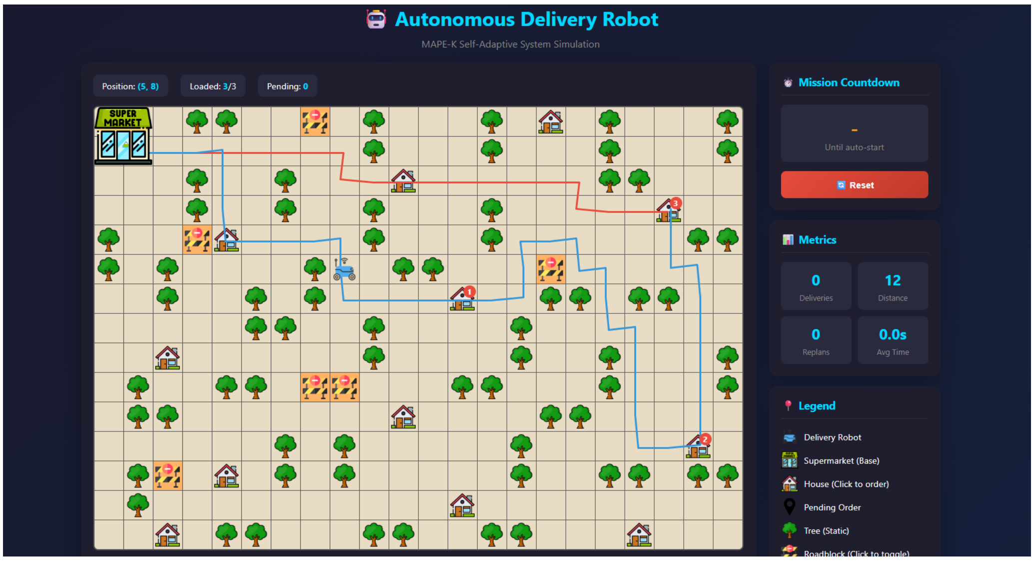
Department of Information Engineering,
Computer Science and Mathematics

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 7, 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.

The system is implemented as a microservices architecture where each MAPE-K component runs as an independent containerized service. Services communicate exclusively via MQTT publish/subscribe messaging, enabling complete decoupling and runtime replaceability of individual components without affecting others.

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

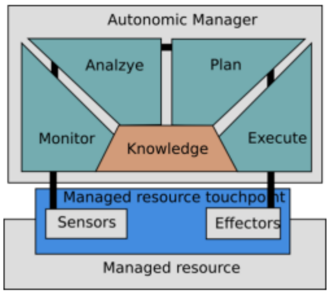
Constraints	
Capacity	Robot can carry maximum 3 orders simultaneously
Movement	Only four cardinal directions (up, down, left, right), no diagonal movement
Obstacles	Cannot traverse cells with trees (static) or roadblocks (dynamic)
Base	Must return to supermarket after completing all deliveries
Boundary	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

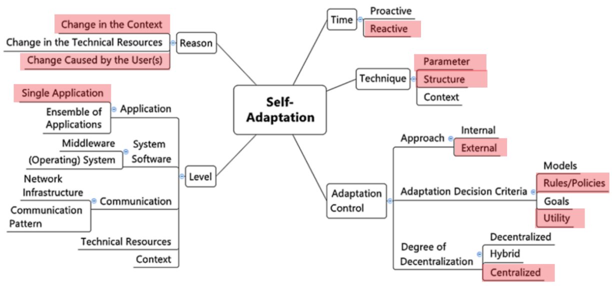
The managed resources are the Robot (responsible for movement and deliveries) and the Grid Environment (the 22×15 map containing obstacles, houses, and the supermarket base). Both are managed through the Environment service which listens to MQTT topics for commands.

Resource	**Sensors (Published topics)**	**Effectors (Subscribed topics)**
Robot	environment/update: position, loaded orders, capacity, is_at_base	environment/move_robot, environment/load_order, environment/deliver_order, environment/clear_orders
Grid	environment/update: obstacles, delivery locations, base position, dimensions	user/toggle_obstacle

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. The implementation follows an external approach, meaning the MAPE-K loop runs as a separate control layer outside the managed resources.

Each MAPE-K component is deployed as an independent microservice in its own Docker container, communicating exclusively via MQTT publish/subscribe messaging. The MAPE-K loop flows through topic chains:

$$\text{Monitor} \xrightarrow{\text{mape/monitor/result}} \text{Analyze} \xrightarrow{\text{mape/analyze/result}} \text{Plan} \xrightarrow{\text{mape/plan/result}} \text{Execute}$$

Each service subscribes to its input topic, processes the message, and publishes to its output topic. This design enables complete decoupling – any component can be stopped, modified, and restarted without affecting other services, as they simply continue listening for messages on their subscribed topics.

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 microservices design with runtime-replaceable MAPE-K components via MQTT

9 Technologies Used

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

Flask & Socket.IO – Web framework serving the user interface with real-time bidirectional communication for live state updates.

MQTT (Mosquitto) – Lightweight publish/subscribe messaging protocol enabling decoupled communication between all MAPE-K services. Each service connects to the broker, subscribes to input topics, and publishes to output topics.

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

Docker & Docker Compose – Containerization platform with multi-container orchestration. Each MAPE-K component runs in its own container, enabling independent deployment and runtime replacement.

10 Microservices Architecture

The system is composed of 8 independent Docker containers that communicate exclusively via MQTT publish/subscribe messaging through the Mosquitto broker.

Service	Description
mqtt (Mosquitto)	MQTT message broker enabling publish/subscribe communication between all services
knowledge	Maintains system state in memory, publishes state updates to knowledge/update topic
environment	Manages Grid and Robot state, publishes updates to environment/update topic
monitor	Subscribes to state topics, publishes monitoring results to mape/monitor/result
analyze	Subscribes to mape/monitor/result, publishes analysis to mape/analyze/result
plan	Subscribes to mape/analyze/result, publishes plans to mape/plan/result
execute	Subscribes to mape/plan/result, commands effectors via environment topics
web	Serves UI, triggers MAPE-K loop by publishing to mape/monitor/request

10.1 MQTT Topic Structure

Topic	Purpose
system/init, system/reset	System initialization and reset commands
user/add_order	User creates a new delivery order
user/toggle_obstacle	User adds/removes a roadblock
mape/monitor/request	Triggers monitoring cycle
mape/monitor/result	Monitor publishes sensor data and conditions
mape/analyze/result	Analyze publishes adaptation decision
mape/plan/result	Plan publishes action with path/sequence
knowledge/update	Knowledge publishes current state
knowledge/set	Execute updates knowledge state
environment/update	Environment publishes grid and robot state
environment/move_robot	Command to move robot
environment/load_order	Command to load order onto robot
environment/deliver_order	Command to deliver order

This architecture enables true runtime replacement of any component. Each service simply connects to the MQTT broker, subscribes to its input topics, and waits for messages. If a service is stopped and restarted (even with modified code), it automatically reconnects and resumes operation without affecting other services.

11 MAPE-K Components

Service	Responsibilities	MQTT Topics
knowledge	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.	Subscribes: system/init, system/reset, user/add_order, knowledge/set. Publishes: knowledge/update
monitor	Reads from environment sensors (robot position, obstacles), detects dynamic obstacle changes, identifies path blockages, checks mission conditions	Subscribes: knowledge/update, environment/update, mape/monitor/request. Publishes: mape/monitor/result
analyze	Determines if mission should start, detects if replanning needed, identifies delivery opportunities, detects mission completion, identifies stuck states	Subscribes: mape/monitor/result. Publishes: mape/analyze/result
plan	Uses A* pathfinding, calculates optimal delivery sequences, creates replanning paths, handles stuck state recovery	Subscribes: mape/analyze/result. Publishes: mape/plan/result
execute	Commands robot through effectors (move, load, deliver), updates knowledge after actions, manages mission lifecycle	Subscribes: mape/plan/result, knowledge/update, environment/update. Publishes: knowledge/set, environment commands

12 Managed Resources Implementation

The Environment service manages both Robot and Grid resources, listening to MQTT command topics and publishing state updates.

Resource	State (in-memory)	MQTT Interface
Robot	Position (row, column), maximum capacity (3), loaded orders list, is_at_base flag	Commands: environment/move_robot, environment/load_order, environment/deliver_order, environment/clear_orders
Grid	Dimensions (22×15), cell types (empty, obstacle, base, delivery), static obstacles (trees), dynamic obstacles (roadblocks), delivery locations (houses), base (2×2 supermarket)	Commands: user/toggle_obstacle. State published to: environment/update

13 Conclusion

This project demonstrates a working self-adaptive autonomous delivery robot using the MAPE-K architecture implemented as a microservices system with MQTT-based communication. The system successfully delivers packages while adapting to dynamic obstacles through automatic path replanning.

Each MAPE-K component runs as an independent containerized service communicating exclusively via MQTT publish/subscribe messaging through the Mosquitto broker. This architecture enables true runtime replacement of individual components – any service can be stopped, modified, and restarted without affecting the rest of the system, as services simply reconnect to the broker and resume listening for messages.

We implemented A* and permutation-based optimization for routing, and built a real-time web interface with Socket.IO. Docker Compose orchestrates all 8 containers, making deployment straightforward across different environments. Possible future work could include supporting multiple robots with distributed MAPE-K loops.