Programming Method 2 – Object-Oriented Control of Mechatronic Systems

Tri Bien Minh

Jan, 2026

Course Information

Course Title: Programming Method 2 – Object-Oriented Control of Mechatronic Systems
Programs: Mechatronics Engineering, Robotics, Automation
Programming Language: Python
Tool: MuJoco, Git, Google Colab, Robotics Python Toolbox **Hardware:** Your laptop, equipments from the robotics lab.

Prerequisites

Students are expected to have completed:

- Programming Method 1 (C or Python)
- Basic control theory concepts
- Basic knowledge of sensors and actuators

Course Description

This course introduces object-oriented programming (OOP) not merely as a syntax, but as a rigorous **Engineering Practice Focus** for controlling complex mechatronic systems. In an industrial environment, software is rarely written from scratch and never exists in isolation; therefore, this course simulates the **professional lifecycle of an engineer.**

1. From Scripting to System Architecture

Students transition from writing "disposable" scripts to developing **production-grade software**. The focus is on creating systems that are:

- **Maintainable:** Code that can be understood by other engineers six months later.
- **Extensible:** Architectures that allow adding a new sensor or a multiple axes to a robot without rewriting the core logic.

2. The Software-Hardware Contract

A central theme is the **Hardware Abstraction Layer (HAL)**. Engineers will learn to design "contracts" (interfaces) that decouple the high-level control logic from the specific electrical hardware.

3. Professional Workflow and Tooling

To prepare for a career in robotics or embedded systems, students will operate within a **modern DevOps ecosystem**. This includes:

- **Version Control (Git):** Managing features via branches and disciplined commit history.
- **Automated Verification:** Shifting from "manual testing" to **Unit and Integration Testing** using `pytest`.
- **Code Quality:** Enforcing industry style guides and utilizing static analysis to catch bugs before they ever reach the physical hardware.

4. Deterministic Logic and Safety

Unlike web development, mechatronic software can cause physical harm. We emphasize **Automata-based design (FSMs)** to ensure the system is always in a known, deterministic state. Students will learn to model safety interlocks and error states as well as understand in their software architecture.

5. Transferability and Language Agnostic Design

While Python is the implementation tool, the **architectural patterns** (State Pattern, Dependency Injection, Observer Pattern) are taught with a view toward **C++ and ROS-based (Robot Operating System)** environments. This ensures that the mental models developed here are directly applicable to high-performance industrial control and autonomous systems.

Course Learning Outcomes (CLOs)

By the end of the course, students will be able to:

1. Model technical systems using software variables and internal states
2. Implement automata-based (finite state machine) control systems
3. Design object-oriented architectures for sensors and actuators
4. Apply abstraction and polymorphism to achieve hardware-independent control
5. Use Python containers and algorithms professionally for data processing
6. Verify software using unit, module, and system-level tests
7. Use professional development tools such as Git, debuggers, build and test systems

---

Teaching and Learning Methods

- Lectures with engineering-focused explanations
- Weekly structured laboratory sessions
- System modeling and design before coding

- Incremental project-based learning
- Code walkthroughs and oral explanations
- **AI tools may be used for conceptual explanation only.**
- **AI-generated code, logic, or architectural designs are not permitted unless explicitly stated.**
- **Students must be able to explain, derive, and modify all submitted code.**

Software and Tools

- Python 3.10+
- Virtual environments (venv)
- Git for version control
- pytest for unit testing
- black for formatting
- logging module for runtime diagnostics
- Debugger (VSCode IDE)

Assessment Breakdown

- **Final project (code, tests, report): 20%** Group of 2-3 students
- **Final writing exam: 80%**

Books and References

- **Clean Code: A Handbook of Agile Software Craftsmanship** *Author:* Robert C. Martin
- **Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures** *Author:* Hassan Gomaa
- **Fluent Python: Clear, Concise, and Effective Programming** *Author:* Luciano Ramalho
- **Robotics, Vision and Control: Fundamental Algorithms in Python** *Author:* Peter Corke *Focus:* The essential bridge between robotics theory (kinematics, dynamics) and Python implementation. Highly relevant for Chapters 7, 8 and 9.

Supplemental Resources (Tools)

- **Python3 Tutorial** - https://docs.python.org/3/tutorial/index.html
- **Pro Git** (Scott Chacon and Ben Straub) – Available for free at git-scm.com.
- **Official MuJoCo Documentation** – https://mujoco.readthedocs.io/en/stable/overview.html For simulation and physics-based modeling.

Course Schedule

Chapter 0 – Introduction to Engineering Software for Mechatronic Systems

Lecture Topics

- Role of software in mechatronic systems
- Difference between scripting and control software
- Review of procedural vs object-oriented programming
- Git workflows and version control discipline
- Static analysis and type checking
- Debugging techniques

Laboratory

- Development environment setup
- Git repository initialization
- Python project structure

Deliverable

- Tooling and debugging report
- Verified development environment and Git repository

Chapter 1 – Introduction to Python Programming

Chapter Objectives

After completing this chapter, students will be able to:

- Understand why Python is used in engineering and mechatronic systems
- Distinguish Python scripting from engineering software development
- Set up a professional Python development environment
- Write and run basic Python programs
- Understand Python's execution model and structure
- Follow professional coding and repository practices

Lecture Topics

1. Python in Engineering and Mechatronics

- Python in robotics, automation, and control
- Python vs C/C++: roles and trade-offs
- Python as a system-level engineering language

2. Scripting vs Engineering Software

- Linear scripts vs structured programs
- Modularity, responsibility, and reuse
- Deterministic behavior in control software

3. Python Execution Model

- Modules and source files
- Interpreter execution flow
- Global vs local scope
- Entry point
- Import mechanisms

4. Basic Python Syntax and Semantics

- Variables and data types
- Conditionals and loops
- Functions and return values

5. Writing First Engineering-Style Programs

- Function-based structure
- Avoiding global state
- Naming and readability

6. Development Environment Setup

- Python versioning
- Virtual environments (venv)
- Package installation
- Project structure
- Debugging with VS Code

7. Professional Coding Practices

- Code formatting and style
- Using black
- Git repository initialization
- Commit message standards

Laboratory

Lab 1.1 – Python Environment Setup

- Install Python 3.10+
- Create and activate a virtual environment
- Verify installation

Lab 1.2 – First Engineering Python Program

- Write and run a structured Python program
- Debug execution flow

Lab 1.3 – Project and Repository Structure

- Initialize Git repository
- Organize project files
- Commit code professionally

Deliverables

- Verified Python development environment
- Structured Python program
- Git repository with proper commits

Chapter 2 – Modeling Technical Variables

Lecture Topics

- Translating physical systems into software variables
- Inputs, outputs, internal states, and constraints
- Naming, units, and responsibility

Laboratory

- Variable analysis for a real system (e.g., servo motor on mujoco simulation)

Deliverable

- Technical Variable Table (approved before coding)

Chapter 3 – System Decomposition and Class Responsibility

Lecture Topics

- Functional decomposition
- Identifying classes from system components
- Single responsibility principle

Laboratory

- Component and responsibility diagrams
- Initial class identification

Deliverable

- Class responsibility document

Chapter 4 – Automata/Finite State Machines (FSM)

Lecture Topics

- FSM concepts: states, events, transitions, guards
- Deterministic behavior in control systems
- State diagrams

Laboratory

- FSM modeling on paper and diagrams
- Peer review of FSM logic

Deliverable

- FSM diagram (mandatory approval)

Chapter 5 – Implementing State Machines in Python and Practices

Lecture Topics

- State representation using enums
- Transition logic and state integrity
- Introduction to the State design pattern

Laboratory

- FSM implementation in Python
- Debugging invalid transitions

Deliverable

- Working FSM module with tests

Chapter 6 – Sensors and Actuators as Software Objects

Lecture Topics

- Sensors as data providers
- Interface-based design using abstract base classes
- Simulation vs real sensors
- Actuators as command receivers
- Safety and limits in actuator software
- Separation of control logic and hardware access

Laboratory

- Sensor interface definition
- Simulated sensor implementations
- Actuator interface and implementations
- Polymorphic actuator usage

Deliverable

- Sensor class hierarchy

- Actuator abstraction and implementation

Chapter 7 – System Integration, Data Processing and Algorithms

Lecture Topics

- Controller–sensor–actuator interaction
- Dependency inversion
- Lists, dictionaries, sets
- Sorting, filtering, and aggregation
- Algorithmic thinking in control software

Laboratory

- Integration of FSM, sensors, and actuators
- Refactoring for clarity and testability
- Managing multiple sensors and actuators
- Data processing using Python standard library

Deliverable

- Integrated control subsystem
- Data processing module

Chapter 8 – Case study: Modeling and Kinematics of a Simple Robot Arm

Chapter Objectives

After completing this chapter, students will be able to:

- Understand the kinematic structure of a simple robot arm
- Identify joints, links, and degrees of freedom
- Translate physical geometry into software models
- Represent joint configurations using vectors and constraints
- Implement forward kinematics for a simple articulated mechanism
- Verify kinematic correctness through simulation and testing

Lecture Topics

1. Robot Arm as an Engineering System

- Degrees of freedom (DOF)
- Joint types: revolute and prismatic
- Link–joint relationships
- Workspace vs configuration space
- Assumptions and simplifications for educational robot arms

2. Mathematical Modeling for Software

- Joint variables as software states
- Vector representation of joint positions
- Units, limits, and constraints
- Separation of model, algorithm, and controller

3. Forward Kinematics (Conceptual Level)

- Purpose of forward kinematics
- Geometric interpretation (planar 2-DOF arm)
- Incremental computation using joint angles
- Numerical implementation without symbolic math libraries

Laboratory

Lab 8.1 – Modeling a Simple Robot Arm

- Define joint and link parameters
- Create a software representation of a planar robot arm
- Define joint limits and initial states

Lab 8.2 – Implementing Forward Kinematics

- Implement forward kinematics for a 2-DOF planar arm
- Verify correctness via simulation visualization
- Compare expected vs computed end-effector positions

Deliverables

- Robot arm software model (classes and documentation)
- Forward kinematics algorithm implementation
- Unit tests validating kinematic correctness

Chapter 9 – Case study: Motion Algorithms and Control Integration for Robot Arms

Chapter Objectives

After completing this chapter, students will be able to:

- Implement basic motion algorithms for articulated mechanisms
- Design reusable, object-oriented motion algorithms
- Handle physical constraints and safety in software
- Integrate motion algorithms with FSMs, sensors, and actuators
- Validate motion behavior through simulation and testing

Lecture Topics

1. Simple Motion Algorithms

- Point-to-point joint motion
- Linear interpolation in joint space
- Time-based vs step-based motion
- Velocity and acceleration constraints
- Saturation and safety handling

2. Algorithm Design in Object-Oriented Form

- Algorithm as a class with inputs, internal state, and outputs
- Stateless vs stateful algorithms
- Strategy pattern for motion algorithms

3. Error Handling and Robustness

- Invalid target configurations
- Joint limit violations
- Numerical instability
- Safe fallback behavior

4. Integration with Control Architecture

- FSM-driven motion execution
- Actuator command abstraction
- System-level behavior observation

Laboratory

Lab 9.1 – Joint-Space Motion Algorithm

- Implement a joint-space interpolation algorithm
- Enforce joint limits and smooth motion

Lab 9.2 – Integration with FSM and Actuators

- Integrate motion algorithm into FSM
- Connect outputs to actuator interfaces
- Observe behavior in MuJoCo simulation

Deliverables

- Joint-space motion algorithm module
- Integrated FSM-controlled robot arm simulation

Chapter 11 – Final Project Integration and Review

Lecture Topics

- Architecture review
- Common design mistakes in control software
- Preparation for final defense

Laboratory

- Final project integration
- Instructor-led design review

Deliverable

- Final project code freeze

Chapter 12 – Final Evaluation

Assessment Overview

The final evaluation is a holistic review of your ability to function as a Junior Software Engineer. It consists of four primary components:

- **Project Demonstration:** Live execution of your software within the MuJoCo environment.
- **Oral Defense:** A 7-minute presentation and Q&A explaining your architectural choices (e.g., "Why this FSM structure?").
- **Code Walkthrough:** A peer-level review of your source code, focusing on Clean Code principles and logic flow.
- **Testing Review:** Validation of your `pytest` suite and your system's response to edge cases.

Final Project: Object-Oriented Robot Control

1. Project Overview

The final project is your opportunity to build a "full-stack" control system for a mechatronic system in the **MuJoCo simulation**. The goal is to move from "writing scripts" to **"building a system."** You will be evaluated on your ability to use Object-Oriented Programming (OOP) to make a robot perform a task reliably and safely.

Key Constraints:

- **Duration:** 5 weeks.
- **Team Size:** 2-3 students.
- **Technical Requirements:** Must implement Classes, a Finite State Machine (FSM), and a Hardware Abstraction Layer (HAL), technical documentations.

2. Some Project Ideas, freely to add more.

- **The Sorting Machine:** Use a robot arm to detect and sort objects into bins based on color or shape.
- **The Path Follower:** Program a mobile robot to navigate waypoints while avoiding obstacles using simulated distance sensors.
- **Automated Assembly:** Coordinate an arm to perform a pick and place task or stack blocks in a specific sequence.

- **Safety Monitor:** Implement a supervisor system that enters an `EMERGENCY_STOP` state if joint limits or collisions are detected.

3. Final Project Deliverables

Phase 1: The Proposal (~500 words), 2-3 weeks

A "forcing function" to crystalize your idea. It must include:

- **The Goal:** A clear definition of the "Success State."
- **The Class Diagram:** A sketch of your intended software architecture.
- **The State Machine:** A diagram showing states, transitions, and guards.
- **HAL Strategy:** How you will decouple control logic from simulation commands.

Phase 2: Progress Update,

A brief check-in to verify your **Virtual Environment** setup and a "Hello World" movement in MuJoCo.

Phase 3: Final Video (2-3 minutes) - final week

A screen-recording demonstrating:

1. The robot successfully completing the task.
2. The **Debug Console** showing real-time FSM state transitions.
3. A brief explanation of one design pattern used (e.g., Strategy or Observer).

Phase 4: Final Technical Report - final week

A professional engineering document including:

- **System Architecture:** How classes interact (Dependency Diagram).
- **Technical Variable Table:** Exhaustive list of inputs, outputs, and internal states.
- **Verification:** Data from testing (e.g., "90% success rate over 20 iterations").
- **Git Repository:** Link to clean, formatted and commented code with professional commit history.

4. Grading Rubric

Criteria	Weight	Focus
OOP Mastery	30%	Proper encapsulation, inheritance, and hardware decoupling.
Reliability	20%	Deterministic behavior and graceful error handling.
Testing	20%	Quality and coverage of `pytest` scripts.
Documentation	15%	Clarity of the report and Git discipline.

Criteria	Weight	Focus
Presentation	15%	Presenation and demo quality.

5. Tips for Success

1. **Keep it Simple:** A robust 2-joint arm is better than a buggy 6-joint arm.
2. **Borrow, but Credit:** Use MuJoCo examples for physics, but the **control architecture** must be your own.
3. **Test Early:** Ensure your logic classes work in isolation before plugging them into the physics engine.

Assessment Breakdown

- **Final project (code, tests, report): 20%**
- **Final writing exam: 80%**

Academic Integrity Policy

- All submitted code must be explainable by the student
- Inability to explain submitted code may result in failure
- AI tools may be used for conceptual explanation only.

Expected Graduate Attributes

Upon successful completion, students will demonstrate:

- Engineering-level software modeling skills
- Object-oriented design competence
- Professional verification and tooling discipline
- Readiness for advanced robotics, embedded systems, and ROS-based development