Indoor Robotics Delivery System - Theoretical Design

Executive Summary

Extension of the indoor navigation system to enable autonomous robotic delivery within mall environments. The robot uses a multi-modal sensor fusion approach combining BLE beacons, AprilTags, computer vision, and IMU sensors to achieve robust navigation and obstacle avoidance.

System Architecture Overview

```
┌─────────────────────────────────────────────────────────────┐
│                DELIVERY ROBOT SYSTEM                  │
├─────────────────────────────────────────────────────────────┤
│                              │
│  ┌───────────────────────────────────────────────────────┐  │
│  │      HIGH-LEVEL CONTROL (Raspberry Pi 4/Jetson)      │  │
│  │  - Multi-modal localization fusion                   │  │
│  │  - Computer vision (YOLO object detection)           │  │
│  │  - LLM-based semantic reasoning                      │  │
│  │  - Path planning & navigation (ROS2)                 │  │
│  │  - Delivery task management                          │  │
│  └───────────────────────────────────────────────────────┘  │
│           │ Serial/CAN Bus                   │
│  ┌───────────────────────────────────────────────────────┐  │
│  │     LOW-LEVEL CONTROL (STM32F4/H7 + Embassy)         │  │
│  │  - PID motor control                                 │  │
│  │  - IMU sensor fusion (accelerometer + gyroscope)     │  │
│  │  - Wheel odometry                                    │  │
│  │  - Motor driver PWM generation                       │  │
│  └───────────────────────────────────────────────────────┘  │
│                              │
└─────────────────────────────────────────────────────────────┘
```

Component 1: Multi-Modal Localization System

Localization Strategy (Hierarchical Fusion)

The robot uses a three-tier localization approach with complementary strengths:

Tier 1: BLE Beacons (Coarse Localization)

- **Purpose**: Global position estimation, area detection
- **Accuracy**: ±1-3 meters

- **Update Rate**: 1-2 Hz
- **Method**: RSSI-based trilateration using existing mall beacon infrastructure
- **Advantages**:
  - Always available throughout the mall
  - No line-of-sight requirement
  - Reuses existing navigation infrastructure
- **Limitations**:
  - Low accuracy
  - No orientation information
  - Signal interference from crowds/obstacles

Tier 2: AprilTags (Precise Localization)

- **Purpose**: Precise position and orientation estimation at key locations
- **Accuracy**: ±2-5 centimeters, ±2 degrees
- **Update Rate**: 10-30 Hz (when visible)
- **Method**:
  - Camera detects AprilTag markers placed at strategic locations
  - PnP (Perspective-n-Point) algorithm solves for camera pose
  - Known tag positions in world frame allow robot localization
- **Advantages**:
  - Very high accuracy
  - Provides full 6-DOF pose (position + orientation)
  - Deterministic, no drift
- **Limitations**:
  - Requires line of sight
  - Limited range (5-10 meters depending on tag size)
  - Need strategic placement throughout mall

AprilTag Placement Strategy:

- Near elevators, escalators, stairs (transition points)
- At intersections and corridor junctions
- Near merchant entrances for precise delivery

- Spacing: Every 10-15 meters in main corridors
- Higher density in complex areas (food courts, atriums)

Tier 3: IMU + Wheel Odometry (Dead Reckoning)

- **Purpose**: Smooth position estimation between sensor updates
- **Accuracy**: Degrades over time (drift)
- **Update Rate**: 100-200 Hz
- **Method**:
  - Wheel encoders measure linear velocity
  - IMU (accelerometer + gyroscope) measures angular velocity and acceleration
  - Sensor fusion using Extended Kalman Filter (EKF)
- **Advantages**:
  - High frequency updates
  - Smooth motion estimation
  - Works in all conditions
- **Limitations**:
  - Accumulates error over time
  - Wheel slip causes inaccuracy
  - IMU drift

Sensor Fusion Architecture

```
BLE RSSI → Distance Estimation → ┐
                                 │
AprilTag Detection → PnP Pose → ─┼→ Extended Kalman Filter → Fused Pose
                                 │   (State: x, y, θ, vx, vy, ω)
IMU + Wheel Odom → Integration → ┘
```

Kalman Filter State Vector:

- Position: (x, y) in world frame
- Heading: θ (yaw angle)
- Linear velocity: (vx, vy)
- Angular velocity: ω

Measurement Updates:

- **BLE**: Low-frequency position update with high uncertainty ($\sigma^2 \sim 2m^2$)
- **AprilTag**: High-frequency pose update with low uncertainty ($\sigma^2 \sim 0.01m^2$)
- **IMU/Odom**: Prediction step for propagating state between measurements

Covariance Handling:

- BLE measurements have large covariance → low weight in fusion
- AprilTag measurements have small covariance → high weight, "resets" drift
- Dead reckoning covariance grows over time → compensated by external measurements

---

Component 2: Computer Vision & Perception

Camera System

- **Hardware**: Wide-angle RGB camera (170° FOV) + optional depth camera
- **Mounting**: Front-facing, ~30cm height, slight downward tilt
- **Resolution**: 640x480 or 1280x720 at 30 FPS
- **Purpose**: Obstacle detection, AprilTag detection, scene understanding

YOLO Object Detection

Object Classes for Detection:

1. **People** (pedestrians, customers, staff)
2. **Kiosks** (information booths, promotional stands)
3. **Strollers** (baby carriages)
4. **Wheelchairs** / mobility aids
5. **Shopping carts**
6. **Cleaning equipment** (wet floor signs, cleaning carts)
7. **Merchandise displays** (temporary obstacles)
8. **Doors** (open vs closed state)
9. **Elevators/escalators** (for navigation assistance)
10. **Delivery targets** (customers, pickup points)

YOLO Model Selection:

- YOLOv8n or YOLOv8s for real-time performance on Raspberry Pi/Jetson
- Quantized INT8 model for faster inference

- Custom training on mall environment dataset

Detection Pipeline:

```
Camera Frame → YOLO Detection → Bounding Boxes + Classes + Confidence
                              ↓
                    Filter (confidence > 0.5) → Tracked Objects
```

PnP-based Obstacle Avoidance

PnP (Perspective-n-Point) for 3D Position Estimation:

For each detected obstacle (especially people and kiosks), estimate 3D position:

1. **Bounding Box to 3D Points**:
    - Use bottom center of bounding box as ground plane contact point
    - Assume object height based on class (person ~1.7m, kiosk ~1.2m)
2. **PnP Solving**:
    - Input: 2D image coordinates + assumed 3D structure + camera intrinsics
    - Output: 3D position in camera frame
    - Transform to world frame using robot pose
3. **Obstacle Map Construction**:
    - Convert 3D positions to 2D occupancy grid
    - Mark cells as occupied, unknown, or free
    - Update dynamically as robot moves

Fallback for Unknown Objects:

- If YOLO doesn't recognize object, use geometric depth estimation
- Assume obstacle extends from ground to ~1m height (safe conservative approach)

Depth Estimation Options

Option A: Stereo Camera / Depth Camera

- Intel RealSense D435 or similar
- Direct depth measurement
- More accurate but adds cost

Option B: Monocular Depth Estimation

- MiDaS or DPT neural network for depth from single image
- Less accurate but cheaper

- Good enough for obstacle avoidance (relative depth matters more)

Option C: PnP + Known Object Sizes

- Use YOLO class to infer typical object dimensions
- PnP solve using assumed 3D structure
- Cheapest option, moderate accuracy

Component 3: LLM-based Semantic Reasoning

Purpose

Use Large Language Model to add semantic understanding and decision-making beyond raw sensor data.

LLM Integration Architecture

```
Scene Context → Prompt Engineering → LLM API (Claude/GPT-4) → Decision/Action
   ↓
YOLO Detections
Current Task
Robot State
Past Observations
```

Use Cases

1. Ambiguous Situation Resolution

Scenario: Robot detects person standing in corridor, blocking path.

LLM Input (Structured Prompt):

```
Current Situation:
- Task: Deliver coffee to Table 5 in Food Court
- Location: Main Corridor B, 10m from destination
- Obstacle: Person standing still, facing away from robot
- Path: Blocked (person centered in corridor)
- Alternative routes: Left detour (+15m, narrow), Right detour (+20m, through crowd)
- Time constraint: Deliver within 3 minutes
- Robot capabilities: Can wait, navigate around, request passage (speaker)

Question: What should the robot do?
```

LLM Output:

```json
```

```
{
  "action": "wait_and_request_passage",
  "reasoning": "Person is likely temporary obstacle (looking at phone/map).
              Waiting 5-10 seconds is faster than detours.
              Polite audio cue ('Excuse me, delivery robot passing through')
              alerts person without being intrusive.",
  "timeout": 10,
  "fallback_action": "take_right_detour"
}
```

2. Delivery Etiquette & Social Navigation

Scenario: Robot approaching table with customers.

LLM Input:

```
Delivery Context:
- Destination: Table 5, Food Court
- Detected: 3 people seated, 1 standing
- Activity: Appears to be eating (plates on table)
- Approach angle: From behind seated customers
- Delivery item: Hot coffee (spillable, urgent)

Question: How should the robot approach and deliver?
```

LLM Output:

```json
{
  "approach": "from_visible_angle",
  "speed": "slow",
  "announcement": "Delivery for Table 5",
  "waiting_position": "2m from table, in field of view",
  "delivery_method": "wait_for_acknowledgment",
  "reasoning": "Approaching from behind while people eat is startling.
              Position in front where visible, announce politely,
              wait for customer to acknowledge before final approach."
}
```

3. Dynamic Obstacle Classification

Scenario: YOLO detects "unknown object" blocking path.

LLM Input (with image embedding if using vision-language model):

```
Object Description:
- Detected by YOLO: unknown class (confidence 0.45)
- Visual features: Brown, rectangular, ~0.5m tall
- Location: Middle of corridor
- Context: Near trash bins and cleaning closet
- Time: 7:30 AM (pre-opening hours)

Question: What is this object likely to be? Should robot proceed or wait?
```

LLM Output:

```json
{
  "classification": "cleaning_equipment_or_box",
  "confidence": 0.7,
  "action": "proceed_slowly_with_clearance",
  "reasoning": "Location near cleaning area and pre-opening hours suggests
               temporary cleaning equipment. Object is stationary.
               Safe to proceed with wide clearance and low speed.",
  "risk_level": "low"
}
```

4. Route Planning with Contextual Awareness

Scenario: Multiple path options with different characteristics.

LLM Input:

```
Route Options:
- Route A: Direct (50m), through food court, currently lunch rush (11:45 AM)
- Route B: Detour (80m), service corridor, empty but poorly lit
- Route C: Detour (70m), retail corridor, moderate traffic
- Delivery: Ice cream (time-sensitive, melts)
- Priority: Speed vs. Safety vs. Customer experience

Question: Which route should robot take?
```

LLM Output:

```json
```

```
{
  "selected_route": "C",
  "reasoning": "Ice cream is time-sensitive but Route A (food court during lunch)
              has high collision risk and would navigate around many people,
              actually slower than Route C. Route B (service corridor)
              is inappropriate for customer-facing delivery robot.
              Route C balances speed with acceptable environment.",
  "estimated_time": "90 seconds",
  "precautions": ["moderate_speed", "extra_awareness_near_stores"]
}
```

LLM API Integration

Implementation Approach:

- Use Anthropic Claude API or OpenAI GPT-4
- Structured JSON prompts for reliability
- Response parsed as JSON for programmatic decision-making
- Fallback to rule-based behavior if API fails
- Cache common scenarios to reduce API calls
- Rate limiting: Only query LLM for ambiguous situations (not every frame)

Latency Considerations:

- LLM inference: 500-2000ms
- Use only for non-time-critical decisions
- Robot can pause briefly while awaiting LLM response
- Async processing: Robot continues with safe default behavior

---

Component 4: Path Planning & Navigation (ROS2)

ROS2 Integration via `rclrs`

Why ROS2?

- Standard robotics middleware
- Rich ecosystem of navigation packages
- Proven algorithms for mobile robots
- Easy integration with existing tools

Key ROS2 Packages (Rust bindings via rclrs):

- `nav2`: Navigation stack
- `costmap_2d`: Obstacle map representation
- `dwb_local_planner` or `teb_local_planner`: Local trajectory planning
- `amcl`: Particle filter localization (backup to our custom solution)

Navigation Pipeline

```
Current Pose + Goal → Global Planner → Waypoints
                           ↓
        Costmap (from obstacles) → Local Planner → Velocity Commands
                           ↓
                    STM32 Motor Control
```

Global Planning

- **Algorithm**: A* or Dijkstra on coarse grid map
- **Input**: Start position, goal position, static mall map
- **Output**: Sequence of waypoints through free space
- **Update Rate**: 0.1-1 Hz (replan if path blocked)

Local Planning

- **Algorithm**: DWA (Dynamic Window Approach) or TEB (Timed Elastic Band)
- **Input**: Waypoints, current velocity, dynamic obstacle map (from YOLO+PnP)
- **Output**: Linear and angular velocity commands
- **Update Rate**: 10-20 Hz (reactive obstacle avoidance)
- **Constraints**:
  - Max velocity: 0.8 m/s (safe indoor speed)
  - Max acceleration: 0.5 m/s²
  - Min obstacle clearance: 0.5m (person), 0.3m (kiosk)

Costmap Layers

1. **Static Layer**: Pre-loaded mall map (walls, permanent fixtures)
2. **Inflation Layer**: Safety buffer around obstacles
3. **Obstacle Layer**: Dynamic obstacles from YOLO detections
4. **Social Layer**: Higher cost near people (respectful navigation)

Component 5: Low-Level Motor Control (STM32 + Embassy)

Hardware Setup

Microcontroller: STM32F4 or STM32H7

- F4: 168 MHz, sufficient for PID control
- H7: 480 MHz, better for complex sensor fusion

Motors: Brushed DC or BLDC with encoders

- Differential drive (2 main wheels + passive casters)
- Encoder resolution: 1000+ PPR for smooth control

Motor Drivers: H-bridge (L298N) or ESC

- PWM input for speed control
- Direction pins for forward/reverse

IMU: MPU6050 or BMI088

- 3-axis accelerometer + 3-axis gyroscope
- I2C or SPI interface
- Sample rate: 100-200 Hz

Embassy RTOS

Why Embassy?

- Async/await in embedded Rust (no_std)
- Hardware abstraction layer for STM32
- Efficient multitasking without RTOS overhead
- Type-safe peripheral access

Task Structure:

```
Task 1: IMU Reading (200 Hz)
Task 2: Encoder Reading (100 Hz)
Task 3: PID Control Loop (50 Hz)
Task 4: Serial Communication with Pi (20 Hz)
Task 5: Watchdog & Safety Monitor (10 Hz)
```

PID Motor Control

Control Architecture:

```
Target Velocity (from Pi) → PID Controller → PWM Duty Cycle → Motor
                                ↑
                         Encoder Feedback
```

PID Tuning:

- **P (Proportional)**: Immediate response to error
- **I (Integral)**: Eliminate steady-state error
- **D (Derivative)**: Dampen oscillations

Separate PID for each wheel:

- Compensates for motor asymmetries
- Enables precise differential drive control

Control Loop (50 Hz):

```
1. Read encoder counts → Calculate actual velocity
2. Compute error = target_velocity - actual_velocity
3. PID calculation: output = Kp*error + Ki*integral + Kd*derivative
4. Clamp output to PWM range [0, 255]
5. Apply PWM to motor driver
```

IMU Sensor Fusion

Complementary Filter (simpler) or **Kalman Filter** (more accurate):

Purpose: Fuse accelerometer and gyroscope for orientation estimation

- **Gyroscope**: High frequency, drifts over time
- **Accelerometer**: Low frequency, affected by motion

Fusion Algorithm:

```
angle = α * (angle + gyro_rate * dt) + (1-α) * accel_angle
```

Where $\alpha \approx 0.98$ (trust gyro more for short term, accel for long term)

Output to Pi:

- Estimated heading (yaw angle)
- Angular velocity
- Linear acceleration (for wheel slip detection)

Communication Protocol

Serial/CAN Bus Message Format:

Pi → STM32 (Commands):

```
[START_BYTE] [MSG_ID] [LINEAR_VEL] [ANGULAR_VEL] [CHECKSUM]
```

STM32 → Pi (Telemetry):

```
[START_BYTE] [MSG_ID] [LEFT_VEL] [RIGHT_VEL] [HEADING] [IMU_X] [IMU_Y] [IMU_Z] [CHECKSUM]
```

Update Rate: 20 Hz (50ms interval)

Component 6: Integration & System Operation

Delivery Task Flow

```
1. Task Assignment
  ↓
2. Route Planning (BLE coarse localization + mall map)
  ↓
3. Navigate to Source
   │ - Follow global plan waypoints
   │ - Use AprilTags for precision at key points
   │ - Avoid obstacles via YOLO + PnP
   │ - LLM reasoning for ambiguous situations
  ↓
4. Pickup Confirmation (user interaction or sensor)
  ↓
5. Navigate to Destination
   │ (same as step 3)
  ↓
6. Delivery (approach, announce, wait for acknowledgment)
  ↓
7. Return to Idle/Charging Station
```

Failure Handling

Localization Loss:

- **Symptom**: No AprilTags visible, BLE signal weak
- **Response**:
  - Rely on dead reckoning (short term)
  - Navigate to known AprilTag location for re-localization

- If lost >30 seconds, request human assistance

Obstacle Blocked Path:

- **Symptom**: No feasible local plan for >10 seconds
- **Response**:
  - Query LLM for situation assessment
  - Attempt alternative route
  - If blocked >60 seconds, notify customer & request assistance

Hardware Failure:

- **Motor failure**: Emergency stop, sound alarm
- **Camera failure**: Switch to BLE-only navigation (reduced capability)
- **IMU failure**: Rely on wheel odometry + AprilTags
- **Communication loss**: STM32 executes emergency stop after 1 second timeout

Power Management

Battery: 24V LiPo, 20-30 Ah capacity **Runtime**: 4-6 hours continuous operation **Charging**: Autonomous return to docking station at <20% battery

---

Technical Challenges & Solutions

Challenge 1: Crowded Environments

Problem: Malls during peak hours have high pedestrian density.

Solution:

- Social navigation costmap (higher cost near people)
- LLM reasoning for crowd navigation strategy
- Polite audio announcements ("Excuse me, robot passing through")
- Slow speed in crowds (0.3 m/s vs 0.8 m/s normal)
- Prediction of pedestrian motion (Kalman filter on person tracks)

Challenge 2: Dynamic Lighting

Problem: Mall lighting varies (bright stores, dim corridors, sunlight from skylights).

Solution:

- Auto-exposure/gain control on camera
- AprilTag detection robust to lighting (high contrast markers)

- YOLO trained on diverse lighting conditions
- Fallback to BLE if vision fails

Challenge 3: Elevator Navigation

Problem: Robot needs to use elevators for multi-floor delivery.

Solution:

- Integrate with mall elevator control system (API or button pressing mechanism)
- AprilTag at elevator entrance and inside car for precise positioning
- Wait for empty car or request exclusive use (during off-peak)
- LLM reasoning: "Is it appropriate to share elevator with customers?"

Challenge 4: Real-Time Performance

Problem: Multiple computationally intensive tasks (YOLO, localization, planning, LLM).

Solution:

- Hardware acceleration: Use Jetson Nano for YOLO inference (GPU)
- Task prioritization: Critical (obstacle avoidance) > Important (localization) > Optional (LLM reasoning)
- Async processing: LLM queries don't block navigation
- Model optimization: Quantized YOLO, efficient AprilTag detector

---

Deployment Strategy

Phase 1: Simulation & Testing

- Gazebo simulation with mall environment model
- Test navigation algorithms without hardware risk
- LLM prompt engineering and validation

Phase 2: Controlled Environment

- Deploy in closed section of mall (after hours)
- Test localization accuracy with real beacons/AprilTags
- Validate obstacle avoidance with staged scenarios

Phase 3: Pilot Program

- Limited deployment during off-peak hours
- 1-2 robots, simple routes (e.g., food court to tables)

- Human safety supervisor present
- Collect data on performance and customer interaction

Phase 4: Scale-Up

- Expand to full mall during all hours
- Fleet management (5-10 robots)
- Autonomous charging, task assignment
- Integration with mall operations (elevators, doors, security)

Key Technologies Summary

Software Stack

Component	Technology	Purpose
High-level control	Rust + `tokio`	Async multi-tasking
BLE scanning	`btleplug`	Coarse localization
Computer vision	`opencv`	Image processing, AprilTag
Object detection	YOLO (via ONNX)	Obstacle identification
LLM reasoning	Claude/GPT-4 API	Semantic decision making
Path planning	ROS2 + `rclrs`	Navigation stack
Low-level control	Rust + `embassy`	Real-time motor control
Sensor fusion	Extended Kalman Filter	Localization fusion

Hardware Stack

Component	Specification	Purpose
Compute (High-level)	Raspberry Pi 4 or Jetson Nano	Vision, planning, LLM
Compute (Low-level)	STM32F4/H7	Motor control, sensor reading
Camera	RGB 640x480, 170° FOV	Vision, AprilTag, YOLO
BLE Radio	Built-in Pi Bluetooth	Beacon scanning
IMU	MPU6050 or BMI088	Orientation estimation
Motors	Brushed DC with encoders	Differential drive
Battery	24V LiPo, 20-30 Ah	4-6 hour runtime
AprilTags	Printed markers (15cm)	Precise localization

Future Enhancements

1. **Multi-Robot Coordination**: Fleet management with task allocation

2. **Human-Robot Interaction**: Touchscreen UI for direct customer orders
3. **Voice Interface**: "Robot, deliver coffee to Table 5"
4. **Predictive Maintenance**: ML models to predict motor/battery failures
5. **3D Navigation**: Use stairs/escalators (requires more complex hardware)
6. **Manipulation**: Robotic arm to pick up items from shelves
7. **Customer Following**: "Follow me" mode to guide customers to stores

---

Conclusion

This robotics delivery system extends the indoor navigation platform into the physical world, creating a complete autonomous delivery ecosystem. By combining proven localization techniques (BLE + AprilTags), modern AI (YOLO + LLM), and robust control (Embassy + PID), the robot can safely and efficiently navigate complex mall environments while maintaining social awareness and adaptability.

The use of Rust throughout—from embedded firmware to high-level planning—ensures memory safety, predictable performance, and code sharing between components. The LLM integration adds a layer of semantic reasoning that goes beyond traditional robotics, enabling the robot to understand context and make nuanced decisions in ambiguous situations.

This design is practical, scalable, and builds directly on the existing indoor navigation infrastructure, making it a natural and valuable extension of the original project.