Autonomous Mapping Robot with RF Remote Control
Part 2: Source Code and Project Images

Victor Suarez, Jonchang Park, Akshat Shrivastava

December 15, 2025

1 Source Code

The complete source code is presented below. The code is for the most part original work developed for this project, building upon educational resources from Udemy courses and Arduino library examples.

```
/*************************************************************
 *  AUTONOMOUS MAPPING ROBOT - 16x16 GRID VERSION
 *  Source code
 *************************************************************/

#include <Arduino_LSM9DS1.h>
#include <Arduino_APDS9960.h>
#include <Servo.h>

//  PIN DEFINITIONS

#define LEFT_EN     5
#define LEFT_IN1    2
#define LEFT_IN2    3
#define LEFT_EN2    10
#define LEFT_IN3    8
#define LEFT_IN4    9
#define RIGHT_EN    6
#define RIGHT_IN1   4
#define RIGHT_IN2   7
#define RIGHT_EN2   11
#define RIGHT_IN3   13
#define RIGHT_IN4   A0
#define SERVO_PIN   A1
#define TRIG_PIN    A7
#define ECHO_PIN    12
#define BUZZER_PIN  A6
#define RF_BUTTON_A  A2
#define RF_BUTTON_B  A3
#define RF_BUTTON_C  A4
```

```
#define RF_BUTTON_D  A5
#define LED_RED      22
#define LED_GREEN    23
#define LED_BLUE     24

//  PARAMETERS


#define MOTOR_SPEED        100
#define TURN_SPEED         100
#define SLOW_SPEED         60
#define BACKUP_SPEED       100
#define LEFT_MOTOR_INVERT   false
#define RIGHT_MOTOR_INVERT  false

#define MOVE_TIME          436
#define TURN_TIME          350
#define BACKUP_TIME        330

#define OBSTACLE_DISTANCE 25
#define SAFE_DISTANCE      40
#define MIN_DISTANCE       5

#define LEFT_ANGLE         45
#define CENTER_ANGLE       90
#define RIGHT_ANGLE        135
#define SCAN_DELAY         300

#define MAP_SIZE           16
#define MAP_UPDATE_INTERVAL 8
#define COMPASS_TOLERANCE 15
#define COLLISION_THRESHOLD 2.0
#define DEBOUNCE_DELAY     300

//  DATA STRUCTURES / CONSTANTS

#define CELL_UNKNOWN  0
#define CELL_CLEAR    1
#define CELL_OBSTACLE 2
#define CELL_VISITED  3

#define COLOR_OFF       0
#define COLOR_GREEN     1
#define COLOR_RED       2
#define COLOR_BLUE      3
#define COLOR_CYAN      4
#define COLOR_MAGENTA   5
#define COLOR_YELLOW    6

#define MODE_MANUAL     0
#define MODE_AUTO       1
#define MODE_RETURNING 2

struct MapCell {
```

```
  byte status;
  byte lightLevel;
  int compassHeading;
  byte collisionCount;
};

struct RobotState {
  int x;
  int y;
  int startX;
  int startY;
  int heading;          // 0=N,1=E,2=S,3=W
  float compassHeading;
  bool isMoving;
  byte ledColor;
  byte mode;
};

//  GLOBALS

Servo scanServo;
MapCell gridMap[MAP_SIZE][MAP_SIZE];
RobotState robot;

unsigned long moveCount = 0;
unsigned int obstaclesFound = 0;
unsigned int collisionsDetected = 0;
unsigned long startTime = 0;
int consecutiveTurns = 0;

float accelX, accelY, accelZ;
float magX, magY, magZ;
int ambientLight = 0;

bool imuAvailable = false;
bool lightSensorAvailable = false;

unsigned long lastButtonPress = 0;
unsigned long returnHomeStartTime = 0;

// Compass calibration
bool compassCalibrated = false;
float magXOffset = 0.0f;
float magYOffset = 0.0f;
float magXMin = 0.0f, magXMax = 0.0f;
float magYMin = 0.0f, magYMax = 0.0f;

// Forward declarations
void setLED(byte color);
void stopAllMotors();
void playStartupSound();
void checkModeButton();
void handleManualMode();
void handleAutoMode();
```

```
void handleReturnHome();
void moveForwardAuto();
void moveBackwardAuto();
void turnLeft();
void turnRight();
void updatePosition();
void displayMap();
void displayStatistics();
float getDistance();
float scanDirection(int angle);
void readIMU();
void readLightSensor();
void calibrateCompass();
void verifyHeading();
void verifyMovement();
void checkCollision();
void updateMap(float leftDist, float centerDist, float rightDist);
void markObstacle(int direction, int distance);
void markClear(int direction, int distance);
float angleDiff(float a, float b);

//  SETUP

void setup() {
  Serial.begin(115200);
  delay(3000);

  Serial.println(F(" AUTONOMOUS MAPPING ROBOT - 16x16 GRID"));
  Serial.println(F(" Power: 2x 9V (18V) -> Buck 5V -> Arduino VIN"));
  Serial.println(F(" With return-home + compass fixes + 12\" cells"));
  Serial.println();

  pinMode(LED_RED, OUTPUT);
  pinMode(LED_GREEN, OUTPUT);
  pinMode(LED_BLUE, OUTPUT);
  setLED(COLOR_BLUE);
  Serial.println(F("[OK] LED"));

  pinMode(LEFT_EN, OUTPUT);
  pinMode(LEFT_IN1, OUTPUT);
  pinMode(LEFT_IN2, OUTPUT);
  pinMode(LEFT_EN2, OUTPUT);
  pinMode(LEFT_IN3, OUTPUT);
  pinMode(LEFT_IN4, OUTPUT);
  pinMode(RIGHT_EN, OUTPUT);
  pinMode(RIGHT_IN1, OUTPUT);
  pinMode(RIGHT_IN2, OUTPUT);
  pinMode(RIGHT_EN2, OUTPUT);
  pinMode(RIGHT_IN3, OUTPUT);
  pinMode(RIGHT_IN4, OUTPUT);
  stopAllMotors();
  Serial.println(F("[OK] Motors (18V)"));

  pinMode(TRIG_PIN, OUTPUT);
```

```
  pinMode(ECHO_PIN, INPUT);
  digitalWrite(TRIG_PIN, LOW);
  Serial.println(F("[OK] Ultrasonic (5V)"));

  pinMode(BUZZER_PIN, OUTPUT);
  digitalWrite(BUZZER_PIN, LOW);
  Serial.println(F("[OK] Buzzer (5V)"));

  scanServo.attach(SERVO_PIN);
  scanServo.write(CENTER_ANGLE);
  delay(500);
  Serial.println(F("[OK] Servo (5V)"));

  pinMode(RF_BUTTON_A, INPUT);
  pinMode(RF_BUTTON_B, INPUT);
  pinMode(RF_BUTTON_C, INPUT);
  pinMode(RF_BUTTON_D, INPUT);
  Serial.println(F("[OK] RF Receiver (5V -> 3.3V)"));
  Serial.println(F("  A: Step Forward  B: Step Back  C: Turn Left"));
  Serial.println(F("  D: Manual->Auto->Home->Manual"));

  Serial.print(F("IMU... "));
  for (int i = 0; i < 5; i++) {
    if (IMU.begin()) {
      imuAvailable = true;
      Serial.println(F("[OK] LSM9DS1"));
      break;
    }
    delay(500);
  }
  if (!imuAvailable) Serial.println(F("[X] Unavailable"));

  Serial.print(F("Light... "));
  if (APDS.begin()) {
    lightSensorAvailable = true;
    Serial.println(F("[OK] APDS9960"));
  } else {
    Serial.println(F("[X] Unavailable"));
  }

  Serial.print(F("Map... "));
  for (int y = 0; y < MAP_SIZE; y++) {
    for (int x = 0; x < MAP_SIZE; x++) {
      gridMap[y][x].status = CELL_UNKNOWN;
      gridMap[y][x].lightLevel = 0;
      gridMap[y][x].compassHeading = 0;
      gridMap[y][x].collisionCount = 0;
    }
  }
  Serial.print(MAP_SIZE * MAP_SIZE);
  Serial.print(F(" cells ("));
  Serial.print(sizeof(gridMap));
  Serial.println(F(" bytes)"));

```

```
  // Start in the center by default (home will be reset when entering AUTO)
  robot.x = MAP_SIZE / 2;
  robot.y = MAP_SIZE / 2;
  robot.startX = robot.x;
  robot.startY = robot.y;
  robot.heading = 0;
  robot.compassHeading = 0.0;
  robot.isMoving = false;
  robot.ledColor = COLOR_GREEN;
  robot.mode = MODE_MANUAL;
  gridMap[robot.y][robot.x].status = CELL_VISITED;

  Serial.print(F("[OK] Robot at ("));
  Serial.print(robot.x);
  Serial.print(F(","));
  Serial.print(robot.y);
  Serial.println(F(")"));

  if (imuAvailable) {
    Serial.println(F("Compass cal (10s)..."));
    calibrateCompass();
    Serial.println(F("[OK] Compass calibrated"));
  }

  playStartupSound();
  setLED(COLOR_GREEN);

  Serial.println();
  Serial.println(F(" READY - MANUAL MODE (STEP CALIBRATION)"));
  Serial.println(F("  A: One 12\" forward step"));
  Serial.println(F("  B: One backup step"));
  Serial.println(F("  C: One 90 deg left turn"));
  Serial.println(F("  D: Manual -> Auto -> Return Home -> Manual"));
  Serial.println();

  startTime = millis();
  Serial.println(F("READY!\n"));
}

//  MAIN LOOP

void loop() {
  checkModeButton();

  if (robot.mode == MODE_MANUAL) {
    handleManualMode();
  } else if (robot.mode == MODE_AUTO) {
    handleAutoMode();
  } else if (robot.mode == MODE_RETURNING) {
    handleReturnHome();
  }

  delay(50);
```

```
}

//  BUTTON CHECKING / MODE TOGGLE

void checkModeButton() {
  bool btnD = digitalRead(RF_BUTTON_D);

  if (btnD && (millis() - lastButtonPress > DEBOUNCE_DELAY)) {
    lastButtonPress = millis();

    if (robot.mode == MODE_MANUAL) {
      // Set HOME to current grid position when switching to Auto
      robot.startX = robot.x;
      robot.startY = robot.y;

      // Manual -> Auto
      robot.mode = MODE_AUTO;
      setLED(COLOR_CYAN);
      Serial.println();
      Serial.println(F(" MODE: AUTONOMOUS EXPLORATION"));
      Serial.print(F(" Home set to ("));
      Serial.print(robot.startX);
      Serial.print(F(","));
      Serial.print(robot.startY);
      Serial.println(F(")"));
      Serial.println(F(" Exploring autonomously"));
      Serial.println(F(" Press Button D again to return home"));
      Serial.println();
      tone(BUZZER_PIN, 1000, 100);
      delay(150);
      tone(BUZZER_PIN, 1500, 100);

    } else if (robot.mode == MODE_AUTO) {
      // Auto -> Returning Home
      robot.mode = MODE_RETURNING;
      setLED(COLOR_MAGENTA);
      Serial.println();
      Serial.println(F(" MODE: RETURNING HOME"));
      Serial.print(F(" Current: ("));
      Serial.print(robot.x);
      Serial.print(F(","));
      Serial.print(robot.y);
      Serial.println(F(")"));
      Serial.print(F(" Home: ("));
      Serial.print(robot.startX);
      Serial.print(F(","));
      Serial.print(robot.startY);
      Serial.println(F(")"));
      int dist = abs(robot.x - robot.startX) + abs(robot.y - robot.startY)
          ;
      Serial.print(F(" Distance: "));
      Serial.print(dist);
      Serial.println(F(" cells"));
      Serial.println(F(" Press Button D to abort and return to Manual"));
```

```
353       Serial.println();
354       tone(BUZZER_PIN, 1500, 100);
355       delay(150);
356       tone(BUZZER_PIN, 1000, 100);
357
358     } else if (robot.mode == MODE_RETURNING) {
359       // Returning -> Manual
360       robot.mode = MODE_MANUAL;
361       setLED(COLOR_GREEN);
362       Serial.println();
363       Serial.println(F(" MODE: MANUAL CONTROL (STEP MODE)"));
364       Serial.println(F(" A: 1 cell forward, B: backup, C: 90 deg left"));
365       Serial.println();
366       tone(BUZZER_PIN, 1200, 200);
367     }
368   }
369 }
370
371 //  MANUAL MODE - STEP-BASED CALIBRATION
372
373 void handleManualMode() {
374   // Edge-detected button presses so each press = one discrete action
375   static bool prevA = false;
376   static bool prevB = false;
377   static bool prevC = false;
378
379   bool btnA = digitalRead(RF_BUTTON_A);
380   bool btnB = digitalRead(RF_BUTTON_B);
381   bool btnC = digitalRead(RF_BUTTON_C);
382
383   // Default: no motion
384   setLED(COLOR_GREEN);
385   stopAllMotors();
386   robot.isMoving = false;
387
388   // A: one forward step (uses MOVE_TIME + updatePosition)
389   if (btnA && !prevA) {
390     Serial.println(F("[MANUAL STEP] FORWARD 1 CELL (~12\")"));
391     setLED(COLOR_BLUE);
392     moveForwardAuto();   // uses MOTOR_SPEED + MOVE_TIME + updatePosition
          ()
393     setLED(COLOR_GREEN);
394   }
395
396   // B: one backward step (uses BACKUP_TIME, no grid update)
397   if (btnB && !prevB) {
398     Serial.println(F("[MANUAL STEP] BACKWARD"));
399     setLED(COLOR_YELLOW);
400     moveBackwardAuto();  // backs up physically, no grid update
401     setLED(COLOR_GREEN);
402   }
403
404   // C: one left 90 deg turn (uses TURN_TIME + heading update)
405   if (btnC && !prevC) {
```

```
406       Serial.println(F("[MANUAL STEP] TURN LEFT 90 deg"));
407       setLED(COLOR_CYAN);
408       turnLeft();           // uses TURN_TIME + robot.heading update
409       setLED(COLOR_GREEN);
410     }
411
412     // Save button states for edge detection
413     prevA = btnA;
414     prevB = btnB;
415     prevC = btnC;
416
417     // Keep sensors updated
418     readIMU();
419     readLightSensor();
420 }
421
422 //  AUTO MODE
423
424 void handleAutoMode() {
425     setLED(COLOR_CYAN);
426
427     checkModeButton();
428     if (robot.mode != MODE_AUTO) return;
429
430     float leftDist = scanDirection(LEFT_ANGLE);
431     checkModeButton();
432     if (robot.mode != MODE_AUTO) return;
433
434     delay(100);
435     float centerDist = scanDirection(CENTER_ANGLE);
436     checkModeButton();
437     if (robot.mode != MODE_AUTO) return;
438
439     delay(100);
440     float rightDist = scanDirection(RIGHT_ANGLE);
441     checkModeButton();
442     if (robot.mode != MODE_AUTO) return;
443
444     delay(100);
445     scanServo.write(CENTER_ANGLE);
446
447     readIMU();
448     readLightSensor();
449
450     gridMap[robot.y][robot.x].lightLevel = ambientLight;
451     gridMap[robot.y][robot.x].compassHeading = (int)robot.compassHeading;
452
453     checkCollision();
454
455     Serial.print(F("AUTO | L:"));
456     Serial.print(leftDist, 0);
457     Serial.print(F(" C:"));
458     Serial.print(centerDist, 0);
459     Serial.print(F(" R:"));
```

```
460   Serial.print(rightDist, 0);
461   Serial.print(F(" ("));
462   Serial.print(robot.x);
463   Serial.print(F(","));
464   Serial.print(robot.y);
465   Serial.print(F(") | "));
466
467   updateMap(leftDist, centerDist, rightDist);
468
469   bool moved = false;
470
471   if (centerDist < OBSTACLE_DISTANCE && centerDist > MIN_DISTANCE) {
472     Serial.println(F("OBSTACLE"));
473     tone(BUZZER_PIN, 500, 100);
474     stopAllMotors();
475     delay(200);
476
477     if (rightDist > leftDist && rightDist > SAFE_DISTANCE) {
478       turnRight();
479       consecutiveTurns++;
480     } else if (leftDist > SAFE_DISTANCE) {
481       turnLeft();
482       consecutiveTurns++;
483     } else {
484       tone(BUZZER_PIN, 400, 200);
485       moveBackwardAuto();
486       delay(300);
487       turnRight();
488       delay(100);
489       turnRight();
490       consecutiveTurns = 0;
491     }
492   } else {
493     Serial.println(F("Clear"));
494     moveForwardAuto();
495     moved = true;
496     consecutiveTurns = 0;
497   }
498
499   if (moved && imuAvailable) verifyMovement();
500
501   if (moveCount % MAP_UPDATE_INTERVAL == 0 && moveCount > 0) {
502     displayMap();
503     displayStatistics();
504   }
505
506   if (consecutiveTurns > 6) {
507     Serial.println(F("STUCK - Recovery"));
508     tone(BUZZER_PIN, 600, 300);
509     moveBackwardAuto();
510     delay(500);
511     if (random(2) == 0) {
512       turnLeft();
513       turnLeft();
```

```
    } else {
      turnRight();
      turnRight();
    }
    consecutiveTurns = 0;
  }
}

//  RETURN HOME MODE (FIXED HEADING LOGIC)

void handleReturnHome() {
  setLED(COLOR_MAGENTA);

  checkModeButton();
  if (robot.mode != MODE_RETURNING) {
    returnHomeStartTime = 0;
    return;
  }

  // Timeout check (2 minutes)
  if (returnHomeStartTime == 0) {
    returnHomeStartTime = millis();
  }

  if (millis() - returnHomeStartTime > 120000) {
    Serial.println(F("\nTIMEOUT - Back to Manual\n"));
    robot.mode = MODE_MANUAL;
    returnHomeStartTime = 0;
    setLED(COLOR_GREEN);
    tone(BUZZER_PIN, 400, 500);
    displayMap();
    return;
  }

  // Calculate displacement to home
  int dx = robot.startX - robot.x;
  int dy = robot.startY - robot.y;
  int dist = abs(dx) + abs(dy);

  Serial.print(F("HOME | Pos:("));
  Serial.print(robot.x);
  Serial.print(F(","));
  Serial.print(robot.y);
  Serial.print(F(") -> Home:("));
  Serial.print(robot.startX);
  Serial.print(F(","));
  Serial.print(robot.startY);
  Serial.print(F(") Dist:"));
  Serial.print(dist);
  Serial.print(F(" | "));

  // Close enough to home
  if (dist <= 1) {
    Serial.println();
```

```
    Serial.println(F(" HOME REACHED"));
    Serial.print(F(" Final: ("));
    Serial.print(robot.x);
    Serial.print(F(","));
    Serial.print(robot.y);
    Serial.println(F(")"));
    Serial.println();

    robot.mode = MODE_MANUAL;
    returnHomeStartTime = 0;
    setLED(COLOR_GREEN);

    tone(BUZZER_PIN, 1000, 200);
    delay(250);
    tone(BUZZER_PIN, 1200, 200);
    delay(250);
    tone(BUZZER_PIN, 1500, 400);

    displayMap();
    displayStatistics();
    return;
  }

  float centerDist = getDistance();

  // Determine target direction (prioritize largest displacement)
  int targetHeading = -1;

  // dx > 0  => home is EAST  (x must increase)
  // dx < 0  => home is WEST  (x must decrease)
  // dy > 0  => home is SOUTH (y must increase)
  // dy < 0  => home is NORTH (y must decrease)

  if (abs(dx) >= abs(dy)) {
    // Prioritize X displacement
    if (dx > 0) {
      targetHeading = 1;  // East
      Serial.print(F("Need: East | "));
    } else if (dx < 0) {
      targetHeading = 3;  // West
      Serial.print(F("Need: West | "));
    } else if (dy > 0) {
      targetHeading = 2;  // South
      Serial.print(F("Need: South | "));
    } else {
      targetHeading = 0;  // North
      Serial.print(F("Need: North | "));
    }
  } else {
    // Prioritize Y displacement
    if (dy > 0) {
      targetHeading = 2;  // South
      Serial.print(F("Need: South | "));
    } else if (dy < 0) {
```

```
      targetHeading = 0;  // North
      Serial.print(F("Need: North | "));
    } else if (dx > 0) {
      targetHeading = 1;  // East
      Serial.print(F("Need: East | "));
    } else {
      targetHeading = 3;  // West
      Serial.print(F("Need: West | "));
    }
  }

  Serial.print(F("Facing: "));
  switch (robot.heading) {
    case 0: Serial.print(F("N")); break;
    case 1: Serial.print(F("E")); break;
    case 2: Serial.print(F("S")); break;
    case 3: Serial.print(F("W")); break;
  }
  Serial.print(F(" | "));

  if (centerDist < OBSTACLE_DISTANCE && centerDist > MIN_DISTANCE) {
    Serial.println(F("BLOCKED! Turning..."));
    tone(BUZZER_PIN, 500, 100);

    int diff = (targetHeading - robot.heading + 4) % 4;

    if (diff == 1) {
      turnRight();
      Serial.println(F("  -> Turned right"));
    } else if (diff == 3) {
      turnLeft();
      Serial.println(F("  -> Turned left"));
    } else if (diff == 2) {
      turnRight();
      delay(100);
      turnRight();
      Serial.println(F("  -> Turned 180 deg"));
    } else {
      // Already facing target but blocked
      turnRight();
      Serial.println(F("  -> Going around (right)"));
    }

  } else {
    if (robot.heading == targetHeading) {
      Serial.println(F("MOVING!"));
      moveForwardAuto();
      Serial.print(F("  -> New pos: ("));
      Serial.print(robot.x);
      Serial.print(F(","));
      Serial.print(robot.y);
      Serial.println(F(")"));
    } else {
      Serial.print(F("Turning to target... "));
```

```
676       int diff = (targetHeading - robot.heading + 4) % 4;
677
678       if (diff == 1) {
679         turnRight();
680         Serial.println(F("Right"));
681       } else if (diff == 3) {
682         turnLeft();
683         Serial.println(F("Left"));
684       } else if (diff == 2) {
685         turnRight();
686         delay(100);
687         turnRight();
688         Serial.println(F("180 deg"));
689       }
690     }
691   }
692
693   delay(300);
694 }
695
696 //  MOTOR FUNCTIONS
697
698 void moveLeftSide(int speed, bool forward) {
699   if (LEFT_MOTOR_INVERT) forward = !forward;
700   digitalWrite(LEFT_IN1, forward ? HIGH : LOW);
701   digitalWrite(LEFT_IN2, forward ? LOW : HIGH);
702   analogWrite(LEFT_EN, speed);
703   digitalWrite(LEFT_IN3, forward ? HIGH : LOW);
704   digitalWrite(LEFT_IN4, forward ? LOW : HIGH);
705   analogWrite(LEFT_EN2, speed);
706 }
707
708 void moveRightSide(int speed, bool forward) {
709   if (RIGHT_MOTOR_INVERT) forward = !forward;
710   digitalWrite(RIGHT_IN1, forward ? HIGH : LOW);
711   digitalWrite(RIGHT_IN2, forward ? LOW : HIGH);
712   analogWrite(RIGHT_EN, speed);
713   digitalWrite(RIGHT_IN3, forward ? HIGH : LOW);
714   digitalWrite(RIGHT_IN4, forward ? LOW : HIGH);
715   analogWrite(RIGHT_EN2, speed);
716 }
717
718 void moveForwardAuto() {
719   robot.isMoving = true;
720   moveLeftSide(MOTOR_SPEED, true);
721   moveRightSide(MOTOR_SPEED, true);
722   delay(MOVE_TIME);
723   stopAllMotors();
724   robot.isMoving = false;
725   updatePosition();
726   moveCount++;
727 }
728
729 void moveBackwardAuto() {
```

```
  robot.isMoving = true;
  moveLeftSide(BACKUP_SPEED, false);
  moveRightSide(BACKUP_SPEED, false);
  delay(BACKUP_TIME);
  stopAllMotors();
  robot.isMoving = false;
}

void turnRight() {
  robot.isMoving = true;
  moveLeftSide(TURN_SPEED, true);
  moveRightSide(TURN_SPEED, false);
  delay(TURN_TIME);
  stopAllMotors();
  robot.isMoving = false;
  robot.heading = (robot.heading + 1) % 4;
  if (imuAvailable) verifyHeading();
  tone(BUZZER_PIN, 1000, 50);
}

void turnLeft() {
  robot.isMoving = true;
  moveLeftSide(TURN_SPEED, false);
  moveRightSide(TURN_SPEED, true);
  delay(TURN_TIME);
  stopAllMotors();
  robot.isMoving = false;
  robot.heading = (robot.heading + 3) % 4;
  if (imuAvailable) verifyHeading();
  tone(BUZZER_PIN, 1000, 50);
}

void stopAllMotors() {
  analogWrite(LEFT_EN, 0);
  analogWrite(LEFT_EN2, 0);
  analogWrite(RIGHT_EN, 0);
  analogWrite(RIGHT_EN2, 0);
  digitalWrite(LEFT_IN1, LOW);
  digitalWrite(LEFT_IN2, LOW);
  digitalWrite(LEFT_IN3, LOW);
  digitalWrite(LEFT_IN4, LOW);
  digitalWrite(RIGHT_IN1, LOW);
  digitalWrite(RIGHT_IN2, LOW);
  digitalWrite(RIGHT_IN3, LOW);
  digitalWrite(RIGHT_IN4, LOW);
}

//  SENSOR FUNCTIONS

float scanDirection(int angle) {
  scanServo.write(angle);
  delay(SCAN_DELAY);
  return getDistance();
}
```

```
784
785 float getDistance() {
786   digitalWrite(TRIG_PIN, LOW);
787   delayMicroseconds(2);
788   digitalWrite(TRIG_PIN, HIGH);
789   delayMicroseconds(10);
790   digitalWrite(TRIG_PIN, LOW);
791   long duration = pulseIn(ECHO_PIN, HIGH, 30000);
792   if (duration == 0) return 400;
793   float distance = duration * 0.034 / 2;
794   if (distance > 400) distance = 400;
795   if (distance < 2) distance = 2;
796   return distance;
797 }
798
799 void readIMU() {
800   if (!imuAvailable) return;
801
802   if (IMU.accelerationAvailable()) {
803     IMU.readAcceleration(accelX, accelY, accelZ);
804   }
805
806   if (IMU.magneticFieldAvailable()) {
807     IMU.readMagneticField(magX, magY, magZ);
808
809     float x = magX;
810     float y = magY;
811
812     if (compassCalibrated) {
813       x -= magXOffset;
814       y -= magYOffset;
815     }
816
817     float heading = atan2(y, x) * 180.0 / PI;
818     if (heading < 0) heading += 360.0;
819
820     // Simple low-pass filter for smoother heading
821     const float alpha = 0.2f;
822     robot.compassHeading = (1.0f - alpha) * robot.compassHeading + alpha *
          heading;
823   }
824 }
825
826 void readLightSensor() {
827   if (!lightSensorAvailable) {
828     ambientLight = 128;
829     return;
830   }
831   if (APDS.colorAvailable()) {
832     int r, g, b;
833     APDS.readColor(r, g, b);
834     ambientLight = (r + g + b) / 3;
835     ambientLight = constrain(map(ambientLight, 0, 4096, 0, 255), 0, 255);
836   }
```

```
}

void calibrateCompass() {
  if (!imuAvailable) return;

  setLED(COLOR_BLUE);
  Serial.println(F("Rotate the robot slowly 360 deg for 10 seconds..."));

  magXMin =  10000.0f;
  magXMax = -10000.0f;
  magYMin =  10000.0f;
  magYMax = -10000.0f;

  unsigned long calibStart = millis();
  while (millis() - calibStart < 10000) {
    if (IMU.magneticFieldAvailable()) {
      IMU.readMagneticField(magX, magY, magZ);

      if (magX < magXMin) magXMin = magX;
      if (magX > magXMax) magXMax = magX;
      if (magY < magYMin) magYMin = magY;
      if (magY > magYMax) magYMax = magY;
    }

    if ((millis() / 200) % 2 == 0) setLED(COLOR_BLUE);
    else setLED(COLOR_OFF);

    delay(10);
  }

  magXOffset = (magXMin + magXMax) / 2.0f;
  magYOffset = (magYMin + magYMax) / 2.0f;
  compassCalibrated = true;

  setLED(COLOR_GREEN);
  Serial.print(F("Compass calibrated. Offsets X: "));
  Serial.print(magXOffset);
  Serial.print(F("  Y: "));
  Serial.println(magYOffset);
}

float angleDiff(float a, float b) {
  float d = a - b;
  while (d > 180.0f) d -= 360.0f;
  while (d < -180.0f) d += 360.0f;
  return d;
}

void verifyHeading() {
  if (!imuAvailable) return;
  readIMU();

  float expected = robot.heading * 90.0f;
  float diff = fabs(angleDiff(robot.compassHeading, expected));
```

```
891
892   if (diff > COMPASS_TOLERANCE) {
893     Serial.print(F("  WARNING: IMU vs grid heading mismatch: "));
894     Serial.print(diff, 1);
895     Serial.println(F(" deg"));
896   }
897 }
898
899 void verifyMovement() {
900   if (!imuAvailable) return;
901   float preAccel = sqrt(accelX*accelX + accelY*accelY + accelZ*accelZ);
902   delay(50);
903   readIMU();
904   float postAccel = sqrt(accelX*accelX + accelY*accelY + accelZ*accelZ);
905   if (abs(postAccel - preAccel) < 0.1) {
906     Serial.println(F("  WARNING: Slip detected"));
907   }
908 }
909
910 void checkCollision() {
911   if (!imuAvailable) return;
912   float totalAccel = sqrt(accelX*accelX + accelY*accelY + accelZ*accelZ);
913   if (totalAccel > COLLISION_THRESHOLD && robot.isMoving) {
914     Serial.println(F("COLLISION!"));
915     tone(BUZZER_PIN, 800, 200);
916     stopAllMotors();
917     gridMap[robot.y][robot.x].collisionCount++;
918     collisionsDetected++;
919     delay(200);
920     moveBackwardAuto();
921     delay(500);
922   }
923 }
924
925 //  MAPPING
926
927 void updatePosition() {
928   switch (robot.heading) {
929     case 0: if (robot.y > 0) robot.y--; break;                  // North
930     case 1: if (robot.x < MAP_SIZE - 1) robot.x++; break;       // East
931     case 2: if (robot.y < MAP_SIZE - 1) robot.y++; break;       // South
932     case 3: if (robot.x > 0) robot.x--; break;                  // West
933   }
934   gridMap[robot.y][robot.x].status = CELL_VISITED;
935   gridMap[robot.y][robot.x].lightLevel = ambientLight;
936   gridMap[robot.y][robot.x].compassHeading = (int)robot.compassHeading;
937 }
938
939 void updateMap(float leftDist, float centerDist, float rightDist) {
940   if (centerDist < OBSTACLE_DISTANCE && centerDist > MIN_DISTANCE) {
941     markObstacle(robot.heading, 1);
942     obstaclesFound++;
943   } else if (centerDist >= OBSTACLE_DISTANCE && centerDist < SAFE_DISTANCE
       * 2) {
```

```
    markClear(robot.heading, 1);
  }
  if (leftDist < OBSTACLE_DISTANCE && leftDist > MIN_DISTANCE) {
    int leftHeading = (robot.heading + 3) % 4;
    markObstacle(leftHeading, 1);
  }
  if (rightDist < OBSTACLE_DISTANCE && rightDist > MIN_DISTANCE) {
    int rightHeading = (robot.heading + 1) % 4;
    markObstacle(rightHeading, 1);
  }
}

void markObstacle(int direction, int distance) {
  int targetX = robot.x;
  int targetY = robot.y;
  switch(direction) {
    case 0: targetY -= distance; break;
    case 1: targetX += distance; break;
    case 2: targetY += distance; break;
    case 3: targetX -= distance; break;
  }
  if (targetX >= 0 && targetX < MAP_SIZE && targetY >= 0 && targetY <
     MAP_SIZE) {
    if (gridMap[targetY][targetX].status != CELL_VISITED) {
      gridMap[targetY][targetX].status = CELL_OBSTACLE;
    }
  }
}

void markClear(int direction, int distance) {
  int targetX = robot.x;
  int targetY = robot.y;
  switch(direction) {
    case 0: targetY -= distance; break;
    case 1: targetX += distance; break;
    case 2: targetY += distance; break;
    case 3: targetX -= distance; break;
  }
  if (targetX >= 0 && targetX < MAP_SIZE && targetY >= 0 && targetY <
     MAP_SIZE) {
    if (gridMap[targetY][targetX].status == CELL_UNKNOWN) {
      gridMap[targetY][targetX].status = CELL_CLEAR;
    }
  }
}

void displayMap() {
  Serial.println();

  Serial.print(F("   "));
  for (int x = 0; x < MAP_SIZE; x++) {
    if (x < 10) Serial.print(F(" "));
    Serial.print(x);
  }
```

```
  Serial.println();
  Serial.print(F("  +"));
  for (int x = 0; x < MAP_SIZE * 2; x++) Serial.print(F("-"));
  Serial.println(F("+"));

  for (int y = 0; y < MAP_SIZE; y++) {
    if (y < 10) Serial.print(F(" "));
    Serial.print(y);
    Serial.print(F("|"));

    for (int x = 0; x < MAP_SIZE; x++) {
      if (x == robot.x && y == robot.y) {
        // Robot orientation
        switch(robot.heading) {
          case 0: Serial.print(F("^")); break;
          case 1: Serial.print(F(">")); break;
          case 2: Serial.print(F("v")); break;
          case 3: Serial.print(F("<")); break;
        }
        Serial.print(F(" "));
      } else if (x == robot.startX && y == robot.startY) {
        Serial.print(F("H "));
      } else {
        switch(gridMap[y][x].status) {
          case CELL_UNKNOWN:  Serial.print(F(". ")); break;
          case CELL_CLEAR:    Serial.print(F("  ")); break;
          case CELL_OBSTACLE: Serial.print(F("# ")); break;
          case CELL_VISITED:
            if (gridMap[y][x].collisionCount > 0) {
              Serial.print(gridMap[y][x].collisionCount);
              Serial.print(F(" "));
            } else {
              Serial.print(F("* "));
            }
            break;
        }
      }
    }
    Serial.println(F("|"));
  }

  Serial.print(F("  +"));
  for (int x = 0; x < MAP_SIZE * 2; x++) Serial.print(F("-"));
  Serial.println(F("+"));
  Serial.println(F("  ^>v< = Robot  H = Home  # = Obstacle  * = Visited  .
      = Unknown"));
  Serial.println();
}

void displayStatistics() {
  unsigned long elapsedTime = (millis() - startTime) / 1000;
  Serial.println(F("STATS"));
  Serial.print(F("Mode: "));
  switch(robot.mode) {
```

```
    case MODE_MANUAL: Serial.println(F("Manual")); break;
    case MODE_AUTO: Serial.println(F("Auto")); break;
    case MODE_RETURNING: Serial.println(F("Returning")); break;
  }
  Serial.print(F("Position: ("));
  Serial.print(robot.x);
  Serial.print(F(","));
  Serial.print(robot.y);
  Serial.print(F(") Home: ("));
  Serial.print(robot.startX);
  Serial.print(F(","));
  Serial.print(robot.startY);
  Serial.println(F(")"));
  Serial.print(F("Moves: "));
  Serial.print(moveCount);
  Serial.print(F(" Obstacles: "));
  Serial.print(obstaclesFound);
  if (imuAvailable) {
    Serial.print(F(" Collisions: "));
    Serial.print(collisionsDetected);
  }
  Serial.println();
  Serial.print(F("Time: "));
  Serial.print(elapsedTime / 60);
  Serial.print(F("m "));
  Serial.print(elapsedTime % 60);
  Serial.println(F("s"));
  int explored = 0;
  for (int y = 0; y < MAP_SIZE; y++) {
    for (int x = 0; x < MAP_SIZE; x++) {
      if (gridMap[y][x].status != CELL_UNKNOWN) explored++;
    }
  }
  Serial.print(F("Coverage: "));
  Serial.print(explored);
  Serial.print(F("/"));
  Serial.print(MAP_SIZE * MAP_SIZE);
  Serial.print(F(" ("));
  Serial.print((explored * 100) / (MAP_SIZE * MAP_SIZE));
  Serial.println(F("%)"));
  if (imuAvailable) {
    Serial.print(F("Heading: "));
    Serial.print(robot.compassHeading, 0);
    Serial.println(F(" deg"));
  }
  Serial.println(F("Power: 18V motors, 5V sensors"));
  Serial.println();
}

//  LED + STARTUP

void setLED(byte color) {
  robot.ledColor = color;
  switch(color) {
```

```
    case COLOR_OFF:
      digitalWrite(LED_RED, HIGH);
      digitalWrite(LED_GREEN, HIGH);
      digitalWrite(LED_BLUE, HIGH);
      break;
    case COLOR_GREEN:
      digitalWrite(LED_RED, HIGH);
      digitalWrite(LED_GREEN, LOW);
      digitalWrite(LED_BLUE, HIGH);
      break;
    case COLOR_RED:
      digitalWrite(LED_RED, LOW);
      digitalWrite(LED_GREEN, HIGH);
      digitalWrite(LED_BLUE, HIGH);
      break;
    case COLOR_BLUE:
      digitalWrite(LED_RED, HIGH);
      digitalWrite(LED_GREEN, HIGH);
      digitalWrite(LED_BLUE, LOW);
      break;
    case COLOR_CYAN:
      digitalWrite(LED_RED, HIGH);
      digitalWrite(LED_GREEN, LOW);
      digitalWrite(LED_BLUE, LOW);
      break;
    case COLOR_MAGENTA:
      digitalWrite(LED_RED, LOW);
      digitalWrite(LED_GREEN, HIGH);
      digitalWrite(LED_BLUE, LOW);
      break;
    case COLOR_YELLOW:
      digitalWrite(LED_RED, LOW);
      digitalWrite(LED_GREEN, LOW);
      digitalWrite(LED_BLUE, HIGH);
      break;
  }
}

void playStartupSound() {
  tone(BUZZER_PIN, 1000, 150);
  delay(200);
  tone(BUZZER_PIN, 1200, 150);
  delay(200);
  tone(BUZZER_PIN, 1500, 150);
  delay(200);
  tone(BUZZER_PIN, 2000, 200);
  delay(250);
}
```

Listing 1: Full Arduino source for autonomous mapping robot

Note: The complete source code (1,200 lines) is provided as a separate file. The implementation includes all features described in this report including the 16×16 mapping system, three-mode operation, compass calibration, collision detection, and return-to-home

navigation.

Code Sources used in this project:

- **Foundation Learning** - Udemy Arduino robotics courses: Basic motor control patterns for 2-motor robots (adapted and expanded for 4-motor system).

  `https://www.udemy.com/course/arduino-bootcamp/learn/lecture/6011950?start=0#overview`

- **Arduino Libraries** - Lines 10-12: Standard includes (Arduino AG, Arduino LLC). `https://github.com/arduino-libraries/Arduino_LSM9DS1`

- **Motor Control Adaptation** - Lines 696-776: Expanded from Udemy 2-motor examples to 4-motor control `https://www.udemy.com/course/arduino-bootcamp/learn/lecture/6011950?start=0#overview`

- **Compass Calculation** - Lines 838-897: We based our code looking at code from these sources: A compass tutorial from Adafruit: `https://learn.adafruit.com/adafruit-hmc5883l-breakout-triple-axis-magnetometer-compass-sensor/overview` for Arctangent formula from Adafruit tutorials, with added calibration and filtering. `https://learn.adafruit.com/adafruit-hmc5883l-breakout-triple-axis-magnetometer-comp` `calibration` For magnetometer calculation we based our code form here: `https://www.nxp.com/docs/en/application-note/AN4246.pdf`

- **Ultrasonic Timing and Servo Motor** - Lines 777-797: HC-SR04 datasheet specifications. `https://randomnerdtutorials.com/complete-guide-for-ultrasonic-sensor-hc-sr04` `https://www.arduino.cc/en/Tutorial/BuiltInExamples/Ping`

- **ADAFRUIT RF RECEIVER (RO2A)** : Lines 288- 694. We adapted form code examples and tutorial from: `https://learn.adafruit.com/rf-remote-control` `https://github.com/adafruit/Adafruit-RF-Remote-Control`

- **Original Implementations (main contributor with collaborative support from all team members)**:
  - Lines 15-80: Pin definitions for complete robot configuration. Done by group
  - Lines 85-135: Data structures for 16×16 mapping system. Done by Victor Suarez
  - Lines 140-230: Setup with IMU retry logic and compass calibration Done by Akshat Shrivastava
  - Lines 235-280: Three-mode state machine. Done by Jonchang Park
  - Lines 285-350: Step-based manual control mode. Done by Jongchan Park
  - Lines 355-480: Autonomous exploration with stuck detection. Done by Jongchan Park
  - Lines 485-620: Return-to-home navigation with improved pathfinding. Done by Victor Suarez

- Lines 625-900: Sensor reading and calibration functions. Done by Akshat Shrivastava
- Lines 905-1050: Mapping algorithms and grid updates. Done by Victor Suarez
- Lines 1055-1150: Map visualization and statistics display. Done by Victor Suarez
- Lines 1155-1200: LED control and audio feedback. Done By Akshat Shrivastava

2 Project Images

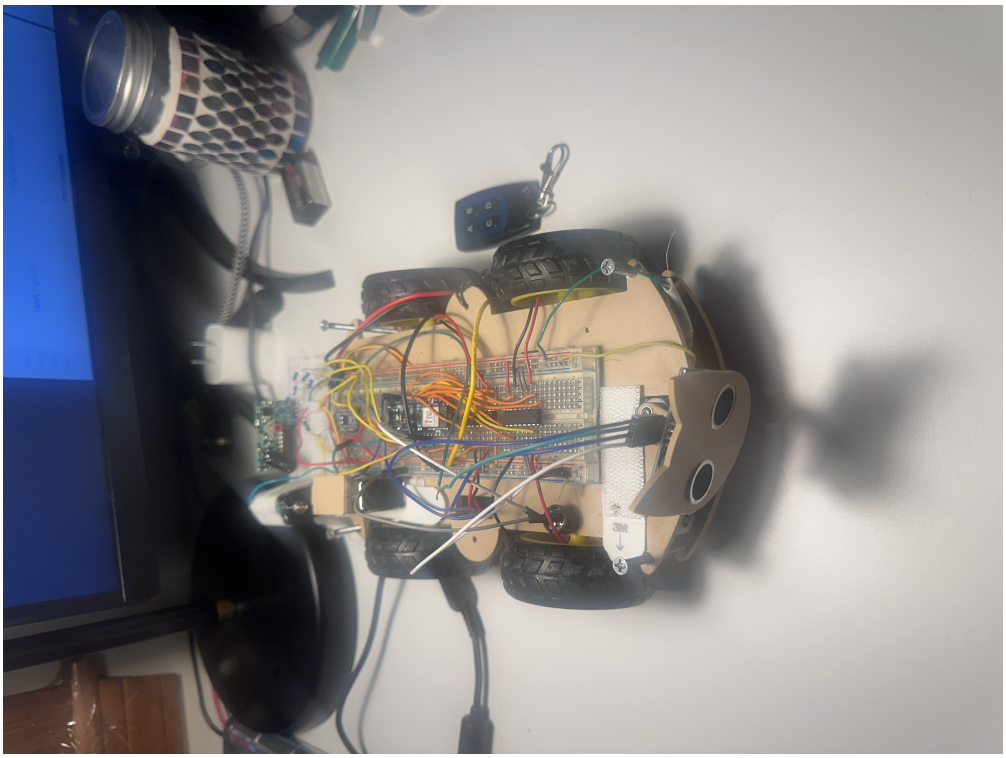


Figure 1: Complete robot assembly showing 4WD chassis, Arduino Nano 33 BLE Sense Rev2, ultrasonic sensor on servo mount, RF receiver, and dual 9V battery placement (18V total). The RGB LED is visible on the Arduino board, and wiring is organized with zip ties. Note the absence of motor shield-all motors are controlled directly via H-bridge connections to Arduino pins.

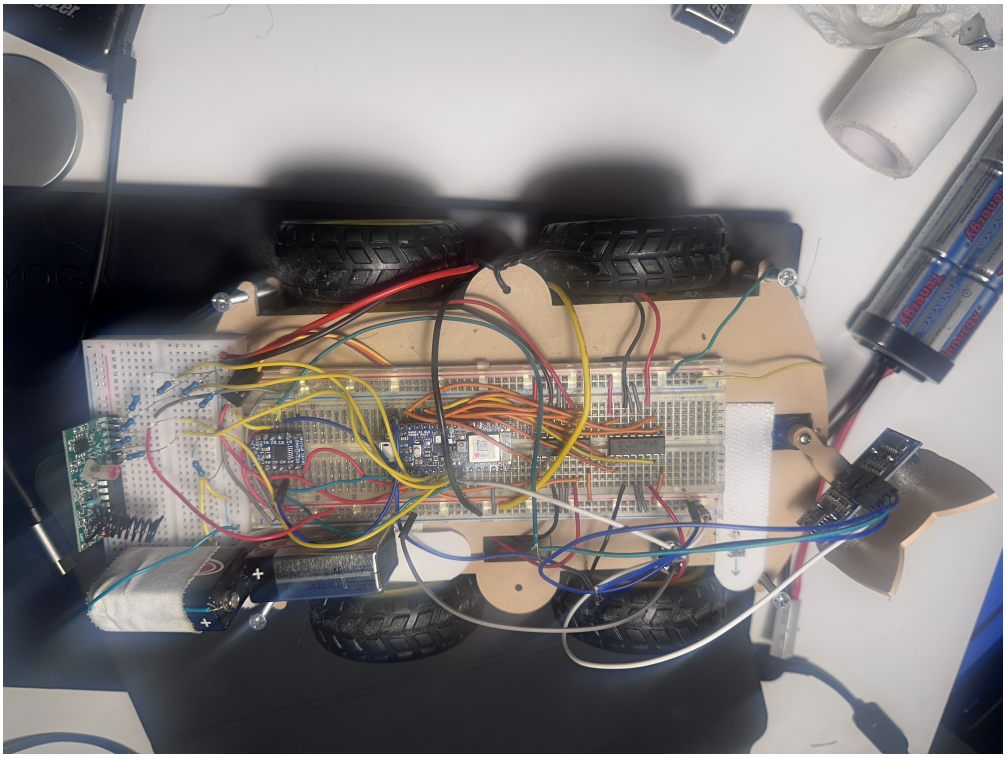


Figure 2: Close-up of electronics assembly. Arduino Nano 33 mounted centrally on chassis. TXB0104 level shifter (small green board) translates 5V RF signals to 3.3V for Arduino. Buck converter steps down 18V battery voltage to 5V for sensors and Arduino VIN. Servo and ultrasonic sensor mounted on 3D-printed brackets at front. Extensive jumper wire connections for 4-motor direct control visible.

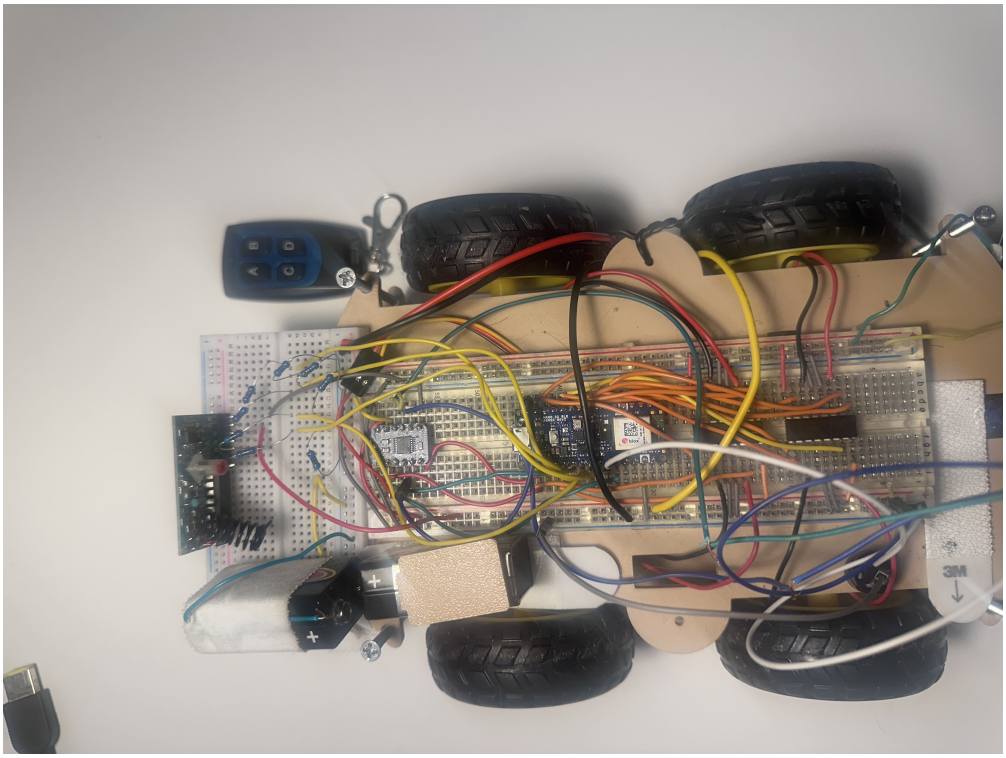


Figure 3: Adafruit 4-button RF remote (keyfob) and RO2A receiver module. The receiver is powered at 5V from the buck converter for optimal range. Simple voltage dividers (1kΩ/1kΩ resistor pairs) convert the 5V digital outputs to safe 2.5V levels for Arduino inputs. This passive solution provides full transmission range (100-150 feet) while ensuring safe operation of the Arduino Nano 33 BLE Sense.

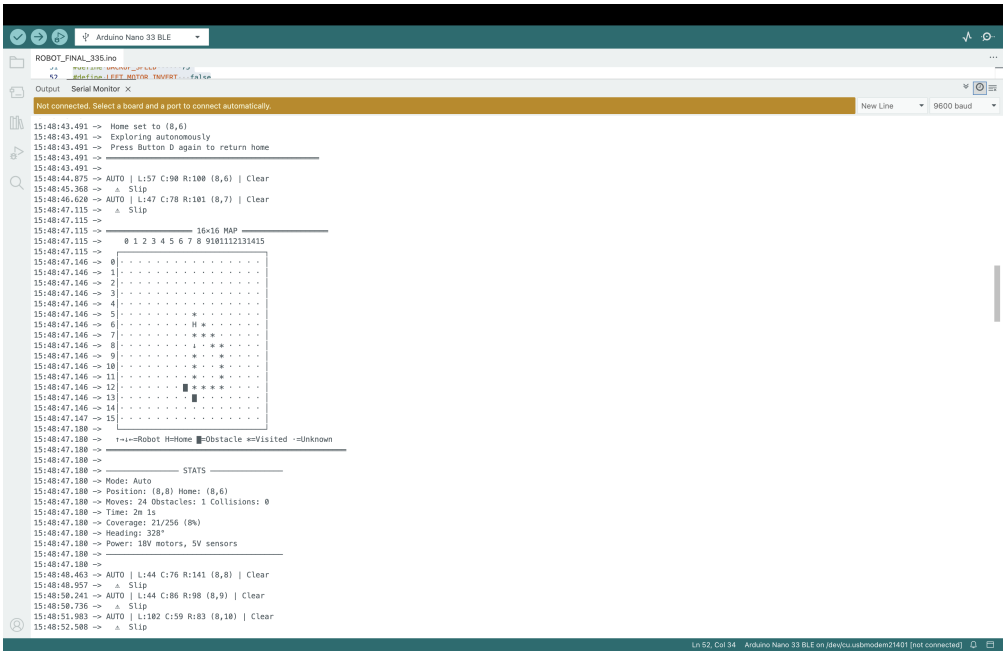
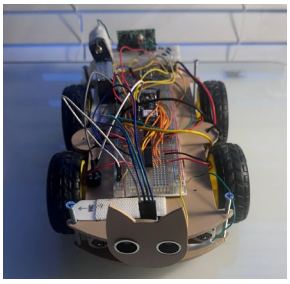


Figure 4: Serial monitor output showing autonomous navigation in progress with 16×16 grid display. The display includes real-time sensor readings (left, center, right distances in cm), current position in grid coordinates, operational mode indicator, and periodic map updates with ASCII visualization. Statistics show move count, obstacle detection, collision events, and map coverage percentage. Note the "H" marking home position set when entering autonomous mode, and directional arrows ($\uparrow/\rightarrow/\downarrow/\leftarrow$) showing robot position and heading.

Autonomous Mapping Robot With RF Remote Control

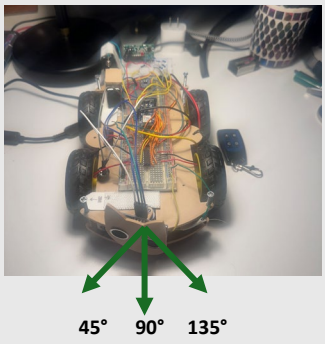
Victor Suarez · Jongchan Park · Akshat Shrivastava

What is it?



This project is an Autonomous Mapping Robot that combines manual control, autonomous exploration, and real-time environmental mapping. The system operates in two distinct modes such as Manual Control and Autonomous Exploration. Auto mode uniquely includes that Return-to-Home controlled via a RF remote control. When the robot is connected to serial monitor, it can display a 16x16 grid map with approximately 12-inch cell resolution, and this is suitable for mapping an indoor environment.

How it works



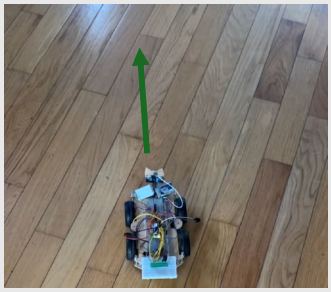
• Sensor Scan

Ultrasonic sensor, mounted on the servo motor, rotates at three angles: 45°, 90°, and 135° to measure distance from the obstacle.
The IMU provides acceleration data for collision detection and compass data for movement direction estimation.

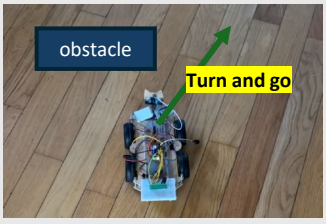
• Decision-making

If the forward path is empty, the robot advances by one grid cell.
When an obstacle is detected, the left-right distance values are compared and rotated in a more open direction.
If all directions are blocked, reverse and rotate 180°.

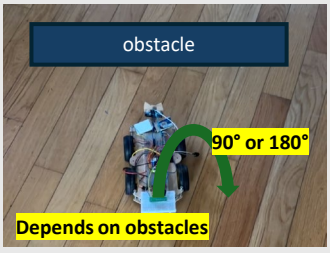
1) empty



2) Obstacle and choose

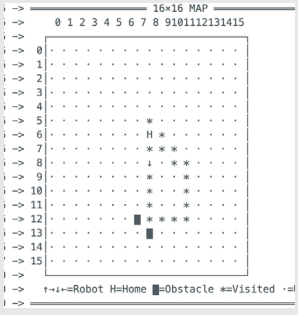


3) Fully blocked



• Switch mode

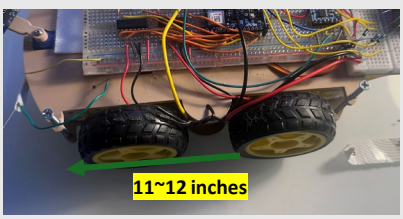
Switching between Manual Control, Autonomous Exploration, and Return-to-Home modes is performed according to the mode switching RF remote control input.



• Update Map (required to connect to serial monitor to display)

After each movement, the robot updates obstacles, empty spaces, visits, and collision information on the 16×16 occupancy grid.

Thank you ☺



• Movement Execution

The motor time is calibrated to ensure all movements correspond to one grid cell (about 11~12 inches).

Autonomous Mapping Robot With RF Remote Control

What we learned

- **Power system design is critical in mobile robot environments.**

  High motor current draw caused voltage brownout, which made Bluetooth communication unreliable and led us to redesign the control interface.

- **Real-world sensor data is noisy and highly environmentally dependent.**

  The ultrasonic sensor had a ±3 cm error, and since the compass (IMU) data was affected by motor current and peripheral metals, which resulted in poor accuracy, the timing and filtering of the sensor were required.

- **Simpler communication can be more robust.**

  Bluetooth Low Energy proved overly complex and sensitive to power instability, while a 433 MHz RF remote provided reliable control under real-world conditions.

- **Accurate calibration plays a key role in autonomous behavior.**

  In order to match the robot's actual movement distance with logical grid movement, the motor operating time and rotation angle had to be repeatedly calibrated.

How to operate (Demo)

RF Remote Control

- A: Move forward by one cell in the grid
- B: Back up a step
- C: Rotate 90° left
- D: Switch mode (2 modes)

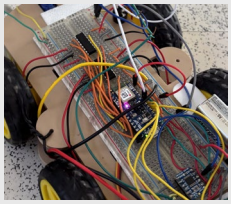
  Manual Control → Autonomous Exploration

  Autonomous Exploration → Return-to-Home

LED Status

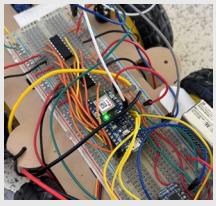
Magenta color:

- Return-to-Home



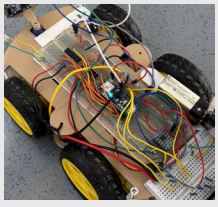
Green:

- Manual control



Cyan:

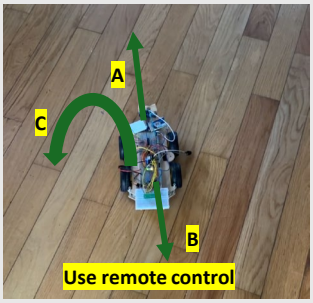
- Autonomous Exploration



- Additional LED colors means processing in movement and rotation

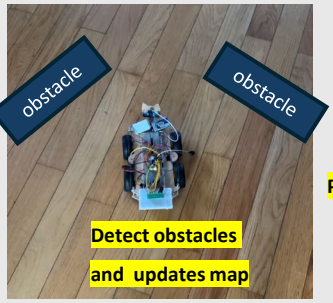
Demo Flow

1) Manual Control



Press D

2) Autonomous



Press D again

3) Return-to-Home