BLDC Drone Motor Fault Predictor Final Report

System Monitor

Team No 9

Members:

Max Goodkind – MAG13C

Gunnar Chauncey – GLC12

Loren Henderson – LH14D

Austen Mellers – ALM10D

Faculty Advisor/s

Dr. Steurer

Sponsor/s

General Atomics

Instructor Reviewers

Dr. Hooker

Dr. Steurer

Dr. Pamidi

Date Submitted

4/4/2017

Contents

Executive Summary

Problem Statement

Unpredictable motor failure can be a costly problem due to system downtime, wasted manpower, recovery costs, and a host of other reasons. Being able to predict if, and even when a motor will fail will be beneficial in the areas of drone operation to allow for better planning and less waste in loss of production. This project will focus on the data collection and monitoring of the motor operating characteristics under various conditions. That data can later be used to build a statistical model that will predict a potential motor fault, preventing catastrophic failures in undesirable locations.

Selected Method

Considering the problem of how to prevent drone motors from failing without prior knowledge, the team decided that the operating characteristics of the motor need to be monitored and measured. Since these motors can have either electrical or mechanical failures, the team realized that both types of data would need to be collected and monitored. This was accomplished using a test rig to safely mount the motor and sensors to collect mechanical and electrical operational data. The sensors were placed on the motor and in-line for mechanical and electrical sensors respectively.

Key Features

The key features of this project consist of the sensors for data collection, the BeagleBone processor to transform the data, and the web interface to display the data in real-time.

Evaluation Method

The evaluation method was based on the team's ability to collect the sensor data, process the data using the BeagleBone and monitor the operating characteristics of the drone motor in real-time on a web interface.

Results

The most important results of this project were the real-time operating characteristics of the motor, namely, voltage, current and temperature. These real-time data are important to the statistical model, as changes in these data are integral in the prediction of possible faults, both electrical and mechanical.

Chapter 1

Introduction

The fault predictor project was sponsored by General Atomics Electromagnetic Systems Group. This project was originally designed to measure data of a motor during operation and determine its likelihood to fail. During the commission of this project, there was an ITAR issue beyond the control of the students that significantly reduced the time allowed for its completion. As a result, it was suggested that the scope of work concerning the fault predictor project be changed to a system monitor, which is a necessary portion of the fault prediction design. The monitoring was performed using the Beaglebone Black microprocessor powered by a TI SITARA. A variety of sensors were connected to the motor to monitor voltage, current, and temperature. The data collected from those sensors was analyzed, processed, and displayed on a website in real-time. The motor operating characteristics were collected under various load conditions.

Unpredictable motor failure can be a costly problem due to system downtime, wasted manpower, recovery costs, and a host of other reasons. Being able to predict if, and when a motor will fail will be beneficial in these areas to allow for better planning and less waste in loss of production. This project will focus on the data collection and monitoring of the motor under various conditions. That data can later be used to predict a motor fault through statistical modeling, preventing catastrophic failure of the motor.

Operating Environment

Sponsorship by General Atomics lead the team to believe that the operating environment of this project to be variable. Fault prediction technology can be operated in dry, dusty, hot, wet, or hazardous environments, just to name a few. The motor was operated in the senior design lab during testing, which is a controlled environment with minimal changes to ambient temperature and humidity.

Intended uses/users

Since system monitoring is a technology that is used to track the characteristics of a system while it is in operation. Because it can technically monitor any system, it is versatile in its use, it is also versatile in its potential user base. This project was specific to a BLDC drone motor, which can be owned by a government or a hobbyist. The system that was designed is

scalable from a small hobby motor to an industrial motor, widening the potential users to anyone that has a need or desire to monitor the operating characteristics of their motor.

Assumptions and Limitations

It was assumed by the team that this project would be straight forward and have few limitations. However, the team discovered that the intricacies of a BLDC motor are vast, especially with little to no forehand knowledge of their operation. The team was limited in its data collection mainly due to the type of sensor selection. For example, the type of vibration sensor selected was ineffective for the collecting vibrational data from the motor. The sensor failed to collect usable data, likely due to the contact time of the inner spring. Time was an additional limitation due an above-mentioned issue during the design process that caused a work stoppage. This time limitation severely limited the team in the amount of work that could be performed.

End Products and Other Deliverables

The team delivered a working system monitor for a 100W BLDC drone motor. After the stop work order and the eventual change in scope the project was still completed on time. The team demonstrated real-time monitoring of the drone motor in a scalable form. The system data is the first phase of a working fault detector, which was the original scope of the project.

Chapter 2

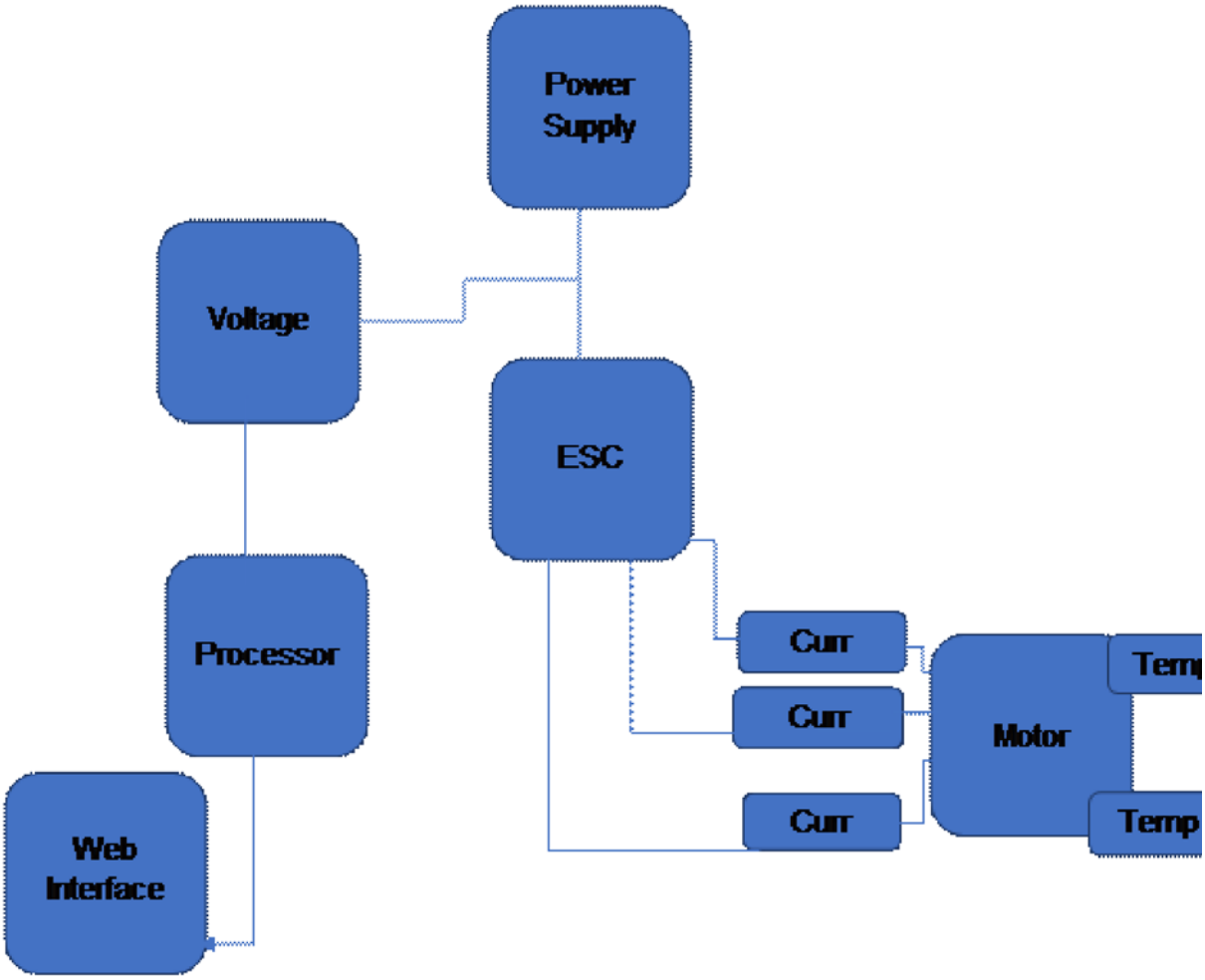
System Design

Due to the scope of the project, understanding of motor design, and time allotted, the team did not have many different design options. They quickly decided what characteristics needed to be monitored and the appropriate sensors to accomplish the task. The time constraint caused the students to employ the simplest approach of monitoring the current, voltage, and temperature of the motor. It was decided to mount the motor vertically due to safety concerns and a rig was suggested by the team's advisor, Dr. Steurer. It was also suggested that the team use another motor connected to the monitored one to create a dynamo that would allow for easy load changes during operation. This however was

outside of the students understanding and unable to be researched due to the time constraint.

The system was ultimately designed to monitor electrical parameters and potential bearing faults, using vibration sensors. The team decided to have the current measured in-line between the three phases of the motor and electronic speed control (ESC). This would allow the user to see whether there were possible faults in the ESC, or any phase between it and the motor. There was a voltage sensor between the input to the ESC and the power supply. This was intended to allow the user to know that there was enough power being input into the system to allow it to operate. There were temperature sensors attached to the coils of the motor to ensure that they didn't overheat due to loading or other electrical issues. The team intended on attaching vibration sensors to the motor as well, but the sensor that was purchased did not yield usable data.

The amount of time the team had to work on the project was the deciding factor in what design was used. Ideally, the team would have liked to try different designs to determine which would best serve their purposes, but with a lack of time to effectively explore other options, the team was forced to make a design decision and stick with it through the entire project.



Block Diagram of the System

Gunnar
Design of major components
Design of major components appendices
Gunnar will do budget

Austen
Test Plan
Test Plan Documentation

Design of Major Components

The design of this system was straightforward. The scope of work for this project made it necessary to create a system that could monitor the characteristics of a motor in real time and display the data the system is collecting. To make it the system as comprehensive as possible, the

team decided to measure several voltage and current values, as well as temperature and vibration of the motor. The system is set up to be able to measure in line DC voltage and current before the electronic speed control as well as AC current in each phase between the speed control and the motor. The system is also able to measure temperature and vibrations of the motor. These sensors are then fed into the BeagleBone Black microprocessor for data transformation and transmit to a database. The specific components are listed below with reasoning for why each was chosen.

- Tiger Motor U3 KV700
- Tiger Electronic Speed Control Flame 25A
- Propeller
- Gravity 50A current sensor
- Phidgets precision voltage sensor
- Vibration sensor
- Analog temperature sensor
- BeagleBone Black Microprocessor
- Testing stand

The components will be discussed in detail.

Tiger Motor

The model of motor used is the U3 KV700, using a 12 inch prop the specifications are as follows:

Item No.	Volts (V)	Prop	Throttle	Amps (A)	Watts (W)	Thrust (G)	RPM	Efficiency (G/W)	Operating temperature(℃)
U3	11.1 (3S)	T-MOTOR 12*4CF	50%	2.5	27.75	350	4000	12.61	40
			65%	4.8	53.28	550	4900	10.32	
			75%	6.6	73.26	700	5500	9.56	
			85%	9.1	101.01	870	6300	8.61	
			100%	11.1	123.21	1000	6600	8.12	
		T-MOTOR 13*4.4CF	50%	2.9	32.19	400	3800	12.43	42
			65%	5.6	62.16	650	4900	10.46	
			75%	7.9	87.69	830	5300	9.47	
			85%	10.5	116.55	1000	6000	8.58	
			100%	12.6	139.86	1100	6400	7.87	
		T-MOTOR 14*4.8CF	50%	4.1	45.51	550	3500	12.09	43
			65%	7.7	85.47	890	4500	10.41	
			75%	10.7	118.77	1060	4900	8.92	
			85%	14.5	160.95	1300	5500	8.08	
			100%	17.3	192.03	1460	5800	7.60	

Figure 1: Tiger U3 KV700 Motor Specification Table

The motor is connected to the Electronic Speed control by three wires. The order in which the wires are connected does not matter, however if it is found that the motor is spinning in the wrong direction, simply change two of the connecting wires. This motor was specifically picked as it fit the statement of work incorporating a drone motor. This motor is designed for use in hobby drone projects. Originally this project was designed to use a minimum 500 W motor. The team decided a motor of that size was too large to safely test in the lab, therefore the size of the motor was scaled down in the interest of safety.

Tiger Electronic Speed Control

The Tiger ESC Flame 25A is the electronic speed control for the motor. This device takes in a DC voltage along with a pulse width modulation signal to control the speed of the motor. There are two methods to use the ESC to change the speed of the motor. The speed of the motor can be changed by adjusting the time that the pwm pulse is on. Therefore by changing the frequency or the duty cycle of the pulse, the speed will change. This specific ESC was chosen because it is designed to control tiger motors directly. The specifications of the input of this device is given in the chart below:

Parameters	FLAME25A			Unit
	MIN	Typical Value	MAX	
Input Voltage	6.4	14.8	17.4	V
PWM	3.0	-	5.0	V
Max Continuons Current	-	-	25	A
Peak Current(3S)	-	-	30	A
Oneshot125 Signal Frequency Compatible	30	-	600	Hz
Regular Signal Frenquency Compatible	30	-	500	Hz
Ambient Temp	-10	25	50	℃

Figure 2: Tiger ESC Flame 25A Specification Table

The ESC has some error states that are determined by the sound of the motor beeping. The following is a chart of the error state and description.

Error State	Description
BBBB...	Please check FC, receiver or RC equipment
B—B—B...	No throttle signal

Figure 3: ESC Error States

This device is connected to a power supply, as well as a PWM signal generated by the BeagleBone Black, it is then connected to the motor by three wires.

Propeller

The propellers used are carbon fiber 12 inch propellers. They are connected to the motor by a pair of screws and are used as the load to be tested. In order to change the load, the propeller can be defaced (drilled holes, cracked, etc.).

Figure 4: Tiger 12 in. Carbon Fiber Propeller

Gravity 50A Current Sensor

The Gravity current sensor is designed to sense up to 50A of AC or DC current. This device is intended to measure the current through each phase of the motor as well as overall current coming from the supply to determine if it is being fed the correct amount of current. This will be connected between the ESC and the Motor and between the power supply and the ESC. The output of this sensor measures 40 mV per 1 A it senses. This sensor takes 5V to power and is powered by the BeagleBone Black board. There are three current sensors used to measure the current of each phase of the motor. Using figure 9 as reference, pins P9_5, P9_6, and P9_7 supplies the 5 V, pins P9_1, P9_2, and P9_43 act as a ground, and pins P9_33, P9_35, and P9_36 are the analog inputs. This sensor was chosen due to the capability of measuring up to 50A. In the future the project can be scaled up and these sensors will be able to accommodate the increase in current going through a higher powered motor. A simple diagram of the setup is shown below:

Figure 5: Gravity Current Sensor Connection Diagram

Phidgets Precision Voltage Sensor

The Phidgets voltage sensor is designed to sense between -30 to 30 V differential. This device only detects DC voltage therefore it will be connected between the power supply and ESC to measure the input voltage. The output of this sensor shows between 0.5V and 4.5V. At 2.5 V output, the sensor is reading 0 V differential. A formula must be applied to the output voltage read to determine the actual differential voltage which is given below. This sensor takes 5V to power and is powered by the BeagleBone Black board. Using figure 9 as reference, the sensor is powered by pin P9_8, grounded by P9_44, and the analog input to the board is P9_37. A simple diagram is shown below:

$$V_{diff} = \frac{\frac{SensorValue}{200} - 2.5}{0.0681}$$

Equation 1: Formula to convert sensor data into usable real values

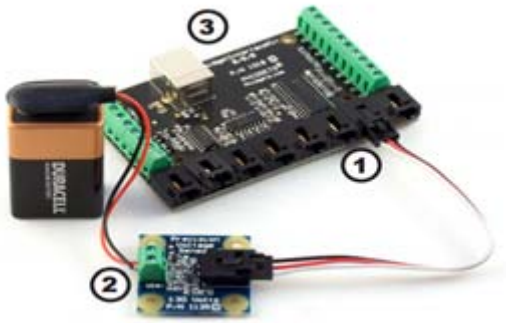


Figure 6: Phidgets Voltage Sensor Connection Diagram

Looking at figure 6, the sensor is being powered by, sending the output voltage, and is grounded by three wires connected to the board on the right. The sensor is measuring the voltage of the battery on the left. This sensor was chosen due to the capability of measuring up to 30V. In the future, the project can be scaled up and these sensors will be able to accommodate the increase in voltage going through a higher-powered motor.

Vibration Sensors

There are three versions of the vibration sensors. They are labeled as Slow, Medium, and Fast. The labels determine the amount of force it takes to trigger them. The vibration sensors works as a simple switch which when vibrated enough will close the switch and allow the current/voltage through them to be read by the BeagleBone Black. These do not have a required power amount, however they will be powered by the 1.8V port on the BeagleBone Black, pin P_32. The reason for purchasing of three different types of sensors, the team was unsure of how sensitive the device would need to be to measure too much vibration. It has been found out that the Fast, or most sensitive device is necessary.

Figure 7: Vibration Sensor

Analog Temperature Sensors

The temperature sensor is designed to read from a range of -50ºC to 125ºC. The sensor is powered by a source between 2.7 and 5.5V and it outputs a voltage between 0V and 1.75V. The formula given to determine the correct temperature based on the voltage reading is given below

$$Temp\ Celcius = 100 * (reading\ in\ V) - 50$$

Equation 2: Formula to convert temperature data into usable values

Using figure 9 as reference, the temperature sensor is connected to pin P9_3 for power, P9_45 for ground, and P9_38 for the analog input. This sensor is capable of measuring from -50

°C to 125 °C. The motor itself does not get much hotter than 40 °C, therefore this sensor is more than capable of handling the temperatures required.

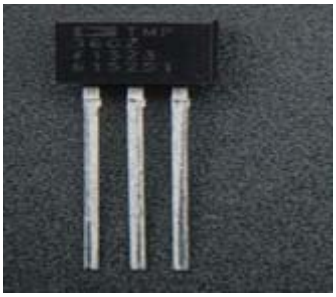


Figure 7: Temperature Sensor

BeagleBone Black

The BeagleBone Black is the processor that will be reading in the measurement parameters while also supplying power to most of the sensors that have been described above. The processor has several power pins to provide power to the other devices as well as many analog input pins to take in the measurements that the devices are reading. The BeagleBone Black itself can be powered through a computer using a USB cable, or through a 5V power supply. This processor is ideal for the requirements of this project. It is capable of handling up to 8 analog input values, as well as providing four separate 5V output pins and two 3.3V output pins. The BeagleBone Black also is capable of providing a pwm signal to control the motor through.

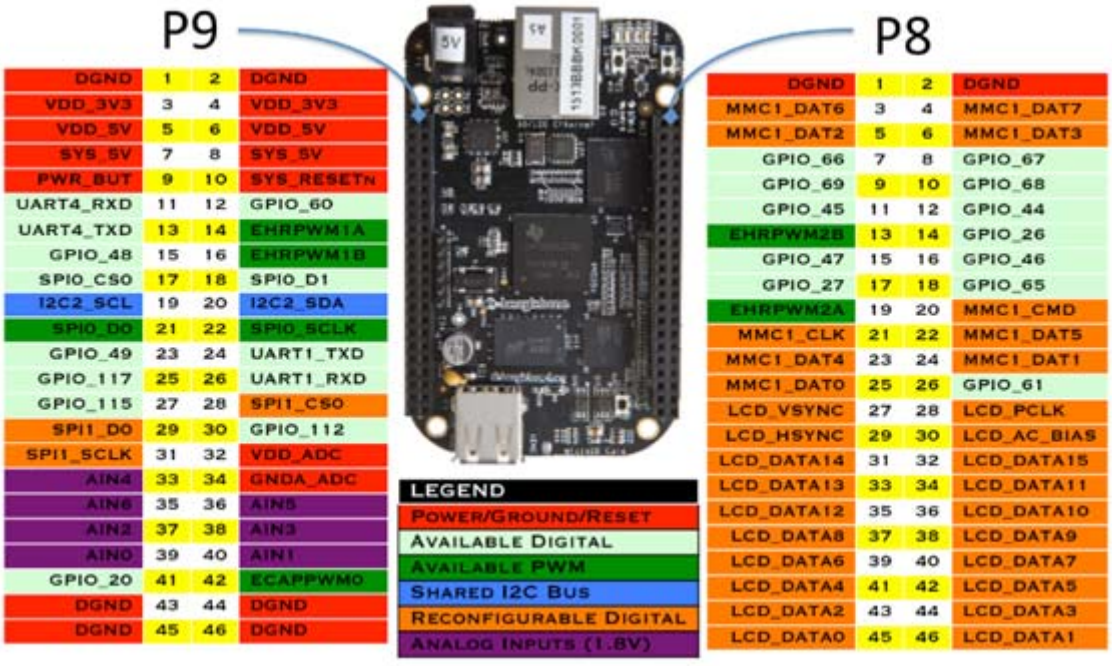


Figure 9: BeagleBone Black Pin Layout

Testing Stand

A testing stand has been designed by the team and built by the College of Engineering machine shop to safely mount and test our motor and other devices. This stand is made of wood and aluminum and is to be clamped to a table when in use. A picture is found below:

Figure 10: Testing Stand for the Motor and Components

Data Sheets

The data sheets for each of these components can be found online. The links below can be used to find the data sheet for each of the products.

Tiger Electronic Speed Control
http://www.rctigermotor.com/html/2016/esc_0621/314.html

Gravity 50A Current Sensor
http://www.robotshop.com/en/gravity-50a-current-sensor-ac-dc.html

Phidgets Voltage Sensor
http://www.phidgets.com/docs/1135_User_Guide

Vibration Sensors
https://www.adafruit.com/product/1766
https://www.adafruit.com/product/2384
https://www.adafruit.com/product/1767

Temperature Sensor
http://cdn.sparkfun.com/datasheets/Sensors/Temp/TMP35_36_37.pdf

Beaglebone Black
http://beagleboard.org/getting-started

Test Plan

This section covers the testing of various elements of the Fault Tolerant Controller. All tests were made using a power supply limited to a maximum of 20 Volts and 5 Amps. The pulse-width modulation (PWM) signal that drove the motor was constant at 400 Hz, with duty cycles ranging from 50% to 90%. A full test cycle is defined as a period of time where the power supply and PWM signal are both off, a period of time where both are on and the motor is in operation, and a period of time where the everything is off after the motor has run.

- **Motor Temperature**

  One of the reasons that a motor can fault is from overheating. Although not the most common fault, it is worth using sensors to monitor. Before testing the temperature of the motor, the actual sensor had to be verified as functional. After wiring it to the Beaglebone Black circuit, running a C++ program captured and stored the data. Varying the temperature that the sensor was exposed to by pressing it against heat sources would cause the sensor readings to react accordingly, verifying the functionality of the sensor. Once verified, the sensor was placed inside the Tiger motor, against the coils. Once wired up, the C++ program was ran for one full test cycle. The data showed an initial temperature around 24 degrees Celsius, which was the ambient room temperature. Once the motor began operation, there was a rather inconsistent spike that ranged from 22 degrees Celsius to 26 degrees Celsius. The instant the motor was shut off, the temperature stopped spiking and read the same ambient room temperature as before motor operation. However as time passed, the temperature rose until it crossed over 26 degrees Celsius before leveling out. The next step for temperature testing is to add the second temperature sensor on the outside of the motor, and compare both sensors simultaneously.

- **Three Phase Current**

  A main metric to monitor for motor systems is the various phase currents. These currents were measured using three separate current sensors. Before testing the phase currents of the motor, the sensors were tested to verify functionality. Each requires a 5V power supply, which was initially provided by the Beaglebone Black's 5V output port. However, after troubleshooting problems with the sensor, it was determined that the Beaglebone Black only output 4.9V. A more accurate power supply was substituted. Using this new power supply, running the C++ program to capture and store data resulted in at least some sort of sensor output. However, after testing the sensors with AC current, DC current, and

through multiple different circuits (including the motor phase currents), the sensor output did not change. Due to this, the functionality of the sensors could not be verified. The next step in testing the three phase currents is to purchase new sensors.

- **Bearing/Vibration**

  Bearing faults are the most common faults in most motors, and as such, require sensors to obtain accurate data for the fault model. The plan to get usable measurements was to use multiple sensors, in hopes that at least one would provide useful data. Three vibration sensors (one for light vibrations, one for medium vibration, one for heavy vibrations) and an accelerometer were to be used to capture this data. Unfortunately, there were multiple issues in obtaining the vibration sensors, so the accelerometer was the only one that was tested. The first step in using the sensor was to verify that the accelerometer was functional. Unfortunately, the accelerometer required certain software to be used that has yet to be implemented into the fault model monitoring system. Future plans for this test are to either purchase a new sensor, or interface the accelerometer code into the system monitoring code.

- **Input Voltage**

  Measuring the input voltage is important for redundancy. Having an accurate measurement verifying the voltage powering the motor helps eliminate faulty power supply as a potential fault source. The test plan for this began by verifying the voltage sensor. The sensor was wired up to a voltage divider circuit and then into the motor circuit. Using a multimeter to read the sensor output, it matched the display on the power supply, verifying the sensor's functionality. The sensor was then wired to the Beaglebone Black, and a C++ program was ran to capture and record the data. The sensor captured an entire test cycle, and verified the accuracy of the power supply display.

- **Input Current**

  The input current is another metric that is important for redundancy. It is also used in eliminating or identifying the power supply as a potential fault source. This test used a different sensor than the phase current sensors. The test plan for this began with verifying the functionality of the current sensor. The sensor was wired up into the motor circuit, and a multimeter was used to read the sensor output. The multimeter readings matched the current display on the power supply, verifying the sensor's functionality. The next step was to wire the sensor to the Beaglebone black. Using a test python script that just output the sensor data, the sensor data was valid and accurate. However, running the C++ program that not only output the sensor data, but also stored it in a database, the numbers were jittery. In some tests the current numbers were acceptable, yet in others without making changes, they were not. The next step in this test is to determine the source of the instability of the numbers when using the C++ code.

Max
Ch5
Project management
The initial project schedule separated into different quarters is listed below:

- **Quarter 1 (September 1, 2016 - October 31, 2016)**
  - **Presentation 1 (October 13 - 18, 2016)**
  - **Midterm Report 1 (October 21, 2016)**
  - **Budgeted Component List Finalized (October 27, 2016)**
  - **Begin Learning SQLite Database and R Programming Language (October 27, 2016)**

- **Quarter 2 (November 1, 2016 - January 15, 2017)**
  - **Presentation 2 (November 18, 2016)**
  - **Begin determination of Fault Model (December 2, 2016)**
  - **Begin Code for SQLite (December 5, 2016)**
  - **Begin Fault Model Coding in R Programming Language (January 2, 2017)**
  - **Poster Presentation (December 1, 2016)**
  - **Final Report (December 5, 2016)**
  - **Website Due (December 5, 2016)**
  - **Finalize decision for Statistical Fault Model (January 9, 2017)**

- **Quarter 3 (January 16, 2017 - February 28, 2017)**
  - **Order Components (January 7, 2017)**
  - **Begin Web Interface Code (February 1, 2017)**
  - **Begin Sensor Interfacing (February 10, 2016)**
  - **Presentation 3 (February 15, 2017)**
  - **Midterm Report (February 24, 2017)**

- **Quarter 4 (March 1, 2017 - April 30, 2017)**
  - **Begin System Testing (March 3, 2017)**
  - **Final Report (April 17, 2017)**
  - **Final Presentation (April 14, 2017)**

On October 28, 2016 a stop work order was ordered by our professor and General Atomics due to an International Traffic in Arms Regulations (ITAR) issue. This lasted until January 25, 2017 when we received the go ahead to once again begin working on the project.

Due to this delay, the project schedule was drastically changed. Instead of creating a system fault predictor the new scope of work changed so that the project could be worked on once again the following year. In order to do this the scope of work for this semester was changed to only include implementing a system monitoring solution rather than both a system monitor and fault predictor. The following year will then be able to use the system monitor to create fault models for a fault prediction algorithm.

The new project schedule is listed below:

- **Quarter 1 (September 1, 2016 - October 31, 2016)**
  - **Presentation 1 (October 13 - 18, 2016)**
  - **Midterm Report 1 (October 21, 2016)**
  - **Budgeted Component List Finalized (October 27, 2016)**
  - **Begin Learning SQLite Database (October 27, 2016)**

- **Quarter 2 (November 1, 2016 - January 15, 2017)**

- **Quarter 3 (January 16, 2017 - February 28, 2017)**
  - **Order Components (February 4, 2017)**
  - **Begin Web Interface Code (February 1, 2017)**
  - **Begin Sensor Interfacing (February 10, 2016)**
  - **Presentation 3 (February 28, 2017)**
  - **Midterm Report (February 24, 2017)**

- **Quarter 4 (March 1, 2017 - April 30, 2017)**
  - **Construct System Design (March 3, 2017)**
  - **Begin System Testing (March 21, 2017)**
  - **Operation Manual (April 7, 2017)**
  - **Final Report (April 17, 2017)**
  - **Final Presentation (April 14, 2017)**

Ch6
Budget

Initial Parts List

Item	**Quantity**	**Cost**
BeagleBone Black	1	$55.00

Super Power Supply 5V 2A	1	$7.95
64 GB micro sd card	1	$19.99
Micro HDMI to HDMI	1	$6.49
Tiger Motor U3 KV700	1	$109.90
Gravity 50A current sensor	1	$16.50
Phidgets precision voltage sensor	1	$19.00
Mid vibration sensor	2	$0.95
Fast vibration sensor	2	$0.95
Slow vibration sensor	2	$0.95
Analog Temperature sensor	2	$1.50
Tiger Electronic Speed Control	1	$15.99
Propeller X2	1	$38.90
Breadboard/jumper wires	1	$4.29

Total Spent: $302.71

Final Official parts list

Item	**Quantity**	**Cost**
BeagleBone Black	4	$55.00
Super Power Supply 5V 2A	1	$7.95
64 GB micro sd card	1	$19.99
Micro HDMI to HDMI	1	$6.49
Tiger Motor U3 KV700	4	$109.90
Gravity 50A current sensor	4	$16.50

Phidgets precision voltage sensor	2	$19.00
Accelerometer MMA 8452	1	$19.95
Mid vibration sensor	2	$0.95
Fast vibration sensor	2	$0.95
Slow vibration sensor	2	$0.95
Analog Temperature sensor	2	$1.50
Tiger Electronic Speed Control	4	$15.99
Propeller X2	1	$38.90
Breadboard/jumper wires	1	$4.29
Miscellaneous/Lowe's	N/A	$34.76

Total spent: $1,201.02
The initial plan for this project was to acquire all of the components listed in the initial parts list. The team determined it necessary to purchase extra of specified parts in case any problems arise. The scope of the project also changed which involved purchasing more sensors as well as an accelerometer. The team felt it necessary to purchase extra microprocessors so multiple members could be testing with them at the same time, not having to rely on one microprocessor allowed work to be done faster.

Ch7

Conclusions

In conclusion, this project was successful in creating a system that monitored the voltage, temperature, and current of a drone motor, specifically the tiger U3 KV700 motor. A Voltage sensor was connected in parallel between the power supply and the esc to continually monitor the supply voltage in real-time while a current sensor was connected in series between the power supply and esc to monitor the supply current. Three more current sensors were connected to each phase of the motor to monitor the phase current of the motor as it ran in real-time. Temperature sensors were attached to the coils inside the motor to measure the internal temperature of the motor as it ran.

With all of these sensors collecting data the Beaglebone successfully collected and stored these values in a SQLite database in numerical form. The stored data was then sent to a google chart where it displayed the voltage, current, or temperature readings in real-time as the sensors collected the data in graphical form.

Possible future updates to this project include using this monitoring system to create fault models for the motor by continuously running tests on the motor until failure and observing the reasons for the motor failure. By doing this it will be possible to create a fault prediction algorithm that will predict when the motor is going to fail and the reason of failure. This can be accomplished by implementing a Bayesian network or using Fuzzy Logic. Once the fault prediction algorithm is completed a motor control system can be implemented. Assuming that the drone the motor is attached to has more than one motor it may be possible to adjust the power for each motor individually so that the working motors make up for the lower performance of the failing motor. As a result the drone will stay airborne without crashing. More sensors can also be added to this project such as vibration sensors, accelerometers, voltage sensors for each phase, etc.

Ch8

References

We used many references for the research of the fault prediction model to write the literature review for Dr. Arghandeh but due to the scope being changed they will not be included. The following are websites used to help us learn how to code the Beaglebone Black as well as professors and other researchers that advised us throughout the timeline of the project.

[1]"Beaglebone Black | Technology Tutorials", *Toptechboy.com*, 2017. [Online]. Available: http://www.toptechboy.com/beaglebone-black/. [Accessed: 2- Apr- 2017].

[2] Dr. Jerris Hooker, professor, FAMU-FSU College of Engineering

[3] Dr. Mischa Steurer, Research Faculty, Center for Advanced Power Systems (CAPS)

[4] Dr. Sastry Pamidi, Associate Professor, FAMU-FSU College of Engineering

[5] Dr. Thomas Lipo, Research Professor, FAMU-FSU College of Engineering

User's guide - content

Materials Included:

- Testing Stand

- Tiger U3 KV700 motor
- Phidgets Voltage Sensors
- Gravity Current Sensors
- Temperature Sensors
- Beaglebone Black Microprocessor
- Screws
- Wires
- Breadboard

General

To safely set up the system, be sure to read and understand the specific manuals for each part. It is the goal to be able to connect everything from this guide, however if some things are missed, refer to each devices manual. Be sure when connecting devices to avoid short circuits between the devices or incorrect connections between any of the pieces of equipment. To avoid unnecessary complications, ensure all of the sensors are mounted to the wooden board in an organized manner.

Hardware

The hardware setup begins with the mounting of all of the equipment onto the testing stand. The holes in the vertical aluminum piece are for the motor to be mounted there. All other pieces of equipment should be mounted onto the wooden base of the stand. There is to be 3 current sensors connected in between the ESC and the Motor to measure the current of each phase. There is one voltage sensor connected between the ESC and the power supply. There are 2 temperature sensors that will be connected in/on the motor. One will be placed inside the motor touching the coils to get an internal measurement of the temperature, while the other is to be placed on the outer shell of the motor. The sensor placed inside the motor should be held there by an insulative epoxy that will not affect the motor. The vibration sensor will be placed on the outside of the motor. The motor/ESC will be powered by a desktop voltage supply, while the sensors will be powered through the BeagleBone Black. The sensors output will also be fed to the BeagleBone Black.

Wiring

A wiring diagram has been drafted to help understand the setup. Each yellow wire of this diagram is made to represent 3 total wires. Two of the wires are to be the power and ground of each device needing to be powered which is supplied by the BeagleBone. The third wire is to be the output of the sensor which will be measured by the BeagleBone. On the schematic, several voltage dividers, simply consisting of resistors, are shown. These are due to the BeagleBone's analog reading limitation of 1.8V. These voltage dividers were created to make sure the voltage going into the BeagleBone's pins is below 1.8 V so no damage to the BeagleBone occurs. The pin numbers that each sensor needs to be plugged into are all listed in the operation manual. Generally speaking, attach the Beaglebone 5 V to one side of the breadboard and the 3.3 V to the

other side of the breadboard. Power the sensors using these voltages, in the case of the current sensors, the 5V that the Beaglebone outputs is actually only 4.89 V which is not enough for these sensors. Instead, connect the sensors to a power supply that is providing a true 5 V. Next, connect the sensor analog output through the voltage divider if needed and then to the analog pins of the Beaglebone. The pins you choose do not matter just make sure the code for each sensor is collecting data from the correct pin that the sensor is connected to.

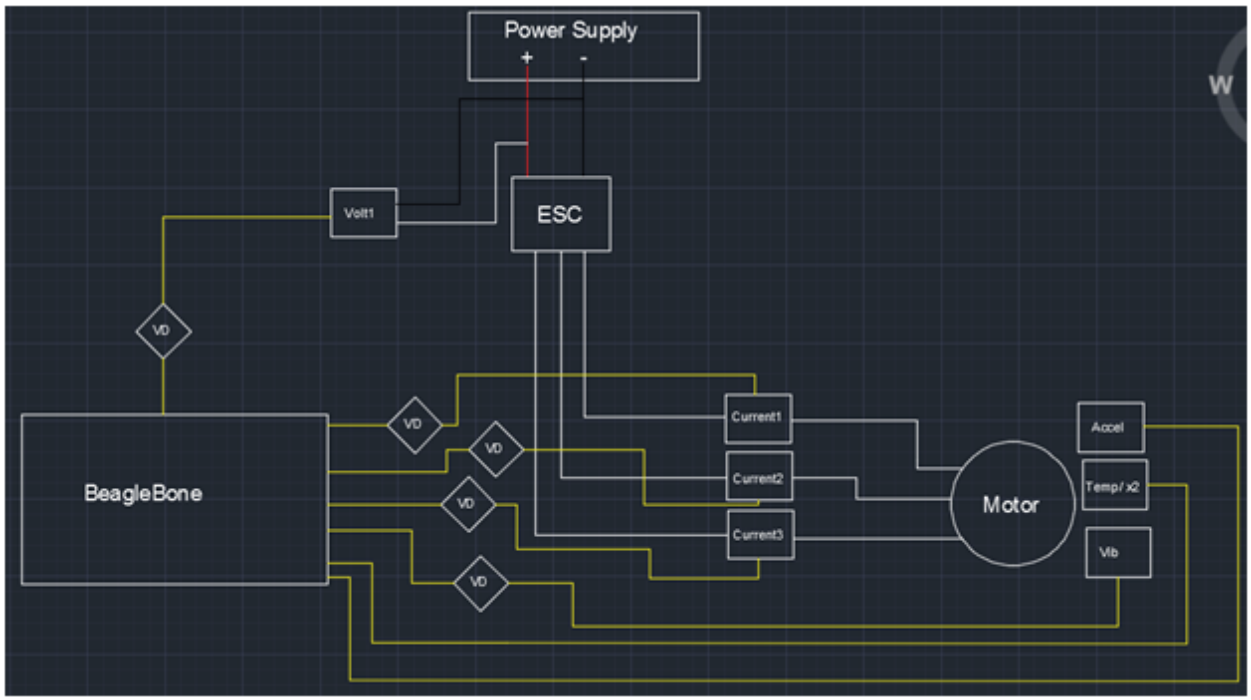


Figure 11: Wiring schematic of testing setup

Software

There have been several programs created by the team in order to test our product. These programs are located in the root directory and they are written in the python programming language. If there is any trouble working with the BeagleBone, it is suggested to go to Beagleboard.org/getting-started. The output of these programs can be seen on the BeagleBone itself, or through the website that the data is being transmitted to. The code used in this project can be found at the end of this document.

Operation

After successful setup is complete, the system can be operated. First make sure the ESC as well as all the sensors are being powered by the correct sources. Next, the system can be operated by simply running a program from the BeagleBone. This program will allow for change in the speed of the motor, by adjusting the PWM signal duty cycle, as well as show all of the characteristics of the sensors measurements. Be sure to test the motor in a safe environment.

Make sure there is proper safety equipment in place when the motor is running in case something were to break. Do not touch the propeller when the motor is operating and be sure to enclose the propeller to prevent injury. Do not touch any of the electronics while operating in case of electrical shock.

Troubleshooting & Maintenance

A common problem found with the motor is not supplying a correct PWM voltage, make sure the ESC has a PWM signal between 3-5 V to allow the motor to work. Make sure no lines are shorted when the experiment is in progress. Be sure to have all the equipment properly mounted. The motor is small but powerful and could get out of control if not mounted.

Code

Below is the code needed to interface the voltage sensor, store the data in MySQL, and display it on the google chart in real-time:

```
#include <mysql.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <fcntl.h>
#include <unistd.h>
#define MAX_BUF 64

/// Prints out a MySQL error message and exits
///
/// This function should be called after a MySQL error has been encountered.  This function will then
/// notify the user of the error that has occurred, clean up the existing MySQL connection, and then
/// exit the program.
///
/// @param The MySQL connection to clean up before exiting
void error_exit(MYSQL *con)
{
fprintf(stderr, "%s\n", mysql_error(con));

if (con != NULL)
{
mysql_close(con);
}

exit(1);
}

//int readADC(unsigned int pin);
```

```
int main(int argc, const char *argv[])
{
// Initialize a connection to MySQL
MYSQL *con = mysql_init(NULL);
if(con == NULL)
{
error_exit(con);
}

// Connect to MySQL
// Here we pass in:
//  host name => localhost
//  user name => bone
//  password => bone
//  database name => TempDB
if (mysql_real_connect(con, "localhost", "bone", "bone", "TempDB", 0, NULL, 0) == NULL)
{
error_exit(con);
}

// Create the TempMeas database (if it doesn't already exist)
if (mysql_query(con, "CREATE TABLE IF NOT EXISTS VoltMeas(MeasTime DATETIME, Volt DOUBLE)"))
        {
        error_exit(con);
        }
MYSQL_STMT *stmt = mysql_stmt_init(con);
if (stmt == NULL)
        {
        error_exit(con);
        }

// Set out insert query as the MySQL statement
const char *query = "INSERT INTO VoltMeas(MeasTime, Volt) VALUES(NOW(), ?)";
if (mysql_stmt_prepare(stmt, query, strlen(query)))
        {
        error_exit(con);
        }

// Create the MySQL bind structure to store the data that we are going to insert
double volt = 0.0;
MYSQL_BIND bind;
memset(&bind, 0, sizeof(bind));
bind.buffer_type = MYSQL_TYPE_DOUBLE;
bind.buffer = (char *)&volt;
bind.buffer_length = sizeof(double);

// Bind the data structure to the MySQL statement
if (mysql_stmt_bind_param(stmt, &bind))
        {
        error_exit(con);
        }

//-----------------------------------------------------------------------------------------------
// Insert multiple records into the database,
```

```
// with different data each time
/*for (int i = 0; i < 10; i++)
        {
        int adc6 = readADC(6);
        double y = (double)adc6;
        y = y / 1000.000;
        y = (y*100) - 50;
        temp = y;
        mysql_stmt_execute(stmt);
        sleep(1);
        }
*/
int adc5;
double y;
int fd;
char buf[MAX_BUF];
char val[4];
//for (int i = 0; i < 10; i++)
while(1)
{
snprintf(buf, sizeof(buf), "/sys/devices/ocp.3/helper.15/AIN%d", 5);
fd = open(buf, O_RDONLY);
if (fd < 0)
        {
        perror("ADC - problem opening ADC");
        }
read(fd, val, 4);
close(fd);
adc5 = atoi(val);
y = (double)adc5;
y = y / 1000.000;
y = ((y/.1732)-2.5)/0.0681;
y = y - 29;
y = y / 1.4;
y = y / 1.3;
volt = y;
printf("Voltage: %f\n",volt);
mysql_stmt_execute(stmt);
sleep(1);
}

//--------------------------------------------------------------------------------------------

// Close the MySQL connection
mysql_close(con);

return 0;
}
```

Below is the code needed to interface the voltage sensor, store the data in MySQL, and display it on the google chart in real-time:

```
#include <mysql.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <iostream>
```

```
#include <fcntl.h>
#include <unistd.h>
#define MAX_BUF 64

/// Prints out a MySQL error message and exits
///
/// This function should be called after a MySQL error has been encountered.  This function will then
/// notify the user of the error that has occurred, clean up the existing MySQL connection, and then
/// exit the program.
///
/// @param The MySQL connection to clean up before exiting
void error_exit(MYSQL *con)
{
fprintf(stderr, "%s\n", mysql_error(con));

if (con != NULL)
{
mysql_close(con);
}

exit(1);
}

//int readADC(unsigned int pin);




int main(int argc, const char *argv[])
{
// Initialize a connection to MySQL
MYSQL *con = mysql_init(NULL);
if(con == NULL)
{
error_exit(con);
}

// Connect to MySQL
// Here we pass in:
//  host name => localhost
//  user name => bone
//  password => bone
//  database name => TempDB
if (mysql_real_connect(con, "localhost", "bone", "bone", "TempDB", 0, NULL, 0) == NULL)
{
error_exit(con);
}

// Create the TempMeas database (if it doesn't already exist)
if (mysql_query(con, "CREATE TABLE IF NOT EXISTS AmpMeas(MeasTime DATETIME, Amp DOUBLE)"))
        {
        error_exit(con);
        }
MYSQL_STMT *stmt = mysql_stmt_init(con);
if (stmt == NULL)
        {
        error_exit(con);
```

```
        }

// Set out insert query as the MySQL statement
const char *query = "INSERT INTO AmpMeas(MeasTime, Amp) VALUES(NOW(), ?)";
if (mysql_stmt_prepare(stmt, query, strlen(query)))
        {
        error_exit(con);
        }

// Create the MySQL bind structure to store the data that we are going to insert
double amp = 0.0;
MYSQL_BIND bind;
memset(&bind, 0, sizeof(bind));
bind.buffer_type = MYSQL_TYPE_DOUBLE;
bind.buffer = (char *)&amp;
bind.buffer_length = sizeof(double);

// Bind the data structure to the MySQL statement
if (mysql_stmt_bind_param(stmt, &bind))
        {
        error_exit(con);
        }

//-----------------------------------------------------------------------------------------------
// Insert multiple records into the database,
// with different data each time
/*for (int i = 0; i < 10; i++)
        {
        int adc6 = readADC(6);
        double y = (double)adc6;
        y = y / 1000.000;
        y = (y*100) - 50;
        temp = y;
        mysql_stmt_execute(stmt);
        sleep(1);
        }
*/
int adc4;
double y;
int fd;
char buf[MAX_BUF];
char val[4];
//for (int i = 0; i < 10; i++)
while(1)
{
snprintf(buf, sizeof(buf), "/sys/devices/ocp.3/helper.15/AIN%d", 4);
fd = open(buf, O_RDONLY);
if (fd < 0)
        {
        perror("ADC - problem opening ADC");
        }
read(fd, val, 4);
close(fd);
adc4 = atoi(val);
y = (double)adc4;
y = y / 1000.000;
y = y - .82;
```

```
y = y / .013;
y = y - .3;
y = y * -1;
printf("Current: %f\n",y);
amp = y;
mysql_stmt_execute(stmt);
sleep(1);
}

//------------------------------------------------------------------------------------------

// Close the MySQL connection
mysql_close(con);

return 0;
}
```

Below is the code needed to interface the Temperature sensor, store the data in MySQL, and display it on the google chart in real-time:

```
#include <mysql.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <iostream>
#include <fcntl.h>
#include <unistd.h>
#define MAX_BUF 64

/// Prints out a MySQL error message and exits
///
/// This function should be called after a MySQL error has been encountered.  This function will then
/// notify the user of the error that has occurred, clean up the existing MySQL connection, and then
/// exit the program.
///
/// @param The MySQL connection to clean up before exiting
void error_exit(MYSQL *con)
{
fprintf(stderr, "%s\n", mysql_error(con));

if (con != NULL)
{
mysql_close(con);
}

exit(1);
}

//int readADC(unsigned int pin);



int main(int argc, const char *argv[])
{
// Initialize a connection to MySQL
```

```
MYSQL *con = mysql_init(NULL);
if(con == NULL)
{
error_exit(con);
}

// Connect to MySQL
// Here we pass in:
//  host name => localhost
//  user name => bone
//  password => bone
//  database name => TempDB
if (mysql_real_connect(con, "localhost", "bone", "bone", "TempDB", 0, NULL, 0) == NULL)
{
error_exit(con);
}

// Create the TempMeas database (if it doesn't already exist)
if (mysql_query(con, "CREATE TABLE IF NOT EXISTS TempMeas(MeasTime DATETIME, Temp DOUBLE)"))
        {
        error_exit(con);
        }
MYSQL_STMT *stmt = mysql_stmt_init(con);
if (stmt == NULL)
        {
        error_exit(con);
        }

// Set out insert query as the MySQL statement
const char *query = "INSERT INTO TempMeas(MeasTime, Temp) VALUES(NOW(), ?)";
if (mysql_stmt_prepare(stmt, query, strlen(query)))
        {
        error_exit(con);
        }

// Create the MySQL bind structure to store the data that we are going to insert
double temp = 0.0;
MYSQL_BIND bind;
memset(&bind, 0, sizeof(bind));
bind.buffer_type = MYSQL_TYPE_DOUBLE;
bind.buffer = (char *)&temp;
bind.buffer_length = sizeof(double);

// Bind the data structure to the MySQL statement
if (mysql_stmt_bind_param(stmt, &bind))
        {
        error_exit(con);
        }

//-------------------------------------------------------------------------------------------
// Insert multiple records into the database,
// with different data each time
/*for (int i = 0; i < 10; i++)
        {
        int adc6 = readADC(6);
        double y = (double)adc6;
        y = y / 1000.000;
```

```
        y = (y*100) - 50;
        temp = y;
        mysql_stmt_execute(stmt);
        sleep(1);
        }
*/
int adc6;
double y;
int fd;
char buf[MAX_BUF];
char val[4];
//for (int i = 0; i < 10; i++)
while(1)
{
snprintf(buf, sizeof(buf), "/sys/devices/ocp.3/helper.15/AIN%d", 6);
fd = open(buf, O_RDONLY);
if (fd < 0)
        {
        perror("ADC - problem opening ADC");
        }
read(fd, val, 4);
close(fd);
adc6 = atoi(val);
y = (double)adc6;
y = y / 1000.000;
y = (y*100) - 50;
printf("Temperature: %f\n",y);
temp = y;
mysql_stmt_execute(stmt);
sleep(1);
}

//------------------------------------------------------------------------------------------------

// Close the MySQL connection
mysql_close(con);

return 0;
}
```

Below is the code needed to generate a PWM signal from the Beaglebone board to the ESC:

```
import Adafruit_BBIO.PWM as PWM
myPWM="P8_13"
PWM.start(myPWM, 0, 400)
for i in range(0,20):
        DC=input("What Duty Cycle Would You Like (0-100)? ")
        PWM.set_duty_cycle(myPWM, DC)
PWM.stop(myPWM)
PWM.cleanup()
```

Appendix B: Test Plan Documentation

This section includes the results of the tests ran on and by various components of the monitoring system.

- **Motor Temperature**

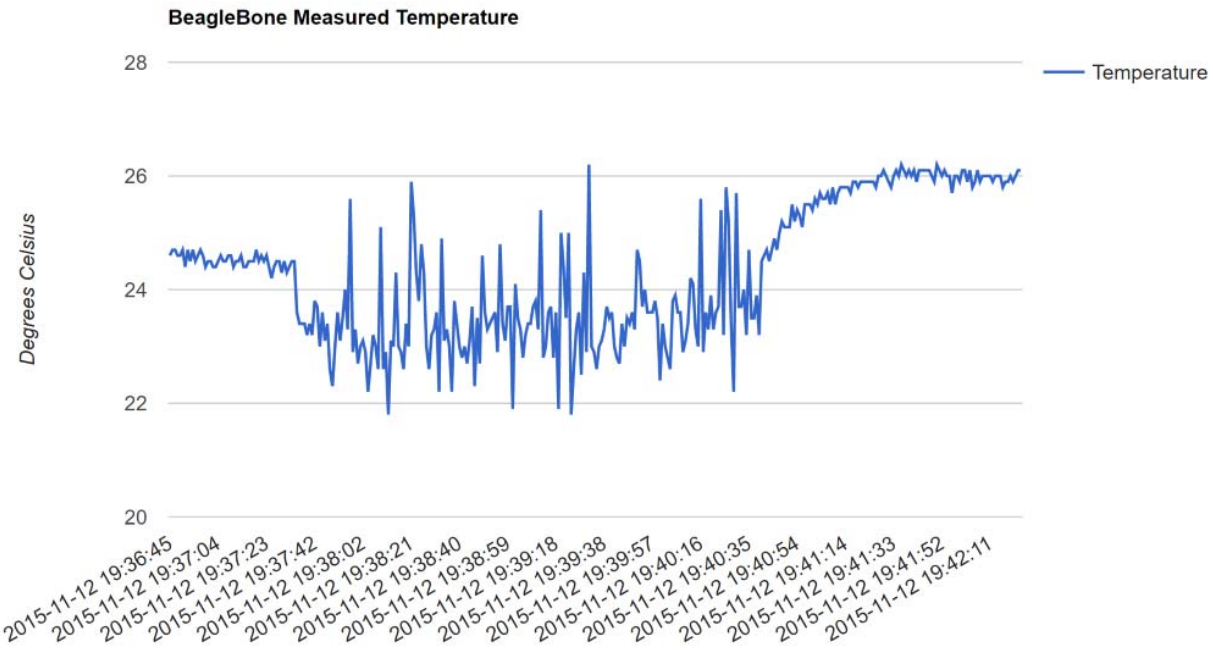


Figure B-1: Graph displaying data from motor temperature test

The above graph shows the data from the temperature sensor as it is placed inside the motor against the coils. The initial temperature slightly above 24 degrees Celsius shows the initial phase of the test where the motor is idle. It immediately drops and continues to rapidly jump from 22 degrees to 26 degrees during the motor's operation. As the motor shuts off, the jumping stops, and the temperature reads identical to the temperature before the motor ran. It immediately rises up until about 26 degrees where it levels out. Reasons for the jumpy data during the motor's operation are attributed to airflow or vibration causing contact issues with the sensor and the motor.

- **Three Phase Current**

The three phase current sensor test data was not saved as no meaningful data was ever produced. Multiple tests using the Beaglebone power supply, as well as a lab power supply, were performed on multiple circuits. The only thing learned from these tests were that the sensors required the lab power supplies to power them, as the Beaglebone's 5V supply was actually outputting 4.9V, and produced no output from the sensors.

- **Bearing/Vibration**

  The vibration sensors never arrived so there are no tests for those. The accelerometer output was never figured out. As well as the software issue discussed in the Test Plan section above, an issue with the accelerometer was that with the test rig currently being used, the motor will not move. This should point to the accelerometer not having any changes in readings.

- **Input Voltage**

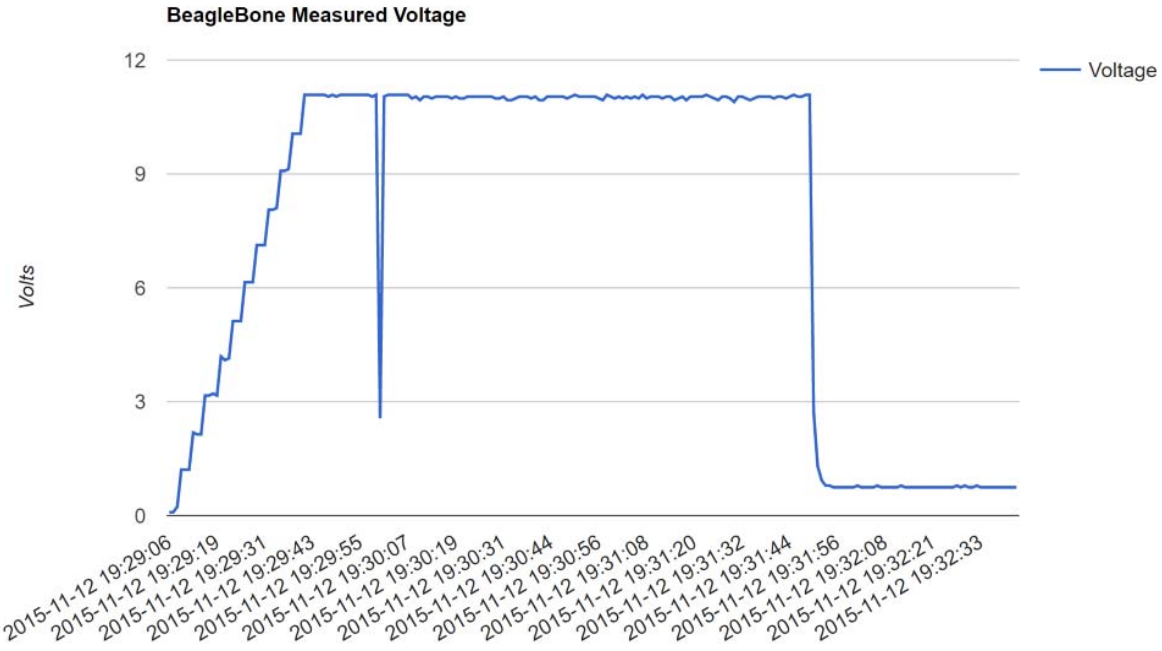


Figure B-2: Graph displaying data from motor input voltage sensor test

The above graph shows the results from a test on the input voltage sensor. The beginning shows the voltage as the power supply for the motor was incrementally scaled up in 1V increments from 0V to 11V. It sat at 11V until the operator realized the supply had to be shut off briefly to get the motor running, which is seen as the quick downward spike. It remained at 11V until the motor was shut off and the power supply was dropped down to 1V, where it remained until the end of the test.

- **Input Current**

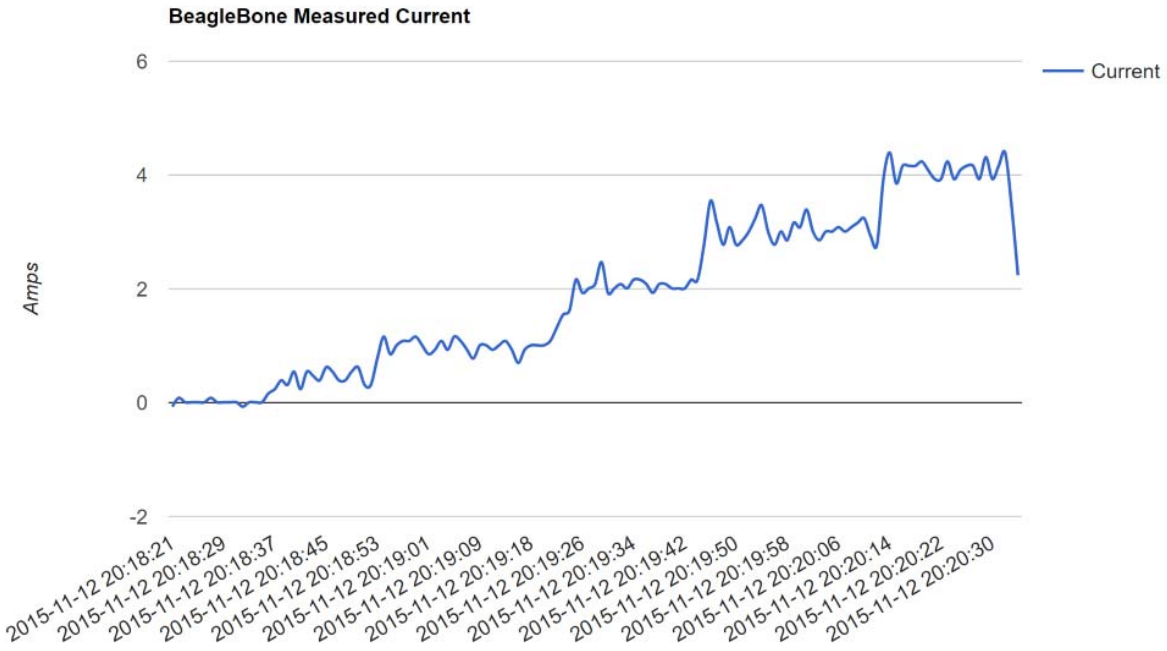


Figure B-3: Graph displaying data from first input current test

The graph above shows the input current as it went through a through scaling from from 0A to 4A in 1A intervals. The current data appears to be jumpier than the voltage data. For this test in particular, it was not clear whether or not it was the C++ code, the sensor, or the actual current being read that caused the sloppy looking step graph.

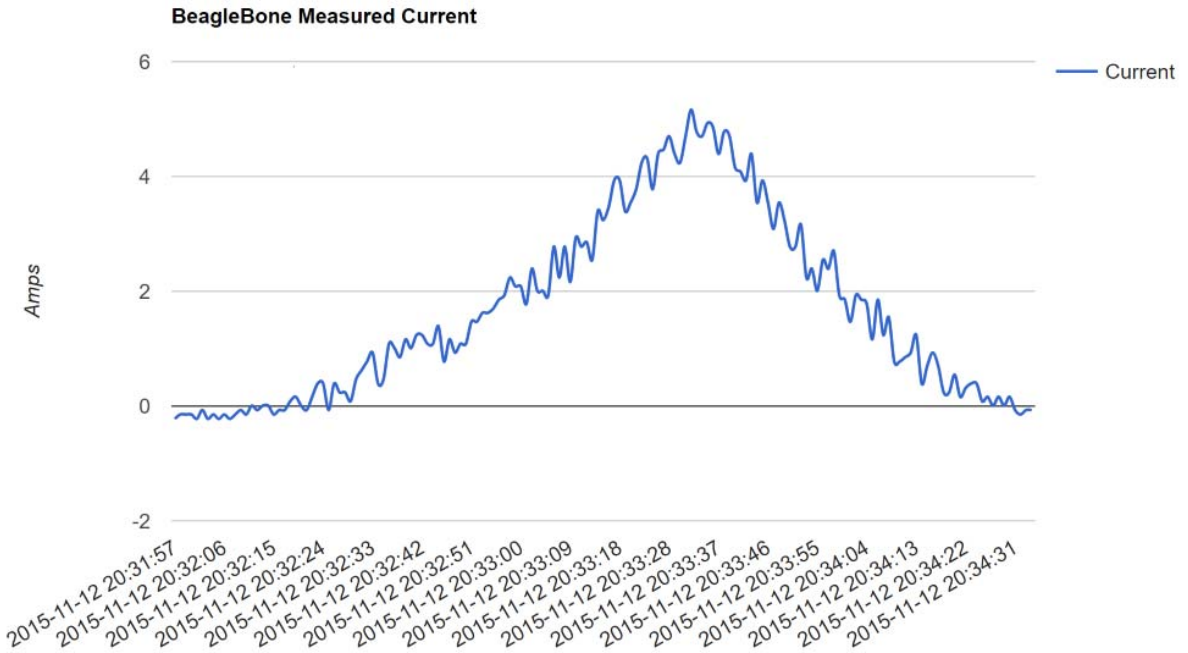


Figure B-4: Graph displaying data from second input current test

This graph shows the data from a test where motor was scaled up from 45% duty cycle up until 67% duty cycle. At 45% duty cycle, the motor does not run, and the 0A was recorded. At 67% duty cycle, the sensor recorded 5A, the maximum provided by the power supply. After hitting 5A, the sensor duty cycle was scaled back down.