Courier Robot – Design Document

1. General Project Description

The project involves designing and simulating an autonomous mobile robot in a structured indoor environment. The robot acts as a courier, capable of transporting items such as medicines or packages while navigating on a static 2D grid (0 = free cell, 1 = occupied cell). Pathfinding is performed using a Breadth-First Search (BFS) algorithm, which guarantees the shortest viable route on a grid and ensures predictable, cell-by-cell navigation. The robot identifies targets via AprilTags, executes pick-up using preset arm poses, and returns to base, all implemented in ROS 2 Jazzy and tested in Gazebo 11 LTS.

2. Choice of the Simulated Robot Model

2.1 Body and Locomotion

- Mobile base with **differential drive** or tracks
- Compact frame suitable for indoor environments
- Stability adequate for **cell-by-cell movement**

2.2 Sensors Set

- **Camera** for AprilTag detection
- **IMU** for orientation and rotation correction
- **Encoders** for odometry
- **Distance sensor** for immediate obstacle detection

2.3 Actuators Set

- Differential drive motors

- Robotic arm with **2–3 DOF**
- Gripper or claw for object pick-up

2.4 Body Shape & Movable Parts

- Body: rectangular compact chassis
- Movable parts:
  - Arm joints (2–3 DOF)
  - Gripper/claw
  - Differential drive wheels/tracks

3. Simulation Environment

- **Gazebo 11 LTS + ROS 2 Jazzy**
- Official plugins for sensors, motors, and AprilTags
- Ready-made models for mobile manipulators
- Realistic simulation transferable to real robot

ROS 2 Jazzy was chosen because it is a stable LTS release that offers reliable, modular, and real-time-capable communication between sensing, control, and actuation nodes, making the system easily transferable from simulation to the real robot. Gazebo 11 LTS complements it by providing a realistic physics environment, accurate sensor simulation (camera, IMU, encoders), and native ROS 2 integration through official plugins. Together, they allow developing, testing, and validating navigation, perception, and manipulation using the exact same architecture used on the physical robot.

4. Robot Goals

4.1 Main Goal

- Enable the robot to autonomously deliver objects from a starting cell (A) to a target cell (B) and return, maintaining accuracy and safety.

4.2 Sub-Goals

- Navigate on a **static 2D grid** using BFS path planning
- Detect and realign using AprilTags and onboard sensors
- Pick up objects via **preset arm poses:**
- Return safely to the base and release the object

We perform the pick-up via preset poses:

The arm executes a sequence of preset poses:

  - **POSE_HOME** – resting position
  - **POSE_ABOVE_OBJECT** – above object
  - **POSE_GRIP** – grasp object
  - **POSE_CARRY** – carry object
  - **POSE_RELEASE** – release object
- Gripper closes/opens at corresponding presets
- Works in both simulation and real robot, with calibration

5. Robot Agent Architecture

5.1 Hierarchical Layers

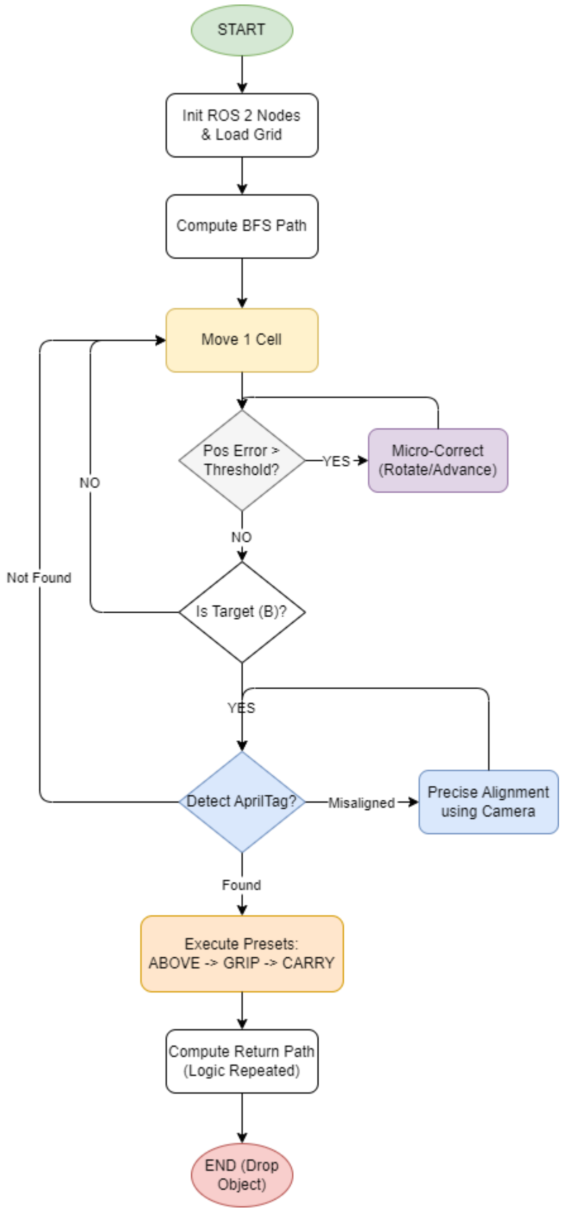
1. **Sensing Layer** – reads sensors (camera, IMU, encoders, distance sensor)
2. **Control Layer** – manages grid, executes BFS path planning, error correction, and mission supervision
3. **Actuation Layer** – controls motors, arm, and gripper

5.2 Internal Description

- The sensing layer provides real-time data on robot position, orientation, and obstacle detection.
- The control layer decides movements, triggers error correction, and supervises mission phases.
- The actuation layer executes locomotion and arm movements according to commands from control.

5.3 Decision Tree

Illustrates robot logic for cell movement, error correction, obstacle handling, pick-up, and return.



6. Software Architecture

6.1 ROS 2 Jazzy Structure

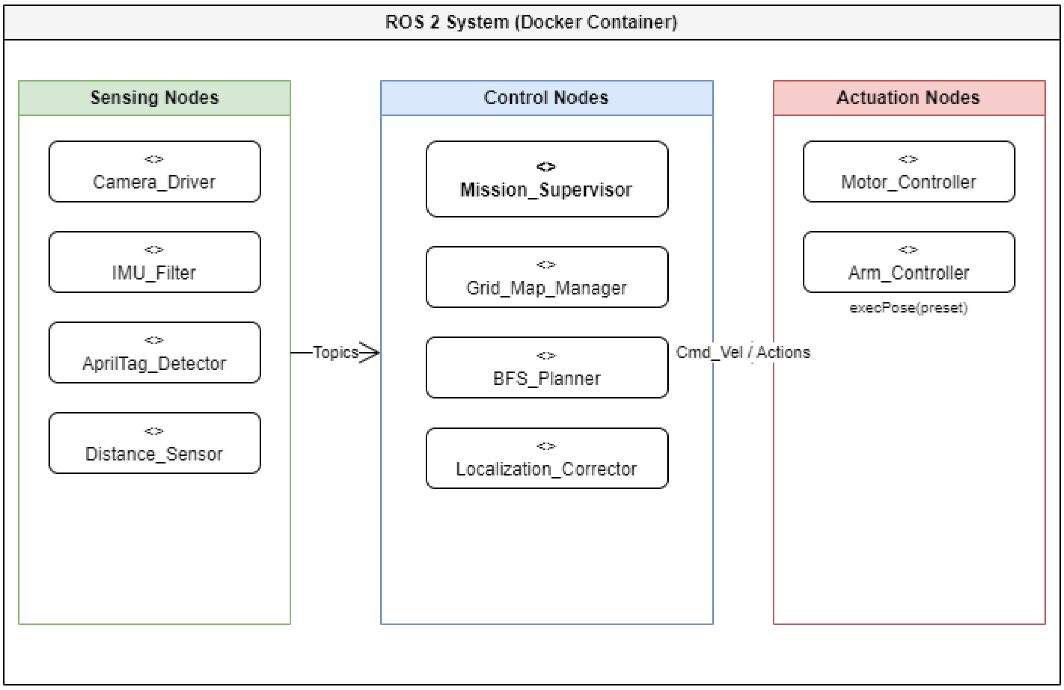
- **Sensing nodes**: read real or simulated sensors
- **Control nodes**: manage grid, BFS path planning, error correction, mission supervision
- **Actuation nodes**: control motors, arm, and gripper
- Communication via ROS 2 topics, services, and actions
- Centralized logging for post-mission analysis

- **ROS 2 Graph (rqt_graph, optional)**: visualize active nodes and topic connections for debugging

6.2 Modular Design

- Logic is independent from drivers (path planning, error handling, arm preset sequence)
- Drivers interface with simulator or real robot
- Same logic code runs in simulation and on real robot

Software Architecture (ROS 2 Jazzy Nodes)



7. Main Functional Requirements

- Cell-by-cell navigation using BFS
- Position error handling and realignment using AprilTags
- Pick-up via robotic arm preset poses
- Return to base and final recognition via AprilTag
- Centralized logging and operator error communication

8. Non-Functional Requirements

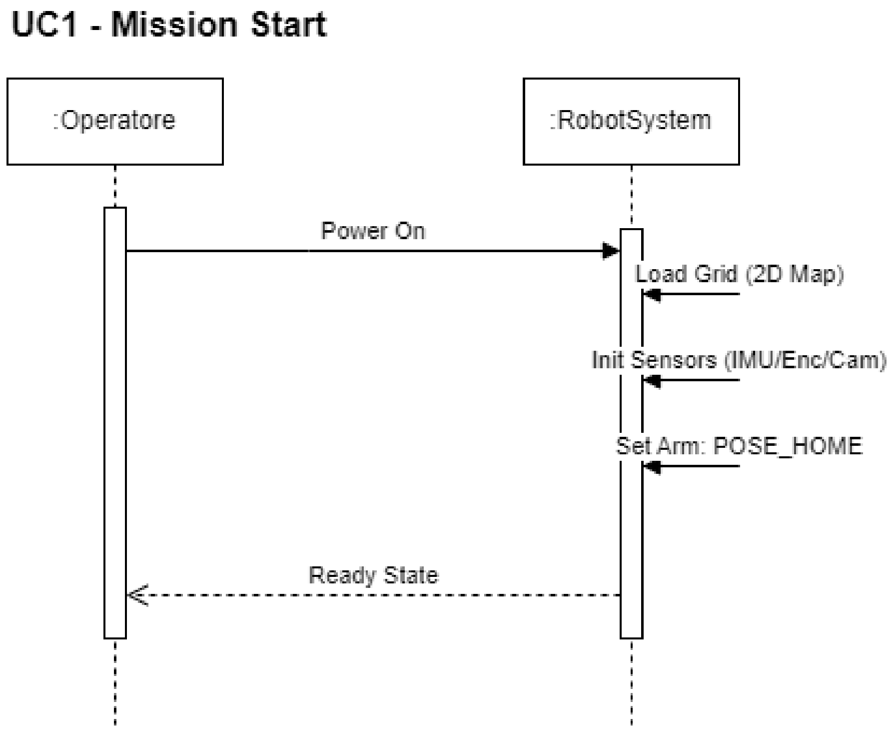
- Position accuracy <10% of cell size
- Error correction and realignment <200 ms
- Modularity: clear separation between logic and drivers
- Portability: same logic for simulation and real robot
- Safety: immediate stop on unexpected obstacles
- **Implementation Requirements**:
    - Dockerized multi-process deployment
    - Message broker system for node communication
    - Centralized logging system
    - GUI (RViz) for sensor and robot state visualization in real-time or simulated time
    - Shared software repository (MS Teams code folder)
    - ReadMe.md instructions for installation and testing

9. Use Cases

UC1 – Mission Start

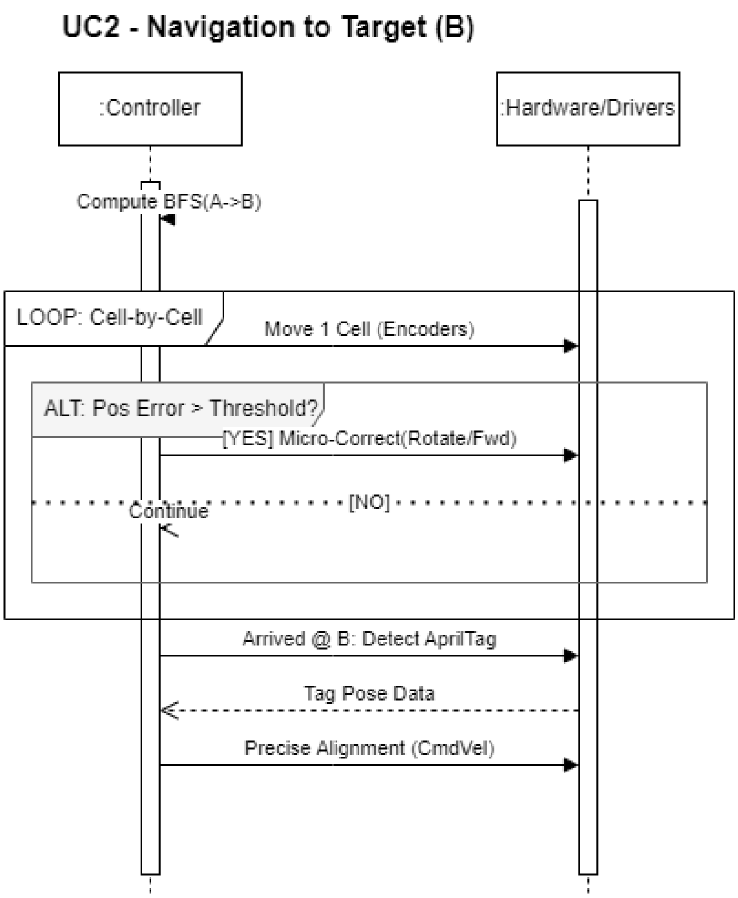
- **Primary Actor:** Operator
- **Goal:** Start robot and initialize sensors and actuators
- **Procedure:**
    - Power on robot or simulation
    - Load static 2D grid
    - Initialize sensors and arm/gripper in POSE_HOME
    - Perform self-diagnostic

- **Post-condition:** Robot ready to start mission



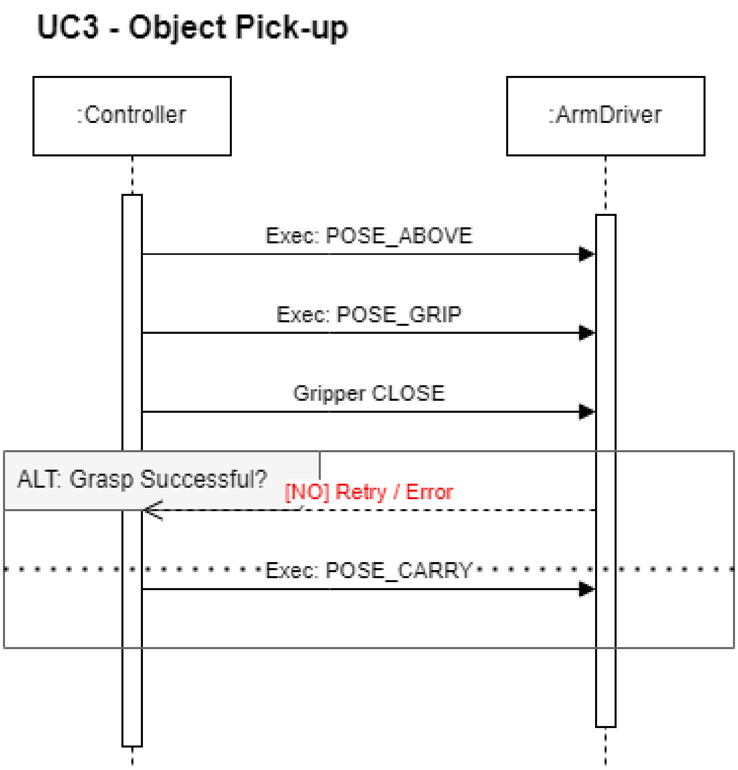
UC2 – Navigation to Target

- **Primary Actor:** Robot
- **Goal:** Reach target cell
- **Procedure:**
  - Read BFS path
  - Move cell-by-cell
  - After each cell, check position relative to cell center and correct errors
  - Detect AprilTag for final alignment
- **Post-condition:** Robot aligned above target



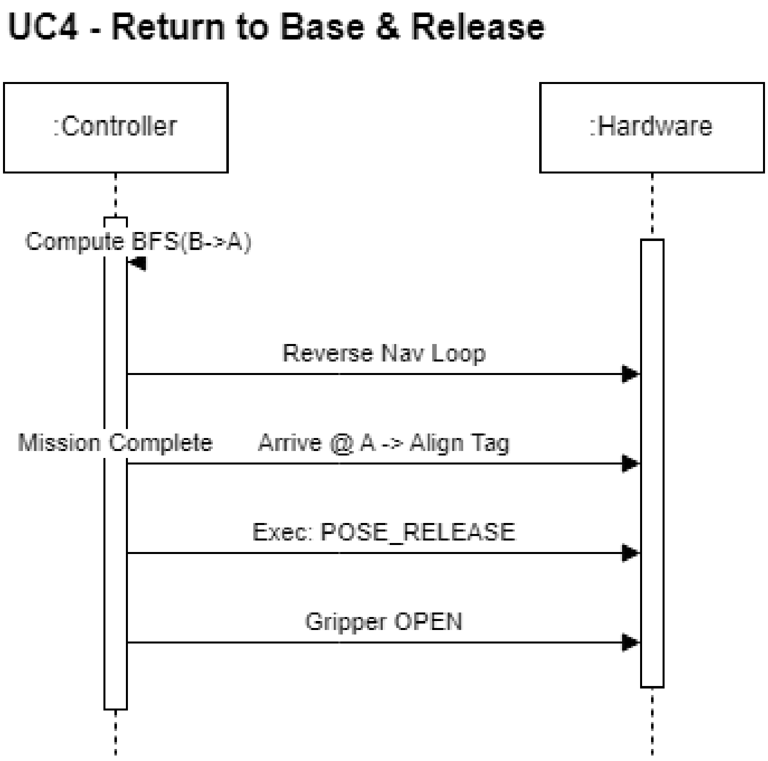
UC3 – Object Pick-up

- **Primary Actor:** Robot (arm)
- **Goal:** Grasp object using preset poses
- **Procedure:**
  - Execute preset sequence: POSE_ABOVE_OBJECT → POSE_GRIP → POSE_CARRY
  - Gripper closes on object
  - Sensor confirmation of secure grasp
- **Post-condition:** Object held, ready for transport



UC4 – Return to Base

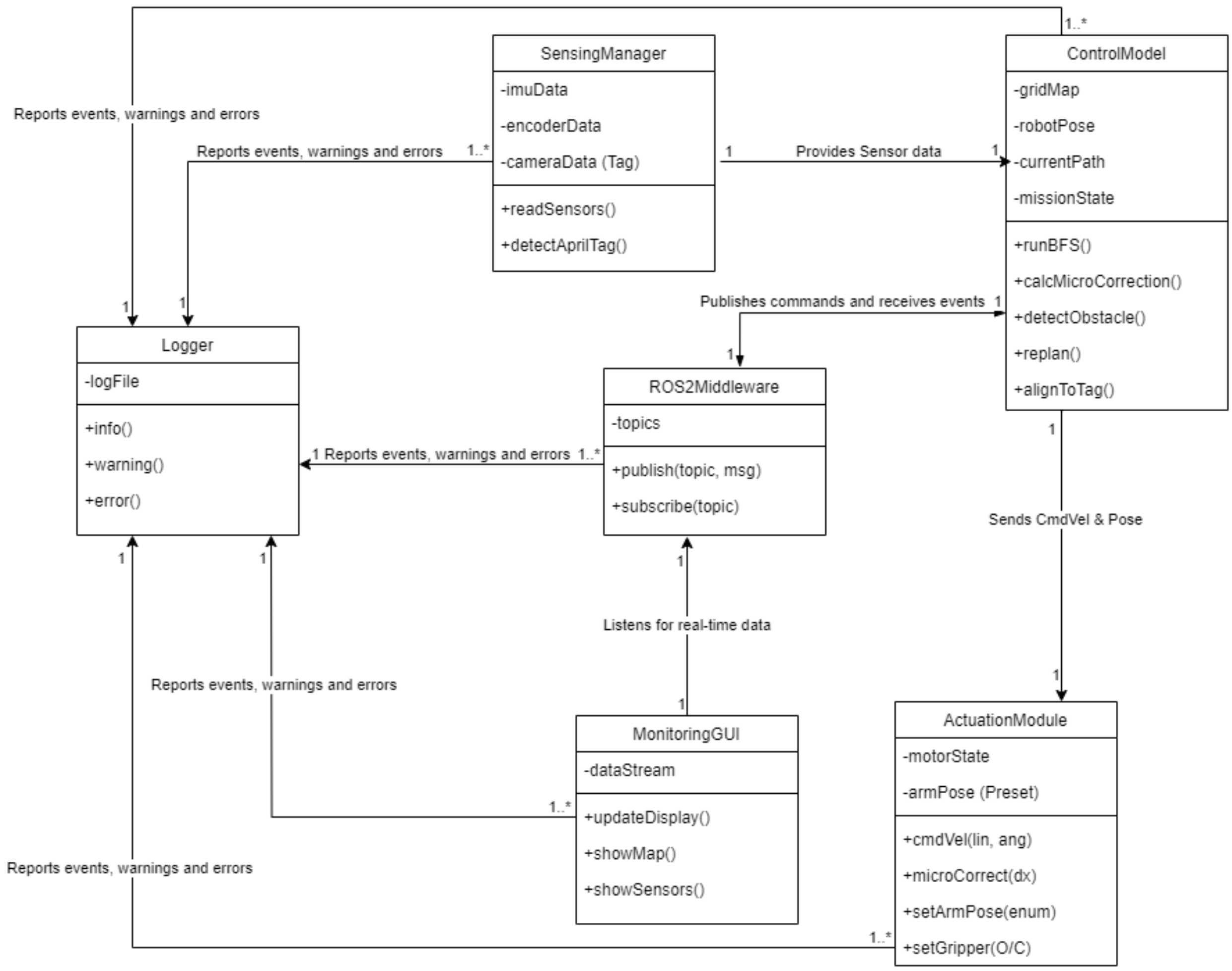
- **Primary Actor:** Robot
- **Goal:** Return to starting cell
- **Procedure:**
  - Compute reverse BFS path (B → A)
  - Move cell-by-cell, correcting position errors
  - Align with base AprilTag
  - Execute POSE_RELEASE and open gripper to release object
- **Post-condition:** Object delivered, robot at starting position



12. UML Diagrams

- Class diagrams for robot modules (Sensing, Control, Actuation)
- Activity diagram for mission flow (navigate → pick-up → return)

Class diagram


SensingManager
-imuData
-encoderData
-cameraData (Tag)
+readSensors()
+detectAprilTag()
ControlModel
-gridMap
-robotPose
-currentPath
-missionState
+runBFS()
+calcMicroCorrection()
+detectObstacle()
+replan()
+alignToTag()
Logger
-logFile
+info()
+warning()
+error()
ROS2Middleware
-topics
+publish(topic, msg)
+subscribe(topic)
MonitoringGUI
-dataStream
+updateDisplay()
+showMap()
+showSensors()
ActuationModule
-motorState
-armPose (Preset)
+cmdVel(lin, ang)
+microCorrect(dx)
+setArmPose(enum)
+setGripper(O/C)
Reports events, warnings and errors
Reports events, warnings and errors
Provides Sensor data
Publishes commands and receives events
Reports events, warnings and errors
Sends CmdVel & Pose
Listens for real-time data
Reports events, warnings and errors
Reports events, warnings and errors
1..*
1
1