INDUCTIVE MONITORING SYSTEMS: A CUBESAT GROUND-BASED PROTOTYPE

A Thesis
presented to
the Faculty of California Polytechnic State University,
San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in Aerospace Engineering

by
Michelle Kristyn Haddock
December 2015

COMMITTEE MEMBERSHIP

TITLE:	Inductive Monitoring Systems: A CubeSat Ground-Based Prototype
AUTHOR:	Michelle Kristyn Haddock
DATE SUBMITTED:	December 2015
COMMITTEE CHAIR:	Eric Mehiel, Ph.D. Professor of Aerospace Engineering
COMMITTEE MEMBER:	Kira Abercromby, Ph.D. Assistant Professor of Aerospace Engineering
COMMITTEE MEMBER:	John Bellardo, Ph.D. Associate Professor of Computer Science
COMMITTEE MEMBER:	Jordi Puig-Suari, Ph.D. Professor of Aerospace Engineering

ABSTRACT

Inductive Monitoring Systems: A CubeSat Ground-Based Prototype

Michelle Kristyn Haddock

Inductive Monitoring Systems (IMS) are the newest form of health monitoring available to the aerospace industry. IMS is a program that builds a knowledge base of nominal state vectors from a nominal data set using data mining techniques. The nominal knowledge base is then used to monitor new data vectors for off-nominal conditions within the system. IMS is designed to replace the current health monitoring process, referred to as model-based reasoning, by automating the process of classifying healthy states and anomaly detection. An IMS prototype was designed and implemented in MATLAB. A verification analysis then determined if the IMS program could connect to a CubeSat in a testing environment and could successfully monitor all sensors on board the CubeSat before in-flight use. This program consisted of two main algorithms, one for learning and one for monitoring. The learning algorithm creates the nominal knowledge bases and was developed using three data mining algorithms: the gap statistic method to find the optimal number of clusters, the K-means++ algorithm to initialize the centroids, and the K-means algorithm to partition the data vectors into the appropriate clusters. The monitoring algorithm employed the nearest neighbor searching algorithm to find the closest cluster and compared the new data vector with the closest cluster. The clusters found were used to establish the knowledge bases. Any data vector within the boundaries of the clusters was deemed nominal and any data vector outside the boundaries was deemed off-nominal. The learning and monitoring algorithms were then adapted to handle the data format used on a CubeSat and to monitor the data in real time. The developed algorithms were then integrated into a MATLAB GUI for ease of use. The learning and monitoring algorithms were verified with a 2-dimensional data set to ensure that they performed as expected. The final IMS CubeSat prototype was verified using 56-dimensional emulated data packages. Both verification methods confirmed that the IMS ground-based prototype was able to successfully identify all off-nominal conditions induced into the system.

ACKNOWLEDGMENTS

I would like to take the time to thank the people that have helped me get to where I am now.

First, I would like to thank John Bellardo, Ph.D., for the inspiration to pursue this thesis topic. Even though he is part of the Computer Science Department, Dr. Bellardo understood my passions and guided me in the direction of experimental health algorithms. Without Dr. Bellardo's guidance, I probably would still be wondering what topic I should do my thesis on.

I would also like to thank Sean Sheen. Despite being a computer science graduate student himself, Sean took many hours out of his days to help me get associated with the CubeSat test setup. Without his assistance, this thesis would not have reached its completion.

I would also like to thank my parents, Randy and Susan, for their unwavering support through this experience.

Finally, I would like to thank the three people who were there for me day in and day out, Chris Barlog, Tim Fitzgerald, and Michael Strange. Whether it be making this experience as enjoyable as it was, encouraging me to get it together, or just being there when things got rough, this journey wouldn't have been the same without them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

Monitoring the health of a spacecraft is vital to the success of a mission and can be a very complex task before and after launch. Health monitoring determines if the spacecraft is working as expected by comparing the current telemetry data values to a range of expected values. If the current telemetry data values fall within the expected range, the spacecraft is considered healthy. If not, the spacecraft is considered unhealthy.

1.1 MOTIVATION

Currently, the process of health monitoring is tedious and requires extensive manual labor. A newer approach that can automate portions of the health monitoring process is currently being researched and developed and an implementation of such an approach is the topic of this thesis.

1.1.1 CURRENT HEALTH MONITORING

When simplifying the current process of health monitoring, there are three separate steps. The first step in health monitoring is determining what constitutes a healthy system. A healthy spacecraft, or one that is operating nominally, is defined as a spacecraft with telemetry values that are within a range of expected values. Creating a reference table of the expected values is the most common practice and requires extensive knowledge of the system. In addition to requiring someone who knows the entirety of the system well, a significant amount of time is needed to develop a reference table of nominal value ranges that accurately describes all the nominal health states of a spacecraft. While determining the health states is reasonable for a simple system, “Determining the health state of [sophisticated and complex] systems using traditional methods is becoming more difficult as the number of sensors and component interactions grows” [10]. At times, creating a model that accurately describes all the interactions and states within a complex system is deemed too difficult or impossible.

The second step in monitoring the health of a spacecraft is to consistently read the spacecraft’s current telemetry values and compare them to the reference table of healthy telemetry values. The health of the spacecraft is then determined by examining this comparison. If all of the spacecraft’s telemetry values are contained within the range of healthy values in the reference table, the system is deemed healthy; the spacecraft is in a nominal state. If one or more values fall outside the healthy value ranges, the system is deemed unhealthy; the spacecraft is in an off-nominal state.

The final step in health monitoring is determining the cause of an off-nominal state, as well as a procedure to remedy the anomaly. This step is executed on the ground by engineers involved with the spacecraft. Determining the cause of the off-nominal state requires a quick investigation into the anomaly and quick decision-making from a team of experts on how to return the spacecraft to a healthy state.

1.1.2 A NEW METHOD OF HEALTH MONITORING

The three steps of health monitoring are, at the time of this work, the standard practice in the spacecraft industry. The main goal of a new proposed method of health monitoring, Inductive Monitoring Systems (IMS), is to automate the first two steps of the process. The IMS method can be broken down into two sections: the learning algorithm and the monitoring algorithm. The learning algorithm builds its own reference table of nominal value ranges from an existing data set, eliminating the need for an expert to determine the nominal value ranges. The cost of the development period is reduced via the process automation. The learning algorithm also increases the reliability of a complex system by being able to learn the healthy value ranges of what previously was too complex to define. The monitoring algorithm uses the nominal value ranges generated by the learning algorithm as the comparison for the system's current telemetry values. The monitoring algorithm then autonomously determines if the spacecraft is operating nominally or off-nominally. If off-nominal, the monitoring algorithm reports a quantitative value that demonstrates how far the system has deviated from nominal.

1.1.3 IMS BACKGROUND

To date, IMS has been integrated into a wide range of applications. As David L. Iverson wrote in his paper *Inductive System Health Monitoring*, "The IMS methodology is domain independent and can be used in a variety of system monitoring situations including aerospace, transportation, manufacturing, power generation and transmission, medical, or process monitoring applications" [9]. IMS is gaining popularity in many fields because "the advantage of using IMS is that it is fast and simple yet very effective" [7].

Halim described IMS in the previous quote when referring to why he chose to utilize IMS to monitor the equipment used in his field of mining, mineral, and metal processing. In another study, IMS was used to "maintain effective plug load management system performance, identify malfunctioning equipment, and reduce building energy consumption" [18]. While IMS is applicable to other industries, it was originally founded in the the aerospace industry and recently gained popularity.

1.1.3.1 STS-107 COLUMBIA SPACE SHUTTLE

The first known investigation into using IMS in an aerospace application was conducted by Iverson in 2004. Iverson applied IMS to the archived data of the STS-107 Columbia Space Shuttle mission. The STS-107 mission and the lives of the crew members on board were lost when the orbiter was destroyed upon re-entry. The cause of the destruction was determined to be "a breach in the Thermal Protection System on the leading edge of the left wing, caused by a piece of insulating foam that struck the wing approximately 82 seconds after launch" [9]. This breach went unnoticed by mission controllers until 17 days later at the time of re-entry. A slight increase in brake line temperature of the left main landing gear was noticed seven minutes before the loss of the vehicle.

Iverson used IMS to determine the nominal ranges of archived temperature sensor data from previous successful Columbia Space Shuttle missions. He then used those nominal ranges to analyze the archived telemetry data from the STS-107 Columbia Space Shuttle mission. Iverson's investigation concentrated on four temperature sensors on each of the two wings. The results of the investigation are shown in Figure 1, where the pink line shows the results of the left wing and the blue line shows the results of the right wing. The figure shows the IMS distance over time. The IMS distance is a measurement of how far from nominal the current telemetry value has deviated. At 15:40:22, a vertical line shows when the breach in the brake line occurred. Before the impact, the right wing and left wing had similar trends. The left wing IMS distance appears to increase before the impact, but this is not the indication of the error. Instead, the indication of the error from IMS is seen after the impact as the overall trend of the left wing drastically strays away from the right wing and does not return to nominal. Iverson concluded that this significant difference in the trends of the left wing and right wing was an early indication that an error had occurred.

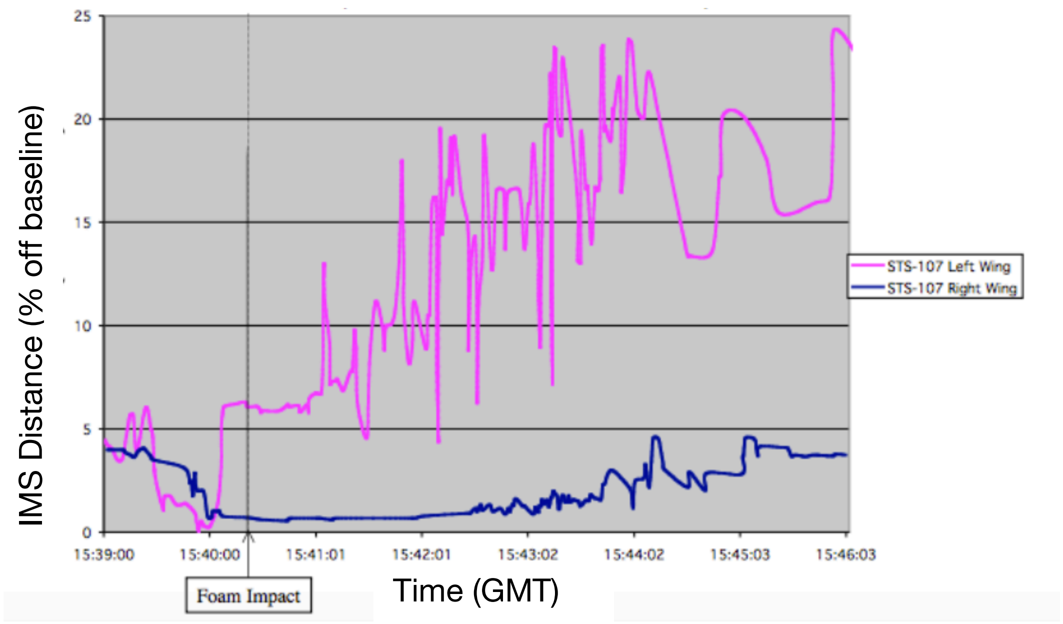


Figure 1: IMS Results for STS-107 [9]

Iverson concluded from this investigation that IMS could provide monitoring capability similar to, if not better than, the current techniques used. IMS could also aid in alerting a mission controller of vehicle health and provide earlier detection of anomalies [9].

1.1.3.2 BEACON-BASED EXCEPTION ANALYSIS FOR MULTI-MISSIONS (BEAM)

The NASA Jet Propulsion Laboratory and the NASA Armstrong Flight Research Center took the next stride in IMS research and application by integrating the JPL-developed IMS into an F/A-18. The idea Mackey et. al. had behind using an aircraft as a proxy to a spacecraft was that

> "A high-performance aircraft provides many of the same relevant characteristics and challenges as a spacecraft and could effectively be used as a surrogate for developing new technologies for space flight. […] Software development

> requirements for data collection, data filtering, and interpretation are comparable. In addition, issues involved in modeling, integrating, and fielding [IMS] are similar for both platforms" [14].

Mackey et. al believed that if IMS performed as expected on an aircraft, the results would translate to a spacecraft.

JPL developed their own version of IMS that could monitor a complex system, such as the F/A-18, without the need for a manually developed model. JPL then integrated their version of IMS with their Beacon-Based Exception Analysis for Multi-Missions (BEAM) software. Mackey et. al describes BEAM as a "software technology that analyzes system data to detect anomalies, classify faults, and track degradation in physical systems" [14] and reduces the amount of telemetry data transferred to the mission controller.

BEAM reduces the amount of data by transmitting only a beacon consisting of one of the following health specifications to the mission controller: healthy, anomalous behavior, degradation, or failure. When the system is specified as healthy, only the beacon, no telemetry data, is transmitted. When the beacon is one of the three latter health specifications, only the pertinent telemetry data associated with the anomalous event is transmitted. A beacon-based program reduces the amount of transmitted telemetry data during the majority of the mission but still allows for an investigation into the data in the case of an anomaly [11].

The JPL-developed IMS and BEAM were integrated into the flight software on-board the F/A-18 and test flights were performed at the Armstrong Flight Research Center. The team concluded that the test bed was appropriate for IMS and that it met all of their success criteria. This conclusion brought IMS up to Technology Readiness Level (TRL) 6, defined as “prototyping implementations on full-scale realistic problems” [4] and readied the process for TRL 7, which is a demonstration of the prototype in its operational environment [12].

1.1.3.3 ARES I-X GROUND DIAGNOSTIC PROTOTYPE

IMS was most recently implemented into the Thrust-Vector Control (TVC) system of an Ares I-X launch. A ground diagnostic prototype of IMS was developed to support the detection of anomalies during launch. The results were later compared to other diagnostic programs in use at the time of launch to assess the abilities of IMS.

Due to the lack of previous test data of the Ares I-X, data collected from the TVC system of previous Space Shuttle missions were used to create the nominal telemetry value ranges. This application of IMS failed to accurately find all of the failures within the TVC system. The team concluded that the performance of IMS is directly related to the quality of the original nominal data set that is provided to create the knowledge base. Due to the difference in the hardware between the two TVC systems the IMS failed to detect some errors, as well as gave false-positives. They concluded that IMS added value to the health determination process, but a nominal data set with higher fidelity would have yielded better results. IMS had some benefits over the other model-based diagnostic programs but would require improvement before it was deemed a success for this application [16].

1.1.4 CUBESAT APPLICATION

Currently, the IMS method is referred to as an experimental algorithm because it has not been proven in-flight on an entire system. By integrating the IMS method into a CubeSat's software architecture, flight heritage of an entire system is gained at a lower risk due to the smaller cost of the overall mission.

Adding IMS to the software architecture of a CubeSat will not only add reliability to the system, but the addition will also improve the scientific goals of the mission. Currently, CubeSats downlink their data, telemetry and scientific, during very short communication passes. The passes occur three to four times per day and are currently only about twelve minutes long. The telemetry portion of data utilizes about 10% of the data downlink when a CubeSat is operating nominally. Due to the limitations inherent to the CubeSat design, the data rate of a CubeSat is very limited. Implementing IMS into the software architecture provides an alternative method to obtaining more scientific data from the CubeSat by reducing the telemetry data. When IMS is fully integrated to detect anomalous conditions, there is no need to transmit the telemetry data during a downlink for post-processing. A reduction in the volume of telemetry data will result in more critical science data (approximately 11%) during a downlink.

In order to fully integrate IMS into the flight software of a CubeSat, a step must be made beforehand to increase the success of such implementation. The first step is to verify that IMS can successfully monitor and determine the faults in a CubeSat. The first verification process of an experimental algorithm, such as IMS, is performed in a non-essential setting. This means that IMS should not be fully relied upon to detect faults until it has been verified that it has the ability to detect all the faults in a system. To verify that IMS can be relied upon in flight to detect faults in the system, a prototype must first be built and applied to a CubeSat during testing on the ground. Once the prototype of the algorithm has demonstrated the ability to to detect faults in the system, it then is developed for flight software, tested, and flown on-board a CubeSat.

1.2 THESIS OVERVIEW

This thesis describes the development of a ground-based prototype of IMS developed for a CubeSat testbed at Cal Poly. This thesis describes how IMS was implemented in MATLAB and a user-friendly interface was developed and integrated into the CubeSat testing environment. Finally, this thesis demonstrates the success of the IMS method in determining faults when monitoring an emulated CubeSat system.

This paper first introduces the methodology of Inductive Monitoring Systems in Chapter 2. Inductive Monitoring Systems are composed of two distinct algorithms: learning and monitoring. Chapter 3 and Chapter 4 describe these algorithms respectively and their specific implementation chosen for this application. Chapter 5 describes how the algorithms are integrated in this implementation of IMS to achieve the expected outcome, as well as how the algorithms interact with the CubeSat testing environment. Chapter 6 explains the verification process that was performed on the developed IMS software. Finally, Chapter 7 concludes the paper and describes the future work needed to fully integrate IMS into CubeSat flight software.

2 INDUCTIVE MONITORING SYSTEMS (IMS)

Currently the spacecraft industry uses model-based reasoning to predict when a system is functioning nominally or off-nominally which requires building a theoretical model of the system. The model uses various ranges of simulation software inputs to compute ranges of outputs that determine the nominal system states. To determine if a system is preforming nominally, the telemetry data collected from the system is compared against the range of outputs generated by the simulation software. If the telemetry data is within the expected nominal range, the system is classified as performing nominally and is healthy. If the telemetry data is outside the expected range, it is off-nominal. This type of error-detection has a long-standing history of success despite the models being difficult and time-consuming to build. The aerospace engineering industry continues to push the limits of design and create even more complex systems. With the movement toward more complex designs, modelling the overall system has become increasingly difficult and in some cases, impossible. This is where the IMS approach has value and thus has developed interest in the research and development of it.

IMS is a software that uses a nominal data set to build a knowledge base of the various states of nominal behavior. IMS then uses that nominal knowledge base to monitor the health of a system in real time. To build the required knowledge bases IMS utilizes techniques developed for machine-learning and data mining [2].

IMS differs from other monitoring systems because IMS doesn't need to see a failure or error in order to assess when an error occurs. Instead, IMS proactively monitors the system's deviation from the nominal state. The deviation is defined as the distance between the system's current state and the nominal state. When the calculated deviation has exceeded the maximum deviation allowed, the system is deemed off-nominal.

The motivation for IMS partially stems from the difficulty in modeling systems due to their complexity. IMS does not require a model but rather a nominal data set [2]. The IMS software can monitor systems with nearly the same fidelity as the model-based reasoning approach but with less effort in the development of the nominal states [2].

Inductive System Health Monitoring (ISHM) is a specific form of IMS that was described by Iverson at the 2004 International Conference on Artificial Intelligence and is the particular method being applied to the CubeSat testbed prototype.

2.1 ISHM METHODOLOGY

IMS is broken down into two separate algorithms: the learning algorithm and monitoring algorithm. Figure 2 gives a visual representation of the relationship between the two algorithms.

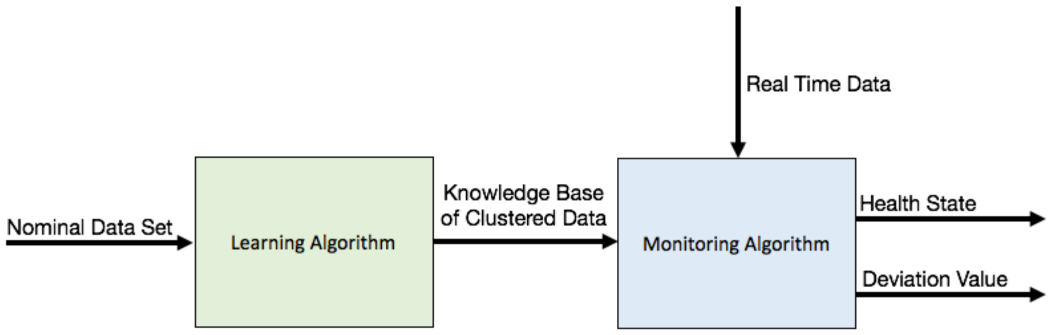


Figure 2: Relationship Between the Learning and Monitoring Algorithms

ISHM first builds a knowledge base of nominal data sets that will later be used for health determination. The learning algorithm uses data-mining and machine-learning algorithms on archived data in order to gather a generalized nominal data set. This generalized data set covers all states of the system. The ISHM algorithm then clusters the data into groups of similar values. Clustering assigns the data vectors into groups such that the data in a group are as similar as possible and data in different groups are as dissimilar as possible. Those groups, or clusters, define the telemetry value limits for a particular state of the system. Each cluster defines a different nominal state of the spacecraft quantitatively.

In order to utilize the learning algorithm, the data must be in a particular form. The form suggested for ISHM is a state vector consisting of parameter values. The parameter values are the individual sensor value measurements included in the telemetry data. Because ISHM is being developed to monitor any number of parameters, the vectors define a point in an *N*-dimensional space, where *N* is the number of parameters being monitored. This data vector is time variant; multiple data vectors would represent the state of the system at multiple times. An example of a vector that would be used in this thesis is shown in Table 1.

Table 1: ISHM Data Vector Example

Sensor 1: Power [A]	Sensor 2: Power [V]	Sensor 3: Temperature [K]	…	Sensor N: Parameter [Unit]
3	32	298	…	Value

After the parameter values are properly formatted into the state vector, the data is then clustered into groups of similar values. While there are various clustering methods that can be used, the K-means clustering algorithm is recommended and is described later in this paper. The maximum and minimum of each parameter in a cluster describe the range of nominal values allowed. An example of the final ISHM cluster structure is shown in Table 2.

Table 2: ISHM Cluster Structure

	Sensor 1: Power [A]	Sensor 2: Power [V]	Sensor 3: Temperature [K]	…	Sensor N: Parameter [Unit]
Minimum	2.1	31.1	295.3	…	Value
Maximum	4.3	33.2	298.3	…	Value

The centroid (or center of each cluster) is then defined as a vector in *N*-dimensional space in which each component of the vector is the average of the sensor values contained within that cluster for that particular parameter. This is where the ISHM algorithm has similarities to the model-based reasoning algorithm; the allowable nominal values are contained within a range. In model-based reasoning, a range of values is sent into the developed simulation software and a range of allowable response values is output to form the nominal range of values.

Now that healthy nominal state vectors have been defined using the learning algorithm, a monitoring algorithm is used to determine the current health of the system. Telemetry is gathered from the system and the monitoring algorithm formats the telemetry data into the same vector format shown in Table 1. The monitoring algorithm then utilizes a nearest-neighbor searching algorithm to locate the closest cluster, measured by distance, to the data vector being analyzed.

Once the closest cluster has been found, the algorithm determines if the telemetry gathered from the CubeSat falls within the cluster limits, shown in Table 2. The cluster limits are referred to as the bounding *N*-dimensional hypercube. If the telemetry data falls within the cluster limits, the algorithm concludes the system is performing nominally (healthy). If the telemetry data does not fall within the limits, the algorithm calculates the deviation value which is a ratio of two quantities: the distance between the data vector and the centroid and the distance between the furthest point of the cluster and the centroid.

If the telemetry does not fall within the limits of the closest cluster's *N*-dimensional hypercube, the algorithm first determines if it falls within a threshold value, ε. The threshold value is previously determined by the user and is defined as the maximum allowable distance between the center of a cluster and the data point being analyzed. If the telemetry falls within the threshold value, the cluster structure is updated to include the new value. This threshold value allows for the knowledge base to be continuously updated as the system is operated. By adding in the threshold value the assumption is made that not every single nominal state vector was accounted for in the nominal data set provided to the learning algorithm.

The user can also make the assumption that all nominal ranges were covered in the learning algorithm and not use the ε threshold value. The size of the threshold value is very important: too large and errors may be missed, too small and false positives for errors may be seen. There is not an exact value to set the threshold value to and it must be adjusted for the exact application at hand.

3 LEARNING ALGORITHM

The learning algorithm uses machine-learning and data mining techniques on archived data to create a nominal knowledge base. A nominal knowledge base is composed of clusters of data that contain similar values. The clusters of data will quantitatively classify the nominal states of a system. The methodology laid out in Iverson's proposal of ISHM was very broad and merely suggested various techniques for clustering the data. This chapter will describe in detail the specific methods chosen for this implementation.

3.1 K-MEANS CLUSTERING ALGORITHM

There are many clustering techniques available in the data mining field. In the ISHM methodology laid out by Iverson, he suggests the use of the K-means algorithm for this application. K-means is a widely utilized unsupervised machine-learning algorithm that has become popular due to its simplicity. An unsupervised algorithm is defined by the ability to run without a response variable [13]. The goal of the K-means algorithm (or any clustering algorithm) is to find groups of data points within a data set in which intra-cluster data points are as similar as possible, while inter-cluster data points are as dissimilar as possible [5]. To define this quantitatively, the K-means algorithm tries to minimize the intra-cluster variance. Variance is defined as the sum of the squared distances from the data points to their assigned cluster's centroids.

The K-means algorithm originates from the vector quantization techniques developed from signal processing. Vector quantization organizes vectors into groups such that each group has approximately the same number of points. This method is usually used for data compression because it significantly reduces the size of the data from the number of data points to the number of clusters. It is considered a lossy compression method because not every data point is remembered; the mean of the data points in a particular cluster is remembered and thus the data resolution is decreased [20]. K-means was developed by J. MacQueen with the intention of taking this idea, altering it slightly, and using it in various applications such as "methods for similarity grouping, nonlinear prediction, approximating multivariate distributions, and nonparametric tests for independence among several variables" [15].

The K-means algorithm is fairly simplistic and iterates as follows [3]:

1. Choose *K* number of points at random from the data set. These points will be the initial centroids.
2. Calculate the distances between every data point and each centroid.
3. Determine the closest centroid to each data point and assign the data point to that centroid's cluster.
4. Calculate the mean of all the data points assigned to a cluster. This mean becomes the new centroid.

5. Iterate on steps 2-4 until the centroid no longer moves or moves less than a set tolerance.

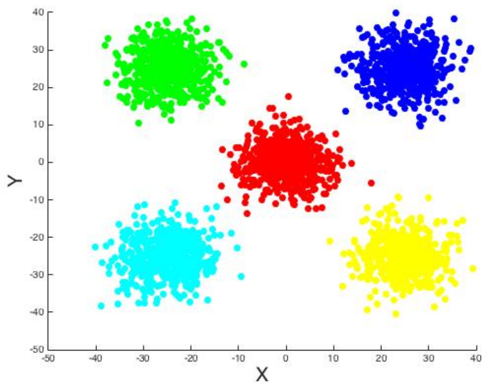
There are many different methods used to determine the distance between a data point and the centroids. For this application, the Euclidean distance was recommended and is shown below for two points, x and y, that are composed of n dimensions each (*i, j, … n*).

$$D_{xy} = \sqrt{(x_i - y_i)^2 + \left(x_j - y_j\right)^2 + \cdots + (x_n - y_n)^2} \tag{1}$$

3.1.1 SHORTCOMINGS OF K-MEANS

The K-means algorithm random initialization was not appropriate for this application. The K-means clustering algorithm is very sensitive to the initial centroids, which are chosen at random. The best possible case, or the global optimum, is when each centroid ends in its own natural cluster. With random initialization, there is a probability that two centroids will end in the same natural cluster or that a centroid will converge on a location in between natural clusters. Any case where a natural cluster does not contain exactly one centroid means that the solution converged on a local optimum. There is no theoretical guarantee on the quality of the centroids that the K-means algorithm finds, just a guarantee that it will find *K* number of clusters.

To demonstrate a local optimum solution, the K-means clustering algorithm was applied to the 2-dimensional data set shown in Figure 3 that contains five natural clusters, visualized with separate colors. As seen in Figure 4, two centroids split one natural cluster and one centroid controls two naturally separate clusters.



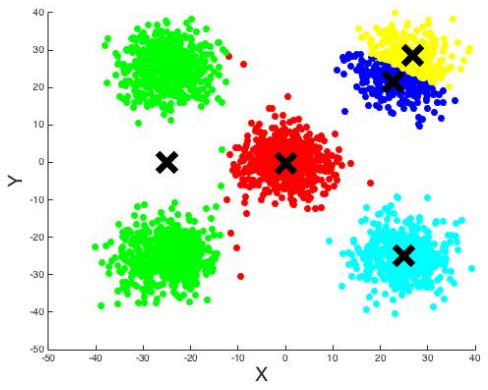


Figure 3: Data With Natural Clusters *Figure 4: Local Optimum Solution*

When the algorithm settles on a local optimum solution, the clusters no longer accurately describe the range of nominal values. Notice in Figure 3 that a natural cluster, shown in green, spans a range of y-values of approximately 10 to 40 and the light blue natural cluster spans y-values of -40 to -10. With the local optimum solution, the light green cluster spans all the y-values between the two clusters, -40 to 40. The local optimum solution has added a span of y-values, -10 to 10, to the solution that is not normally there. The local optimum solution would not serve as a good representation of the nominal value ranges. Upon further research, local optimum convergence was found to be a known issue of the K-means algorithm. There are many extensions to the K-means algorithm that are less prone to the issue of local optimum convergence.

In addition to converging on a local optimum solution, the time K-means algorithm takes to converge is directly related to the size of the data set. So, large data sets will take a long time to converge. For the learning algorithm, larger data sets are ideal to ensure all nominal states are represented in the nominal data set. A larger data set used for the learning algorithm results in more confidence that the centroids accurately represent all the nominal states of the system.

To confirm the well-known issue of the K-means convergence time, the K-means algorithm was implemented in MATLAB and performed on data sets with five natural clusters of varying sizes. The K-means algorithm was performed on each data set 100 times to demonstrate the distribution of convergence times. Figure 5 shows the time the K-means algorithm takes to converge as the size of the data set grows. A full verification analysis of the IMS algorithms was performed and is explained later in Chapter 6.

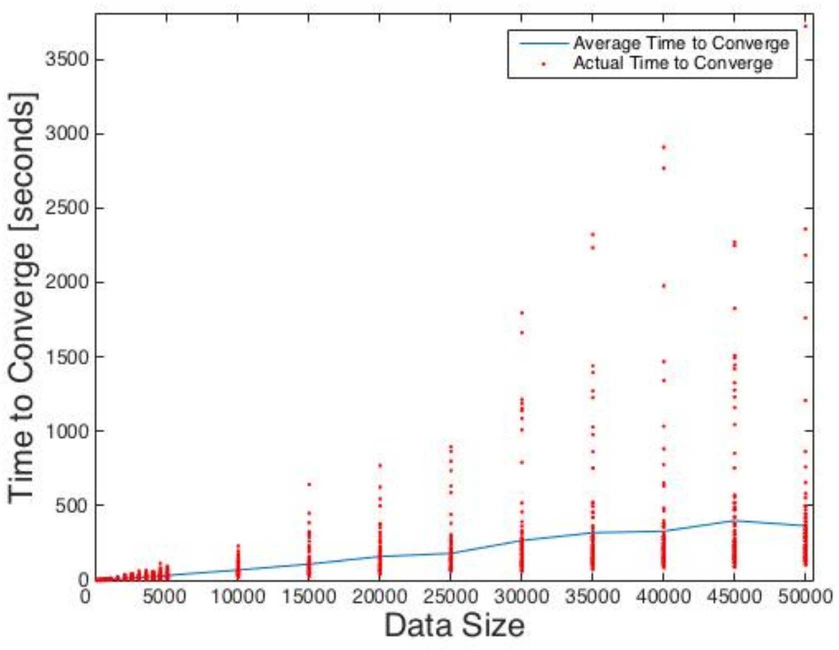


Figure 5: K-Means Time to Converge

As expected, Figure 5 shows that an increase in the data size increases the convergence time as well. To give a rough estimate of the size of the data set, a data set with 50,000 points (the maximum points shown) would come from a nominal test that lasted 13.88 hours and was sampling 2-dimensional data at a rate of 1 Hz. The average time to converge on a data set of that size was about 6 minutes. The worst case took 62.02 minutes to converge. A reduction in this time would add efficiency to the algorithms.

K-means requires the user to understand the data set at-hand and supply the number of clusters. While this may seem like an easy task for the data sets previously laid out in Figure 3, the data sets in Figure 3 were manually created with natural clusters so it could easily be confirmed if they converged to the correct solution. However, in real scenarios, the clusters may not be as easy to distinguish and count.

3.2 USING K-MEANS++ FOR CENTROID INITIALIZATION

K-Means++ is an algorithm that replaces the first step of K-means where the centroids are initialized through random selection. K-means++ chooses the initial centroids for the K-means algorithm to then use instead of having them randomly initialized. Using K-means++ to select the initial centroids decreases the convergence time as well as decreases the probability of converging on local optimum solutions.

The K-means++ algorithm chooses centroids that are far away from each other so they are more apt to converge on separate clusters. The algorithm starts by choosing one center at random from the data set. The distance from each point in the data set to this chosen centroid is calculated. The remaining centroids are chosen based on the following D^2 weighting.

$$\frac{D_i(x)^2}{\sum_{x \in X} D_i(x)^2} \tag{2}$$

where x is the individual state vector and X the set of all the state vectors. Let D_i^2 be defined as

$$D_i{}^2 = \min\left(||x_i - x_{cl}|| \ldots ||x_i - x_{cl}||\right)^2 \tag{3}$$

where the subscript, *cl*, is a vector that denotes all the previously determined centroids. The size of this vector increases as more centroids are chosen [1].

The remaining centroids were chosen using the above weighting scheme and the roulette wheel selection process proposed by Holland [8]. This selection method is derived from the genetic algorithm that stems from Charles' Darwin theory of natural selection. The main premise behind this selection method is that the probability of selection is based upon the fitness value [6]. When applied to this application, the fitness value was defined as the D^2 weighting value.

3.2.1 IMPROVEMENT OF ADDING K-MEANS ++

After the K-means++ algorithm was implemented, a short verification analysis was performed to see if improvements in the algorithm were seen. This analysis only sought to demonstrate the benefits K-means++ added to the system; a full verification analysis of all the IMS software is explained later in Chapter 6.

In order to see the results in a side-side comparison, a data set of 2,500 2-dimensional data points was used and each type of centroid initialization (K-means and K-means++) was performed on the data set. Each initialization method was set to run 1,000 times on the data set and the average of the results from the test are shown in Table 3.

Table 3: K-Means and K-Means++ Comparison

	K-Means	K-Means++
Number of Local Optimum Convergences	130	1
Time to Converge on Global Optimization	14.53 seconds	8.45 seconds

As the table shows, there was an improvement when K-Means++ was used to initialize the centroids. K-means converged on the local optimum 13% of the time and K-means++ reduced this percentage to 0.1%. When examining only the cases where the global optimum was found, the K-means convergence time reduced to almost half when K-Means++ was added. Overall, K-Means++ proved to be a beneficial addition to the learning algorithm.

3.3 GAP STATISTIC

The gap statistic method eliminates the need for the user to determine the number of clusters present in the data. The gap statistic is based on the idea behind clustering which is to maximize the intra-cluster similarity and minimize the inter-cluster similarity.

3.3.1 BACKGROUND

One way to quantitatively define intra-cluster and inter-cluster similarity is with the variance value, W_k. The variance value is the calculation of dispersion within each cluster. To calculate it, let there be a set of data points in which i=1, 2, 3, …n, where n is the number of observations, and j=1, 2, 3, …p, where p is the number of dimensions of each observation. Let $d_{ii'}$ be the squared Euclidean distance between point i and it's assigned centroid i' such that

$$d_{ii'} = \sum_j (x_{ij} - x_{i'j})^2 \tag{4}$$

For a data set that has been grouped into k clusters, where each cluster is defined individually by $C_1, C_3, C_3, \dots C_k$, respectively, let the sum of the distance between the center of a cluster and each data point assigned to the cluster be defined by

$$D_r = \sum_{i,i' \epsilon\, C_k} d_{ii'} \tag{5}$$

The variance quantity is then calculated by adding all of the intra-cluster distances.

$$W_k = \sum_{r=1}^{k} \frac{1}{2n_r} D_r \tag{6}$$

When the optimal number of clusters for a given data set is unknown, the variance of the data set with respect to varying number of clusters can give insight into what the optimal k value is. The best way to view this data is shown in Figure 6.

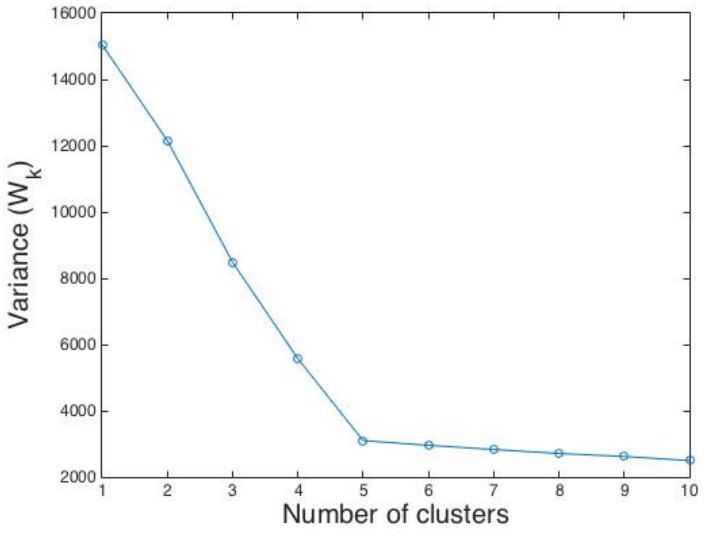


Figure 6: Variance for a Data Set With 5 Natural Clusters

The above figure was created using the same data set shown in Figure 3 which has five natural clusters in it. The data was clustered with *k*=1, 2, 3, …10 using the K-means algorithm previously described. The variance of the clusters for each k value was calculated and the results are shown in Figure 6.

As seen in the graph of Figure 6, the variance decreases when the *k* value increases, meaning clusters that are more tightly packed are being chosen. The ideal *k* value is equal to the number of data points, so that each data point has its own classification. However, this many classifications are not useful in the overall learning algorithm.

The optimal number of clusters is interpreted from the graph. From the trend in the graph, one can see that a natural bend in the graph exists. Before the bend in the graph, the variance decreases rapidly with an increase in the number of clusters. The slope before the bend is steeper than the slope after the bend. At the bend in the graph, referred to as the 'elbow' of the graph, the optimal number of clusters exists. After the 'elbow', the gain of adding more clusters becomes marginal, and thus the point right at the 'elbow' is chosen.

3.3.2 HEURISTIC APPROACH TO DETERMING OPTIMAL K

Using the method previously stated will get you the optimal number of clusters. However, this method requires the user to determine where the 'elbow' in the graph occurs. The previous example had a very distinct 'elbow' and it was fairly easy to distinguish where the 'elbow' occurred. For data sets that do not have as distinct clusters, the 'elbow' point will be less easy to distinguish.

The gap statistic is a heuristic approach to determining where the 'elbow' occurs in the data set and minimizes the user's input. The gap statistic method is a very versatile approach and can be applied to any clustering method because the gap statistic method does not evaluate the actual clustering method, only the results.

The goal of using the gap statistic method developed by Tibshirani, Walther, and Hastie is to "standardize the graph of $log\ (W_k)$ by comparing it with its expectation under an appropriate null reference distribution of the data" [19]. Using this method, the optimal value for k can be estimated as the value that lies furthest from the reference distribution curve. To define the distance from the reference curve, let

$$Gap_n(k) = E_n^*\{\log{(W_k)}\} - \log{(W_k)} \tag{7}$$

where E_n^* , referred to as the expected value, is the average of the reference distributions. The asterisk from here on denotes that accompanying variable is calculated for the reference distribution and not the original data set. By subtracting off the $log\ (W_k)$ from this expected value, we get the distance from the reference curve, $Gap_n(k)$. The value for k is chosen at the maximum $Gap_n(k)$ in order to achieve the value that lies furthest from the reference curve.

Knowing that the optimal *k* is the one such that $log\ (W_k)$ falls furthest from the reference comes from the following. Let there be a data set of *n* uniform data points in *p* dimensions with *K* clusters where the centers have converged equally spaced. The approximate E_n^* of this data set is

$$\log({}^{pn}/_{12}) - ({}^{2}/_{p})\log(k) + constant \tag{8}$$

If the data set actually has *K* natural clusters, the expected rate of decay is similar to $({}^{2}/_{p})\log(K)$ when *p* and *n* are held constant across the data sets. A k value less than the *K* natural clusters is expected to decrease faster. When the k value is greater than the *K* natural clusters, there is an additional centroid in the middle of a natural cluster. The equation shows that the $log\ (k)$ will decrease slower than the expected rate of *K* natural clusters. The optimal k is selected when the gap statistic is largest, which occurs at *k*=*K.*

In order to calculate the expected value of the null reference distribution, the Monte Carlo method was used to create *B* number of data sets. The data sets are restrained to a reference distribution defined by the boundaries of the original data set. The average of $log(W_k^*)$ of each of the *B* copies of data determines the estimated $E_n^*\{log\ (W_k)\}$. The standard deviation of the data sets is calculated and denoted by $sd\ (k)$.

$$sd(k) = \sqrt{\left(\frac{1}{B}\right)\sum_b(\log(W_{kb}^*) - \left(\frac{1}{B}\right) * \sum_b \log(W_{kb}^*))^2} \tag{9}$$

Finally, accounting for the simulation error of $E_n^*\{\log\ (W_k)\}$ along with the standard deviation, let

$$s_k = \sqrt{(1 + {}^{1}/_{B})} * sd(k). \tag{10}$$

In order to choose the optimal cluster size, the smallest k is chosen such that the following is still true [19].

$$Gap(k) \geq Gap(k+1) - s_{k+1} \tag{11}$$

3.3.3 GAP STATISTIC IMPLEMENTATION

The following procedure using the methodology described above was implemented into MATLAB in order to calculate the optimal K for the data set at hand [19]:

1. Cluster the given data with a varying amount of clusters, k=1,2,…n.
2. Calculate the variance, W_k, for each number of clusters.
3. Find the maximum and minimum of each dimension of the data vectors.
4. Create B number of data sets using the Monte Carlo method with the same number of points as the original data set and bounded by the maximum and minimum of the original data set.
5. Cluster the new data sets with the same varying amount of clusters as before: k=1,2,…n.
6. Calculate the variance, W_k^*, for each cluster amount for each new data set.
7. For each k value, calculate $E_n^*\{\log(W_k)\}$ by taking the average of the variances, W_k^*, across the B number of data sets.
8. Calculate the gap statistic for each value of k using Eqn. 7.
9. Calculate the standard deviation for each k value using Eqn. 9 and then apply to Eqn. 10.

10. Choose the smallest *k* value such that Eqn. 11 holds true.

3.4 CONCLUSION

The learning algorithm developed for this specific application is composed of three separate algorithms. The gap statistic algorithm determines the optimal number of clusters needed for a data set. The K-means++ algorithm selects the initial centroids for optimal solutions. Finally, the K-means algorithm groups the data sets into clusters of similar values.

4 MONITORING ALGORITHM

The monitoring algorithm receives telemetry data vectors from the system being monitored and compares them to a knowledge base of nominal clusters derived from the learning algorithm. The monitoring algorithm selects the closest cluster to the telemetry data point and uses this cluster for comparison. The monitoring algorithm then decides if the system is operating nominally or off-nominally by determining if the telemetry data vector is contained within the bounds of the cluster's *N*-dimensional hypercube. Data points within the bounds are classified as nominal and data points outside the bounds are classified as off-nominal. This chapter will describe how the closest cluster is chosen for the monitoring algorithm.

4.1 NEAREST NEIGHBOR SEARCHING

Nearest neighbor searching is a method of finding the closest point in a data set to the query point. There are many different methods of nearest neighbor searching, the most common being the k-d tree and R-tree. However, these algorithms are very complex in design and the complexity grows exponentially with an increase in the dimension of the data being analyzed [17]. As the design gets more complex, the amount of memory needed also grows. Since memory is an issue on-board a CubeSat, a method that utilizes less is ideal. K-d tree and R-tree are suggested for dimensions of 15 or less but not high-dimensional data sets, which are defined as any set greater than 25 dimensions [17]. Since

CubeSats usually have 50 or more sensors, the state vector would be 50 dimensions or more. The method suggested by Sameer Nene and Shree Neer at Columbia University was found to be a better fit for a CubeSat application. The method is detailed in *A Simple Algorithm for Nearest Neighbor Search in High Dimensions*.

4.1.1 A SIMPLE ALGORITHM IN HIGH DIMENSIONS

A brute force way to perform a nearest neighbor search would be to find the Euclidean distance between the query point and every other data point in the set. The minimum Euclidean distance calculated would result in the "nearest neighbor." With a high-dimensional data set, the time to compute all of the Euclidean distances increases. Nene and Neer suggest a solution that minimizes this search area.

To better explain Nene and Neer's method, a 3-dimensional data set will be used as an example, but this method is meant to be scaled to higher dimensions. The purpose of this method is to find the closest point within ε-distance of a given query point, *Q(x,y,z)*. A cube with side lengths of 2ε is formed around the query point to define this search space. The tolerance ε is usually chosen to be relatively small so that a minimal amount of points is contained within the search space. The Euclidean distance between the query point, *Q(x,y,z)*, and all the points that fall within the search space are calculated and the point with the shortest Euclidean distance to *Q(x,y,z)* is deemed the 'nearest neighbor'.

To create the search space in the *N*-dimensional space, a 'search-by-slicing' method is used. First, a slice of one dimension is analyzed by placing two parallel planes in the first dimension that are each ε-distance away from the query point. In the 3-dimensional example the parallel planes would be X_1 and X_2. Any points in the data set that fall between these two planes are added to the "candidate list." The next dimension is then analyzed, which in this example would be the y-dimension. Again, two planes ($Y_1$and Y_2) are placed ε-distance away from the query point in the y-dimension. Instead of looking in the data set for data points that fall between these two parallel planes, the candidate list is now referenced. Any points within the candidate list that do not fall between the two planes, Y_1 and Y_2, are eliminated. The process is then repeated for the remaining dimensions until the candidate list only contains data-points contained within the search space. Because this example was shown in 3-dimensions, the search space results in a cube and a visual representation of this search space produced by Nene and Nayar is shown in Figure 7.

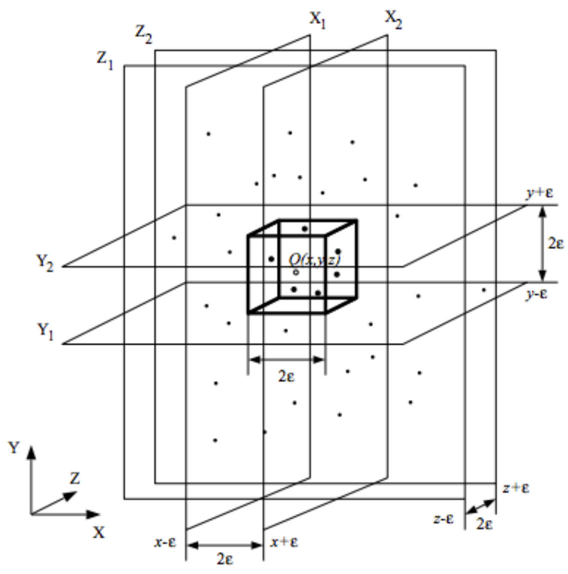


Figure 7: Search Space Around Query Point, Q(x,y,z) [17]

4.2 CALCULATING THE DEVIATION VALUE

To demonstrate how far the system has deviated from the nominal state, the monitoring algorithm calculates a deviation value. The deviation value is a ratio of the distance between the queried state vector, *i*, and its assigned closest centroid, *i'*, and the distance from the closest centroid to the furthest point in the cluster, i_{max}. With state vectors containing *j*-dimensions, the deviation value is

$$Deviation\ Value = \frac{\sum_j \left(x_{ij} - x_{i'j}\right)^2}{\sum_j (x_{i'j} - x_{i_{max}j})^2} \tag{12}$$

4.3 CONCLUSION

The monitoring algorithm determines the health of the system by comparing the current health state vector to the knowledge base of nominal clusters. To make this comparison, the monitoring algorithm employs nearest neighbor searching to find the closest centroid. For this application, the search space is reduced by Nene and Nayar's 'search-by-slicing' method. Once the closest centroid is found, the monitoring algorithm classifies the health of the system by determining if the current state is contained within the bounds defined by the *N*-dimensional hypercube of the closest centroid. Finally, to demonstrate how far the system has deviated from the nominal state, the deviation value is calculated.

5 IMPLEMENTATION

IMS is composed of two distinct algorithms: learning and monitoring. As discussed in Chapters 3 and 4, the learning and monitoring algorithms are broken down further into smaller algorithms that were chosen for the CubeSat application. The individual algorithms were implemented into MATLAB code to form a ground-based prototype. The following chapter will discuss how the chosen algorithms function together, their integration with the CubeSat testing environment, and the user-interface developed for the CubeSat prototype.

5.1 FUNCTIONAL RELATIONSHIP WITHIN IMS

In the two previous chapters, the details of the learning and monitoring algorithms were explained. Individually, none of the algorithms explained are able to monitor the health of a system, but when used in succession the algorithms form an IMS prototype that is able to monitor the health. A visual representation of the relationship between the functions is shown in Figure 8.

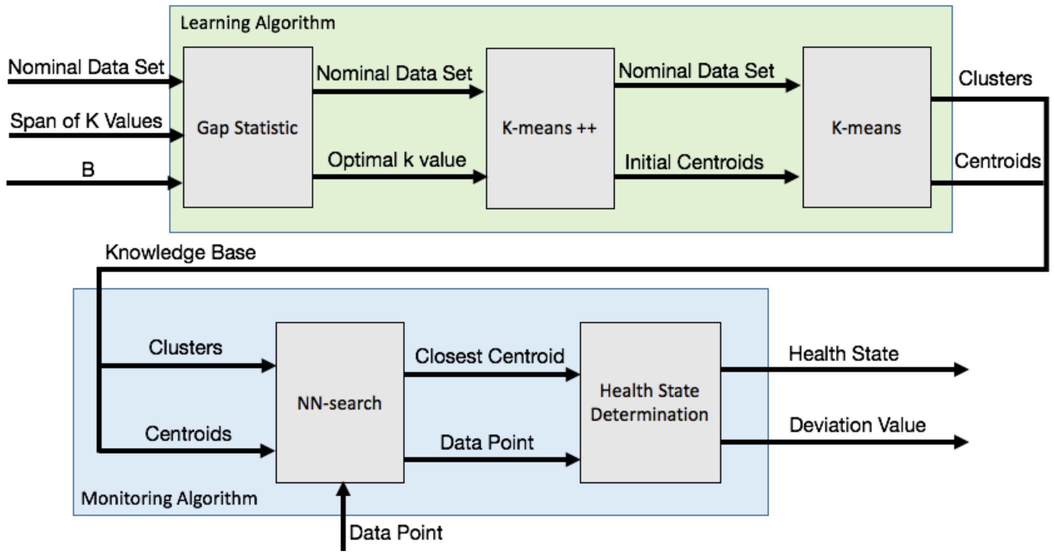


Figure 8: Functional Relationship Within IMS

First, the learning algorithm, shown in green, needs to form the knowledge base of nominal data sets that will later be used for health determination. A data set containing vectors of nominal data will be used to form the knowledge base. The gap statistic algorithm is used first to determine the optimal number of clusters for the data set provided to it. In addition to the nominal data set, the gap statistic algorithm needs a span of *k* values from which the gap statistic algorithm will determine the optimal *k* value. The gap statistic method also needs the number of null reference distributions the algorithm should use for the Monte Carlo data generation. Using these inputs, the gap statistic determines the optimal k value and passes the information, along with the data set, to the K-means++ algorithm. The K-means++ algorithm then selects *k* number of centroids based upon their fitness value, or D^2 weighting. The K-means algorithm uses the selected initial centroids and the nominal data set to build the appropriate knowledge base, which is composed of the clusters and their centroids. The knowledge base is the final outcome of the learning algorithm.

The learning algorithm passes the generated knowledge base to the monitoring algorithm, shown in blue in Figure 8. When a new data point is received, the monitoring algorithm uses the nearest neighbor searching algorithm to find the closest cluster to that data point in the knowledge base. When that cluster is found, the health monitoring algorithm determines the health state of the system by concluding whether or not the new data point is contained within the bounds of its closest cluster. As part of the health determination, the deviation value of the data point is also calculated. The monitoring algorithm finally outputs the state of the data point, nominal or off-nominal, and the deviation value.

5.2 FUNCTIONAL RELATIONSHIP WITH CUBESAT

After the IMS software was developed in MATLAB, the software was adapted to allow for real-time health monitoring within the CubeSat testing environment. A visual representation of the functional relationship between IMS and the CubeSat testing environment is shown below in Figure 9.

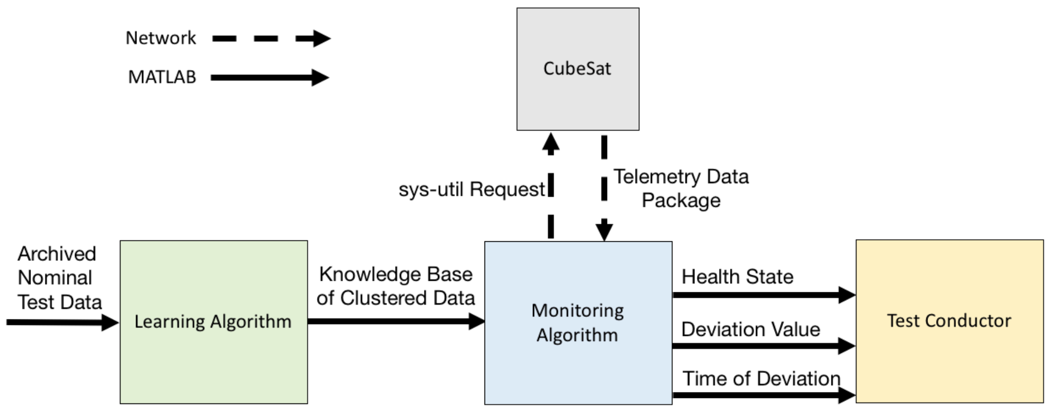


Figure 9: Functional Relationship Between IMS and CubeSat

The learning algorithm uses archived CubeSat test data to form the knowledge base. When the learning algorithm receives the data, the data must be in the format outlined in Table 1. The learning algorithm uses this data to form clusters as described in the previous section. The clusters contained in the knowledge base quantitatively describe the CubeSat's nominal states within the archived test data. This knowledge base is passed through to the monitoring algorithm where the knowledge base is used to assess the system's health state.

In order to monitor the health of the CubeSat, the monitoring algorithm must proactively request the CubeSat's current state and then determine the health of that state. The IMS prototype is executed on a standalone laptop that is connected to the CubeSat's network via an Ethernet cable. The monitoring algorithm requests the state of the CubeSat over this connection using the sys-util program. The sys-util program is an executable c-file that queries the CubeSat and requests the current telemetry data values. When the sys-util command is sent, the CubeSat responds with a data package containing all the telemetry values, housekeeping data, and time of package generation. The health monitoring algorithm parses this data package for the telemetry values and formats them into the data vector format in Table 1. Once the data vector has been formatted, the monitoring algorithm determines the closest cluster in the nominal knowledge base.

In order to allow for continuous use, a slight adjustment to the prescribed nearest neighbor algorithm had to be made to avoid having an empty *N*-dimensional hypercube that would cause an error in the program and interrupt the monitoring. The program has a set ε; however, if the cube is empty after the search is complete, the ε-value is doubled and the program repeats the search with a larger cube. This is done until a nearest neighbor solution is found.

The monitoring algorithm then determines the health state of the CubeSat by deciding if the data vector is contained within the bounds of the closest cluster's *N*-dimensional hypercube. Determining the health state also include calculating the deviation value. Finally, the monitoring algorithm reports the health state conclusion, along with the time and deviation value, to the test conductor.

5.3 USER INTERFACE

A Graphical User Interface (GUI) was created for the developed IMS software which makes the interaction between the user and the IMS software a fluid and seamless process. A user interface eliminates the need to understand the inputs and output formats of the MATLAB functions.

The GUI consists of two panels, one for the learning algorithm and one for the monitoring algorithm, shown in Figure 10 and Figure 11 respectively. The learning algorithm assists the user in clustering the data set by prompting for the necessary inputs, shown in Figure 8. When commanded to cluster, the GUI then performs the appropriate MATLAB functions in the order shown in Figure 8 to achieve the knowledge base. The GUI autonomously passes the new information gained from each function to the next function.

The monitoring algorithm panel serves as a visual representation of the outcome of the monitoring algorithm. The panel contains a view graph of the calculated deviation value over time that continuously updates while IMS is monitoring. The GUI also alerts the user of any off-nominal conditions. For more information on the GUI, a user guide can be found in Appendix C.

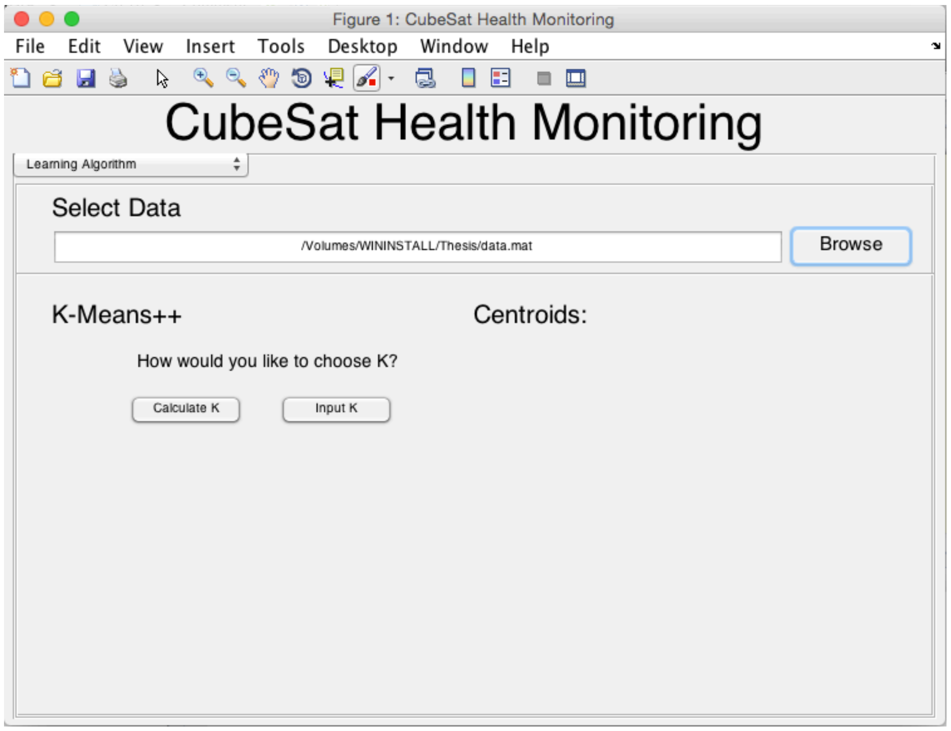


Figure 1: CubeSat Health Monitoring

File Edit View Insert Tools Desktop Window Help

CubeSat Health Monitoring

Learning Algorithm

Select Data

/Volumes/WININSTALL/Thesis/data.mat

Browse

K-Means++

Centroids:

How would you like to choose K?

Calculate K

Input K

Figure 10: Learning Algorithm Panel

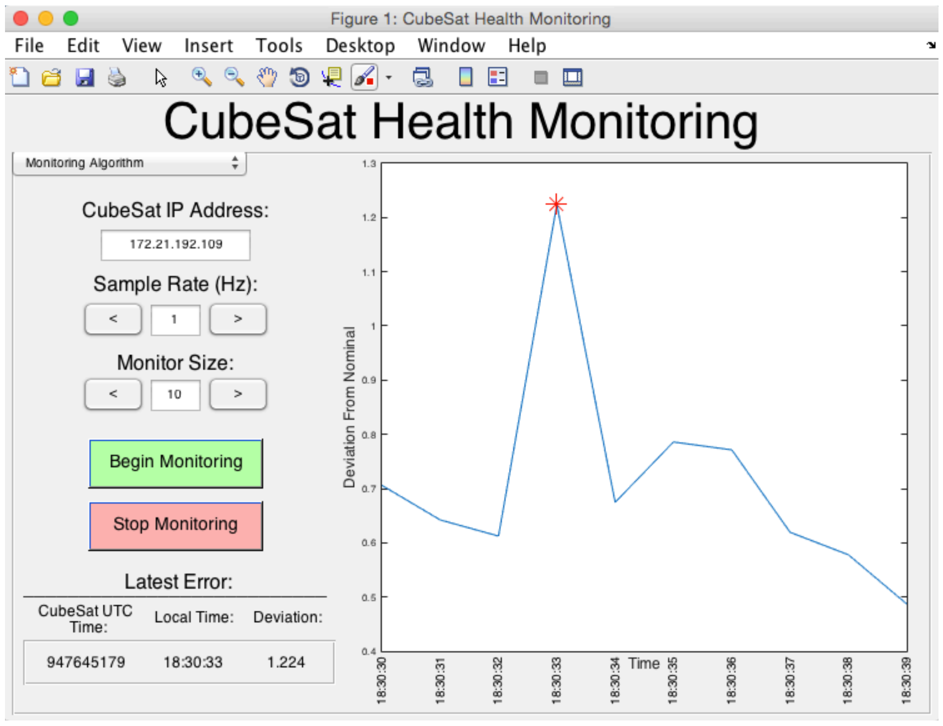


Figure 1: CubeSat Health Monitoring

File Edit View Insert Tools Desktop Window Help

CubeSat Health Monitoring

Monitoring Algorithm

CubeSat IP Address: 172.21.192.109

Sample Rate (Hz): 1

Monitor Size: 10

Begin Monitoring

Stop Monitoring

Latest Error:

CubeSat UTC Time:	Local Time:	Deviation:
947645179	18:30:33	1.224

Figure 11: Monitoring Algorithm Panel

5.4 CONCLUSION

The learning algorithm creates a knowledge base that classifies the nominal states of the CubeSat. The learning algorithm produces this knowledge base by utilizing the gap statistic method to determine the appropriate number of clusters, the K-means++ algorithm to initialize the centers of the clusters, and the K-means algorithm to partition the data set into the appropriate clusters. The knowledge base output is composed of these clusters.

The monitoring algorithm monitors data by utilizing the nearest neighbor searching algorithm to find the closest cluster in the knowledge base. The monitoring algorithm then determines if the telemetry data from the CubeSat falls inside or outside the bounds of the closest cluster, which translates to nominal telemetry data or off-nominal telemetry data.

To integrate the IMS software with the CubeSat testing environment, the monitoring algorithm was modified slightly. The software was altered to actively request data packages from the CubeSat, parse them, and format the data into the appropriate data vector for use in the monitoring algorithm. The IMS software was also altered to output a visual representation of the results to the test conductor. To assist with the interaction between the test conductor and the IMS software, a GUI was created.

6 VERIFICATION

The developed IMS software was verified using two different methods. The first method verified that the specific algorithms chosen for the learning algorithm and monitoring algorithm produced the expected outcome. A unit test of each of the five algorithms described in Section 5.1 (gap statistic, K-means++, K-means, nearest neighbor searching, and health determination) was performed in succession with 2-dimensional data vectors. The second method verified that the developed ground-based prototype could successfully monitor for errors when integrated with the CubeSat testing environment. An acceptance test of the IMS ground-based prototype was performed using 56-dimensional emulated data packages.

6.1 ALGORITHM VERIFICATION

The algorithms themselves were verified with a 2-dimensional data set, which allowed the author to see a visual representation and manually check the progression of the algorithms. First, five clusters containing 500 data points each were created randomly. The five clusters were centered around the following centroids:

Table 4: Centroids Used for Verification

	Dimension 1	Dimension 2
Centroid 1	0.16	-0.14
Centroid 2	24.88	25.03
Centroid 3	24.88	-24.87
Centroid 4	-24.88	24.93
Centroid 5	-25.24	-24.95

The gap statistic algorithm was performed on the data set over a span of K values of 2-8 using 10 copies of the data (*B* =10) for the Monte Carlo data generation. The gap statistic algorithm was run 10 times on the data to ensure the same solution was found each time. A visual representation of the outcome of the algorithm is shown in Figure 12.

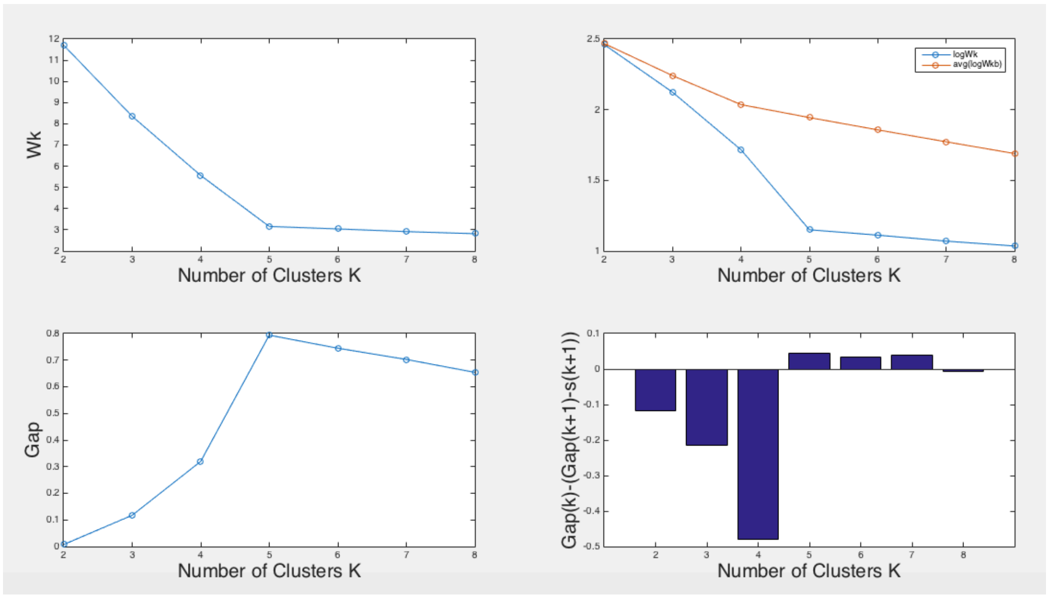


Figure 12: Gap Statistic Results

The algorithm took an average of 44.9 minutes to run and concluded that there were 5 clusters within the data set. The information gained from the graphs above support this conclusion. One can see that there is an 'elbow' at *k*=5 in the W_k vs Number of Clusters graph, which indicates the optimal number of clusters. In the final graph, in the bottom right hand corner, one can see that at *k*=5 the graph flips from negative to positive. The value switching sign is the first indication that $Gap(k) \geq Gap(k+1) - s_{k+1}$. The gap statistic algorithm correctly chose the optimal k value.

The K-Means++ algorithm was then applied to the generated data set to find the initial centroids of the five clusters. An example of how the K-means++ first initialized the centroids is shown in Figure 13.

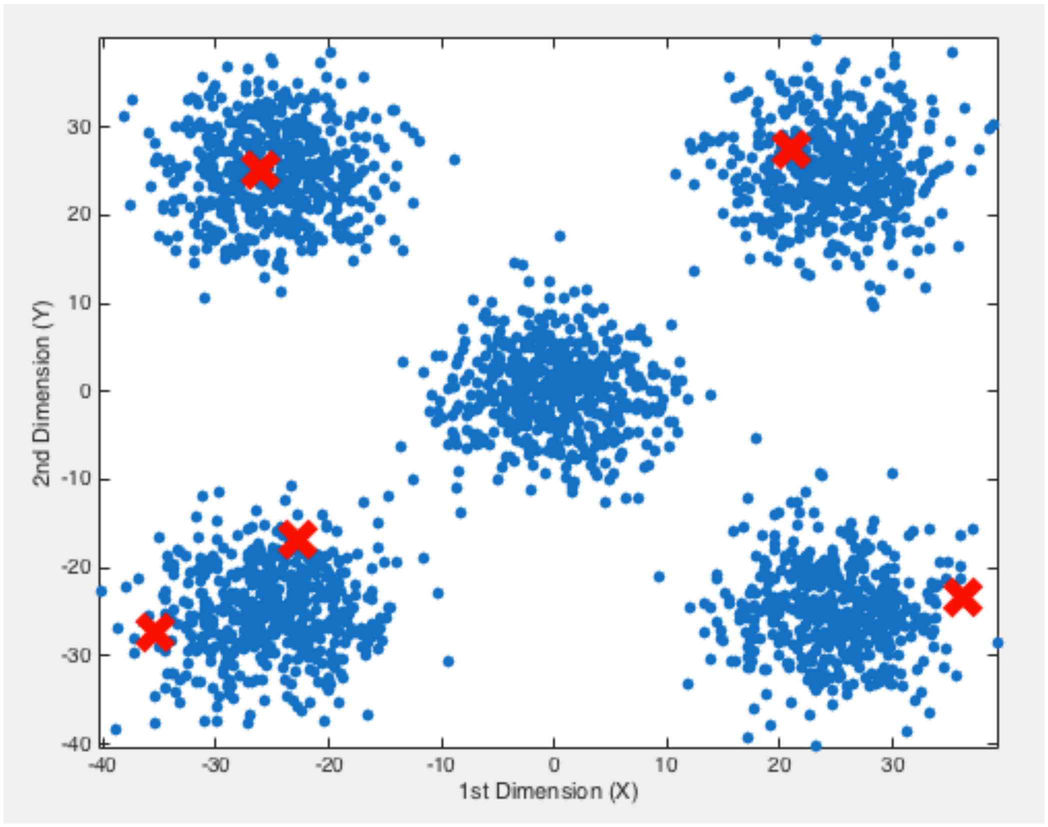


Figure 13: Initial Centroids from K-means++ Algorithm

In the above graph, the red X's represent the centroids that the K-means++ chose for initialization. The K-means algorithm was then applied to the centroids and iterated until the algorithm converged on a solution. The K-means++ algorithm followed by the K-means algorithm was performed 5 times and the solutions are shown in the table below:

Table 5: Centroids Found by K-Means++ and K-Means

	Solution 1		Solution 2		Solution 3	
	X	Y	X	Y	X	Y
Centroid 1	0.16	-0.14	0.16	-0.14	0.16	-0.14
Centroid 2	24.88	25.03	24.88	25.03	24.88	25.03
Centroid 3	24.88	-24.87	24.88	-24.87	24.88	-24.87
Centroid 4	-24.88	24.93	-24.88	24.93	-24.88	24.93
Centroid 5	-25.24	-24.95	-25.24	-24.95	-25.24	-24.95

	Solution 4		Solution 5	
	X	Y	X	Y
Centroid 1	0.16	-0.14	0.16	-0.14
Centroid 2	24.88	25.03	24.88	25.03
Centroid 3	24.88	-24.87	24.88	-24.87
Centroid 4	-24.88	24.93	-24.88	24.93
Centroid 5	-25.24	-24.95	-25.24	-24.95

The table shows that the K-means++ and K-means algorithms consistently found the accurate values for the centroids. Once the centroids were found, the monitoring algorithm was tested. A data set with 500 2-dimensional data points was created for monitoring. Of the 500 data points, 480 fell within the clusters and 20 fell outside the clusters. The 20 that fell outside the clusters are listed in Table 6.

Table 6: Off-Nominal Data Points Used for Verification

X	Y	X	Y	X	Y	X	Y	X	Y
-30	0	-60	30	30	4	0	-35	-5	-40
43	0	8	22	-28	-55	-50	38	40	40
20	0	60	60	-40	-40	45	46	32	6
0	29	8	49	-45	60	23	73	4	89

The monitoring algorithm which employed the nearest neighbor searching method checked all 500 points and returned how many points were found that fell outside the bounds of the clusters. Each time the monitoring program was executed, 10 times in total, exactly 480 points were found that fell within the clusters and 20 that fell outside the clusters. The monitoring algorithm successfully discovered all off-nominal data points.

6.2 PROTOTYPE VERIFICATION

Once the algorithms were verified, the ground-based prototype needed to be verified to ensure compatibility with the CubeSat testing environment and the CubeSat's response. However, due to the fact that there was not a working CubeSat available at the time of verification, this became a two-step process.

First, the interaction between MATLAB and the CubeSat needed to be verified because the CubeSat had never been commanded by MATLAB before. The sys-util package was installed on the laptop that the prototype would be running on and the laptop was connected via Ethernet cable to the CubeSat network. Once the program was installed, the commands that query the CubeSat were sent from MATLAB to the computer. From there, the commands were autonomously executed and sent across the network to the specific IP address assigned to the CubeSat. The CubeSat responded as expected each time the command was sent and returned a data package containing the status of the CubeSat, which was composed of the telemetry of all the sensors on board the CubeSat, 56 in total. Although the CubeSat responded with the expected data package, the telemetry values all read zero. This was due to an internal problem within the CubeSat and was expected. An example of the CubeSat's null response can be seen in Appendix A. MATLAB's ability to query the spacecraft and receive the expected package was verified.

Next, the performance of the prototype itself needed to be verified. Because the CubeSat was only returning null telemetry values, the testing had to be done on emulated data packages. First, data vectors, formatted the same as in Table 1, were created that represented a nominal test. Since there are 56 sensors on board the CubeSat, a 56-dimensional vector was created for each nominal data point. Each scalar in the vector represented the response from an individual sensor. The individual response from each sensor was chosen at random within the sensor's range of nominal values. In order to have multiple nominal states of the CubeSat, the ranges for the sensors were varied 6 times. In total, 1200 random nominal data vectors were created. This meant that there were 200 data points for each nominal state of the CubeSat. The range of values for each sensor within the six nominal states created for this testing can be seen in Appendix B.

The prototype imported the nominal data set and determined how many clusters were ideal. The optimal number of clusters was found to be six, which is what was expected due to the six separate nominal states generated. The data was then clustered into six clusters of 56-dimensional data vectors.

Once the clustering algorithm had converged on an appropriate solution, those clusters and centroids needed to be tested to determine if the system could detect an anomaly occurring in the system based on the nominal knowledge base. An anomaly occurring in the system would be represented by a sensor value occurring outside of its nominal value range.

In MATLAB, 500 nominal data packages were composed that emulated the CubeSat's response to the sys-util command. The data packages included the telemetry data, the housekeeping data, and the time. Of the 500 data packages, 10 of them were replaced by manually created data packages containing off-nominal values. All of the data packages were then placed into a cell structure. Instead of sending the sys-util command to the CubeSat and expecting a response, the software called the first data package from the cell. The next time a data package was requested, the second data package from the cell was called, and so on. MATLAB received the data package in the same format as the direct response from the CubeSat: a string of letters and numbers. The IMS prototype then parsed the emulated data package in the same manner that the CubeSat's response would be parsed.

The IMS prototype was prompted to begin monitoring with a sample rate of 1 Hz and view size of 10 data points. The data was monitored for 8.43 minutes and the IMS prototype concluded that there were 20 errors in the data monitored, which was expected. The IMS prototype successfully monitored the data and detected all errors that had been induced into the emulated data.

6.3 CONCLUSION

The ground-based prototype of IMS described in this paper was verified in two separate steps. The developed IMS algorithms' functionalities were verified using a manually-created 2-dimensional data set with five natural clusters. The prototype developed for the CubeSat testing environment was verified using emulated 56-dimensional data packages. In both cases of verification, the algorithms were able to detect all injected faults.

7 CONCLUSION

Inductive Monitoring Systems (IMS) have great potential in the aerospace industry. When completely integrated, they automate the process of classifying the healthy states of a system and anomaly detection.

Before IMS can be completely integrated into large flight systems, they must demonstrate more flight heritage and CubeSats pose as a very well-developed test bed to do so. There are two steps to integrating an IMS program into a CubeSat. The first of which is to prove that the program works as expected on the ground and the second being that the program functions properly while in flight. This thesis created a program that completed the first of the two steps: a prototype that monitors the CubeSat during on-ground testing.

This prototype was a program developed in MATLAB that contained two algorithms: learning and monitoring. The learning algorithm creates data vectors that classify the nominal states of the CubeSat through data mining techniques. This algorithm utilizes the gap statistic method to determine the appropriate number of clusters, the K-means++ algorithm to initialize the centers of the clusters, and the K-means algorithm to partition the data set into the appropriate clusters. The monitoring algorithm monitors data received directly from the CubeSat in real time by utilizing the nearest neighbor searching algorithm to find the closest cluster. The monitoring algorithm then determines if the telemetry data from the CubeSat falls inside or outside the bounds of the closest cluster, which translates to nominal telemetry data or off-nominal telemetry data respectively.

The IMS CubeSat prototype was tested through two verification techniques, algorithm verification using a 2-dimensional system and GUI verification using 56-dimensional data packages. The prototype successfully found all of the off-nominal data points that were induced into the system. The first of the two steps to fully integrating IMS into CubeSat's software architecture is complete.

7.1 FUTURE WORK

The work laid out here is the first step in the IMS CubeSat implementation. To attain flight heritage of an entire system IMS needs to be flown on-board the spacecraft during flight.

Due to the bad timing of the development of this software, a functioning CubeSat was not available. The next step in developing the IMS prototype is to confirm the success of the verification process performed in this thesis with a telemetry data set from a functioning CubeSat, i.e. IPEX.

The next step to developing IMS for CubeSats would be to run the algorithm regularly during CubeSat testing. In the beginning, it is important that the system not be relied on for the anomaly detection. The CubeSat test conductors should perform their usual procedure for anomaly detection while IMS is running and confirm that IMS detects the same results as they do, possibly even more.

Once the results of tests performed for this thesis have been confirmed with real, active tests, the next step in the development process would be to implement IMS as flight software. The software developer should have a rough estimate of how large the software package will be based on the code written for this thesis. All of the code used for the IMS ground-based prototype was written specifically for the prototype and no built-in MATLAB functions were used.

The final aspect of giving IMS flight heritage is to fly the IMS software on-board an active spacecraft to analyze the performance during flight.

BIBLIOGRAPHY

[1] Arthur, David, and Sergei Vassilvitskii. "K-means++: The Advantages of Careful Seeding." *Journal of the Franklin Institute* 59.1 (1855): 68-70. *Ilpubs*. Standford. Web. 30 Aug. 2015.

[2] Beato-Day, Pam. "Inductive Monitoring System." (n.d.): n. pag. *Technology Opportunity.* NASA. Web. 11 Nov. 2015.

[3] "Clustering - K-means." *Clustering - K-means*. N.p., n.d. Web. 4 Sept. 2015.

[4] Definition Of Technology Readiness Levels." (2009): n. pag. *Earth Science Technology Office*. NASA. Web. 1 Dec. 2015.

[5] "Finding the K in K-Means Clustering." *The Data Science Lab*. N.p., 27 Dec. 2013. Web. 8 Sept. 2015.

[6] Gen, Mitsuo, and Runwei Cheng. *Genetic Algorithms and Engineering Design*. New York: Wiley, 1997.

[7] Halim, Enayet, Harigopal Raghavan, and Sarish Shah. "Application of Inductive Monitoring System for Equipment Condition Monitoring Automation in Mining, Mineral and Metal Processing." *2009 IFAC Workshop on Automation in Mining, Mineral and Metal Industry* (n.d.): 324-29. Web.

[8] Holland, John *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* Cambridge, Mass.: MIT Press, 1992.

[9] Iverson, David “Inductive System Health Monitoring”. In Proceedings of The 2004 International Conference on Artificial Intelligence (IC-AI04), Las Vegas, Nevada, June 2004. CSREA Press

[10] Iverson, David, Rodney Martin, Mark Schwabacher, Liljana Spirkovska, William Taylor, Ryan Mackey, and J.patrick Castle. "General Purpose Data-Driven System Monitoring for Space Operations." *AIAA Infotech@Aerospace Conference* (2009): n. pag. Web.

[11] James, M., Mackey, R., Park, H., & Zak, M. *BEAM: Technology for Autonomous Self Analysis*, IEEE Aerospace Conference, March 2001.

[12] Johnson, Stephen B. *System Health Management: With Aerospace Applications*. Hoboken, NJ: Wiley, 2011. Print.

[13] "Machine Learning Technique for Finding Hidden Patterns or Intrinsic Structures in Data." *Unsupervised Learning*. Mathworks, n.d. Web. 8 Nov. 2015.

[14] Mackey, Ryan, David Iverson, Greg Pisanich, Mike Toberman, and Ken Hicks. "Integrated System Health Management (ISHM) Technology Demonstration Project Final Report." (2006): n. pag. Web.

[15] MacQueen, J. "Some Methods For Classification and Analysis of Multivariate Observations." University of California, Los Angeles, n.d. Web. 03 Nov. 2015.

[16] Martin, Rodney A., Mark A. Schwabacher, and Bryan L. Matthews. *Data-Driven Anomaly Detection Performance for the Ares I-X Ground Diagnostic Prototype*. Rep. N.p.: Annual Conference of the Prognostics and Health Management Society, 2010. Web.

[17] Nene, Sameer A., and Shree K. Nayar. *A Simple Algorithm for Nearest Neighbor Search in High Dimension*. Tech. no. CUCS-030-95. N.p.: n.p., n.d. Oct 1995. Web. 12 Nov 2015

[18] Teubert, Christopher, and Scott Poll. *Application of Inductive Monitoring System to Plug Load Anomaly Detection* (2012): n. pag. Web.

[19] Tibshirani, Robert, Guenther Walther, and Trevor Hastie. "Estimating the Number of Clusters in a Data Set via the Gap Statistic." *J.R. Statist. Soc. B* (n.d.): 411-23. Nov. 200. Web. 8 Sept. 2015.

[20] "Vector Quantization." *Data Compression*. N.p., 2000. Web. 03 Nov. 2015.

A: CUBESAT RESPONSE TO SYS-UTIL

```
sys-util: getting status...
daughter_aTmpSensor  temp:        0.000000 C
-
daughter_bTmpSensor  temp:        0.000000 C
-
threeV_plTmpSensor   temp:        0.000000 C
-
rf_ampTmpSensor      temp:        0.000000 C
-
tempNz               temp:        0.000000 C
-
tempPz               temp:        0.000000 C
-
tempNx               temp:        0.000000 C
-
tempPx               temp:        0.000000 C
-
tempNy               temp:        0.000000 C
-
tempPy               temp:        0.000000 C
-
atmelPwrSensor       volt:        0.000000 V
atmelPwrSensor       current:     0.000000 A
-
threeVPwrSensor      volt:        0.000000 V
threeVPwrSensor      current:     0.000000 A
-
threeV_plPwrSensor   volt:        0.000000 V
threeV_plPwrSensor   current:     0.000000 A
-
fiveV_plPwrSensor    volt:        0.000000 V
fiveV_plPwrSensor    current:     0.000000 A
-
daughter_aPwrSensor  volt:        0.000000 V
daughter_aPwrSensor  current:     0.000000 A
-
daughter_bPwrSensor  volt:        0.000000 V
daughter_bPwrSensor  current:     0.000000 A
-
fuelGaugeOne         volt:        0.000000 V
fuelGaugeOne         current:     0.000000 A
fuelGaugeOne         currentAccum:    0.000000 A
-
fuelGaugeTwo         volt:        0.000000 V
fuelGaugeTwo         current:     0.000000 A
fuelGaugeTwo         currentAccum:    0.000000 A
-
sidePanel3v3         volt:        0.000000 V
sidePanel3v3         current:     0.000000 A
-
sidePanel5v0         volt:        0.000000 V
sidePanel5v0         current:     0.000000 A
-
solar2PwrNz          volt:        0.000000 V
solar2PwrNz          current:     0.000000 A
-
solar2PwrPz          volt:        0.000000 V
solar2PwrPz          current:     0.000000 A
-
solar2PwrNx          volt:        0.000000 V
solar2PwrNx          current:     0.000000 A
-
solar2PwrPx          volt:        0.000000 V
solar2PwrPx          current:     0.000000 A
-
solar2PwrNy          volt:        0.000000 V
solar2PwrNy          current:     0.000000 A
-
solar2PwrPy          volt:        0.000000 V
solar2PwrPy          current:     0.000000 A
-
solar1PwrNz          volt:        0.000000 V
solar1PwrNz          current:     0.000000 A
-
solar1PwrPz          volt:        0.000000 V
solar1PwrPz          current:     0.000000 A
-
solar1PwrNx          volt:        0.000000 V
solar1PwrNx          current:     0.000000 A
-
solar1PwrPx          volt:        0.000000 V
solar1PwrPx          current:     0.000000 A
-
solar1PwrNy          volt:        0.000000 V
solar1PwrNy          current:     0.000000 A
-
solar1PwrPy          volt:        0.000000 V
solar1PwrPy          current:     0.000000 A
-
Curr_Accum: 0
Max_Accum: 511180
UTC Epoch time: 946689462
UTC time: 1:17:42  1/1/2000
usage_dString=0
```

```
usage_dUInt=0
userTime=4231
niceTime=0
sysTime=1266
idleTime=6025
pageIn=121
pageOut=0
swapIn=90
swapOut=0
interrupts=1184053
context_swaps=180031
boottime=946689344
processes=737
procs_running=1
procs_blocked=0
memFree=49012
buffers=1004
cached=59156
active=10468
inactive=54168
vmallocTotal=899072
vmallocUsed=271856
freeDataFlash=184169 Kb
freeSD=605748 Kb
LDC=0
```

B: NOMINAL CUBESAT RANGES

Name	Parameter Value
'daughter_aTmpSensor [C]'	1
'daughter_bTmpSensor [C]'	2
'threeV_plTmpSensor [C]'	3
'rf_ampTmpSensor [C]'	4
'tempNz [C]'	5
'tempPz [C]'	6
'tempNx [C]'	7
'tempPx [C]'	8
'tempNy [C]'	9
'tempPy [C]'	10
'atmelPwrSensor [V]'	11
'atmelPwrSensor [A]'	12
'threeVPwrSensor [V]'	13
'threeVPwrSensor [A]'	14
'threeV_plPwrSensor [V]'	15
'threeV_plPwrSensor [A]'	16
'fiveV_plPwrSensor [V]'	17
'fiveV_plPwrSensor [A]'	18
'daughter_aPwrSensor [V]'	19
'daughter_aPwrSensor [A]'	20
'daughter_bPwrSensor [V]'	21
'daughter_bPwrSensor [A]'	22
'fuelGaugeOne [V]'	23
'fuelGaugeOne [A]'	24
'fuelGaugeOne [A]'	25
'fuelGaugeTwo [V]'	26
'fuelGaugeTwo [A]'	27
'fuelGaugeTwo [A]'	28
'sidePanel3v3 [V]'	29
'sidePanel3v3 [A]'	30
'sidePanel5v0 [V]'	31
'sidePanel5v0 [A]'	32
'solar2PwrNz [V]'	33
'solar2PwrNz [A]'	34
'solar2PwrPz [V]'	35
'solar2PwrPz [A]'	36
'solar2PwrNx [V]'	37
'solar2PwrNx [A]'	38
'solar2PwrPx [V]'	39
'solar2PwrPx [A]'	40

'solar2PwrNy [V]'	41
'solar2PwrNy [A]'	42
'solar2PwrPy [V]'	43
'solar2PwrPy [A]'	44
'solar1PwrNz [V]'	45
'solar1PwrNz [A]'	46
'solar1PwrPz [V]'	47
'solar1PwrPz [A]'	48
'solar1PwrNx [V]'	49
'solar1PwrNx [A]'	50
'solar1PwrPx [V]'	51
'solar1PwrPx [A]'	52
'solar1PwrNy [V]'	53
'solar1PwrNy [A]'	54
'solar1PwrPy [V]'	55
'solar1PwrPy [A]'	56
'Curr_Accum'	57
'Max_Accum'	58
'UTC Epoch time'	59

Param Value	State 1		State 2		State 3		State 4		State 5		State 6	
	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max
1	26.0	28.0	26.0	28.0	26.0	28.0	26.0	28.0	26.0	28.0	31.0	33.0
2	26.0	28.0	26.0	28.0	26.0	28.0	26.0	28.0	26.0	28.0	31.0	33.0
3	26.0	28.0	26.0	28.0	26.0	28.0	26.0	28.0	26.0	28.0	31.0	33.0
4	27.0	29.0	27.0	29.0	27.0	29.0	27.0	29.0	27.0	29.0	27.0	29.0
5	23.0	29.0	23.0	29.0	26.0	29.0	23.0	29.0	23.0	26.1	23.0	29.0
6	23.0	29.0	23.0	29.0	23.0	29.0	23.0	29.0	23.0	29.0	23.0	29.0
7	23.0	29.0	23.0	29.0	18.0	22.0	23.0	29.0	18.0	22.0	23.0	29.0
8	24.8	29.0	24.8	29.0	18.0	22.0	23.0	27.3	18.0	22.0	23.0	29.0
9	23.0	29.0	23.0	29.0	18.0	22.0	23.0	29.0	18.0	22.0	23.0	29.0
10	23.0	29.0	23.0	29.0	18.0	22.0	23.0	29.0	18.0	22.0	23.0	29.0
11	3.80	4.19	3.80	4.19	3.80	4.19	3.80	4.19	3.80	4.19	3.80	4.19
12	0.01	0.50	0.01	0.50	0.01	0.11	0.01	0.50	0.01	0.11	0.01	0.11
13	2.80	3.20	2.80	3.20	2.80	3.20	2.80	3.20	2.80	3.20	2.80	3.20
14	0.01	0.50	0.01	0.50	0.01	0.11	0.01	0.50	0.01	0.11	0.01	0.11
15	1.11	1.80	1.11	1.80	1.11	1.80	1.11	1.80	1.11	1.80	1.11	1.80
16	0.01	0.50	0.01	0.50	0.01	0.50	0.01	0.50	0.01	0.11	0.01	0.11
17	0.01	0.11	0.01	0.11	0.01	0.11	0.01	0.11	0.01	0.11	0.01	0.11
18	0.01	0.50	0.01	0.50	0.01	0.11	0.01	0.50	0.01	0.11	0.01	0.11
19	0.70	0.80	0.70	0.80	0.70	0.80	0.70	0.80	0.70	0.80	0.70	0.80
20	0.01	0.11	0.01	0.11	0.01	0.11	0.01	0.11	0.01	0.11	0.01	0.11
21	0.20	0.30	0.20	0.30	0.20	0.30	0.20	0.30	0.20	0.30	0.20	0.30
22	0.01	0.11	0.01	0.11	0.01	0.11	0.01	0.11	0.01	0.11	0.01	0.11
23	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50
24	0.01	0.11	0.01	0.11	0.01	0.11	0.01	0.11	0.01	0.11	0.01	0.11
25	6.50	7.50	6.50	7.50	6.50	7.50	6.50	7.50	6.50	7.50	6.50	7.50

26	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50
27	0.01	0.11	0.01	0.11	0.01	0.11	0.01	0.11	0.01	0.11	0.01	0.11
28	6.50	7.50	6.50	7.50	6.50	7.50	6.50	7.50	6.50	7.50	6.50	7.50
29	2.75	3.25	2.75	3.25	2.75	3.25	2.75	3.25	2.75	3.25	2.75	3.25
30	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20
31	2.75	3.25	2.75	3.25	2.75	3.25	2.75	3.25	2.75	3.25	2.75	3.25
32	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20
33	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50
34	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20
35	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50
36	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20
37	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50
38	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20
39	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50
40	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20
41	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50
42	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20
43	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50
44	0.20	1.20	0.20	1.20	0.00	0.20	0.20	1.20	0.00	0.20	0.00	0.20
45	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50
46	0.20	1.20	0.20	1.20	0.00	0.20	0.20	1.20	0.00	0.20	0.00	0.20
47	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50
48	0.20	1.20	0.20	1.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20
49	3.50	4.50	3.50	4.50	4.50	5.50	3.50	4.50	3.50	4.50	3.50	4.50
50	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20
51	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50
52	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20
53	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50
54	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20
55	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50	3.50	4.50
56	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20	0.00	0.20

C: USER GUIDE FOR THE GROUND-BASED PROTOTYPE

The IMS algorithms explained above were implemented into MATLAB and a graphical user interface (GUI) was created for ease of use.

When the script, CubeSatHealthMonitoring.m, is executed, the screen shown in Figure 14 appears.

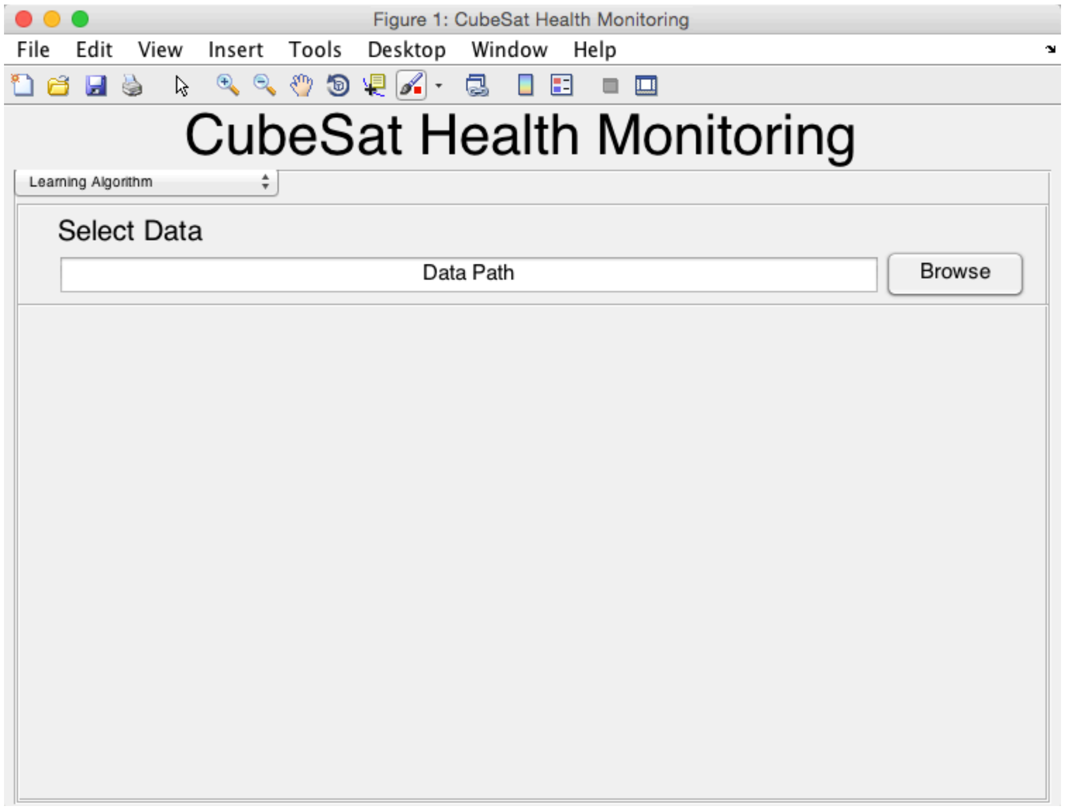


Figure 1: CubeSat Health Monitoring
File Edit View Insert Tools Desktop Window Help
CubeSat Health Monitoring
Learning Algorithm
Select Data
Data Path
Browse

Figure 14: Initialization Screen of CubeSat Health Monitoring GUI

This screen is the beginning of the learning algorithm. First, a nominal data set must be selected and can be done in two ways. If known, the data path can be manually entered in the edit box where it currently says ‘Data Path’. The other option is to click the ‘Browse’ button, which brings up the directory to the computer in which the IMS program is being run. Here, the user can click through the computer's directory in order to find the nominal data file. Once the user finds and selects the data file, the program uses the path to the file and imports the data from that location to the current workspace. The file selected must have been previously formatted to match the data vector presented in Figure 2.

The data needs to be grouped into clusters after being selected and imported. There are two options for doing this and once the data has been selected the GUI allows for the choice to be made.

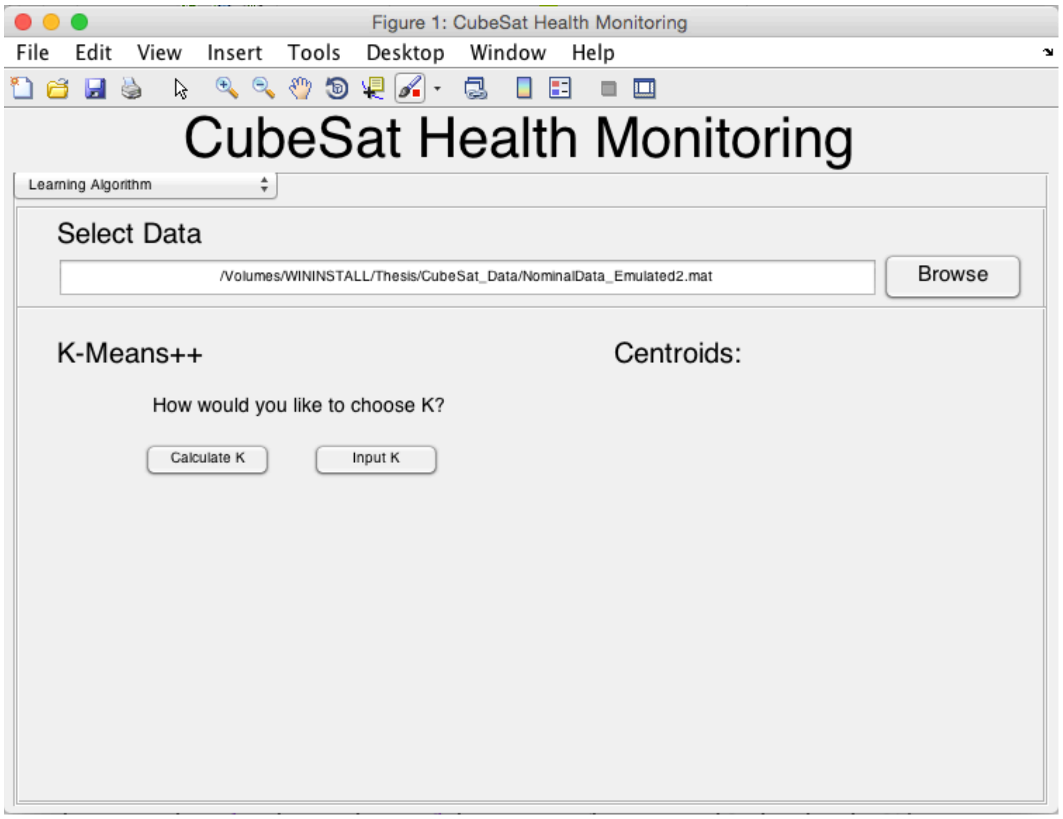


Figure 1: CubeSat Health Monitoring
File Edit View Insert Tools Desktop Window Help
CubeSat Health Monitoring
Learning Algorithm
Select Data
/Volumes/WININSTALL/Thesis/CubeSat_Data/NominalData_Emulated2.mat
Browse
K-Means++
Centroids:
How would you like to choose K?
Calculate K
Input K

Figure 15: Clustering Options

The two options for clustering depend on the knowledge of data that the user has. If the user knows from prior knowledge how many clusters there are, then the user would choose to ‘Input K’. If the user has no prior knowledge of the dataset being imported, then they would choose to have K calculated by clicking the ‘Calculate K’ button.

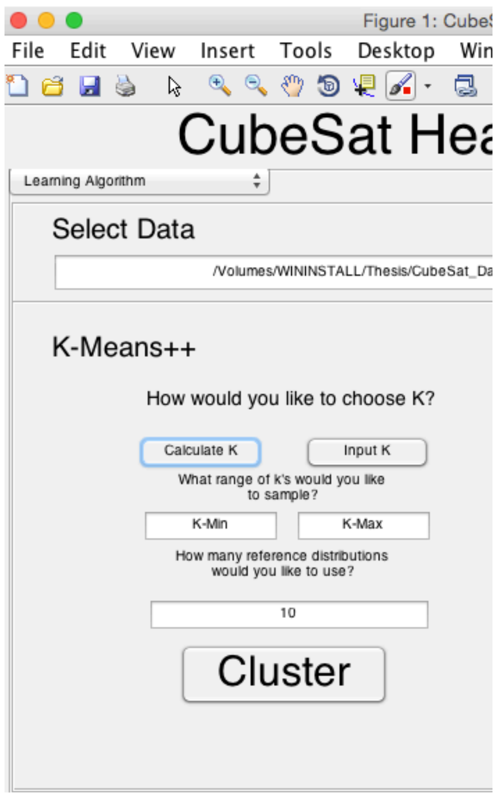


Figure 1: CubeS
File Edit View Insert Tools Desktop Win
CubeSat Hea
Learning Algorithm
Select Data
/Volumes/WININSTALL/Thesis/CubeSat_Da
K-Means++
How would you like to choose K?
Calculate K
Input K
What range of k's would you like to sample?
K-Min
K-Max
How many reference distributions would you like to use?
10
Cluster

Figure 16: Response to 'Calculate K'

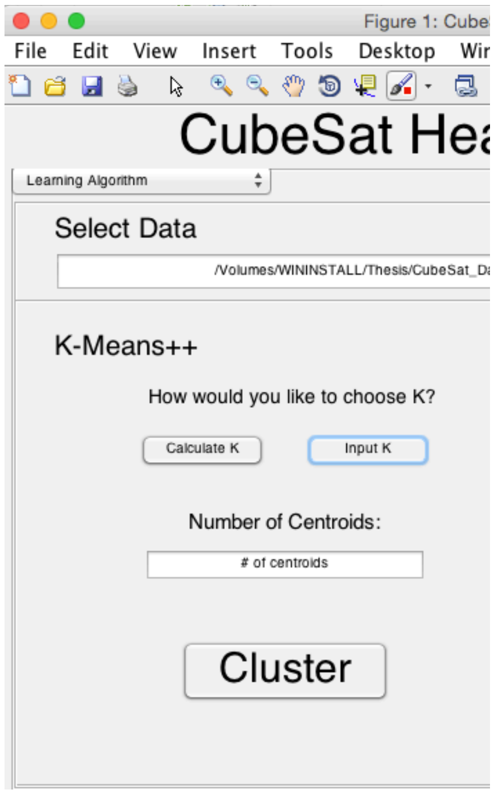


Figure 17: Response to 'Input K'

Both of the ‘Cluster’ buttons shown in the Figure 16 and 17 have the same end goal, to group the imported data into clusters and build the nominal knowledge base. They both use the K-means++ and K-means algorithm to obtain the centroids. However, the difference between the two buttons is that the ‘Cluster’ for ‘Calculate K’ uses the gap statistic algorithm to find the optimal K value to use. While the choice may be obvious to always let the software decide on K, the option to input K is available because the gap statistic takes a substantial amount of extra time to run. If the user does know the appropriate K value, the program allows for the option to skip the unnecessary step.

When the program has converged on a solution to the K-means algorithm, the program demonstrates this by displaying the centroids to the right, shown in Figure 18. The centroids are shown to indicate that the algorithm is complete and the program is ready to move on. The user can view them at this point to better understand how the system is quantitatively describing the nominal states of the CubeSat. If the centroids are longer than the 2-dimensions shown in Figure 18, the user can scroll to see all the dimensions.

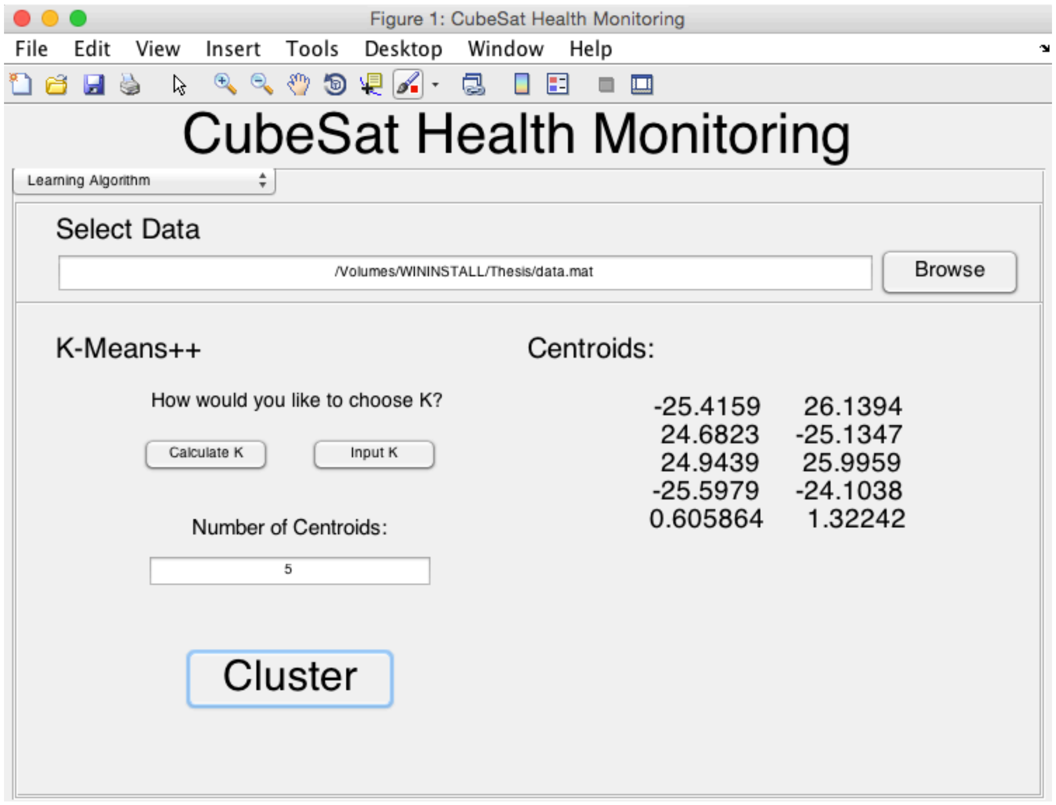


Figure 1: CubeSat Health Monitoring
File Edit View Insert Tools Desktop Window Help
CubeSat Health Monitoring
Learning Algorithm
Select Data
Browse
K-Means++
How would you like to choose K?
Calculate K
Input K
Number of Centroids:
Cluster
Centroids:

/Volumes/WININSTALL/Thesis/data.mat

Number of Centroids: 5

```
-25.4159    26.1394
 24.6823   -25.1347
 24.9439    25.9959
-25.5979   -24.1038
0.605864    1.32242
```

Figure 18: Response to Clustering Being Complete

Once the centroids have been determined for the nominal data set, the program stores those centroid values for later use in the monitoring algorithm.

In order to start the monitoring algorithm, the user must first switch to the appropriate screen via the drop down list at the upper left of the window, which currently says 'Learning Algorithm' in Figure 13. By selecting 'Monitoring Algorithm' from the drop down list, the program switches the GUI to the screen shown in Figure 19.

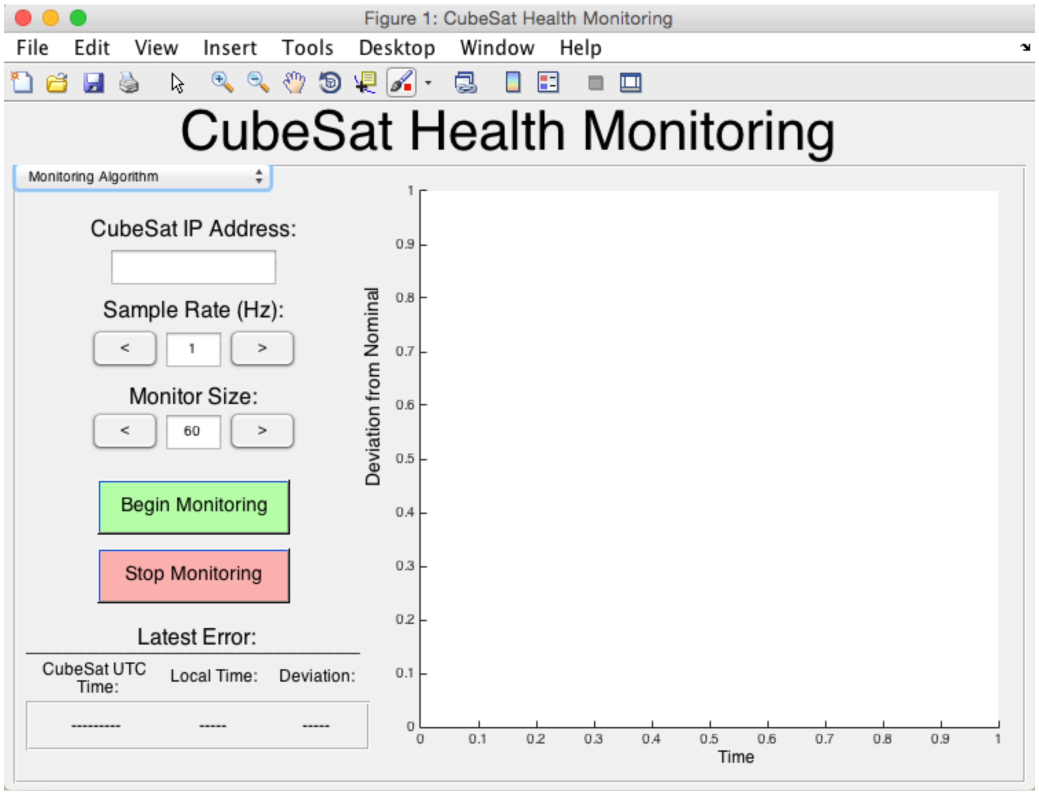


Figure 19: Monitoring Algorithm

The screen in Figure 19 is the gateway into the health monitoring algorithm. First, the user inputs the IP address that is associated with the CubeSat that will be monitored.

Once the IP address has been added, the user then chooses the sample rate that they would like the program to conduct. This sampling rate indicates how often the program should query the CubeSat for data and check the telemetry values for nominal or off-nominal operation. The sampling rate can be increased or decreased; however, it is limited by the internal rate of the CubeSat. If the sampling rate exceeds the internal rate, the algorithm will still perform but will have stagnant data until the internal data updates.

The 'Monitor Size' input allows the user to choose how many previous data points they would like to have visible on the screen at all times. The monitor size is currently set at 10 data points; coupled with the 1 Hz sampling rate, the user would be able to monitor the past 10 seconds of deviation values. This 'Monitor Size' can be increased or decreased during testing, and the view graph will reflect the update.

The final two buttons on this screen are 'Begin Monitoring' and 'Stop Monitoring'. When the 'Begin Monitoring' button is pushed, the system queries the CubeSat through the sys-util program installed on the monitoring computer. The current state of the CubeSat is requested with the sys-util program. In response, the CubeSat responds with a packet of telemetry and housekeeping data. An example of this response is laid out in Appendix A. Once the response is received, the program then parses and formats the data to match the data vector presented in Figure 2. Once this data is formatted correctly, the program performs a nearest neighbor search. In the nearest neighbor search, the formatted response from the CubeSat is the query point and the centroids that were calculated in the learning algorithm are the data set.

Once a nearest neighbor is found, the system determines if each parameter value of the new telemetry data is within the bounds of the cluster. This deviation from nominal is presented to the user in the view graph on the screen. The view graph shows deviation and the correlated time of the query in local 24-hour clock time. The local 24-hour clock time allows the user to easily see when an error occurred.

The view graph notifies the user of an off-nominal condition by turning red when one occurs. As the system updates, the error progresses to the left in the view graph and a large red asterisk marks that an error occurred at that time. In addition to showing the errors graphically on the screen, the GUI also includes the most recent error in the bottom left-hand corner. If a test conductor gets distracted for more time than is visual on the screen, then they can quickly determine if any off-nominal conditions have occurred in that time. Figures 20 and 21 demonstrate an error occurring in the system and the monitoring algorithm's response.

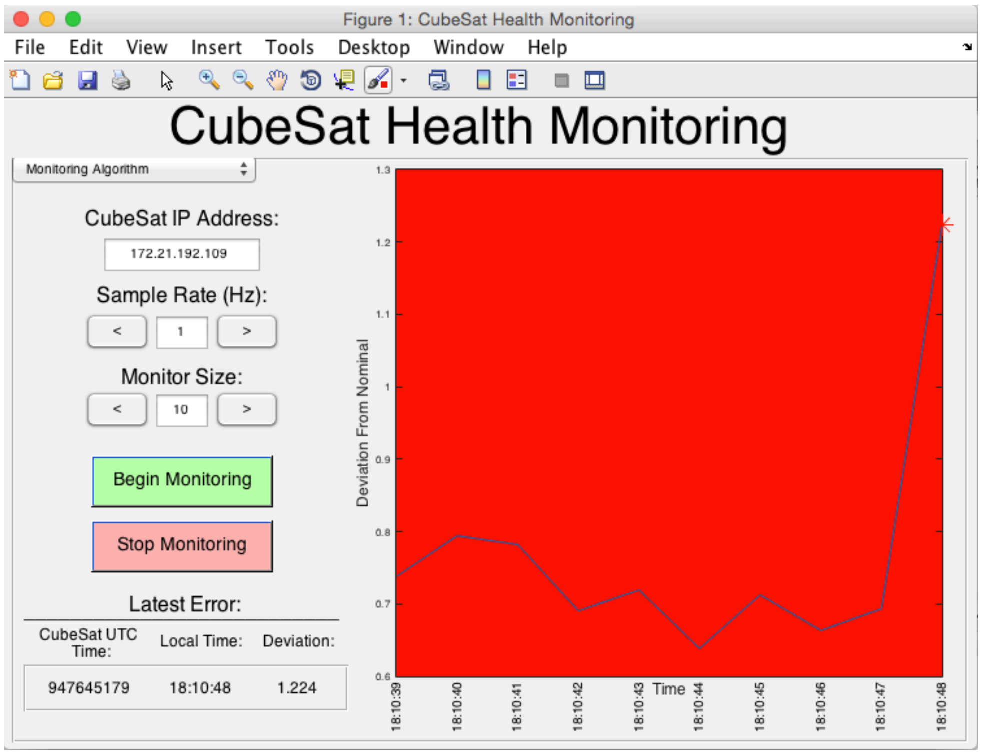


Figure 20: Initial Response to an Error

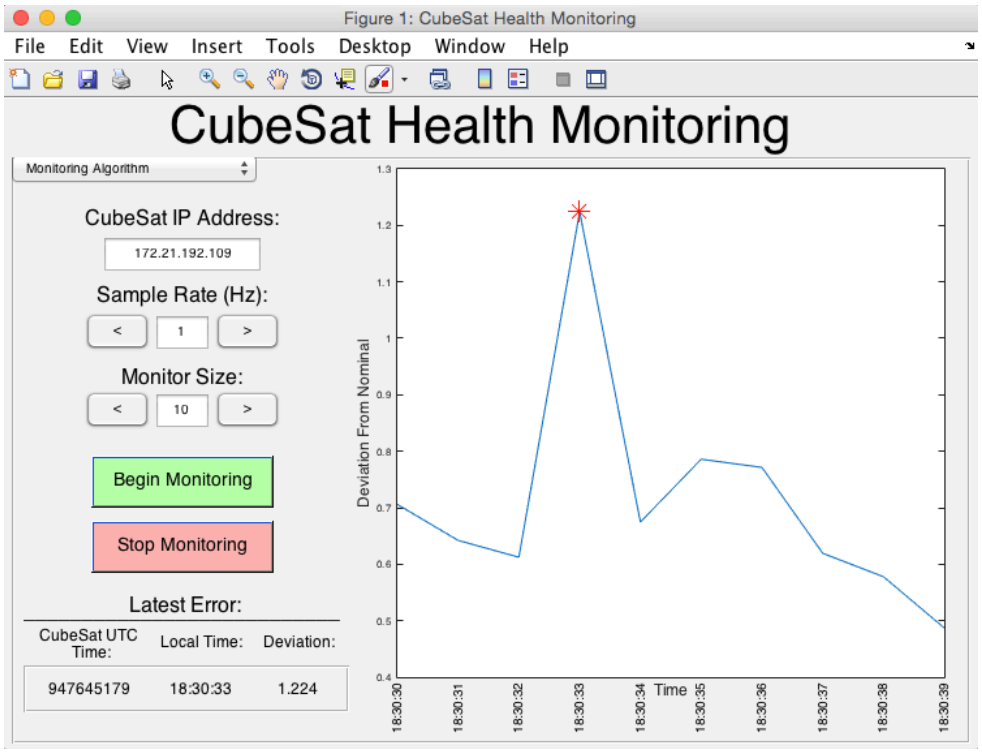


Figure 21: Previous Error Indication

Figure 20 shows what happens when the error initially occurs. The user is notified of an error occurring by the view graph flashing red for 1 second. Figure 21 shows how the error is still visible to the user as time passes, even if the state of the CubeSat returns to normal.

The final aspect to the ‘Monitoring Algorithm’ panel is the ‘Stop Monitoring’ button. When this button is pressed, the monitoring algorithm stops querying the spacecraft for new data and stops updating the view graph. The user is then given the option to save the data. If the user selects ‘yes,’ he or she is prompted for a file name. A suggestion of ‘Monitoring_Test_Data_DD-MMM-YYYY’ is given where the DD-MM-YYYY is automatically filled in with the current date. If the user selects a file name that has already been used, then the program notifies the user and asks if the user would like to replace the existing file or change the file name. When saving the data, the program saves the deviation from nominal values and the CubeSat UTC times. This is so that the times can be easily cross-referenced with any other testing files to investigate the anomaly and have equivalent time frames.