Investigating Compression Techniques For Integer Time Series Data For Low Power Applications

Parth Patel

***Abstract*—The**

I. INTRODUCTION

Compressing data to reduce its size has been important for a long time and its importance only continues to increase. With the ubiquity of mobile devices as well as the prevalence of the practice of collecting user and corporation data, compressing data even by small margins could result in large saving in the data that is stored on servers.

The reduction in size of saved files is one benefit of creating better compression algorithms but it also provides benefits in regard to the transmission of data from one device to another. Transmitting data between devices consumes power and reducing the size of the data that is transferred between devices will result in lower power consumption and therefore longer operability if the device runs on a limited supply of energy such as a battery.

When considering compression for space probes, limiting the size of data transfer provides power savings as well as allowing for more bandwidth will be available. Due to the large distance between the probe and Earth, reducing the size of the data needed to be transmitted results in power savings which are more significant than savings that would be seen in household devices. Having more available bandwidth could enable the installation of more devices on the probe as they could have some reserved bandwidth due to the savings gained from compressing data further.

We will be investigating how much of the original data can be compressed using techniques such as Huffman coding and Golomb coding. Lossy compression techniques will result in uncertainties in the final data that is received so we will be only investigating lossless compression methods.

II. THEORY

Reducing the size of a file through compression relies on the ability of being able to decompress it without losing data. This principle means that all compressed objects should be able to be decompressed using an algorithm that can correctly interpret all parts of the file. This is known as encoding where an encoding scheme maps each character or symbol to a unique binary value. This set of binary values can then be transmitted, and the receiver can use the same encoding scheme to map the binary values back to the characters or symbols that are required.

Reducing the size of the binary file is the purpose of compression methods and they work in various ways. The main method of reducing the size of the data lies in exploiting patterns such as repetitions that may occur and assigning repetitive characters a smaller binary value will reduce the overall size of the binary data.

A. Initial Value Encoding

The data is provided as integers which are magnetic field readings around the spacecraft. One of the simplest methods to compress will be to use the first value in the list of integers as an initial value and calculate all proceeding values as a difference from the initial value.

This technique works to reduce the number of bits that required to encode each integer to binary. If the integers in the original data require 10 bits to encode and the largest difference is 15, only 4 bits will be required to encode the difference hence

resulting in 40% compression when not including the overhead (discussed below).

A glaring weakness of this method is that it requires the initial value to be transmitted separately from the rest of the data and any mistakes in the transmission of the initial value could render the entire data packet useless. If the initial value is included within the same packet as the data it can add a small overhead. This overhead is always present and becomes significant for small packets as the overhead represents a large part of the packet size when compared to sending a larger packet.

Despite the weakness, initial value encoding is a relatively simple compression method which can be used for devices with low computational power and is especially effective for lists of integers with large values but relatively small changes.

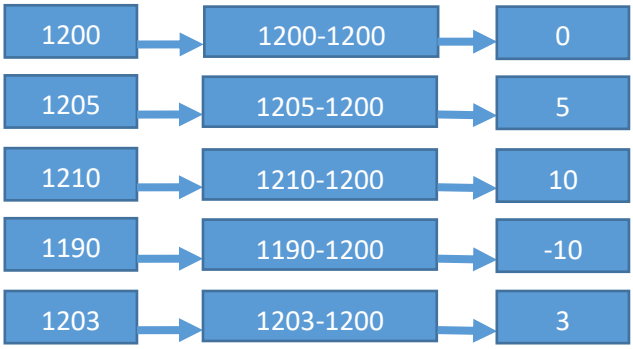

Figure 1: Example of initial value encoding with an initial value of 1200.

Initial value encoding is simple and as it does not control the encoding process of the integers, it can be combined with other techniques that are more efficient in the encoding process to further improve compression ratios.

B. Previous Value Encoding

This compression method builds upon the previously stated initial value encoding. Instead of listing the numbers as a difference from a static initial value, we could use the previous value within the list as a reference point when calculating the difference.

The effectiveness of initial value encoding is evident when dealing with a list of integers that doesn't stray too far from the initial value. Once the list of readings from the magnetometer becomes long enough, there is a large possibility of having differences which rival the size of the initial value. With previous value encoding, the largest possible size of the difference between any two values will be the maximum change that the magnetometer will be able to measure.

Whilst previous value encoding could possibly reduce the size further from initial value encoding, it still relies on the same overhead that initial value encoding was bound by. Due to this overhead, smaller packet sizes might still end up using a larger amount of transmission bandwidth over time.

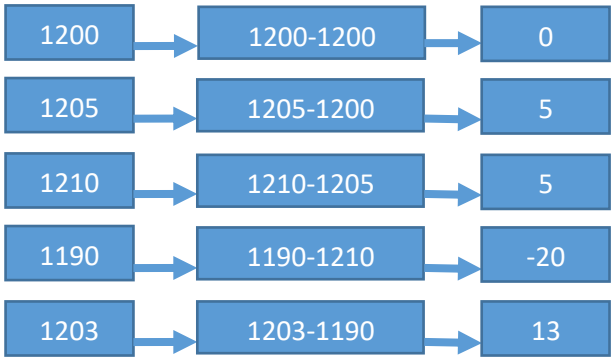

Figure 2: Example of previous value encoding.

Even with the requirement of the overhead, previous value encoding still holds the advantage of being able to be combined with other encoding techniques just like initial value encoding.

C. Huffman Encoding

Both initial value encoding and previous value encoding work on the manipulation of the integer data but Huffman encoding delves into the full encoding process and creates a binary packet.

Huffman encoding is used for strings and works based on reducing the bits required for more repetitive characters while allocating larger bit representations for sparse characters. The allocation of the number of bits for each character is built upon a tree structure. The two least frequent characters are combined to form a node and each branch represents a 0 or 1. The next least frequent character and the node are combined to form another node. This process of combining the least frequent terms continues until all the frequencies of the characters add together to the length of the full string. The binary packet for each character is formed by following the branches until you reach the desired character. The binary packet can be decompressed by using the same tree to follow the branches based on the binary packet.

Creating a tree for encoding each integer for the data is far more complex than the initial and previous value encodings that were discussed before. This will create a much larger load for the processor and will result in more time being required to form the full binary packet which needs to be transmitted.

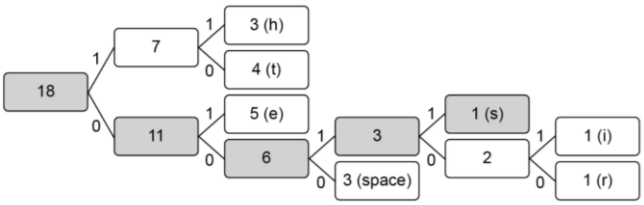

Figure 3:

In addition to the increased computational power that is required, the overhead that is created is far larger than the initial value which is required for initial and previous value encodings. This larger overhead means that this method will likely provide better compression for longer lists of integers rather than small lists.

Huffman encoding is used for string encodings whereas our data is based on integers so instead of being able to fully encode a number in one go, each digit will need to be separately encoded (eg. 3 separate encodings will be required for the number 184). Furthermore, it will not be sufficient to encode all the values directly using Huffman as a separator symbol is required to differentiate which number belong to which reading. This will increase the final packet size but it is required to ensure the data is separable.

The full list of integer readings from the magnetometer can be directly encoded using Huffman but we can also use the simple number manipulation provided by initial and previous value encodings before applying Huffman to possibly reduce the size of the final binary packet even further.

D. Golomb Encoding

Through the use of a divider, Golomb encoding uses manipulation of divisors and logarithms to create an encoding scheme that is specific to the chosen divider. A divider is a positive integer that is chosen by the user or algorithm that enables the encoding scheme to be built and the divider impacts the final size of the compressed packet.

A small divider results in fewer bits being required to encode numbers close the size of the divider but for numbers far larger than the divider, the binary representation can be much larger. A large divider results in longer binary representation for smaller numbers but far fewer bits are required to represent larger values. For a list of integers with larger values such as the spinning axis of the magnetometer, a larger divider is ideal despite the overall larger representation of the smaller values. For the axis that is relatively stationary with regard to the magnetic field, a smaller divider is ideal as the numbers will rarely reach far above the value of the divider.

Creating the encoding scheme is more complex than Huffman and it relies on the divider not the dataset as is the case with Huffman. A quotient q is formed from the number to be encoded and the divider. The remainder r from the quotient is also required after which the full binary representation relies on the size of the remainder.

$$q = \left\lfloor \frac{n}{m} \right\rfloor \tag{1}$$

$$r = n - qm$$

The quotient is converted to binary using unary encoding and the remainder is encoded using a number of bits which is specified based on the size of the remainder. If the remainder is less than 2^b-m, b-1 bits are used to represent r, else b bits are used to represent r+2^b-m. The full representation is a combination of the unary code followed by the binary representation of the remainder. The full process is shown in Figure 4.

$$b = \lceil \log_2 m \rceil \tag{2}$$

Quotient values, remainders and the use of a divider make this process very complicated and thus it will require more computational power or more time with the same processor than any of the previous methods. Despite its complexity, all values calculated in the process such as b, q or r are all integers due to the floor and ceiling functions. This means that this compression technique can still be carried out on any processor as only integer calculations are carried out.

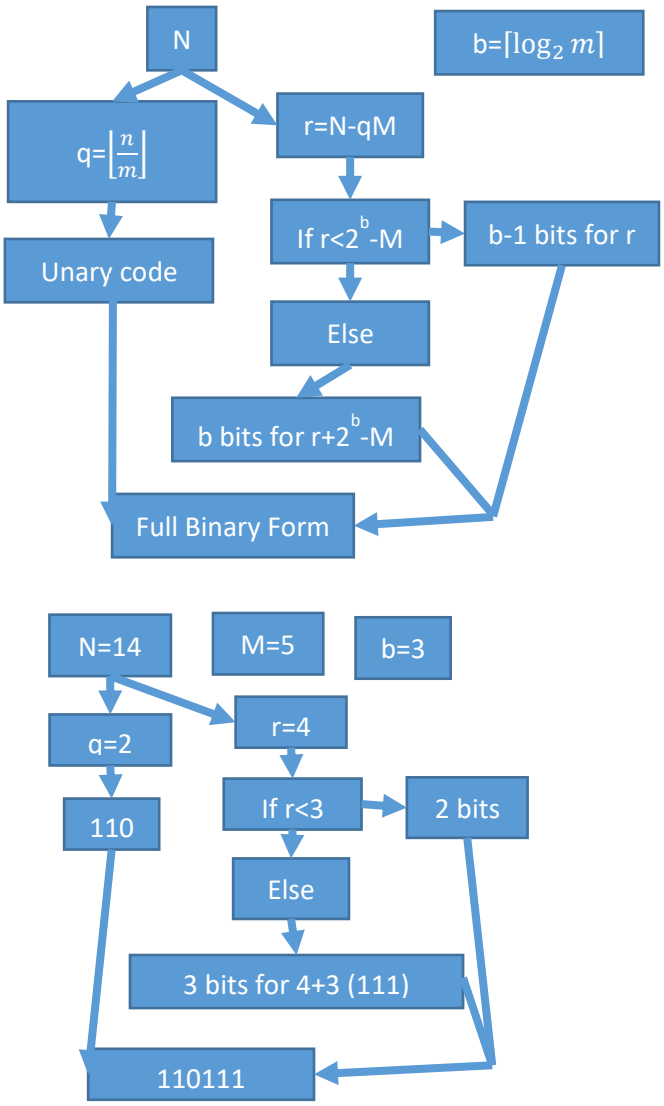

Figure 4: Flowchart (with example below) of how to apply Golomb coding to encode a number N.

This method is a full encoding method like Huffman and can therefore be combined with initial or previous value encoding. Unlike Huffman which requires each integer in a number to be separately encoded, Golomb coding works with the entire number and so each value can be encoded in its entirety.

III. Method

As the main purpose for investigating these techniques is for use in the comet interceptor, we will need to consider the limitations of the hardware and software that will be deployed on the probe. All of the compression techniques will be implemented through Python. The probe will not have access to such a high level language as it is far slower compared to lower level languages and we do not require the full functionality of the Python language. The implementation is in Python to allow us to fully visualize and investigate how changing various aspects of the compression technique impacts the final binary packet.

The processor on-board the probe is not computationally powerful so it will not be able to do floating point calculations. This does not present a problem for us as none of the compression techniques requires any floating-point calculations.

To test the different compression methods, we will be using solar wind data which is provided as floating-point numbers. We converted this to the original 16-bit integers to simulate a data flow that is likely to be similar to the one seen on comet interceptor. The data accrued by the comet interceptor will be in the form of a 14-bit signed integer but this is accounted for in the comparison between original and compressed size. Another reason for using solar wind data is due to the nature of the axial rotation impacting the readings. The comet interceptor will have one axis of the magnetometer which is somewhat stationary relative to the magnetic field. This will result in one axis which has very small readings and deviations in the reading while the other two axes will rotate relative to the magnetic field. This produces a sinusoidal curve which is observed within the data used to test compression.

Initial exploration of the data shows that the x axis has the smallest magnitude readings and the smallest range of readings. This seems to suggest that this axis is stationary relative to the magnetic field. The y axis has the largest magnitude readings and the largest range followed by the z axis. Since the y axis precedes the z axis, both should have similar readings and deviations but this is not the case which implies a possible offset on one of the axis due to the instrument.

To understand how the binary packet sizes vary, we need to understand how the data is transmitted. With no compression, the transmission of the binary packet consists of a header followed by a block of binary which contains the magnetometer readings (eg.42 bits representing the first reading with 14 bits for each reading). We investigated how we could best compress the binary block as it will be the largest portion of the transmitted packet and this is the best opportunity to reduce the size of the overall transmission.

We did not investigate how each of the compression methods compares in terms of the time taken to produce the compressed binary packet. This is due to the unbalanced software and hardware being used. Processors in the computers that we are using are far more capable and thus can run operations very fast but Python is a slower language compared to the actual compression algorithm which will likely be programmed using a lower level language such as C++. Investigating the time to compress will likely be very skewed and unreliable so only compression ratios will be compared.

Both initial value encoding and previous value encoding are methods which focus on reducing the overall size of the numbers involved so we will be looking at both compression percentage as well as the distribution of number of bits for each axis. The number of bits required is based on the largest number that is present in a sample after the differences have been accounted for (eg.5 bits are required for the entire sample if the max number is 31). Varying bits cannot be used because this removes the ability to decompress the binary packet.

Golomb encoding is a more unique method of compression compared to the others so we will be evaluating and analyzing its compression differently. The requirement of choosing an integer value for optimizing compression means that different integers will impact each axis independently. We cannot use the number of bits as a measure of compression because the mapping of Golomb encoding leads to varying bit values for each number needing to be compressed.

Preliminary testing showed that using the full integer values from our data would give us very inflated compression ratios so we decided to first use previous value encoding before applying Golomb coding. For the encoding process, we created a table of values for each value of ‘M’ after which the list of integers from previous value encoding was mapped using the appropriate table.

IV. Results

The comet interceptor will be collecting data at a rate of 67 readings per second so we decided for our initial comparisons between the compression methods to be at a fixed number of readings which would be 350. This is 5.22s of data which will be collected into one binary packet with a header. The graphs do not include the overhead cost associated with each compression method.

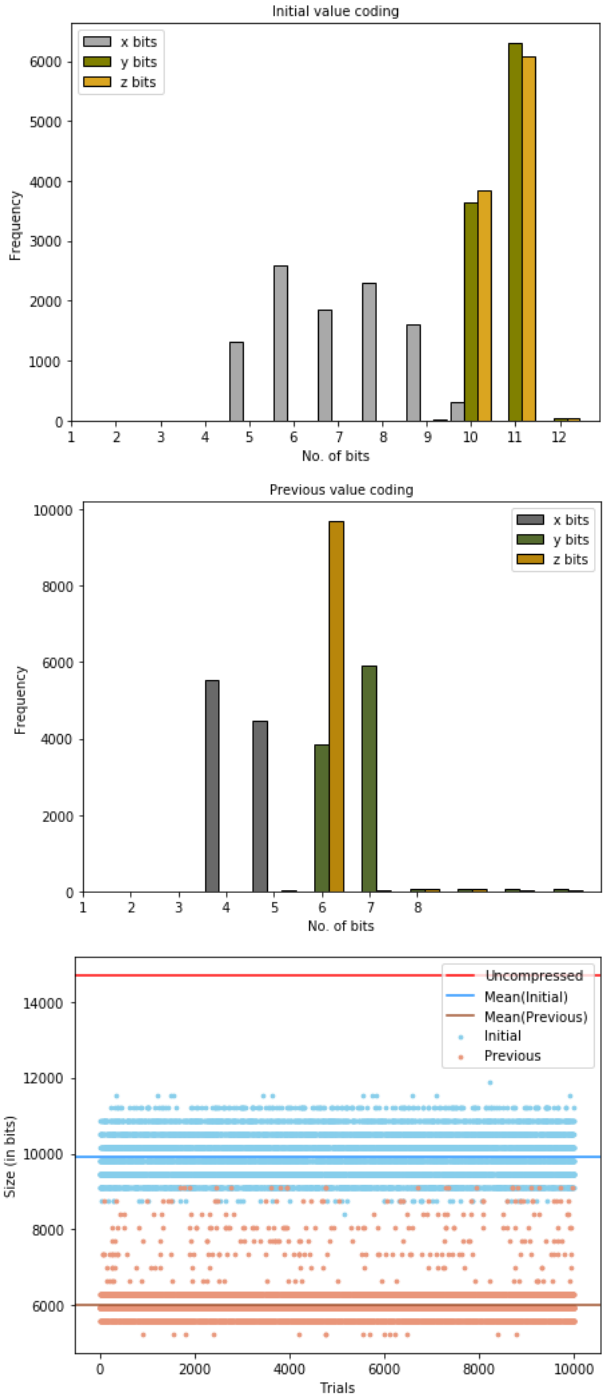

Figure 5: Histogram comparison of magnetometer axis readings (350 sample size for each run) when encoded with initial value encoding and previous value encoding. The final graph shows the distribution of total binary packet size after combining all axes.

When comparing initial and previous value encoding, we looked at both the number of bits for each axis for each method as well as the compression percentage. The axis comparison for each method is shown in Figure 5.

For initial value encoding, we can clearly see that the x axis has an overall lower number of bits

required to fully encode all values. There are rare occurrences of the x axis requiring bits similar to the other axes, but this is quite unlikely. The y and z axis required 10 or 11 bits in most cases which is not a very large reduction from the uncompressed numbers requiring 14 bits.

We see a slightly different distribution when looking at previous value encoding. The x axis still requires the least bits bit the z axis bits are largely all require 6 bits. The y axis seems to have larger values overall as it is spread closer to 7 bits. Previous value encoding ranges from 4 to 8 bits being required for any axis with some rare outliers being present but this range is overall far better than initial value encoding which ranges from 5 to 11 bits. The range alone may not be enough to fully justify the better compression of previous value encoding so the final graph of Figure 6 shows the size of the full binary packet which uses the number of bits that each axis requires. The uncompressed binary packet is 14700 bits and we can see that either compression method reduces the binary packet size to far lower than the uncompressed size. The distribution of initial value encoding has a far smaller distribution than previous value encoding which suggests that it is more consistent and predictable. Despite the consistency, previous value encoding seems to always outperform initial value encoding in terms of final binary packet size as the mean size is around 6000 bits compared to 10000 bits for initial value encoding.

Using differences as an encoding method also enables us to see how the different axis compare to each other. This is seen in Figure 7. The first histogram comparison between the initial and previous value encodings allows us to see a clear difference between the encoding methods. The compression percentage in these histograms refers to the amount of data that can be removed while still retaining all information. Initial value encoding has a far higher distribution of compression percentage compared to previous value encoding. This further proves the consistency of previous value encoding although it is possible that this is mainly due to the dataset being used to test these compression methods. It is clear that previous value encoding is almost always better for the x axis. While there was slight overlap in x axis, this overlap is completely removed when looking at the y and z axis. Initial and previous value encodings have their own regions where they are prevalent and for initial value encoding, the bits removed from the original only amount to 20% which is far lower compared to the 50% of bits that can be removed when using previous value encoding. Figure 6 gave an impression of the encodings being similar to each other but the clear distinction in Figure 7 shows that previous value encoding is the better compression method to use for all axes.

The use of Huffman on our data produced very surprising results. Figure 8 shows the outcome of using Huffman coding for compressing integers. The histogram shows compression ratios being very close to 100%. This means that the compressed size of the binary packet was not much smaller than the original binary packet. With an average compression ratio of 95%, Huffman coding provides very little (only 5%) compression to our data. Even more surprising is that we can see that many samples had compression ratios that were over 100%. This would make the compressed binary packet larger than the original which would be disastrous if bitrate limits were implemented. Huffman coding is used alongside Lempel-Ziv in zip compression which is a testament to its capability but it seems to be completely

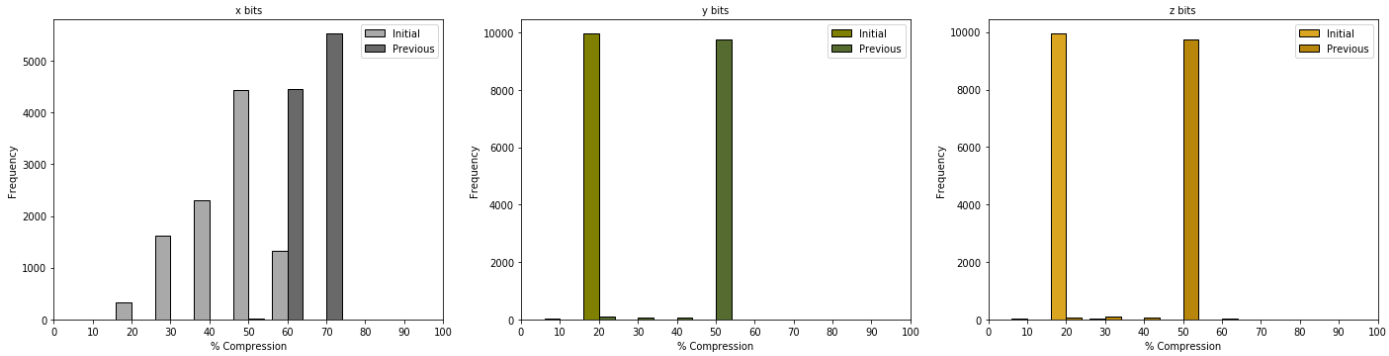

Figure 7: Bit requirements for each axis when encoded with initial and previous value encoding (350 samples for each run).

inappropriate for our usage.

Possible explanations could lie in the processing of our data as well as the type of data itself. In initial and previous value coding, having the same number of bits for each reading makes the final data easy to separate as the next block of bits will be of certain length and it will belong to a certain axis. With Huffman coding, the binary conversion of each character means that there is no way to distinguish when one axis ends, and another begins. To combat this, we added commas between each reading. This means for any sample of length n, we add n-1 commas to be able to separate the axis readings. This adds a large amount of overhead costs and is likely the reason for the reduced compression ratios.

One possible solution to the comma is to add signs instead of commas in front of all integers (eg. -12,4,-6 becomes -12+4-6). The signs would allow for the differentiation between each axis but would only be effective for situations when a negative number follows a comma. Considering the sinusoidal nature of the data, this would remove approximately half of the commas which would be an improvement but wouldn't provide a significant boost in compression ratios since the main cause of the large size is due to the encoding of each digit rather than the number as a whole.

The

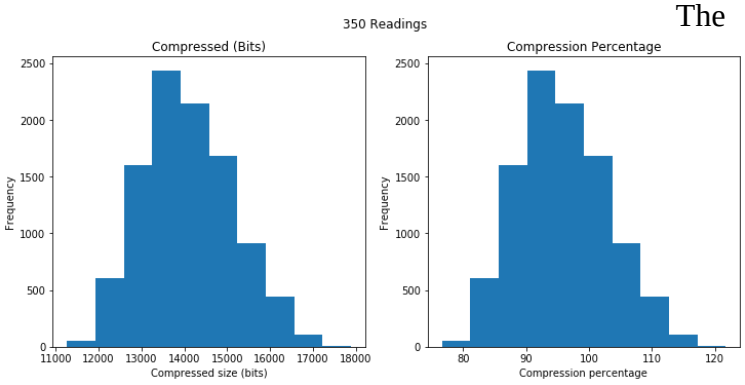

Figure 8: Compression ratios and binary packet sizes when Huffman is applied after initial value encoding (350 samples for each run).

results for the Huffman encoding seen in Figure 8 is when the integers are processed through initial value coding first. This would reduce the impact of encoding each digit since the integers would be smaller after some pre-processing.

Since we see previous value to be better than initial value encoding, we tried to apply previous value before using Huffman but we still see very mediocre results. With average compression of only 60%, we can see from Figure 9 that the overall spread of compression ratios is far smaller than when initial value encoding is applied. This further reinforces the position of previous value coding as being superior in both compressed amount and consistency.

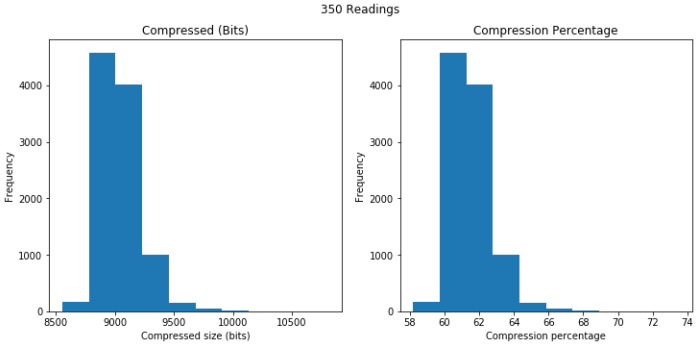

Figure 9: Compression ratios and binary packet sizes when Huffman is applied after previous value encoding (350 samples for each run).

Golomb coding was the final compression method and due to its unique nature of requiring a value dictated by the user, we needed to use more than just compression percentages to quantify its performance. We compared the average compressed size for each value of 'M' for each axis (over 350 samples) to find how it impacts compression ratios. This is shown in Figure 9.

The results seen for the x axis are strange yet justifiable. The magnetic field for the x axis is relatively stationary compared to the others and this means it will have the smallest fluctuations when encoded through previous value encoding. Having smaller values results in smaller M having the best compression ratios which is what we see in Figure 10 with M=2 having the lowest compressed size of 1200 bits compared to an uncompressed size of 4900 bits. Larger M values are unsuitable because of the increases in the value of b (Figure 4) as M increases.

Both the y and z axis readings show a similar pattern of decreasing to a minimum value after which there is a rapid increase in compressed size. The increase begins when the M value reaches a power of 2. This increase is due largely to the increase in the value of b and therefore each value now requires more bits hence the binary packet size increases.

Despite the similar patterns on the y and z axis, they have different optimal M values. The optimal value for the y axis is 15 whereas it is 7 for the z

axis. Through the preliminary investigation of the solar wind data, we found the y axis to have both larger values and range than the z axis. Since the best value of M depends on the size of the integers in the sample, larger samples (such as those in the y axis) have larger values of M providing optimal compression.

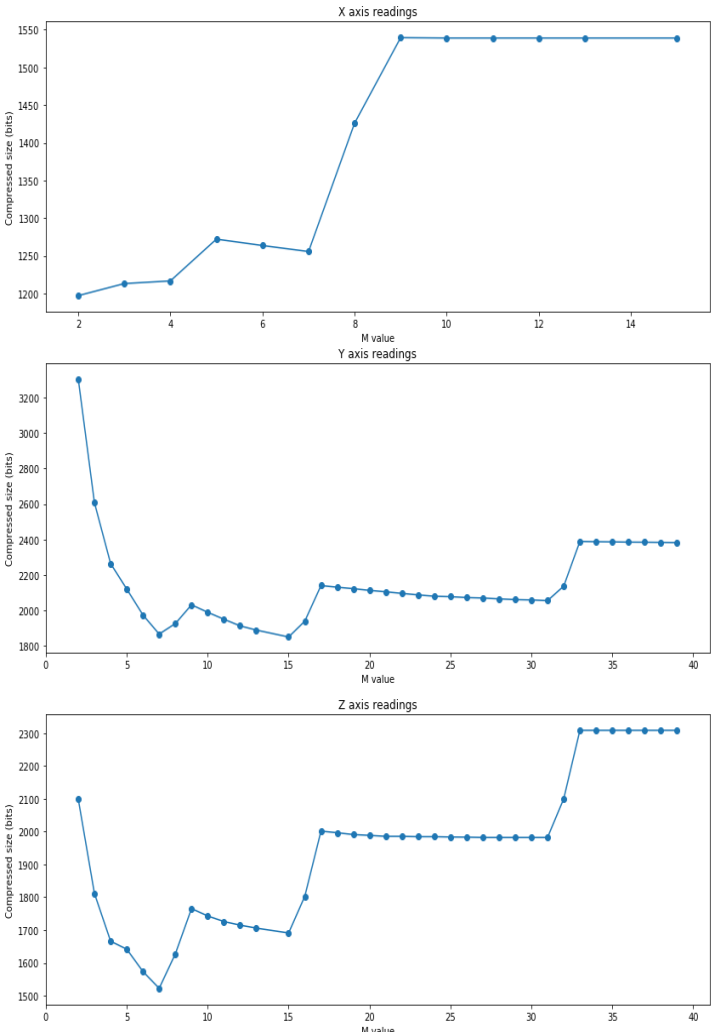

Figure 10: Compressed size (bits) for each axis across different M values for Golomb coding.

When the best M value for each axis is combined, we reach a compression ratio of 31% (69% of the binary packet can be removed and still retain all information). While this is impressive, it creates another challenge of how it could be put into a system with low computational capabilities.

V. Conclusion

Processors with little computational power face many challenges when working on compressing data. We compared simple implementations of compression techniques such as Huffman, initial value coding and previous value coding on solar wind data. Our solution for the comet interceptor magnetometer compression uses Golomb coding after pre-processing through previous value coding for a compression ratio of 31%. Despite it providing the best compression, its processing requirements are relatively high, and some sample size optimization could help offload this.

VI. Further Work

Combining multiple algorithms seems to be the way forward when further developing compression techniques. As such, we believe that further researching into computationally low power compression methods and combining with the methods stated in this paper could give us significant insights.

Since the algorithms were tested in Python, we couldn't fully quantify how the algorithms perform in terms of time taken to compress. Reproducing the algorithms in the language that will eventually be used will also help give us a better idea of the limitations of the algorithms.

Given more time, we would have also liked to investigate how the overhead costs of each algorithm impacts its overall packet size. Furthermore, changing the size of the compression sample will enable us to select an optimal number of readings. Experimenting with the number of readings and compression ratios could also allow us to select an optimal sample size which would also minimize latency as this is a mission which requires fast data transfer.

Appendix

Acknowledgement

References

- Include examples for all techniques (theory)
- Mention the use for this is for comet interceptor (intro)
- Source for Huffman diagram