Archives of Current Research International

Volume 24, Issue 11, Page 296-304, 2024; Article no.ACRI.123953
ISSN: 2454-7077

Smart Fuel Monitoring for Agricultural Fleet Operations Using Tilt Sensors and Cloud Data Processing

Anand B A [a++*], Kathyayini H S [b#] and Teju [a†]

[a] University of Agricultural Sciences, GKVK, Bengaluru, India.
[b] University of Agricultural Sciences, Raichur, India.

Authors' contributions

This work was carried out in collaboration among all authors. All authors read and approved the final manuscript.

Article Information

DOI: https://doi.org/10.9734/acri/2024/v24i11971

Open Peer Review History:
This journal follows the Advanced Open Peer Review policy. Identity of the Reviewers, Editor(s) and additional Reviewers, peer review comments, different versions of the manuscript, comments of the editors, etc are available here: https://www.sdiarticle5.com/review-history/123953

Original Research Article

Received: 28/08/2024
Accepted: 30/10/2024
Published: 16/11/2024

ABSTRACT

This paper presents a novel Internet of Things (IoT)-based fuel monitoring and tracking system designed to address limitations of traditional fuel gauges and enhance fleet management efficiency. The system leverages an accelerometer and Arduino microcontroller to measure fuel level by detecting tilt angles within the tank. This data, along with real-time location from GPS, is transmitted to a cloud platform for processing and visualization. A mobile application, "Tracer," empowers users with insights into fuel level, location on a map, and distance traveled. Graphical fuel consumption patterns further aid in optimization strategies. This solution offers real-time data for improved fuel efficiency, proactive route planning, and overall fleet management. The proposed system is successfully implemented IOT based technology. This sys in Ada fruit IO cloud issued to provide information about vehicle location, fuel level in tank and distance travelled by the vehicle.

[++] Assistant Professor;
[#] Ph.D Scholar;
[†] Research Scholar;
**Corresponding author: Email: baanand1586@gmail.com;*

Cite as: *B A, Anand, Kathyayini H S, and Teju. 2024. "Smart Fuel Monitoring for Agricultural Fleet Operations Using Tilt Sensors and Cloud Data Processing". Archives of Current Research International 24 (11):296-304. https://doi.org/10.9734/acri/2024/v24i11971.*

Keywords: Fuel level monitoring; accelerometer; microcontroller; GPS; fleet management.

1. INTRODUCTION

The transportation sector significantly impacts the environment and economy, making efficient fuel management a critical concern (Chiwhane, et al., 2017). Traditional fuel gauges often lack accuracy, hindering efforts to optimize fuel consumption and vehicle performance. This research addresses these limitations by presenting an Internet of Things (IoT)-based fuel monitoring and tracking system. This innovative system leverages advancements in sensor technology, wireless communication, and cloud computing to provide a holistic solution. It offers real-time fuel level monitoring, vehicle location tracking, and mileage calculation. By capturing and analyzing these critical data points, the system empowers stakeholders. This includes fleet managers and individual vehicle owners to make data-driven decisions. These data-driven decisions can optimize fuel efficiency, reduce operational costs, and minimize environmental impact (Sørensen & Bochtis, 2010).

The system achieves this by integrating a combination of hardware and software components. These components include accelerometers, GPS modules, microcontrollers, software applications, and cloud-based infrastructure (Serianni et al., 2019). This integration offers a comprehensive solution for real-time fuel level monitoring, vehicle location tracking, and mileage calculation (Shinde, et al.,2015; Hussain et al., 2020). The system aims to provide valuable insights into fuel consumption patterns, enabling data-driven decision-making for optimizing fleet management and reducing operational costs. The following sections delve into the specific hardware and software components employed, their integration, and the system's overall functionality.

2. METHODOLOGY

This section investigates the methodology behind a system designed for IoT-based digital fuel monitoring and tracking systems. The system utilizes various components including the ESP8266 Wi-Fi module, Arduino boards, GPS modules, and accelerometers (Fig. 1).

The ESP8266 is a cost-effective Wi-Fi microcontroller that serves as the core for data collection, storage, and transmission to a cloud server via Wi-Fi. It offers features like integrated cache memory for optimized performance, the ability to function as a Wi-Fi adapter for microcontrollers and secure data transmission with encryption algorithms. The Arduino is an open-source electronics platform that provides user-friendly hardware and software for interacting with various sensors and actuators (Almishari, et al., 2017). It is popular due to its affordability, cross-platform compatibility, easy-to-use programming environment, and open-source nature allowing for customization. The GPS comprises three segments: space, control, and user. The user segment includes GPS receivers that process satellite signals to determine location (latitude, longitude) and time (Mistary, P. V and Chile, R.H. 2015). The ADXL335 is a three-axis accelerometer used to measure static and dynamic accelerations. A capacitor is used to store electrical charge.

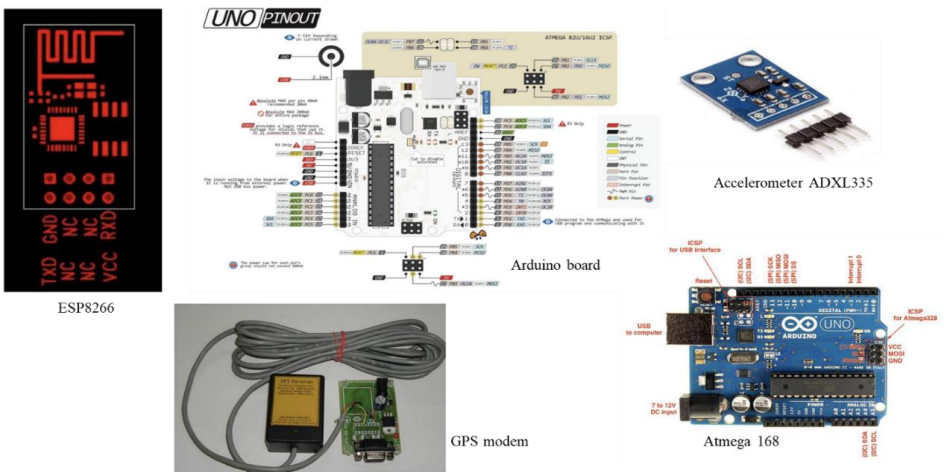

Fig. 1. Components of digital fuel level detector

3. APP GENERATION

Task 1: Install Android Studio: Specifically designed to facilitate the creation of Android applications, Android Studio serves as Google's official Integrated Development Environment (IDE). It offers a comprehensive suite of tools that streamline the entire app development process (Khatun et al., 2019). These tools encompass code editing with intelligent code completion and refactoring capabilities, project structuring with customizable templates to jumpstart development, a robust debugger to pinpoint and resolve coding issues, a testing framework to ensure app functionality and performance, and performance optimization profilers to identify and rectify bottlenecks within the code. Android Studio empowers developers to test applications on a variety of emulators or physical devices, and ultimately generate production-ready APKs for distribution.

Note: Given the continuous evolution of Android Studio, it is imperative to consult the official documentation at developer.android.com for the most current system requirements and installation guidelines.

To initiate Android Studio development:

- Java Development Kit (JDK): Ensure Java 7 or a later version is installed on the system.
- Android Studio Installation: Download and install Android Studio for Windows, macOS, or Linux, following the platform-specific instructions.

By establishing Android Studio as the development environment, developers gain access to a robust platform for creating, testing, and refining Android applications.

a) Installing the Java Development Kit

- On computer, open a terminal window.
- Type java -version

The output includes a line: Java (™) SE Runtime Environment (build1.X.0_05-b13) and X is the version number to look at.

- To install Android Studio, The system requires Android version 7 or later.
- If Java SE version is below 7 or not installed, you must install the latest Java SE Development Kit (JDK) before proceeding with Android Studio.

Installing Java SE Development Kit (JDK):

- Download the JDK from the Oracle Java SE website.
- Accept the License Agreement included in the JDK package.
- Download the JDK version compatible with the development machine (avoid demos and samples).
- Install the JDK. This process typically takes a few minutes.
- Verify the installation by opening a terminal window and typing java -version.
- Set the JAVA_HOME environment variable to point to the JDK installation directory.

Once Java is correctly installed, you can proceed with the Android Studio installation.

Note: Clear and concise instructions with numbered steps improve readability and followability.

Windows:

- Access the system environment variables. On Windows, navigate to System -> Advanced System Settings -> Environment Variables. On macOS or Linux, consult the users system's documentation.
- Create a new system variable named JAVA_HOME.
- Set the variable value to the exact directory path of the users JDK installation (e.g., C:\Program Files\Java\jdk1.8.0_202).
- If a JAVA_HOME variable already exists, modify its value to match the users JDK path.
- Open a command prompt and execute echo %JAVA_HOME% to verify the correct path.

b) Installing Android Studio

To establish the Android development environment, follow these steps:

1. Ignite the users journey as an Android app developer by acquiring the latest version of Android Studio. The official Android developer website is the users one-stop shop for the download. Android Studio

functions as the users comprehensive development environment, empowering you with the essential tools and functionalities to materialize the users app concepts into reality.

2. Installation:

- Execute the downloaded installer.
- Accept the default settings throughout the installation process.
- Ensure all available components are selected for installation.

3. Additional Component Download: Upon installation completion, the Android Studio Setup Wizard will initiate the download of supplementary components. This process may be time-consuming, depending on internet speed.
4. Once the download has finished, Android Studio will automatically launch, eagerly awaiting the users input to begin crafting the users first Android project. This is gateway to the exciting world of Android development, where you'll bring the users app ideas to life.

By adhering to these steps and exercising patience during the component download phase, you will successfully set up Android Studio for application development.

Task 2: create a "Hello World" app:

To embark on Android development adventure, we'll begin by crafting a fundamental "Hello World" application. This application, though simple, serves as a springboard to validate a successful Android Studio installation and introduce core Android development concepts (Fig. 2).

1. Launch Android Studio: Open Android Studio if it is not already running.
2. New Project: Initiate a new Android Studio project by clicking "Start a new Android Studio project" on the welcome screen.

3. Project Configuration:

- Assign a name to the application (e.g., "HelloWorld").
- Specify the desired project location.
- Choose a unique company domain.
- Select "Phone and Tablet" as the target Android devices.
- Set the minimum SDK to API 15: Android 4.0.3 Ice Cream Sandwich.
- If required, Android Studio will automatically install additional components for the chosen target SDK.

4. Activity creation:

- Opt for the "Empty Activity" template for the simplest project structure.
- Name the main activity "MainActivity" (recommended).
- Ensure the "Generate Layout file" and "Backwards Compatibility (App Compat)" options are checked.
- Accept the default layout name "activity_main" and click "Finish."

(By following these steps, a new Android Studio project is generated, providing a foundation for building the "Hello World" application).

5. Upon completing the project creation steps, Android Studio undertakes the following actions:

- Project Directory Creation: Generates a dedicated folder to house the newly created Android Studio project.
- Gradle Build System Integration: Incorporates the Gradle build system to manage project compilation and dependencies.
- Code Editor Interface: Opens the code editor, providing access to project files.
- Introductory Tips: Displays initial tips and recommendations to familiarize the user with the development environment.
- Android Studio offers a comprehensive range of keyboard shortcuts to enhance productivity. The provided tips serve as a valuable resource for gradually acquiring proficiency in these shortcuts.

Note: Gradle is a build automation tool widely used for Android projects. For in-depth configuration details, refer to the official Android developer documentation.

Task 3: Explore the project structure:

This section explores the organizational structure of Android Studio projects, focusing on the key components within the "Hello World" application template.

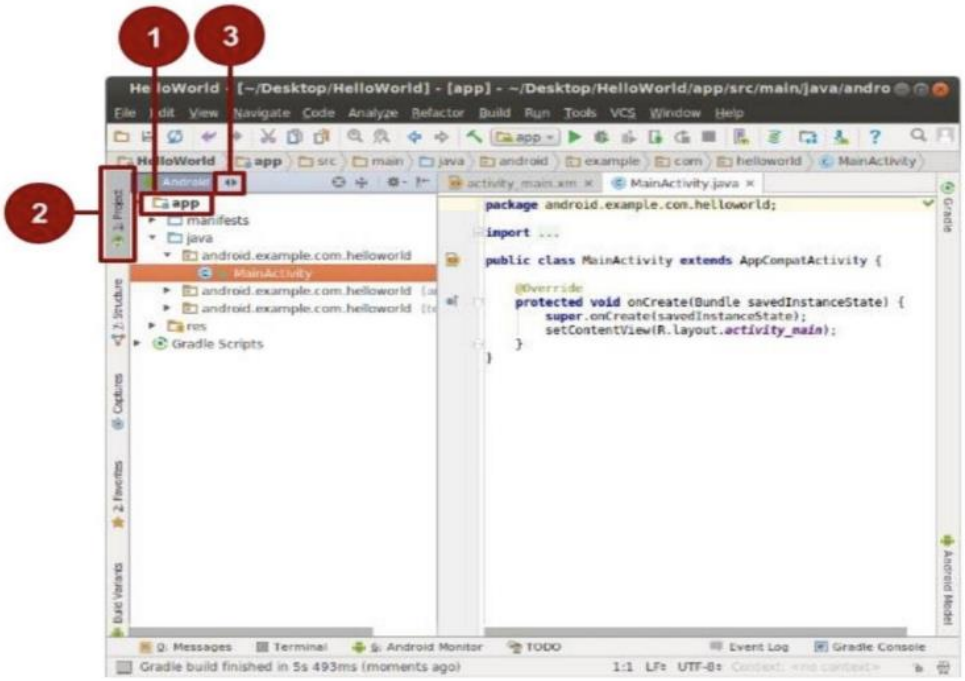

Fig. 2. App generation (Creating "Hello World")

a) Project structure and layout

The Android Studio project is hierarchically organized into several primary directories:

1. **manifests:** Contains the AndroidManifest.xml file, which outlines the app's components and configuration settings.
2. **java:** Houses Java source code files, categorized by package structure.
3. **res:** Stores non-code resources such as images, layout files, strings, and styles.

Core Directory Breakdown:

Java Directory:

i. Package Structure: Java code is organized into packages, with each package residing in a corresponding folder. For example, the com.example.hello.helloworld package contains the MainActivity.java file for the main activity.
ii. Test Code: The test and android Test subdirectories hold unit tests and instrumented tests, respectively.

Res Directory:

iii. Drawable: Stores image assets used within the application.
iv. Layout: Contains XML-based layout files defining the user interface for each activity. The activity_main.xml file is the default layout for the main activity.
v. mipmap: Houses launcher icons for different screen densities, ensuring optimal display across various devices.
vi. values: Includes XML files for defining strings, dimensions, colors, and styles, promoting code reusability and maintainability.

By understanding this project structure, developers can effectively manage and organize the various components of their Android applications.

Note: The res directory plays a crucial role in providing a structured approach to managing app resources, enhancing code readability, and facilitating localization efforts.

b) The Gradle build system

Android Studio employs Gradle as its primary build system. This robust tool automates the build process, enabling efficient management and customization. As development progresses, a deeper understanding of Gradle's capabilities becomes essential. The Gradle Scripts folder houses configuration files crucial for the build

process. Of particular importance is the build. gradle (Module:app) file. This file serves as the primary configuration point for app-specific dependencies. When incorporating external libraries or modules, their inclusion is typically defined within this file. By exploring the Gradle build system and effectively utilizing the build. gradle file, developers can optimize the app's build process and incorporate additional functionalities as needed.

Task 4: Create a virtual device (emulator): An Android Virtual Device (AVD) is a simulated Android device environment used for app testing and development. The AVD Manager facilitates the creation and management of these virtual devices.

To establish an AVD:

1. Access AVD Manager: Launch the AVD Manager from Android Studio (Tools -> Android -> AVD Manager).
2. Create New Device: Click the "Create Virtual Device" button to initiate the configuration process.
3. Select Hardware: Choose a preconfigured hardware device from the available options. Factors such as screen size, resolution, and pixel density influence this selection. For instance, the Nexus 5 with xxhdpi density requires using launcher icons from the xxhdpi folder and corresponding layout and drawable resources.
4. System Image Selection: Specify the desired Android system version for the virtual device. Multiple system images may be available, including recommended, x86, and other image categories. Download required system images if necessary.
5. Configuration Verification: Review the device configuration and finalize the creation process.

By defining the hardware specifications and choosing an appropriate system image, developers can create realistic virtual device environments for comprehensive app testing.

Note: AVDs offer a flexible platform for testing app compatibility across various device configurations without requiring physical hardware.

Task 5: App development and testing: To execute the "Hello World" application, the following steps are undertaken:

1. **App Launch:** Initiate the app execution process by navigating to **Run -> Run app** or clicking the corresponding toolbar icon within Android Studio.
2. **Emulator Selection:** Choose the "Nexus 5 API 23" emulator as the deployment target from the available emulators.
3. **Emulator Initialization:** The emulator starts, simulating a physical device's boot-up process. This can be time-consuming, depending on system resources. Upon completion, Android Studio deploys and launches the app.
4. **App Verification:** The "Hello World" app's user interface should be displayed within the emulator, confirming successful deployment and execution.

Note: To optimize testing efficiency, it is recommended to keep the emulator running between app iterations to avoid redundant boot-up processes.

- Emulator Customization and Compatibility: The AVD Manager offers customization options for creating tailored virtual device configurations. Developers can experiment with different hardware specifications and system images to simulate various device scenarios. However, it's essential to be aware that not all hardware-system image combinations are compatible, potentially impacting app execution. By effectively utilizing the AVD Manager and understanding compatibility constraints, developers can create realistic testing environments to ensure app robustness across diverse Android devices.

Task 6: Add a log statement to the app: Log statements are essential tools for debugging and monitoring application behavior. By strategically placing log messages within the code, developers can track variable values, and execution flow, and identify potential exceptions (Fig. 3). The Android Monitor provides a platform for viewing log messages generated by the app during runtime. To access the Android Monitor:

- **Launch Android Monitor:** Click the dedicated Android Monitor button located at the bottom of the Android Studio interface. The default view, Logcat, displays real-time log messages from the app.
- **Adjust Log Level:** Modify the log level from the default "Verbose" to "Debug" using the dropdown menu. This filtering option allows for focused viewing of debug-level messages.

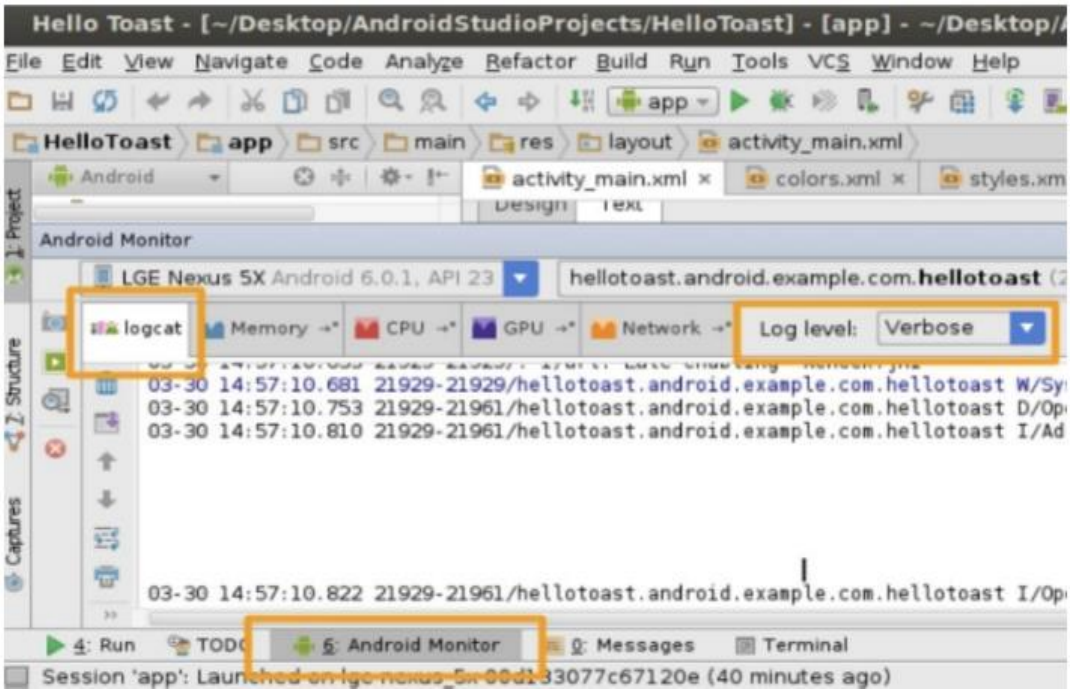

Fig. 3. App generation (Adding log statement to the app)

By incorporating log statements and effectively using the Android Monitor, developers can gain valuable insights into app execution and efficiently troubleshoot issues.

4. SYSTEM ARCHITECTURE

A new digital fuel level indicator system is proposed to tackle the inaccuracy issues of conventional fuel gauges (Fig. 4). This system utilizes an accelerometer and a floating bob to measure fuel level by detecting tilt angles. The tilt data is processed by an Arduino and converted into fuel readings. This system offers several advantages over traditional gauges: it's more accurate, provides real-time location data via GPS, calculates distance traveled, and is cost-effective due to its use of common components. Furthermore, the system's design allows for use in various vehicles. The data is transmitted to a cloud platform for processing and then displayed on a user-friendly phone app, providing valuable information to both individual drivers and fleet managers.

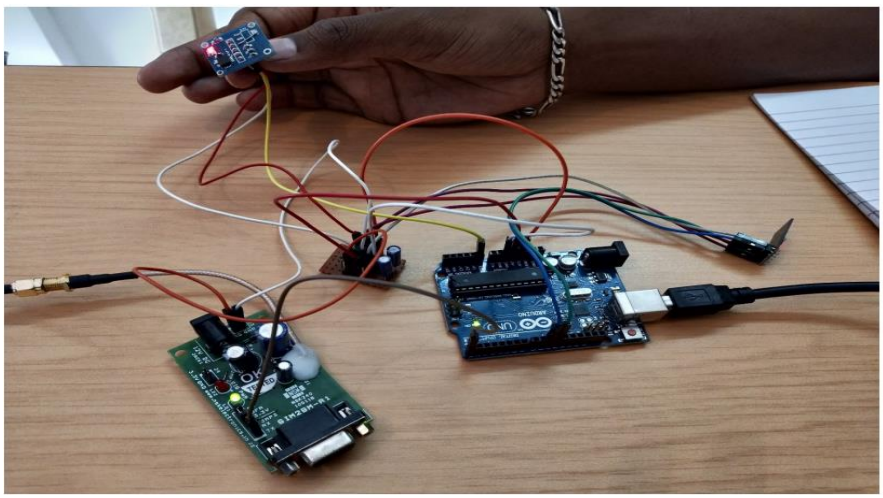

Fig. 4. System architecture section

5. RESULTS AND DISCUSSION

This paper proposes an Internet of Things (IoT)--based fuel monitoring and tracking system to improve fleet management efficiency and enable proactive journey planning. The system addresses two key challenges in fleet management: fuel optimization and route planning in dynamic traffic conditions. The system leverages real-time data on fuel levels, vehicle location, and mileage to achieve these goals. An accelerometer mounted within the fuel tank detects changes in fuel level by measuring tilt variations. An Arduino microcontroller processes this tilt data and translates it into a corresponding fuel level reading. Simultaneously, a GPS module captures the vehicle's latitude and longitude. These critical data points (fuel level, location) are then transmitted to a cloud platform via a Wi-Fi module (ESP8266) for centralized storage and processing. To facilitate real-time visualization of vehicle location, a mobile application named "Tracer" is developed (Nandimath, et al., 2017). This application connects to the cloud platform and displays the vehicle's current position on a user-friendly Google Maps interface. For enhanced data accessibility, the system updates the cloud platform every 30 seconds, ensuring users have access to the most recent information on fuel level and location through the Tracer app. Furthermore, the system generates graphical representations of fuel consumption patterns, enabling fleet managers to analyze driving behavior and identify opportunities for fuel optimization. In essence, this IoT-based solution offers a comprehensive approach to fleet management by providing detailed insights into fuel usage, vehicle location, and driving patterns (Table 1). This data empowers fleet managers to optimize fuel consumption, plan routes proactively based on real-time traffic conditions, and ultimately enhance overall fleet management efficiency.

Table 1. Details on fuel consumption, vehicle location, and driving pattern on the generated app

Tractor Tracking

Tractor ID	Fuel Level	Fuel Consumed	Distance Traveled	Latitude	Longitude	Date/Time
123	12	23	45	null	null	2019-04-28 02:55:22.0
123	12	23	45	100	200	2019-04-28 02:56:11.0
123	12	23	45	100	200	2019-04-28 02:58:02.0
123	12	23	45	100	200	2019-04-28 05:37:35.0
123	12	23	45	100	211	2019-04-28 05:37:57.0
123	12	23	45	100	200	2019-04-28 07:04:45.0
123	0	0.023	45.66	10.99	21.00	2019-04-28 07:05:15.0
123	12	23	45	150	250	2019-04-28 07:30:28.0
123	12	23	45	150	250	2019-04-28 07:40:28.0
123	12	23	45	15.880788	25.7708989	2019-04-28 07:42:44.0
123	12	23	45	15.880788	25.7708989	2019-04-28 07:56:59.0
123	12	23	45	15.880788	25.7708989	2019-04-28 08:04:59.0
123	0	0.000	0.000	13.03356	77.55791	2019-04-28 08:08:50.0
123	12	23	45	15.880788	25.7708989	2019-04-28 08:09:33.0
123	12	23	45	15.880788	25.7708989	2019-04-28 08:20:17.0
123	12	23	45	15.880788	25.7708989	2019-04-28 08:32:26.0
111	10	0.000	0.000	13.03357	77.55797	2019-04-28 08:32:52.0
111	9	0.000	0.000	13.33333	77.33333	2019-04-28 08:36:27.0
111	9	0.000	0.000	13.03362	77.55795	2019-04-28 08:37:09.0
111	0	0.000	0.000	13.03362	77.55795	2019-04-28 08:37:51.0

6. CONCLUSION

This research has investigated the methodology behind a novel digital fuel monitoring and tracking system designed for the Internet of Things (IoT) environment (Le-Tien, T., & Phung-The, V. (2010). The system leverages readily available components, including an accelerometer, Arduino microcontroller, ESP8266 Wi-Fi module, and GPS module, to collect real-time data on fuel level, vehicle location, and mileage. This data is then transmitted to a cloud platform for processing and

visualization. A user-friendly mobile application, "Tracer," provides fleet managers and individual drivers with critical insights such as current fuel level, location on a Google Maps interface, and distance traveled. Additionally, the system generates graphical representations of fuel consumption patterns, enabling further analysis and identification of fuel optimization opportunities. By offering a comprehensive approach to fleet management through detailed fuel usage, vehicle location, and driving pattern data, this IoT-based solution empowers fleet managers to optimize fuel consumption, proactively plan routes based on real-time traffic conditions, and ultimately enhance overall fleet management efficiency.

DISCLAIMER (ARTIFICIAL INTELLIGENCE)

Author(s) hereby declare that NO generative AI technologies such as Large Language Models (ChatGPT, COPILOT, etc.) and text-to-image generators have been used during the writing or editing of this manuscript.

COMPETING INTERESTS

Authors have declared that no competing interests exist.

REFERENCES

Almishari, S., Ababtein, N., Dash, P., & Naik, K. (2017). An energy efficient real-time vehicle tracking system. In *2017 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)* (pp. 1–6). IEEE. https://doi.org/10.1109/PACRIM.2017.8121884

Chiwhane, S. A., Deepa, M., & Shweta, K. (2017). IoT based fuel monitoring for future vehicles. *International Journal of Advanced Research in Computer and Communication Engineering, 6*, 295–297.

Khatun, R., Antor, S. A., Ullah, A., & Hossain, A. (2019). Vehicle fuel activities monitoring system using IoT. *Advances in Internet of Things, 9*(4), 63–71. https://doi.org/10.4236/ait.2019.94005

Le-Tien, T., & Phung-The, V. (2010). Routing and tracking system for mobile vehicles in large area. In *2010 Fifth IEEE International Symposium on Electronic Design, Test & Applications* (pp. 297–300). IEEE. https://doi.org/10.1109/DELTA.2010.38

Mistary, P. V., & Chile, R. H. (2015). Real time vehicle tracking system based on ARM7 GPS and GSM technology. In *2015 Annual IEEE India Conference (INDICON)* (pp. 1–6). IEEE. https://doi.org/10.1109/INDICON.2015.7443571

Nandimath, J. N., Sayali, J., & Pradnya, C. (2017). IoT based fuel monitoring for future vehicles. *International Research Journal of Engineering and Technology (IRJET), 4*(2), 1361–1363.

Shinde, P. A., Mane, Y. B., & Tarange, P. H. (2015). Real time vehicle monitoring and tracking system based on embedded Linux board and android application. In *2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015]* (pp. 1–7). IEEE. https://doi.org/10.1109/ICCPCT.2015.71594

Sørensen CG, Bochtis DD. Conceptual model of fleet management in agriculture. Biosystems Engineering. 2010 Jan 1;105(1):41-50.

Serianni A, Raimondo P, Palmieri N. An energy aware smart station for an UAV fleet in the smart farming application. InAutonomous Air and Ground Sensing Systems for Agricultural Optimization and Phenotyping IV 2019 May 14 (Vol. 11008, pp. 132-139). SPIE.

Hussain S, Mahmud U, Yang S. Car e-talk: An IoT-enabled cloud-assisted smart fleet maintenance system. IEEE Internet of Things Journal. 2020 Apr 8;8(12): 9484-94.

Peer-review history:
The peer review history for this paper can be accessed here:
https://www.sdiarticle5.com/review-history/123953