System Design Proposal for Humanoid Robot Software

Author: Victor Angelier
Date: 15 September 2025
Github:
https://github.com/va-angelier/va-angelier.github.io/blob/main/OOP_PCOM7E_Assignment/docs/index.md

---

Background Research: Terms & Rationale

robot, humanoid robot — scope & capabilities in industry tasks.
human-robot interaction (HRI) — interface/feedback and task alignment with operators.
collaborative robot (cobot) — safety, task sharing, and guard conditions.
path planning, A* — efficiency in obstacle grids and performance ceiling.
state machine, safety — predictability, recovery paths, and testable guards.
manipulation / grasping — grasp failure, undo actions, and operator feedback.

System Design Rationale

The software implemented in robot/robot_system.py supports navigation, object manipulation, and communication for a warehouse humanoid robot, accessible via python -m robot. It prioritises safety, maintainability, and testability, aligning with SDLC principles (Sommerville, 2015). A composition-based architecture uses a Robot façade to orchestrate Navigation, Manipulator, Communicator, Environment, and MemoryStore, reducing coupling and enabling test-time substitution (stubs/mocks). This avoids inheritance hierarchies that often tighten coupling and blur responsibilities (Derezińska, 2015; Siciliano and Khatib, 2016).

The RobotState enum (OFF, IDLE, MOVING, MANIPULATING, COMMUNICATING, CHARGING, ERROR) encodes explicit transitions with guards. Commands are rejected during CHARGING (except tick for progress), while a low-battery guard triggers auto-docking by planning a path to the charger and switching to CHARGING until 100% state of charge; this behaviour is validated in robot/tests/test_auto_charge.py (Mukherjee et al., 2022). The state machine yields predictable recovery (e.g., tick from ERROR back to IDLE when safe), outperforming ad-hoc flags; the model is reflected in uml/state_diagram.puml (Rumbaugh, Jacobson and Booch, 2005).

Data structures are purpose-driven (Siciliano and Khatib, 2016):

- **List** — *Environment.objects* and *sensor_readings* (dynamic append, O(1)).
- **Dictionary** — *object_index* for O(1) ID lookups, trading memory for speed.
- **Stack** — Manipulator.*grasp_history* and MemoryStore.*breadcrumbs* for LIFO undo (O(1)).
- **Queue** — Navigation.*path_queue* and CLI.cmd_queue for FIFO processing (O(1)).
- **Search/Planning**: A* planner with Euclidean heuristic and obstacle checks.

Navigation employs A* with a Euclidean heuristic (time complexity O(V log V + E)), chosen over Dijkstra's (no heuristic) for faster convergence and over purely greedy approaches (which sacrifice optimality) (Russell and Norvig, 2020; Thrun, Burgard and Fox, 2005). Obstacle checks via Environment.is_obstacle() keep the planner decoupled from map representation. Polymorphism is explicit: Navigation depends on a PathPlanner Strategy; AStarPlanner is the default and GreedyPlanner an alternative. Tests inject planners and stubs to demonstrate substitutability without modifying Robot (DIP/OCP). Case-insensitive object lookup improves CLI ergonomics, at negligible overhead, matching HRI expectations (Ackerman, 2023; Mulko, 2023).

Error handling covers low battery, no path, and grasp failure with stable operator messages (e.g., "ERROR: No path to target"), mirrored in activity/sequence diagrams and verified by robot/tests/test_* suites. Determinism is supported by narrow interfaces (is_obstacle, sense) and by isolating randomness in tests through stubs/fixed inputs (Derezińska, 2015; Kang, Lo and Lawall, 2019). Linting and coverage provide continuous feedback on quality.

Critical analysis. A* assumes a mostly static map; in dynamic settings D* Lite re-plans efficiently but increases complexity and test surface (Thrun, Burgard and Fox, 2005). In-memory Lists/Stacks/Queues are lightweight and perfectly adequate for the assignment, but lack persistence and durability guarantees (Sommerville, 2015). The CLI yields determinism and simplicity compared with a ROS-based GUI, at the cost of interactivity and richer visual feedback (Ackerman, 2023). An event-driven supervisor could overlap sensing and motion to reduce latency. For explore-space growth or continuous change, PRM/RRT families become relevant for manipulation, trading optimality for probabilistic completeness (Russell and Norvig, 2020).

For concrete **traceability**, the PlantUML sources live under uml/ and map 1-to-1 to the entries in the traceability matrix. I attached them as Appendix A to this PDF report.

In summary, a guarded state machine, compositional boundaries, purposeful data structures, and a polymorphic planning Strategy deliver a system that is safe, testable, and extensible, directly addressing the assessment objectives and reflecting industry practice (Mukherjee et al., 2022; Ackerman, 2023; Russell and Norvig, 2020).

UML Artefacts (Appendix A)

- **Figure 1 — Class Diagram.** Structure (Robot orchestrates Navigation/Manipulator/Communicator/Environment/Memory) and key attributes/ops; matches `robot_system.py`.
  `assets/img/class_diagram.png`

- **Figure 2 — Activity (Navigate).** Guards for OFF, low battery, no-path/timeout; loop over `nextStep`; maps to `Robot.tick(navigate)`.
  `assets/img/activity_navigate.png`

- **Figure 3 — Activity (Pick).** Sense→find object→plan→pick→undo on failure; maps to `Robot.tick(pick)` and `Manipulator`.
  `assets/img/activity_pick.png`

- **Figure 4 — Activity (Errors).** Low battery, no path, grasp fail; recovery and notifications.
  `assets/img/activity_errors.png`

- **Figure 5 — Sequence.** CLI→Robot→Environment/Navigation/Manipulator/Memory; alt blocks for errors; mirrors tests.
  `assets/img/sequence_diagram.png`

- **Figure 6 — State Diagram.** OFF <> IDLE, IDLE→MOVING/MANIPULATING/COMMUNICATING, CHARGING, ERROR; “tick” recovery if battery >= 10%.
  `assets/img/state_diagram.png`

Traceability mini-matrix

Use-case	**Sequence**	**Activity/State**	**Code evidence (examples)**
Navigate to (x,y)	uml/sequence_diagram.puml	Activity “navigation”; State IDLE→MOVING	`Robot.tick(navigate)`, Navigation
Pick object	uml/sequence_diagram.puml	Activity “pick”; State IDLE→MANIPULATING	`Robot.tick(pick)`, Manipulator
Handle errors	uml/sequence_diagram.puml	Activity “errors”; State	Guards, try/except, auto-dock, auto-charge

Use-case	Sequence	Activity/State	Code evidence (examples)
		ERROR/CHARGIN G	

Unit Mapping

- **Unit 1 (OOP & UML):** Full UML set (Class, Sequence, Activity, State) → `uml/*.puml`, `docs/assets/img/*.png`.
- **Unit 2 (Requirements & Use-cases):** Use-case diagrams and CLI-interacties → `docs/assets/img/use_case_diagram.png`, `robot/robot_system.py`.
- **Unit 3 (Abstraction/Encapsulation):** Composition-based `Robot → (Navigation/Manipulator/Communicator/MemoryStore)` → `robot/robot_system.py`.
- **Unit 4 (Design Principles: SRP, DIP/OCP):** Planner as Strategy, injectable in tests → `robot/tests/test_polymorphism.py`.
- **Unit 5 (Polymorphism):** `PathPlanner` interface with `AStarPlanner` and `GreedyPlanner` → `robot/tests/test_polymorphism.py`.
- **Unit 6 (Algorithms & Search):** A* with Euclidische heuristic; obstacle API → `robot/robot_system.py` (Navigation).
- **Unit 7 (Debugging / Error Handling / DS & Search):** Try/except paths (planning/parse), guards (`battery<10%`), data structures (List/Dict/Stack/Queue), linters and tests (~95% coverage) → `robot/robot_system.py`, `robot/tests/*.py`.

Evidence of Independent Working

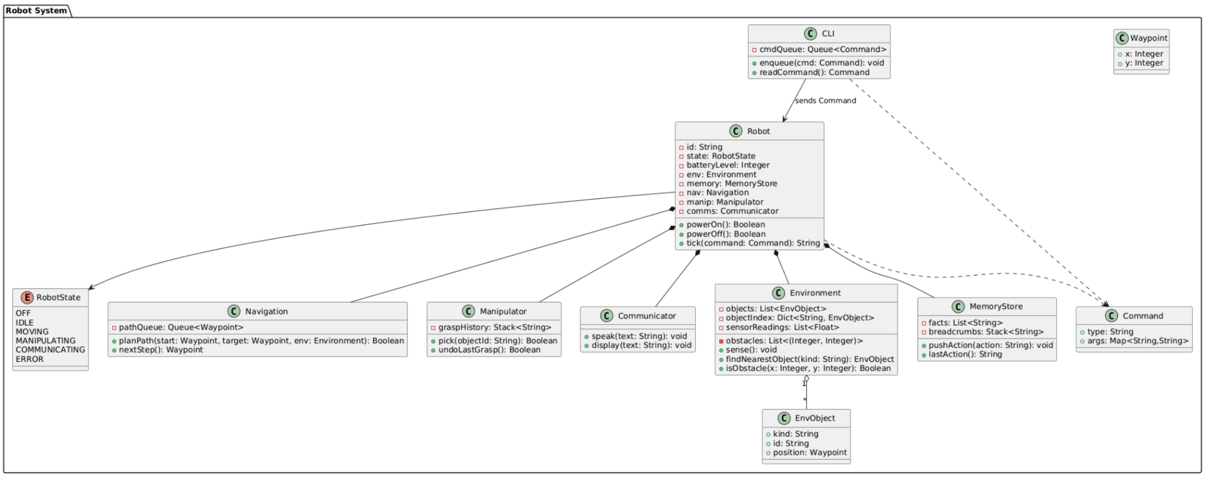
Local Git history shows modelling, implementation, and polish phases; unit tests and coverage reports were produced by the author.

Result summary (latest run): 38 passed, **~95% coverage** (robot package).
Repro steps: create venv → `pip install -r requirements.txt` → `flake8` → `pytest -q --cov=robot --cov-report=term-missing` → `python -m robot`.

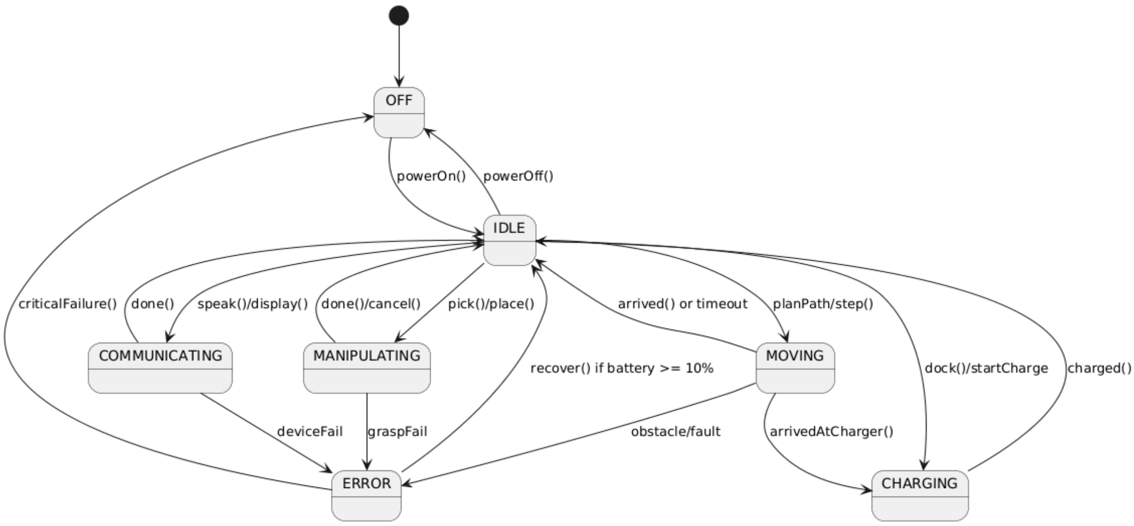
References (Harvard)

- Ackerman, E. (2023) ‘Humanoid Robots Are Getting to Work’, *IEEE Spectrum*. Available at: https://spectrum.ieee.org/humanoid-robots (Accessed: 11 September 2025).
- Derezińska, A. (2015) ‘Improving mutation testing process of Python programs’, in *Software Engineering Techniques in Progress*, pp. 233–246.
- Kang, H.J., Lo, D. and Lawall, J. (2019) ‘BugsInPy: A database of existing bugs in Python programs to enable controlled testing and debugging studies’, in *Proceedings of the 33rd European Conference on Object-Oriented Programming*, pp. 1–6. doi:10.4230/LIPIcs.ECOOP.2019.1.
- Mukherjee, D. et al. (2022) ‘A Survey of Robot Learning Strategies for Human-Robot Collaboration in Industrial Settings’, *Robotics and Computer-Integrated Manufacturing*, 73, p. 102231. doi:10.1016/j.rcim.2021.102231.
- Mulko, M. (2023) ‘5 of the World’s Most Realistic Humanoid Robots Ever’, *Interesting Engineering*. Available at: https://interestingengineering.com/innovation/humanoid-robots (Accessed: 11 September 2025).
- Rumbaugh, J., Jacobson, I. and Booch, G. (2005) *The Unified Modeling Language Reference Manual*. 2nd edn. Addison-Wesley.
- Russell, S. and Norvig, P. (2020) *Artificial Intelligence: A Modern Approach*. 4th edn. Pearson.
- Siciliano, B. and Khatib, O. (eds) (2016) *Springer Handbook of Robotics*. 2nd edn. Springer.
- Sommerville, I. (2015) Software Engineering. 10th edn. Pearson.
- Thrun, S., Burgard, W. and Fox, D. (2005) *Probabilistic Robotics*. MIT Press.

Appendix A

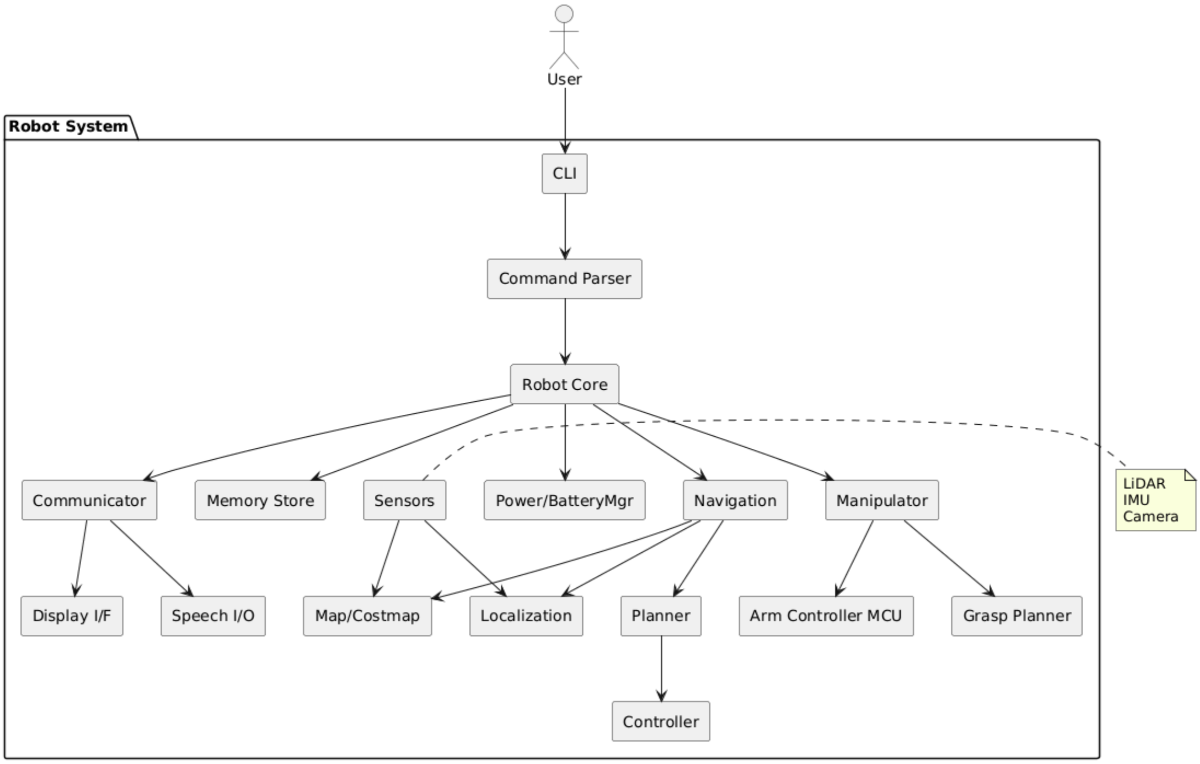


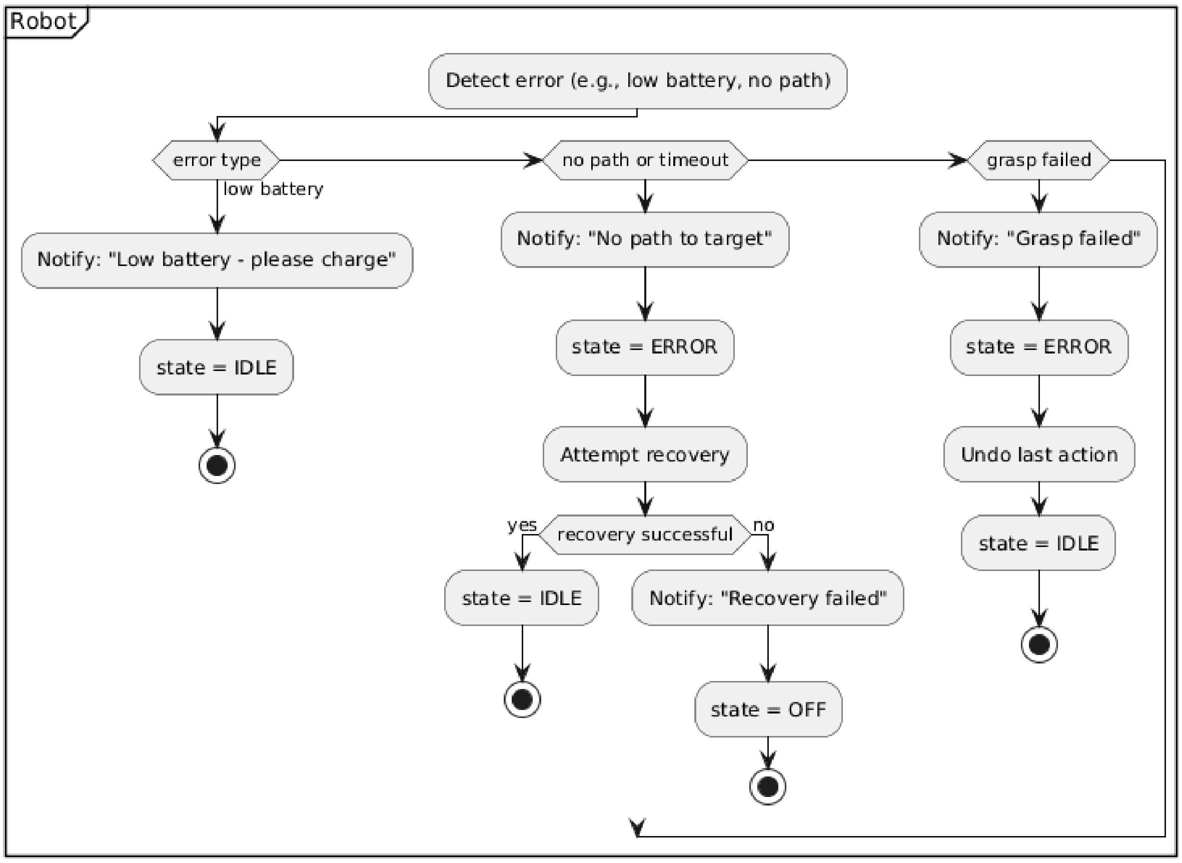
Robot System
CLI
cmdQueue: Queue<Command>
enqueue(cmd: Command): void
readCommand(): Command
Waypoint
x: Integer
y: Integer
sends Command
Robot
id: String
state: RobotState
batteryLevel: Integer
env: Environment
memory: MemoryStore
nav: Navigation
manip: Manipulator
comms: Communicator
powerOn(): Boolean
powerOff(): Boolean
tick(command: Command): String
RobotState
OFF
IDLE
MOVING
MANIPULATING
COMMUNICATING
ERROR
Navigation
pathQueue: Queue<Waypoint>
planPath(start: Waypoint, target: Waypoint, env: Environment): Boolean
nextStep(): Waypoint
Manipulator
graspHistory: Stack<String>
pick(objectId: String): Boolean
undoLastGrasp(): Boolean
Communicator
speak(text: String): void
display(text: String): void
Environment
objects: List<EnvObject>
objectIndex: Dict<String, EnvObject>
sensorReadings: List<Float>
obstacles: List<(Integer, Integer)>
sense(): void
findNearestObject(kind: String): EnvObject
isObstacle(x: Integer, y: Integer): Boolean
MemoryStore
facts: List<String>
breadcrumbs: Stack<String>
pushAction(action: String): void
lastAction(): String
Command
type: String
args: Map<String,String>
1
*
EnvObject
kind: String
id: String
position: Waypoint

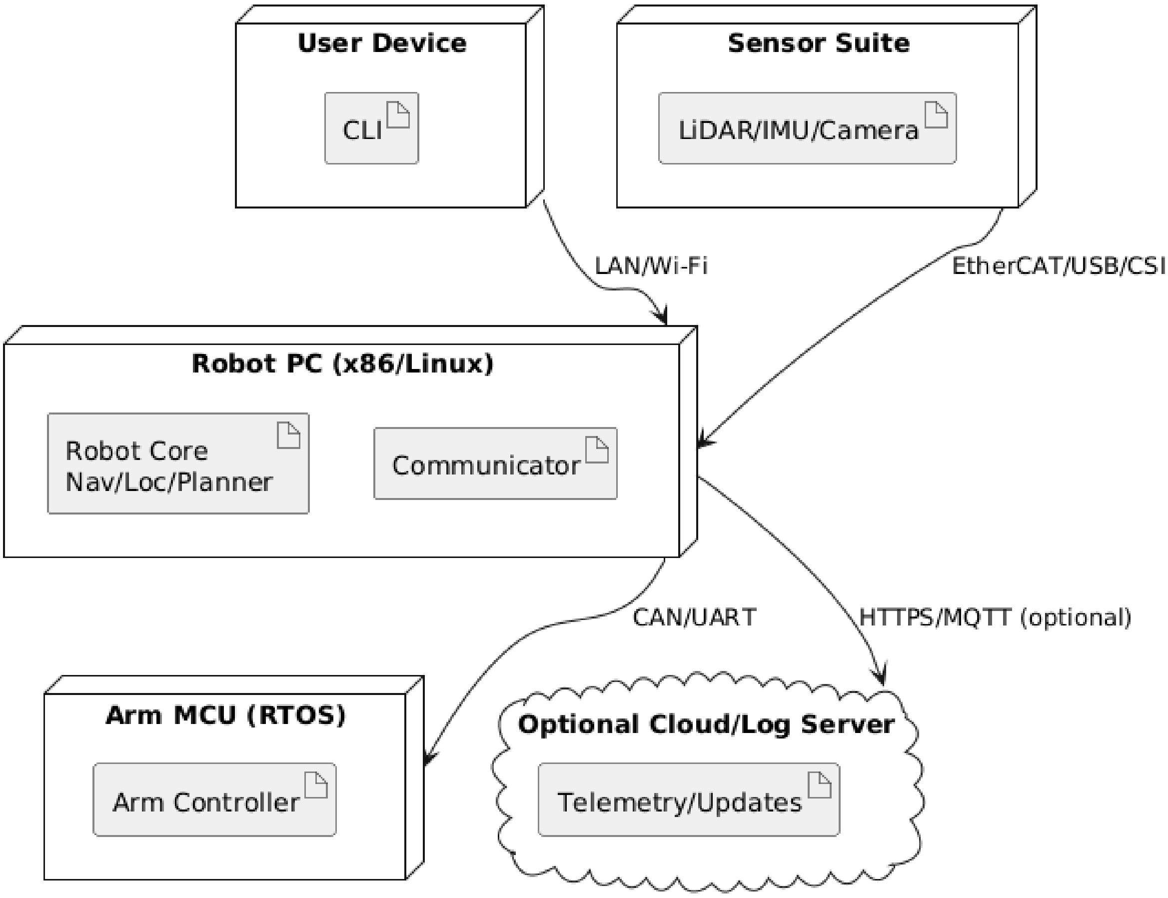


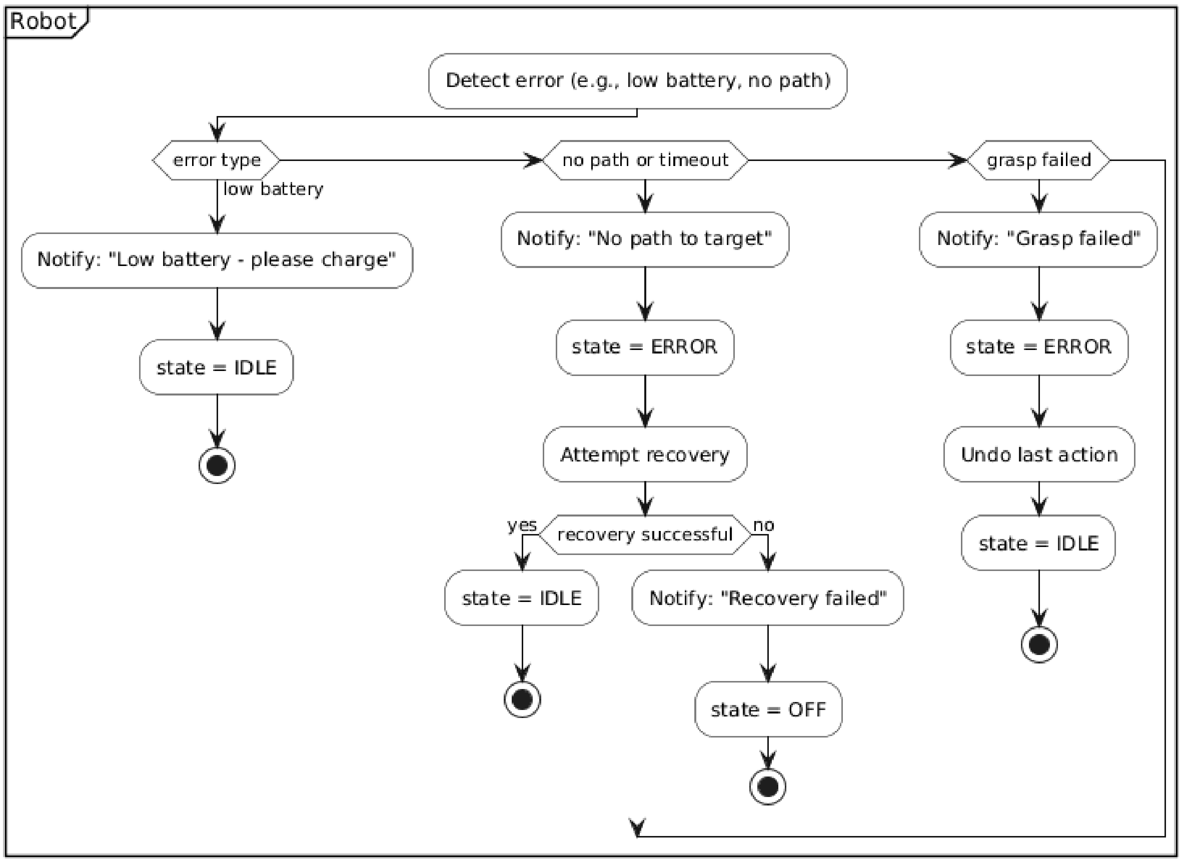
OFF
powerOn()
powerOff()
IDLE
criticalFailure()
done()
speak()/display()
done()/cancel()
pick()/place()
arrived() or timeout
planPath/step()
COMMUNICATING
MANIPULATING
MOVING
recover() if battery >= 10%
dock()/startCharge
charged()
deviceFail
graspFail
obstacle/fault
arrivedAtCharger()
ERROR
CHARGING

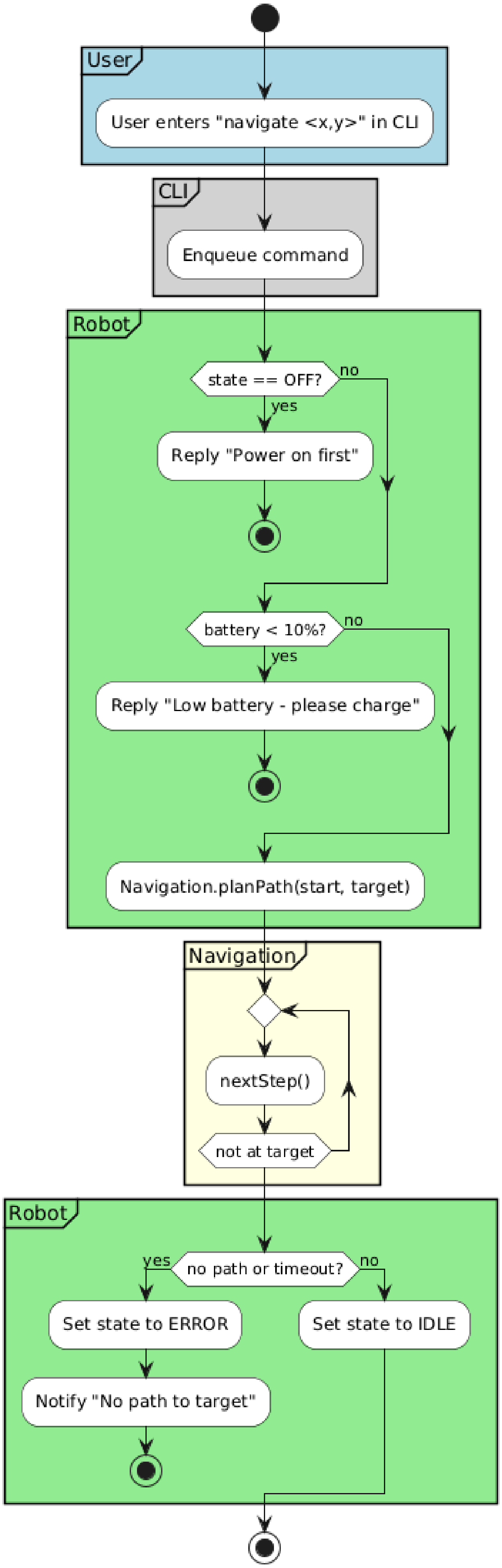
Components - Robot System



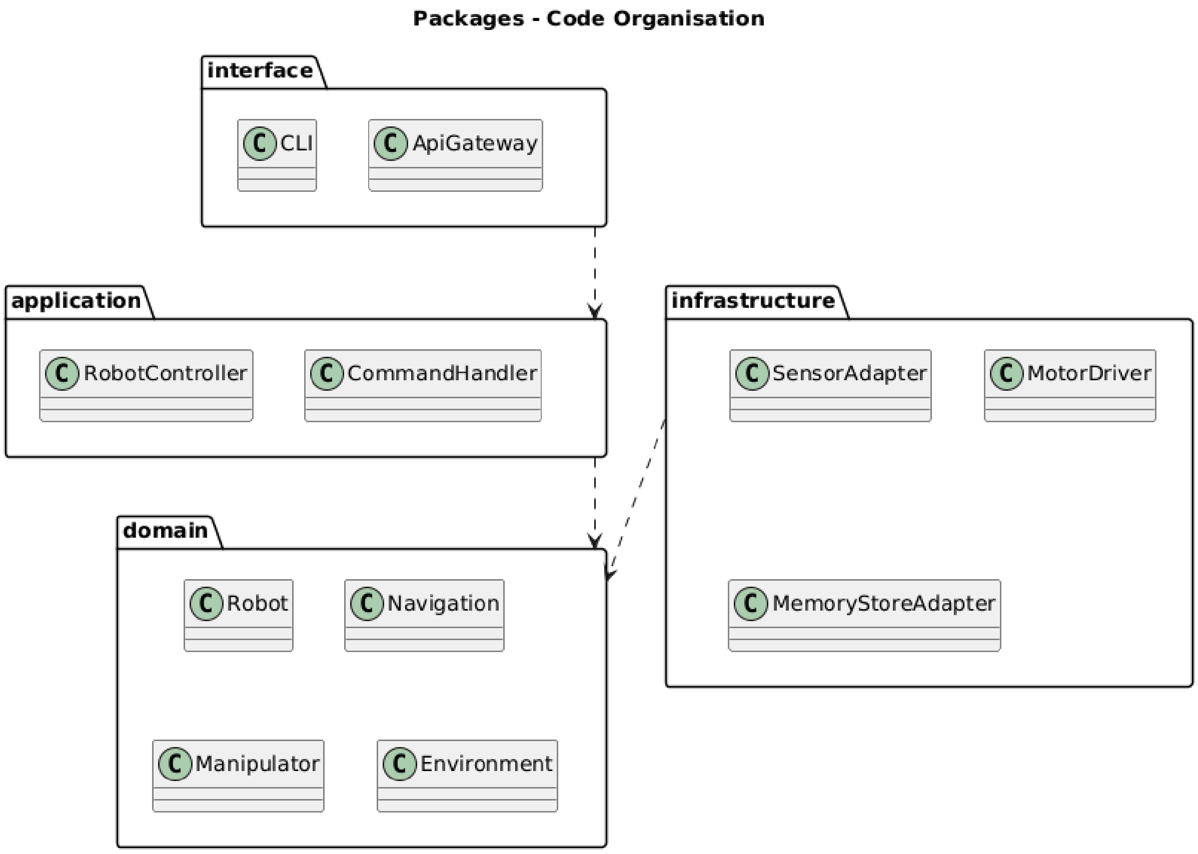


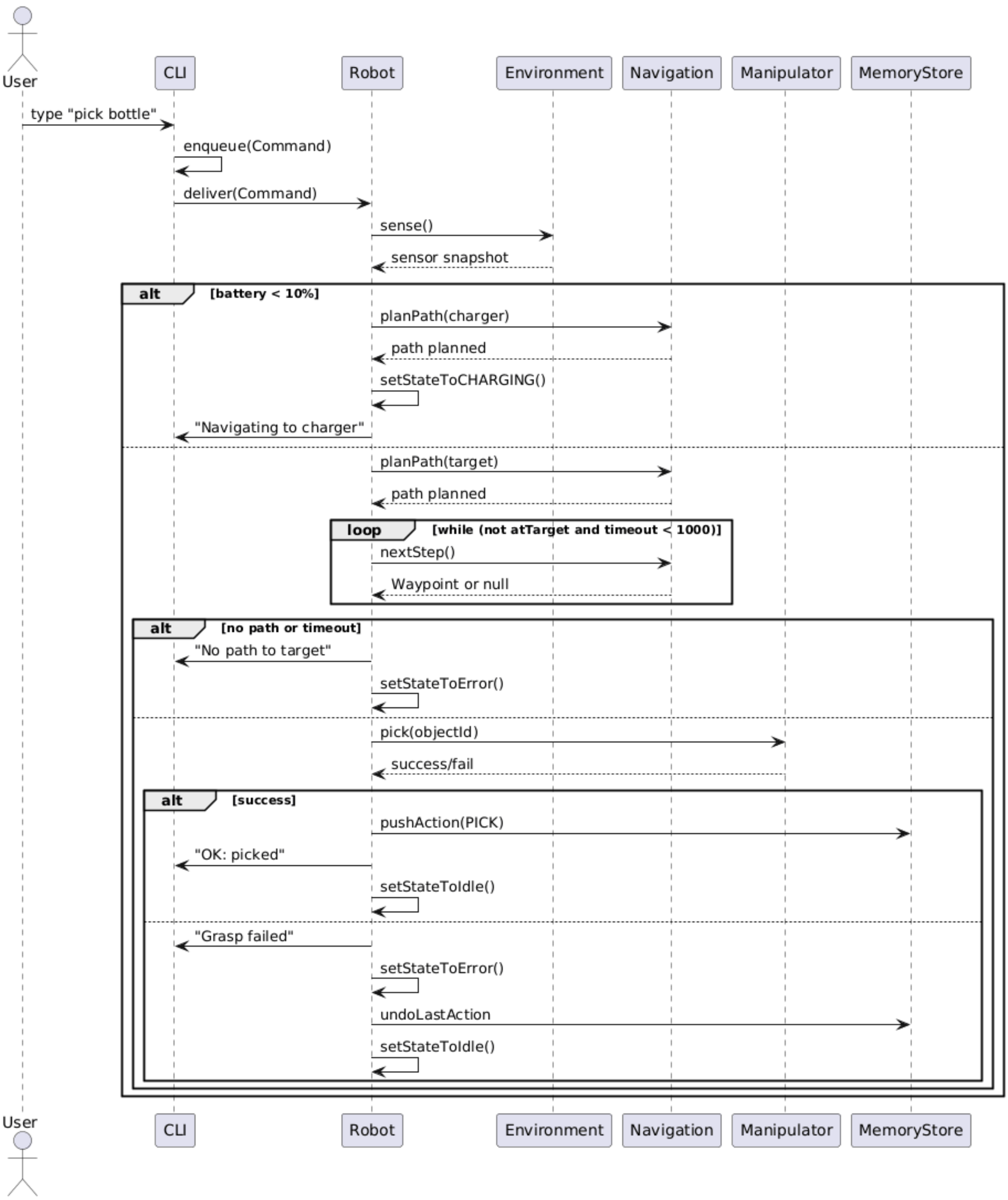

User Device
CLI
Sensor Suite
LiDAR/IMU/Camera
LAN/Wi-Fi
EtherCAT/USB/CSI
Robot PC (x86/Linux)
Robot Core
Nav/Loc/Planner
Communicator
CAN/UART
HTTPS/MQTT (optional)
Arm MCU (RTOS)
Arm Controller
Optional Cloud/Log Server
Telemetry/Updates

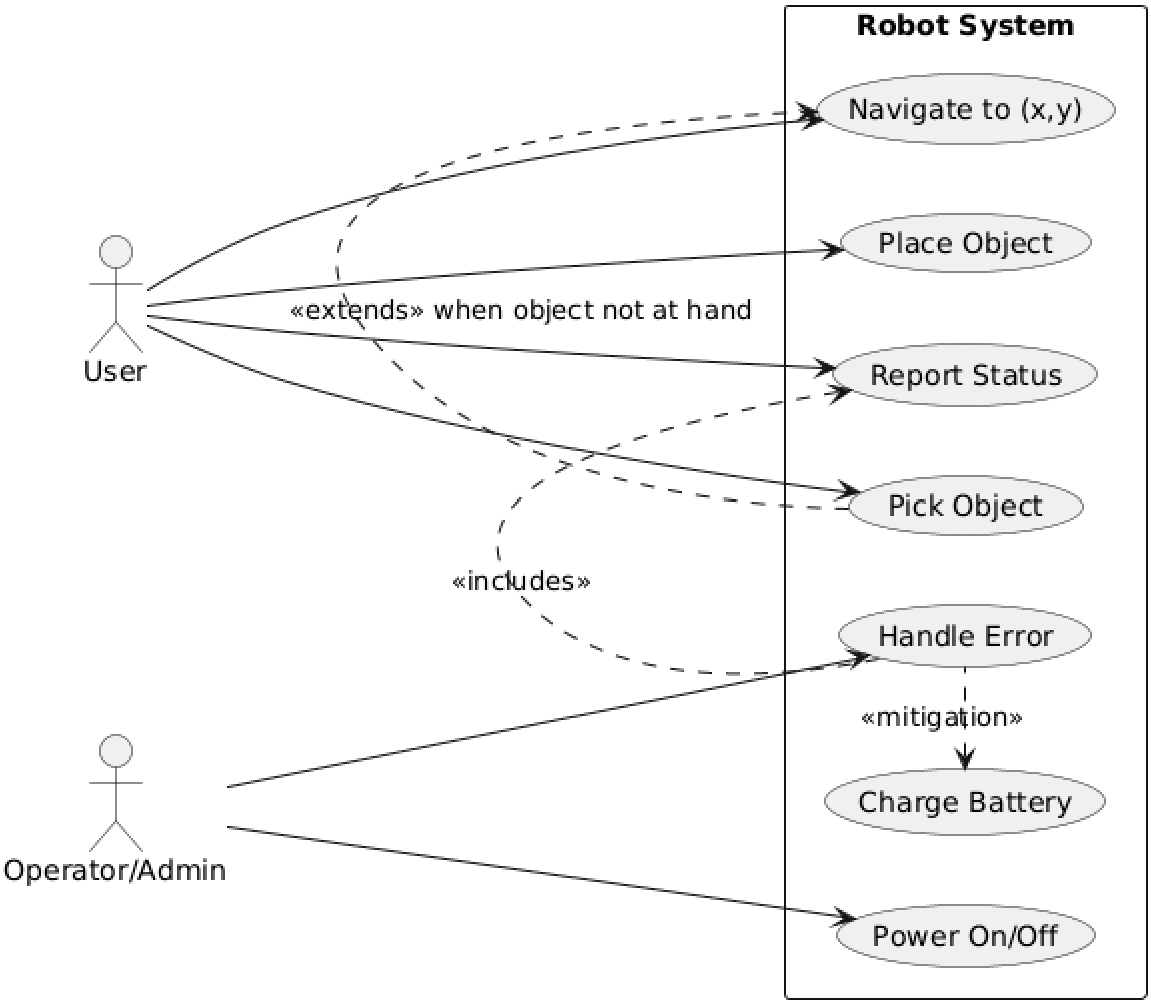

Robot
Detect error (e.g., low battery, no path)
error type
low battery
Notify: "Low battery - please charge"
state = IDLE
no path or timeout
Notify: "No path to target"
state = ERROR
Attempt recovery
recovery successful
yes
no
state = IDLE
Notify: "Recovery failed"
state = OFF
grasp failed
Notify: "Grasp failed"
state = ERROR
Undo last action
state = IDLE



User
User enters "navigate <x,y>" in CLI
CLI
Enqueue command
Robot
state == OFF?
no
yes
Reply "Power on first"
battery < 10%?
no
yes
Reply "Low battery - please charge"
Navigation.planPath(start, target)
Navigation
nextStep()
not at target
Robot
yes
no path or timeout?
no
Set state to ERROR
Set state to IDLE
Notify "No path to target"


Packages - Code Organisation
interface
CLI
ApiGateway
application
RobotController
CommandHandler
infrastructure
SensorAdapter
MotorDriver
MemoryStoreAdapter
domain
Robot
Navigation
Manipulator
Environment


User
CLI
Robot
Environment
Navigation
Manipulator
MemoryStore
type "pick bottle"
enqueue(Command)
deliver(Command)
sense()
sensor snapshot
alt
[battery < 10%]
planPath(charger)
path planned
setStateToCHARGING()
"Navigating to charger"
planPath(target)
path planned
loop
[while (not atTarget and timeout < 1000)]
nextStep()
Waypoint or null
alt
[no path or timeout]
"No path to target"
setStateToError()
pick(objectId)
success/fail
alt
[success]
pushAction(PICK)
"OK: picked"
setStateToIdle()
"Grasp failed"
setStateToError()
undoLastAction
setStateToIdle()
User
CLI
Robot
Environment
Navigation
Manipulator
MemoryStore


Robot System
Navigate to (x,y)
Place Object
«extends» when object not at hand
Report Status
Pick Object
«includes»
Handle Error
«mitigation»
Charge Battery
Power On/Off
User
Operator/Admin