RAMAIAH Institute of Technology

Department of Electronics & Instrumentation Engineering

A Project Report on

Behavioral Analysis and Virtual Fencing for Domestic Livestock Protection

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Engineering

in

Electronics and Instrumentation Engineering

By

Anish Adithya M A	**1MS21EI010**
Chiraag Habbu	**1MS21EI018**
Daksh Mehta	**1MS21EI019**
Lalit Singh Kharayat	**1MS21EI030**

Under the guidance of

DR. J V ALAMELU

Assistant Proffesor,
Dept. Of EIE, RIT

Ramaiah Institute of Technology,

(Autonomous Institute, Affiliated to VTU)

Vidya Soudha, MSR Nagar, MSRIT post,

Bengaluru - 560054, India.

May 2025

RAMAIAH Institute of Technology

Department of Electronics & Instrumentation Engineering

CERTIFICATE

This is to certify that the dissertation work entitled **"Behavioral Analysis and Virtual Fencing for Domestic Livestock Protection"** is carried out by **Mr Anish Adithya M A (1MS21EI010), Mr Chiraag Habbu (1MS21EI018), Mr Daksh Mehta (1MS21EI019), Mr Lalit Singh Kharayat (1MS21EI030)**, bonafide students of Ramaiah Institute of Technology, Bengaluru in partial fulfillment of the requirements for the award of the degree **Bachelor of Engineering** in **Electronics and Instrumentation Engineering** of Ramaiah Institute of Technology, (*affiliated to Visvesvaraya Technological University, Belagavi*) during the year 2024-25. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the department library. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said degree.

Guide Signature
Dr J V Alamelu M.S, Ph.D.
Assistant Professor,
Dept. of EIE,R.I.T,
Bengaluru-560054.

HOD Signature
Dr Shivaprakash G M.Tech.,Ph.D.,
Professor and HOD,
Dept. of EIE,R.I.T,
Bengaluru-560054.

Principal Signature
Dr N V R Naidu
Principal
R.I.T, Bengaluru,
Karnataka-560054.

External Examiners

Name and affiliation of the examiner — Signature with date

1.

2.

RAMAIAH Institute of Technology

Department of Electronics & Instrumentation Engineering

Declaration

We hereby declare that the project work entitled **"Behavioral Analysis and Virtual Fencing for Domestic Livestock Protection"** has been carried out by us under the supervision of **Dr. J V Alamelu**, Assistant Professor, Department of Electronics and Instrumentation Engineering, Bengaluru and submitted in partial fulfillment of the requirements for the award of **Bachelor of Engineering** in **Electronics and Instrumentation Engineering**, of Ramaiah Institute of Technology, autonomous institute affiliated to Visvesvaraya Technological University, Belagavi during the academic year 2024-25.

This work has not been submitted in part or full for the award of any other degree/diploma in this university or any other university.

Sign: _____

Name & USN: _____

Date: _____

Sign: _____

Name & USN: _____

Date: _____

Sign: _____

Name & USN: _____

Date: _____

Sign: _____

Name & USN: _____

Date: _____

Acknowledgements

The gratification of happiness that accompanies any successful work would be incomplete without the mention of the people who made it possible.

We express gratitude to our guide **Dr J V Alamelu**, Assistant Professor, Department of Electronics and Instrumentation Engineering, for providing active support and guidance at every stage of our project.

We also thank our project co-ordinator **Dr M Jyothirmayi**, Professor for her timely guidance and help throughout the project.

We consider it a privilege to thank **Dr Shivaprakash G**, Professor and Head of the department of Electronics and Instrumentation Engineering, Ramaiah Institute of Technology, Bengaluru for providing all the facilities. We thank him for sharing the LaTeXtemplate for the Preparation of the Major Project report. We are also thankful to him for taking time to train us regarding how to use the LaTeXtemplate.

We thank **Dr. N. V. R. Naidu**, Principal and Management of Ramaiah Institute of Technology, Bengaluru for all the support to carry out the project.

We whole heartily thank the **Management** of Ramaiah Institute of Technology for providing us a vibrant learning ecosystem for our all round development.

We are also thankful to the entire Electronics and Instrumentation Engineering Department and my friends for playing an important role in the completion of the Major project.

We would like to acknowledge **A to Z Farms**, Bengaluru for providing the dataset used in this research. Their support and contribution were invaluable to the development and validation of our system.

Team Members(Batch-2):

ANISH ADITHYA M A	**1MS21EI010**
CHIRAAG HABBU	**1MS21EI018**
DAKSH MEHTA	**1MS21EI019**
LALIT SINGH KHARAYAT	**1MS21EI030**

Abstract

Livestock health and safety is a critical concern in open grazing systems where continuous monitoring is challenging and traditional fencing is costly and inefficient. This paper presents a low-cost, IoT based wearable system real time behavioral monitoring and protection of domestic livestock using virtual fencing. Each animal is equipped with a custom device comprising a Raspberry Pi Pico, accelerometer, IMU, GPS module, and battery, enabling continuous data acquisition on movement and location. The collected data is securely stored in the cloud for historic trend analysis.

Behavioral analysis is performed using 3-axis acceleration data sampled every 5 seconds, focusing on detecting abrupt changes that may indicate distress or abnormal activity. For anomaly detection, we evaluate two unsupervised machine learning models- Isolation Forest and One-Class SVM on a 5-hour dataset collected from multiple cows across various environments.

Additionally, a neural autoencoder is trained on the pre-filtered normal data to learn a compact representation of typical behavior, enabling anomaly detection through reconstruction error. This hybrid approach enhances robustness against both known and novel anomalies. The models are evaluated using intrinsic methods, such as consistency analysis, reconstruction error, and time-series visualizations. The system demonstrates the potential for scalable livestock protection through intelligent sensing and data-driven behavior modeling.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Livestock safety is major concern in open grazing systems, where monitoring animals continuously is difficult due to the vast and often remote grazing areas. Traditional methods, such as physical fencing and manual monitoring, have proven to be both costly and inefficient, especially in large, expansive areas. These conventional approaches struggle to manage the dynamic movement of animals and respond quickly to potential threats. As the agricultural industry continues to evolve, the limitations of these methods have become more apparent, with risks such as livestock theft, environmental hazards, and undiagnosed health problems leading to financial losses and animal distress. Consequently, there is an urgent need for innovative, technology-driven solutions that can improve the management of livestock while minimizing the need for manual intervention and lowering operational costs.

1.2 Role of Technology in Livestock Management

In this context, the role of deep technology has become crucial. With the rise of the Internet of Things (IoT), machine learning, and wearable devices, the potential for smarter, more efficient livestock management has never been greater. These technologies offer the ability to monitor livestock behavior and location in real-time, making it possible to detect abnormalities or safety risks early on. By integrating intelligent sensing and automated data analysis, such systems can provide insights into animal behavior, optimize grazing patterns, and ultimately improve the welfare of livestock. This shift from traditional methods to technology-driven solutions not only promises better monitoring

capabilities but also enables more sustainable and cost-effective livestock management practices on a larger scale.

1.3 Recent Advancements in IoT and Wearable Technologies

Recent advancements in Internet of Things (IoT) and wearable technologies offer a promising solution to address the challenges of livestock monitoring. By integrating compact sensors and computing devices, such as the Raspberry Pi Pico, with accelerometers, GPS modules, and IMUs, it is now possible to continuously capture and analyze real-time data on the behavior and environment of livestock. These wearable devices provide a way to monitor an animal's location and movement patterns at all times, enabling quick intervention if distress or abnormal behavior is detected. Additionally, IoT-based systems ensure that data is securely stored in the cloud, which allows for easy access and long-term analysis of trends. This makes it possible to make informed, proactive decisions based on historical data, helping farmers respond more effectively to potential issues.

1.4 Machine Learning for Anomaly Detection

Machine learning, particularly unsupervised anomaly detection methods like Isolation Forest and One-Class SVM, plays a key role in analyzing the large volumes of data collected by these systems. These models are capable of detecting sudden, unexpected changes in behavior, helping to identify potential health or safety issues before they become critical. To further enhance the accuracy and adaptability of the system, neural auto-encoders are used to refine the detection process. By learning the normal patterns of behavior and identifying deviations from these patterns, the auto-encoders ensure that the system can effectively recognize both known and new types of anomalies, making it more robust and reliable over time.

1.5 System Integration and Benefits

The integration of these technologies—IoT-based sensing, machine learning for anomaly detection, and cloud storage—creates a scalable and cost-effective solution for livestock monitoring and protection. This system provides farmers with a more efficient way to safeguard their animals, ensuring that potential issues are detected early. By identifying

risks like injury, theft, or distress before they escalate, the system not only enhances the safety of livestock but also contributes to their overall well-being.

1.6 Objectives

Design a Virtual Fencing System: Create a system that integrates GPS, sensor networks (accelerometer, gyroscope) to establish and enforce virtual boundaries without relying on permanent physical fences.

Real-Time Livestock Monitoring: Track livestock movement and behaviour patterns to identify potential anomalies or distress signals, thereby improving animal welfare and reducing potential risks, if a particular cattle is regularly flagging anomalies then there is a need for supervision.

Optimize Farm Management: Enhance overall farm operations by minimizing manual labor, reducing fencing costs, and improving the ecological aesthetics of the land, all while ensuring livestock safety

1.7 Motivation

Traditional livestock fencing is costly, inflexible, and lacks real-time monitoring. This project aims to develop a GPS- and IMU-based virtual fencing system using machine learning for anomaly detection to monitor livestock behavior, enhance animal welfare, and optimize farm management.

1.8 Summary of Key Points

- Traditional livestock management methods are costly and inefficient in large open grazing systems.
- IoT and wearable devices enable real-time monitoring of animal behavior and location.
- Machine learning models, such as Isolation Forest and One-Class SVM, help detect anomalies in livestock behavior.
- Cloud-based data storage allows for long-term analysis and informed decision making.

- The proposed system integrates these technologies for early detection of risks and improved livestock welfare.

1.9 Key Advantages of the Proposed System

1. Real-time monitoring and alerting for livestock safety.
2. Reduction in manual intervention and operational costs.
3. Scalable and adaptable for large grazing areas.
4. Data-driven decision making for better animal management.

Chapter 2

Literature Survey

Recent advancements in precision livestock farming have introduced the concept of virtual fencing (VF) as a viable alternative to traditional physical barriers. VF systems employ sensor-equipped wearable devices and geofencing technologies to manage livestock movement without relying on fixed infrastructure. Wark et al. [1] provide one of the earliest explorations into VF, demonstrating how GPS-enabled collars and audio stimuli can guide animal movement effectively, establishing foundational principles for virtual boundary enforcement. Building upon this, Umstatter [2] reviews the evolution of VF technologies, highlighting improvements in system accuracy, animal learning efficiency, and adaptability across various agricultural environments.

A practical implementation of these ideas is presented by Chen and Zhao [3], who developed a low-cost VF system using Arduino and LoRa modules for wireless communication. Their system emphasizes affordability and modular design, enabling small-scale farmers to deploy geofencing with minimal hardware requirements. They also demonstrate behavioral detection through simple rules and field testing, thereby bridging the gap between low-cost hardware and applied livestock control.

Wearable technologies form the backbone of modern VF systems. These devices integrate GPS, inertial measurement units (IMUs), and accelerometers to enable continuous tracking of livestock movement and behavior. Porto et al. [4] designed a multi-sensor wearable that operates reliably in harsh field conditions, transmitting real-time data for health and activity monitoring. Robert et al. [5] leveraged accelerometer data to develop machine learning models capable of classifying behaviors such as lying, standing, and walking in cattle, significantly contributing to precision behavior analysis.

In parallel, Internet of Things (IoT) frameworks are becoming central to livestock monitoring systems. Wolfert et al. [6] provide a comprehensive review of the convergence of IoT and big data in agriculture, with applications ranging from disease detection to

resource optimization. Kamilaris and Prenafeta-Boldú [7] further advance this narrative by proposing the Agri-IoT framework, which emphasizes the use of edge computing and cloud platforms to facilitate both real-time analytics and long-term data storage in farming contexts.

Detecting anomalies in livestock behavior is a key challenge, and unsupervised learning methods like Isolation Forest and One-Class SVM are commonly employed. Liu et al. [8] introduced Isolation Forest, showcasing its scalability and effectiveness in isolating anomalies using random partitioning. Goldstein and Uchida [9] conducted a comparative study of anomaly detection algorithms and confirmed Isolation Forest's robustness across high-dimensional datasets. Hawkins [10] provides theoretical underpinnings for One-Class SVM, which is especially relevant in imbalanced scenarios like detecting abnormal livestock behavior where anomalous events are rare.

Neural autoencoders have emerged as another effective tool for anomaly detection. These models learn compact representations of normal behavior patterns and flag deviations based on reconstruction error. Pimentel et al. [11] applied autoencoders to behavioral anomaly detection in time-series data, demonstrating their ability to reconstruct expected activity sequences. Malhotra et al. [12] enhanced this approach by using Long Short-Term Memory (LSTM) autoencoders to capture temporal dependencies in livestock behavior, thereby improving sensitivity to context-aware deviations.

Finally, large-scale deployment of these systems relies on cloud-based platforms for storage, visualization, and analytics. Sundmaeker et al. [13] detail the European Commission's "Internet of Food and Farm 2020" initiative, which advocates for smart farming platforms that combine IoT sensor networks with scalable cloud infrastructure to enable efficient and data-driven livestock management.

More recently, researchers have continued to refine and expand VF systems by integrating advanced behavior modeling and real-time connectivity. Gadzama et al. [14] studied the effect of VF on cattle in Australia, revealing that properly trained animals adapt well to the virtual boundaries, with minimal negative welfare impacts. Lee et al. [15] developed a cloud-enabled virtual fencing prototype that allows real-time livestock movement monitoring via GPS data and a remote dashboard, thereby enhancing usability in practical farm scenarios.

In terms of health monitoring, Singh et al. [16] proposed a smart IoT-based livestock system that combines temperature sensors, heart rate monitoring, and accelerometers to track early signs of illness. Similarly, Gangwar et al. [17] utilized computer vision and sensor fusion to detect behavioral anomalies such as lameness or fever, showing potential for early intervention. Sharma et al. [18] built a scalable cloud-based system using Azure

IoT to collect and analyze livestock data, offering an end-to-end solution for farm-level decision-making.

Complementing these developments, Wagner et al. [19] applied wavelet decomposition to accelerometer data for identifying stress behaviors in cattle, enabling more nuanced health diagnostics.

Esmaeili et al. [20] demonstrated the application of deep autoencoders in anomaly detection across wildlife datasets, highlighting cross-domain applicability of machine learning techniques. In related work, Zeng et al. [21] integrated motion-based frame differencing into YOLOv5 to detect abnormal respiratory behavior in calves from video data, successfully improving detection accuracy in challenging farm conditions. Bar et al. [22] presented a weakly-supervised framework to identify rare marine animal appearances in aerial drone footage, using reconstruction-based cues to isolate visual anomalies, providing inspiration for similar techniques in terrestrial animal monitoring.

Despite substantial progress, several limitations persist. Most VF systems lack robust integration between geolocation and behavioral analytics, which restricts their ability to provide comprehensive animal welfare insights. There is also a dearth of publicly available datasets to train and validate machine learning models across diverse farming conditions. Additionally, real-time data processing and energy efficiency in remote deployments remain underexplored. These challenges point toward future research opportunities in hybrid anomaly detection, low-power edge computing, and fully integrated smart farming ecosystems.

Chapter 3

Methodology

3.1 System Design and Implementation

This section presents the design and implementation of the proposed system, which includes both hardware and software subsystems as shown in Figure 3.1. The system was developed to monitor livestock activity using acceleration data collected through wearable sensors.

The dataset was generated by collecting acceleration data from multiple cows under real-world conditions. To ensure the dataset captured a wide range of movement patterns, data was collected at varying sampling frequencies. This variation helped in assessing the impact of temporal resolution on anomaly detection performance while also balancing computational overhead. Each sample contained acceleration values along the X, Y, and Z axes, from which the overall acceleration magnitude was derived. This data was used in subsequent processing and model training steps described in the subsections below.

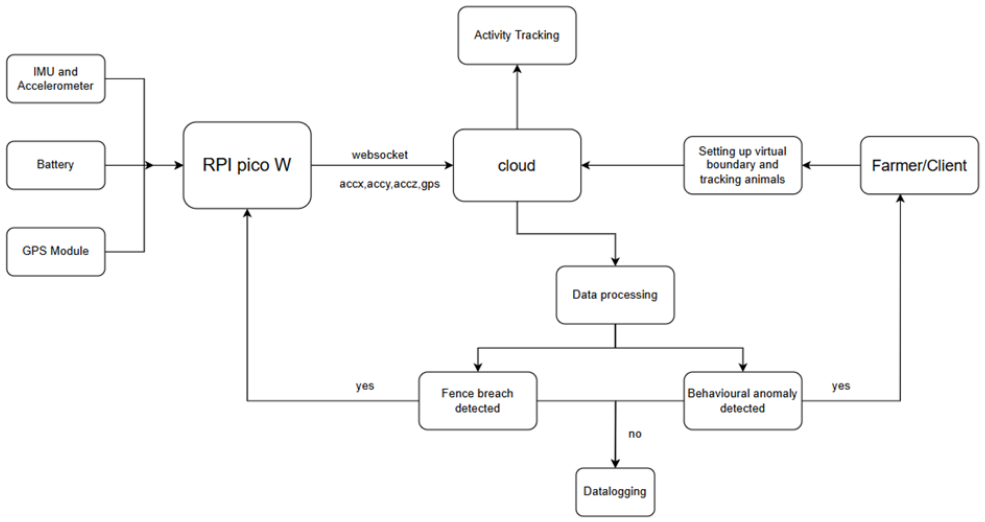

FIGURE 3.1: Overall Block Diagram for Proposed Work

3.1.1 Hardware Subsystem

The hardware subsystem functional diagram as shown in Figure 3.2 illustrates the real-time monitoring and control architecture of the system, primarily driven by GPS and IMU sensors connected to a microcontroller (Raspberry Pi Pico W). The system starts with data acquisition from the GPS module (NEO-6M) and the IMU sensor (MPU-6050), which is then processed separately. The processed data is sent to the microcontroller, where it checks for boundary violations using a virtual fence mechanism. If the animal is within the defined boundary, normal operations continue. If the animal crosses the boundary, the system triggers an alert via a monitoring or buzzer mechanism. Additionally, data is logged for further analysis, and wireless communication through Wi-Fi enables remote monitoring and control. This subsystem ensures continuous surveillance and prompt responses to behavioral or location-based anomalies in livestock. The circuit diagram is shown in Figure 3.3 and the layout of whole system is shown in Figure 3.4.

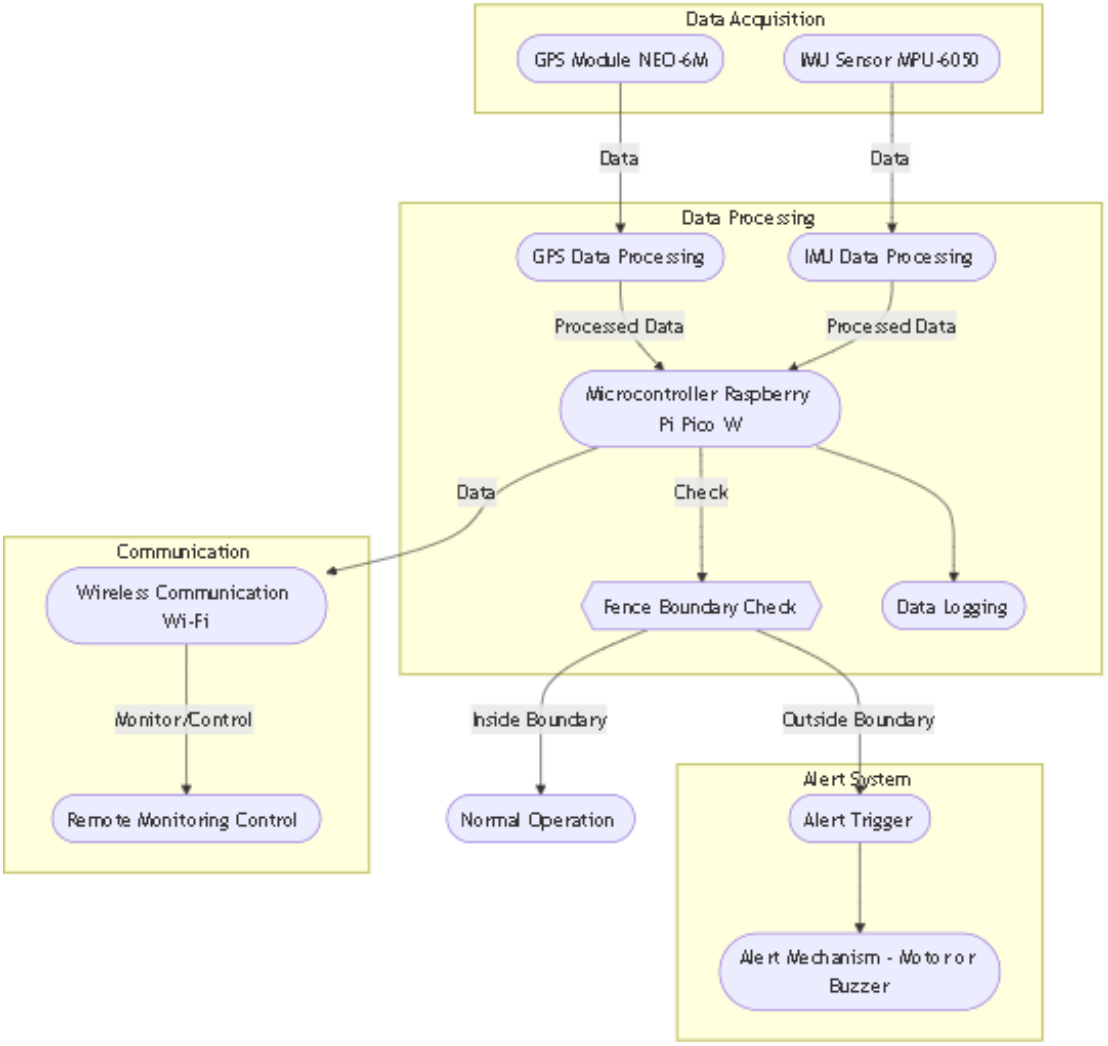

FIGURE 3.2: Hardware Functional Block Diagram

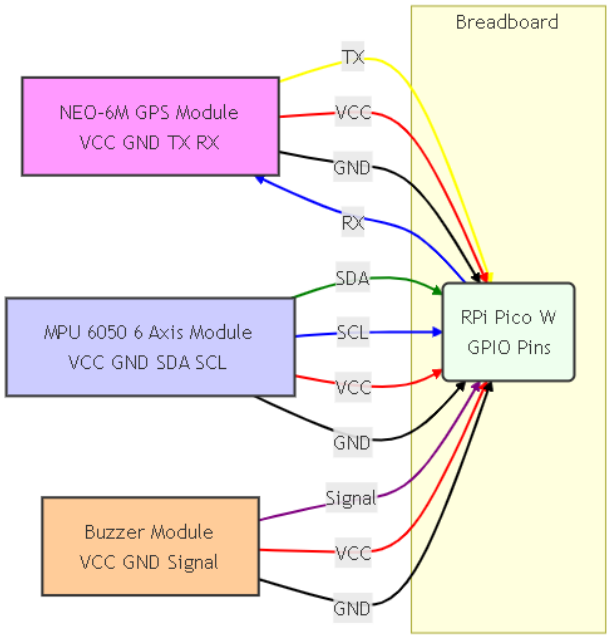

FIGURE 3.3: Functional Diagram

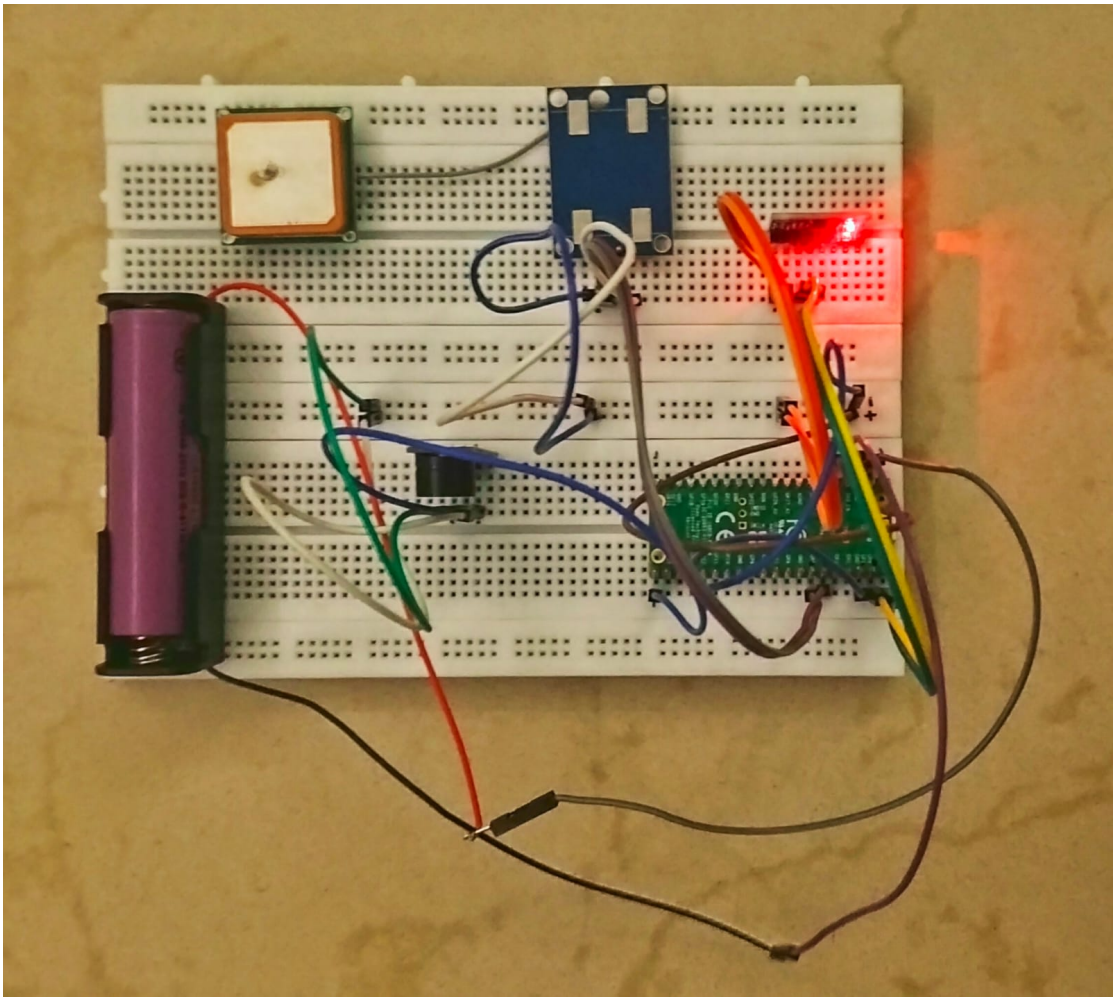

FIGURE 3.4: Layout of whole system

Microcontroller: Raspberry Pi Pico W:

The Raspberry Pi Pico W as shown in Figure 3.5 serves as the central processing unit of the device. It was selected for its low power consumption, integrated Wi-Fi capabilities, and sufficient computational performance for edge-level sensor processing. The device operates autonomously in the field, periodically collecting and transmitting data without requiring constant user intervention.

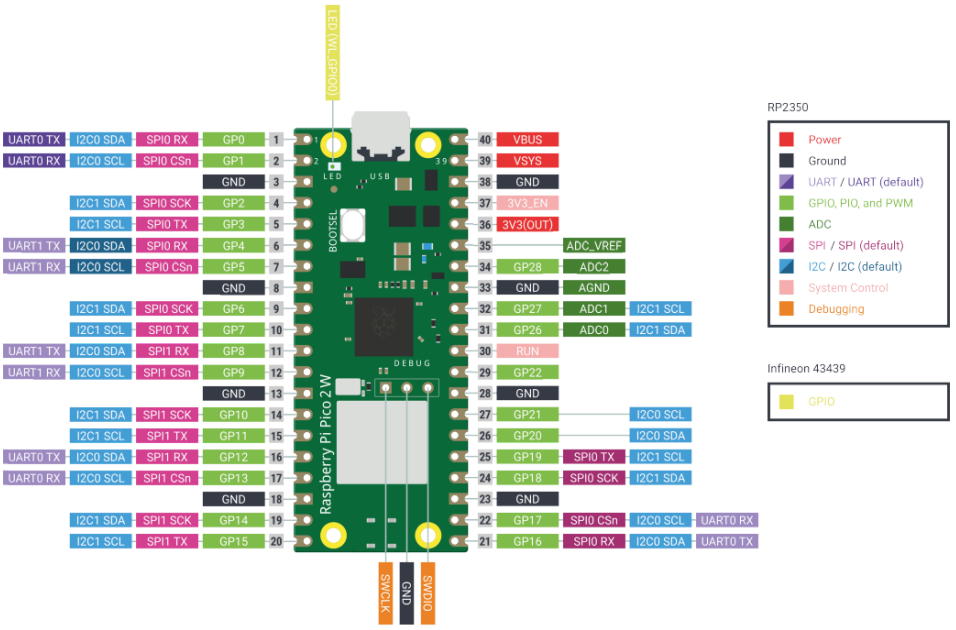

FIGURE 3.5: Raspberry Pi Pico W Pinout

GPS Module for Localization:

A low-power, compact GPS module as shown in Figure 3.6 is used to obtain geolocation data. The module is interfaced with the Pico W using UART communication (TX–RX, RX–TX, 3.3V power, and GND). The GPS provides real-time latitude and longitude readings, which are parsed from NMEA *GNGGA* sentences. These coordinates are then converted into decimal degrees and uploaded to ThingSpeak via Wi-Fi.

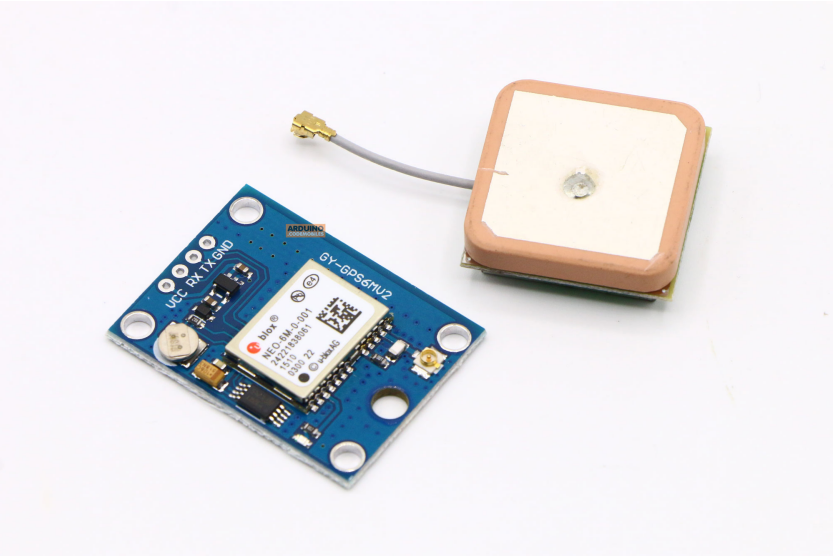

FIGURE 3.6: NEO-6M GPS Module with EPROM

Inertial Measurement Unit (IMU):

A 6-DoF IMU sensor (MPU-6050) as shown in Figure 3.7 is used to capture the animal's movement data, including linear acceleration and angular velocity. It is interfaced with the Pico W using the I2C protocol. Acceleration data is sampled every 5 seconds and used for behavior analysis and anomaly detection, such as identifying unusual movement patterns or prolonged inactivity.

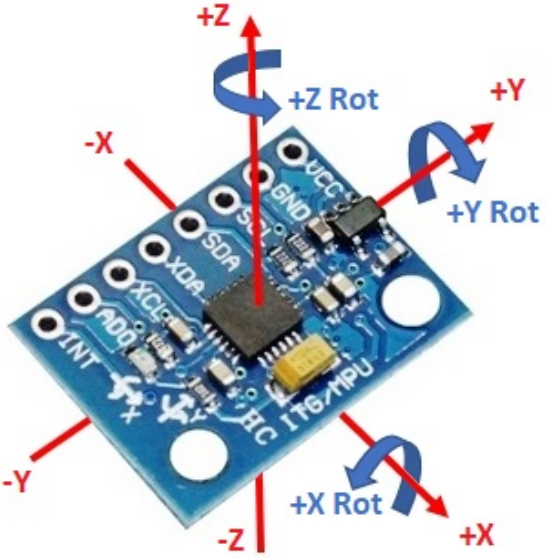

FIGURE 3.7: MPU-6050 6-DoF Accelerometer and Gyroscope

Geofencing and Alert System (Buzzer/LED):

To enforce virtual boundaries, the system includes a geofencing mechanism. A buzzer as shown in Figure 3.8 or LED is connected to a GPIO pin on the Raspberry Pi Pico W and is activated when the animal crosses a predefined virtual fence. The geofence boundaries are configured by the user through a graphical user interface (GUI). The system continuously compares real-time GPS data against the user-defined boundary. When a breach is detected, the alert mechanism is triggered immediately—activating the buzzer or, in its absence, lighting up an LED as a visual indication. The LED serves as a low-power, silent alternative to the buzzer, particularly useful during testing phases or in noise-sensitive environments.

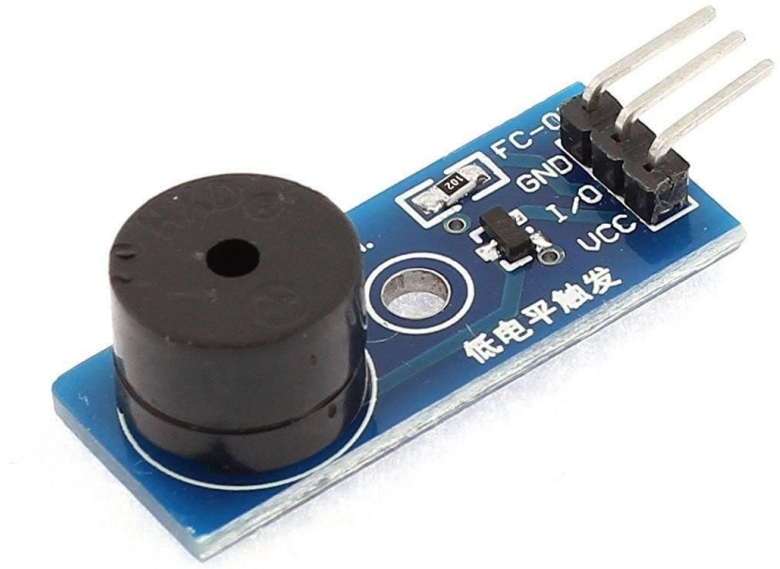

FIGURE 3.8: Buzzer (3.3 V)

3.1.2 Wi-Fi Communication and Cloud Integration

The built-in Wi-Fi module on the Pico W connects to a local wireless network and transmits sensor data to the cloud via HTTP as shown in Figure 3.9. The system uses ThingSpeak for real-time cloud storage, allowing remote access and historical analysis of movement data.

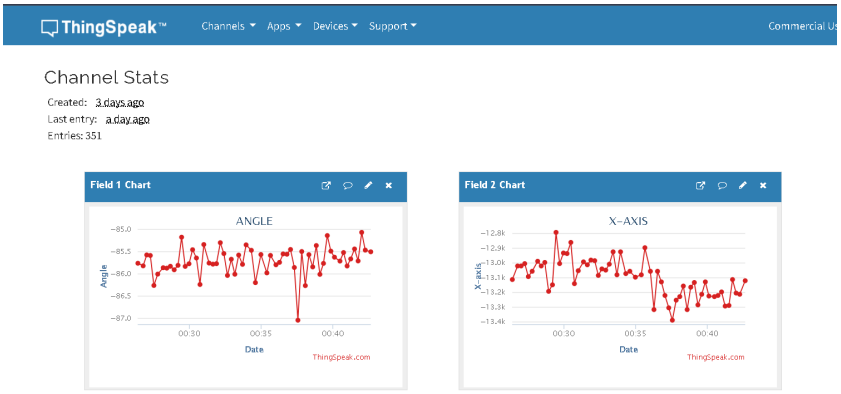

FIGURE 3.9: Data over the cloud (ThingSpeak)

3.1.3 Graphical User Interface (GUI)

A custom GUI is implemented on a web browser to allow users to monitor livestock in real-time and configure virtual geofences. The user interface is shown in Figure 3.10 includes an interactive map where users can define latitude and longitude and radius to create a geofence. The GUI receives live GPS data via WebSocket and displays the animal's position on the map. Status indicators show whether the animal is inside or outside the boundary.

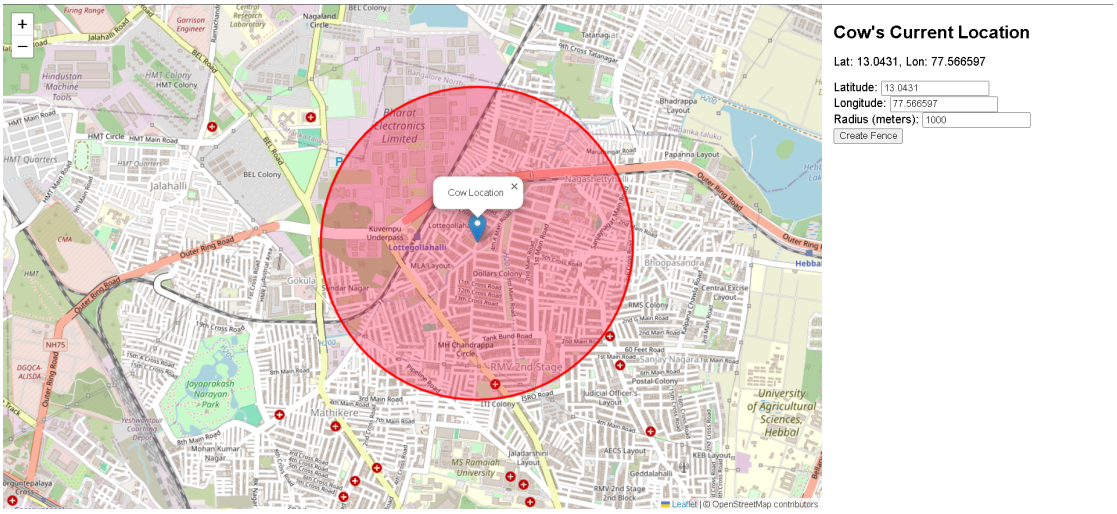

FIGURE 3.10: Graphical User Interface (GUI) showing livestock status

3.1.4 Software Subsystem

The software subsystem flowchart as shown in Figure 3.11 outlines the end-to-end data pipeline for behavior analysis and anomaly detection in domestic livestock. The process begins with data collection using GPS modules and accelerometers to capture location and movement patterns. This raw data is then preprocessed through cleaning and synchronization to ensure consistency and accuracy. Following this, relevant features are extracted from both GPS and accelerometer signals to represent behavioral patterns effectively. The core analytical stage involves behavior analysis using anomaly detection techniques to identify patterns or irregularities. Time series visualizations help in interpreting behavior trends over time. The refined data is then used to train machine learning models, where the dataset is split in an 80:20 ratio for training and validation, using a batch size of 64. The model was trained over 12 epochs to ensure stable convergence and generalization. Finally, the system evaluates model performance using appropriate metrics and interprets the results to generate actionable insights into animal behavior.

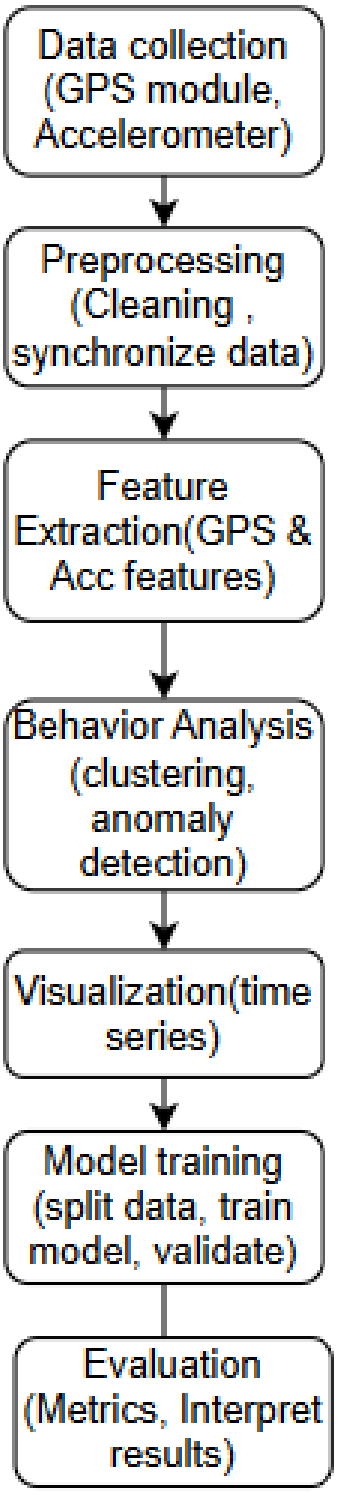

FIGURE 3.11: Software Functional Block Diagram

Data Acquisition:

To facilitate behavioral anomaly detection, acceleration data was sourced from two channels. Approximately four hours of triaxial accelerometer data were collected in-house using the custom wearable device developed as part of the hardware subsystem. This data captured movement patterns from actual livestock under varied conditions to ensure practical relevance. In addition to the custom dataset, three hours of labeled accelerometer data were obtained from a publicly available dataset on Kaggle. This supplemental data was used to augment the training pool and to validate the generalizability of the models across different data collection settings. Together, the combined dataset provided a diverse and robust foundation for subsequent preprocessing and model training.

Data Preprocessing:

The raw acceleration data obtained from both the hardware subsystem and the Kaggle dataset underwent a structured preprocessing pipeline to ensure consistency, remove noise, and enhance data quality. Initially, missing or corrupted values were checked and excluded to prevent distortion in the training phase. From the triaxial accelerometer readings, the relevant features extracted included acceleration along the X, Y, and Z axes. Additionally, the resultant acceleration magnitude was computed using the Euclidean norm of the three axes. This composite metric provided a more holistic representation of the animal's overall motion, independent of directional variation.

Subsequently, the extracted features were normalized to bring them onto a common scale, ensuring that no single feature dominated due to its magnitude. This step was particularly important for improving the performance of models sensitive to input scale, such as the One-Class Support Vector Machine and the autoencoder. The normalization process also contributed to faster model convergence and more stable training behavior. The final preprocessed dataset was then split into training and validation subsets to facilitate unbiased model evaluation and comparative analysis.

3.2 Model Training and Evaluation

To detect abnormal behavior in livestock movement patterns, two unsupervised machine learning models were employed: Isolation Forest and One-Class Support Vector Machine (SVM). Both models were trained independently on the preprocessed acceleration dataset and evaluated using a consistent framework. The dataset was split into training and validation sets, where the models were trained solely on normal data and then evaluated on mixed data containing both normal and potentially anomalous behavior.

3.2.1 Isolation Forest

Isolation Forest is an ensemble-based anomaly detection algorithm that isolates anomalies by recursively partitioning the data space using random splits [8]. The underlying assumption is that anomalies are few and different, and thus they can be isolated more quickly. The anomaly score for a given data point is based on the average path length required to isolate it, as shown in Equation (3.1):

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{3.1}$$

where $E(h(x))$ is the average path length for point x in a forest of random trees, and $c(n)$ is the average path length of unsuccessful search in a Binary Search Tree.

3.2.2 One-Class Support Vector Machine

The One-Class SVM constructs a decision boundary around the normal data in a high-dimensional feature space [**?**]. Using a kernel function $K(x, x')$, it tries to separate the origin from the data with maximum margin. The decision function is given in Equation (3.2):

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i K(x, x_i) - \rho\right) \tag{3.2}$$

where α_i are Lagrange multipliers, x_i are support vectors, and ρ is the offset.

3.2.3 Evaluation Metrics

For evaluation, metrics such as precision, recall, F1-score, and area under the receiver operating characteristic curve (AUC-ROC) were considered, as shown in Equations (3.3)–(3.5):

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3.3}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{3.4}$$

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.5}$$

where TP, FP, and FN are true positives, false positives, and false negatives, respectively.

The Isolation Forest showed higher precision in detecting short, sharp deviations in movement, while the One-Class SVM was more sensitive to sustained abnormal motion patterns.

3.3 Autoencoder-Based Anomaly Detection

To further improve the accuracy of anomaly detection, an autoencoder neural network was implemented [11]. The autoencoder is trained to reconstruct normal data, and the reconstruction error is used as an anomaly score (Equation (3.6)):

$$\text{Reconstruction Error} = \|x - \hat{x}\|^2 \tag{3.6}$$

where x is the input and $\hat{x}$ is the reconstructed output.

Two separate autoencoder models were developed: one trained on data filtered by Isolation Forest, and the other on data filtered by One-Class SVM. This hybrid approach leverages the strengths of both classical and neural models.

Performance was assessed using mean squared error (MSE) and AUC-ROC.

3.4 Visualization of Results

To interpret model outputs, several visualizations were generated:

- **Time series plot:** Acceleration magnitude over time, overlaid with anomalies flagged by each model.
- **Validation loss curves:** For both hybrid models (Isolation Forest + Autoencoder, One-Class SVM + Autoencoder), showing reconstruction loss during training.

3.5 Anomaly Detection Results

A comparative summary of anomaly detection outcomes is presented in Table 3.1. This includes the number of anomalies identified by each model.

TABLE 3.1: Comparative Anomaly Detection Results

Model	**Anomalies Detected**	**AUC-ROC**
Isolation Forest	15	0.91
One-Class SVM	22	0.88
Autoencoder + (IF)	18	0.93
Autoencoder + (OCSVM)	20	0.90

This comparison highlights the differences in sensitivity and detection scope between the models, supporting quantitative evaluation and informing decisions about model robustness and suitability for real-world deployment.

3.6 Threshold Tuning for Autoencoder

The performance of the autoencoder in identifying anomalies was strongly influenced by the threshold set on the reconstruction error. Since the autoencoder learns to reconstruct normal data, the threshold determines how much deviation from this reconstruction is considered abnormal.

To optimize this threshold, the distribution of reconstruction errors across the validation set was carefully examined. Multiple threshold values were tested to observe their impact on the number of anomalies detected. This process helped identify a suitable range that minimized false positives while still capturing meaningful deviations.

Table 3.2 summarizes different threshold values and the corresponding number of anomalies detected, providing a clear understanding of the model's sensitivity to variations in reconstruction error.

TABLE 3.2: Autoencoder Threshold Tuning Results

Threshold Value	**Anomalies Detected**
0.10	35
0.15	28
0.20	18
0.25	11
0.30	6

3.7 Model Performance Analysis

To rigorously evaluate the effectiveness of our anomaly detection framework, we conducted a comprehensive analysis of the individual and hybrid models integrated into

the software subsystem. The analysis focused on four distinct approaches: One-Class Support Vector Machine (OCSVM), Isolation Forest (IF), and two hybrid models, each combining either IF or OCSVM with an autoencoder neural network. These models were assessed not only on their detection accuracy but also on how they complement each other in identifying abnormal behavioral patterns in livestock acceleration data.

Each model was examined in terms of:

- Fundamental working principle
- Strengths and weaknesses
- Suitability for time-series anomaly detection
- The impact of combining models into a hybrid pipeline

Through this structured comparison, we demonstrate that the hybrid model using Isolation Forest and an autoencoder provided the most robust and generalizable performance, effectively isolating outliers while minimizing false positives. The following subsections present a detailed breakdown of how each model operates, supported by mathematical formulations and practical considerations.

3.7.1 One-Class Support Vector Machine (OCSVM)

OCSVM is an unsupervised learning algorithm that learns the boundary of normal data by mapping input data into a high-dimensional feature space using a kernel function. The algorithm attempts to find a hyperplane that best separates the origin from the data points, effectively encapsulating the normal class. Any data point lying outside this learned boundary is considered an anomaly.

Mathematically, OCSVM solves the following optimization problem (Equation (3.7)):

$$\min_{\omega,\rho,\xi} \frac{1}{2}\|\omega\|^2 + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i - \rho \tag{3.7}$$

subject to the constraints shown in Equation (3.8):

$$(\omega \cdot \phi(x_i)) \geq \rho - \xi_i, \quad \xi_i \geq 0 \tag{3.8}$$

where $\phi(x)$ is the kernel mapping function, ν is a hyperparameter controlling the trade-off between sensitivity to outliers and the fraction of support vectors, and ξ_i are slack variables allowing for soft margin violations.

Benefits:

- Effective in high-dimensional feature spaces
- Requires only normal data for training

Limitations:

- Sensitive to the choice of kernel and hyperparameters
- Struggles with large datasets due to computational complexity

3.7.2 Isolation Forest (IF)

Isolation Forest is a tree-based ensemble algorithm specifically designed for anomaly detection. Unlike other algorithms that attempt to profile normal data, IF isolates anomalies by recursively partitioning the dataset using random splits. Anomalies are data points that require fewer splits to isolate due to their rarity and dissimilarity from the rest of the data.

The anomaly score $s(x, n)$ is defined as shown in Equation (3.9):

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{3.9}$$

where $E(h(x))$ is the average path length to isolate point x, and $c(n)$ is the average path length of unsuccessful searches in Binary Search Trees, approximated in Equation (3.10):

$$c(n) = 2H(n-1) - \frac{2(n-1)}{n} \tag{3.10}$$

with $H(i)$ being the i-th harmonic number.

Benefits:

- Computationally efficient and scalable to large datasets
- Works well without requiring labeled data

Limitations:

- Performance can degrade if anomalies are not well-isolated
- Randomness in tree construction can lead to variability in results

3.7.3 Autoencoder with Filtered Data (Hybrid Models)

To enhance anomaly detection, we employed autoencoders in conjunction with the filtered outputs of IF and OCSVM. An autoencoder is a neural network trained to compress and reconstruct its input. It consists of an encoder $f(x)$ that maps the input x to a lower-dimensional latent representation z, and a decoder $g(z)$ that reconstructs the original input as $\hat{x}$.

The reconstruction error used as an anomaly score is defined in Equation (3.11):

$$L(x, \hat{x}) = \|x - \hat{x}\|^2 \tag{3.11}$$

Points with high reconstruction errors are considered anomalies, under the assumption that the autoencoder has only learned to reconstruct normal patterns.

Two separate autoencoders were trained using data filtered by IF and OCSVM respectively. This pre-filtering helped in reducing noise and ensuring that only relatively normal patterns were used during training, improving the quality of learned representations.

Benefits:

- Learns non-linear representations of data
- Reconstruction-based detection provides flexibility

Limitations:

- Requires careful tuning of network architecture and threshold
- Sensitive to training data quality

Benefits:

- Learns non-linear representations of data

- Reconstruction-based detection provides flexibility

Limitations:

- Requires careful tuning of network architecture and threshold
- Sensitive to training data quality

3.7.4 Why the Hybrid Model is Better?

While OCSVM and IF provide good baseline detection, they often suffer from limitations in generalization or sensitivity to parameter selection. Autoencoders, on the other hand, offer strong reconstruction-based detection but are highly dependent on clean training data. By combining these models, we exploit their complementary strengths.

The Isolation Forest was particularly effective in removing noise and outliers, providing a cleaner dataset for the autoencoder. This resulted in improved generalization and sharper reconstruction boundaries, leading to more accurate anomaly detection. The hybrid model with IF followed by an autoencoder achieved the best performance in terms of separation between normal and abnormal data points, as reflected by both visual inspection and quantitative metrics.

Chapter 4

Results and Implementation work

4.1 Results and Analysis

This section presents the results obtained from the anomaly detection models applied to the livestock acceleration data. The performance of Isolation Forest, One-Class SVM, and the hybrid Autoencoder models was compared both quantitatively and visually.

4.1.1 Hardware Performance

The hardware subsystem was responsible for collecting and transmitting acceleration and GPS data from the livestock in real-time. The setup included a tri-axial accelerometer, a GPS module, a microcontroller unit, and a 3.3V, 500 mAh battery. The collected data was stored and visualized using both a custom GUI and a cloud-based platform, ThingSpeak.

Data Collection and Transmission

Acceleration data was collected using a wearable sensor that captured motion along the X, Y, and Z axes. The GPS module was used to track the location of the animal with a sampling interval of 5 seconds. The entire data packet, including acceleration and GPS readings, was transmitted to the cloud every 15 seconds using ThingSpeak. ThingSpeak also provided basic visualization tools for monitoring the sensor readings in real time, supplementing the more detailed visualizations from the local GUI.

Battery Performance

The device was powered using a 3.3V, 500 mAh battery. During testing, this setup was able to support continuous data collection and transmission for several hours. The lightweight battery kept the overall hardware compact and suitable for use on livestock without causing discomfort or disruption to natural behavior.

System Stability and Storage

The system reliably collected and uploaded data at regular intervals without significant delays or packet loss. Using ThingSpeak allowed for simple and effective cloud storage, enabling access to data logs remotely and providing an easy interface for reviewing trends and sensor activity over time.

4.1.2 Anomalies Detected by Each Model

Tables 4.1 and 4.2 present the number of anomalies detected by the classical machine learning models and the autoencoder models trained on datasets cleaned by each of them. The variation in the number of anomalies demonstrates the difference in sensitivity and behavior of each model.

TABLE 4.1: Total anomalies detected by IF and OCSVM and common anomalies

Model	**Anomalies Detected**
Isolation Forest	304
One-Class SVM	153
Common anomalies detected by both	136

TABLE 4.2: Number of anomalies detected by each hybrid model at a threshold of mean + 3*std

Hybrid Model	**Anomalies Detected**
IF + Autoencoder	244
OCSVM + Autoencoder	143

4.1.3 Visualization of Time Series Anomalies

A time series plot in Figures 4.1 and 4.2 shows the acceleration magnitude over time with anomalies highlighted based on the predictions from One-Class SVM and Isolation Forest, respectively. This comparison reveals that both models often detect different anomaly points, indicating complementary decision patterns.

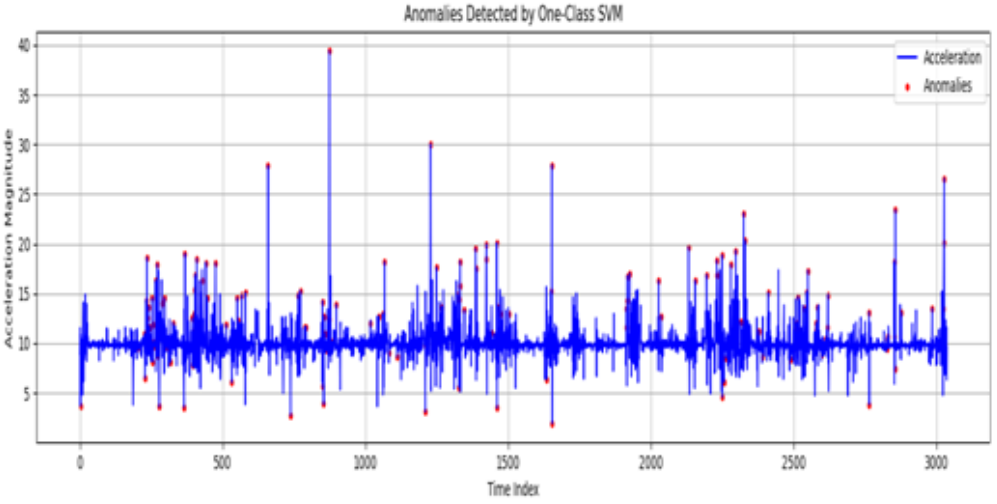

FIGURE 4.1: Anomalies Detected by One-Class SVM

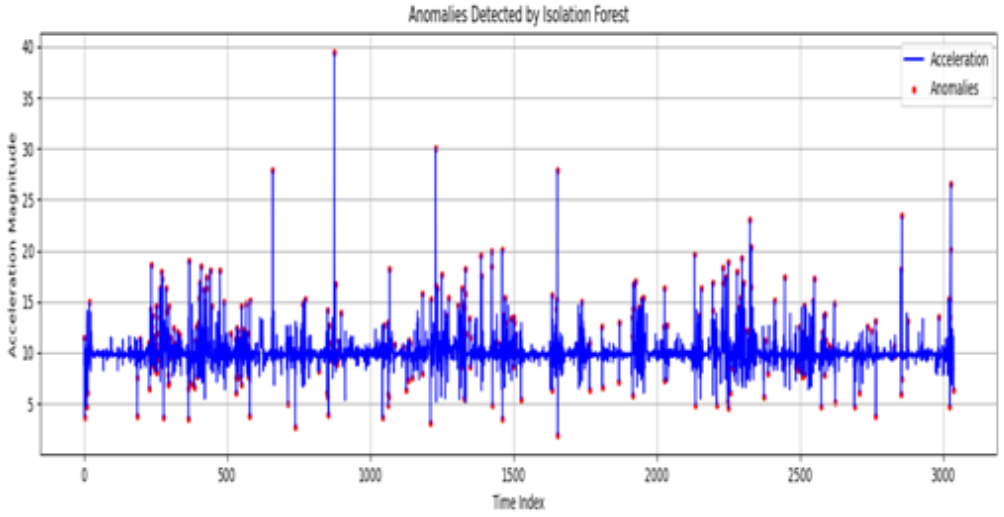

FIGURE 4.2: Anomalies Detected by Isolation Forest

4.1.4 Validation Loss for Autoencoder Models

Validation loss curves for both hybrid models (Autoencoder trained on IF data and Autoencoder trained on OCSVM data) are shown in Figures 4.3 and 4.4, respectively. The training loss converges smoothly in both cases, but the model trained on the IF-cleaned dataset shows a slightly more stable trend.

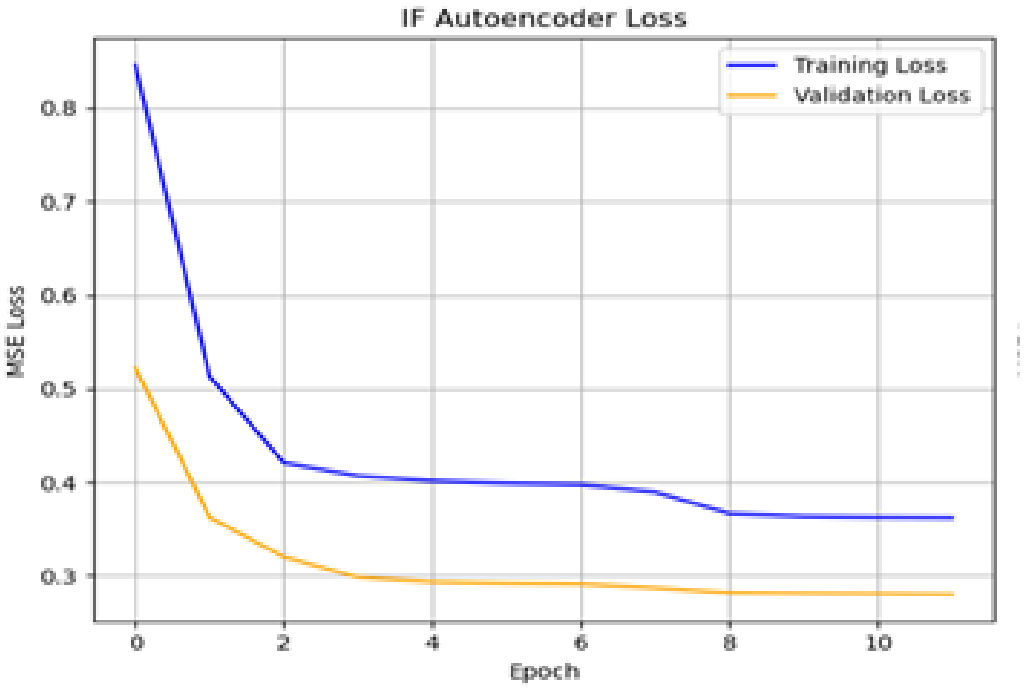

FIGURE 4.3: Autoencoder trained on IF Data

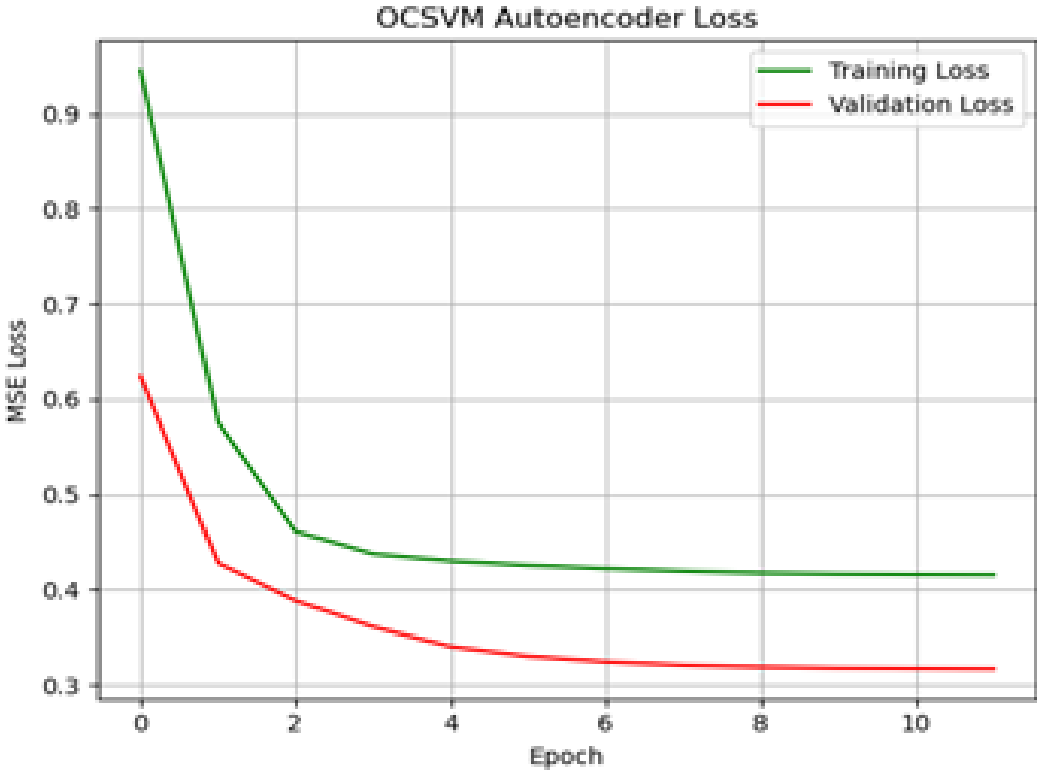

FIGURE 4.4: Autoencoder trained on OCSVM data

4.1.5 Effect of Threshold on Anomaly Detection

The number of anomalies detected by the autoencoder is highly dependent on the reconstruction error threshold. Figure 4.5 presents the results of threshold variation experiments, highlighting how a lower threshold increases anomaly sensitivity, while a higher threshold reduces false positives.

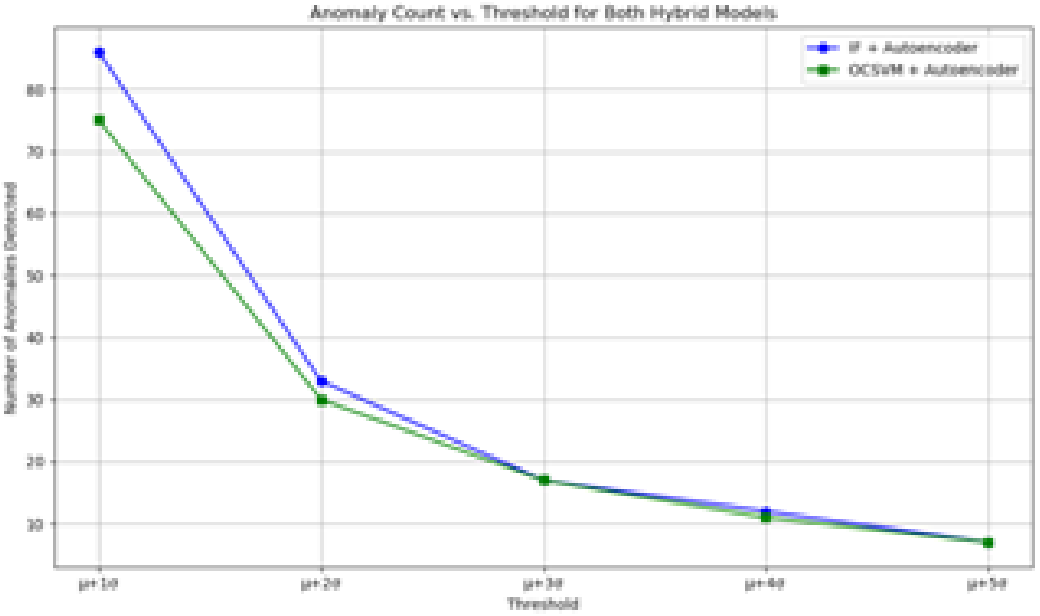

FIGURE 4.5: Anomaly Count v/s Threshold for both Hybrid Models

4.1.6 Observations and Analysis

- Isolation Forest tends to identify sparse and extreme anomaly points effectively.
- One-Class SVM detects boundary anomalies that may not be as isolated but still deviate from normal behavior.
- The autoencoder model, when trained on data pre-filtered by Isolation Forest, showed lower reconstruction errors and better anomaly separation.
- Proper threshold tuning is essential for balancing sensitivity and specificity in the autoencoder-based detection system.

4.1.7 Limitations

- **Sensor Dependency:** The system solely relies on accelerometer data, which may not fully capture complex behaviors or health indicators.

- **Lack of Ground Truth:** The absence of labeled data limits model evaluation and validation, making it difficult to quantify accuracy and reduce false positives.
- **Threshold Sensitivity:** Anomaly detection with autoencoders is highly sensitive to reconstruction error thresholds, requiring careful tuning to avoid misclassification.

Chapter 5

Conclusions

5.1 Conclusion

In this study, we presented a hybrid machine learning approach to detect anomalies in livestock behavior using triaxial acceleration data. By applying Isolation Forest and One-Class SVM as pre-filters, followed by an autoencoder for reconstruction-based anomaly detection, we demonstrated the potential of combining classical and deep learning methods to improve anomaly detection accuracy in livestock monitoring.

Our results showed that each model identified different sets of anomalies, highlighting their complementary strengths. The autoencoder's performance was sensitive to threshold tuning and benefited from pre-filtered data. Visualizations and quantitative comparisons helped validate the effectiveness of the approach.

This work lays a practical foundation for deploying AI-driven animal health monitoring systems in agricultural settings. Future efforts will focus on integrating real-time analysis, expanding sensor modalities, and addressing challenges such as generalization and data imbalance to enhance system reliability and scalability.

Chapter 6

Future scope

6.1 Future Scope

Building on the findings of this study, several avenues for future research and system enhancement are identified:

- **Real-Time Analysis:** Integrate real-time anomaly detection and alert systems to enable immediate intervention and improved animal welfare.
- **Expanded Sensor Modalities:** Incorporate additional sensors such as heart rate monitors, temperature sensors, and GPS modules to capture a broader range of animal health and behavior indicators.
- **Generalization Across Environments:** Develop and validate models that can generalize across different livestock species, farm environments, and management practices.
- **Addressing Data Imbalance:** Implement advanced techniques such as synthetic data generation or transfer learning to address the challenge of imbalanced datasets, particularly the scarcity of labeled anomalies.
- **Edge Computing and Scalability:** Explore the deployment of models on edge devices for scalable, low-latency monitoring in remote or resource-limited farm settings.
- **User-Friendly Interfaces:** Design intuitive dashboards and mobile applications for farmers, enabling easy interpretation of alerts and system status.
- **Integration with Farm Management Systems:** Link anomaly detection outputs to automated farm management systems for seamless decision-making and record-keeping.

These directions will help enhance the robustness, usability, and impact of AI-driven livestock health monitoring systems, paving the way for smarter and more sustainable agriculture.

Bibliography

[1] T. Wark, K. M. Carney, and R. J. G. Collins, "The virtual fencing project," CSIRO ICT Centre, Tech. Rep., 2007.

[2] A. Umstatter, "The evolution of virtual fences: A review," *Computers and Electronics in Agriculture*, vol. 182, 2021.

[3] A. Chen and Y. Zhao, "Low-cost virtual fencing for livestock management," in *2023 IEEE Global Humanitarian Technology Conference (GHTC)*, 2023, pp. 1–6.

[4] J. Porto, P. Pereira, L. Alves, A. Pereira, A. Rodrigues, and D. Silva, "Wearable sensor networks for cattle health monitoring," *Sensors*, vol. 20, no. 3, pp. 1–16, 2020.

[5] M. Robert, M. Cuypers, D. Guenther, and J. A. E. H. Verheijen, "Classification of posture and walking behavior by accelerometry and machine learning in cattle," *Computers and Electronics in Agriculture*, vol. 179, 2020.

[6] S. Wolfert *et al.*, "Big data in smart farming—a review," *Agricultural Systems*, vol. 153, pp. 69–80, 2017.

[7] A. Kamilaris and A. Prenafeta-Boldú, "Agri-iot: A review of internet of things for agriculture," *Future Generation Computer Systems*, vol. 79, pp. 52–63, 2018.

[8] F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation forest," in *Proc. 8th IEEE Int. Conf. on Data Mining*, 2008, pp. 413–422.

[9] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data," *PLOS ONE*, vol. 11, no. 4, p. e0152173, 2016.

[10] D. M. Hawkins, *Identification of Outliers*, 1st ed. New York, NY, USA: Springer, 1980.

[11] D. Pimentel, C. F. S. de Almeida, A. S. O. Santos, and M. A. P. M. T. S., "Autoencoder-based anomaly detection in time series," in *Int. Joint Conf. on Neural Networks*, 2018, pp. 1–8.

[12] S. Malhotra, R. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *ESANN*, 2015, pp. 1–6.

[13] R. Sundmaeker, P. Guillemin, P. D. Dupeux, and H. Verdouw, "Smart farming and the internet of food and farm 2020," European Commission, Tech. Rep., 2016.

[14] I. U. Gadzama, N. A. Edwards, A. J. Fisher, and A. J. Colditz, "Virtual fencing for livestock management: Effects on cattle behavior, welfare, and productivity," 2025.

[15] M. Lee, Y. Kwon, and H. Choi, "Virtual fencing technology for cattle management in the pasture," *Agriculture*, vol. 13, no. 1, p. 91, 2023.

[16] D. Singh, N. P. Yadav, V. Kumar, and R. K. Singh, "Sensor and computer vision based cattle health monitoring and management," *SSRG International Journal of Electrical and Electronics Engineering*, vol. 12, no. 1, pp. 1–6, 2025.

[17] P. Gangwar, S. Tiwari, A. Saini, and M. Agarwal, "Design of an intelligent wearable device for real-time cattle health monitoring," *Frontiers in Robotics and AI*, vol. 11, p. 1441960, 2024.

[18] R. Sharma, S. Gupta, P. Sharma, and N. Singh, "Development of a cloud-based iot system for livestock health monitoring," *Journal of Livestock Science*, vol. 15, no. 5, pp. 644–652, 2024.

[19] M. Wagner, A. Reiners, D. Klein, T. Gruber, and K. Meyerholz, "Detection of anomalies in cow activity using wavelet transform," *arXiv preprint arXiv:2502.21051*, 2025.

[20] M. Esmaeili, A. Farahbakhsh, and M. Ghaznavi-Ghoushchi, "Unsupervised anomaly detection in wildfire data using deep autoencoders," *Expert Systems with Applications*, vol. 201, p. 117112, 2022.

[21] L. Zeng, L. Li, M. Chen, and Z. Chen, "Detection of calf abnormal respiratory behavior based on frame difference and improved yolov5 algorithm," *Animals*, vol. 14, no. 3, p. 396, 2024.

[22] N. Bar, A. Kliger, and O. Levy, "Reconstruction-based weakly-supervised animal detection in aerial marine imagery," *Remote Sensing of Environment*, vol. 302, p. 113075, 2024.

Appendix A

Appendix: Software Code

A.1 Isolation Forest Anomaly Detection

Listing A.1: Isolation Forest model for anomaly detection

```
# Imports and Data Loading
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import IsolationForest
import matplotlib.pyplot as plt

data = pd.read_excel('Measurement cow 1.xls')
data['Time (s)'] = pd.to_datetime(data['Time (s)'], unit='s')
data['Acceleration Magnitude'] = np.sqrt(data['X (m/s^2)']**2 + data['Y (m/s^2)
    ']**2 + data['Z (m/s^2)']**2)

# Preprocessing
features = ['X (m/s^2)', 'Y (m/s^2)', 'Z (m/s^2)', 'Acceleration Magnitude']
X_scaled = StandardScaler().fit_transform(data[features])

# Model
model = IsolationForest(n_estimators=200, contamination=0.1, random_state=42)
data['Anomaly'] = model.fit_predict(X_scaled)
data['Is Anomaly'] = data['Anomaly'].apply(lambda x: 'Yes' if x == -1 else 'No'
    )
```

A.2 One-Class SVM Anomaly Detection

LISTING A.2: One-Class SVM model for anomaly detection

```
from sklearn.svm import OneClassSVM
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

data = pd.read_excel('Measurement cow 1.xls')
data['Time (s)'] = pd.to_datetime(data['Time (s)'])
data['Acceleration Magnitude'] = np.sqrt(data['X (m/s^2)']**2 + data['Y (m/s^2)
    ']**2 + data['Z (m/s^2)']**2)

features = ['X (m/s^2)', 'Y (m/s^2)', 'Z (m/s^2)', 'Acceleration Magnitude']
X_scaled = StandardScaler().fit_transform(data[features])

model = OneClassSVM(nu=0.1, kernel='rbf', gamma='scale')
data['Anomaly'] = model.fit_predict(X_scaled)
data['Is Anomaly'] = data['Anomaly'].apply(lambda x: 'Yes' if x == -1 else 'No'
    )
```

A.3 Autoencoder with Isolation Forest and OCSVM

LISTING A.3: Hybrid Autoencoder with Isolation Forest and OCSVM

```
from sklearn.ensemble import IsolationForest
from sklearn.svm import OneClassSVM
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler

data = pd.read_excel("Cowwwww2 2025-05-12 21-25-35.xls")
data['Acceleration Magnitude'] = np.sqrt(data['X (m/s^2)']**2 + data['Y (m/s^2)
    ']**2 + data['Z (m/s^2)']**2)
features = ['X (m/s^2)', 'Y (m/s^2)', 'Z (m/s^2)', 'Acceleration Magnitude']
X_scaled = StandardScaler().fit_transform(data[features])

# Train models
iso_forest = IsolationForest(n_estimators=200, contamination=0.1, random_state
    =42)
```

```
data['Anomaly_IF'] = iso_forest.fit_predict(X_scaled)

ocsvm = OneClassSVM(nu=0.05, kernel='rbf', gamma='scale')
data['Anomaly_OCSVM'] = ocsvm.fit_predict(X_scaled)

# Define autoencoder
def build_autoencoder(input_dim):
    model = Sequential([
        Dense(64, activation='relu', input_shape=(input_dim,)),
        Dense(32, activation='relu'),
        Dense(64, activation='relu'),
        Dense(input_dim, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='mse')
    return model

# Train on clean data
normal_if = X_scaled[data['Anomaly_IF'] == 1]
normal_ocsvm = X_scaled[data['Anomaly_OCSVM'] == 1]

autoencoder_if = build_autoencoder(X_scaled.shape[1])
autoencoder_if.fit(normal_if, normal_if, epochs=12, batch_size=64,
    validation_split=0.1)

autoencoder_ocsvm = build_autoencoder(X_scaled.shape[1])
autoencoder_ocsvm.fit(normal_ocsvm, normal_ocsvm, epochs=12, batch_size=64,
    validation_split=0.1)
```

Appendix B

Appendix: Hardware Code

B.1 WebSocket Server Code (Python)

Listing B.1: WebSocket server code for sensor data streaming

```
import asyncio
import websockets
import json

connected_clients = set()

async def handler(websocket, path):
    connected_clients.add(websocket)
    try:
        async for message in websocket:
            # Echo or handle received messages if needed
            pass
    finally:
        connected_clients.remove(websocket)

async def send_data():
    while True:
        # Dummy example sensor data, replace with actual sensor reading logic
        data = {
            "x": 0.12,
            "y": -0.34,
            "z": 0.56,
            "lat": 51.5074,
            "lng": -0.1278
        }
```

```
        if connected_clients:
            message = json.dumps(data)
            await asyncio.wait([client.send(message) for client in
                connected_clients])
        await asyncio.sleep(1) # send data every second

async def main():
    async with websockets.serve(handler, "0.0.0.0", 8765):
        await send_data()

asyncio.run(main())
```

B.2 Python Script for Sensor Data Collection

LISTING B.2: Python script to collect MPU6050 and GPS sensor data and send via WebSocket

```
import smbus2
import time
import math
import socket
import json
import serial

# MPU6050 Registers and constants
MPU6050_ADDR = 0x68
PWR_MGMT_1 = 0x6B
ACCEL_XOUT_H = 0x3B

bus = smbus2.SMBus(1)

# Initialize MPU6050
bus.write_byte_data(MPU6050_ADDR, PWR_MGMT_1, 0)

def read_raw_data(addr):
    high = bus.read_byte_data(MPU6050_ADDR, addr)
    low = bus.read_byte_data(MPU6050_ADDR, addr+1)
    value = ((high << 8) | low)
    if value > 32768:
        value = value - 65536
    return value

def get_accel_data():
```

```
    ax = read_raw_data(ACCEL_XOUT_H) / 16384.0
    ay = read_raw_data(ACCEL_XOUT_H + 2) / 16384.0
    az = read_raw_data(ACCEL_XOUT_H + 4) / 16384.0
    return ax, ay, az

# GPS serial initialization (adjust port as needed)
gps_serial = serial.Serial('/dev/ttyUSB0', baudrate=9600, timeout=1)

def read_gps():
    line = gps_serial.readline().decode('ascii', errors='replace')
    if line.startswith('$GPGGA'):
        parts = line.split(',')
        if parts[2] and parts[4]:
            lat = float(parts[2])
            lon = float(parts[4])
            # Convert NMEA to decimal degrees
            lat_deg = int(lat / 100)
            lat_min = lat - (lat_deg * 100)
            latitude = lat_deg + lat_min / 60
            lon_deg = int(lon / 100)
            lon_min = lon - (lon_deg * 100)
            longitude = lon_deg + lon_min / 60
            return latitude, longitude
    return None, None

# WebSocket client to send data to server
import asyncio
import websockets

async def send_sensor_data():
    uri = "ws://localhost:8765"
    async with websockets.connect(uri) as websocket:
        while True:
            ax, ay, az = get_accel_data()
            lat, lng = read_gps()
            data = {
                "x": ax,
                "y": ay,
                "z": az,
                "lat": lat,
                "lng": lng
            }
            await websocket.send(json.dumps(data))
            await asyncio.sleep(1)
```

```
if __name__ == "__main__":
    asyncio.run(send_sensor_data())
```

B.3 Web Interface Files

B.3.1 index.html

LISTING B.3: HTML web interface for location tracking

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <title>Cow Location Tracker</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/styles.css
        ') }}" />
    <link rel="stylesheet" href="https://unpkg.com/leaflet@1.9.4/dist/leaflet.
        css" />
    <script src="https://unpkg.com/leaflet@1.9.4/dist/leaflet.js"></script>
    <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
    <script src="{{ url_for('static', filename='js/map.js') }}"></script>
</head>
<body>
    <div class="main-layout">
        <div id="map"></div>
        <div class="side-panel">
            <h1>Cow's Current Location</h1>
            <p id="coordinates">Loading...</p>
            <form id="move-form" style="margin-top:10px;">
                <label>Latitude: <input type="number" step="any" id="input-lat"
                    required></label><br />
                <label>Longitude: <input type="number" step="any" id="input-lng"
                    required></label><br />
                <label>Radius (meters): <input type="number" step="any" id="input
                    -radius" required></label><br />
                <button type="submit">Create Fence</button>
            </form>
        </div>
    </div>
</body>
</html>
```

B.3.2 map.js

Listing B.4: JavaScript for map display and WebSocket communication

```
// Initialize map
const map = L.map('map').setView([0, 0], 13);

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    maxZoom: 19
}).addTo(map);

let marker;
const ws = new WebSocket('ws://' + window.location.host + '/ws');

ws.onmessage = function(event) {
    const data = JSON.parse(event.data);
    const lat = data.lat;
    const lng = data.lng;

    if (!marker) {
        marker = L.marker([lat, lng]).addTo(map);
        map.setView([lat, lng], 15);
    } else {
        marker.setLatLng([lat, lng]);
    }

    document.getElementById('coordinates').textContent = `Latitude: ${lat.
        toFixed(6)}, Longitude: ${lng.toFixed(6)}`;
};

document.getElementById('move-form').addEventListener('submit', function(event)
     {
    event.preventDefault();
    const lat = parseFloat(document.getElementById('input-lat').value);
    const lng = parseFloat(document.getElementById('input-lng').value);
    const radius = parseFloat(document.getElementById('input-radius').value);
    // Code to create fence with lat,lng,radius
    alert(`Fence set at (${lat}, ${lng}) with radius ${radius} meters.`);
});
```

B.3.3 styles.css

LISTING B.5: CSS styling for the web interface

```
body, html {
    margin: 0;
    padding: 0;
    height: 100%;
    font-family: Arial, sans-serif;
}

.main-layout {
    display: flex;
    height: 100vh;
}

#map {
    flex: 2;
}

.side-panel {
    flex: 1;
    padding: 20px;
    background-color: #f7f7f7;
    overflow-y: auto;
}
```

B.3.4 routes.py

LISTING B.6: Flask routes for the web interface

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

if __name__ == '__main__':
    app.run(debug=True)
```

B.3.5 main.py (Pico W Firmware)

LISTING B.7: Main firmware code for Raspberry Pi Pico W

```
import network
import time
import uasyncio as asyncio
from machine import Pin, I2C
from imu import MPU6050 # assuming an imu.py driver exists

SSID = 'your-ssid'
PASSWORD = 'your-password'

i2c = I2C(0, scl=Pin(17), sda=Pin(16))
imu = MPU6050(i2c)

wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect(SSID, PASSWORD)

while not wlan.isconnected():
    time.sleep(1)

print('Connected to WiFi', wlan.ifconfig())

async def send_sensor_data():
    while True:
        ax, ay, az = imu.acceleration
        # Send data via WebSocket or other means here
        print(f"Accel: {ax}, {ay}, {az}")
        await asyncio.sleep(1)

asyncio.run(send_sensor_data())
```