Autonomous Mapping Robot with RF Remote Control
Part 1: Project Documentation

Victor Suarez, Jonchang Park, Akshat Shrivastava

December 15, 2025

1 Project Description

This project is an Autonomous Mapping Robot that combines manual control, and autonomous exploration with real-time environmental mapping. The system operates in three distinct modes controlled via a 433MHz RF remote control, which provides flexibility for direct user control or autonomous operation. The robot creates a 16×16 grid map with approximately 12-inch cell resolution, drawing a map with ASCII characters in the Arduino IDE serial monitor.

1.1 Hardware Components of this project:

1.1.1 Core Controller

- **Arduino Nano 33 BLE Sense**: Main micro-controller with an integrated 9-axis IMU (LSM9DS1), and ambient light sensor (APDS9960)

1.1.2 Mechanical Components

- **4WD Chassis**: We decided to go with a four-wheel drive configuration with independent DC motors to provide enhanced traction and maneuverability because with only two motors we were afraid it won't move properly in a carpeted floor. The chasis and the ultrasonic sensor mount were both found in the webpage thingiverse.com. Initially we thought on buying the chasis through amazon but later decided to use a 3D printer because it gave us more room for customization and adaptation to the needs of our project. The 3D printer we used was a Bambu A1.
- **Motor Driver**: Direct H-bridge L293D motor driver using Arduino PWM pins.

1.1.3 Sensors

- **HC-SR04 Ultrasonic Sensor**: Mounted on servo for 90° surroundings scanning, capable of measuring distances from 2cm to 400cm

- **SG90 Micro Servo**: This component moves the ultrasonic sensor between 45°, 90°, and 135° for left, center, and right scanning
- **LSM9DS1 IMU**: This is the micro-controller own 9-axis inertial measurement unit with 3-axis accelerometer and gyroscope for motion detection and collision sensing, plus 3-axis magnetometer for heading calculation and navigation
- **APDS9960**: This is the own micro-controller ambient light and RGB color sensor, used for environmental data collection

1.1.4 Control Interface

- **Adafruit Simple RF M4 Receiver (RO2A)**: 433MHz receiver module operating at 5V with resistive voltage dividers for 3.3V Arduino compatibility. We decided to use resistors instead of a level shifter for lack of space reasons.
- **4-Button RF Keyfob Remote**: This is the wireless control with approximately 100-150 foot range
- **Voltage Dividers**: Eight 1kΩ resistors (two per button) that convert 5V signals to safe 2.5V levels.

1.1.5 Feedback Systems

- **RGB LED**: Microcontroller own LED lights indicating operational status (Green: Manual, Cyan: Auto, Magenta: Returning)
- **Passive Buzzer**: Piezzo buzzer for Audio feedback for events (startup, mode changes, obstacle detection, collisions)

1.1.6 Power System

- **2$\times$ 9V Batteries (18V Total)**: Primary power source for motors
- **Buck Converter**: Steps down 18V to stable 5V for sensors, servo, and Arduino VIN
- **Arduino 3.3V Regulator**: Powers 3.3V logic components from 5V input

1.2 How the Robot-Car operates in different modes:

1.2.1 Mode 1: Manual Control (Green LED)

In manual mode, the robot responds when buttons are pressed. It was necessary to perform some calibrations :

- **Button A (Forward)**: Executes one forward step (approximately 12 inches using calibrated MOVE_TIME of 436ms) and a speed motor of 75 RPM. However; for a different type of surface, this calibration will have to be performed again.

- **Button B (Backward)**: Executes one backward step for repositioning.
- **Button C (Rotate Left)**: Executes a calibrated 90° counter-clockwise rotation. The rotation angle depends on the ROTATION_TIME and also in the ROTATION_SPEED. It has to be calibrated for different surfaces.
- **Button D (Mode Toggle)**: Switches to autonomous mode and sets current position as ”home”.

The LED light shows blue for forward motion, yellow for backward, and cyan for rotation, which gives us instant visual feedback. In this step-based manual mode, each button press moves the device one step, which helps with calibration and precise positioning.

1.2.2 Mode 2: Autonomous Exploration (Cyan LED)

When Button D is pressed from manual mode, the robot enters autonomous exploration and establishes the current grid position as ”home”:

1. **Sensor Scanning**: Servo sweeps the ultrasonic sensor to three positions (45°, 90°, 135°), measuring distances at each angle. It reads the distance first, and then advances if path is clear; however if an obstacle comes in once the reading is done, the robot is going to advance and hit the obstacle, unless in the new step it ”sees” the obstacle before hitting it. Also we noticed that some materials are not detected with this sensor, especially soft materials like cushions, in this case the robot just keeps going forward and hits them without detection.
2. **Obstacle Detection**: If center distance ¡ 20cm, an obstacle is detected and the robot avoids it.
3. **Navigation Decision**:
   - If path is clear: Move forward one grid cell (12 inches).
   - If obstacle ahead: The robot compares left vs. right distances, and turn towards more open space.
   - If trapped (both sides blocked): The robot reverse and execute a 180° turn.
4. **Map Updates**: After each move, the robot updates the 16×16 occupancy grid with obstacle/clear/visited status.
5. **Collision Recovery**: The micro-controller IMU detects sudden acceleration spikes ($> 2.0g$), which triggers automatic backup and re-routing.

The robot keeps track of its heading (0 for North, 1 for East, 2 for South, 3 for West) and its position using grid coordinates. After every eight moves, it prints statistics such as the number of moves, obstacles found, map coverage percentage, and elapsed time. Also after eight moves it prints the 16-16 grid updated map.

1.2.3 Additional Mode: Return to Home (Magenta LED) based shortest distance vector

Pressing the Button D during autonomous mode activates the return-to-home navigation. The robot does this by following these steps:

1. Calculate displacement vector: $\Delta x = x_{home} - x_{current}$, $\Delta y = y_{home} - y_{current}$
2. Distance: $d = |\Delta x| + |\Delta y|$
3. Priority-based navigation: Resolve largest displacement first (X or Y), then secondary axis
4. Turn toward required heading using shortest rotation path
5. Check for obstacles before moving forward
6. Move forward one grid cell if path clear, otherwise navigate around
7. Repeat until $d \leq 1$ (within one cell of home)
8. Once it gets to the home grid position, the robot a play success tone sequence with the buzzer and returns to manual mode
9. For secure operation we implemented a 2-minute timeout that prevents infinite loops if the path is completely blocked

1.3 Mapping System used by the Robot:

The robot maintains a 16×16 grid (256 cells total) where each cell stores:

- **Status**: Unknown (dot), Clear (space), Obstacle (block), or Visited (*)
- **Light Level**: 0-255 ambient brightness
- **Compass Heading**: 0-360° when cell was visited
- **Collision Count**: Number of collisions detected in cell (displayed as digit)

Map visualization uses ASCII symbols displayed via serial monitor:

- H = Home position (set when entering autonomous mode)
- $\uparrow/\rightarrow/\downarrow/\leftarrow$ = Current robot position and heading
- Numbers = Collision frequency at that location

Memory space: Each cell uses 5 bytes, totaling 1,280 bytes for the entire 16×16 map.

1.4 Signal Processing

1.4.1 Ultrasonic Distance Measurement

1. Trigger pin pulsed HIGH for 10µs
2. Echo pin times the reflected ultrasonic pulse
3. Distance calculated: $d = \frac{t \times 0.034}{2}$ cm, where t is pulse duration in microseconds
4. Invalid readings (timeout or ¡2cm) clamped to safe values (2cm minimum, 400cm maximum)
5. 30ms timeout prevents indefinite blocking

1.4.2 Compass Heading Calculation with Calibration

1. Read magnetometer X and Y components: m_x, m_y
2. Apply hard-iron offset calibration: $m'_x = m_x - offset_x$, $m'_y = m_y - offset_y$
3. Calculate heading: $\theta = \arctan 2(m'_y, m'_x) \times \frac{180}{\pi}$
4. Normalize to 0-360°: if $\theta < 0$, then $\theta = \theta + 360$
5. Apply low-pass filter: $\theta_{filtered} = (1 - \alpha)\theta_{old} + \alpha\theta_{new}$ with $\alpha = 0.2$
6. 10-second calibration period at startup rotates through full 360° to capture magnetic field extrema
7. Offsets calculated as: $offset_x = \frac{max_x + min_x}{2}$, $offset_y = \frac{max_y + min_y}{2}$

1.4.3 Collision Detection

1. Calculate acceleration magnitude: $a = \sqrt{a_x^2 + a_y^2 + a_z^2}$
2. If $a > 2.0g$ while motors active: collision detected
3. Immediate motor stop, increment collision counter for current cell
4. Execute recovery sequence: reverse 330ms, update map with hazard marking

1.5 Limitations

- **Map Resolution**: 16×16 grid with 12" cells provides reasonable coverage but coarse detail; finer resolution requires more memory
- **Ultrasonic Accuracy**: ±3cm measurement error; unreliable on angled or soft surfaces. In soft surfaces the sensor sometimes reads as it is clear ahead.

- **Return-to-Home**: Uses grid-based navigation with basic obstacle detection; it could fail if direct path to home grid is consistently blocked
- **RF Range**: Approximately 100-150 feet line-of-sight with 5V operation; reduced through walls or obstacles
- **Battery Life**: Approximately 45-60 minutes continuous operation depending on motor usage
- **Compass Interference**: Metal objects and motor current create magnetic field distortion; readings only accurate when motors stopped
- **Bluetooth Unavailable**: Initial Bluetooth implementation failed due to voltage brownout from motor current power pull.

2 Development Process

2.1 Phase 1: Hardware Selection and Integration

2.1.1 Chassis Selection

There were two different options for the driving system of our robot car: 2WD and 4WD. At first, we chose a 2WD, and tested the autonomous exploration mode on the common floor of the computer science building. Unfortunately, the car struggled to move in carpeted floors, and the intended motion could not be performed properly when the grip on the floor was weak. In addition, the car sometimes lost balance because the wheel size was not large enough hold up the body weight of the car.

In order to solve this issue, Victor suggested 4WD to support the robot in balance, and Jongchan and Akshat agreed with this opinion. Jongchan decided to update code related to the driving system for 4WD. Also, he updated the motor control and movement software accordingly. During this process, Victor and Akshat provided feedback based on their mapping and sensor system observations to maintain stability and consistent movement for accurate sensing. As the number of wheels increased, the main body of the car was more stably supported. As a result, the robot was able to move as we intended in the same environment and avoided obstacles more reliably.

2.1.2 Motor Control Evolution

Initial Approach:
In order to streamline the robot's motor control from a hardware standpoint, Victor first proposed utilizing the Adafruit Motor Shield v3 (victor had one from a previous project). This Motor shield was designed to be stacked on top of an Arduino One. We tried to adapt this motor shield to the Arduino Nano using a different wire configurations, but we were unsuccessful, we tested the motor shield and discovered that it was not outputting any voltage to the micro-controller.

Final decision:
Jongchan suggested doing away with the motor shield completely and using the Arduino's PWM pins to switch to a direct H-bridge motor control method in light of system complexity, spatial limitations, and software-level stability. This method enables more predictable software control, but it does require more GPIO pins (a total of 12 pins for four motors: two direction pins and one PWM enable pin per motor). Both Victor and Akshat concurred that this design enhanced sensor reliability and hardware stability. Finally, overall stability of the robot was greatly improved.

Motor control connections

- Left motors: EN=5, IN1=2, IN2=3, EN2=10, IN3=8, IN4=9
- Right motors: EN=6, IN1=4, IN2=7, EN2=11, IN3=13, IN4=A0

2.1.3 Voltage Level Conversion

The Arduino Nano 33 BLE Sense operates at 3.3V logic, while the RF receiver requires 5V for an optimal transmission range and reliable signal detection. Initial attempts to power the receiver at 3.3V resulted in unreliable operation with intermittent button detection and single-press failures.

Solution Implemented: Resistive voltage dividers using 1kΩ resistor pairs:

The RF receiver operates at full 5V from the buck converter for maximum range, while simple voltage dividers reduce the 5V digital outputs to safe 2.5V levels for Arduino inputs:

- RF Receiver VCC → Buck converter 5V output
- Each RF output (D0-D3) → 1kΩ resistor → Arduino input (A2-A5)
- Arduino input → 1kΩ resistor → GND
- Output voltage: $V_{out} = 5V \times \frac{R_2}{R_1+R_2} = 5V \times \frac{1k\Omega}{2k\Omega} = 2.5V$

Testing confirmed reliable button detection with no missed presses or single-operation failures that occurred during 3.3V operation attempts.

2.1.4 Power Distribution Network

A critical aspect of this project was managing multiple voltage domains:

1. **18V Domain**: Two 9V batteries in series directly powering the DC motors (higher voltage = more torque). We used this because the motors pull too much power that was weakening the overall operation of the robot.
2. **5V Domain**: Buck converter steps down 18V to 5V for:
   - Arduino VIN (which then regulates to 3.3V internally)

- Servo motor (requires 5V, draws up to 500mA at stall)
- Ultrasonic sensor (5V tolerant, better range at 5V)
- RF receiver (5V for maximum transmission range and reliability)
- Buzzer (5V for louder audio output)

3. **3.3V Domain**: Arduino's onboard regulator powers:
   - IMU sensor (LSM9DS1 on I2C)
   - Light sensor (APDS9960 on I2C)
   - RGB LED (through current-limiting resistors)
4. **Signal Level Conversion**: Resistive voltage dividers (1kΩ/1kΩ) convert RF receiver 5V outputs to 2.5V for safe Arduino input

A Common ground was established across all the circuit to ensure proper voltage ground referencing.

2.2 Phase 2: Software Development

2.2.1 Library Integration

The following Arduino libraries were utilized:

1. **Arduino_LSM9DS1.h**: Official Arduino library for onboard 9-axis IMU
   - Used for: Accelerometer, gyroscope, and magnetometer readings
   - Modification: None required
   - Initialization challenges: IMU sometimes fails to initialize on first attempt; implemented 5-retry loop with 500ms delays
2. **Arduino_APDS9960.h**: Official Arduino library for ambient light and color sensor
   - Used for: Ambient light level measurement
   - Modification: Implemented scaling from 12-bit ADC values (0-4096) to 8-bit (0-255) for memory efficiency
3. **Servo.h**: Standard Arduino servo control library
   - Used for: Positioning ultrasonic sensor at scan angles
   - Modification: None required
   - Note: Attached to pin A1 (digital pin 15), requires 300ms settle time between positions

2.2.2 Learning Resources

The development process involved learning from multiple educational sources:

Udemy Courses: Several Udemy courses on Arduino robotics and motor control provided foundational knowledge:

- Basic motor control patterns and H-bridge operation
- Ultrasonic sensor integration for 2-motor robots
- Example code targeted at Arduino UNO platform

Adaptation Required: The Udemy examples were designed for:

- 2-motor (2WD) robots rather than 4-motor (4WD) systems
- Arduino UNO microcontroller with different pin layouts and 5V logic
- Simpler navigation without mapping capabilities

All code was substantially modified and expanded to work with the Arduino Nano 33 BLE Sense, implement 4-motor control, add compass-based navigation, create the 16×16 mapping system, and integrate RF remote control.

2.2.3 RF Remote Integration

Initial Bluetooth Attempt: The project initially attempted Bluetooth Low Energy (BLE) connectivity using the Arduino BLE library for real-time map visualization on a computer or smartphone.

Critical Bluetooth Failure: During development, a problem emerged that prevented Bluetooth operation:

- **Initial Testing Success**: Bluetooth worked perfectly when tested without the four motors in operation.
- **Failure with Motors**: Once the four DC motors were operational, Bluetooth connection immediately timed out
- **Root Cause**: Motor current draw caused voltage brownout on the entire system.
- **Symptoms**: System voltage dropped below minimum operating threshold when motors activated, causing the BLE radio to lose power and disconnect

Attempted Solutions (All Unsuccessful):

1. Power isolation: Separated motor power supply from microcontroller power
2. Buck converters: Added voltage regulators to step down 8-12V to stable 5V
3. Larger batteries: Upgraded to higher capacity batteries

4. Multiple code revisions: Attempted power management in software
5. Consultation with experts: Visited experienced engineers who confirmed brownout diagnosis
6. Different battery configurations: We tested different battery setups, including various series and parallel combinations, as well as more powerful batteries.
7. Common ground : We made sure all grounds were properly tied together

None of these solutions resolved the fundamental issue: the four motors simultaneously drawing current created a voltage sag that the Bluetooth radio could not tolerate.

Final Solution - RF Remote with Voltage Dividers: After consulting with the professor, Jongchan shared the discussion and conclusions with the entire team. Based on this, we decided to switch to a 433 MHz RF remote and use resistive voltage dividers for signal level conversion. This approach provided the following advantages.

- No pairing required
- Longer range than BLE (100-150 ft at 5V vs. 30-50 ft for BLE)
- Simple digital pin reading (HIGH when button pressed)
- No complex protocol implementation needed
- More robust to voltage fluctuations
- Voltage dividers provide safe 2.5V signals to Arduino from 5V receiver outputs

Demonstration Workaround: For project demonstrations, map visualization is shown via long USB cable connected to Serial Monitor, proving that collision detection, compass navigation, and mapping functionality all work correctly. The team continues to investigate Bluetooth solutions.

We added software button debouncing with a 300-millisecond timeout to stop the physical RF remote buttons from registering as multiple quick presses due to mechanical bounce. Without this, the robot would quickly cycle through all three modes instead of switching modes with a single press.

2.2.4 Motor Calibration

We calibrated the system to make sure that each 12 inch grid cell matched a single move command from the remote control through experiments. We fine tuned the parameters related to time under many trials and errors.

- **Forward Movement**: 436ms at PWM 100 moves approximately 12 inches (one grid cell)
- **Turn Timing**: 350ms differential drive at PWM 100 achieves 90° rotation
- **Backup Distance**: 330ms achieves approximately 9 inches (0.75 cell)
- **Motor Balance**: LEFT_MOTOR_INVERT and RIGHT_MOTOR_INVERT flags correct for wiring differences

Overall calibration process:

1. Mark a 12-inch grid on the floor with tape
2. Execute 10 forward movements, measure final position
3. Adjust MOVE_TIME until average distance = 12 inches ± 1 inch
4. Repeat for turns using a protractor to measure rotation angle
5. Test on different surfaces (tile, carpet, wood) and average results

Note:

- MOVE_TIME was scaled from original 400ms (which produced ∼11 inches) by factor of 12/11 to achieve 436ms for ∼12 inch movement.
- Victor conducted experiments on both surfaces carpet and wooden floors, and checked average results of them.

2.3 Phase 3: Mechanical Assembly

2.3.1 Component Mounting

- **Chassis Base**: 4WD platform provides pre-drilled mounting holes
- **Arduino**: Positioned centrally using breadboard glued to the chasis.
- **Servo Mount**: The chasis has a hole and a screw hole to mount the servo to the front of chassis
- **Ultrasonic Bracket**: 3D-printed holder mounts HC-SR04 to servo horn
- **Battery Placement**: Two 9V batteries positioned at rear for weight balance, secured with Velcro strap
- **Buck Converter**: Heat-shrink wrapped and zip-tied to chassis side rail
- **RF Receiver**: Positioned for antenna clearance, hot-glued to back of chasis and mounted on a bsmall breadboard.
- **Wiring Management**: We inserted cables into the holes of the breadboard to make connections. For the sensors, we used connectors with male and female ends to attach the different components.

2.3.2 Fabrication Details

3D Printed Components:

- 3d Chasis : The chassis is simple and was obtained from www.thingiverse.com. It has pre-drilled holes for the servo motor and the dc motors.
- Ultrasonic sensor mount: From thingiverse.com we found a mount for the ultrasonic sensor that fit perfectly into the chasis we had.

2.4 Phase 4: Testing and Refinement

2.4.1 Unit Testing

Each subsystem was tested independently:

1. **Motors**: Verified forward/backward/turn operations, confirmed PWM speed control
2. **Servo**: Tested angle accuracy with protractor, verified 300ms settle time requirement
3. **Ultrasonic**: Measured known distances (10cm, 30cm, 50cm, 100cm), confirmed ±3cm accuracy
4. **IMU**: Logged accelerometer during manual movements, verified collision detection threshold
5. **Compass**: Compared calculated heading to phone compass app, aligned to within 15° after calibration
6. **RF Remote**: Tested range in open field (achieved 120 feet with 5V operation), verified button recognition with voltage dividers
7. **Voltage Dividers**: Measured with multimeter - RF receiver outputs: 5V, Arduino inputs after divider: 2.5V (verified safe operation)

2.4.2 Integration Testing

Full system testing revealed several issues:

Problem 1: Robot drifted to one side during straight forward movement.

- **Cause**: Motor speed mismatch due to manufacturing variations
- **Solution**: We combined Victor's own parts and new stuff that we bought from Amazon into our robot. Therefore, there was a problem of motor output imbalance due to differences in the manufacturing process of parts. Victor checked this problem when he saw the abnormalities of straight forward driving during the test, and took a video. After that, he shared it with our group message room. Akshat sent me an Udemy course in which they dealt with the same problem to fix the motor direction setting. After watching this video and discussing with them, I added LEFT_MOTOR_INVERT and RIGHT_MOTOR_INVERT flags by adjusting individual motor PWM values experimentally as well. After that, everyone met for a test and it worked as we intended.

Problem 2: Compass readings fluctuated wildly near motors

- **Cause**: Magnetic interference from motor current
- **Solution**: At first, when the motors were running, the compass readings differed significantly, particularly in the vicinity of the motor area. During testing sensor part, Akshat noticed this issue, and noted that the compass values became erratic when the robot was moving. Victor estimated that the issue might be caused by the motors' electrical current interfering with the compass sensor during moving. The compass was tested with the motors off to verify whether this problem happened due to other electrical sensor or not, and the measurements significantly improved in stability. Thus, Akshat presented a final solution to significantly increase directional accuracy by implementing policies in navigation logic to read compass values only when the robot is stationary, and Jongchan supported it.

Problem 3: Robot occasionally got stuck in corners

- **Cause**: Reactive algorithm has limited memory of recent maneuvers
- **Solution**: During testing, the robot sometimes got stuck in corners because the reactive algorithm kept turning without making forward progress. Victor suggested adding code to detect repeated turning behavior. Based on this idea, Jongchan decided to add a consecutiveTurns counter, and if more than six turns occur without forward movement, the robot performs a forced 180° escape maneuver with a random turn direction. The reason why it was set as the number 6 is since it was judged that a lot of movement has already been made from the 6th time in the actual test. This fix worked well in our tests and resolved the issue.

Problem 4: RF receiver unreliable at 3.3V operation

- **Cause**: Insufficient voltage for reliable signal detection; receiver draws excessive current from Arduino 3.3V pin during reception
- **Solution**: When testing was conducted to connect the RF receiver to Arduino's 3.3V pin, the signal was frequently cut off, and the button input was not properly recognized. This is because the voltage became unstable since the receiver was not supplied with enough current while operating. Therefore, the power of the RF receiver was supplied separately at a stable 5V from the buck converter. Instead, the RF signal output was input by lowering the voltage with a 1kΩ/1kΩ resistor pairs to protect Arduino. As a result, signal recognition was stabilized and communication distance and reliability were greatly improved.

Problem 5: Return-to-home navigation improvements needed

- **Initial Issue**: Simple vector navigation often chose suboptimal paths
- **Solution**: Victor noticed that the robot was choosing inefficient paths during the Return-To-Home mode even though the map and position tracking were working correctly. After we talked about it, we realized the issue was in the navigation logic, not

the mapping. With Victor's help, Jongchan fixed this by adding priority-based heading selection and a timeout to the movement system. Akshat also helped by improving obstacle detection and heading verification during the return mode. Because of this collaboration, the robot was able to return home more reliably.

Problem 6: Bluetooth connectivity failure (Critical)

- **Cause**: Voltage brownout from 4-motor current draw
- **Impact**: Unable to implement wireless map visualization
- **Solution**: At first, we put huge effort to match the voltage. Using 3,000 and 10,000 Amps that Victor and Jongchan have, we tested different voltages. However, it did not work properly, contrary to our intention. Thus, Jongchan sent an e-mail asking for help to the professor, and he had a meeting with the professor as a representative. After this, he shared the process with everyone. However, connecting bluetooth continued to cause problems, and we decided to use the remote control that the professor gave us. Victor connected the remote control to the hardware and Jongchan and Akshat implemented code logic. After that, we were able to control the robot car without wires.
- **Status**: Ongoing investigation for future resolution

Problem 7: Burned microcontroller during development

- **Cause**: Configuring the robot with Victor and Akshat as the main contributor resulted in a lot of complex wiring connections. Under these circumstances, Victor was enthusiastically testing at home, and caused an overvoltage through an incorrect connection that he made by mistake. Victor shared this situation with us.
- **Impact**: Destroyed one Arduino Nano 33 BLE Sense
- **Mitigation**: Team had purchased two spare microcontrollers specifically anticipating this risk
- **Outcome**: Development continued with spare unit without significant delay

2.5 Code Developement

The project code is for the most part original work developed specifically for this robot. However; it builds upon:

- **Udemy courses**: Basic motor control patterns for 2-motor Arduino UNO robots
- **Arduino library examples**: IMU initialization patterns, servo control basics
- **Adafruit tutorials**: Compass heading calculation formulas
- **HC-SR04 datasheet**: Ultrasonic timing specifications

Our Original Implementations:

- 4-motor (4WD) control system adapted from 2-motor examples from Udemy.com courses.
- 16×16 occupancy grid mapping system.
- Three-mode state machine (Manual/Auto/Returning)
- Step-based manual control for calibration. Adjusting speed of motor and time of running to make sure a forward button pressed was equivalent to 12 inches advanced.
- Return-to-home navigation with priority-based pathfinding
- Compass calibration with hard-iron offset correction
- Collision detection and recovery
- RF remote integration
- All the system integration of the code

3 Lessons Learned

3.1 Technical Knowledge Gained

3.1.1 Power System Design and Brownout

The Bluetooth failure taught critical lessons about power system design:

Voltage Brownout: When the four DC motors draw current simultaneously, the battery internal resistance causes voltage drop. Even with 9V batteries rated at high capacity, the instantaneous voltage sag during motor acceleration can drop the system below the minimum operating voltage for sensitive components like the BLE radio.

Insights:

- Test power-hungry components under realistic load conditions early in development
- Voltage regulators have current limits that must accommodate peak draw, not just average
- Battery capacity (mAh) is different from instantaneous current delivery capability
- Separate power domains (motor vs. logic) may require separate battery packs for high-current applications
- Bluetooth and WiFi radios are particularly sensitive to supply voltage stability

Lesson: Always measure actual system voltage under full motor load before assuming power supply adequacy.

3.1.2 Voltage Management

Prior to this project, our understanding of voltage level translation was theoretical. Implementing the TXB0104 level shifter taught us practical lessons about:

- The critical importance of matching logic levels between interconnected devices
- How bidirectional level shifters use MOSFETs with pull-up resistors to translate signals in both directions

Insight: Always verify voltage compatibility before making connections. A $3 level shifter prevented a $25 Arduino replacement.

3.1.3 Component Fragility

Burning one microcontroller during development emphasized important practices:

Plan for Failure: The decision to purchase backup microcontrollers before starting proved wise. Complex wiring with 12+ motor connections creates many opportunities for mistakes.

Incremental Testing: After the incident, we adopted a strict protocol: test every new connection with multimeter before powering on, verify voltages at each component, and add one connection at a time.

Lesson: In complex systems, it is wise to budget 10-20% extra for replacement parts. The cost is insignificant compared to project delays.

3.1.4 Sensor Fusion Challenges

Combining data from multiple sensors proved more complex than anticipated:

Compass Calibration: The magnetometer requires calibration to compensate for hard-iron (permanent magnets) and soft-iron (ferromagnetic materials) interference. The 10-second calibration routine rotates the robot through 360° to capture the min/max magnetic field values, creating an offset calibration.

Learned: Environmental factors dramatically affect sensor readings. The same robot will need recalibration in different locations due to Earth's magnetic field variations and local metal structures (rebar, appliances, wiring).

Accelerometer Noise: Raw accelerometer readings fluctuate by ±0.2g even when stationary. This necessitated the 2.0g threshold for collision detection rather than a lower value.

Learned: Real sensors have noise. Software must implement filtering and thresholds to distinguish signal from noise. A future enhancement would implement a moving average filter or Kalman filter for smoother readings.

3.1.5 Motor Magnetic Interference

Discovery that compass readings were unreliable while motors were running revealed:

- DC motor brushes create significant magnetic fields

- Current pulses through power wiring act as electromagnets
- Magnetometer must be physically separated from motors or readings taken only when stationary
- Metal chassis components can distort Earth's magnetic field

Lesson: Sensor placement is critical. Future designs should position magnetometer on a raised mast away from motors and power wiring.

3.1.6 Wireless Communication Trade-offs

The BLE vs. RF decision highlighted important wireless communication principles:

BLE Complexity: Bluetooth Low Energy uses GATT profiles with services and characteristics. Custom UUIDs require specific phone app support. Connection establishment involves advertising, scanning, pairing, and service discovery—each a potential failure point. Additionally, BLE radio requires stable power supply.

RF Simplicity: The 433MHz RF remote uses amplitude shift keying (ASK) modulation. When a button is pressed, the transmitter sends a coded signal. The receiver decodes it and asserts the corresponding output pin HIGH. This simplicity trades protocol features for reliability and power supply tolerance.

Learned: Choose communication protocols based on application requirements and system constraints. BLE's complexity is justified for bidirectional data exchange and smartphone integration, but overkill for simple button presses. RF's lack of acknowledgment and limited data throughput is acceptable for control applications, and its robustness to power supply variations proved critical.

3.1.7 Real-Time Embedded Programming

Managing multiple concurrent tasks on a single-threaded microcontroller required careful timing analysis:

Blocking Operations: The ultrasonic sensor's pulseIn() function blocks for up to 30ms waiting for echo return. The servo requires 300ms settling time. These delays accumulate:

- 3 ultrasonic readings: $3 \times 30\text{ms} = 90\text{ms}$
- 3 servo movements: $3 \times 300\text{ms} = 900\text{ms}$
- Total scan cycle: 1000ms

This 1 Hz update rate is acceptable for slow-moving robots but would be inadequate for faster applications.

Learned: Embedded systems require careful attention to execution timing. Future improvements could use:

- Non-blocking ultrasonic library with state machine
- Simultaneous servo movement and distance measurement
- Interrupt-driven RF button handling to prevent missed presses

3.1.8 Learning from Educational Resources

The Udemy courses provided valuable foundation but required significant adaptation:

Value of Structured Learning: Video courses explaining motor control theory and basic implementation accelerated understanding compared to reading datasheets alone.

Limitations of Generic Examples: Code designed for 2-motor, Arduino UNO robots couldn't be used directly:

- Different pin configurations required complete rewrite
- 4-motor control needed new logic for synchronized movement
- 5V vs 3.3V logic levels changed all GPIO interactions
- No mapping or navigation examples to build from

Lesson: Educational resources provide conceptual foundation, but real projects require substantial original implementation. Understanding principles is more valuable than copying code.

3.2 Problem-Solving Skills Developed

3.2.1 Systematic Debugging

When the RF receiver initially appeared non-functional, developed a systematic approach:

1. **Isolate**: Tested receiver with simple LED blink code
2. **Verify Power**: Multimeter confirmed 5V at receiver VCC pin
3. **Check Signal**: Multimeter showed D0 pin toggling $0V \rightarrow 5V$ when button A pressed
4. **Level Shifter**: Multimeter confirmed TXB0104 output at 3.3V
5. **Software**: Added Serial.println() to confirm digitalRead() returning correct values

The issue was traced to incorrect pin assignments in code (initially used D0/D1 which are RX/TX for serial communication, conflicting with Serial.begin()).

Learned: Debugging embedded systems requires understanding multiple layers (hardware, electronics, software). Always verify assumptions at each layer before moving to the next.

3.2.2 Persistence Through Failure

The Bluetooth issue required multiple attempted solutions over several weeks:

- First assumption: Software bug in BLE code → rewrote from scratch
- Second assumption: Insufficient decoupling → added capacitors
- Third assumption: Ground loops → verified common ground

- Fourth assumption: Power supply inadequate → tried multiple battery types
- Fifth assumption: Current spikes → added various voltage regulators

Each failed attempt required reassessing assumptions and trying a different approach. The final decision to switch to RF remote required acknowledging that the Bluetooth approach, while initially promising, was fundamentally incompatible with the 4-motor power requirements given the available hardware.

Lesson: Sometimes the best solution is changing the requirement rather than fighting an intractable technical limitation. Knowing when to pivot is as important as persistence.

3.2.3 Constraints-Based Design

The Arduino Nano 33 BLE's limited pin count (20 GPIO, many shared with I2C/SPI) forced creative pin allocation:

- 12 pins required for 4-motor control
- 2 pins for ultrasonic sensor
- 1 pin for servo
- 3 pins for RGB LED
- 1 pin for buzzer
- 4 pins for RF receiver
- Total: 23 pins needed, but only 20 available!

Solution: Careful assignment using analog pins as digital I/O (A0-A7), sharing I2C pins for IMU/light sensor, and ensuring no conflicts.

Learned: Successful embedded systems design requires careful resource allocation. Planning pin usage before building prevents costly redesigns.