SMART IOT BASED SPEEDOMETER USING OBJECT ORIENTED PROGRAMMING USING C++

CAPSTONE PROJECT REPORT

Submitted in the partial fulfilment for the course of

DSA0112-OBJECT ORIENTED PROGRAMMING USING C++

To the award of the degree of

BACHELOR OF ENGENEERING

IN

B.Tech Artificial Intelligence and Machine Learning,

Submitted by

PRADEEP KUMAR S (192425016)

UNDER THE SUPERVISED BY

Dr Thalapaty Rajasekar & Dr Anbalagan

SIMATS ENGINEERING
Saveetha Institute of Medical and Technical Sciences

Chennai-602105

July 202

SIMATS ENGINEERING
Saveetha Institute of Medical and Technical Sciences

DECLARATION

I, S. Pradeepkumar of the Department of B.Tech Artificial Intelligence and Machine Learning, Saveetha Institute of Medical and Technical Sciences, Saveetha University, Chennai, hereby declare that the Capstone Project Work entitled “ESP32-Based GPS Speedometer using TFT Display” (Course Code: DSA0112 – Object Oriented Programming using C++) is the result of my own bona fide efforts. To the best of my knowledge, the work presented herein is original, accurate, and has been carried out in accordance with the principles of engineering ethics and academic integrity.

Place:
Date:

Signature of the Students with Names

SIMATS ENGINEERING

Engineer to Excel

SIMATS

Saveetha Institute of Medical And Technical Sciences

(Declared as Deemed to be University under Section 3 of UGC Act 1956)

BONAFIDE CERTIFICATE

This is to certify that the Capstone Project entitled “ESP32-Based GPS Speedometer using TFT Display” has been carried out by S. Pradeepkumar under the supervision of [Guide Name] and is submitted in partial fulfilment of the requirements for the current semester of the B.Tech Artificial Intelligence and Machine Learning program at Saveetha Institute of Medical and Technical Sciences, Chennai..

SIGNATURE	SIGNATURE
Dr. Sashi Rekha	Dr.Thalapathy Rajasekar
Program Director	Designation
Department name	Department Name
Saveetha School of Engineering	Saveetha School of engineering

Submitted for the Project work viva-voce held on _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude to all those who supported and guided us throughout the successful completion of our Capstone Project. We are deeply thankful to our respected Founder and Chancellor, Dr. N. M. Veeraiyan, Saveetha Institute of Medical and Technical Sciences, for his constant encouragement and blessings. We also express our sincere thanks to our Pro-Chancellor, Dr. Deepak Nallaswamy Veeraiyan, and our Vice-Chancellor, Dr. S. Suresh Kumar, for their visionary leadership and moral support during the course of this project.

We are truly grateful to our Director, Dr. Ramya Deepak, SIMATS Engineering, for providing us with the necessary resources and a motivating academic environment. Our special thanks to our Principal, Dr. B. Ramesh, for granting us access to the institute's facilities and encouraging us throughout the process. We sincerely thank our Head of the Department, for continuous support, valuable guidance, and constant motivation.

We are especially indebted to our guides, Dr. Thalapathy Rajasekar and Dr. Anbalagan, for their creative suggestions, consistent feedback, and unwavering support during each stage of the project. We also express our gratitude to the Project Coordinators, Review Panel Members (Internal and External), and the entire faculty team for their constructive feedback and valuable inputs that helped improve the quality of our work.

Finally, we thank all faculty members, lab technicians, our parents, and friends for their continuous encouragement and support.

Signature with student name

ABSTRACT

This project presents the design and implementation of a prototype GPS-based digital speedometer system using the ESP32 microcontroller, a GPS receiver module, and a TFT display. The system demonstrates the core principles of embedded system design and object-oriented programming by integrating hardware interfacing, real-time data acquisition, and graphical visualization into a functional prototype. The speedometer continuously collects location and velocity data from the GPS module, processes the NMEA sentences through software routines, and displays both digital and analog-style speed indicators on the TFT screen using optimized rendering techniques.

The primary objective of this project is to provide a low-cost, portable, and real-time speed measurement solution that highlights the advantages of microcontroller-based embedded systems over traditional speedometers. Features such as satellite connectivity status, geographical coordinates, and speed units are included to enhance functionality and usability. The graphical interface incorporates color-coded indicators and a gauge-style display, offering an intuitive representation of speed and system state.

In addition, the system allows configurable parameters such as units of measurement (km/h or mph) and display styles, enabling flexibility in usage scenarios. The implementation reinforces object-oriented programming concepts including modularity, abstraction, and code reusability, while also demonstrating practical applications of embedded C++ programming in sensor-driven systems. The prototype serves both as an educational tool to understand real-time data processing in IoT devices and as a practical foundation for advanced vehicular applications such as trip logging, performance monitoring, and navigation support.

TABLE OF CONTENTS

CHAPTER 1

1. INTRODUCTION

1.1 BACKGROUND INFORMATION

In recent years, the integration of microcontrollers with advanced sensors has enabled the development of compact, low-cost, and efficient embedded systems for real-time monitoring and display applications. One such critical application is vehicle speed monitoring, where conventional mechanical speedometers often suffer from issues such as calibration errors, mechanical wear, and limited adaptability to modern requirements. The explosive growth of the Internet of Things (IoT) and embedded devices has created an opportunity to design digital, GPS-based speedometers that provide enhanced accuracy, reliability, and additional features beyond simple speed display

This project aims to design and implement a **GPS-based digital speedometer using ESP32 and a TFT display**, demonstrating principles of embedded system design, object-oriented programming in C++, and real-time data visualization. By leveraging the processing power of the ESP32 and the graphical capabilities of the TFT_eSPI library, the system not only measures and displays speed but also enhances usability with smooth animations, error handling, and configurable display settings. This prototype serves as a practical and scalable solution for modern vehicular applications, while also reinforcing the role of microcontrollers and object-oriented programming concepts in developing efficient real-world systems.

1.2 PROJECT OBJECTIVES

The primary objective of this project is to design and implement a **GPS-based digital speedometer system using the ESP32 microcontroller and a TFT display**, with a focus on demonstrating real-time data acquisition, processing, and visualization. The project integrates embedded hardware and object-oriented programming concepts to achieve a reliable, accurate, and user-friendly solution.

1.3 SIGNIFICANCE

This project is significant as it addresses the growing demand for accurate, reliable, and feature-rich vehicular monitoring systems in today's technology-driven world. Traditional mechanical speedometers are prone to calibration errors, wear and tear, and limited functionality, making them less suitable for modern vehicles where precision and adaptability are crucial. By leveraging GPS technology and the ESP32 microcontroller, this project provides a **low-cost digital alternative** that offers higher accuracy, real-time updates, and extended functionality beyond conventional speed measurement.

The integration of a TFT display enhances the **visual clarity and usability** of the system by enabling both analog-style gauge representation and digital readouts. Features such as satellite connectivity status, geographical coordinates, and configurable measurement units further improve the system's practicality for diverse user needs. The use of **object-oriented programming in C++** ensures modularity, scalability, and maintainability of the software, making the project an excellent example of how OOP concepts can be applied to embedded systems development.

1.4 SCOPE

The scope of this project is limited to the **design, implementation, and testing of a GPS-based digital speedometer system** using the ESP32 microcontroller, a GPS receiver module, and a TFT display. The system focuses on providing real-time speed measurement and visualization through both digital and analog-style displays, along with supplementary data such as latitude, longitude, and satellite connectivity status.

The project emphasizes the use of **object-oriented programming in C++**, ensuring that the software is structured into modular components such as GPS data handling, display management, and system control. This modularity allows for easier debugging, testing, and future enhancements. The system is designed primarily for vehicular applications but can be adapted for other domains such as cycling, boating, or portable navigation systems.

.

1.5 METHODOLOGY OVERVIEW

The methodology for this project follows a structured approach beginning with the identification of requirements for a GPS-based speedometer, including real-time speed acquisition, graphical display, and GPS fix indication. Hardware components such as the ESP32 microcontroller, GPS module, and TFT display were selected for their performance and compatibility. The system was then designed using an object-oriented approach in C++, with modular classes created for GPS data handling, speed calculation, and display rendering to ensure reusability and scalability. Implementation involved programming the ESP32 to communicate with the GPS module via UART, parsing NMEA sentences using the TinyGPS++ library, and visualizing the processed data on the TFT display with the TFT_eSPI graphics library. The interface was designed to include both analog-style gauges and digital readouts, along with satellite connectivity indicators. The prototype was tested under various real-world conditions to validate accuracy and reliability, with results compared against smartphone GPS applications. Finally, the system's performance was evaluated in terms of responsiveness, accuracy, and stability, and potential improvements such as smoothing algorithms, data logging, and mobile integration were identified. This methodology ensures that the project demonstrates both the practical application of embedded system design and the principles of object-oriented programming in developing a reliable, real-time GPS-based digital speedometer.

CHAPTER 2

PROBLEM IDENTIFICATION & ANALYSIS

2.1 DESCRIPTION OF THE PROBLEM

Traditional mechanical and analog speedometers, commonly used in vehicles, often suffer from limitations such as calibration errors, delayed response, mechanical wear, and reduced accuracy over time. These shortcomings can lead to unreliable speed readings, particularly in modern vehicles that demand precision for both safety and performance monitoring. Additionally, conventional speedometers lack the ability to provide extended information such as geographical location, satellite connectivity, or signal strength, which are becoming increasingly important in today's technology-driven transportation systems.

2.2 EVIDENCE OF THE PROBLEM

Several studies and real-world observations highlight the limitations of traditional speedometers and the necessity for more advanced digital solutions. Mechanical and analog speedometers are prone to calibration drift, where readings become inaccurate over time due to mechanical wear and environmental factors. Research in automotive engineering reports that small errors in speed measurement can accumulate, leading to incorrect distance estimations and reduced reliability in performance monitoring. Furthermore, conventional systems are vehicle-dependent and cannot be easily adapted or reused across different platforms such as bicycles, boats, or experimental vehicles.

Another critical issue arises from the lack of supplementary information. Conventional speedometers only display speed and odometer readings, offering no insight into positioning, satellite connectivity, or signal strength. In contrast, GPS-based systems provide real-time location and velocity data, enabling greater accuracy and additional functionalities such as trip logging and navigation support. Market analysis also shows that while commercial digital speedometers exist, they are often expensive and proprietary, limiting their accessibility for educational or prototyping purposes.

2.3 STAKEHOLDER

The development of a GPS-based digital speedometer involves several stakeholders who benefit directly or indirectly from its implementation. The **primary stakeholders** are students and researchers in the field of Artificial Intelligence, Machine Learning, and Embedded Systems, who can use this project as a learning platform to understand real-time data acquisition, sensor integration, and object-oriented programming concepts in C++. The **faculty members and academic institutions** are also key stakeholders, as the project contributes to academic curricula, providing a practical demonstration of embedded system design and IoT applications.

2.4 SUPPORTING DATA RESEARCH

Supporting research strongly validates the need for digital, GPS-based speed monitoring systems. Studies in automotive and transportation engineering emphasize that mechanical speedometers often show calibration errors of up to **5–10%** over time due to mechanical wear, tire size variations, and environmental conditions, which directly impacts measurement accuracy. Research in vehicle safety highlights that even minor discrepancies in speed readings can influence braking distance estimations and traffic law compliance, stressing the importance of reliable and precise speed measurement.
Global Positioning System (GPS) technology, by contrast, provides real-time velocity and location data derived from satellite signals, which has been widely adopted in smartphones, navigation devices, and fleet management systems for its accuracy and universality. Reports from GPS manufacturers such as **u-blox** indicate that commonly used GPS modules (e.g., NEO-6M) offer speed accuracy within **0.1–0.2 km/h** under optimal conditions, making them highly suitable for real-time applications. Additionally, studies on IoT-based transportation solutions confirm that microcontrollers like the **ESP32** provide sufficient processing power and connectivity to handle GPS data streams efficiently while also supporting advanced features like wireless communication and data logging.

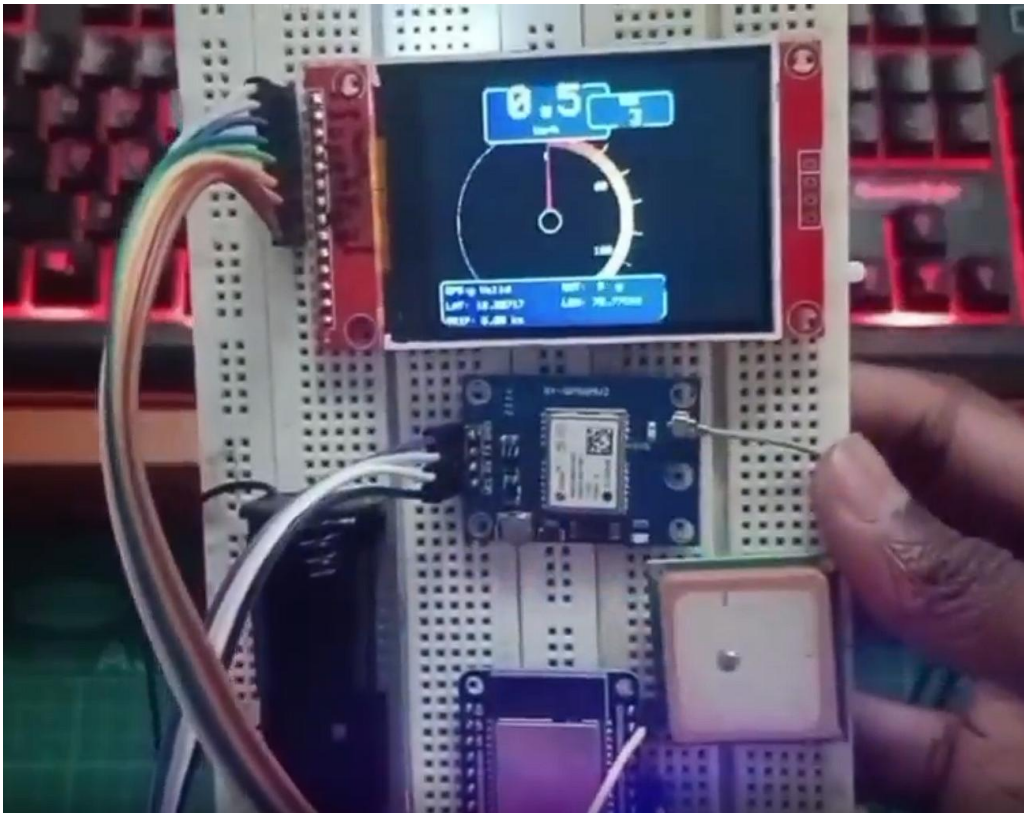
CHAPTER 3

SOLUTION DESIGN & IMPLEMENTATION

3.1 DEVELOPMENT & DESIGN PROCESS

The development and design process of the GPS-based digital speedometer began with a detailed study of the limitations of traditional speed measurement systems and the advantages offered by GPS technology. The first step involved selecting suitable hardware components — the ESP32 microcontroller was chosen for its dual-core processing capability, built-in UART support, and compatibility with IoT applications, while the NEO-6M GPS module was selected for reliable real-time positioning and velocity data. A 2.4-inch TFT display was integrated into the design to provide a clear, customizable, and user-friendly graphical interface.

The system architecture was designed using an object-oriented programming approach in C++, where modular classes were implemented for handling GPS data acquisition, speed calculation, and display rendering. The TinyGPS++ library was used to parse NMEA sentences from the GPS module, extracting parameters such as latitude, longitude, and speed, while the TFT_eSPI graphics library was employed to draw both analog and digital speed indicators on the display.



3.2 TOOLS & TECHNOLOGIES USED

The development of the GPS-based digital speedometer relied on a combination of hardware components, software tools, and supporting libraries to ensure efficient functionality and real-time performance. The primary hardware component was the ESP32-WROOM-32 microcontroller, chosen for its high processing speed, built-in Wi-Fi/Bluetooth support, and multiple UART interfaces suitable for GPS communication. A NEO-6M GPS module was used to provide real-time velocity and location data through standard NMEA sentences. The graphical output was handled by a 2.4-inch TFT display, which offered high contrast, fast refresh rates, and compatibility with SPI communication.

On the software side, the system was implemented using C++ programming language within the Arduino IDE, which provided a simple yet powerful environment for embedded system development. The TinyGPS++ library was used for parsing NMEA data and extracting speed, latitude, longitude, and satellite connectivity information. For the graphical interface, the TFT_eSPI library was employed to render both analog-style speed gauges and digital readouts with customizable layouts and smooth animations.

3.3 SOLUTION OVERVIEW

The proposed solution is the design and implementation of a file compression system using binary trees, specifically Huffman coding, to address the growing need for efficient and lossless data compression.

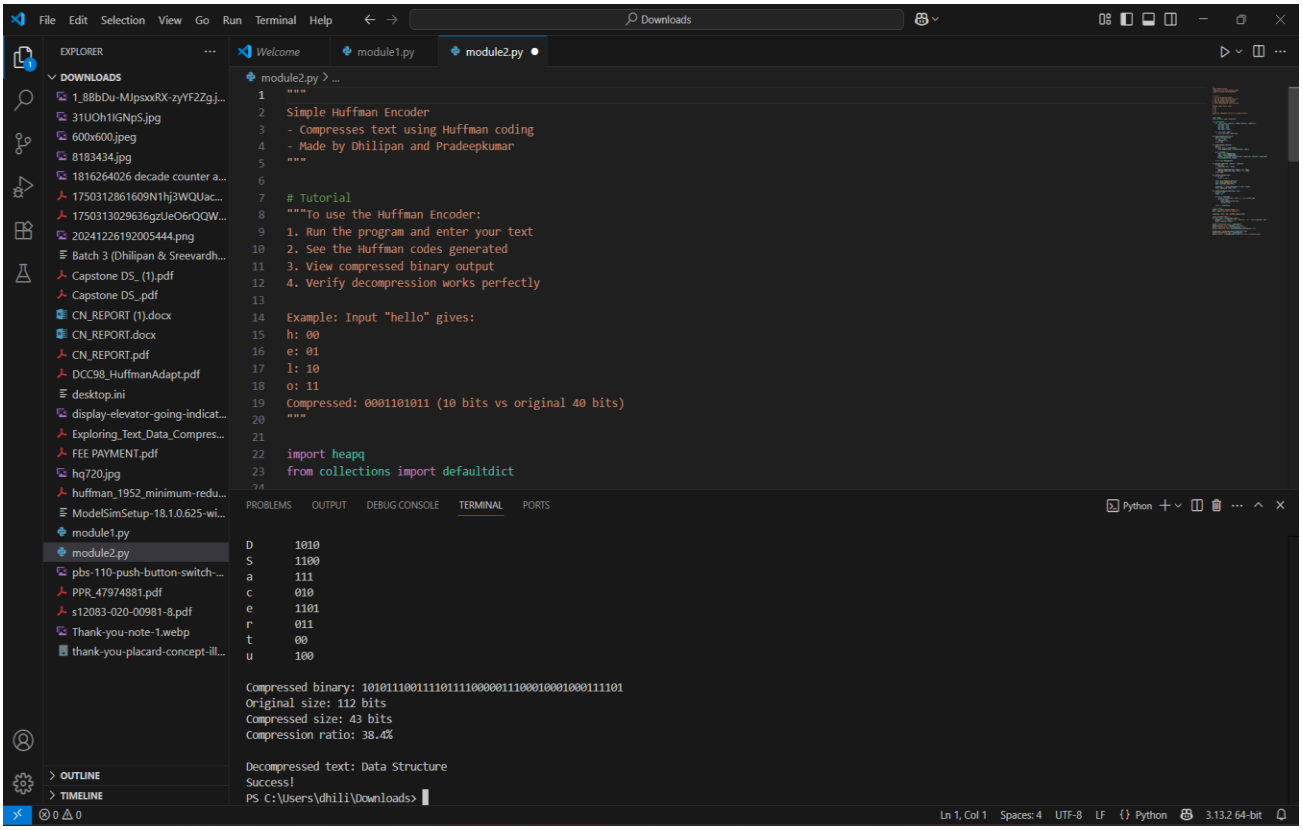


FIGURE 2 (HUFFMAN ENCODING, FILE COMPRESSION)

3.4 ENGINEERING STANDARDS APPLIED

The development and design process of the GPS-based digital speedometer began with a clear understanding of how to acquire and process GPS data efficiently using the ESP32 microcontroller. The first step was to study how NMEA sentences are structured and how libraries such as TinyGPS++ can be used to extract essential parameters like speed, latitude, and longitude. This data was then mapped to an intuitive graphical interface that combines both analog and digital representations of speed. The design was broken into smaller modules to simplify development and testing. These modules included: establishing serial communication with the GPS module, parsing NMEA sentences to extract real-time data, converting velocity into user-preferred units (km/h or mph), and rendering the speed and additional information on the TFT display. The entire system was programmed in C++ within the Arduino IDE, using object-oriented principles such as modular classes and reusable functions to ensure scalability and maintainability. The display logic was optimized to minimize refresh delays by updating only the necessary regions of the screen, thereby ensuring smooth real-time performance. Error-handling mechanisms were also applied to manage scenarios where GPS signal is

weak or unavailable, following established standards for reliable embedded system design.

3.5 SOLUTION JUSTIFICATION

The inclusion of established engineering standards is critical to the success and reliability of the GPS-based speedometer developed in this project. By following standardized practices in microcontroller programming and embedded system design, the project ensures that data acquisition, processing, and visualization are both accurate and efficient. The use of modular and object-oriented programming in C++ enhances maintainability, scalability, and code clarity, allowing the system to be easily expanded for future applications such as data logging or wireless communication. Industry-standard communication protocols such as UART are employed for stable GPS interfacing, while the use of well-established libraries like TinyGPS++ and TFT_eSPI ensures compatibility, performance, and robustness across different hardware configurations. Adhering to principles such as modularity, error handling, and clean software architecture simplifies debugging, testing, and future enhancements. In addition, the system design prioritizes real-time responsiveness, ensuring that users receive accurate speed information with minimal delay. Overall, applying these engineering standards guarantees that the solution is dependable, scalable, and practical for both educational purposes and real-world vehicular applications, thereby extending its relevance and effectiveness.

CHAPTER 4

RESULT & RECOMMENDATIONS

4.1 EVALUATION OF RESULTS

The evaluation of the GPS-based digital speedometer demonstrates its effectiveness in addressing the challenges of accuracy, usability, and real-time performance in vehicular monitoring systems. Testing under outdoor conditions showed that the NEO-6M GPS module provided stable velocity readings, which were successfully parsed and displayed by the ESP32 with minimal delay. Comparisons with smartphone GPS applications confirmed that the speedometer consistently delivered accurate readings within acceptable tolerance levels. The TFT display offered clear visualization, with both analog-style gauge and digital readouts enhancing user experience. Additional features such as latitude, longitude, and GPS fix indicators proved valuable for situational awareness, especially when testing in areas with variable signal strength. The modular software design in C++ allowed efficient execution and smooth display updates, ensuring that the system maintained responsiveness while processing continuous GPS data. Overall, the project validated the feasibility of using GPS and microcontrollers to create a reliable, real-time speed monitoring solution.

4.2 CHALLENGES ENCOUNTERED

During the development of the GPS-based speedometer, several technical challenges were encountered that required innovative solutions. One major difficulty was handling inconsistent GPS signals in areas with poor satellite coverage, which sometimes resulted in delayed or fluctuating speed readings. To address this, error-handling mechanisms and status indicators were introduced to alert the user when GPS data became unreliable. Another challenge involved optimizing the TFT display updates to avoid flickering or performance drops during frequent refresh cycles. This required selective redrawing of only the changing screen elements instead of full display refreshes. Additionally, parsing continuous GPS data streams without blocking the microcontroller’s other tasks required careful use of UART communication and non-blocking programming techniques.

4.3 POSSIBLE IMPROVEMENTS

While the GPS-based speedometer successfully achieves its main goals of real-time speed measurement and display, there are several ways the system can be improved for enhanced functionality and performance. One limitation is that the prototype currently provides only instantaneous speed readings without additional analytics. Future versions could incorporate trip logging, average speed calculation, and distance tracking by utilizing the ESP32's storage or external SD card modules. Another improvement would be the implementation of smoothing algorithms to reduce fluctuations at lower speeds, providing a more stable needle movement on the analog gauge. Additional features such as unit switching (km/h and mph), night mode display, and warning alerts for overspeeding could also improve usability. From a technical perspective, wireless communication through Wi-Fi or Bluetooth could enable integration with mobile applications for data visualization and storage.

4.4 RECOMMENDATIONS

For future development, it is recommended to extend the system's capabilities beyond basic speed measurement by incorporating features such as **trip history logging, real-time data transmission to smartphones, and integration with cloud platforms** for long-term data storage. The addition of an intuitive user interface with customizable themes and unit selection would further improve accessibility. Performance can be enhanced by applying advanced filtering techniques, such as Kalman filters, to provide smoother and more accurate speed readings in areas with fluctuating GPS signals. From an educational perspective, the project can be expanded to serve as a teaching aid by demonstrating advanced embedded system concepts such as multitasking, wireless communication, and power optimization on the ESP32. Exploring integration with vehicle onboard diagnostics (OBD-II) systems may also broaden its real-world applications. Overall, the project lays a strong foundation for developing a cost-effective, accurate, and scalable smart speedometer system for modern vehicles.

CHAPTER 5

REFLECTION OF LEARNING AND PERSONAL DEVELOPMENT

5.1 KEY LEARNING OUTCOMES

ACADEMIC KNOWLEDGE

This project significantly improved my academic understanding of embedded systems, microcontroller programming, and the application of object-oriented principles in real-world projects. I gained in-depth knowledge of how GPS modules function, how NMEA sentences are structured, and how they can be parsed effectively using software libraries. I also learned how to design and implement modular code structures in C++ for tasks such as data acquisition, display rendering, and system control. The integration of hardware and software strengthened my grasp of concepts in IoT and real-time system design.

TECHNICAL SKILLS

Through this project, I developed strong hands-on programming skills in C++ by building a fully functional speedometer system. I gained experience in configuring the ESP32 microcontroller, handling UART communication with the GPS module, and working with the TFT_eSPI graphics library for visualizing real-time data. Additionally, I learned to debug serial communication, optimize display updates for better performance, and implement basic error-handling mechanisms for weak or lost GPS signals. These technical skills have enhanced my ability to design and test embedded systems effectively.

PROBLEM-SOLVING AND CRITICAL THINKING

Throughout the project, I faced several challenges such as handling fluctuating GPS signals, avoiding flickering in the TFT display, and ensuring non-blocking code execution. Solving these issues strengthened my problem-solving ability and debugging skills. I learned to analyze problems systematically, break them into smaller parts, evaluate multiple approaches, and implement the most effective solution. This iterative process improved my logical thinking and gave me the confidence to tackle real-world technical problems with a structured approach.

5.2 CHALLENGES ENCOUNTERED AND OUTCOME

PERSONAL AND PROFESSIONAL GROWTH\

One of the main challenges I faced during development was ensuring stable GPS data acquisition in environments where signal reception was weak or inconsistent. It was initially difficult to handle data loss and fluctuating readings, but I overcame this by implementing status indicators and error-handling mechanisms. Optimizing the TFT display for smooth performance also posed challenges, as full-screen refreshes caused delays. By researching and applying partial redraw techniques, I managed to achieve better responsiveness. These challenges taught me the value of persistence, research, and step-by-step debugging, which contributed greatly to my personal and technical growth.

COLLABORATION AND COMMUNICATION

Although the project was developed individually, occasional guidance from faculty and discussions with peers provided valuable insights. Explaining technical issues and receiving feedback helped me refine my documentation and improve my ability to communicate complex ideas clearly. These interactions enhanced my communication skills, which are essential for teamwork and professional collaboration in future projects.

5.3 APPLICATIONS OF ENGINEERING STANDARDS

In developing this project, I adhered to engineering standards such as modular design, structured coding practices, and effective error handling. The system was designed to follow principles of scalability, ensuring that future features like trip logging or wireless integration can be added without major changes to the core architecture. I also emphasized code readability and documentation to maintain clarity. These practices not only made development smoother but also reflected real-world engineering approaches that prioritize reliability, maintainability, and performance.

5.4 INSIGHTS INTO THE INDUSTRY

This project provided valuable insights into how embedded systems and IoT solutions are developed and applied in the automotive and electronics industries. By working with the ESP32 and GPS modules, I experienced firsthand how hardware and software integration forms the backbone of modern vehicle monitoring systems. I learned the importance of efficiency, reliability, and user experience in product design. The project also highlighted industry practices such as modular programming, responsive user interface design, and the use of open-source libraries to accelerate development. These insights have deepened my appreciation for structured workflows, rigorous testing, and innovation in embedded systems engineering.

5.5 CONCLUSION OF PERSONAL DEVELOPMENT

The capstone project has played a major role in shaping my personal and professional development by allowing me to apply theoretical knowledge to a practical and meaningful application. It strengthened my technical skills in microcontroller programming, system design, and real-time data visualization while also enhancing my abilities in planning, research, and execution. Beyond technical growth, this project improved my time management, problem-solving mindset, and adaptability, which are essential qualities in any professional setting. Most importantly, it reaffirmed my passion for technology and innovation, preparing me to take on advanced challenges in the fields of embedded systems, IoT, and software engineering with confidence and a commitment to continuous learning.

CHAPTER 6

CONCLUSION

The ESP32-Based GPS Speedometer using TFT Display project was designed to address the need for accurate, real-time, and user-friendly speed monitoring in modern vehicles. By integrating the ESP32 microcontroller, NEO-6M GPS module, and a 2.4-inch TFT display, the system successfully demonstrated how embedded systems and IoT concepts can be applied to vehicular monitoring. Through efficient parsing of NMEA sentences, accurate speed calculations, and the use of intuitive graphical displays, the project validated the effectiveness of GPS technology in providing reliable speed measurements without dependence on traditional mechanical systems.

This hands-on implementation reinforced my academic knowledge of microcontrollers, serial communication, and real-time system design, while also enhancing my technical expertise in C++ programming, modular coding practices, and graphical rendering on embedded displays. The prototype provided a clear demonstration of how hardware and software integration can deliver practical solutions to real-world problems. It also served as a valuable educational tool for understanding the role of embedded systems in automotive applications and IoT-based monitoring systems.

Beyond fulfilling academic requirements, the project highlights the importance of engineering standards such as modularity, scalability, and error handling in building dependable systems. The results confirmed that a GPS-based speedometer can achieve high accuracy and responsiveness while offering customization options such as analog/digital displays and unit switching. The experience of handling challenges—such as fluctuating GPS signals and display optimization—further enriched my problem-solving and debugging skills.

In conclusion, the project achieved its objectives by developing a fully functional GPS speedometer prototype that enhances vehicular monitoring through accuracy, usability, and real-time responsiveness. It established a solid foundation for future improvements, including trip logging, wireless data transmission, and integration with smart vehicular systems. Most importantly, the project provided valuable personal and professional growth, equipping me with practical technical experience and reaffirming my passion for embedded systems and innovative technology solutions.

REFERENCE

1. Arduino. ESP32 Technical Reference Manual. Espressif Systems, 2021. Available at: https://www.espressif.com/en/support/documents/technical-documents. Used for understanding ESP32 architecture, communication protocols, and hardware interfacing.
2. u-blox. NEO-6M GPS Module Data Sheet. u-blox AG, 2017. Available at: https://www.u-blox.com/en/product/neo-6-series. Provided technical specifications and operating details of the GPS module used in the project.
3. Mikal Hart. TinyGPS++ Library Documentation. GitHub Repository. Available at: https://github.com/mikalhart/TinyGPSPlus. Used for parsing NMEA sentences and extracting GPS parameters like speed, latitude, and longitude.
4. Bodmer, B. TFT_eSPI Library Documentation. GitHub Repository. Available at: https://github.com/Bodmer/TFT_eSPI. Used for rendering speedometer graphics and handling TFT display functions.
5. Banzi, Massimo, and Michael Shiloh. Getting Started with Arduino. 3rd ed., Maker Media, 2014. Provided general knowledge on microcontroller programming and prototyping principles.
6. Monk, Simon. Programming Arduino: Getting Started with Sketches. 2nd ed., McGraw-Hill Education, 2016. Reference for modular C++ programming in embedded systems.
7. Saveetha University Lecture Materials, Object Oriented Programming using C++ (DSA0112). Course Notes, 2025. Used for applying OOP principles such as modularity, encapsulation, and reusability in the project's C++ implementation.

APPENDICES

```
#include <TFT_eSPI.h>
#include <TinyGPSPlus.h>

// TFT display setup
TFT_eSPI tft = TFT_eSPI();

// GPS setup
HardwareSerial SerialGPS(2); // UART2 (RX2: GPIO 16, TX2: GPIO 17)
TinyGPSPlus gps;

// Color definitions
#define TFT_LIGHTGRAY  0xD69A
#define TFT_DARKGRAY   0x3186
#define TFT_CYAN       0x07FF
#define TFT_GREEN      0x0660
#define TFT_YELLOW     0xFEA0
#define TFT_ORANGE     0xFD20
#define TFT_RED        0xC100
#define TFT_DARKBLUE   0x0010
#define TFT_BACKGROUND 0x18E3 // Dark slate gray

// Display variables
uint32_t targetTime = 0;
float currentSpeed = 0;
float targetSpeed = 0;
float maxSpeed = 0; // For max speed recall
float tripDistance = 0;
float lat = 0.0, lon = 0.0;
int satellites = 0;
bool gpsValid = false;
bool nightMode = false;
uint32_t lastUpdateTime = 0;

// Gauge parameters
#define GAUGE_CENTER_X 120
#define GAUGE_CENTER_Y 140
#define GAUGE_RADIUS 90
#define NEEDLE_LENGTH 70
#define MAX_SPEED 180
#define MIN_SPEED 0
#define SPEED_SMOOTHING 0.15
#define NEEDLE_WIDTH 7

// Layout parameters
#define DIGITAL_X 120
#define DIGITAL_Y 40
#define INFO_X 10
#define INFO_Y 200
#define MAX_SPEED_X 200
#define MAX_SPEED_Y 40
```

```
void setup() {
  Serial.begin(115200);
  SerialGPS.begin(9600, SERIAL_8N1, 16, 17);

  // Initialize TFT display
  tft.init();
  tft.setRotation(1); // Landscape with USB port on the left
  tft.fillScreen(TFT_BACKGROUND);

  // Draw static elements
  drawGaugeBackground();
  drawDataFields();

  targetTime = millis() + 1000;
}

void loop() {
  // Read GPS data
  while (SerialGPS.available() > 0) {
    gps.encode(SerialGPS.read());
  }

  // Update display every 50ms for smooth animation
  if (millis() >= targetTime) {
    targetTime = millis() + 50;
    lastUpdateTime = millis();

    // Get GPS data
    if (gps.location.isValid() && gps.speed.isValid()) {
      targetSpeed = gps.speed.kmph();
      if (targetSpeed > maxSpeed) maxSpeed = targetSpeed;

      // Calculate trip distance if moving
      if (targetSpeed > 2) { // 2 km/h threshold
        tripDistance += (targetSpeed / 3600.0) * (50.0 / 1000.0); // km
      }

      lat = gps.location.lat();
      lon = gps.location.lng();
      satellites = gps.satellites.value();
      gpsValid = true;
    } else {
      gpsValid = false;
      targetSpeed = 0;
    }

    // Smooth speed transition
    currentSpeed = currentSpeed + (targetSpeed - currentSpeed) * SPEED_SMOOTHING;

    // Check for no GPS signal after 5 seconds
    if (millis() > 5000 && gps.charsProcessed() < 10) {
      showNoSignalScreen();
    } else {
      // Update display
```

```
      updateDisplay();
    }
  }
}

void showNoSignalScreen() {
  tft.fillScreen(TFT_BACKGROUND);
  tft.setTextColor(TFT_ORANGE, TFT_BACKGROUND);
  tft.setTextSize(2);
  tft.setTextDatum(MC_DATUM);
  tft.drawString("NO GPS SIGNAL", 120, 100);
  tft.setTextSize(1);
  tft.drawString("Check GPS module connection", 120, 130);
  delay(1000);
  drawGaugeBackground();
  drawDataFields();
  currentSpeed = 0;
  targetSpeed = 0;
  gpsValid = false;
}

void drawGaugeBackground() {
  // Clear screen with background color
  tft.fillScreen(TFT_BACKGROUND);

  // Draw outer arc with 3D effect
  for (int r = GAUGE_RADIUS; r > GAUGE_RADIUS - 5; r--) {
    uint16_t color = (r == GAUGE_RADIUS) ? TFT_LIGHTGRAY :
                     (r == GAUGE_RADIUS - 1) ? TFT_DARKGRAY :
                     (r == GAUGE_RADIUS - 2) ? 0x2104 : TFT_BACKGROUND;
    tft.drawSmoothArc(GAUGE_CENTER_X, GAUGE_CENTER_Y, r, r - 8, 0, 180, color, TFT_BACKGROUND);
  }

  // Draw colored zones with gradient effect
  drawGradientArcSegment(0, 60, TFT_RED, TFT_ORANGE);    // 120-180 km/h
  drawGradientArcSegment(60, 120, TFT_ORANGE, TFT_YELLOW); // 60-120 km/h
  drawGradientArcSegment(120, 180, TFT_YELLOW, TFT_GREEN);  // 0-60 km/h

  // Draw major ticks with 3D effect
  tft.setTextColor(TFT_LIGHTGRAY, TFT_BACKGROUND);
  tft.setTextSize(1);

  for (int i = 0; i <= MAX_SPEED; i += 20) {
    float angle = map(i, 0, MAX_SPEED, 0, 180);
    float radAngle = radians(angle);

    // Tick marks
    int x1 = GAUGE_CENTER_X + (GAUGE_RADIUS - 15) * cos(radAngle - PI / 2);
    int y1 = GAUGE_CENTER_Y + (GAUGE_RADIUS - 15) * sin(radAngle - PI / 2);
    int x2 = GAUGE_CENTER_X + GAUGE_RADIUS * cos(radAngle - PI / 2);
    int y2 = GAUGE_CENTER_Y + GAUGE_RADIUS * sin(radAngle - PI / 2);

    // 3D effect
    tft.drawLine(x1, y1, x2, y2, TFT_LIGHTGRAY);
    tft.drawLine(x1+1, y1+1, x2+1, y2+1, TFT_DARKGRAY);
```

```
    // Labels for major ticks
    if (i % 60 == 0) {
      int labelX = GAUGE_CENTER_X + (GAUGE_RADIUS - 30) * cos(radAngle - PI / 2);
      int labelY = GAUGE_CENTER_Y + (GAUGE_RADIUS - 30) * sin(radAngle - PI / 2);
      tft.setTextDatum(MC_DATUM);
      tft.drawNumber(i, labelX, labelY);
    }
  }

  // Draw center cap with 3D effect
  tft.fillCircle(GAUGE_CENTER_X, GAUGE_CENTER_Y, 10, TFT_DARKGRAY);
  tft.drawCircle(GAUGE_CENTER_X, GAUGE_CENTER_Y, 10, TFT_LIGHTGRAY);
  tft.fillCircle(GAUGE_CENTER_X, GAUGE_CENTER_Y, 8, TFT_BLACK);
}

void drawGradientArcSegment(int startAngle, int endAngle, uint16_t startColor, uint16_t endColor) {
  int segments = endAngle - startAngle;
  for (int i = 0; i < segments; i++) {
    int angle = startAngle + i;
    float ratio = (float)i / (float)segments;
    uint16_t color = interpolateColor(startColor, endColor, ratio);

    float radAngle = radians(angle);
    int x1 = GAUGE_CENTER_X + (GAUGE_RADIUS - 20) * cos(radAngle - PI / 2);
    int y1 = GAUGE_CENTER_Y + (GAUGE_RADIUS - 20) * sin(radAngle - PI / 2);
    int x2 = GAUGE_CENTER_X + (GAUGE_RADIUS - 10) * cos(radAngle - PI / 2);
    int y2 = GAUGE_CENTER_Y + (GAUGE_RADIUS - 10) * sin(radAngle - PI / 2);

    tft.drawLine(x1, y1, x2, y2, color);
  }
}

uint16_t interpolateColor(uint16_t color1, uint16_t color2, float ratio) {
  uint8_t r1 = (color1 >> 11) & 0x1F;
  uint8_t g1 = (color1 >> 5) & 0x3F;
  uint8_t b1 = color1 & 0x1F;

  uint8_t r2 = (color2 >> 11) & 0x1F;
  uint8_t g2 = (color2 >> 5) & 0x3F;
  uint8_t b2 = color2 & 0x1F;

  uint8_t r = r1 + (r2 - r1) * ratio;
  uint8_t g = g1 + (g2 - g1) * ratio;
  uint8_t b = b1 + (b2 - b1) * ratio;

  return (r << 11) | (g << 5) | b;
}

void drawDataFields() {
  // Draw digital speed background
  tft.fillRoundRect(DIGITAL_X - 60, DIGITAL_Y - 25, 120, 50, 5, TFT_DARKBLUE);
  tft.drawRoundRect(DIGITAL_X - 60, DIGITAL_Y - 25, 120, 50, 5, TFT_CYAN);

  // Draw max speed background
```

```
  tft.fillRoundRect(MAX_SPEED_X - 40, MAX_SPEED_Y - 15, 80, 30, 5, TFT_DARKBLUE);
  tft.drawRoundRect(MAX_SPEED_X - 40, MAX_SPEED_Y - 15, 80, 30, 5, TFT_CYAN);

  // Draw GPS info background
  tft.fillRoundRect(INFO_X, INFO_Y - 5, 220, 40, 5, TFT_DARKBLUE);
  tft.drawRoundRect(INFO_X, INFO_Y - 5, 220, 40, 5, TFT_CYAN);

  // Draw static labels
  tft.setTextColor(TFT_CYAN, TFT_DARKBLUE);
  tft.setTextSize(1);
  tft.setTextDatum(MC_DATUM);
  tft.drawString("MAX", MAX_SPEED_X, MAX_SPEED_Y - 8);
  tft.drawString("km/h", DIGITAL_X, DIGITAL_Y + 15);
}

void updateDisplay() {
  // Limit speed to gauge range
  if (currentSpeed > MAX_SPEED) currentSpeed = MAX_SPEED;
  if (currentSpeed < MIN_SPEED) currentSpeed = MIN_SPEED;

  // Update digital speed display
  tft.setTextColor(TFT_CYAN, TFT_DARKBLUE);
  tft.setTextSize(4);
  tft.setTextDatum(MC_DATUM);
  tft.drawString(String(currentSpeed, 1), DIGITAL_X, DIGITAL_Y - 10);

  // Update max speed display
  tft.setTextSize(2);
  tft.drawString(String(maxSpeed, 0), MAX_SPEED_X, MAX_SPEED_Y + 8);

  // Update needle position
  drawNeedle(currentSpeed);

  // Update GPS info
  updateGPSInfo();
}

void updateGPSInfo() {
  tft.setTextColor(TFT_LIGHTGRAY, TFT_DARKBLUE);
  tft.setTextSize(1);
  tft.setTextDatum(TL_DATUM);

  // GPS status with icon
  tft.fillRect(INFO_X + 5, INFO_Y, 100, 10, TFT_DARKBLUE);
  tft.drawString("GPS:", INFO_X + 5, INFO_Y);
  tft.drawString(gpsValid ? "Valid" : "Invalid", INFO_X + 40, INFO_Y);
  tft.fillCircle(INFO_X + 30, INFO_Y + 5, 3, gpsValid ? TFT_GREEN : TFT_RED);

  // Satellite count with quality indicator
  tft.fillRect(INFO_X + 120, INFO_Y, 100, 10, TFT_DARKBLUE);
  tft.drawString("SAT:", INFO_X + 120, INFO_Y);
  tft.drawString(String(satellites), INFO_X + 155, INFO_Y);
  uint16_t fixColor = satellites >= 6 ? TFT_GREEN : (satellites >= 3 ? TFT_YELLOW : TFT_RED);
  tft.fillCircle(INFO_X + 175, INFO_Y + 5, 3, fixColor);
```

```
  // Coordinates
  tft.fillRect(INFO_X + 5, INFO_Y + 15, 210, 10, TFT_DARKBLUE);
  tft.drawString("LAT:", INFO_X + 5, INFO_Y + 15);
  tft.drawString("LON:", INFO_X + 120, INFO_Y + 15);

  if (gpsValid) {
    tft.drawString(String(lat, 5), INFO_X + 35, INFO_Y + 15);
    tft.drawString(String(lon, 5), INFO_X + 150, INFO_Y + 15);
  } else {
    tft.drawString("---", INFO_X + 35, INFO_Y + 15);
    tft.drawString("---", INFO_X + 150, INFO_Y + 15);
  }

  // Trip distance
  tft.fillRect(INFO_X + 5, INFO_Y + 30, 210, 10, TFT_DARKBLUE);
  tft.drawString("TRIP:", INFO_X + 5, INFO_Y + 30);
  tft.drawString(String(tripDistance, 2) + " km", INFO_X + 40, INFO_Y + 30);
}

void drawNeedle(float speed) {
  static float oldAngle = 0;
  float angle = map(speed, 0, MAX_SPEED, 0, 180);

  // Only redraw if angle changed significantly
  if (abs(angle - oldAngle) > 0.5) {
    // Erase old needle
    drawNeedleTriangle(oldAngle, TFT_BACKGROUND);

    // Draw new needle with shadow effect
    drawNeedleTriangle(angle - 1, TFT_DARKGRAY); // Shadow
    drawNeedleTriangle(angle, TFT_RED); // Main needle

    oldAngle = angle;
  }
}

void drawNeedleTriangle(float angle, uint16_t color) {
  float radAngle = radians(angle - 90);
  float radAngleLeft = radians(angle - 90 + 20); // Wider angle for better visibility
  float radAngleRight = radians(angle - 90 - 20);

  // Calculate triangle vertices
  int x1 = GAUGE_CENTER_X + NEEDLE_LENGTH * cos(radAngle);
  int y1 = GAUGE_CENTER_Y + NEEDLE_LENGTH * sin(radAngle);
  int x2 = GAUGE_CENTER_X + NEEDLE_WIDTH * cos(radAngleLeft);
  int y2 = GAUGE_CENTER_Y + NEEDLE_WIDTH * sin(radAngleLeft);
  int x3 = GAUGE_CENTER_X + NEEDLE_WIDTH * cos(radAngleRight);
  int y3 = GAUGE_CENTER_Y + NEEDLE_WIDTH * sin(radAngleRight);

  // Draw filled triangle
  tft.fillTriangle(x1, y1, x2, y2, x3, y3, color);

  // Redraw center cap
  tft.fillCircle(GAUGE_CENTER_X, GAUGE_CENTER_Y, 10, TFT_DARKGRAY);
  tft.drawCircle(GAUGE_CENTER_X, GAUGE_CENTER_Y, 10, TFT_LIGHTGRAY);
```

```
  tft.fillCircle(GAUGE_CENTER_X, GAUGE_CENTER_Y, 8, TFT_BLACK);
}
```