WAYPOINT TRACKER ROBOT CAR

A Project Report submitted
for evaluation of the course
EE 396 : Design Lab

by

Piyush Dhar Patel
(220102070)
Thirumala Punith Kumar Naidu
(220102124)

under the guidance of
Prof. Salil Kashyap

DEPARTMENT OF ELECTRONICS AND ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

Contents

1 Introduction

The **Waypoint Tracker Robot Car** is an autonomous vehicle designed to navigate toward user-defined GPS coordinates. This robotic system integrates multiple modules including a GPS sensor (NEO-6M), a digital compass (HMC5883L), a Bluetooth module (HC-05), and a microcontroller (Arduino UNO). The robot receives destination coordinates via Bluetooth or sets the current location as the target and uses sensor fusion for accurate navigation.

2 Components and Modules Used

- Arduino UNO
- NEO-6M GPS Module
- HC-05 Bluetooth Module
- HMC5883L Compass Module
- L298N Motor Driver Module
- 4 DC Motors
- Wheels
- 2xi8650 Li-on Battery
- Jumper Wires and Breadboard

3 Working of Modules (Detailed)

3.1 NEO-6M GPS Module

The NEO-6M is a GPS receiver module capable of decoding GPS signals to provide real-time latitude, longitude, altitude, and UTC time.

Internal Working:

- The GPS antenna captures radio waves from GPS satellites.
- The RF front-end filters and amplifies the signals.
- A baseband processor decodes the timing and orbital data encoded in the signals.
- The module triangulates position using time differences from at least four satellites.
- It outputs `NMEA` sentences via UART, which are parsed using the `TinyGPS++` library.

3.2 HC-05 Bluetooth Module

The HC-05 module allows serial communication over Bluetooth with a range of up to 10 meters.

Internal Working:

- Operates using an onboard CSR BlueCore chip.
- Supports the Serial Port Profile (SPP).
- Uses 3.3V logic for RX/TX communication.
- Arduino sends AT commands for configuration.
- Destination coordinates are received from the Dabble app.

3.3 HMC5883L Compass Module

The HMC5883L is a 3-axis magnetometer used to determine the robot's heading.

Internal Working:

- Detects Earth's magnetic field in X, Y, Z axes.
- Uses I2C interface to send data to the Arduino.
- Returns raw data in micro-Teslas.
- The Arduino computes heading using the arctangent of Y/X values.
- Magnetic declination and calibration applied for true heading.

4 Compass Calibration

The HMC5883L digital compass is a critical component in our waypoint navigation robot, providing directional heading information used to steer the vehicle toward its destination. However, its readings can be significantly affected by surrounding magnetic disturbances, leading to incorrect navigation decisions. To ensure reliable and accurate performance, proper **calibration** of the compass is essential.

Sources of Error

The sensor is sensitive to two main types of magnetic distortion:

- **Hard iron distortion:** This type of error occurs due to permanent magnets or ferromagnetic materials (like screws, motors, or batteries) placed near the sensor. Hard iron effects introduce a constant bias in the X and Y magnetic field components, resulting in a shift of the origin in the raw magnetic data.
- **Soft iron distortion:** This occurs when nearby objects distort the shape of the magnetic field around the sensor, often due to induced magnetism. Instead of a perfect circle in the X-Y plot of magnetic readings (when the sensor is rotated), soft iron effects cause an elliptical shape.

Calibration Procedure

To correct for these distortions, we perform the following calibration steps:

1. **Data Collection:** The robot is rotated slowly in all directions (ideally a full 360° in the horizontal plane) while continuously recording raw magnetic readings from the X and Y axes.

2. **Determine Extremes:** Over the course of rotation, we collect the maximum and minimum values of both the X and Y axes:

$$\max_x, \min_x, \max_y, \min_y$$

3. **Offset Correction:** The offset due to hard iron distortion is calculated using:

$$\text{offset}_x = \frac{\max_x + \min_x}{2}, \quad \text{offset}_y = \frac{\max_y + \min_y}{2}$$

   These offsets are subtracted from subsequent readings to recenter the data.

4. **Scaling for Ellipse Correction (Optional):** If the plot is still elliptical (soft iron distortion), a scale factor may be applied:

$$\text{scale}_x = \frac{\max_x - \min_x}{2}, \quad \text{scale}_y = \frac{\max_y - \min_y}{2}$$

   The normalized magnetic readings become:

$$x_{\text{norm}} = \frac{x_{\text{raw}} - \text{offset}_x}{\text{scale}_x}, \quad y_{\text{norm}} = \frac{y_{\text{raw}} - \text{offset}_y}{\text{scale}_y}$$

5. **Compute Heading:** The corrected heading angle is calculated using:

$$\text{heading} = \tan^{-1}\left(\frac{y_{\text{norm}}}{x_{\text{norm}}}\right)$$

   Adjustments are made to ensure the angle lies within 0–360°.

Project Implementation

In our robot, we implemented a **software-based offset correction** to account for hard iron distortions. The offset values were determined experimentally by analyzing the compass output while rotating the robot. These offsets were then hardcoded and applied during heading calculation. Additionally, to reduce soft iron effects from the DC motors, we mounted the HMC5883L compass **on an elevated platform above the chassis**. This physical separation minimized magnetic interference and improved heading accuracy.

The impact of calibration was evident in smoother navigation, reduced oscillations during turns, and more consistent arrival at the destination. Without calibration, the robot would often drift or make unnecessary corrections due to inaccurate heading readings.

Future Improvements

In future versions, real-time auto-calibration or graphical visualization tools could be integrated to simplify the process. Logging X-Y magnetic plots to a serial monitor or mobile app could help visually fine-tune the correction. Additionally, incorporating magnetometer fusion with accelerometer and gyroscope data (i.e., using an IMU) could further stabilize heading estimates.

5 Hardware Implementation

5.1 Architecture Overview

The robot consists of multiple components communicating via standard interfaces (Serial, I2C). The Arduino is at the center, coordinating the inputs and controlling the motors based on the control logic.

5.2 Block Diagram

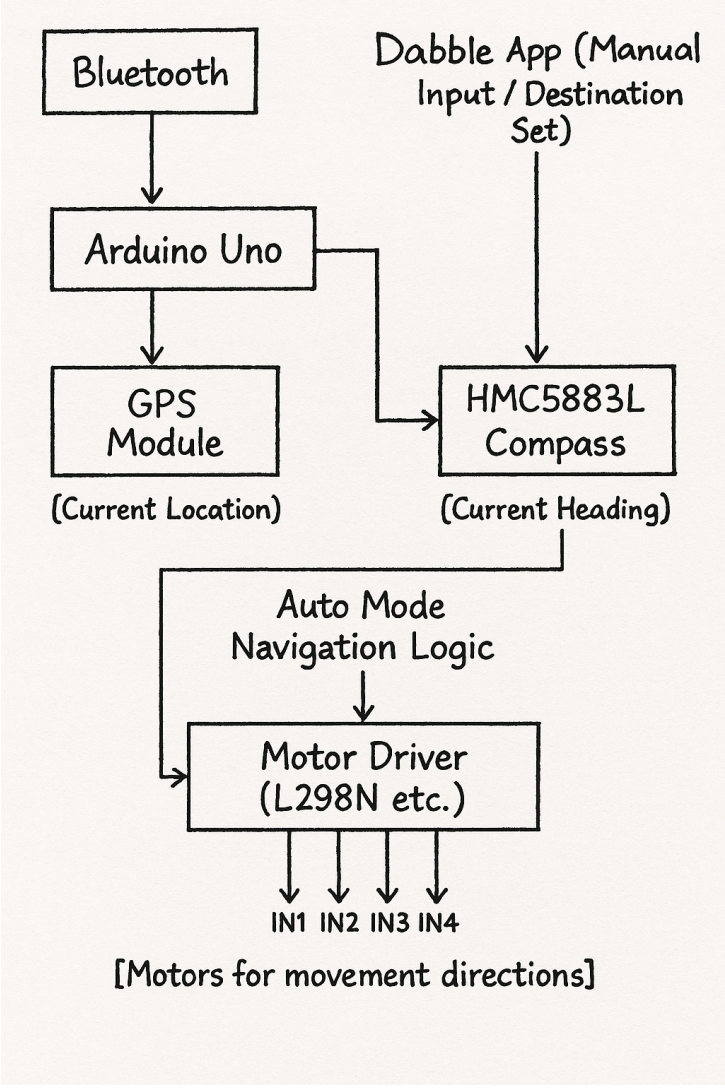

Figure 1: Block Diagram of the System

5.3 Connections and Wiring

- GPS to AltSoftSerial (pins 8, 9)
- HC-05 Bluetooth to SoftwareSerial (pins 4, 5)
- Compass to I2C (A4, A5)
- Motors to L298N IN1-IN4 (pins 2, 3, 6, 7)
- Power shared from 7.4V battery to all modules

6 Software Implementation

6.1 Operating Modes

- **Manual Mode:** User controls the car with Dabble gamepad.
- **Autonomous Mode:** Robot drives to destination using GPS and Compass.

6.2 Software Libraries Used

- `TinyGPS++` - Parses GPS data
- `Dabble` - Bluetooth gamepad and terminal support
- `Adafruit_HMC5883_U` - Compass sensor

6.3 Arduino Code Overview

Highlights include:

- GPS data smoothing via averaging
- Real-time navigation logic based on bearing vs heading
- Bluetooth command parsing

6.4 Full Arduino Code

```
#define CUSTOM_SETTINGS
#define INCLUDE_GAMEPAD_MODULE
#define INCLUDE_TERMINAL_MODULE
#include <AltSoftSerial.h>
#include <Dabble.h>
#include <TinyGPS++.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_HMC5883_U.h>
#include <SoftwareSerial.h>

float minrawLat=30;
```

```
const int IN1 = 6, IN2 = 7, IN3 = 2, IN4 = 3;
TinyGPSPlus gps;
AltSoftSerial GPS_;
bool isAutoMode = false;
Adafruit_HMC5883_Unified mag = Adafruit_HMC5883_Unified(12345);
float destLat = 0.0, destLon = 0.0;
bool destinationSet = false;
#define GPS_AVG_WINDOW 5
float latBuffer[GPS_AVG_WINDOW] = {0};
float lonBuffer[GPS_AVG_WINDOW] = {0};
int bufferIndex = 0;
unsigned long lastUpdate = 0;

void setup() {
  Serial.begin(9600);
  GPS_.begin(9600);
  Dabble.begin(9600, 4, 5);
  mag.begin();
  pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);
  Serial.println("Ready. Manual mode active.");
}

void loop() {
  if(GPS_.available()>0) gps.encode(GPS_.read());
  Dabble.processInput();
  if (Terminal.available()) {
    String input = Terminal.readString();
    input.trim();
    int commaIndex = input.indexOf(',');
    if (commaIndex > 0) {
      destLat = input.substring(0, commaIndex).toFloat();
      destLon = input.substring(commaIndex + 1).toFloat();
      destinationSet = true;
      isAutoMode = true;
    }
  }
  if (GamePad.isSelectPressed()) {
    if (gps.location.lat() && gps.satellites.value()>=4) {
      destLat = gps.location.lat();
      destLon = gps.location.lng();
      destinationSet = true;
    }
  }
  if (GamePad.isStartPressed()) {
    isAutoMode = !isAutoMode;
    delay(500);
  }
```

```
  if (isAutoMode) handleAutoMode();
  else handleManualMode();
}

void handleAutoMode() {
  if (!destinationSet || !gps.location.lat() || gps.satellites.value()<4) r
  if (millis() - lastUpdate >= 1000) {
    float rawLat = gps.location.lat();
    float rawLon = gps.location.lng();
    latBuffer[bufferIndex] = rawLat;
    lonBuffer[bufferIndex] = rawLon;
    bufferIndex = (bufferIndex + 1) % GPS_AVG_WINDOW;
    float currLat = getAverage(latBuffer);
    float currLon = getAverage(lonBuffer);
    float dist = distanceBetween(currLat, currLon, destLat, destLon);
    if (dist < 4.0) stopMotors();
    else {
      float bearing = calculateBearing(currLat, currLon, destLat, destLon);
      float heading = getCompassHeading();
      float diff = bearing - heading;
      if (diff > 180) diff -= 360;
      if (diff < -180) diff += 360;
      if (abs(diff) < 15) moveForward();
      else if (diff > 0) turnRight();
      else turnLeft();
    }
    lastUpdate = millis();
  }
}

void handleManualMode() {
  if (GamePad.isUpPressed()) moveForward();
  else if (GamePad.isDownPressed()) moveBackward();
  else if (GamePad.isLeftPressed()) turnLeft();
  else if (GamePad.isRightPressed()) turnRight();
  else stopMotors();
}

float getAverage(float* buffer) {
  float sum = 0;
  for (int i = 0; i < GPS_AVG_WINDOW; i++) sum += buffer[i];
  return sum / GPS_AVG_WINDOW;
}

float distanceBetween(float lat1, float lon1, float lat2, float lon2) {
  const float R = 6371000;
  float dLat = radians(lat2 - lat1);
  float dLon = radians(lon2 - lon1);
```

```
  float a = sin(dLat / 2) * sin(dLat / 2) +
            cos(radians(lat1)) * cos(radians(lat2)) *
            sin(dLon / 2) * sin(dLon / 2);
  return R * 2 * atan2(sqrt(a), sqrt(1 - a));
}

float calculateBearing(float lat1, float lon1, float lat2, float lon2) {
  float y = sin(radians(lon2 - lon1)) * cos(radians(lat2));
  float x = cos(radians(lat1)) * sin(radians(lat2)) -
            sin(radians(lat1)) * cos(radians(lat2)) * cos(radians(lon2 - lo
  return fmod((degrees(atan2(y, x)) + 360), 360);
}

float getCompassHeading() {
  sensors_event_t event;
  mag.getEvent(&event);
  float heading = atan2(event.magnetic.y+47.45, event.magnetic.x-8.55)+((77
  if (heading < 0) heading += 2 * PI;
  if (heading >= 2*PI) heading -= 2 * PI;
  return degrees(heading);
}

void moveForward() {
  digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);
}

void moveBackward() {
  digitalWrite(IN1, LOW); digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW); digitalWrite(IN4, HIGH);
}

void turnLeft() {
  digitalWrite(IN1, LOW); digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH); digitalWrite(IN4, LOW);
}

void turnRight() {
  digitalWrite(IN1, HIGH); digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW); digitalWrite(IN4, HIGH);
}

void stopMotors() {
  digitalWrite(IN1, LOW); digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW); digitalWrite(IN4, LOW);
}
```

7 Navigation Logic

Distance Calculation

$$d = 2r \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1) \cdot \cos(\phi_2) \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right)}\right)$$

Where:

- d: Distance between two geographical points (in meters)
- r: Radius of the Earth
- ϕ_1, ϕ_2: Latitudes of the two points (in radians)
- λ_1, λ_2: Longitudes of the two points (in radians)
- $\Delta\phi = \phi_2 - \phi_1$: Difference in latitude
- $\Delta\lambda = \lambda_2 - \lambda_1$: Difference in longitude

Bearing Calculation

$$\theta = \arctan 2\left(\sin(\Delta\lambda) \cdot \cos(\phi_2), \cos(\phi_1) \cdot \sin(\phi_2) - \sin(\phi_1) \cdot \cos(\phi_2) \cdot \cos(\Delta\lambda)\right)$$

Where:

- θ: Initial bearing (forward azimuth) from point 1 to point 2 (in radians)
- All other variables are the same as defined above

8 Challenges Faced and Solutions

Challenge	**Solution**
No GPS fix indoors	Tested in open space; waited for satellite lock
Compass fluctuation	Applied calibration routine; averaged multiple readings
Bluetooth delay	Used `Dabble.processInput()` in `loop()` to reduce lag
Inconsistent GPS data	Implemented buffer and average of 5 readings

9 Limitations

Despite the successful implementation of the waypoint tracking system, several limitations were observed during testing and deployment:

- **GPS Accuracy:** The GPS module used typically provides an accuracy of around 3–5 meters under ideal conditions. In urban or closed environments, this accuracy can degrade significantly due to multipath errors or weak satellite signals.
- **Compass Interference:** The magnetic compass is sensitive to nearby electronic components and metallic structures, which can cause heading deviations and affect navigation performance.
- **Signal Delay and Drift:** Real-time position updates may experience delays, and small drifts in sensor data over time can accumulate, leading to gradual errors in navigation.
- **Limited Obstacle Avoidance:** The current system does not include sensors or logic for real-time obstacle detection and avoidance, making it unsuitable for autonomous navigation in dynamic or cluttered environments.
- **Power Constraints:** The system's operation duration is limited by battery capacity. Continuous GPS and Bluetooth communication significantly contribute to power consumption.
- **Manual Calibration Requirements:** The compass requires frequent recalibration to maintain heading accuracy, especially when the device is relocated or reoriented.
- **Environmental Dependence:** The system's performance can be affected by environmental conditions such as heavy rain, dense foliage, or poor sky visibility, which impact satellite connectivity and sensor reliability.

10 Future Improvements

- Add obstacle detection using ultrasonic sensors
- Use IMU instead of magnetometer for robust heading in interference zones
- Optimize path with GPS waypoints
- Add remote tracking and telemetry via GSM or LoRa

11 Conclusion

This project demonstrates the practical implementation of embedded systems and sensor fusion techniques for autonomous robotic navigation. By integrating GPS, a digital compass, and Bluetooth communication with a motorized robotic base, the system effectively navigates to arbitrary GPS coordinates with minimal human intervention. The modular design ensures that each component—whether for location tracking, directional

alignment, or wireless control—can be independently upgraded or replaced, offering scalability and adaptability for a wide range of future applications.

In addition to being cost-effective, the system emphasizes real-time decision-making and error correction based on environmental feedback, showcasing the importance of sensor calibration and data fusion in robotics. The use of Bluetooth not only enables easy configuration and remote control but also opens possibilities for swarm robotics and multi-agent coordination in future iterations.

Overall, this work lays a robust foundation for more complex autonomous systems, including obstacle avoidance, dynamic path planning, and integration with machine learning algorithms for intelligent behavior. It also serves as a valuable educational platform for students and enthusiasts interested in embedded development, control systems, and robotic

12 Images of Final Robot

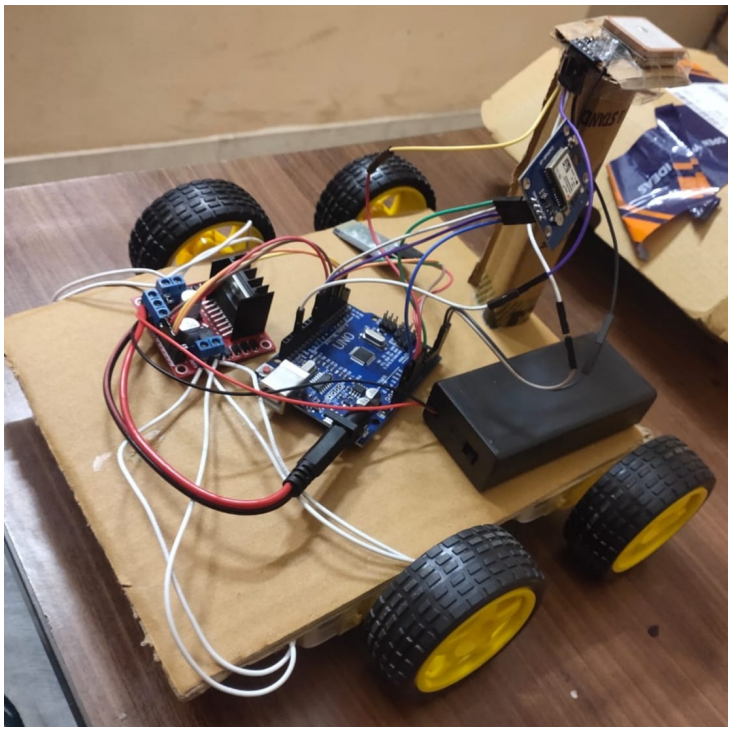

Figure 2: Image of robo Car

There are still some improvements to be done

13 References

1. NEO-6M GPS Datasheet
2. HC-05 Bluetooth Module Documentation
3. Adafruit HMC5883L Sensor Guide
4. TinyGPS++ Library: `https://github.com/mikalhart/TinyGPSPlus`

5. Dabble App: https://thestempedia.com/docs/dabble/

6. Arduino Docs: https://www.arduino.cc/