UNIVERSITÀ DEGLI STUDI DELL'AQUILA

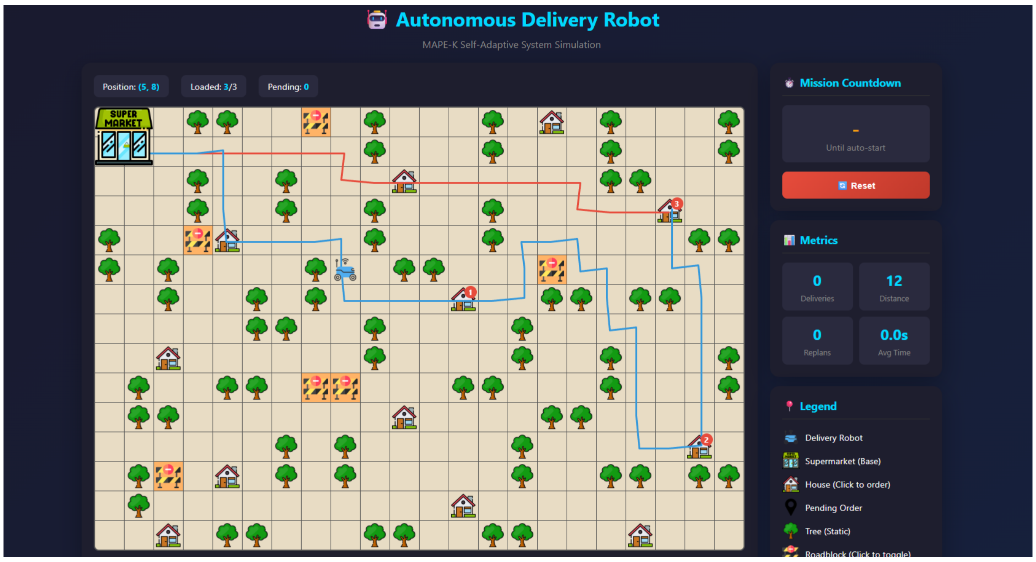
DISIM
Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Department of Information Engineering,
Computer Science and Mathematics

University of L'Aquila, Italy

Final Report

Autonomous Delivery Robot



Course:
Software Engineering for Autonomous Systems (SE4AS)

Supervisor:
Professor Davide Di Ruscio

Members:
Mete Harun Akcay
Thanh Phuc Tran
Pragati Manandhar

February 2, 2026

Contents

1 Introduction

This report presents an Autonomous Delivery Robot system, a self-adaptive application built using the MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) architecture. The system simulates a delivery robot operating in a grid environment where it picks up orders from a supermarket and delivers them to houses while adapting to obstacles that may appear during operation.

Users interact with the system through a web interface where they can click on houses to create delivery orders and click on empty cells to place or remove roadblocks. The robot then autonomously plans optimal routes, executes deliveries, and replans when obstacles block its path.

2 System Goals

The autonomous delivery robot aims to achieve the following goals:

1. Deliver packages from the supermarket to destination houses using optimal routes
2. Detect and respond to dynamic obstacles by automatically replanning alternative routes
3. Autonomously decide when to start missions based on order capacity (3 orders) or timeout (30 seconds)
4. Handle stuck states and resume operation when paths become available
5. Provide real-time visual feedback including planned paths, delivery sequence, and metrics

3 Constraints and Optimization

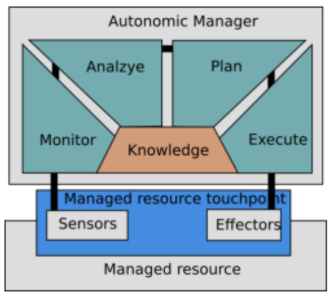
Constraints	
Capacity	Robot can carry maximum 3 orders simultaneously
Movement	Only four cardinal directions (up, down, left, right), no diagonal movement
Obstacles	Cannot traverse cells with trees (static) or roadblocks (dynamic)
Base	Must return to supermarket after completing all deliveries
Boundary	Must stay within the 22×15 grid
Optimization	
Path	A* algorithm guarantees shortest path between any two points
Tiebreaker	When routes have equal distance, prefer delivering closer houses first

4 Managed Resources, Sensors and Effectors

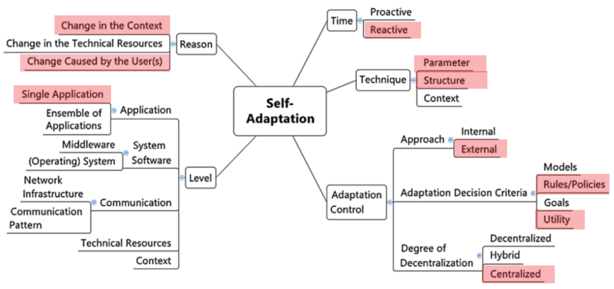
The managed resources are the Robot (responsible for movement and deliveries) and the Grid Environment (the 22×15 map containing obstacles, houses, and the supermarket base).

Resource	Sensors	Effectors
Robot	Position (row, col), loaded orders count, current delivery sequence, distance to waypoint	move(direction), load_order(), deliver_order(), wait()
Grid	Obstacle positions, delivery locations, base position, path validity	add_obstacle(pos), remove_obstacle(pos), is_valid(pos)

5 Architectural Pattern



(a) MAPE-K architecture



(b) Chosen self-adaptation architectural pattern

Figure 1: Overview of the MAPE-K model and the selected self-adaptation architectural pattern

The system uses a fully centralized MAPE-K architecture where a single autonomic manager governs the entire system through a shared knowledge base. It is a single application with one robot operating in one environment. The implementation follows an external approach, meaning the MAPE-K loop runs as a separate control layer outside the managed resources.

Adaptation is reactive, triggered by changes in the environment (dynamic obstacles appearing or disappearing) and changes caused by users (adding orders or placing roadblocks). The system uses both parameter adaptation (adjusting plan index and delivery sequence) and structure adaptation (replanning entire paths when blocked). Decision making combines rule-based criteria in the Analyze component (e.g., if path blocked then replan) with utility-based optimization in the Plan component (comparing route permutations by total distance to select the best one).

6 Adaptation Goals

Goal	**Description**	**Evaluation Metric**
Optimal Routing	Find shortest delivery route visiting all destinations	Total distance, sum of delivery times
Obstacle Avoidance	Detect and avoid obstacles by replanning	Successful replans, zero collisions
Timely Delivery	Deliver packages within reasonable time	Average delivery time
Mission Efficiency	Start missions at appropriate times	Orders per mission, capacity utilization
Stuck Recovery	Recover when path becomes available	Time in stuck state, successful recoveries

7 Decision Function Approach

The autonomic manager uses a hybrid approach combining rule-based and search-based decision making.

The Analyze component follows predefined rules such as: if path is blocked then replan; if robot at delivery location then deliver; if pending orders ≥ 3 or timeout reached then start mission; if no valid path exists then wait.

The Plan component uses the A* algorithm for pathfinding and evaluates all permutations to find optimal delivery sequences. Since the capacity of the robot is three orders, there are $3! = 6$ combinations, which does not effect the decision time significantly. However, the system is made suitable for the future such that if robot's model capacity is increased to 5 or more, the Plan component applies nearest-neighbor heuristic instead of brute-force option in order not to waste time with thousands of computations; meaning the robot chooses to always go to the closest unvisited house.

8 Requirements

Functional Requirements	
FR1	Users can create delivery orders by clicking on houses
FR2	Mission starts automatically when 3 orders pending or after 30 seconds
FR3	Robot navigates from base to all destinations and returns
FR4	System calculates and displays optimal delivery sequence
FR5	System replans path when obstacles block current route
FR6	Users can add/remove roadblocks during operation
FR7	Planned path displayed with color coding (blue: delivery, red: return)
FR8	System tracks metrics (deliveries, distance, replans, average time)
FR9	System detects and indicates stuck states
FR10	Reset function restarts the simulation
Non-Functional Requirements	
NFR1	UI updates in real-time with latency <100ms
NFR2	User clicks processed within 50ms
NFR3	System handles edge cases gracefully
NFR4	Web interface is intuitive and requires no training
NFR5	System runs in any Docker-supporting environment
NFR6	Modular design with clear MAPE-K separation

9 Technologies Used

Python 3.11 – Main programming language used for designing all MAPE-K components, managed resources, and the web server backend.

Flask – Lightweight web framework serving the application and handling HTTP routes for the user interface.

Socket.IO – Enables real-time bidirectional communication between server and client for live state updates and user interactions.

HTML / CSS / JavaScript – Frontend technologies for building the interactive web interface with grid visualization and controls.

Docker – Containerization platform ensuring consistent deployment across different environments.

10 MAPE-K Components

Component	**Responsibilities**	**Output**
knowledge.py	Pure data storage: maintains order queues (pending, loaded, completed), tracks current plan and execution index, stores delivery sequence and robot position, manages mission state and timing, tracks metrics.	Provides shared state access to all other components
monitor.py	Reads from environment sensors (robot position, obstacles), detects dynamic obstacle changes, identifies path blockages, writes sensor data to knowledge	Sensor data dictionary with robot position, obstacles, orders, path validity, mission state
analyze.py	Reads from knowledge, determines if mission should start, detects if replanning needed, identifies delivery opportunities, detects mission completion, identifies stuck states	Analysis dictionary specifying required action (start_mission, replan, deliver, continue, end_mission, wait)
plan.py	Reads from knowledge, uses grid_world for A* pathfinding, calculates optimal delivery sequences, creates replanning paths, handles stuck state recovery	Plan dictionary with action and parameters (paths, sequences, orders)
execute.py	Commands robot through effectors (move, load, deliver), updates knowledge after actions, manages mission lifecycle	Executes actions: continue, start_mission, replan, deliver, end_mission, wait

11 Managed Resources Implementation

Resource	**Attributes**	**Methods**
robot.py	Position (row, column), maximum capacity (3), loaded orders list	move(pos): update position; load_order(order): add to loaded list; deliver_order(id): remove and return order; get_position(): return current position
grid_world.py	Dimensions (22×15), cell types (empty, obstacle, base, delivery), static obstacles (trees), dynamic obstacles (roadblocks), delivery locations (houses), base (2×2 supermarket)	is_valid_position(pos): check traversability; add_dynamic_obstacle(pos): place roadblock; remove_dynamic_obstacle(pos): remove roadblock; get_neighbors(pos): return adjacent valid cells

12 Conclusion

This project demonstrates a working self-adaptive autonomous delivery robot using the MAPE-K architecture. The system successfully delivers packages while adapting to dynamic obstacles through automatic path replanning. We implemented all MAPE-K components with clear separation, used A* and permutation-based optimization for routing, and built a real-time web interface with Socket.IO. The centralized architecture worked well for our single-robot simulation, and Docker containerization made deployment straightforward. Possible future work could include supporting multiple robots.