Robotic Arm Control, Python Programming & Coding Practice

Outline

- Object Oriented Programming for Robotic Arm Control
- Coding Style
- Environment Setup

OOP for Robotic Arm Control

What functions should we realize in this bin-picking competition?

- Identify the position of the bolts or nuts
- Move the robotic arm or gripper
- Close the gripper to pick up the bolts and nuts
- Open the gripper to drop down the objects
- Classify the bolts with specific length
- …

Robotic Arm Control

Object

What is an object in object-oriented programming?

- An object is a collection of **data** and **method**
- **Data:** the attribute, properties or features
- **Method:** the action to perform, operating on the data
- An **object** is an instance of a **class**
- A class is an abstract of the objects (a blueprint)

Example

(Ask AI to illustrate)

```python
class Car:
    def __init__(self, color, make, model):
        self.color = color
        self.make = make
        self.model = model

    def drive(self):
        print(f"The {self.color} {self.make} {self.model} is driving.")

# Creating an object (instance) of Car
my_car = Car("red", "Toyota", "Corolla")
my_car.drive()  # Output: The red Toyota Corolla is driving.
```

Data/Features/Properties

Method operating on data

Object is an instance of Class

OOP for Robotic Arm Control

- Object Oriented: We can design a class without details

```
RoboticArm:
  data:
    joint_angles
    end_effector_position
  methods:
    move_joint()
    move_gripper()
    close_gripper()
    release_gripper()
```

OOP for Robotic Arm Control

- Object Oriented: We can design a class without details
- Modularity: we can break down complex class into smaller, manageable classes

```
RoboticArm:
  data:
    gripper:Gripper
    joint_angles
    end_effector_position
  methods:
    move_joint()
    move_gripper()
```

```
Gripper:
  data:
    finger_distance
    force_applied
  methods:
    close(max_force)
    release(max_force)
    set_distance(expected_distance)
```

OOP for Robotic Arm Control

- Object Oriented: We can design a class without details
- Modularity: we can break down complex class into smaller, manageable classes
- Changes on a class will not affect others. Easy to maintain and update

```
RoboticArm:
  data:
    gripper:Gripper
    joint_angles
    end_effector_position
  methods:
    move_joint()
    move_gripper()
```

Delete a method from RoboticArm

```
Gripper:
  data:
    finger_distance
    force_applied
  methods:
    close(max_force)
    release(max_force)
    set_distance(expected_distance)
```

No change

```
Car:
  data:
    color
    make
    model
  methods:
    drive()
```

No change

```
Cat:
  data:
    age
    weight
  methods:
    eat()
```

No change

OOP for Robotic Arm Control

- Object Oriented: We can design a class without details
- Modularity: we can break down complex class into smaller, manageable classes
- Changes on a class will not affect others. Easy to maintain and update
- Inheritance: we can create a new class that inherits data and methods from the existing class

```
RoboticArm:
  data:
    gripper:Gripper
    joint_angles
    end_effector_position
  methods:
    move_joint()
    move_gripper()

MyRobot(RoboticArm):
  methods:
    pick_and_place()
    peg_in_hole()
```

OOP for Robotic Arm Control

- Object Oriented: We can design a class without details
- Modularity: we can break down complex class into smaller, manageable classes
- Changes on a class will not affect others. Easy to maintain and update
- Inheritance: we can create a new class that inherit data and methods from existing classes
- Reusability, Encapsulation, Polymorphism, ...

Robotic Arm Control Structure

Dobot Hardware

Motor ↔ Embedded System

Dobot Software

pydobotplus (Open-Source Python Library)

```python
from serial.tools import list_ports
from pydobotplus import Dobot, CustomPositi

available_ports = list_ports.comports()
print(f'available ports: {[x.device for x i
port = available_ports[2].device

device = Dobot(port=port)

# Create a custom position
pos1 = CustomPosition(x=200, y=50, z=50)

# Move using direct coordinates
device.move_to(x=200, y=50, z=50)

# Move using custom position
device.move_to(position=pos1)
```

Customized Software

MyDobot (Class inherited from Dobot)

```python
class MyDobot(dobot.Dobot):
    def jump_to(self, x, y, z, r, height=50, wait=
        current_pose = self.get_pose().position
        self._set_ptp_cmd(current_pose.x, current_
        self._set_ptp_cmd(x, y, z + height, r, mod
        self._set_ptp_cmd(x, y, z, r, mode=dobot.M

    def suck(self, enable, wait_time=0):
        self._set_end_effector_suction_cup(enable)
        time.sleep(wait_time)
```

Coding Style

- Beyond “It works”
- Readability
- Maintainability

Beyond “It works”

- Fulfilling functionality requirement is not enough for coding
- Example: Create a class for a circle, and calculate its area
- It works:

```python
class A:
    def __init__(self, v):
        self.v = v

    def m(self):
        return self.v * 2

obj = A(5)
print(obj.m())
```

What are the potential issues? How will you modify?

Beyond "It works"

- Easy to understand
  - Clear naming
  - Docstring
  - Comment
- Easy to modify
  - OOP
- Easy to use
  - Example usage

```python
class Circle:
    """A class to represent a circle."""

    def __init__(self, radius):
        """Initialize the circle with a radius."""
        self.radius = radius

    def calculate_area(self):
        """Calculate the area of the circle."""
        import math
        return math.pi * (self.radius ** 2)

# Example usage
circle = Circle(5)
area = circle.calculate_area()
print(f"Circle with radius {circle.radius} has area: {area:.2f}")
```

In collaborative project, coding practice is important!

Readability

- Idea: Code should be easy to read and understand for both your teammates and **yourself**.
- Practice:
  - Descriptive Naming: Use meaningful names for variables, functions, and classes.
  - Consistent Formatting: Follow a consistent style guide (e.g., indentation, spacing) to improve visual flow. (Search: Google Python Style Guide)
  - Docstring and Comment: Explain the purpose of the functions and classes. Add comments to clarify complex logic. (Avoid over-commenting, ensure concise and complementary with code)
  - Ask AI for help ☺

styleguide

Google Python Style Guide

▼ Table of Contents

- 1 Background
- 2 Python Language Rules
  - 2.1 Lint
  - 2.2 Imports
  - 2.3 Packages
  - 2.4 Exceptions
  - 2.5 Mutable Global State
  - 2.6 Nested/Local/Inner Classes and Functions
  - 2.7 Comprehensions & Generator Expressions
  - 2.8 Default Iterators and Operators
  - 2.9 Generators
  - 2.10 Lambda Functions
  - 2.11 Conditional Expressions
  - 2.12 Default Argument Values
  - 2.13 Properties
  - 2.14 True/False Evaluations
  - 2.16 Lexical Scoping
  - 2.17 Function and Method Decorators
  - 2.18 Threading
  - 2.19 Power Features
  - 2.20 Modern Python: from __future__ imports
  - 2.21 Type Annotated Code
- 3 Python Style Rules
  - 3.1 Semicolons
  - 3.2 Line length
  - 3.3 Parentheses
  - 3.4 Indentation
    - 3.4.1 Trailing commas in sequences of items?
  - 3.5 Blank Lines
  - 3.6 Whitespace
  - 3.7 Shebang Line
  - 3.8 Comments and Docstrings
    - 3.8.1 Docstrings
    - 3.8.2 Modules
    - 3.8.2.1 Test modules
    - 3.8.3 Functions and Methods
    - 3.8.3.1 Overridden Methods
    - 3.8.4 Classes
    - 3.8.5 Block and Inline Comments
    - 3.8.6 Punctuation, Spelling, and Grammar
  - 3.10 Strings
    - 3.10.1 Logging
    - 3.10.2 Error Messages
  - 3.11 Files, Sockets, and similar Stateful Resources

- 3.12 TODO Comments
- 3.13 Imports formatt
- 3.14 Statements
- 3.15 Accessors
- 3.16 Naming
  - 3.16.1 Names tc
  - 3.16.2 Naming (
  - 3.16.3 File Nami
  - 3.16.4 Guideline
- 3.17 Main
- 3.18 Function length
- 3.19 Type Annotatio
  - 3.19.1 General F
  - 3.19.2 Line Brea
  - 3.19.3 Forward I
  - 3.19.4 Default V
  - 3.19.5 NoneTyp
  - 3.19.6 Type Alia:
  - 3.19.7 Ignoring
  - 3.19.8 Typing Va
  - 3.19.9 Tuples vs
  - 3.19.10 Type var
  - 3.19.11 String ty
  - 3.19.12 Imports For Typing

3.16 Naming

`module_name`, `package_name`, `ClassName`, `method_name`, `ExceptionName`, `function_name`, `GLOBAL_CONSTANT_NAME`, `g`
`instance_var_name`, `function_parameter_name`, `local_var_name`, `query_proper_noun_for_thing`, `send_acronym_via`

Names should be descriptive. This includes functions, classes, variables, attributes, files and any other type of name

Avoid abbreviation. In particular, do not use abbreviations that are ambiguous or unfamiliar to readers outside you abbreviate by deleting letters within a word.

Always use a `.py` filename extension. Never use dashes.

3.16.1 Names to Avoid

- single character names, except for specifically allowed cases:
  - counters or iterators (e.g. `i`, `j`, `k`, `v`, et al.)
  - `e` as an exception identifier in `try/except` statements.
  - `f` as a file handle in `with` statements
  - private type variables with no constraints (e.g. `_T = TypeVar("_T")`, `_P = ParamSpec("_P")`)
  - names that match established notation in a reference paper or algorithm (see Mathematical Notation)

Maintainability

- Idea: Make adding functions, debugging, testing manageable. Try not to overwhelm yourself when project grows larger and more complicated.
- Practice:
  - Modularity: keep functions short and focused on a single task; try to write reusable code (utility)
  - Error Handling: manage exception and prevent crashes
  - Version Control: learn using Git, track your changes
  - Decoupling: try to reduce dependencies between components so that you can modify independently without affecting other parts

Environment Setup

- **Definition**: A virtual environment is an isolated workspace that contains its own Python (or other programming language) installation, libraries, and dependencies.
- **Key Features**:
    - Separates project-specific dependencies.
    - Avoids conflicts between package versions.
    - Ensures projects' reproducibility across systems.
- **Example**: A Python 3.8 environment for one project, Python 3.10 for another, each with distinct libraries.

Why Do We Need Virtual Environments?

- **Dependency Management**:
  - Different projects require different package versions (e.g., NumPy 1.19 vs. 1.23).
  - Prevents version conflicts that could break applications.
- **Isolation**:
  - Keeps global Python installation clean.
  - Avoids unintended interactions between projects.
- **Reproducibility**:
  - Share exact environment setups with teammates.
  - Simplifies deployment to production or testing.
- **Portability**:
  - Run projects consistently across different machines.

Introducing Conda

- **What is Conda?**:
  - **Open-source** package and environment management system.
  - Works across Python, R, and other languages.
  - Available via Anaconda or Miniconda.
- **Why Use Conda?**:
  - Manages both Python and non-Python dependencies.
  - **Cross-platform** (Windows, macOS, Linux).
  - Simplifies complex dependency resolution.

How to use Conda

- **Creating an Environment**:
  - `conda create -n myenv python=3.9`
- **Activating an Environment**:
  - `conda activate myenv`
- **Installing Packages**:
  - `conda install numpy pandas`
- **Exporting for Reproducibility**:
  - `conda env export > environment.yml`
- **Deactivating**:
  - `conda deactivate`

Benefits of Using Conda

- **Ease of Use**:
  - User-friendly commands for environment creation and management.
- **Flexibility**:
  - Supports multiple Python versions and non-Python tools.
- **Community and Ecosystem**:
  - Access to thousands of packages via Anaconda repository.
- **Environment Sharing**:
  - Share environment.yml for consistent setups across teams.
  - Listing the packages required in a 'requirements.txt' file.

Installing Conda

- Instruction
  - https://www.anaconda.com/docs/getting-started/miniconda/install#windows-installation
- Practice
  - Create an environment with python 3.10 and check the python version
  - Activate the environment
  - Install opencv-python package through conda
  - Deactivate the environment
  - Creating a requirements.txt

Arm Control

Download the following files: Code to distribute (Public)

Instructions:

https://www.notion.so/Arm-Control-1a4c652096f2804ba3f0e5e4668b375f