AIAA Telemetry Transmission for NASA Rover Challenge

Melissa Jean Anderson

Follow this and additional works at: https://louis.uah.edu/honors-capstones

Recommended Citation
https://louis.uah.edu/honors-capstones/16

This Thesis is brought to you for free and open access by the Honors College at LOUIS. It has been accepted for inclusion in Honors Capstone Projects and Theses by an authorized administrator of LOUIS.
AIAA Telemetry Transmission for NASA Rover Challenge

by

Melissa Jean Anderson

An Honors Capstone

submitted in partial fulfillment of the requirements

for the Honors Diploma

to

The Honors College

of

The University of Alabama in Huntsville

4/20/18

Honors Capstone Director: David Fikes

Lecturer for Mechanical Engineering

Student (signature)  4/20/18

Director (signature)  4/22/2018

Department Chair (signature)  Date

Honors College Dean (signature)  Date
Honors Thesis Copyright Permission

This form must be signed by the student and submitted as a bound part of the thesis.

In presenting this thesis in partial fulfillment of the requirements for Honors Diploma or Certificate from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department, Director of the Program, or the Dean of the Honors College. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this thesis.

Melissa Anderson

Student Name (Printed)

[Signature]

Student Signature

4/20/18

Date
**Table of Contents**

Abstract........................................................................................................................................................ 2

Introduction ..................................................................................................................................................... 3

Design Overview ......................................................................................................................................... 4

I. Telemetry System ................................................................................................................................. 4

II. Ground Station....................................................................................................................................... 7

III. Mechanical System ............................................................................................................................. 8

Implementation ........................................................................................................................................... 9

I. Telemetry System ................................................................................................................................. 9

II. Ground Station....................................................................................................................................... 11

III. Mechanical System ............................................................................................................................. 17

Testing .......................................................................................................................................................... 19

Results ......................................................................................................................................................... 21

Conclusion ................................................................................................................................................... 23

References.................................................................................................................................................... 25
Abstract

This report details the design and manufacturing of an electronics system for the NASA Human Exploration Rover Telemetry Challenge. This challenge requires useful telemetry to be sent from the rover to a ground station where it can be viewed in real time. The telemetry transmitted for this project was collected using a GPS and an IMU. The data collected was displayed and plotted on a GUI created by MATLAB. A mechanical housing was also created that protected the system from outside influences like weather and impact, but was still lightweight and cost effective.

The system was successfully manufactured, but an accident on the day of the race prevented it from completing a run on the official course. The results presented in this paper are from a test of the system in a car on the campus of the University of Alabama in Huntsville. A map was provided to compare the results of the GPS mapping and an analysis of the GPS performance was performed. Several suggestions are noted on ways future students may improve upon this project to better assist the rover team.
Introduction

This year, UAH competed in the NASA Human Exploration Rover Challenge. This competition, which began in 1994, is an international race in which teams design and manufacture a rover that traverses a course at the US Space and Rocket Center. The object of the race is to complete the course quickly, while accruing points by completing obstacles and tasks. The competition also challenges teams to extra projects, like the Technology Challenge and Drive Train Challenge. These challenges are designed to push participants past simply designing a fast rover and are meant to introduce new aspects of engineering into the competition. One of these challenges is the AIAA Telemetry Challenge.

The AIAA Telemetry Challenge requires teams to design and manufacture a functioning, transmission system that attaches to the rover and captures data related to the race. The type of data to be captured is a decision made by the team, but it must be relevant to the race and help analyze how the team completes the course. During the race, the data must be transmitted to a ground station that displays the data in real time.

For the first time in many years, the UAH team chose to complete the Telemetry Challenge Award. The design and manufacturing of this design was conducted by Melissa Anderson, though input from the team was used to decide on overall functionality. This paper will describe the process of designing and manufacturing the system, from initial concept to final product, and include results from tests of the system.
Design Overview

The solution to this challenge consisted of three subsystems: the telemetry system, the ground station, and the mechanical housing for the telemetry system. The first step to completing this project was to determine what data should be collected by the telemetry system and transmitted to the ground station. After meeting with the rover team to discuss possible options, it was decided that the data to be collected was rover orientation, location, speed, and GPS strength. The GUI was set up to map the orientation and location, while also displaying the location, speed, satellite strength, and time. An IMU was used to collect the orientation data and a GPS was used for all of the other data.

I. Telemetry System

The rover sensor system is comprised of four major components: a radio, a GPS, an IMU, and a microcontroller. Trade studies were conducted to narrow down the possible options for each of the components. Some of the criteria used in the trade studies were weight, cost, and whether or not the component had a break out board. To fit within the given design parameters and budget, the components had to be lightweight and cost effective. A breakout board was desired for each component as it made assembly of the overall system much easier and negated the need to design a system specific PCB, which is expensive and often takes several attempts to get right.

The first trade study examined possible options for the microcontroller that runs the telemetry system. The two options included in this study were an Arduino Uno and an Arduino Nano. Previous experience with Arduino products and the availability of several resources for coding questions and support made the Arduino products desirable. The Arduino Nano was eventually chosen as the microcontroller. The Arduino Nano was cheap, light, and breadboard
friendly – which helped with testing. It also had a lower power consumption than the Arduino Uno. The main drawback to the Arduino Nano was that it has a smaller processing capability when compared to the Uno. After some analysis, it was determined that the Nano was capable of running the system in the desired capacity, so higher processing power was unnecessary.

The next decision was to determine the radio that would be used for both transmitting and receiving. To limit possible complications, it was decided that the same radio module would be used at both the ground station and on the rover system. The modules examined for the radio were the XBEE S2C and the XBEE-PRO S2C. The deciding factors in this trade study were transmit power and range. The rover course winds through outside areas of the Space and Rocket Center, where there are obstructions like buildings, trees, and rockets. A concern was that the obstructions would cause data to be lost. The XBEE-PRO S2C has a range of over 2 miles and a transmit power of 63 mW. This met the requirements of the system by a large margin, while the other module just barely met them. An issue brought forth with the XBEE-PRO S2C was that it has a current consumption of 120 mA, while the XBEE S2C only draws 33 mA. A power budget was used to determine that the system could support this draw with a reasonable factor of safety.

The two modules considered for the GPS were the U-Box NEO-6M and the MTK3339 chipset. Both modules came with breakout boards that would be easy to incorporate into the electrical design and could work on breadboards for easy testing. The U-Blox module is able to get its satellite fix 7 seconds quicker than the MTK, but the MTK has a 5Hz higher update rate and has a 20 mA lower current draw than the U-Blox. To mitigate the fact that it takes over 30 seconds for the MTK chipset to achieve its fix, an extra step was added into the ground station system. The step requires the user to signify the beginning of the race so data can start being
collected. By doing this, the telemetry system can be turned on prior to the beginning of the race, giving the GPS time to achieve fix.

The final component decision that had to be made was on the IMU, or inertial measurement unit. This device can be used to measure the specific force, angular rate, and the magnetic field surrounding the system. The IMU sensors that were examined all had 9 DOF, which means it collects a combination of accelerometer, gyroscope, and magnetometer data. Two IMU’s were compared, one worked with I2C protocol and the other with SPI. The SPI sensor, the LSM9DS0 module, was chosen as it also had a lower power consumption than the other model. Previous experience with SPI also made the LSM9DS0 module more desirable.

A power budget was performed for this system to ensure that a 9V battery would be able to support this system for at least an hour of run time. Tables 1 and 2 show the power consumption for the rover system. It should be noted that the power will be drawn through a linear regulator, which will cause a significant amount of loss. As the tables show, the system will be able to run for over an hour with no power concerns.

<table>
<thead>
<tr>
<th>Device</th>
<th>Current Draw (mA)</th>
<th>Power Required (mW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino Nano</td>
<td>19</td>
<td>171</td>
</tr>
<tr>
<td>GPS</td>
<td>20</td>
<td>180</td>
</tr>
<tr>
<td>IMU</td>
<td>6.45</td>
<td>58.05</td>
</tr>
<tr>
<td>Radio</td>
<td>120</td>
<td>1080</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source</th>
<th>Power (mWh)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supply</td>
<td>9V Battery</td>
</tr>
<tr>
<td>Consumed</td>
<td>Sensors</td>
</tr>
<tr>
<td>MARGIN:</td>
<td></td>
</tr>
</tbody>
</table>

The code for this system was constructed using several libraries to simplify the main code. The libraries were provided by the manufacturers of the breadboard components and
included basic functions for parsing GPS data, reading IMU data, and converting the raw IMU data. Another library used for this project was Software Serial, which was used to create a serial port for the XBEE and the GPS.

II. Ground Station

The ground station for this project utilized the GUI and mathematical capabilities of MATLAB to plot and display the data sent from the telemetry system. The ground station consists of a laptop with running MATLAB code, an XBEE, and an antenna that increases the range of the XBEE. At the end of the program, the data is exported to excel where it can be viewed later and even run back through another program that can show the race in real time.

After discussing the ground station with the team, it was decided that the data that would be displayed was the latitude, longitude, speed, number of satellites, and time. The latitude and longitude are displayed in decimal form, with a negative meaning either south or west. The latitude and longitude was mapped so the path of the rover could be tracked from the starting line. A box was also mapped that tilted with the rover. To complete the main portion of the ground station, a button was placed under the displayed data to end the race. Text boxes and buttons were made using the uicontrol function available on MATLAB.

To complete this project, two MATLAB toolboxes needed to be downloaded to the laptop that was running the ground station. The first toolbox needed was the Mapping Toolbox, which allows the user to manipulate maps of the world and compare population sizes by drawing information from the large database of information stored. This toolbox was used to create geopoint objects from latitude and longitude data and to graph these data points. The other toolbox used was the Aerospace Toolbox. This toolbox was necessary for converting the IMU data into angles using the angle2dcm function, which was crucial to plotting the rover tilt.
III. Mechanical System

When designing the mechanical housing for the power system on the rover, three major considerations had to be taken into effect: weight, protection, and cost. The telemetry system is comprised of several small components that can break if a strong enough force is applied. This force could be achieved on the lunar crater obstacle or the boulder pit. The gravel and sand particles may also kick up dust and coat the electronics which could cause interference in the GPS signals. A box was chosen for the mechanical design as it can protect the device. Foam pieces were placed inside the box to keep the system from jostling during the competition.

The design of the box was modified when weight constraints were placed on the system. Though the box is small, the team wants to meet a goal of a total rover weight of 130 pounds, which they were dangerously close to when the housing was made. To limit the weight of the telemetry system, slits were placed on the sides of the box and cutouts were placed on the lid. The cutouts and slits were then covered with packing tape so the components remained protected. The cutouts on the top of the box also helped to decrease the GPS interference by decreasing the amount of obstructions in its view.

Once the design was finalized, materials had to be selected. Since a large portion of the budget for this telemetry project was spent on the electrical components, the mechanical system had a limited budget. As a result, the finished product for the mechanical design ended up costing nothing. All materials used were scraps from other portions of the rover. The main housing was 3D printed with PLA already available to the team. The foam placed beneath the electrical components was left over from the sample collection box, as was the metal rod that connected the lid to the box. The tape covering was packing tape bought by a previous rover team.
Implementation

I. Telemetry System

The circuit for the telemetry system was designed using the schematics for each component and a breadboard. The components were first arranged on the breadboard before being officially soldered into place on a PCB. An XBEE shield was chosen to connect the XBEE module and the Arduino Nano. This shield was designed by RobotShop. The GPS module was connected to the D2 and D3 pins for transmitting and receiving data. The IMU was connected to analog pins A4 and A5. All of the modules receive power through the Arduino power pins. The Arduino itself is powered using a 9V battery, connected to the Arduino via a battery clip. A switch was soldered into the circuit as well to make turning the system on and off slightly easier. Figure 1 shows the electrical diagram for this system.

![Telemetry System Electrical Diagram](image)

Figure 1: Telemetry System Electrical Diagram

The code for this program begins with the configuration code that sets objects for the serial ports, the GPS, and the IMU. Then the program moves to initialize the serial ports at a baud rate of 9600, which matches the baud rate of the ground station XBEE. The settings for the GPS are then set for the basic data parsing and the update rate of 1 Hz. The basic data parsing only parses two NMEA sentences: RMC and GGA. These two sentences include all the desired telemetry. The program then configures the IMU sensor, after checking to be sure it is connected.
It then sends a message to the ground station that the system has finished initializing. Figure 2 shows this portion of the code.

```cpp
//Initialize the serial ports
XBEE.begin(9600);
GPS.begin(9600);

//Initialize the GPS settings
GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMC+GGA);
GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_1HZ); // 1 Hz update rate

// Initialise the IMU
if (!ism.begin())
{
    //Alert that the IMU cannot be detected
    XBEE.println(F("Oops, no LSM9DS0 detected"));
    while (1);
}
configureIMU();
XBEE.println("O"); //Send initialized message to ground station
```

Figure 2: Telemetry System Initialization

After initializations, the code enters the loop function that reads in the characters of the NMEA sentences and parses them. Data from the parsing is then saved to the GPS object. When the parsed GPS data indicates the GPS has been fixed, a message is sent to the ground station. Figure 3 shows the code for this portion of the system.

```cpp
//Read and parse the GPS NMEA sentences
char c = GPS.read();
if (GPS.newNMEAReceived()) {
    if (!GPS.parse(GPS.lastNMEA()))
        return;
}

//Send GPS fixed message to grounds ttation
if (GPS.fix && !sent) {
    sent = true;
    XBEE.println("@");
}
```

Figure 3: GPS Parse and Fix

Once the GPS has fixed, the final portion of the code for the rover system begins. Every 1 second (or 1000 milliseconds), the program enters the ‘if’ statement shown below in Figure 4.
Inside this statement, the data from the IMU is collected and sent out in a packet with the other information to the ground station. The order of the data when it is transmitted is crucial to how the ground station interprets it. The data is sent in the following packet: <longitude direction, latitude direction, number of satellites, accelerometer x, accelerometer y, accelerometer z, speed, longitude magnitude, latitude magnitude>. The program repeats the loop function until it is turned off.

```c
//Every second send this data so long as the GPS is fixed
if (millis() - timer > 1000 & & GPS.fix) {
    timer = millis(); //Reset the timer

    //Read data from IMU
    sensors_event_t accel, mag, gyro, temp;
    lsm_getEvent(&accel, &mag, &gyro, &temp);

    //Send data in packet form
    XBBF print(GPS.len);
    XBBF print(", ");
    XBBF print(GPS.lat);
    XBBF print(", ");
    XBBF print((int)GPS.satellites);
    XBBF print(", ");
    XBBF print(accel.acceleration.x);
    XBBF print(", ");
    XBBF print(accel.acceleration.y);
    XBBF print(", ");
    XBBF print(accel.acceleration.z);
    XBBF print(", ");
    XBBF print(GPS.speed);
    XBBF print(", ");
    XBBF print(GPS.longitude, 8);
    XBBF print(", ");
    XBBF print(GPS.latitude, 8);
}
```

Figure 4: Telemetry Packet Creation

II. Ground Station

The code for the ground station can be separated into two functionalities: GUI creation and data handling. The GUI creation consists of code that creates the dialog windows and the text, buttons, and graphs within the dialog windows. The code below in Figure 5 shows the code
to create a dialog box and text on that dialog box. These lines of code were used in varying
fashions throughout the program.

```matlab
fig1 = figure('Position', [500, 300, 250, 75]);
uicontrol('Style', 'text', 'String', 'Waiting for Arduino to turn on...',
'Position', [0 25 250 25], 'HorizontalAlignment', 'center',
'FontSize', 12);
```

Figure 5: Dialog Box Code

A series of dialog boxes were made to make using the ground station user friendly. The
dialog boxes mark several different sections of the program. The first dialog box appears after
the ground station has opened the serial port for the XBEE and alerts the user that the ground
station is waiting for the Arduino to turn on. Once the message is received from the telemetry
system that all sensors are initialized and in working order, the window closes and another
opens. This dialog box informs the user that the ground station is waiting for the GPS to attain a
fix, which requires five satellites to be in view. After another message from the telemetry system
is received, the window closes. A final start up window is displayed that contains a single button
labeled ‘Begin race!’ To begin the telemetry display and collection, the user must click the
button. This will cause the window to once again close and the main GUI window will open.

Figure 6 shows the startup dialog boxes.

![Figure 6: Startup Dialog Boxes](image)

Creating the main GUI took more precision. Text boxes were aligned through several
implementations of the code. A total of 12 uicontrols were used as textboxes to display the text
data in real time. Another uicontrol was utilized to create a button that ends the program. When the button is clicked, the XBEE port and dialog box are closed and the data is exported to excel. Figure 7 shows the window for the main GUI.

![Main GUI Window](image)

**Figure 7: Main GUI Window**

The subplot function was used to evenly space the plots in the above window. The GPS data was plotted using the function geoshow, which takes latitude and longitude data and creates a geopoint for each data point. These geopoints are then plotted. The IMU data was plotted using a 3D plot with a set range of [-2 2] for each axis. Figure 8 shows the code for these plots.

```matlab
subplot(1,3,2)
geoshow(latArray, longArray);

subplot(1,3,3,'Parent',fig4);
plot3(P(:,1),P(:,2),P(:,3));
axis([-2 2 -2 2 -2 2]);
xlabel('X');
ylabel('Y');
zlabel('Z');
grid on;
```

**Figure 8: Inserting Plots into the Window**

The second portion of the GUI focused on data handling. This includes reading the data from the XBEE, parsing the data, and preparing it to be presented on the GUI. Reading the data
from the XBEE requires only a few lines of code. The first step is to set up the serial port for the
XBEE on the correct port. The Serial functions serial and fopen are used to do this. Serial creates
the port and fopen opens the port for communication. The XBEE is already programmed to
communicate only with its telemetry system counterpart so it will only receive transmissions
from that source. Figure 9 shows the code for this process.

```matlab
XBEE = serial('COM3');
fopen(XBEE);
```

Figure 9: Opening Serial Port in MATLAB

Once the port is open, the data must be set up to read data continuously while checking
for specific messages. To do this, a while loop was created with a condition statement to stay in
the while loop so long as the serial port was open. An if statement inside the while loop checks to
see if any new data is available on the XBEE’s buffer. If there is new data to be read, it is
scanned in. In some cases the data being read in must be a specific character that denotes a
checkpoint has been reached. Instances of these are when the Arduino is turned on and the GPS
has achieved a fix. To accomplish this task, the first character of the received message is
compared to the specific character assigned to that task. If the characters match, then the
program breaks out of the current loop and moves on. The code for this progress is shown in
Figure 10.

```matlab
while (strcmp(XBEE.Status,'open'))
    if(XBEE.BytesAvailable)
        string = fscanf(XBEE);
        end
    if(string(1) == 'o')
        break;
        end
end
```

Figure 10: Reading in Messages in MATLAB
For the majority of the program, the messages received by MATLAB need to be parsed and not compared to other messages. To do this, the data is read in and all but the last two characters are saved to be parsed. The last two characters of all the messages received by the ground station are return carriage and new line. This was done to easily denote when a message ends and a new one begins. The data is then split, using the commas as delimiters, into a cell array. The latitude and longitude are split once more. NMEA sentences combine degrees and minutes into one number, where the first two numbers are the degrees and the rest are the minutes. This data was therefore split into degrees and minutes and placed at the end of the cell array. Figure 11 shows the code for parsing the data as described above.

```
string = fscanf(XBEE);
string = string(1:end-2);
h = strsplit(string,',');
h(10) = extractBefore(h(8),3);
h(11) = extractAfter(h(8),2);
h(12) = extractBefore(h(9),3);
h(13) = extractAfter(h(9),2);
```

Figure 11: Parsing Telemetry Data in MATLAB

After the message is parsed, most of the data needs to be altered or graphed in some form. To manipulate them in the necessary ways, all of these data points are transformed from strings to numbers using the str2double function. Before the data is modified further, an error check was put in place to weed out corrupted GPS data. Sometimes the GPS doesn’t receive all parts of a message due to interference and the data is then parsed incorrectly. The easiest way to determine whether or not a mistake has occurred is by testing to see if there are exactly 13 characters in the latitude and longitude sections. Testing showed that when lines got corrupted, these readings would always drop to only 10 characters. Therefore, if the latitude and longitude do not have exactly 13 characters, that data point is ignored.
If the data is not corrupted, it goes on to be modified. The latitude and longitude are modified in two ways. First, the direction is checked to see if the GPS data needs to be negative or not. Then the latitude and longitude values are changed from degree-minutes to degrees in decimal form. The speed is then changed from knots per sec, which is what the NMEA sentences send it as, to miles per hour. The previously discussed process is shown in Figure 12.

```matlab
for i = 4:13
    temp(i) = str2double(h(i));
end

if strlen(h(8)) == 13 && strlen(h(9)) == 13
    if isequal(h(1), 'S')
        temp(10) = temp(10) * -1;
    end
    if isequal(h(2), 'W')
        temp(12) = temp(12) * -1;
    end
    latTemp = dm2degrees([temp(12) temp(13)]);
    longTemp = dm2degrees([temp(10) temp(11)]);
    temp(7) = temp(7) * 1.150778;
end
```

Figure 12: Data Modifications in MATLAB

The graphing of the GPS data is done by adding each latitude and longitude data point to an array every three seconds. Through testing, it was determined that an update rate on the graph of every three seconds did not slow down the MATLAB program and still allowed a clear picture of the path of the rover. Figure 13 shows the code for this process.

```matlab
if count == 3
    count = 0;
    latArray = [latArray; latTemp];
    longArray = [longArray; longTemp];
end
```

Figure 13: GPS Array Additions

Setting up the IMU data required using the atan2 function in MATLAB. This function returns the tangential angle between two values in the range of \((\pi, -\pi)\). This range allows the
angles to be converted in the full range of 360°, as opposed to the 180° available when using the original tan function. To tilt the cube using the accelerometer data, the angles between the x and y accelerometer values and the resultant must be calculated. These angles represent the roll and pitch, respectively. The yaw was assumed to be 0 for the purposes of this project. The angles then needed to be transformed from angles to a direction cosine matrix to map the information. Figure 14 shows the code for this portion of the program.

```
pitch = atan2(-temp(5), (temp(4)^2+temp(6)^2)^0.5);
roll = atan2(temp(4), (temp(5)^2+temp(6)^2)^0.5);
yaw = 0;
dcm = angle2dcm(yaw, pitch, roll);
```

Figure 14: Determining the Direction Cosine Matrix in MATLAB

Once the direction cosine matrix was determined, it was multiplied by a unit cube and centered on the origin. To keep the points centered on the origin, the cube’s location was placed off center before the direction matrix was included. It was then placed back at the center. Figure 15 shows the code for this process.

```
P = [A;B;F;H;G;C;A;D;E;H;F;D;E;C;G;B];
P = P-Cent;
P = P*dcm;
P = P+Cent;
```

Figure 15: Tilting the Cube in MATLAB

**III. Mechanical System**

Figure 16 shows the overall design for the mechanical housing of the telemetry system. The box shown is 6.5 x 4.125 x 1.625 in. with 0.25 in. thick walls, except for the bottom which is only 0.1225 in. thick. The side slits are spaced 0.5 in. apart and the front and back slits are spaced 0.375 in. apart. All slits are 0.25 in. wide. The lid is 6.5 x 4.125 x 0.25 in. The three center cutouts are the same size and spaced 0.375 in. apart. The first and last cutout are spaced
the same distance, but are slightly larger to accommodate for the XBEE antennae. The orientation of the telemetry system when placed within the box had yet to be determined when the box was designed, so both sides were designed to fit the antennae. The lid and box are connected via a steel rod with a diameter of 1/16”.

Figure 16: Mechanical Housing Design

The telemetry system was placed in the back corner of the rover so it was out of the way of the drivers and had the least amount of interaction with moving parts. The box was attached to the rover using industrial strength Velcro.
Testing

The creation of this project involved testing the parts of each system in increments. The first aspect that was tested was the circuitry. All of the components were purchased in a breadboard compatible form, so the electrical circuitry was tested using breadboards. After all connections were confirmed, the components were then placed onto a blank PCB and soldered by one of the team members. A voltmeter was used to make sure that all the connections were soldered and that no lines were crossed. Once the circuit board was completed, testing of the code could begin.

The code was tested in increments. First implemented was the XBEE communication from the rover system to the ground station on MATLAB. This was tested first to ensure that the rest of the system could be run in full transmission mode. This helped pinpoint transmission rate limitations within the system. These limitations are a result of the GPS data collection and parsing, which was implemented next. To ensure GPS data is not missed or misinterpreted, the GPS collects at a rate of 1 HZ and is then parsed. Using the provided libraries made the GPS and IMU collection simple and little testing was required for that portion of the project.

Once all of the transmission and collection was functional and tested, the GUI was created. The GUI was created and tested in increments, starting with creating the dialog boxes and the transitions from each dialog box. Testing these revealed that the messages sent from the Arduino had new line and return carriage characters at the end of each line, which made implementing the data parsing easier. After the parsing and exporting were written and tested, full system tests began. While these first few runs did not include the graphing components, they were useful in tracking the sentence corruption in the GPS readings and testing the range of the XBEE communication. Range was tested by walking around campus with the telemetry package.
and ground station separated by a large distance. The only known limitation on the range is if a building came in between the telemetry system and the rover. The radio signals had trouble reaching the ground station with that interference, which was expected.

The GUI graphing was tested in increments. First, the IMU graphing was created and run on its own without being implemented directly into the rover system code. The system code had at that point become more complex and simple testing provided a quicker and easier way to find issues. The testing showed a mislabeled axis on the graph, which has since been fixed.

The GPS plotting was more difficult to test as it required being outside. The GPS cannot achieve a fix inside a building, so the only time it could be tested was when the weather cooperated. This was tested once again by walking around campus. The limitations found by testing this portion of the code was once again related to interference. Buildings and trees affected the GPS readings much worse than they did the radio signals. This was also expected as the GPS needs to have a clear view of the satellites to work properly.

Two weeks before the competition date, the telemetry system was able to be tested on a full system scale, including the mechanical housing. It was run on the rover a total number of 7 times, with varying results. This testing revealed an issue with the entire system tilting within the mechanical housing because of the weight of the antennae. This threw off the IMU readings. This was rectified by modifying the foam that the electronics sat on. The foam was cut in some places and built up in others, so that there was less room for the system to tilt. It also revealed that the XBEE would occasionally come unplugged from the system, which ceased all communication to the ground station. This was fixed by securing the XBEE with string to keep it on the shield.
Results

Unfortunately, moments before the race on August 12th, the battery lead to the rover system broke free from its soldered joint. There was an attempt to fix the break, but a strong enough connection could not be made in the few minutes between the break and the race. Therefore, the telemetry package was unable to be run on the course when the team raced. The break was fixed that evening, but the race on Saturday was canceled due to weather, so the telemetry package was not implemented during an official race. The results presented in this section are therefore the results from a test run of the system. The results from this test are shown in Figure 17.

For this test, the system was driven around a parking lot on UAH’s campus. The maximum speed on this run was 9.04 miles per hour. The total test run lasted for 4 minutes exactly. This length of a run should have resulted in 240 data points. However, examination of the data file shows only 219 saved data points. This discrepancy can be attributed to the data filter implemented into the system that ignores erroneous readings from the GPS. The erroneous readings are more common in bad weather conditions and around buildings and trees. This particular test was run during a storm in an area with some trees. These results showed that 21
total readings were ignored because they were corrupted. This is 8.75% of the total readings taken. It is safe to assume that in less stormy conditions, fewer data points would have been lost.

Figure 18 can be used to compare the graph with the actual path. The test was run in the parking lot outside of the Business Administration Building and Wilson Hall. The red on the map shows the exact path taken. A visual comparison shows a similar model on the graph completed by the ground station.

![Figure 18: Test Run Path](image-url)
Conclusion

This telemetry project was overall successful. A system was designed and manufactured that took speed, latitude, longitude, satellite number, and IMU information. Data was transmitted over a large distance to a ground station that held a useful GUI and saved data for future review. It met the goals set by the advisor, David Fikes, and the wishes of the team it was designed for. Though the system didn’t get to race on the actual course, it was used to test the rover and was helpful in planning strategies for the drivers on the course.

Though the system was completed successfully, there are several ways that a future member of the Rover Senior Design class could improve upon and alter this design. The first is within the IMU plotting. Currently, only the accelerometer data from the IMU is being used to plot the tilt. While the results of this are acceptable, there is a way to make them more accurate within the confines of the given module. The LSM9DS0 has 9 degrees of freedom, which means it includes a gyroscope and magnetometer as well as an accelerometer. Incorporating the results from the gyroscope into the plot can make the tilt even more accurate and the magnetometer can be used to calculate the yaw. It was decided by the team that the yaw was not entirely necessary in the competition, but it would make the IMU plot more accurate.

Another way to improve this project would be to find a way to calculate the overall distance traveled by the rover. During testing of the rover, the team realized that would be useful in determining whether their drivers were capable of finishing a 0.5 mile course in the allotted minutes. A final change that could be made is to implement a buzzer on the telemetry system that is paired to a button on the ground station. That way, when the team hits the six minute mark and needs to rush to the finish line, the ground station control can alert them with the buzzer instead of trying to yell it at them as they ride on the course. The knowledge learned and resources
accrued will be passed along to the next Rover Senior Design class in the hopes that it will help the program improve with each iteration.
References

“Adafruit LSM9DS0 Library.” Github, 2018, github.com/adafruit/Adafruit_LSM9DS0_Library.


