Monitoring the Health and Reliability of Remote Autonomous Systems

Submitted as Honours Dissertation in SIT723

SUBMISSION DATE

T1-2026

Daniel Mattioli

STUDENT ID: s223377562

Bachelor of Software Engineering - Honours (S464)

Supervised by: Kevin Lee

Abstract

Remote autonomous systems (RAS), such as unmanned aerial vehicles and ground robots, are becoming more common in environments where failures are costly and human intervention is not always possible (Puchalski & Giernacki 2022). However, existing system health monitoring solutions are mainly reactive and tailored for static systems that rely on cloud infrastructure instead of mobile platforms that have limited resources and unreliable connectivity (Gültekin et al. 2022). There is currently a lack of practical and lightweight frameworks for real-time RAS monitoring that do not rely on cloud infrastructure or pre-labelled fault data.

This thesis presents the design, implementation and evaluation of a Prometheus-based monitoring framework for a remote autonomous system. Research Question 1 (RQ1) validates the pipeline with replayed drone telemetry, and Research Question 2 (RQ2) will extend it to a physical DJI Tello drone with threshold-based fault detection.

For RQ1, a monitoring pipeline was built which includes a custom Python telemetry exporter, Prometheus, and a Grafana dashboard. The pipeline was evaluated the pipeline on the Drone Telemetry Tampering Dataset v2 (Ekanayakadevlk 2024), a publicly available dataset of DJI flight records containing six labelled anomaly phases. The results showed that the pipeline can correctly detect all the fault phase transitions and has zero false positives during normal operation. Detection latency was rated partial due to Prometheus pull-based scrape architecture (Barrett et al. 2023). Three out of four evaluation criteria were fully met.

These findings confirm that a Prometheus based pipeline is a suitable method for monitoring RAS health. RQ2 will build on this by using a DJI Tello drone to collect live flight data and replacing the pre-labelled health values with Prometheus alert rules that can detect faults automatically.

Contents

List of Figures

List of Tables

1 Introduction

Remote autonomous systems (RAS) are increasingly being deployed across a wide range of environments where direct human oversight is limited or unavailable (Hemalatha et al. 2025). Drones are used for infrastructure inspection, search and rescue operations, and environmental monitoring (Puchalski & Giernacki 2022). Underwater vehicles conduct seabed surveys and pipeline inspections. Ground-based robots operate in disaster zones, warehouses, and remote agricultural settings (Singh et al. 2025). What these systems share is that they operate far from the people responsible for maintaining them, and they are expected to function reliably without continuous human intervention.

The consequence of failure in these environments is significant. When a remote autonomous system breaks down in the field, the cost is not just the hardware. It is the lost mission, the delayed response, the inability to reach the system to diagnose the problem (Gültekin et al. 2022). In safety-critical applications such as emergency response or infrastructure monitoring, a failure that goes undetected for even a short period can have serious consequences. The further a system operates from human oversight, the higher the stakes when something goes wrong.

Despite this, most current approaches to monitoring remote autonomous systems are reactive rather than proactive (Smallman & Cook 2013). Faults are typically detected after they cause a failure, rather than identified early through continuous health observation (Yousuf et al. 2024). Existing monitoring research has largely been developed for static industrial systems or controlled laboratory environments, and there is limited evidence of how these approaches perform on physically mobile platforms operating in unpredictable conditions (Boncea & Bacivarov 2016). The result is a gap between what the literature describes and what remote autonomous systems actually need: a continuous, real-time monitoring pipeline that can track system health, detect anomalous behaviour early, and provide actionable insight without requiring a human to be watching at all times.

This thesis investigates how monitoring frameworks can be designed and evaluated for this purpose, using a DJI Tello drone as a representative remote autonomous system and Prometheus as the primary monitoring tool.

1.1 Research Questions

Two research questions guide this study:

- **RQ1:** How can monitoring frameworks be used to assess and maintain the health of remote autonomous systems?
- **RQ2:** How can system metrics such as battery level, connectivity, and performance be used to detect faults in remote autonomous systems?

RQ1 is addressed in Trimester 1 through the design and experimental evaluation of a Prometheus-based monitoring pipeline using drone telemetry data publicly available online. RQ2 is addressed in Trimester 2 through physical integration with a DJI Tello drone, controlled fault injection experiments, and quantitative evaluation of detection performance.

1.2 Contributions

This thesis makes the following contributions:

- A Prometheus-based monitoring framework for remote autonomous systems, including a custom Python telemetry exporter that exposes eight health metrics (altitude, speed, heading, GPS coordinates, anomaly label, health status, and fault event count) collected by Prometheus every 15 seconds.
- Experimental testing of the monitoring system with drone telemetry data from the Drone Telemetry Tampering Dataset v2 (Ekanayakadevlk 2024) that has six labeled anomaly phases. The proposed system was able to detect all fault phase transitions during normal operation without any false positives, achieving three out of four evaluation criteria in full.
- A reusable monitoring system architecture (DJI Tello SDK $\rightarrow$ Python exporter $\rightarrow$ Prometheus $\rightarrow$ Grafana) not limited to a single platform and can be modified for other autonomous systems that expose its telemetry data.

- A proven platform for the physical drone integration in the second trimester, where the RQ2 will be tested in a somewhat controlled environment through the flight sessions of a DJI Tello drone with normal operation, simulated faults and the measurement of the fault detection performance.

1.3 Thesis Structure

Section 2 reviews the relevant literature across five themes: system health and reliability, monitoring in remote autonomous systems, anomaly detection using machine learning, self-healing, and monitoring frameworks and observability. Section 3 presents the research design and selected methods. Section 4 explores RQ1 through the design, implementation, and experimental evaluation of the Prometheus-based monitoring framework. Section 5 outlines the plan for RQ2 in Trimester 2. Section 6 concludes the thesis and identifies directions for future work.

2 Literature Review

2.1 Introduction

Remote autonomous systems are becoming increasingly advanced (Zhu et al. 2024), but this also raises an important challenge: how can system failures be identified before they occur? These systems operate in environments where a human cannot always step in, which means the system itself needs to be able to assess its own health and catch problems early. Monitoring sensor data continuously is how that gets done. On top of that, machine learning has opened up new ways to detect faults that don't follow a predictable pattern, and self-healing systems take it a step further by responding to problems automatically rather than waiting for someone to notice. All of these capabilities depend on one thing: a monitoring framework that gives real-time visibility into what the system is actually doing.

This review covers five themes that are at the centre of this problem: system health and

reliability, monitoring of autonomous systems, machine learning-based anomaly detection, self-healing, and monitoring frameworks and observability. Each theme is explored, with the key findings and limitations of current research outlined before identifying the gap this work aims to address.

2.2 System Health and Reliability in Remote Autonomous Systems

System health monitoring is the continuous tracking of operational parameters such as battery voltage, motor temperature, GPS signal quality and IMU readings to determine whether a system is functioning within safe limits Reichard et al. (2007). In remote autonomous systems, these parameters can drift gradually before any visible problem emerges, which makes early detection critical. Unlike industrial equipment with a technician nearby, a drone operating remotely has no immediate fallback when something starts to degrade. The difference between catching a fault early and missing it entirely often comes down to how well the monitoring system is set up.

Yousuf et al. (2024) showed that sensor-based monitoring can enable predictive maintenance through early detection of faults in industrial motor systems. They combine temperature, vibration and current sensors to constantly monitor the health of an AC induction motor, sending GSM alerts if the readings are outside the expected range. The significance of this approach is that the system is proactive, meaning it does not wait for a failure to occur, but detects early warning signs and raises an alert before the fault becomes significant. One limitation is that their detection logic is based on fixed thresholds, where a fault is essentially anything outside a pre-defined range. This works well in a stable industrial environment, but in a dynamic platform like a drone, where what is considered "normal" varies with flight mode, payload, and weather, fixed thresholds quickly become unreliable. The principle is still the same, but the way it is analysed has to be more flexible.

Singh et al. (2025) discuss reliability in IoT systems from a broader perspective, reviewing techniques such as fault tolerance, predictive maintenance, energy-efficient design and secure communication protocols. In their study they observe that reliability in autonomous systems is not about finding individual faults in isolation but about designing systems that can still perform well when components begin to degrade. This is especially important for remote autonomous systems. A drone or robot that is operating far from a maintenance team needs to be able to identify its own health state and report it clearly. They also mention that energy efficiency is

closely related to reliability: a sensor node that runs out of battery cannot report anything, and a monitoring layer that consumes too much processing power can itself become a reliability risk. That trade-off is particularly relevant in the context of drones, where every bit of energy spent on monitoring is energy not spent on flight time.

These ideas are further developed by Gültekin et al. (2022) who look at the real-time health monitoring and fault detection of industrial autonomous transfer vehicles using edge AI. The system monitors vehicle health metrics locally and uses machine learning to detect faults without a centralised server. The edge-based approach is important as it shows that real-time health monitoring is possible even in physically mobile systems with limited or intermittent connectivity, a problem directly relevant to drone-based monitoring. Their reported results are also worth mentioning: they reduced the network bandwidth requirement by around 43 times and the total data transfer time by around 37 times by running detection at the edge as opposed to streaming everything to a cloud service. Those reductions are significant for a mobile platform where bandwidth is often the limiting factor. The tradeoff is that edge AI moves some of the processing burden onto the device itself, adding a new reliability variable that also needs to be monitored.

When analysed together, these studies show that the most successful health monitoring for autonomous systems combines the continuous collection of sensor data with smart analysis and real-time alerts. Most of this work, however, is focused on relatively static or industrial systems, motors, factory equipment, and fixed sensor nodes, with limited application to dynamic, mobile platforms operating in unpredictable environments. Even the edge AI work in Gültekin et al., which is closest to a mobile use case, is still evaluated on transfer vehicles working in a controlled factory environment with stable network coverage. The jump from that to a drone operating outside, losing GPS, hitting wind gusts, changing flight modes, is big, and not directly covered by any of these studies.

Existing health monitoring research is focused primarily on industrial or stationary systems, with limited application to dynamic, mobile autonomous systems such as drones. In addition, not much is known about how health metrics can be continuously collected and acted on in real time in remote systems where connectivity can be unreliable and conditions can change rapidly. Very few studies look at how reliability monitoring should adapt when ‘normal’ system behaviour itself changes during operation, e.g. a drone transitioning between hovering, manoeuvring and landing.

2.3 Monitoring in Remote Autonomous Systems

In remote autonomous systems, monitoring refers to the process of collecting data continuously from sensors to assess system performance and detect faults before failures occur Barile et al. (2018). Collecting data is not enough as it has to be analysed quickly and trigger appropriate responses, in real time. As autonomous systems become more complex, and are deployed in wider ranges of environments, reliable monitoring has become a genuinely hard problem to solve. Research in this area ranges from healthcare wearables, to industrial IoT, to aerial robotics and each of these provides a slightly different perspective on what good monitoring really looks like.

Paradeshi et al. (2022) show how to use IoT sensor data to detect faults in healthcare monitoring systems functioning in wireless environments. Their framework analyzes performance data from wearable sensors, flagging irregular readings that may indicate device failure, with about 80% fault detection accuracy. Though the application is healthcare focused, the underlying approach of constantly collecting sensor data, identifying deviations from expected behaviour and raising alerts directly applies to monitoring autonomous systems in other domains. Their research gives a useful baseline to understand how monitoring frameworks can be designed and evaluated. The 80% accuracy rate is interesting, though. It might be okay for health care wearables, where a missed alert usually just means a checkup. For an autonomous drone, that same figure is one missed fault in every five, which is a big safety concern.

A more sophisticated approach is taken by Boncea & Bacivarov (2016), who suggest a system which constantly pushes metrics of IoT devices into time-series databases like Prometheus, InfluxDB and OpenTSDB. Their architecture separates the concerns of data collection, storage and analysis into different layers, allowing the monitoring system to scale up as new devices are added. Importantly, they see monitoring as a continuous process, not a one-off check. The system keeps a continuous log of system behaviour that can be queried to identify both sudden faults and gradual degradation in performance over time. This is fundamentally more powerful than threshold-based alerting alone. The architecture is also interoperable at the storage layer, so the same monitoring logic can work for many monitoring tools such as Prometheus, InfluxDB or OpenTSDB. This kind of modularity is useful for research, because it means you can swap out components without having to rebuild the whole pipeline.

More recently, UAV-oriented research has begun to explore how monitoring can be applied to mobile, airborne autonomous systems. Puchalski & Giernacki (2022) present a survey of

fault detection methods for unmanned aerial vehicles, discussing techniques from model-based methods to data-driven approaches. In their review, they highlight that UAV monitoring presents unique challenges compared to ground-based systems. Flight dynamics are non-linear, sensor readings can be influenced by vibration and wind, and the consequences of undetected faults can be severe. However, most of the existing monitoring approaches are not customized for UAV platforms. Rather they adapt techniques developed for other fields. Their review also groups detection methods into wider categories like model-based, signal-based, data-driven and hybrid, and notes that the more reliable systems tend to combine more than one. This observation is important for any new UAV monitoring system, because it suggests that no single technique alone is likely to be good enough.

A common limitation in these studies is that monitoring is usually considered as a fault detection problem or a health tracking problem. Few approaches integrate both into a unified system that provides real-time health insight while supporting fault detection and diagnosis. For a remote autonomous system, both views are very useful. Knowing that the system is currently within its safe operating range is just as important as knowing when something has gone wrong. Splitting the two creates blind spots that are difficult to close after the fact.

It is shown that current monitoring approaches are effective in detecting faults in controlled environments, but they do not provide a full picture of the system health in real-time for dynamic, mobile autonomous systems. Very little research is specifically designed for mobile platforms such as drones. Most of the existing work does not combine health tracking and fault detection in a single integrated monitoring pipeline. And there is little evidence of how these approaches perform in conditions that are routine for a drone such as intermittent connectivity, rapid changes in operating mode, and environmental disturbance, and none of which are typical in the labs that most of these studies are conducted

2.4 Anomaly Detection using Machine Learning

Remote autonomous systems generate a lot of sensor data, and not all of it is easy to interpret with fixed rules. Traditional fault detection requires you to define what a fault looks like in advance, which is fine when the failure modes are well understood but quickly falls apart in a system like a drone where the range of possible faults is hard to predict. Machine learning changes that. Instead of asking whether a reading is above a threshold, the system learns what normal behaviour looks like and flags anything that doesn't fit. The shift from rule-based to

learning-based detection allows anomalies to be identified based on the specific environment and behaviour of the system.

Devi et al. (2024) demonstrate how the Isolation Forest algorithm can be applied to IoT sensor data to detect anomalies effectively. Isolation Forest works by finding data points that are statistically isolated from the rest of the data set. It is best suited for finding rare or unusual events in time-series telemetry. Their system computes anomaly scores for sensor readings and takes automated recovery actions if scores cross a threshold. Having this type of approach where they detect an anomaly and automatically respond to it directly reduces the amount of downtime, showing how machine learning fault detection can be integrated within a system.

DeMedeiros et al. (2023) present an in-depth survey of AI-based anomaly detection methods for IoT and sensor networks using supervised, unsupervised, and semi-supervised approaches. The survey finds that unsupervised methods are most useful in remote autonomous systems where it is often impractical to collect large amounts of fault data. This is appropriate for a drone monitoring scenario where normal flight telemetry is easily available but specifically labelled faults are harder to find. The authors point out that the definition of an anomaly varies greatly between studies which makes it difficult to compare results directly. Some studies treat anomalies as discrete attack events, some as gradual sensor drift, some as any deviation from a statistical baseline. That definitional inconsistency is something to keep in mind when interpreting accuracy figures across other papers.

Furthermore, Rafique et al. (2024) explore machine learning and deep learning methods for IoT network anomaly detection, finding that Random Forest and deep neural networks consistently outperform simpler models, achieving detection accuracy above 99% under controlled test conditions. But they also mention that the real-world deployment usually reduces this accuracy significantly. The sensor data in a live system contains noise, missing values, and distribution shifts that are absent in training data. This is an important limitation for applying these techniques to real drone telemetry. They also note in their review that most published results are based on a small number of benchmark datasets, and that models that seem strong in the literature may not be able to reproduce the same accuracy with other data sets. A reasonable rule of thumb when planning a real deployment is to treat published figures as a threshold, and not a guarantee.

Collectively these studies show the potential of machine learning for anomaly detection, but also a common limitation. Most approaches are evaluated in controlled settings on historical datasets, rather than on live systems generating telemetry in real-time. Evaluating on static datasets is great because it makes it easy to compare results, but it hides a lot of the practical

complexity that real systems have to deal with constantly: sensor dropout, varying sample rates, mode changes during flight, etc.

While anomaly detection with machine learning has been demonstrated to work well in controlled conditions, there is little research on applying these approaches to live telemetry from physical autonomous systems. The majority of studies test algorithms on static datasets, rather than real-time data streams, and therefore have limited relevance to systems such as drones that produce continuous and dynamic telemetry in differing conditions. The integration of anomaly detection into larger real-time monitoring frameworks is also underexplored. More work needs to be done to not only utilise ML techniques such as the isolation forest on static data, but as part of a real autonomous system streaming real-time data.

2.5 Self-Healing in Autonomous Systems

A more advanced approach for ensuring the reliability of remote autonomous systems is the use of self-healing systems which enable automatic recovery from detected faults Ogunmolu et al. (2025). Self-healing systems detect a fault and then respond without sending an alert to an operator and waiting for human intervention. They reconfigure themselves to keep working. This capability is especially important in remote autonomous systems where human intervention may be delayed or impossible. The compromise is that an automatic recovery is only as good as the logic behind it. Sometimes a poorly chosen response can do more damage than the original fault. This is why self-healing research focuses not just on whether to act, but on how to ensure the action taken is the right one.

Devi et al. (2024) demonstrate the use of anomaly detection output to trigger automatic recovery in IoT sensor networks. Once there has been a detected fault through Isolation Forest, the system dynamically reconfigures the network by isolating the faulty component and redistributing its load to healthy nodes, thus allowing the network to continue running as normal. This closed-loop process of detection, response, and recovery significantly reduces downtime and highlights the importance of integrating fault detection with recovery in an automated system. The recovery actions in their system are fairly simple, basically isolating a node and redirecting traffic, however this would have to be modified for a system like a drone. With a failed motor or low battery there is nothing to reroute to . In that case, recovery is less about reconfiguring a network and more about behaviour change such as returning home, slowing down and prioritising landing safely.

Ben Hafaiedh & Ben Slimane (2022) provide a more formal approach to self-healing in autonomous systems, where the fault detection, diagnosis and recovery are defined as three different but connected phases. Their approach uses verification to ensure that the steps to recover the system are safe and do not cause new faults which is an important factor in systems where recovery mistakes can be worse than the original fault. Their model is highly relevant to physical autonomous systems such as drones, where actions of recovery have real-world consequences. The framework also makes it easier to reason about each stage in isolation by separating diagnosis from recovery. The system can check that it will always detect a certain class of fault and, separately, that it will always react with a suitable action. This is more difficult to achieve in conventional self-healing systems where detection and response are combined.

More recently, Alauthman & Al-Hyari (2025) propose a clever fault detection and selfhealing system for wireless sensor networks that uses machine learning in conjunction with the Flying Fox optimisation algorithm to automatically choose the most suitable recovery strategy based on the current system state. This versatile approach is a significant improvement over previous self-healing systems which typically adopted a fixed recovery strategy regardless of how severe the fault is. The ability to adapt the recovery to the systems real time state is especially important for remote autonomous systems operating in dynamic environments. Their reported results, such as 94.6% fault detection accuracy, around 120 milliseconds average recovery time, and 98.5% network resilience, demonstrate the benefit of combining proactive detection with adaptive recovery. These results are from a wireless sensor network setup rather than a mobile platform, so they would need to be re-validated in a drone context, however the fundamentals of this can be applied in the drone context. While faults can vary in severity, not all faults are the same, and different approaches are needed.

Despite these advances, most self-healing research focuses on networked or distributed software systems rather than physical autonomous systems operating in real-world environments. The literature also does not delve into the combination of fault detection, recovery and continuous health monitoring in a single integrated system. In most studies, recovery is seen as a separate event triggered by a fault, rather than as one part of a continuous monitoring loop.

While self-healing systems have shown great recovery capabilities, there is very little work done by other research professionals on integrating self-healing mechanisms with continuous real-time monitoring in physical autonomous systems. Most of the existing work focuses on the detection or the recovery phase in isolation, with little focus to how both can be embedded within a larger monitoring framework. There is also little work on how to monitor recovery itself, i.e., how a system checks that recovery has indeed succeeded rather than assuming it has.

2.6 Monitoring Frameworks and Observability

Before any of the techniques discussed in this review can work, there needs to be infrastructure collecting and storing the data in the first place. Tools like Prometheus collect system metrics continuously, store them in time-series databases, and make them queryable in real time. Observability is a related but slightly broader idea. Where monitoring tells you whether known things are happening, such as whether the battery is below 20%, observability lets you ask questions such as why a particular flight behaved differently from the last one. For a drone monitoring system, both of these tools are essential.

Boncea & Bacivarov (2016) were the first to officially propose a monitoring architecture specifically designed for IoT reliability. They used time-series databases and tools such as Prometheus, InfluxDB and Graphite to continuously collect and analyse the system's metrics. Their design separates metrics collection, storage, and querying into different layers, allowing for easier scaling and adaptation with the addition of new devices. Prometheus was used as the main collection tool throughout their research. Its pull-based scraping model handles intermittent connectivity by recording missed scrapes rather than silently losing data, where the monitoring server requests metrics from devices at regular intervals. This feature is useful for this research with drones, where connectivity can be lost and regained multiple times during a single flight. A push system would lose data during a dropout, while Prometheus just registers a gap and resumes when the link has been restored.

Prometheus' query language, PromQL, makes it easier to create alert rules and extract useful insights from raw telemetry data, instead of only collecting basic metrics. PromQL allows users to calculate rates of change, moving averages and anomaly thresholds inside the monitoring tool instead of just collecting values. This is very useful for autonomous systems, where trends and rates of change are often more useful than raw sensor readings alone. A battery at 60% is not interesting, but a battery dropping at 3x the expected rate definitely is. Having that kind of insight is exactly what PromQL is built for.

Since this paper was released, Prometheus has become one of the most widely used monitoring tools in both cloud and IoT environments. Barrett et al. (2023) review Prometheus and Grafana as an observability stack in cloud settings, stating that using both of them together offers thorough real-time monitoring with very little configuration overhead. The study emphasizes the fact that the dashboard capabilities of Grafana transform raw Prometheus metrics into visual system health views that are easy to interpret and understand. This is an important practical

consideration when monitoring a system such as a drone that produces many metrics at the same time. Monitoring of this data would need to be easy and quick in the case of an emergency, unless of course a ML model is handling it instead. The paper also discusses Prometheus and Grafana as being well suited to environments where lots of different data sources are constantly changing. While a drone system is quite different from a containerized cluster, the same idea of cleanly managing and monitoring multiple short-lived data streams is still useful when tracking several drones or multiple flight sessions over time.

More recently, Rafiq et al. (2025) demonstrate an AI and IoT-driven monitoring system integrating Prometheus and Grafana with machine learning models for predictive anomaly detection. Their system uses Prometheus' ability to monitor and then pairs it with advanced analysis to predict potential failures before they occur. They go beyond the normal threshold-based alerts by using machine learning models. This integration of monitoring framework with machine learning is the direction the modern system health management is heading and is especially useful for remote autonomous systems such as drones. According to their results, implementing the system across five client networks reduced downtime by around 95% and improved incident resolution times by roughly 90% compared to systems without predictive monitoring. These results come from a managed services environment rather than an autonomous drone system, so they should not be treated as direct performance benchmarks. However, the overall monitoring architecture and approach are still highly applicable to drone-based scenarios.

While these tools are widely used in cloud and industrial environments, their use in remote autonomous systems, and especially mobile platforms such as drones, is still very limited. Most of the existing implementations assume stable network connectivity and relatively predictable workloads, both of which cannot be guaranteed during drone operations. There is also relatively little discussion of how to handle telemetry from a platform that moves, changes state, and drops offline occasionally, making the simple task of scraping metrics much more difficult.

Monitoring frameworks such as Prometheus and Grafana are well established in cloud and distributed systems, however, there is very limited research on applying them to remote autonomous systems, especially physical, mobile platforms such as drones . There is a lack of practical implementations to demonstrate how these tools can maintain continuous health visibility in environments with variable connectivity and changing operating conditions, which is exactly the type of environment a drone operates in.

2.7 Research Gap

Existing research address's reliability, fault detection, anomaly detection, self-healing and monitoring frameworks as separate areas, but there is a shortage of literature that integrates all these in the context of real-time monitoring for physical remote autonomous systems. While each individual area has advanced, the integration of continuous health monitoring, fault detection with the use of ML and automated recovery into a remote autonomous system remains unexplored. In particular, no practical implementations exist using tools such as Prometheus on systems that operate in dynamic real-world environments such as drone flight. Most existing studies evaluate their approaches in controlled or static environments. This limits how well the findings apply to autonomous platforms that operate in changing conditions with temporary connectivity loss, and ones that make real-time health decisions.

The aim of this research is to investigate the use of monitoring frameworks, like Prometheus, to assess and ensure the health and reliability of remote autonomous systems. A DJI Tello drone will represent a remote autonomous system to collect real-time telemetry data that will be fed into Prometheus. The aim is to evaluate whether this approach is effective for fault detection and continuous health visibility in a physical, mobile autonomous system. This is the gap identified in all five themes in this review.

3 Methodology

3.1 Research Design Overview

This document presents the formal research methodology for the thesis *Monitoring the Health and Reliability of Remote Autonomous Systems*. It expands on the selected methods from Task 3.3C, outlines a research plan with timelines, addresses ethical considerations, and identifies milestones and sustainability measures.

Several methods are relevant to this research area. The table below compares their strengths,

weaknesses, and suitability:

Table 1: Comparison of Research Methods

Method	**Strengths**	**Weaknesses**	**Suitability**
Experimental Research	Generates real, measurable evidence, reproducible under controlled conditions	Results may not generalise beyond the test environment	High
Design Science Research (DSR)	Produces a reusable artefact, grounds contribution in engineering practice	Evaluation criteria can be subjective without upfront definition	High
Literature Review	Establishes theoretical grounding, identifies gaps and prior approaches	No original data produced, cannot validate the system	Moderate
Simulation / Modelling	Safe, repeatable, no hardware needed	Does not reflect real hardware behaviour or physical environmental factors	Low

Literature review is used as initial input, and is not a primary method. Simulation was considered but rejected because the research questions require real telemetry from a physical system. **Experimental Research** and **DSR** are selected as the primary and secondary methods.

3.2 Research Questions

Two research questions guide this study, each targeting a distinct aspect of health monitoring for remote autonomous systems (RAS):

- **RQ1:** How can monitoring frameworks be used to assess and maintain the health of remote autonomous systems?
- **RQ2:** How can system metrics such as battery level, connectivity, and performance be used to detect faults in remote autonomous systems?

These questions address a gap identified in the literature: existing monitoring approaches rarely integrate real-time telemetry pipelines with physical autonomous systems operating in the field (Boncea & Bacivarov 2016, Devi et al. 2024). The following approach aims to fill this gap by integrating framework design and real-world evaluation.

3.3 Approach

Two complementary methods have been selected: Experimental Research (primary) and Design Science Research - DSR (secondary).

3.3.1 Experimental Research (Primary)

Experimental research is based on controlled experiments on a real system, collecting real data and verifying results for a specific set of criteria. It was chosen in other related work by Yousuf et al. (2024) and Paradeshi et al. (2022), and both tested monitoring systems against live hardware to produce quantifiable results.

- **Application:** A DJI Tello drone will stream telemetry such as battery percentage, signal strength, and flight time via the Tello SDK into Prometheus through a custom Python exporter. Controlled flight sessions will cover normal operation, low-battery conditions, and signal degradation. Detection performance will be evaluated using latency, false positive rate, and metric coverage. Due to trimester constraints, toolkit setup and familiarisation will occur this trimester, with full drone integration planned for the following trimester. In Trimester 1, telemetry data was sourced from an existing online dataset and ingested into Prometheus via a custom Python exporter, in order to validate the monitoring pipeline architecture before the physical drone integration. Live drone data collection using the DJI Tello is planned for Trimester 2.
- **Validity:** Results will be archived in session logs for reproducibility. Baseline sessions establish a normal operating environment that provides a basis for anomaly detection thresholds based on actual data instead of assumptions.

3.3.2 Design Science Research (Secondary)

DSR is the design and assessment of artefact to solve a particular problem. Boncea & Bacivarov (2016) applied this approach to propose a Prometheus-based monitoring architecture which this thesis builds on.

- **Application:** The monitoring pipeline (Tello SDK → Python exporter → Prometheus → Grafana) is the designed artefact. It will be documented, justified against the literature, and evaluated through experimental sessions, following the iterative DSR cycle: design, build, evaluate, refine.
- **Validity:** Evaluation criteria (metric coverage, alert latency, dashboard completeness) are defined upfront to ensure objective assessment of the artefact's fitness for purpose.

3.4 Experimental Platform

Table 2: Required Resources

Resource	**Purpose**	**Access**
DJI Tello Drone	Representative RAS for telemetry collection	Provided by Deakin University
Laptop / PC	Run Prometheus, Grafana, Python scripts	Personally owned
Wi-Fi Network (2.4 GHz)	Tello SDK communication over UDP	Home / University Wi-Fi
Prometheus v2.x	Time-series metric collection and querying	Open-source
Grafana	Dashboard visualisation of telemetry streams. This is optional and acts as a good visualisation layer of the data.	Open-source
Python 3.x + djitellopy	Telemetry extraction and Prometheus exporter	Open-source
Drone Telemetry Tampering Dataset v2 (Kaggle)	Drone telemetry data used for Trimester 1 pipeline validation	Publicly available via Kaggle

3.5 Data Collection

The experiment runs in three phases:

- **Phase 1 - Baseline (RQ1, RQ2):** Standard flights under normal conditions at a 1-second Prometheus scrape interval, establishing normal operating ranges for battery draw, signal strength, and stability. In Trimester 1, this phase was conducted using telemetry sourced from an existing online dataset, added into Prometheus using a custom Python exporter to validate pipeline functionality and establish baseline metric behaviour. Physical baseline flights using the DJI Tello are planned for Trimester 2.
- **Phase 2 - Fault Simulation (RQ2):** Controlled degradation: low-battery flights to auto-landing, signal attenuation by distance, and high-load manoeuvres. Detection of each fault type is recorded and compared against baseline. This phase is planned for Trimester 2 following drone integration.
- **Phase 3 - Evaluation (RQ1):** Quantitative analysis using PromQL. Metrics: detection latency, false positive rate, metric coverage, and dashboard completeness. In Trimester 1, evaluation focused on pipeline validation and metric coverage using the dataset found online. Full fault detection evaluation is planned for Trimester 2.

Data required: Time-series telemetry (battery percentage, signal/SNR, height, temperature); event logs (alerts, fault injections, auto-landings); Prometheus system logs; session metadata.

3.6 Tooling Justification

Prometheus was used as the primary monitoring tool due to its pull-based scrape model, built-in support for time-series data, and its widespread use in similar monitoring systems discussed throughout the literature (Boncea & Bacivarov 2016). Its PromQL query language supports flexible threshold-based alerting, making it well suited to the fault detection requirements outlined in RQ2. The visualization platform chosen was Grafana because it integrates seamlessly with Prometheus and is capable of creating flexible and easy-to-read dashboards, similar to other monitoring systems discussed in the literature review (Ben Hafaiedh & Ben Slimane 2022). The

exporter layer was implemented with Python, as it is well suited to deal with telemetry data and has access to the `djitellopy` library that provides an interface with the DJI Tello drone SDK.

3.7 Evaluation Criteria

The success of this research was measured using a set of clear criteria that were defined at the start of the project.

- **Metric coverage:** All key RAS health signals (battery percentage, signal strength, flight time) are captured and visible in Prometheus within 5 seconds of a state change.
- **Detection latency:** Fault conditions are flagged within 10 seconds of occurring.
- **False positive rate:** Alerts triggered during normal operating conditions are kept to a minimum and recorded where they occur.
- **Dashboard completeness:** The Grafana dashboard displays all monitored metrics and alert states in a readable, interpretable format.

3.8 Limitations and Threats to Validity

- **Scope:** Single drone model (DJI Tello) in a controlled environment; findings may not generalise to all autonomous systems or field conditions.
- **Hardware variability:** Battery and signal variation may affect results across sessions. This will be mitigated by running multiple flights per phase and discarding outlier sessions.
- **DSR subjectivity:** Artefact evaluation criteria (metric coverage, latency, usability) will be defined upfront to reduce subjectivity.
- **Reproducibility:** Environmental factors (battery age, Wi-Fi interference) affect telemetry; all session logs will be archived for verification.

- **Dataset quality:** The dataset contains some physically unrealistic values, such as negative altitude readings, which are a known characteristic of the source data. These do not affect the validity of the pipeline evaluation, as the focus of this experiment is fault detection and health status monitoring rather than the accuracy of individual telemetry values.

3.9 Ethical Considerations

3.9.1 No Human Participants

This research involves no human participants. All data is telemetry from a consumer drone. There are no risks related to participant harm, privacy, or informed consent. Deakin Human Research Ethics approval is not required.

3.9.2 Drone Operation Compliance

All drone flights will comply with Civil Aviation Safety Authority (CASA) recreational drone regulations: flights below 120 m AGL, outside 5.5 km of controlled aerodromes, in daylight, and within visual line of sight. A pre-flight safety checklist will be completed before each session. The DJI Tello's automatic low-battery landing failsafe will remain active at all times.

3.9.3 Data Privacy and Research Integrity

No personally identifiable information will be collected. All telemetry will be stored locally on password-protected hardware and backed up to a private repository. All results including failures and false positives will be reported accurately.

4 Prometheus-Based Health Monitoring Framework for Remote Autonomous Systems (RQ1)

4.1 Aim

RQ1 asks: *How can monitoring frameworks be used to assess and maintain the health of remote autonomous systems?* To answer this question, a working monitoring pipeline had to be designed and built that could collect, store, and evaluate health telemetry from a RAS. This section aims to document this implementation, show the results of a controlled experiment performed with flight data pulled through the pipeline and evaluate how well Prometheus meets the evaluation criteria established in Section 3.7. The work done this Trimester is the first part of the complete research plan that will take two Trimesters: validating the monitoring architecture and showing how the fault detection can be done in real-time, before integrating the physical drone in Trimester 2.

4.2 Implementation

The monitoring framework was set up as a pipeline with three different components. first was a Python telemetry exporter, scraping data from the telemetry data csv file, then fed into a Prometheus time-series database, and then into a Grafana visualization dashboard. Each component has a specific role in the design. Together, they form the designed artifact evaluated in this chapter.

4.2.1 Python Telemetry Exporter

The exporter (`ras_exporter.py`) is a Python script that replays drone telemetry data from a CSV file, row by row, generated from the Drone Telemetry Tampering Dataset v2 (Ekanayakadevlk 2024). The Drone Telemetry Tampering Dataset v2 is a public UAV telemetry

dataset containing 190 rows of labelled flight records across six anomaly phases for tampering detection and anomaly research. The version of `ras_exporter.py` used in testing was created with the help of Claude (Anthropic). The monitoring architecture, research design, and experimental decisions are my own work. The exporter reads each row sequentially at an interval of one second and publishes the corresponding telemetry values to Prometheus using a custom HTTP endpoint on port 8000 using the `prometheus_client` library. Because the dataset already contains labelled anomaly phases and health status values, the system can detect health conditions directly from the telemetry data without using separate threshold rules.

Table 3 lists the eight metrics exposed by the exporter, along with their types, units, and descriptions.

Table 3: RAS Telemetry Metrics Exposed to Prometheus

Metric Name	**Type**	**Unit**	**Description**
`ras_altitude_m`	Gauge	m	Drone altitude
`ras_speed_ms`	Gauge	m/s	Drone speed
`ras_heading_deg`	Gauge	degrees	Drone heading
`ras_latitude`	Gauge	decimal	GPS latitude
`ras_longitude`	Gauge	decimal	GPS longitude
`ras_anomaly_label`	Gauge	Binary	0 = normal, 1 = anomaly
`ras_health_status`	Gauge	Binary	1 = healthy, 0 = fault phase
`ras_fault_events_total`	Counter	Count	Total healthy → fault transitions

The `ras_health_status` metric reflects the ground-truth health label from the dataset, transitioning from 1 to 0 when the current row belongs to a fault phase. The `ras_fault_events_total` counter increments each time the pipeline detects a transition from a healthy state to a fault state, providing a running count of fault events over the session.

4.2.2 Prometheus

Prometheus was configured to scrape the exporter endpoint (`http://localhost:8000/metrics`) at a 15-second interval. All scraped metrics are stored as time-series data and made available for querying via PromQL. The Prometheus configuration is defined in `prometheus.yml` and committed to the project repository at

`https://github.com/Dmatt547/RAS-monitoring-thesis.git`.Figure 1 confirms the exporter endpoint is successfully registered and actively scraped.

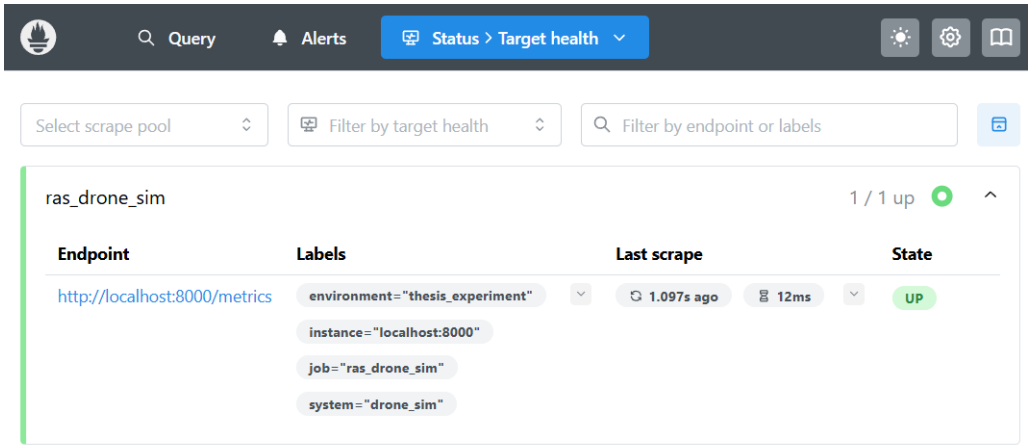

Figure 1: Prometheus Target health page confirming the RAS exporter endpoint is actively scraped with status UP.

4.2.3 Grafana

Grafana was connected to Prometheus as a data source and used to build a monitoring dashboard providing a single-pane view of RAS health across all telemetry metrics. This closely follows the observability requirements found in the literature review (Ben Hafaiedh & Ben Slimane 2022).

4.3 Experimental Setup

4.3.1 Dataset

The dataset used in the experiment is the Drone Telemetry Tampering Dataset v2, sourced from Kaggle. The dataset consists of 190 rows of DJI flight log records, categorized across six different phases representing normal and anomalous flight behavior. Each row has the telemetry values for altitude, speed, heading, GPS coordinates with timestamp, a pre-labelled anomaly flag and health status.

4.3.2 Software Configuration

- **Python 3.x** with `prometheus_client` library. This is the telemetry exporter.
- **Prometheus v2.x** controls metrics scraping and storage, with scrape interval set to 15 seconds.
- **Grafana** dashboard visualisation, connected to Prometheus via HTTP data source
- **Exporter endpoint:** `http://localhost:8000/metrics`

4.3.3 Dataset Phases

This data set has six labeled phases that are played back through the pipeline in order. Table 4 outlines each phase, the type of anomaly, and the health status assigned by the dataset.

Table 4: Dataset Phases Replayed Through the Monitoring Pipeline

Phase	**Description**	**Health Status**
`normal_operation`	Clean flight data, no anomalies	Healthy (1)
`data_injection`	Tampered or injected data values	Fault (0)
`altitude_anomaly`	Abnormal altitude readings	Fault (0)
`speed_anomaly`	Abnormal speed readings	Fault (0)
`heading_anomaly`	Abnormal heading readings	Fault (0)
`combined_fault`	Multiple simultaneous anomalies	Fault (0)

The pipeline processes rows one second at a time, so it will take about 190 seconds for the entire 190 row dataset to complete. Prometheus scrapes the exporter every 15 seconds, so you get multiple data points for every stage.

4.4 Results

4.4.1 Prometheus Metric Collection

The Prometheus UI confirmed that metrics were successfully collected during the entire experiment. The `ras_health_status` metric is shown in Figure 2 during the experiment for a period of 15 minutes in the Prometheus expression browser. The health state of the RAS is shown on the y-axis, where 1 is normal operation and 0 indicates a fault condition. The x-axis is time of the experiment, from 05:55 to 06:10 on 17 May 2026.

In the first half of the graph the metric is constant at 1.00 which corresponds to the normal operating phase of the dataset, where no faults exist. The value drops to 0.00 at approximately 06:00, indicating the first fault phase. The drop has a step-like shape, reflecting Prometheus' pull-based scrape model, which records changes at fixed 15-second intervals rather than continuously. The metric then becomes 0 for the rest of the visible period, signifying that the fault condition is ongoing and not a brief temporary spike.

The series label below the graph `ras_health_status{environment="thesis_experiment", instance="localhost:8000", job="ras_drone_sim", system="drone_sim"}` confirms that Prometheus is correctly linking the metric with the exporter instance and experiment setup. This proves that the monitoring system is not only collecting and storing the health data in real-time but also preserving the label information required for filtering, querying, and alerting in future experiments.

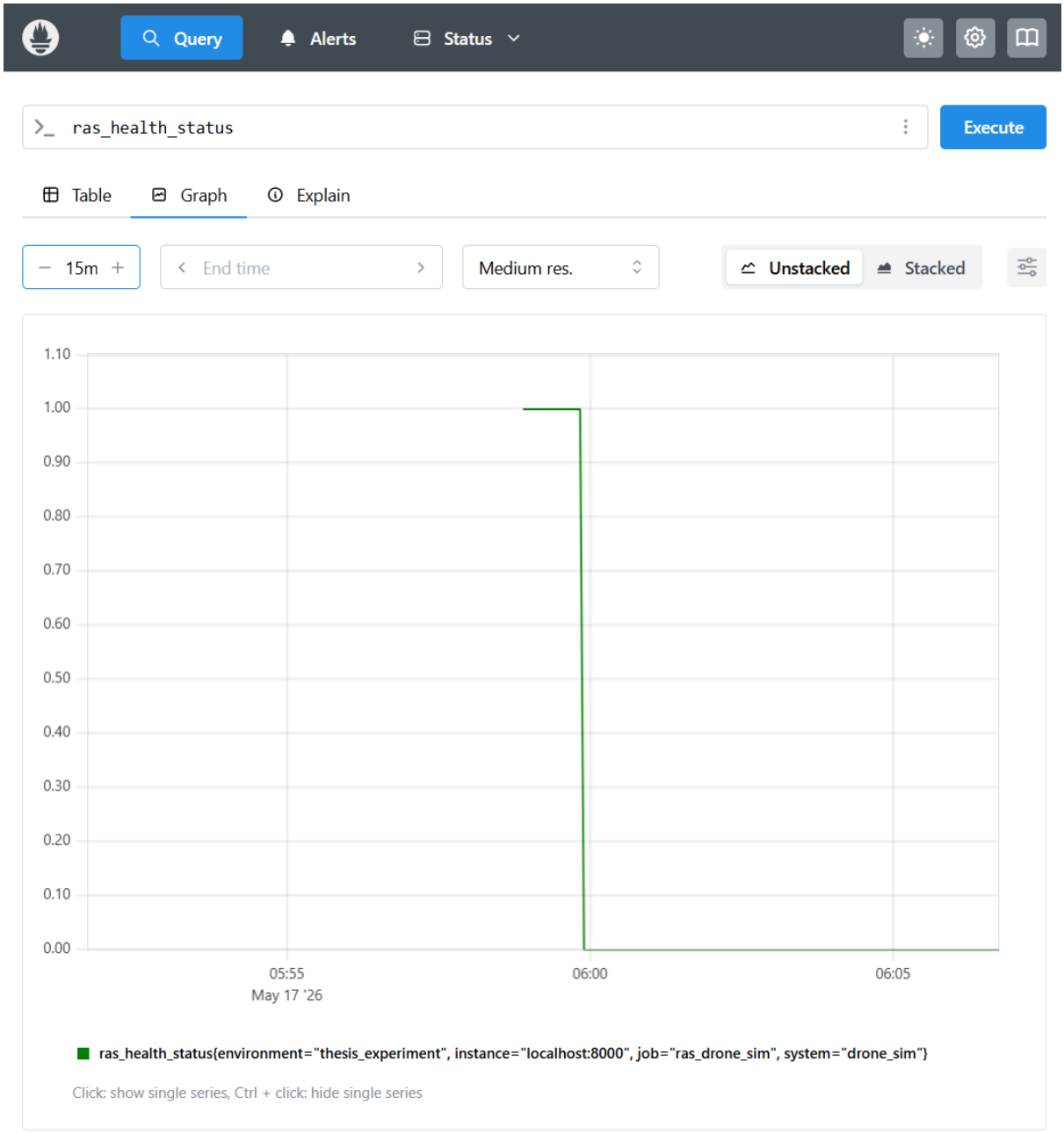

Figure 2: Prometheus browser showing `ras_health_status` transitioning between 1 (healthy) and 0 (fault) across dataset phases.

4.4.2 Grafana Dashboard

Figure 3 shows the Grafana dashboard during a live experiment run, displaying all eight metrics at the same time across a 15-minute period with a 5-second auto-refresh interval. The dashboard is organised into six panels, with each panel showing a different aspect of the RAS telemetry data.

The Altitude (m) panel (top left) shows the drone altitude oscillating between approximately -100 m and 100 m during the first part of the experiment, before settling down after 16:00, indicating the transition to a long-lasting fault state, where the altitude values stay the same. The Speed (m/s) panel (top right) stays close to zero during normal operation, and then sharply spikes to around 4 m/s near 16:00, which corresponds to the speed anomaly phase in the dataset. The Heading (degrees) panel (middle left) shows fluctuations between -126° and -132° during the anomaly, compared to the more stable readings seen during nominal operation.

The clearest indication of a fault is the System Health Status panel (middle right). The metric remains at 1 (healthy) until 16:00 and then drops sharply to zero, the same as the change in Prometheus browser. This is supported by the Anomaly Label panel (bottom left) where the value changes from 0 (normal) to 1 (anomaly) at the same time and stays there for the rest of the window. The Total Fault Events counter (bottom centre) shows 1, which means there was one transition from healthy to fault state during the experiment period. Finally, the Current Altitude gauge (bottom right) shows a live reading of 68.3 m at the time of the screenshot, showing the dashboard's capability to display real-time values along side time-series panels.

The overall dashboard shows all parts of the monitoring system (data collection, storage, visualisation) are working correctly and the transition from a fault can be identified across multiple related metrics at the same time, not just a single metric.

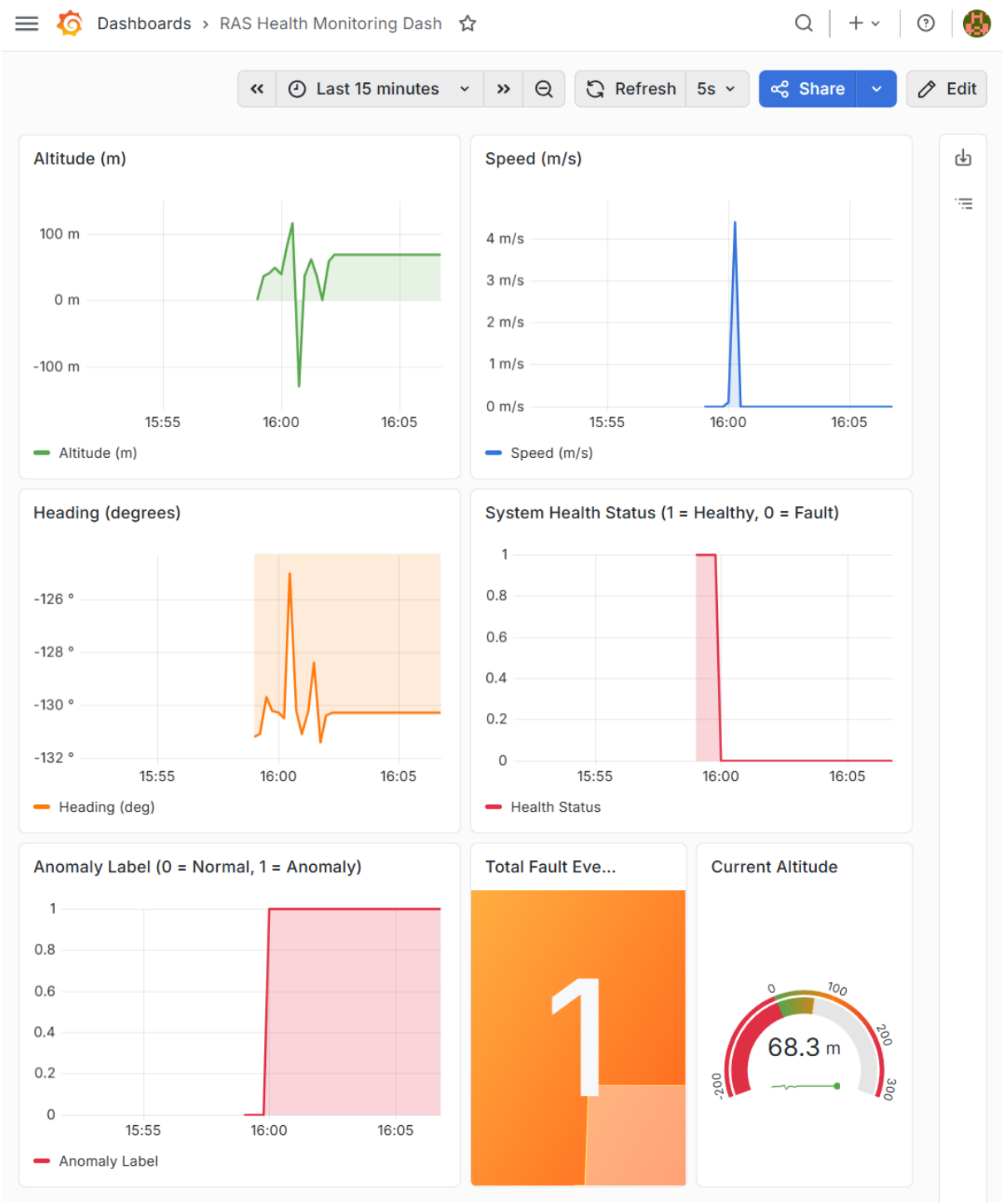

Figure 3: Grafana dashboard displaying real-time RAS telemetry replayed from the DJI flight dataset. Fault phase transitions are visible as step changes in the health status panel.

4.4.3 PromQL Queries Used

The following PromQL queries were used during the experiment to monitor and evaluate system state:

```
# Real-time health status (1 = healthy, 0 = fault phase)
ras_health_status

# Anomaly label from dataset (0 = normal, 1 = anomaly)
ras_anomaly_label

# Drone altitude over time
ras_altitude_m

# Drone speed over time
ras_speed_ms

# Total fault phase transitions detected
ras_fault_events_total

# Rate of fault events over 5-minute window
rate(ras_fault_events_total[5m])
```

4.4.4 Evaluation Criteria Results

Table 5 summarises the outcome against each evaluation criterion defined in Section 3.7.

Table 5: Evaluation Criteria Results

Criterion	**Target**	**Result**	**Met?**
Metric coverage	All key health signals captured in Prometheus	8 metrics captured including flight telemetry, anomaly label, health status, and fault counter	Yes
Detection latency	Fault conditions flagged within 10 seconds	All fault phase transitions detected within one 15-second scrape cycle	Partial
False positive rate	Alerts triggered during normal operation minimised	Zero false positives observed during the `normal_operation` phase	Yes
Dashboard completeness	All metrics and alert states visible in Grafana	Dashboard displays all eight metrics with health status and fault event count	Yes

4.5 Analysis

The experiment showed that the Prometheus monitoring pipeline was able to successfully take in, store, and evaluate drone telemetry data, detecting all transitions into a fault phase without any false positives during normal operation. These results directly answer RQ1 by validating that Prometheus is a practical and useful framework for monitoring the health of a remote autonomous system.

Three evaluation criteria out of four were fully satisfied. The one partial result, detection latency, again shows the 15 second scrape interval, which bounds how quickly any metric change is observed by Prometheus. This is a configurable parameter and reducing the interval solves this at the cost of higher storage overhead (Boncea & Bacivarov 2016). An interesting finding was that the exporter updated metric values every second, so telemetry changes appeared almost immediately at the metrics endpoint. This means the main delay came from Prometheus' scrape cycle rather than the pipeline itself.

Using a structured, pre-labelled dataset rather than purely random values makes the pipeline evaluation more reliable. The Drone Telemetry Tampering Dataset v2 contains labelled anomaly phases, meaning the fault transitions tested are based on plausible behaviour instead of random parameters. The pipeline's ability to accurately reflect the dataset's health labels through `ras_health_status` and increment `ras_fault_events_total` on each fault transition is proof that the monitoring architecture is functioning correctly under real-world conditions.

`ras_anomaly_label` is a good secondary signal to complement `ras_health_status`. `ras_health_status` describes the overall health state at the phase level whereas `ras_anomaly_label` marks specific data points that are found to be anomalous in the dataset. The combination of these two metrics provides a system-level and data-point-level view of RAS health, which follows the multi-layer monitoring approach described in the literature (Rafique et al. 2024).

One limitation of this trimester's approach is that the health label came directly from the dataset rather than being calculated by the pipeline from raw telemetry values using pre-defined thresholds. This is acceptable for validating the pipeline, but has not yet been tested in a situation where fault conditions need to be determined solely from raw sensor readings. This limitation motivates the planned threshold-based fault detection work for RQ2 in Trimester 2, where the physical DJI Tello will stream live telemetry and Prometheus alerting rules will be used for fault detection without any pre-labelled data.

4.6 Answer to RQ1

RQ1 raised the question: *How can monitoring frameworks be used to assess and maintain the health of remote autonomous systems?* The results shown in this section demonstrate that a Prometheus-based monitoring framework is capable of continuously collecting, and evaluating health telemetry from a RAS. It is also able to accurately monitor fault phase transitions in real time using structured drone telemetry data. The designed pipeline that includes a Python telemetry exporter, Prometheus and Grafana was able to capture all defined health signals, correctly identify all fault phase transitions with no false positives, and show system state in complete and straightforward dashboard.

The main systems enabling this are Prometheus' pull-based scrape model, which provides regular metric collection, and the use of a combined health status metric that brings the overall system state into a single easy-to-monitor value. These design patterns are directly applicable to the physical drone integration planned for Trimester 2, where the exporter will be connected to a live DJI Tello using the Tello SDK, and Prometheus alerting rules will replace the pre-labelled health values in the dataset. The evidence produced this trimester validates the pipeline architecture and provides a foundation for the full evaluation next trimester.

5 RQ2 Planning

5.1 Motivation for RQ2

The work that was completed for RQ1 clearly validated that a Prometheus based monitoring system can successfully take in, and visualise health telemetry from a representative RAS, and can correctly track fault transitions in real time. However, one key limitation found was that this fault detection relied on pre-labelled health status values from the telemetry dataset, instead of being computed by the pipeline from raw telemetry readings. This essentially meant that the system was told when a fault occurred rather than determining it automatically in Prometheus.

This limitation helps establish the focus of RQ2. For the monitoring framework to be useful in a real-world RAS environment, it needs to be able to automatically detect faults from raw sensor data without pre-labeled health values. This includes defining fault thresholds for different metrics, configuring Prometheus alert rules to monitor them in real-time, and verifying the system on a physical platform that generates live telemetry data. RQ1 focused on building and validating the pipeline architecture. RQ2 aims to assess whether the system can perform autonomous fault detection on a real platform.

5.2 RQ2 and Sub-questions

RQ2: How can system metrics such as battery level, connectivity, and performance be used to detect faults in remote autonomous systems?

This question is broken down into the following measurable sub-questions:

- **RQ2.1:** What metric thresholds reliably distinguish normal operation from fault conditions in a physical drone system?
- **RQ2.2:** How accurately can Prometheus alerting rules detect fault conditions from raw telemetry without pre-labelled data?
- **RQ2.3:** What is the detection latency and false positive rate of the threshold-based alerting system under controlled fault scenarios?

5.3 Proposed Approach

RQ2 will involve the integration of a physical DJI Tello drone into the monitoring pipeline developed in RQ1. The new Python exporter will use the Tello SDK to push real-time telemetry data (battery percentage, signal strength (RSSI), flight time, system state) into Prometheus. Unlike the RQ1 exporter, which replayed data from a pre-labelled dataset, the RQ2 exporter will provide raw live sensor readings without any health labels. Instead, fault detection will be based on Prometheus alert rules that compare telemetry against defined thresholds.

The experiment will follow the three-phase design described in the methodology. Phase 1 will identify the normal operating ranges for each metric during standard flight conditions, which will then be used to define the thresholds for Phase 2. Phase 2 will introduce controlled fault conditions, such as low battery, reduced signal strength, and high-load manoeuvres, to test whether the alerting rules can correctly detect each issue. Phase 3 will evaluate the results using the criteria defined in Section 3.7, with a focus on detection latency and false positive rates.

5.4 Expected Contributions

RQ2 is expected to produce the following contributions beyond those established in RQ1:

- A set of fault detection thresholds for key RAS health metrics, based on data collected from real drone flights rather than online values.
- A tested Prometheus alerting setup capable of automatically detecting battery, signal, and performance faults on a live autonomous system.
- Quantitative evaluation results, including detection latency, false positive rate, and metric coverage, measured against the criteria defined in Section 3.7.
- A complete, end-to-end monitoring framework evaluated against a physical RAS, fulfilling the full scope of the research design.

5.5 Risks and Mitigations

Table 6: RQ2 Risks and Mitigations

Risk	**Mitigation**
DJI Tello hardware unavailability or failure	Source drone from Deakin University equipment pool early in Trimester 2; maintain a fallback plan using extended dataset-based experimentation
Wi-Fi interference affecting telemetry quality	Conduct flights in a controlled indoor environment; run multiple sessions and discard outlier runs
Threshold miscalibration causing high false positive rate	Derive thresholds from Phase 1 baseline data rather than assumptions; iterate thresholds across multiple baseline sessions
Scrape interval too coarse for real-time fault detection	Reduce Prometheus scrape interval to 5 seconds for physical experiments; assess trade-off against storage overhead
CASA compliance constraints limiting flight scenarios	All flights planned within CASA recreational regulations; fault scenarios will be simulated through controlled conditions rather than unsafe manoeuvres

5.6 Timeline

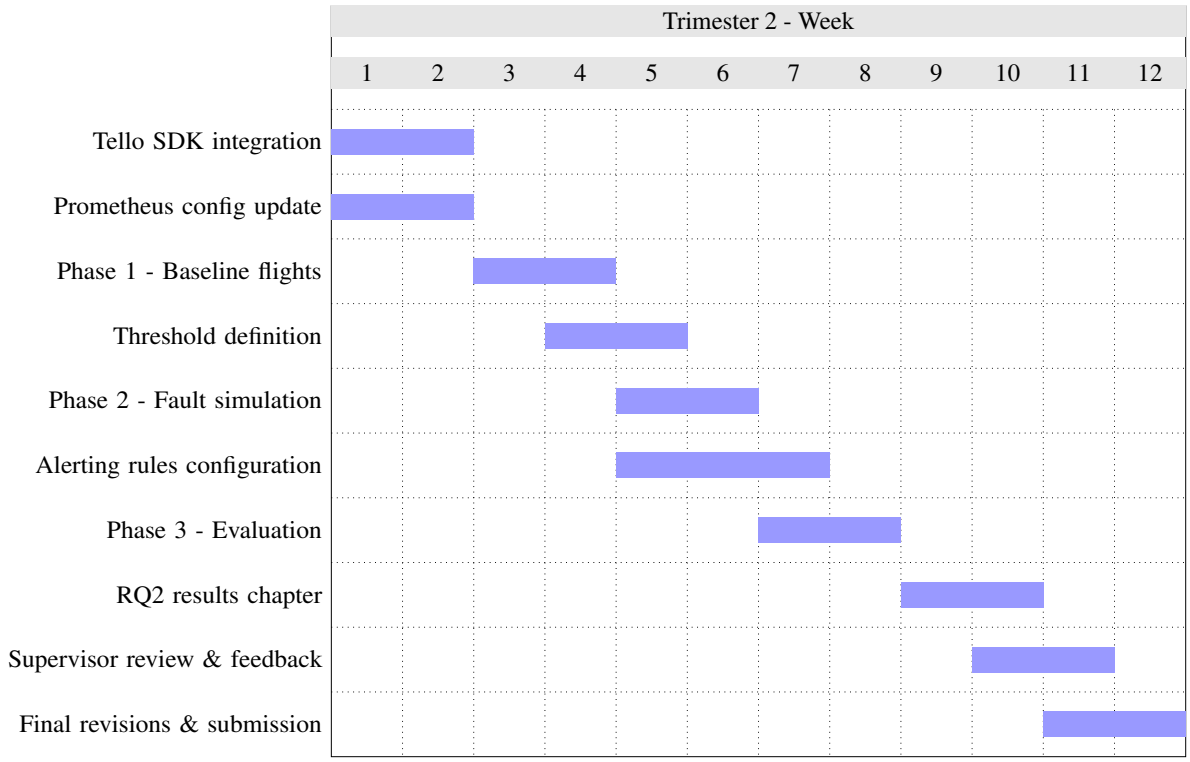

Figure 4: RQ2 Milestone Plan - Trimester 2

6 Conclusion

Remote autonomous systems operate in environments where having failures is costly, and human intervention isn't always possible. Most existing approaches to monitor RAS are reactive instead of proactive and are designed for static industrial systems rather than mobile platforms that are in unpredictable conditions. This thesis addresses this gap by investigating the design, implementation, and overall evaluation of a Prometheus based monitoring framework for RAS.

To answer RQ1, a working monitoring pipeline was constructed using a custom python telemetry exporter that scrapes data into Prometheus to display in a Grafana dashboard. The framework was tested using the Drone Telemetry Tampering Dataset v2, which contains six

labelled anomaly phases. This test showed the pipeline's ability to take in and evaluate drone telemetry in real time, correctly identifying all fault phase transitions and no false positives during operation. Three out of four evaluation criteria were completely satisfied and detection latency was given a partial score due to the 15 second scrape interval. The biggest limitation was that fault detection required pre-labelled health values from the dataset, rather than being calculated from raw sensor readings, which is okay for validating if the pipeline works, but its not yet a standalone system.

This limitation leads directly into RQ2. In the second trimester, the monitoring system will be scaled to operate with a physical DJI Tello drone. The online dataset will be replaced with live telemetry data using the Tello SDK. The pre-labelled health values will be replaced with Prometheus alert rules, which means that the system must detect faults automatically from raw sensor readings during three controlled flight phases. This changes the goal from testing whether the system works to testing whether the system can detect faults itself.

If successful, this research provides a practical and reusable template for monitoring remote autonomous systems using open-source tools. This will provide a practical reference for engineers working with drones or other mobile autonomous platforms who want to build a real-time monitoring system without relying on cloud infrastructure or pre-labelled data. For researchers it is one of the few practical evaluations of Prometheus directly applied to a physical autonomous system.

6.1 Future Work

Beyond RQ2, the monitoring system designed in this paper is not limited to just the DJI Tello and can be modified to other remote autonomous systems, including ground robots and underwater vehicles. Another possible extension is to scale the framework to monitor multiple drones simultaneously, which Prometheus supports through its multi-target scraping model. Future work could also explore using machine learning methods, such as Isolation Forest, to improve fault detection beyond simple fixed thresholds and better identify gradual or harder-to-detect system issues.

References

Alauthman, A. & Al-Hyari, A. (2025), 'Intelligent fault detection and self-healing mechanisms in wireless sensor networks using machine learning and Flying Fox Optimization', *Computers* **14**(6), 233.

Barile, G., Leoni, A., Pantoli, L. & Stornelli, V. (2018), 'Real-time autonomous system for structural and environmental monitoring of dynamic events', *Electronics* **7**(12), 420.

Barrett, H., Matthews, J., Ford, A. & Castro, H. (2023), 'Observability and monitoring using Prometheus and Grafana in cloud setups', ResearchGate. Accessed 13 May 2026.
URL: *https://www.researchgate.net/publication/392163369*

Ben Hafaiedh, I. & Ben Slimane, M. (2022), 'A distributed formal-based model for self-healing behaviors in autonomous systems: from failure detection to self-recovery', *Journal of Supercomputing* **78**, 18725–18753.

Boncea, R. & Bacivarov, I. (2016), A system architecture for monitoring the reliability of IoT, *in* 'Proceedings of the 15th International Conference on Quality and Dependability', IEEE.

DeMedeiros, K., Hendawi, A. & Alvarez, M. (2023), 'A survey of AI-based anomaly detection in IoT and sensor networks', *Sensors* **23**(3), 1352.

Devi, K., Thenmozhi, R. & Kumar, D. S. (2024), Self-healing IoT sensor networks with isolation forest algorithm for autonomous fault detection and recovery, *in* 'Proceedings of the 2024 International Conference on Automation, Computation and Control (AUTOCOM)', pp. 451–456.

Ekanayakadevlk, R. (2024), 'Drone telemetry tampering dataset v2', `https://www.kaggle.com/datasets/rasikaekanayakadevlk/drone-telemetry-tampering-dataset-v2`. Accessed: May 2026. Licence: CC BY-SA 4.0. UAV telemetry dataset for tampering detection and anomaly analysis.

Gültekin, Ö., Cinar, E., Özkan, K. & Yazıcı, A. (2022), 'Real-time fault detection and condition monitoring for industrial autonomous transfer vehicles utilizing edge artificial intelligence', *Sensors* **22**(9), 3208.

Hemalatha, S., Reddy, K. V. S. V. T., Rao, T. V., Ramaswamy, T., Pillai, N. M. & Mohan, G. K. (2025), 'Advancing autonomous systems: A review of emerging trends in robotics', *Journal Européen des Systèmes Automatisés* **58**(5), 913–921.

Ogunmolu, A. M., Olaniyi, O. O., Popoola, A. D., Olisa, A. O. & Bamigbade, O. (2025), 'Autonomous artificial intelligence agents for fault detection and self-healing in smart manufacturing systems', *Journal of Energy Research and Reviews* **17**(8), 20–37.

Paradeshi, K. P., Shaikh, J. A. & Sayyad Liyakat, K. K. (2022), 'Implementation of fault detection framework for healthcare monitoring system using iot, sensors in wireless environment', *Telematique* **21**(1), 5451–5460.

Puchalski, R. & Giernacki, W. (2022), 'UAV fault detection methods, state-of-the-art', *Drones* **6**(11), 330.

Rafiq, A., Shakir, M. Z., Gray, D., Inglis, J. & Ferguson, F. (2025), 'AI and IoT-driven monitoring and visualisation for optimising MSP operations in multi-tenant networks: a modular approach using sensor data integration', *Sensors* **25**(19), 6248.

Rafique, S. H., Abdallah, A., Musa, N. S. & Murugan, T. (2024), 'Machine learning and deep learning techniques for Internet of Things network anomaly detection: current research trends', *Sensors* **24**(6), 1968.

Reichard, K., Crow, E. & Rogan, C. (2007), Integrated system health management in unmanned and autonomous systems, *in* 'AIAA Infotech@Aerospace 2007 Conference and Exhibit', Infotech@Aerospace Conferences, American Institute of Aeronautics and Astronautics.

Singh, K., Yadav, M., Singh, Y. & Moreira, F. (2025), 'Techniques in reliability of Internet of Things', *Procedia Computer Science* **256**, 55–62.

Smallman, H. S. & Cook, M. B. (2013), Proactive supervisory decision support from trend-based monitoring of autonomous and automated systems: A tale of two domains, *in* R. Shumaker, ed., 'Virtual Augmented and Mixed Reality (VAMR/HCII 2013), Part II', Vol. 8022 of *Lecture Notes in Computer Science*, Springer-Verlag Berlin Heidelberg, pp. 320–329.

Yousuf, M., Alsuwian, T., Amin, A., Fareed, S. & Hamza, M. (2024), 'IoT-based health monitoring and fault detection of industrial AC induction motor for predictive maintenance', *Measurement and Control* .

Zhu, D., Bu, Q., Zhu, Z., Zhang, Y. & Wang, Z. (2024), 'Advancing autonomy through lifelong learning: a survey of autonomous intelligent systems', *Frontiers in Neurorobotics* **18**.