))
```

```
401
402     def _set_ptp_cmd(self, x, y, z, r, mode, wait):
403         msg = Message()
404         msg.id = 84
405         msg.ctrl = 0x03
406         msg.params = bytearray([])
407         msg.params.extend(bytearray([mode]))
408         msg.params.extend(bytearray(struct.pack('f', x)))
409         msg.params.extend(bytearray(struct.pack('f', y)))
410         msg.params.extend(bytearray(struct.pack('f', z)))
411         msg.params.extend(bytearray(struct.pack('f', r)))
412         return self._send_command(msg, wait)
```

Source: Dobot communication Protocal Menu

1.10.5 SetPTPCmd

This command is to execute PTP command, the issued command packet format is shown in Table 101, and the returned command packet format is shown in Table 102.

Table 101 The command packet of SetPTPJointParams

Header	Len	Payload				Checksum
		ID	Ctrl		Params	
			rw	isQueued		
0xAA 0xAA	2+17	84	1	1 or 0	PTPCmd（See Program 17）	Payload checksum

method set_ptp_cmd() sends a command to move the robot.

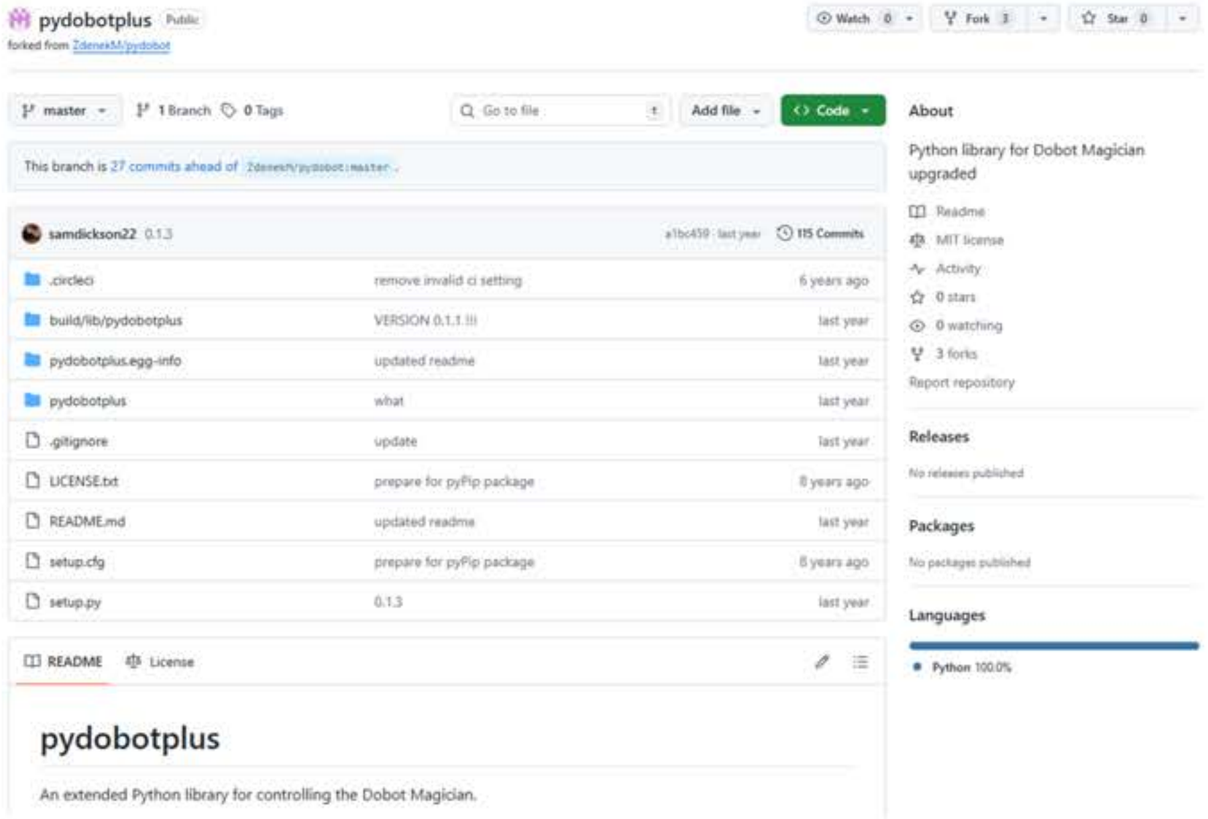
- **SetPTPCmd corresponds to protocol ID 84**

https://download.dobot.cc/product-manual/dobot-magician/pdf/en/Dobot-Communication-Protocol-V1.1.5.pdf

Extra Information

Using Open-Source Libraries

- Open-source libraries are a **bridge** between your code and the robot's API



https://github.com/sammydick22/pydobotplus/blob/master/pydobotplus/dobotplus.py

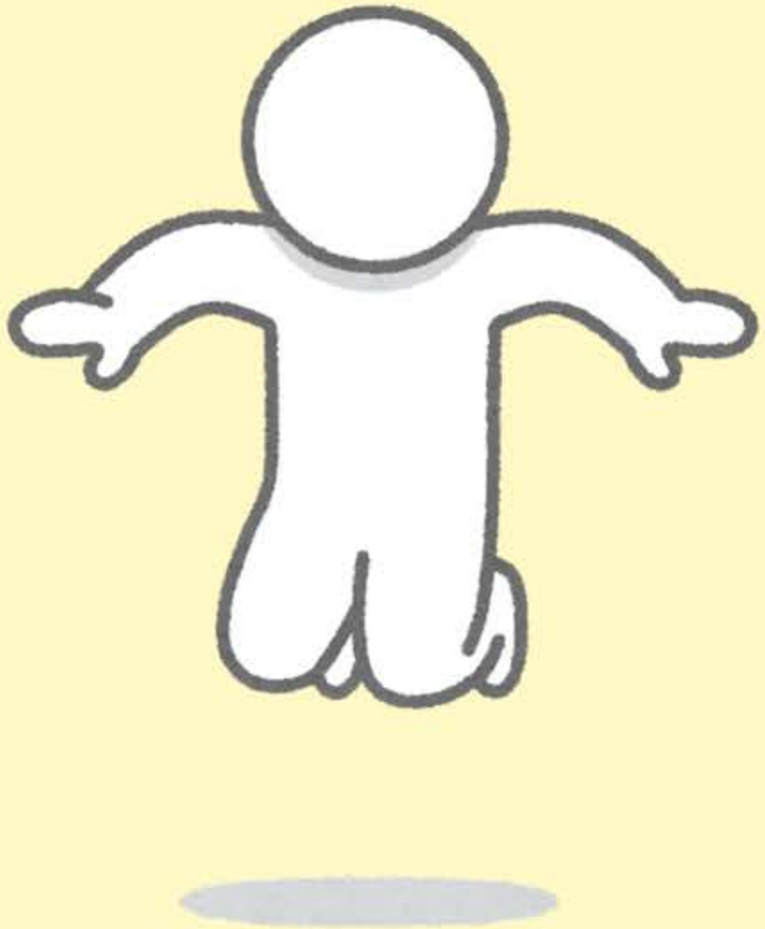
Different movement method

- Try to write you own method, follow the “Don’t Repeat Yourself” Principle
- Prevent collision by using **jump_to()** movement method (Create this method by yourself)

Programming and Coding

Exercise 5

create a **.jump_to** function!



Coordinate System

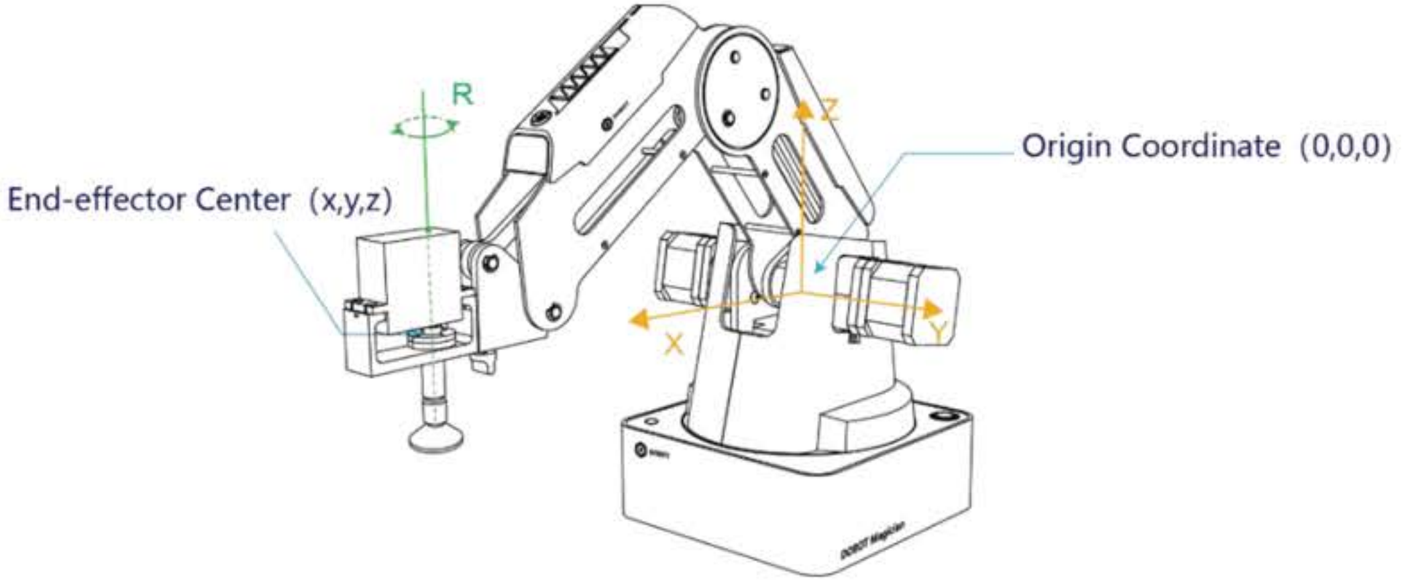


Figure 2.5 Cartesian coordinate system

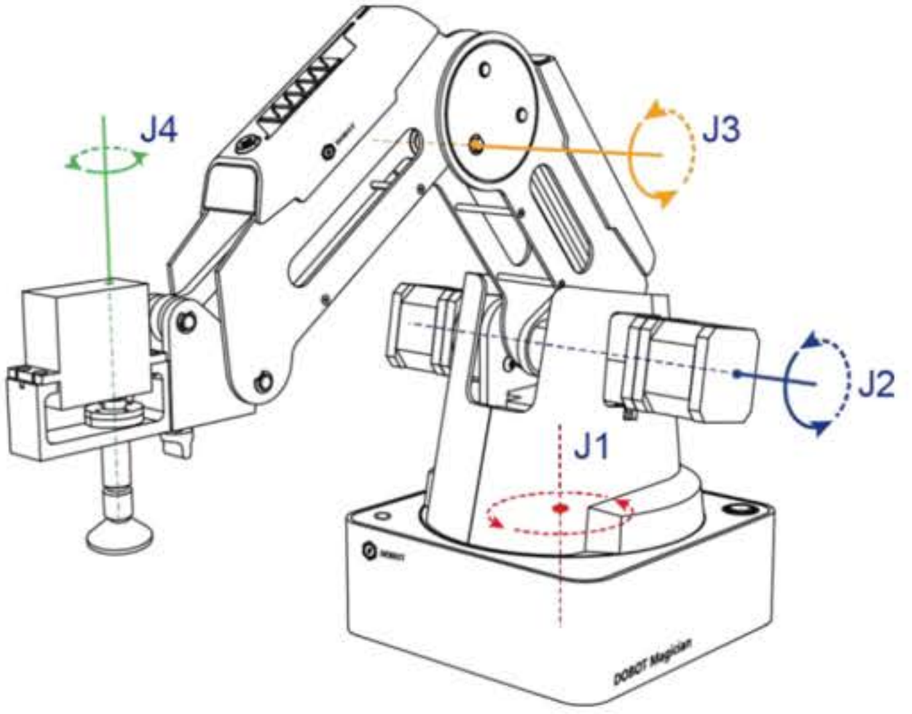


Figure 2.4 Joint coordinate system

Workspace

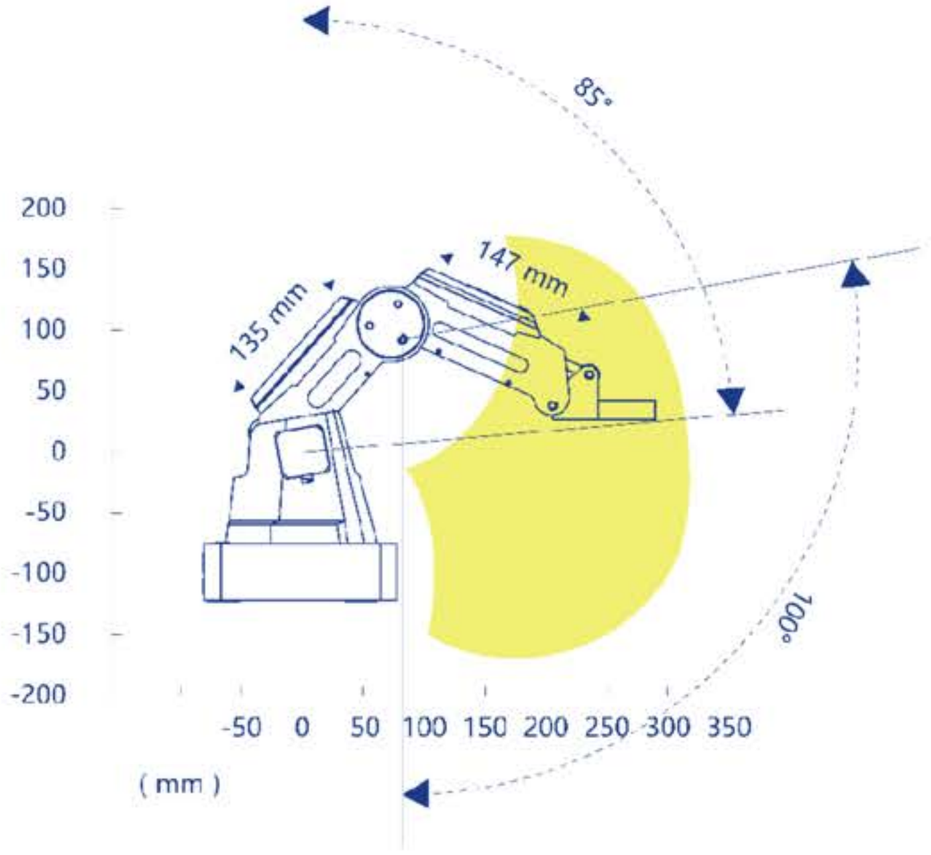


Figure 3.2 Workspace of Dobot Magician (1)

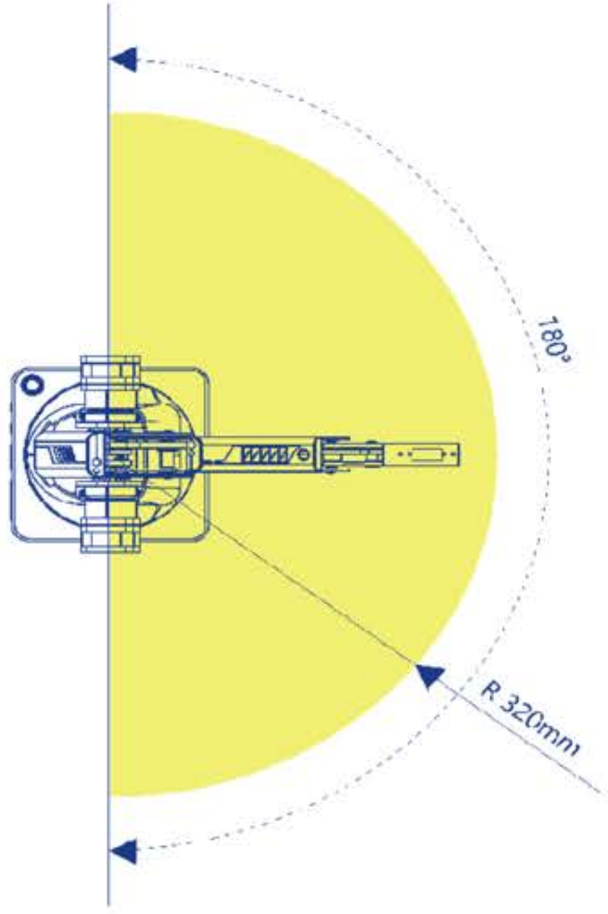


Figure 3.3 Workspace of Dobot Magician (2)

Workspace

Name	Dobot Magician	
Maximum payload	500g	
Maximum reach	320mm	
Motion range	Base	- 90°~+90°
	Rear Arm	0°~85°
	Forearm	- 10°~+90°
	End-effector rotation	- 90°~+90°

Exercise 6

1. Pick the screw in a fixed position
2. Pick it up and move the dropping zone
3. Drop the screw and return to home pose

Part 3.
Maths and Theories behind

How do we know the gripper's pose?

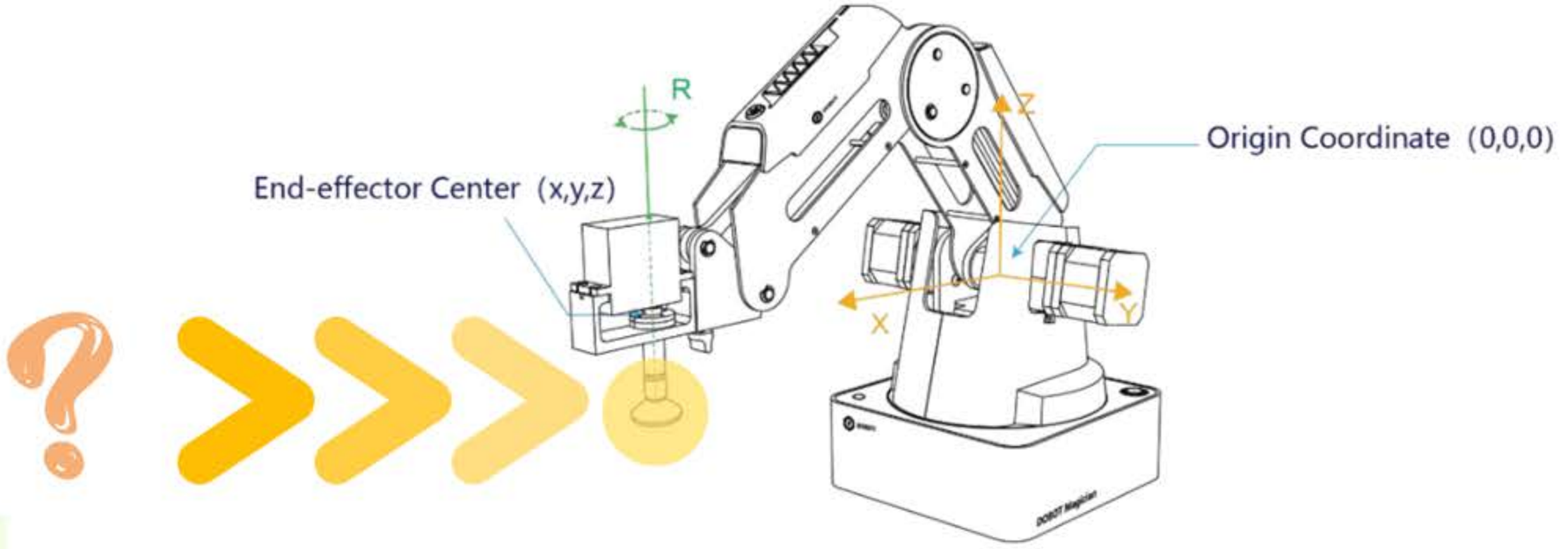


Figure 2.5 Cartesian coordinate system

Recall the get_pose function:

Programming and Coding

Basic robot control

.get_pose()

It is a function to get the current coordinate of the gripper

The Math and Theories behind

Locating the End Effecter

- The robotic arm knows the angle of each joint through the encoders* inside the motor
- By “Forward Kinematics”, we can calculate the pose (e.g. x, y, z coordinates) of the end effector

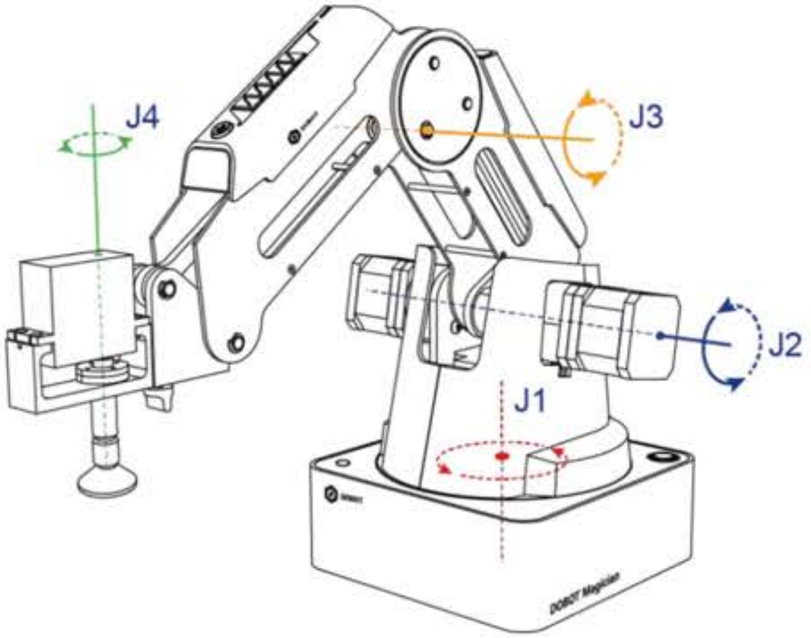


Figure 2.4 Joint coordinate system

End-effectors are devices attached to the end of a robot's arm to help it interact with the surrounding environment.

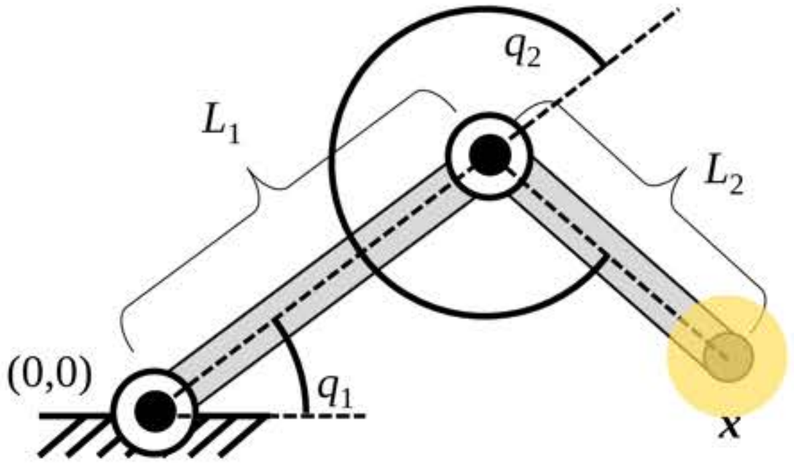
* You may learn more about Encoder here

The Math and Theories behind

Forward Kinematics (FK)

- **Input (Known):**
  - Joint angle
  - Length between each joint
- **Output:**
  - End-effector position and orientation (also known as “pose”)

Example



The robotic arm in-built controller already solves the FK for you, so you don’t need any manual calculation!

How does the robotic arm know the correct joint movement, to move to a goal pose?

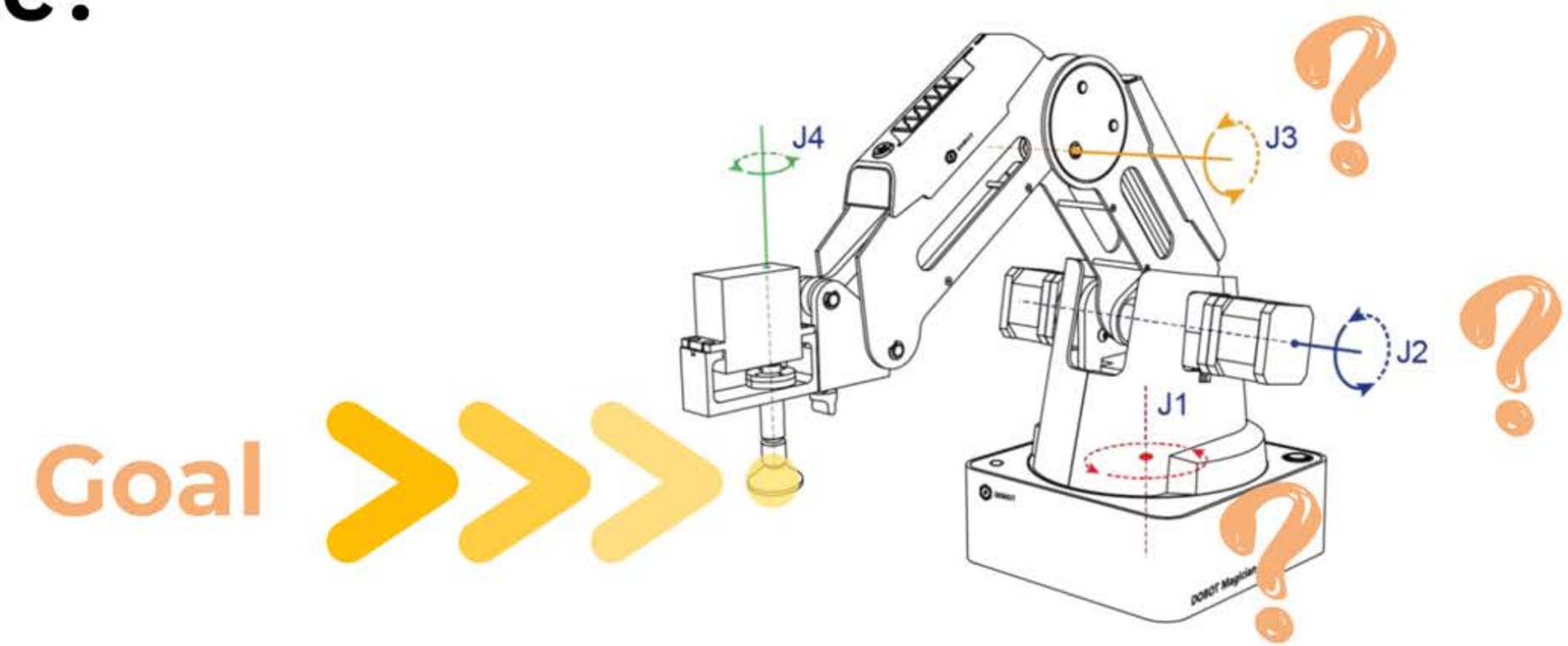


Figure 2.4 Joint coordinate system

The Math and Theories behind

Inverse Kinematics (IK)

- **Input:**
  - End-effector pose (e.g. x, y, z)
  - The length between each joint
- **Output:**
  - Joint angles

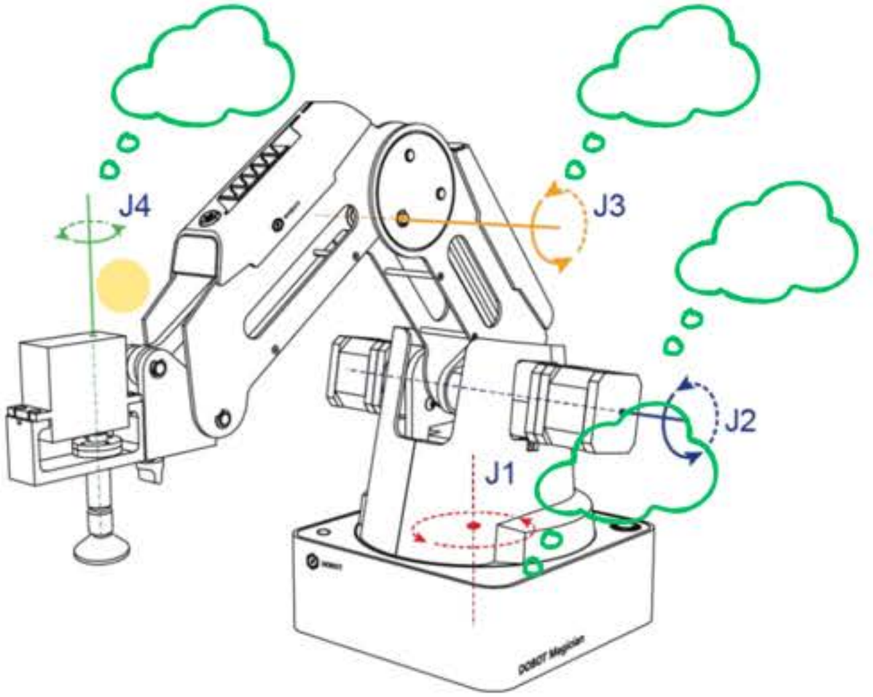
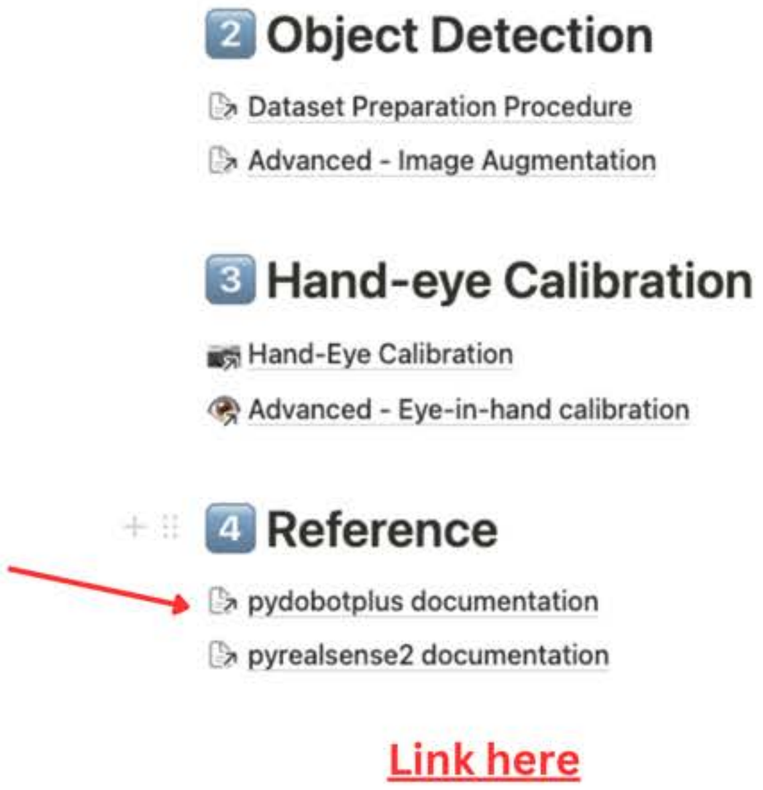


Figure 2.4 Joint coordinate system

The robotic arm in-built controller already solves the IK for you, so you don't need any manual calculation!

More robot control method

- See our self-learning material



Link here

pydobotplus documentation

An extended Python library for controlling the Dobot Magician robotic arm.

Table of Contents

I am now using pydobotplus, what does the function .move_to do?

1:18 AM

Assistant

In `pydobotplus`, the `.move_to` function is used to command a Dobot robotic arm to move to a specified position in Cartesian coordinates. This function typically takes parameters such as the x, y, and z coordinates, as well as the speed of the movement and possibly other configuration settings, depending on the specific implementation.

Here's a general overview of how it works:

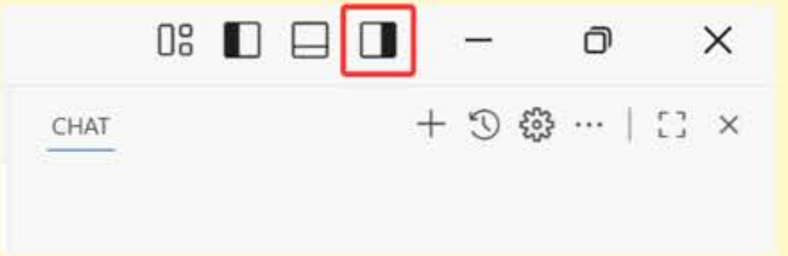
Parameters

- **x**: The target x-coordinate for the arm to move to.
- **y**: The target y-coordinate.
- **z**: The target z-coordinate.
- **speed**: (optional) The speed at which to move to the target position.
- **wait**: (optional) A boolean indicating whether to wait for the movement to complete before proceeding.

Hmm, seems not so accurate

Take-home Exercise

Try to ask Copilot in Github about how to use the function “move_rel” in pydobotplus. Compare it to the official documentation and see if it’s correct.



.move_rel(x=0, y=0, z=0, r=0, wait=True) Moves the Dobot relative to its current position, keeps any unspecified parameters the same.

- **x**: *float* relative x coordinate
- **y**: *float* relative y coordinate
- **z**: *float* relative z coordinate
- **r**: *float* relative r rotation
- **wait**: *bool* waits until the command has been executed before returning to the process

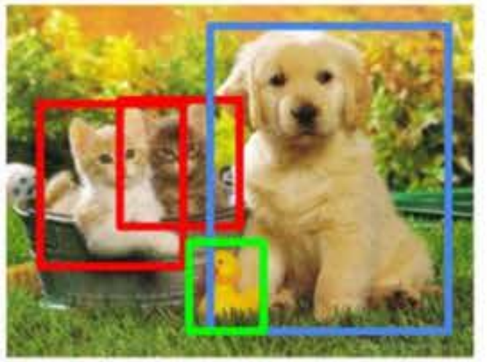
Next workshop – Object Detection

Teach your robot to observe the world!

Classification

CAT

Object Detection



CAT, DOG, DUCK

Object Segmentation

How to configure your own PC to control the robotic arm?

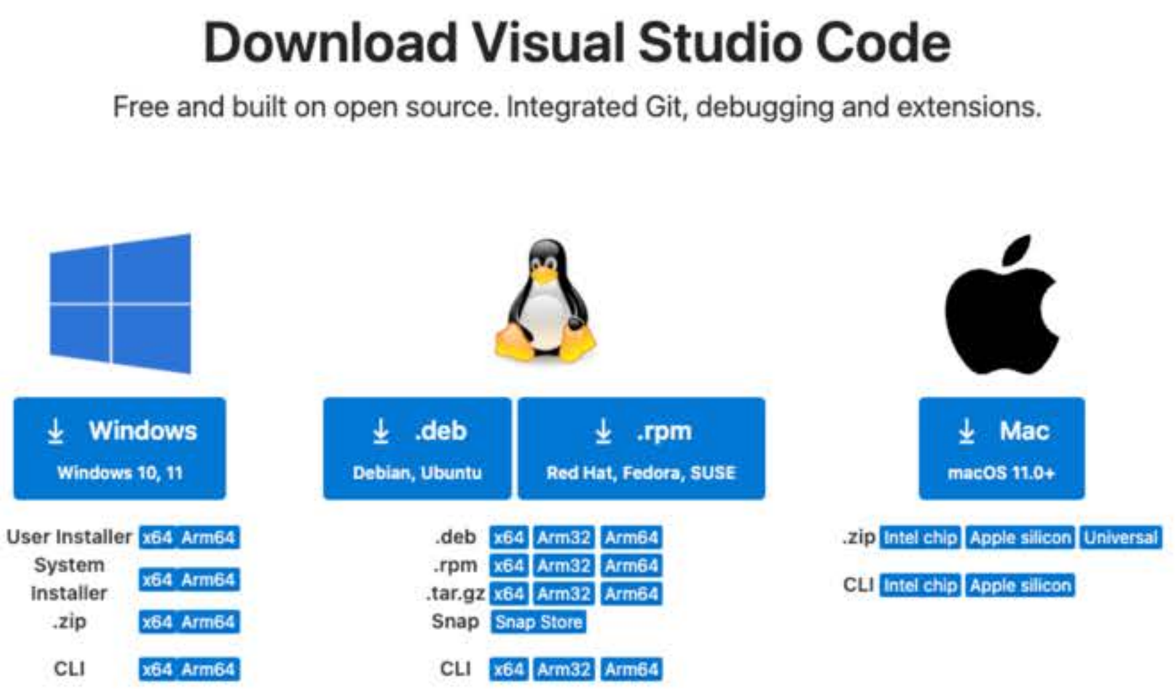
Software requirement

VS Code

Python 3.10

Git / GitHub Desktop

VS Code Install



https://code.visualstudio.com/Download

Python Install

- We recommend install conda to simplify the environment config

```
conda --version
```

Run this command to verify your installation

GitHub Account

- GitHub is a web-based platform used primarily for version control and collaborative software development.
- We recommend you use GitHub to collaborate with your teammate

If you don’t familiar with using Git CLI, we also suggest you install GitHub Desktop

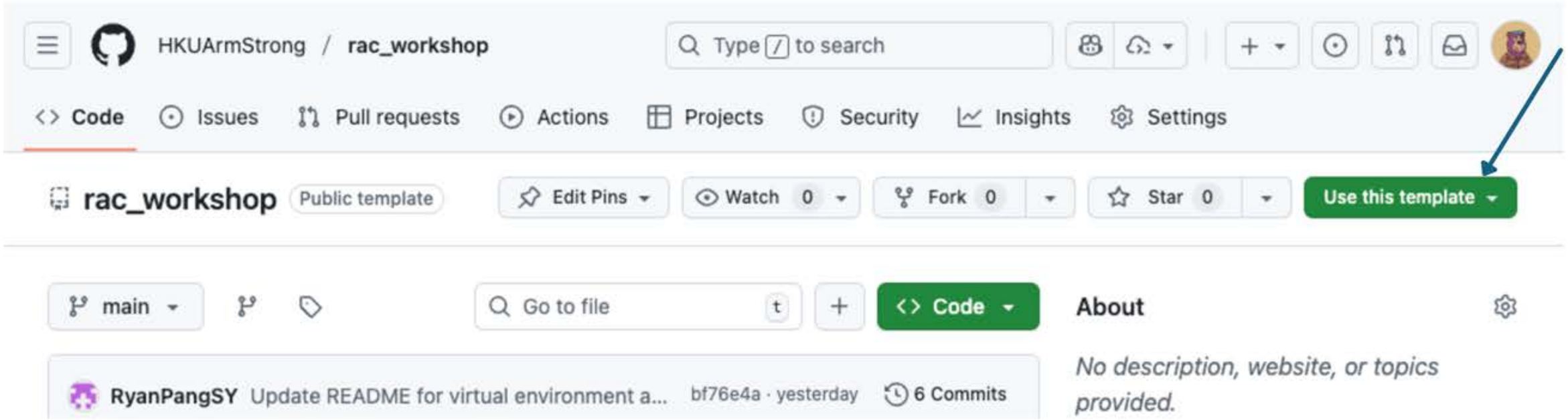
GitHub Student Developer Pack

- FREE GPT 4.1 for Coding

https://github.com/education/students

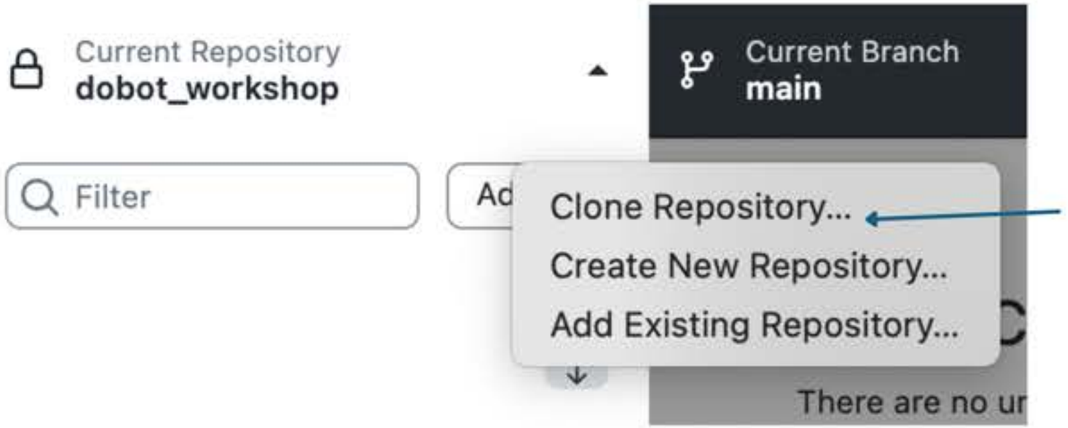
GitHub

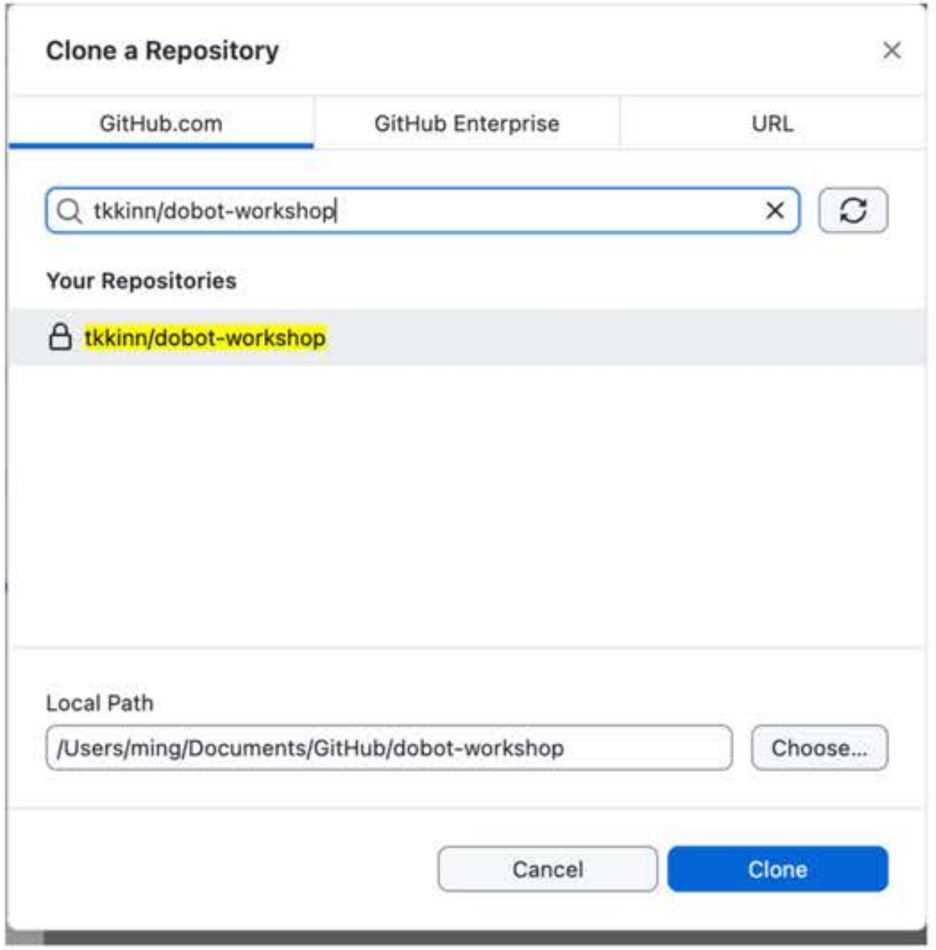
Create you own repository



https://github.com/HKUArmStrong/rac_workshop

Clone your repository

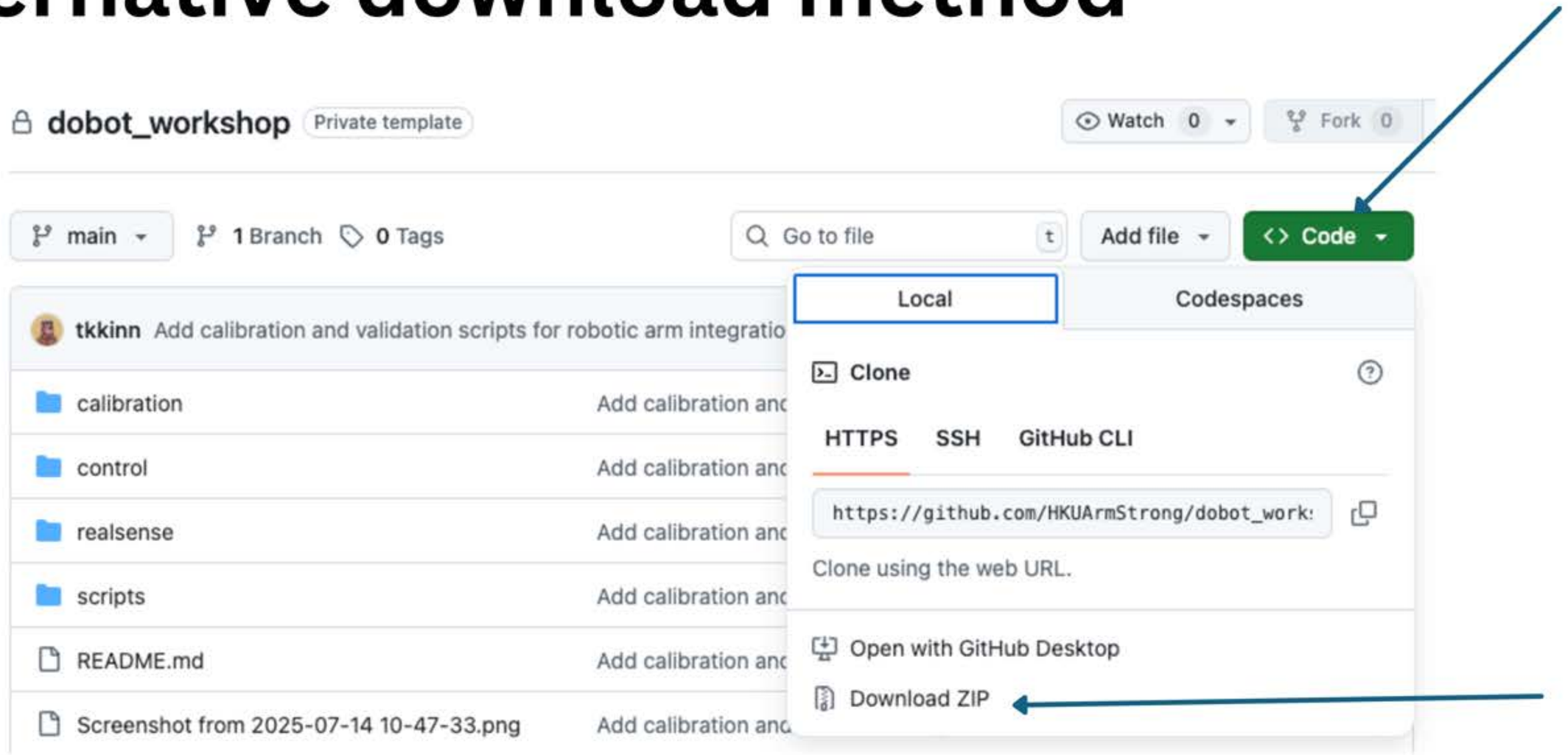




Clone your repository to your computer

GitHub

Alternative download method




[Community] Robot Arm Challenge
WhatsApp group
Scan or upload this QR code using the WhatsApp camera to join this group