Autonomous Robot Development Plan

Phase 1 — Basic Motor Control via ESP32

Objective: Ensure motors are controllable via ESP32 with a simple program.

Tasks:

- Set up ESP32 development environment (Arduino IDE or PlatformIO).
- Connect ESP32 to motor driver (MX1508 / TB6612FNG).
- Write a basic program to control:
  - Forward / Reverse
  - Left / Right turns
  - Stop
- Test PWM signals for speed variation.
- Ensure smooth motion on a straight line without sensors.

Deliverable: Motors respond reliably to ESP32 commands.

Phase 2 — Add IR Sensors for Wall Collision Avoidance

Objective: Implement basic obstacle/wall collision avoidance.

Tasks:

- Connect IR sensors to STM32 / ESP32 GPIO pins.
- Write a fast real-time loop to read IR sensor values.
- Compute PWM corrections based on left/right distance readings.
- Apply corrections to motor PWM signals to avoid collisions.
- Test wall-following or obstacle avoidance in a small environment.

Deliverable: Robot can move and automatically avoid walls/obstacles.

Phase 3 — Establish Communication between Raspberry Pi and STM32

Objective: Enable Pi to send high-level navigation commands to the microcontroller.

Tasks:

- Choose communication protocol: UART (recommended) or I2C.
- Connect Pi TX/RX to STM32 UART (with common ground).
- Write simple Pi program to send commands:
  - Forward / Reverse
  - Turn Left / Turn Right
  - Stop / Rotate
- Implement a command queue on STM32 to safely store incoming commands.
- Test sending commands from Pi and executing on MCU while maintaining wall collision avoidance.

Deliverable: Pi can control robot movements remotely through high-level commands.

Phase 4 — Implement High-Level Control Functions on Pi

Objective: Create modular control functions for navigation.

Tasks:

- Write Pi functions:
  - move_forward(distance)
  - move_backward(distance)
  - turn_left(angle)
  - turn_right(angle)
  - stop()
  - rotate(degrees)
- Integrate sensor feedback (if needed) to adjust commands dynamically.
- Test commands in sequence to ensure precise movement and turns in your maze.

Deliverable: Pi can issue complex movement commands reliably.

Phase 5 — Integrate Pi, STM32, and Motors

Objective: Combine low-level reflexes with high-level navigation.

Tasks:

- Run IR wall collision avoidance on STM32 in a high-priority loop.
- Receive high-level movement commands from Pi via UART.
- Merge high-level commands and low-level corrections:
  - Base PWM from Pi command
  - Modifications from IR sensor feedback
- Test smooth navigation: straight lines, turns at junctions, obstacle avoidance.
- Fine-tune PID / speed corrections if needed.
- Develop a RTOS on STM32 to always run the wall avoidance routine in parallel with command processing.

Deliverable: Robot navigates maze autonomously while avoiding walls and executing precise turns.

Phase 6 — Establish the Connection between Raspberry Pi and ESP32

Objective: Enable communication between Raspberry Pi and ESP32 to get the navigation commands like Google Maps. *Tasks:*

- Choose communication protocol: Wi-Fi
- Make navigation commands logics in ESP32 to command the raspberry pi.
- Test sending commands from ESP32 and executing on Pi while maintaining wall collision avoidance. - Deliverable:* ESP32 can control robot movements remotely through high-level commands.

Phase 7 — Digitalize the Maze on ESP32 via the sensor network

Objective: Create a digital map of the maze using the sensor placement details *Tasks:*

- Place sensors at strategic locations in the maze (target the junctions) and then feed the data and make the digital map of the maze on ESP32.
- Write a program to read sensor data and update the robot position on the digital map.
- Test the digital map accuracy by navigating through the maze and comparing with actual layout.

Deliverable: ESP32 can create and update a digital map of the maze in real-time.

Phase 8 — Implement Pathfinding Algorithm on ESP32

Objective: Enable the robot to find the optimal path through the maze using the digital map using A* algorithm. *Tasks:*

- Implement A* pathfinding algorithm on ESP32 to calculate the shortest path from start to end point in the maze.
- Integrate the pathfinding algorithm with the digital map created in Phase 7.
- Use a weighted graph approach to represent the maze for better pathfinding efficiency.
- Test the pathfinding algorithm by navigating through the maze and verifying the optimal path taken. *Deliverable:* ESP32 can calculate and follow the optimal path through the maze using A* algorithm.

Phase 9 — Devolop the web interface of the maze navigation system

Objective: Create a user-friendly web interface to monitor and control the robot's navigation through the maze. *Tasks:*

- Develop a web interface using HTML, CSS, and JavaScript to display the digital 3d map of the maze.
- Implement real-time updates of the robot's position on the web interface using WebSockets.
- Implement the code to show the caluculated optimal path on the web interface (calculated by esp32 in phase 8).
- Test the web interface for usability and responsiveness.
- Implement the fucntion where user can set the desitnation point on the web interface and send it to ESP32 for pathfinding. *Deliverable:* A functional web interface to monitor and control the robot's navigation through the maze

---

Phase 10 — Final Testing and Optimization

Objective: Ensure the entire system works seamlessly and optimize performance.
Tasks:

- Conduct thorough testing of the entire system, including all hardware and software components.
- Identify and fix any bugs or issues that arise during testing.
- Optimize the performance of the robot's navigation and pathfinding algorithms.
- Fine-tune the web interface for better user experience and responsiveness.
- Prepare documentation and user manuals for the entire system.

Deliverable: A fully functional and optimized maze navigation system.

PROJECT TO-DO LIST

Phase 1: Basic Motor Control via ESP32

- [ ] Install and configure ESP32 development environment (Arduino IDE/PlatformIO)
- [ ] Wire ESP32 to motor driver (MX1508/TB6612FNG)
- [ ] Implement forward/reverse motor control functions
- [ ] Implement left/right turn functions
- [ ] Implement stop function
- [ ] Test and calibrate PWM signals for speed control
- [ ] Validate smooth linear motion without sensors
- [ ] Document motor control pin configurations

Phase 2: IR Sensor Integration for Collision Avoidance

- [ ] Connect IR sensors to STM32/ESP32 GPIO pins
- [ ] Configure GPIO pins for sensor input
- [ ] Develop real-time sensor reading loop
- [ ] Implement PWM correction algorithm based on sensor data
- [ ] Create wall-following logic
- [ ] Test obstacle avoidance in controlled environment
- [ ] Calibrate sensor thresholds for optimal detection
- [ ] Document sensor placement and wiring diagram

Phase 3: Raspberry Pi ↔ STM32 Communication

- [ ] Select communication protocol (UART recommended)
- [ ] Wire Pi TX/RX to STM32 UART with common ground
- [ ] Develop Python command sender on Raspberry Pi
- [ ] Implement command receiver on STM32
- [ ] Create command queue system on STM32
- [ ] Define command protocol and message format
- [ ] Test basic command transmission (forward, reverse, turn, stop)
- [ ] Validate command execution with wall avoidance active
- [ ] Document communication protocol specification

Phase 4: High-Level Control Functions on Raspberry Pi

- [ ] Implement `move_forward(distance)` function
- [ ] Implement `move_backward(distance)` function
- [ ] Implement `turn_left(angle)` function
- [ ] Implement `turn_right(angle)` function
- [ ] Implement `stop()` function
- [ ] Implement `rotate(degrees)` function

- [ ] Add sensor feedback integration for dynamic adjustments
- [ ] Create movement sequence testing script
- [ ] Validate precision of distance and angle calculations
- [ ] Document API for control functions

Phase 5: System Integration (Pi + STM32 + Motors)

- [ ] Implement high-priority IR avoidance loop on STM32
- [ ] Integrate UART command reception with motor control
- [ ] Develop command-sensor fusion algorithm
- [ ] Implement real-time PWM adjustment logic
- [ ] Test straight-line navigation with obstacles
- [ ] Test turning at maze junctions
- [ ] Tune PID parameters for smooth motion
- [ ] Implement RTOS on STM32 for parallel processing
- [ ] Conduct end-to-end navigation tests
- [ ] Document integration architecture

Phase 6: Raspberry Pi ↔ ESP32 Wi-Fi Communication

- [ ] Configure Wi-Fi on both Raspberry Pi and ESP32
- [ ] Establish network connection between devices
- [ ] Develop navigation command logic on ESP32
- [ ] Implement command relay from ESP32 to Pi
- [ ] Create API endpoints for remote control
- [ ] Test remote command transmission over Wi-Fi
- [ ] Validate navigation commands with collision avoidance
- [ ] Implement error handling and reconnection logic
- [ ] Document Wi-Fi communication architecture

Phase 7: Digital Maze Mapping on ESP32

- [ ] Design sensor placement strategy for maze junctions
- [ ] Install sensors at strategic locations
- [ ] Develop data structure for digital maze representation
- [ ] Implement sensor data collection routine
- [ ] Create map update algorithm
- [ ] Develop robot position tracking system
- [ ] Test map accuracy in actual maze environment
- [ ] Implement map visualization for debugging
- [ ] Validate map against physical maze layout
- [ ] Document sensor network and mapping logic

Phase 8: A* Pathfinding Algorithm Implementation

- [ ] Research and design A* algorithm for maze navigation
- [ ] Implement weighted graph representation of maze
- [ ] Code A* pathfinding algorithm on ESP32

- [ ] Integrate pathfinding with digital map from Phase 7
- [ ] Implement heuristic function for optimal performance
- [ ] Create path calculation testing suite
- [ ] Test algorithm with various start/end points
- [ ] Optimize algorithm for ESP32 memory constraints
- [ ] Validate optimal path selection
- [ ] Document pathfinding algorithm and complexity analysis

Phase 9: Web Interface Development

- [ ] Design UI/UX wireframes for web interface
- [ ] Develop HTML structure for 3D maze visualization
- [ ] Implement CSS styling for professional appearance
- [ ] Code JavaScript for interactive controls
- [ ] Implement WebSocket connection for real-time updates
- [ ] Display robot position updates on 3D map
- [ ] Visualize calculated optimal path on interface
- [ ] Implement user destination selection feature
- [ ] Create ESP32 endpoint for receiving destination commands
- [ ] Test interface responsiveness and usability
- [ ] Implement mobile-friendly responsive design
- [ ] Document web interface API and user guide

Phase 10: Final Testing and Optimization

- [ ] Conduct comprehensive hardware integration tests
- [ ] Perform end-to-end software system tests
- [ ] Execute stress testing under various conditions
- [ ] Identify and log all bugs and issues
- [ ] Debug and resolve critical issues
- [ ] Optimize navigation algorithm performance
- [ ] Optimize pathfinding algorithm efficiency
- [ ] Fine-tune web interface responsiveness
- [ ] Conduct user acceptance testing
- [ ] Optimize power consumption
- [ ] Create system architecture documentation
- [ ] Write technical documentation for developers
- [ ] Prepare user manual with setup instructions
- [ ] Create troubleshooting guide
- [ ] Record demonstration videos
- [ ] Prepare final project presentation

General Project Management Tasks

- [ ] Set up version control repository (Git)
- [ ] Create project timeline with milestones
- [ ] Establish regular testing schedule

- [ ] Document all design decisions
- [ ] Maintain component inventory list
- [ ] Track budget and expenses
- [ ] Schedule regular progress reviews
- [ ] Prepare backup plans for critical components
- [ ] Create risk assessment and mitigation plan
- [ ] Set up continuous integration (if applicable)

Total Tasks: 123
Phases: 10
Status: Not Started

Last Updated: October 29, 2025