

# CS 556: Final Project Description

## Warehouse AMR Autonomy: Mini-Kiva Autonomy Stack.

CS 556 Robotics, Spring 2025

Dr. Mary Pourebadi

### Objective

Design a warehouse AMR autonomy system for a mobile robot resembling a Kiva-style robot (as used by Amazon). In the next four sessions, you will implement a control system to allow your robot to localize within a known warehouse layout (aisles/shelves), start from a staging/charging dock, service three pick locations by navigating to tagged inventory bins, simulate an align/lift/engage gesture, and then return to the dock station to charge and signal completion. Note: The maximum time your robot gets to traverse the maze is 120 seconds.

### What is Warehouse Robotics?

Warehouse robotics are machines that can perform various warehouse tasks, including storing, retrieving, and transporting inventory. They increase efficiency by taking the fastest feasible routes and improve safety by operating in hazardous or high-traffic environments.

**Autonomous Mobile Robots (AMRs)** are the most common types of warehouse robots. They independently navigate the warehouse, utilizing technologies such as machine learning, AI, and sensors to plan and adjust routes in real-time while managing unforeseen challenges (e.g., avoiding crowds/obstacles or evading falling objects).



# 1 Prerequisites

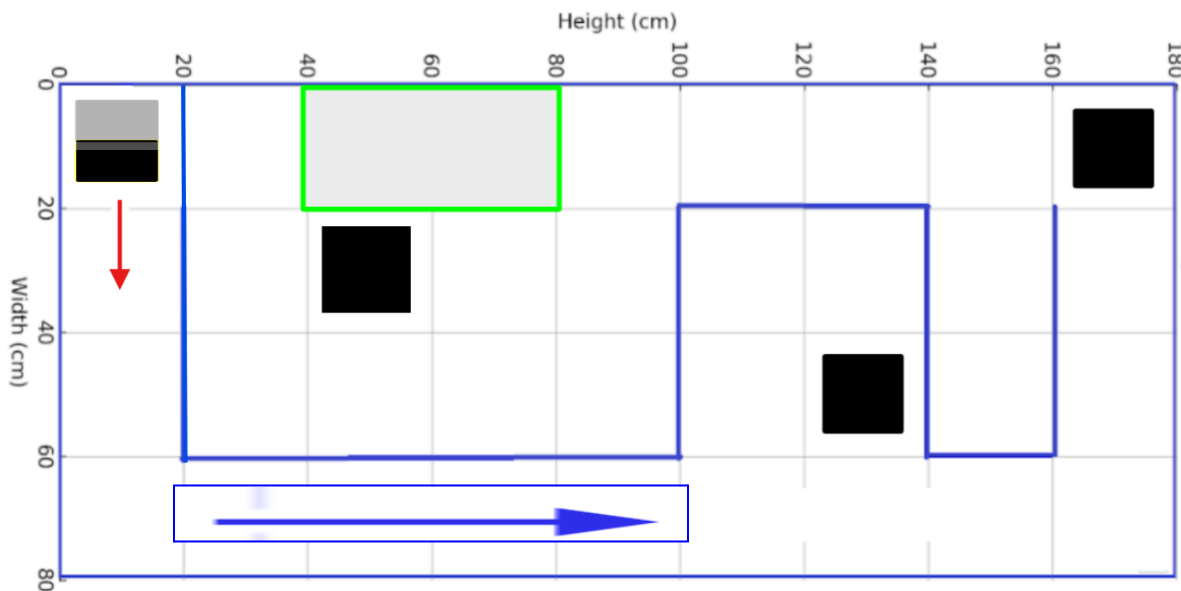
- Review the summary of your labs (CS556-Lecture13-Labs Review)
  - Bring your laptop.
- 

# 2 Project Description:







Each team will program a Pololu robot to autonomously navigate a **dynamic course simulating a warehouse environment**. The course consists of aisle-like corridors, line following segments, obstacles, and shelf (wall) sections.

---

## Overview:



*Warehouse AMR Demo Course — 180 × 80 cm Testbed*

-  = Pallet / Obstacle
-  = Pick Location (Bin)
-  = Shelf / Wall Boundary
-  = Fast Track Aisle
-  = Start / Dock / Calibration area
-  = Exit Direction

- The maze map represents a **warehouse layout** with **aisles and shelving** (treated as fixed walls). Warehouse AMR Demo Course — 180 × 80 cm Testbed. Cell size (e.g., 20×20 cm), world origin (0,0) and robot origin (10x10 cm), axis directions, and walls lie **on** cell borders.
- **Obstacles** are placed in the **operations area (marked in green)** and resemble **pallets, totes, or carts**. The setup will remain the same during demo day.
- The robot starts at a **start/charging dock (marked in black/white)**, and the start/dock exit path is marked with an **arrow**.
- The robot can travel as fast as it wants once it detects the **Fast Track Aisle/path** marked with **blue tape**. The fast-track aisle is marked with blue tape on a light floor. You should still use `readLineWhite()` and treat `lineSensorValues[2]` in the range 200–350 as “on blue tape” after calibration. These numeric ranges are based on tests conducted under lighting in our CS425 classroom for this course setup.
- The primary objective is to **locate and service three pick locations (bins)** placed in the layout, each marked by a **black square** for IR detection.
- **Layout assumption:** The map (size, cells) is known to students and may be pre-coded. Additionally, the placement of obstacles (green area items) is known and fixed. Bin/Pick Location markers will be unknown / not fixed and must be sensed at runtime. The fast-track aisle/path may change between runs and must be sensed at runtime.
- Color/legend consistency with the provided diagram.

## Project Phases:

- **Phase 1: Navigation and Localization** – The robot navigates and localizes throughout the entire known layout.
- **Phase 2: Return to Staging/Charging Dock** – Once all three pick locations (bins) have been collected, the robot will return to the staging and charging dock.
- **Phase 3: Pick Detection & Service** – The robot identifies and services three pick locations (bins).
- **Phase 4: Speed management and safety considerations** – speed may vary (static or dynamic). Operate conservatively to protect the robot and the course.

# Execution Phases

## Phase 1: Navigation and Localization

- For the standalone Phase 1 requirement, your robot must visit every traversable cell (all V). In the full project run, it is acceptable to stop coverage early once the third bin is picked, provided your navigation and localization methods would support full coverage.
- **Coverage**
  - Traverse the entire map and mark each traversable cell as visited.
  - Maintain a visit array (same dimensions as the map):
    - V = visited, N = not visited (initialize all traversable cells to N).
  - Stop when all traversable cells are V.
- **Navigation Method (choose one)**
  - **Wall following (left or right):** Move cell by cell; update the visit array each time you reach a cell center. Continue until all cells are V.
  - **Predefined path:** Execute a hardcoded route that passes through every traversable cell, marking V as you go.
  - Other equivalent methods are acceptable if they achieve full coverage.
- **Record traversed Path**
  - Moreover, during traversal, the robot needs to record its path using **waypoints and movement logs**. Hint: This can be done through various techniques, for example, having a new array for waypoints log and record the order of movements made from the beginning (departing the charging station) until collecting the third bin (for example: L L L L U R ... L), and then traverse this in reverse.
- **Localization**

- Maintain an accurate pose estimate while moving (e.g., using odometry with PID control or a particle filter).
- Use any combination of odometry, PID controllers, particle filters, and/or environmental cues to limit drift.
- **Completion Criteria**
  - All traversable cells are marked **V** (unless the third bin/location is picked).
  - The traversed path is recorded.
  - The robot remained accurately localized throughout the run (using the show method).

## Phase 2: Return to Staging/Charging Dock

- Assuming that your robot completed three pick locations (bins), **return to the staging/charging dock** by either:
  - (is allowed) (Efficient technique) **Reversing the recorded path**. Once all three pick locations (bins) have been serviced, the robot will use the path to return to the dock, **or**
  - (extra credit) (Most efficient technique) Computing a **shortest path** on the known grid with updated obstacle costs, for example, preferred or efficiency, **or**
  - (Partial credit, but allowed) (Inefficient technique) The robot uses the first part of the algorithm from Phase 1, and traverses the entire map until it goes back to the charging dock (regardless of noting when three pick locations (bins) are serviced).
- **Return tolerance:** Docking tolerance:  $|x\_error| \leq 10 \text{ cm}$ ,  $|y\_error| \leq 10 \text{ cm}$ , and heading error  $\leq 30^\circ$ .
- Upon successful return to the docking station, **emit a beep** to signal completion.
- Logging: minimal required log (timestamps from start to end, robot's pose(x, y, theta), and events: bin#i pick-confirmed, docked).

## Phase 3: Pick Detection & Service

- To implement the behavioral logic for Phase 3 & Phase 4, your robot must use **embedded IR sensor readings** directly in its state machine. All conditional logic should follow this structure:
- **Set up State (Robot at Start / Charging Deck)**
  - If you are in setup(), the robot must assume it is at the starting station.
  - → Calibrate all IR sensors (same method used in Labs 8 & 9).
  - After calibration, transition into your Phase 1/2 behavior.
- **Detecting Pick Locations**
  - Continuously monitor the **center IR sensor** while running normal tasks.
  - Condition: Center sensor > BLACK\_THRESHOLD
  - This means the robot is positioned on a **black square pick marker**.
  - When detected, the robot must:
    - **Stop immediately**
    - **Perform a 360° rotation** (brief alignment/display motion)
    - **Increment and display pick count** on the OLED
  - Continue performing Phase 3 actions **until all three pick bins have been serviced**.
  - Then transition back to line monitoring behavior.
- **Note:**
  - To implement the blue line following functionality, you can use the functions and predefined values from BlueLineFollowing\_Starter.ino into your final project's .ino file.
  - The provided predefined calibration values (BLUE\_MIN\_CAL = 200, BLUE\_MAX\_CAL = 500, BLACK\_THRESHOLD = 900) were determined by running multiple tests on the same maze used in the CS425 environment.

## Phase 4: Speed Management and Safety Considerations

- **Detecting Blue Line**
  - Check center sensor value again — if it falls within your calibrated blue range, meaning that condition is met: BLUE\_MIN\_CAL < center sensor < BLUE\_MAX\_CAL, the robot is now on the blue fast-track line, so → Call predefined functions for blue-line navigation.
  - When on the fast track, the robot must:
    - Increase its speed (Base speed may vary within 75–300 (static or dynamic). Operate conservatively to protect the robot and the course. Base speed for the robot must be a minimum of 50.)

- Continuously read the line (adjust steering using IR data)
    - Stay in fast-track mode while the blue line persists.
  - **All Other Cases (Default Behavior):**
    - If the IR reading does **not** match black or blue conditions:
    - → The robot is **off special markers**
      - **Continue normal Phase 1/2 behaviors**
      - (e.g., obstacle handling, standard navigation, or whatever your base logic defines).
  - **Note:**
    - If noise or reflectance variation causes flicker around boundaries, you may want to define **priority** explicitly: For example,
      - If a reading is barely > BLACK\_THRESHOLD, should it trigger Phase 3 even if you were already in Phase 4?
      - Or should Phase 3 only trigger when *not currently in fast-track mode*?
- 

## Suggestions for Improved Performance

- Search/Heuristics: Use A\* or Greedy Best-First Search on the known grid; bias heuristics to minimize aisle changes and prefer the fast-track aisle when clear.
- Path Optimization: Prefer shortest-path returns; treat blocked aisles and temporary obstacles as dynamic costs.
- Speed Adjustment: Slow near shelf faces, intersections, tight corners, and dynamic obstacles. (*Optional: add an E-stop input.*)
- Pick Task Enhancements: Add an LED/buzzer to the pick confirmation; implement fine-alignment (e.g., proportional control to center on a fiducial/marker).
- Testing & Debugging: Test with varying pallet/tote placements and temporary aisle blocks to validate dynamic re-routing. Simulate edge cases to ensure robustness.
- Safety notes: Make a beep sound when backing up (move backward).
- Other improvements as you see fit ...