Abstract

FOODIE is a smart campus food delivery simulation system featuring autonomous robots operating within a dynamic warehouse environment. This project presents the design, simulation, and analysis of an automated warehouse system where multiple robots deliver orders intelligently. The system dynamically generates orders, adapts to warehouse congestion, adjusts robot speeds based on obstacle proximity, and prioritizes customer orders based on waiting times. Multiple pathfinding algorithms were tested (A*, BFS, DFS, Dijkstra) with A* ultimately chosen due to its balance between speed and optimality. The system is fully visualized using Pygame.

1. Introduction

Automated delivery systems are becoming increasingly important for industries like food delivery, warehousing, and logistics. To model such environments, FOODIE simulates multiple autonomous robots within a dynamic, obstacle-filled warehouse. This project aims to develop an efficient order management and robotic navigation system capable of handling real-time challenges like congestion and path blockages while ensuring timely deliveries.

2. System Design

2.1 Map Generation

- The warehouse environment is modeled as a 2D grid.
- Obstacles are randomly placed to simulate racks, walls, or furniture.
- A fixed warehouse hub (`FW_LOCATION`) serves as the central starting and returning point for all robots.

2.2 Robots

Each robot is defined with:

- Unique Robot ID
- Current position on the map
- Queue of assigned orders
- Dynamic speed control based on obstacle proximity
- State flags like busy, waiting, and warehouse status

Robots dynamically adjust their speed:

- **Fast Mode**: Far from obstacles
- **Normal Mode**: Moderately close to obstacles
- **Cautious Mode**: Near obstacles

2.3 Orders

Orders are randomly generated across the map every few seconds, with:

- Unique Order ID
- Location
- Timestamps for creation and delivery
- Status flags: Assigned, Delivered

Rules enforced:

- No duplicate order locations.
- No orders generated on obstacles or the warehouse.
- If the number of active orders exceeds 80% of the available non-obstacle space, new order generation is paused.

2.4 Visualization

Using Pygame:

- **Gray Circles**: New, unassigned orders
- **Red Circles**: Assigned but not yet delivered orders
- **Green Circles**: Recently delivered orders (stay for 2 seconds)
- **Orange Squares**: Robots
- **Blue Square**: Warehouse

The bottom panel displays:

- Total orders generated
- Deliveries per robot
- Average delivery time

3. Algorithms Used

3.1 Pathfinding

Initially, multiple algorithms were explored:

- **A*** (final choice): Balances speed and path optimality.
- **BFS**: Simple and complete but slightly slower in large maps.
- **DFS**: Performed poorly in finding optimal paths.
- **Dijkstra**: Guaranteed shortest path but slower than A*.

The A* algorithm was ultimately selected due to its:

- Fast convergence to optimal paths.
- Efficient use of Euclidean heuristics.
- Compatibility with dynamic environments.

3.2 Intelligent Order Assignment

Orders are assigned based on a scoring function:

- **Score = (Age * 1.5) - Distance to warehouse**
- Older and closer orders have higher priority.
- Robots at the warehouse batch multiple nearby orders (up to 4) in optimized sequences (via permutations).

3.3 Congestion Management

To prevent the map from becoming overcrowded:

- If 80% of non-obstacle tiles have active orders, order generation halts until sufficient deliveries are completed.

3.4 Robot Movement

- Speed adapts dynamically based on obstacle proximity.
- Robots wait for 1.5 seconds at the warehouse before departing again.

4. Data Flow Diagram

1. Random map generation with obstacles and warehouse location.
2. Robot initialization at warehouse.
3. Orders randomly generated.
4. Batch assignment of orders to robots based on priority.
5. Robots deliver orders using A* paths.
6. Delivered orders turn green and disappear after 2 seconds.
7. Robots return to warehouse and await new tasks.

5. Experiments and Results

5.1 Pathfinding Algorithm Performance

Using A Algorithm (500 orders):*

```
==== DELIVERY STATS ====
R1: 142 deliveries, avg time = 19.28 sec
R2: 166 deliveries, avg time = 19.19 sec
R3: 144 deliveries, avg time = 20.06 sec
```

Using DFS Algorithm (500 orders):

```
==== DELIVERY STATS ====
R1: 37 deliveries, avg time = 601.38 sec
R2: 39 deliveries, avg time = 616.69 sec
R3: 34 deliveries, avg time = 707.12 sec
```

Using BFS Algorithm (500 orders):

```
==== DELIVERY STATS ====
R1: 146 deliveries, avg time = 19.86 sec
R2: 148 deliveries, avg time = 20.28 sec
R3: 152 deliveries, avg time = 19.68 sec
```

Using Dijkstra Algorithm (500 orders):

```
==== DELIVERY STATS ====
R1: 152 deliveries, avg time = 20.13 sec
R2: 144 deliveries, avg time = 18.99 sec
R3: 152 deliveries, avg time = 20.47 sec
```

5.2 Observations

- **A*** and **BFS** produced the best balance of speed and completeness.
- **DFS** was inefficient and resulted in extremely high delivery times.
- **Dijkstra** performed similarly to A*, but slower pathfinding made A* preferred.

The adaptive speed system near obstacles slightly increased average delivery time but greatly improved safety.

6. Major Modules

- **simulation.py**: Controls simulation loop, order generation, robot coordination, real-time drawing.
- **robot.py**: Defines robot behavior, movement, dynamic speed handling.
- **pathfinding.py**: Implements pathfinding algorithms (A*, BFS, DFS, Dijkstra).
- **config.py**: Stores global constants like map size, warehouse location.
- **map.py**: Handles map generation with obstacles.
- **data.py**: Defines initial robot starting data.
- **main.py**: Entry point to launch the simulation.

7. How to Run

```
pip install pygame
python main.py
```

8. Conclusion

This project successfully models a realistic smart warehouse environment with autonomous robots handling dynamic deliveries. The combination of intelligent batching, congestion control, dynamic speed adjustment, and efficient A* pathfinding leads to effective and smooth operations. Through experimenting with different algorithms and system parameters, the simulation demonstrates how real-world delivery systems must balance speed, safety, and order fulfillment priorities.The techniques developed could easily extend to larger real-world logistics environments, showcasing the importance of efficient task scheduling, path optimization, and congestion management in robotic systems.