DATA COLLECTION AND ANALYSIS TECHNIQUES FOR SOLAR CAR TELEMETRY DATA

Michael Rouse, Miranda Sauer, and Kurt Kosbar
Telemetry Learning Center
Department of Electrical and Computer Engineering
Missouri University of Science and Technology
Rolla, MO, USA

ABSTRACT

Data collected from a solar car is monitored in real-time, which allows for intelligent decision making, efficient debugging, and high-quality testing for solar car teams. This paper compares three databases (MySQL, PostgreSQL, and MongoDB) to determine the optimal database system that should be used at solar car competitions. Each database system was tested using simulated solar car data to measure read and write speeds, and quality of performance on a low-power computer. Data were analyzed and displayed with custom interfaces to improve the user experience at solar car competitions.

Keywords: Solar Car, MySQL, PostgreSQL, and MongoDB

INTRODUCTION

The Missouri S&T Solar Car Team designs and constructs solar-electric vehicles that compete in the Formula Sun Grand Prix [1] and the American Solar Challenge [2]. Since 1991, the team has designed and manufactured thirteen solar cars [3]. Each vehicle has improved upon the previous design with the intent of becoming more efficient and reliable.

As technology has advanced, solar car systems have gotten more complex and the competitions have become more competitive. There is an increased demand to have telemetry data easily accessible during competitions. Data from the vehicle allows for intelligent decision-making regarding strategy required to succeed at competitions. An easy-to-understand user interface plays an important role in this process.

The Missouri S&T Solar Car Team currently utilizes a database to save and protect information. This stored information can be retrieved and reviewed after collection to further enhance the team's competition strategy. Stored data is also used to review unique situations that occurred during competitions or testing. Situations such as mechanical or electrical failures are better understood after analyzing stored telemetry data.

This paper compares the team's current method of data collection that uses the MySQL database system, with two new potential methods that could provide more efficient data storage. PostgreSQL [4] and MongoDB [5] were chosen as potential database solutions because of their popularity. All three are backed by large open-source communities.

BACKGROUND

A CAN bus (ISO 11898) [6] is used on the solar car for communication between systems, e.g., turn signals and motor control. The CAN bus provides a robust communication method that is easy to expand, understand, and implement. During competition (Figure 1), messages are transmitted from the solar car to a support vehicle or base station where they are processed and used to visualize the current state of the vehicle.

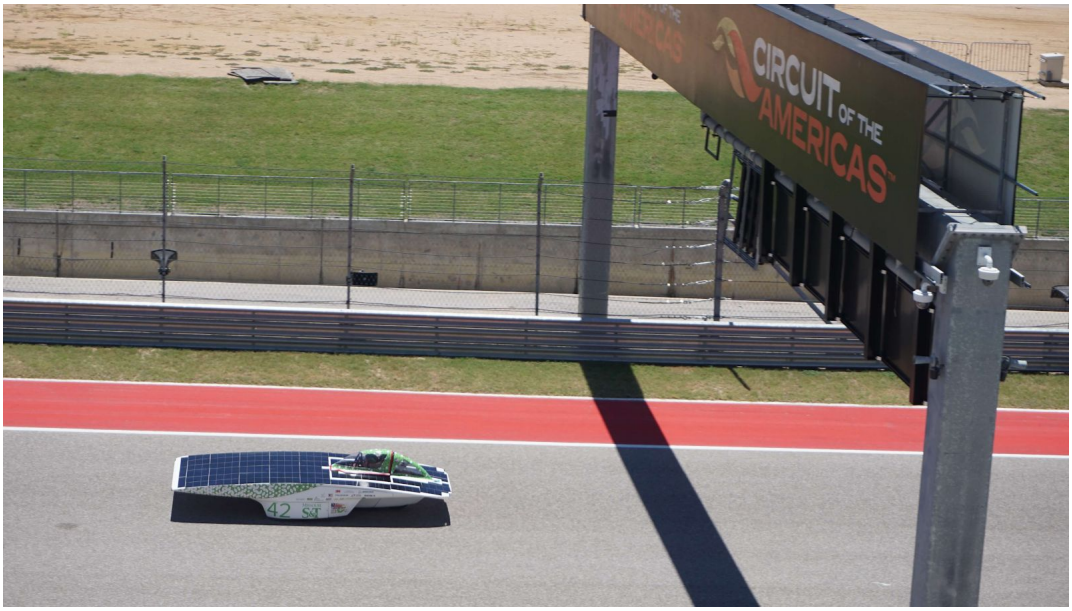

Figure 1 - *Solar Miner* in the 2017 Formula Sun Grand Prix

Every message sent on the CAN bus is converted to UDP Multicast [7] packets using a Tritium CAN to Ethernet Bridge [8]. For non-line-of-sight distances, packets are parsed by software and sent to a remote server over a 3G connection to be stored in a database. For line-of-sight distances, packets can alternatively be sent over an IEEE 802.11N [9] Wi-Fi network where laptops with the telemetry software can receive the packets to display telemetry data.

In previous years telemetry data has been saved in a text file in comma-separated value (CSV) format, or in a database. Previous databases have been overly complicated and difficult to expand as new data points are required.

DATA STRUCTURE

Data sent over a CAN bus is sent in the form of packets composed of an 11 byte identifier and 8 bytes of data. This results in a total of 19 bytes in each message, with a new CAN message roughly every 10 milliseconds. For the sake of making a generic database, a single-table schema (Table 1) storing the raw CAN message bytes is preferred over a multi-table schema (Table 2) which stores values based on the state of all systems on the solar car.

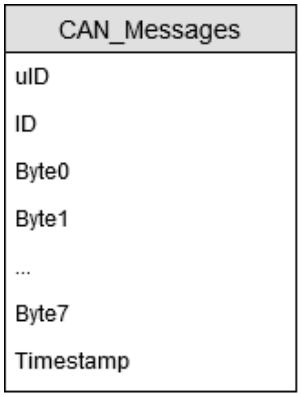

Table 1 - Single-Table Schema

The single-table structure is more favorable when compared to a more elaborate relational structure for many reasons. A more complicated structure would log the current state of the solar car at certain intervals. This would be separated in tables all joined by a common table that stores the timestamp of the entry. The structure of this database would look similar to a multi-table schema shown in Table 2.

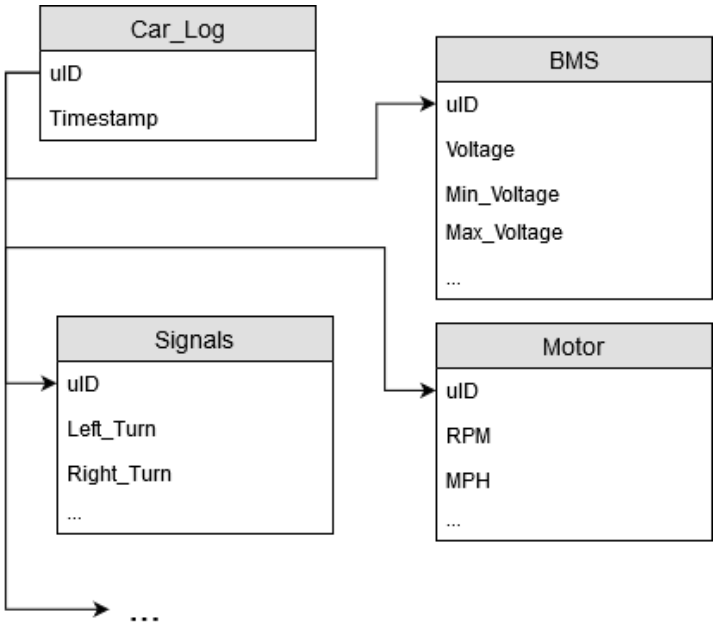

Table 2 - Multi-Table Schema

A single-table method is advantageous when compared to a multi-table method due to the code and database structures being simpler. Rather than having to keep track of the entire state of the solar car, the code just needs to store each CAN message in the table. If a new message is required on the solar car, no changes to the database schema are required when using the single-table method. Similarly, replaying CAN messages in real time is only possible using the single-table method.

MongoDB is different from MySQL and PostgreSQL because it is a NoSQL database system that stores data in JSON-like documents [10, 11]. For the single-table implementation it will use the following JSON schema:

```
{
        "id": 0x000,
        "data": [0, 0, 0, 0, 0, 0, 0, 0],
        "timestamp": "01/01/1970 00:00:00 AM"
}
```

TELEMETRY VISUALIZATION

Telemetry software is designed to display real-time data to team members that then make decisions on how the solar car should be driven (e.g., speed). The telemetry software will ideally be extensible, portable, and easy-to-understand. Currently, a Java Swing user interface is used by team members to create this software. The software uses Reflections [12] for automatic discovery and loading of classes at runtime. This feature allows quick additions to the user interface that displays the telemetry data. All of these factors lead to a robust and cross-platform software solution (Figure 2) that is easy to customize and extend.

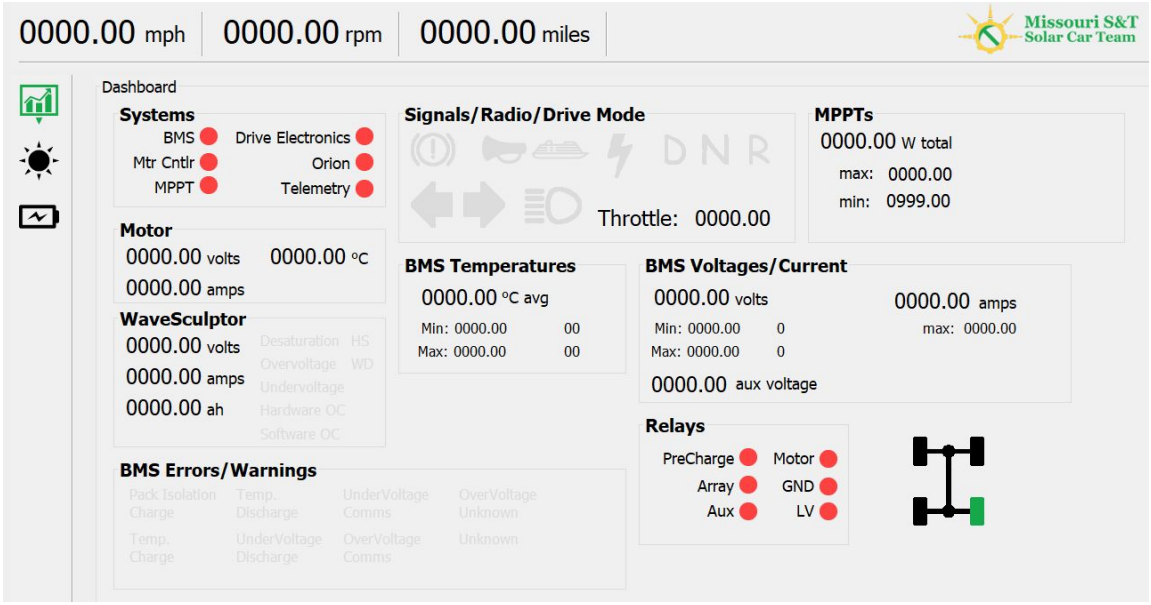

Figure 2 - Telemetry Software User Interface

Analysis of data can be done in both real-time and after-the-fact. Figure 3 is an example of a graph that can be produced from telemetry data that is collected from the solar car. It shows speed and motor current over time. The motor current can be seen fluctuating with little variation in speed. From this, it can be concluded that the car was driving in a hilled area, which was the case.

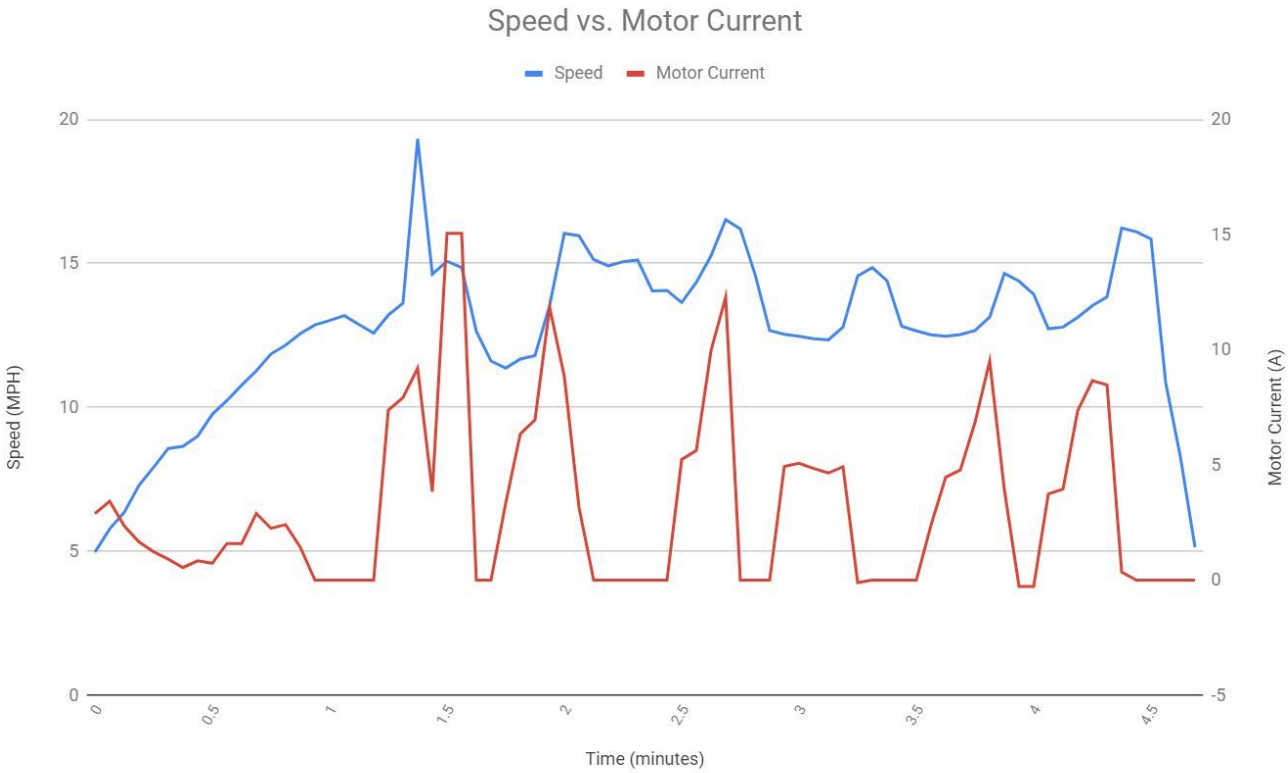

Figure 3 - Speed vs. Motor Current Graph Example

DATABASE PERFORMANCE ANALYSIS

For database testing, 255 entries were inserted into each database system. To simulate varying loads on the database, queries were performed at different intervals and the query time was averaged based on the interval.

Figure 4 shows an interesting trend that was reproduced several times. As the interval gets larger between operations, the average query time gets longer for PostgreSQL and MySQL, and peaks around 20 milliseconds.

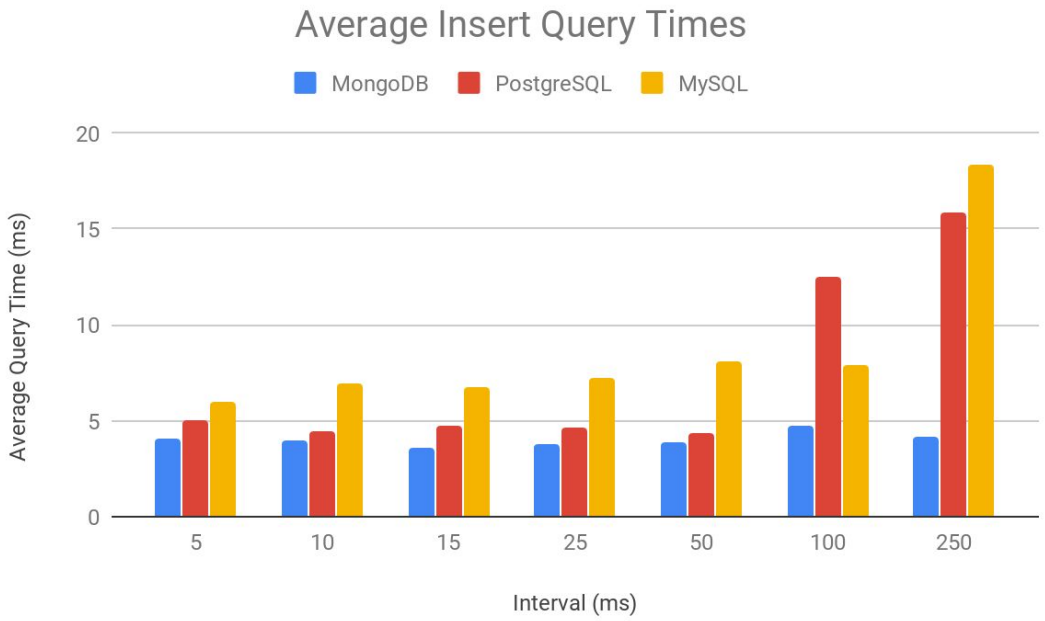

Figure 4 - Average Insert Query Times (ms)

After analyzing the data, it was apparent that both MySQL and PostgreSQL are rather similar in average insert query time. MongoDB stood out as almost twice as fast as the others. However, when MongoDB was slower, it was much slower. Figure 5 shows the maximum recorded insert query times for all the database systems. MongoDB was observed to be 15 times slower than PostgreSQL's slowest query, and six times slower than the slowest MySQL query.

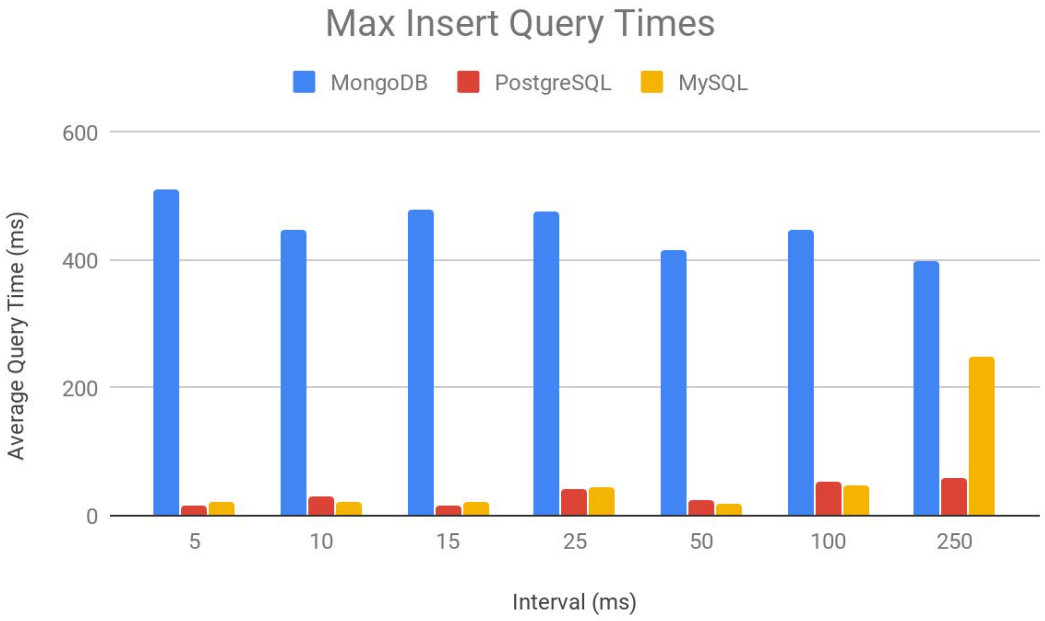

Figure 5 - Max Insert Query Times (ms)

To measure the performance of reading from the database, 255 entries were pulled from each database system. The time for that action was measured and averaged.

For all the database systems tested there appears to be minor fluctuations in the times for read queries (Figure 6).

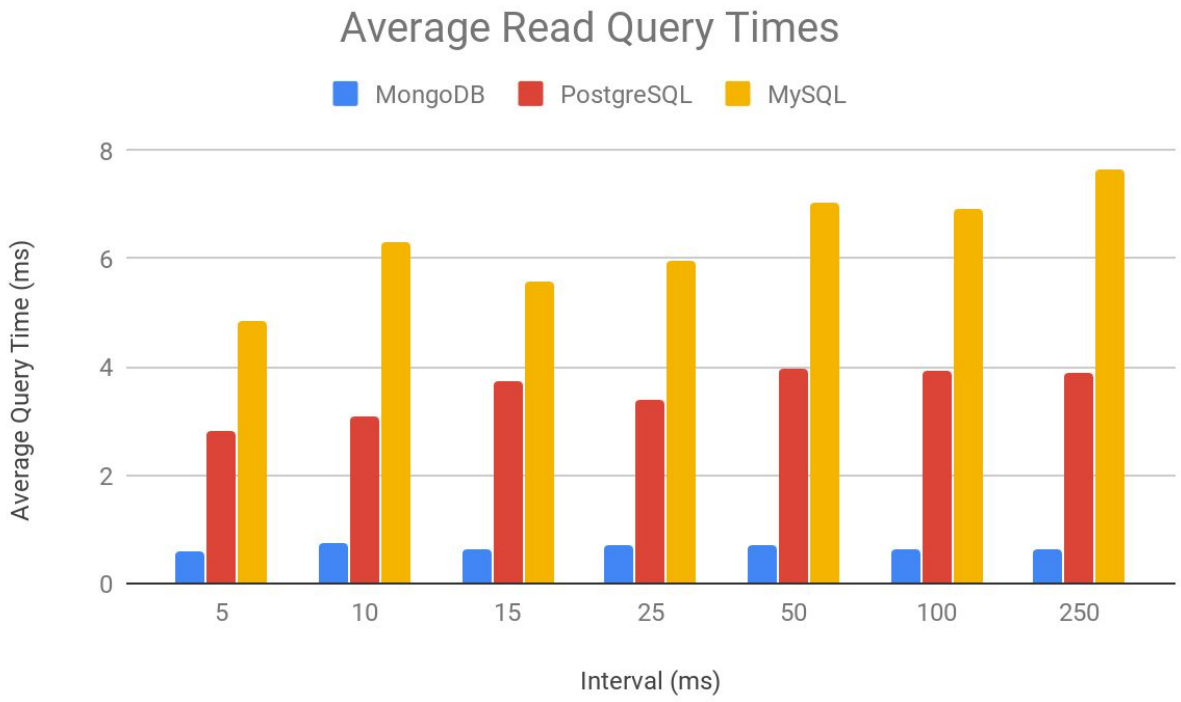

Figure 6 - Average Read Query Times (ms)

When comparing database system performance on a low-power computer the CPU and RAM usage when performing these tests was monitored. There was no major difference between the systems when performing the read and write tests. All three of the systems used on average 40-50% of the CPU, and 20-25% of the RAM when tested on a 1.4GHz Raspberry Pi 3 Model B+ with 1GB of RAM [13].

CONCLUSION

After gathering and analyzing the data collected for each database system, it appears that MongoDB is faster on-average on a lower-power computer. PostgreSQL and MySQL both, on average, take longer for read and write operations. MongoDB does have the chance of being occasionally many times slower than PostgreSQL and MySQL.

It is understood that many factors play a role in these results, and that query time might not be the best way to measure the performance of a database. The team is considering using MongoDB over MySQL due to faster read and write query times; however, MongoDB may be challenging

for future team members to understand due to its different architecture, so MySQL currently remains the best option.

Going forward, it would be beneficial to continue to look at the performance of the database systems under actual conditions, not just simulated conditions. This would include making sure that database systems are simple to set up and get running, so that future team members could make use of this research.

ACKNOWLEDGEMENTS

To all the dedicated and hardworking members of the Missouri S&T Solar Car Team, both past and present, without them the team would not exist.

To Mr. John Tyler, who has been with the team nearly as long as it has existed. His technical expertise in all things related to solar racing is second-to-none. He has been not only support, but a technical resource in every way possible.

To the volunteers at the American Solar Challenge and Formula Sun Grand Prix who make solar car raycing in North America possible.

REFERENCES

[1] American Solar Challenge, “Formula Sun Grand Prix,” 29 July 2018. [Online]. Available: https://americansolarchallenge.org/about/formula-sun-grand-prix/. [Accessed June 2019].
[2] American Solar Challenge, “The Challenges,” 29 July 2018. [Online]. Available: https://americansolarchallenge.org/. [Accessed June 2019].
[3] Missouri S&T Solar Car Team, “Our Cars,” 2016. [Online]. Available: https://solarcar.mst.edu/cars. [Accessed June 2019].
[4] PostgreSQL, "The World's Most Advanced Open Source Relational Database," n.d. [Online] Available: https://www.postgresql.org/. [Accessed June 2019]
[5] MongoDB, "The most popular database for modern apps," n.d. [Online] Available: https://www.mongodb.com/. [Accessed June 2019]
[6] International Organization for Standardization, "ISO 11898-1:2015," 2015. [Online]. Available: https://www.iso.org/standard/63648.html. [Accessed June 2019].
[7] IETF, "User Datagram Protocol," 28 August 1980. [Online]. Available: https://www.ietf.org/rfc/rfc768.txt. [Accessed June 2019].
[8] Tritium Pty. Ltd., "tritium.com.au," 28 August 2011. [Online]. Available: http://tritium.com.au/wp-content/uploads/2012/07/TRI82.007v4_Ethernet_Interface.pdf. [Accessed June 2019].
[9] IEEE Standards Association, "802.11n-2009," 2009. [Online]. Available: https://standards.ieee.org/findstds/standard/802.11n-2009.html. [Accessed June 2019]
[10] MongoDB, “What Is MongoDB?,” 2019. [Online]. Available: https://www.mongodb.com/what-is-mongodb. [Accessed June 2019].

[11] JavaScript Object Notation, “Introducing JSON,” n.d. [Online]. Available: http://json.org/. [Accessed June 2019].
[12] GitHub, “Ronmamo/Reflections,” 26 Feburary 2019. [Online]. Available: https://github.com/ronmamo/reflections. [Accessed June 2019].
[13] Raspberry PI. "Buy a Raspberry Pi 3 Model B – Raspberry Pi," n.d. [Online] Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/. [Accessed June 2019]