Data Compression for Remote Laboratories

https://doi.org/10.3991/ijim.v11i4.6743

Olawale B. Akinwale
Obafemi Awolowo University, Ile-Ife, Nigeria
olawale.akinwale@oauife.edu.ng

Lawrence O. Kehinde
Obafemi Awolowo University, Ile-Ife, Nigeria
lkehinde@oauife.edu.ng

Abstract—Remote laboratories on mobile phones have been around for a few years now. This has greatly improved accessibility of these remote labs to students who cannot afford computers but have mobile phones. When money is a factor however (as is often the case with those who can't afford a computer), the cost of use of these remote laboratories should be minimized. This work addressed this issue of minimizing the cost of use of the remote lab by making use of data compression for data sent between the user and remote lab.

Keywords—Data compression; Lempel-Ziv-Welch; remote laboratory; WebSockets

1 Introduction

The mobile phone is now the de facto computational device of choice. They are quite powerful, connected devices which are almost always with us. This is so much so that about every important online service has a mobile version or at least a version compatible with mobile phones and tablets (for this paper we would adopt the phrase mobile device to represent mobile phones and tablets). This has caused online laboratory developers to develop online laboratory solutions which are mobile device-friendly [1, 2, 3, 4, 5, 6, 7, 8].

One of the main benefits of online laboratories is the possibility of using fewer resources to serve more people i.e. the money constraint [9, 10]. This fact (the fact that online laboratories are particularly useful for cash-strapped situations) has led to some research into reducing the cost of use of online laboratories [11]. Also, many online laboratory developers have tackled this cost of use minimization by using technologies which minimize the amount of data to be sent over the network such as LabVIEW Remote Panels [12, 13, 14]. Educational institutions in developing countries often fall in the class of "challenged educational institutions" where funding is not as available. Students in these institutions are often tied to low bandwidth networks or face steep costs for fast internet access. Hence data usage is a consideration of students in most challenged educational institutions – some of the main users of remote laboratories. The implication of this point is that in developing an online laboratory, every "trick in the book" should be employed to attempt to minimize the cost of use of online laboratories [11]. One "trick" which has not been used or at least which has not been reported so far is the compression of data being sent between the mobile devices and the online laboratory back-end (Service Broker or Laboratory Server).

By way of definitions, in this paper, the term "online laboratory" is used to refer to a laboratory whose experiments are performed over a network. The term "remote laboratory" is used to refer to an online laboratory which has physical equipment at the back-end. The term "virtual laboratory" is used to refer to a laboratory which uses models (mathematical models, graphical models, verbal models, etc.) of physical equipment instead of actual equipment and may or may not be performed over a network. A virtual lab can be installed on a user's phone or computer for example [8] or may be accessed over the internet (e.g. Phet Interactive Simulations [15]).

This paper is structured as follows: Section 2 discusses a few lossless compression algorithms. There are several lossless compression algorithms and this section will not attempt to talk about them all. It will however highlight a couple of the common ones and then present a comparison to suggest their usability in remote laboratory applications. Section 3 presents work we have done in implementing lossless data compression in an online laboratory. Section 4 discusses the results we obtained and Section 5 concludes this paper and makes some recommendations.

2 Data Compression

Data compression is divided into two: lossless compression and lossy compression. Lossy compression is what is used for pictures and videos where the raw data contains redundant information and not all the information is perceptible. Lossless compression on the other hand is necessary for data which we cannot discard any part of. With lossy compression, it is impossible to reconstruct the exact original data from the compressed data with zero error but a requirement for lossless compression is that the exact original data must be recoverable from the compressed data [16]. Lossless compression is what is discussed below.

2.1 ASCII

ASCII is a fixed length code which is used for encoding characters. Every keyboard converts the characters which are typed into an ASCII bitstream which is then sent to the CPU for use. ASCII makes use of eight bits to encode each character. (The ASCII standard is actually a 7-bit code but this is mostly always extended to 8 bits to cater for mathematic symbols or non-English European languages) [17, 18]. In other words, to send five characters ASCII would make use of forty bits. In this digital world which we live in, ASCII is regarded as raw data. Hence if the data is in ASCII code we just say that it is uncompressed data. Table 1 shows the ASCII codes for a few characters

Table 1. ASCII Code for a Few Characters

Character	ASCII Code
Space	00100000
+	00101011
1	00110001
2	00110010
@	01000000
A	01000001
B	01000010
A	01100001
B	01100010

2.2 Huffman's Algorithm

Huffman's algorithm [19] produces what is called a variable length code. A drawback of ASCII is the fact that it uses the same number of bits to encode each character – i.e. it is a fixed-length code. Huffman coding is based on the logic that we should use fewer bits as code words for characters we transmit often. This way we will transmit short codes (codes for frequently occurring characters) often and long codes (codes for infrequently occurring characters) less frequently.

Huffman's algorithm works in a three-step process. First it determines the different characters that exist in the data to be compressed. Next it determines how many times each of these characters appears in the data. This is often called the probability of occurrence of each character. Finally, it creates the code table by assigning fewer numbers of bits to the characters with the highest probability of occurrence. For example, Table 2 shows a result of Huffman's algorithm when run on the word 'ABRACADABRA'. From Table 2 we see that only 23 bits are used to encode 'ABRACADABRA'. If a fixed length code were to be used, it would have required at least 3 bits per character, and hence at least 33 bits would have been used to encode it. 7-bit ASCII would use 77 bits.

Table 2. Huffman Code Table for 'ABRACADABRA'

Character	Frequency	Huffman Code
A	5	0
B	2	111
C	1	1100
D	1	1101
R	2	10

It is important to note however that the advantage of Huffman coding over ASCII coding is not really the fact that Huffman uses a variable length code. Rather it is the fact that Huffman uses a statistical method i.e. the algorithm depends on the frequency of occurrence of each character.

More recently an adaptive Huffman method has emerged [20, 21]. Here the algorithm does not look at the entire data to be converted and create its code table before it begins encoding the data. Rather it updates its code table on the fly in a similar way to Lempel-Ziv's method (discussed below). The probabilities of occurrence of the characters is updated as it does the encoding and hence its code table is updated accordingly.

2.3 Lempel-Ziv's Algorithm

Abraham Lempel and Jacob Ziv, in 1977 [22] and 1978 [23, 24] created the algorithm which is popularly known as the Lempel-Ziv (LZ) algorithm. It is also a statistical method. One major difference between LZ and Huffman is the fact that the LZ algorithm does not need a code table to be prepared for the data before it is compressed which also means that it does not need to transmit its code table along with the data being transmitted.

Unlike Huffman coding, which uses the fewest number of bits for the most common characters, LZ works by adding character phrases to its code table as these phrases come up and then using the code for these phrases as the phrases show up later. For example, to transmit the string 'TO BE OR NOT TO BE', once the phrases 'TO' and 'BE' have been placed in code table, their code words are used when these phrases show up later in the string. Different implementations of LZ algorithm have yielded a whole slew of algorithms which are typically named beginning with LZ. For example, LZMA is the Lempel-Ziv-Markov algorithm, LZSS is the Lempel-Ziv-Storer-Szymanski method, and LZRW is the Lempel-Ziv-Ross-Williams method [25].

2.4 Lempel-Ziv-Welch's Algorithm

Terry Welch came up with a compression algorithm based on the LZ algorithm [26, 27]. This algorithm is called the Lempel-Ziv-Welch algorithm (LZW). In LZW, the compressor (and decompressor) begins with the ASCII table as its initial code table. When it sees the first character to be transmitted, it transmits its ASCII code. LZW then looks ahead to the next character to be transmitted and creates a phrase comprising the current character and the next character and adds this phrase to the code table. Whenever this phrase appears again in the data to be transmitted, LZW would simply transmit the code for this phrase rather than the individual codes for each character. Hence LZW uses a fixed-length code to represent data which has variable phrase lengths.

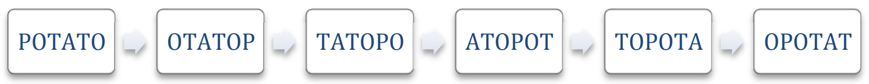
For example, to transmit 'TOBEORNOTTOBE', LZW would first transmit the ASCII code for 'T' and then add 'TO' to its code table. It then transmits 'O' and then adds 'OB' to its code table. Next it transmits 'B' and then adds 'BE' to its code table. This will go on and 'EO', 'OR', 'RN', 'NO', 'OT' and 'TT' are added to the table. At this time, it has transmitted 'TOBEORNOT'. At this point, it sees a phrase which it already has in its table. Hence instead of transmitting ASCII for 'T' next, instead it transmits its own code for 'TO' and then adds 'TOB' to its code table. Then its transmits its own code for 'BE' and the transmission ends. Hence as LZW sees repeated character phrases, it adds even longer phrases to its code table so that after transmitting a sizeable chunk of data, it can have a 13-bit code which represents a 25-letter phrase.

As can be inferred from the discourse above, LZW is good when phrases get repeated in the data. It also requires that a decision be made as to how many bits to use to represent individual phrases in its code table. The receiver would have to know how many phrases to bits to read before attempting to decode (decompress) it.

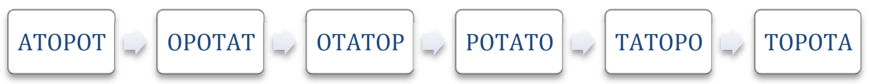
2.5 Burrows-Wheeler's Transform

The Burrows-Wheeler transform (BWT) [28] goes about the business of data compression via a different route from most other compression methods. The BWT method is based on the fact that a number of compression methods do very well with the compression of data which has several characters repeated. The BWT takes the data to be compressed and then first sorts it into a form such that a number of similar characters in the text are first grouped together and then a good compression algorithm can be used for the compression. In the strictest sense, the BWT is not a compression method but a sorting method which sorts characters in such a way that the original order of the characters can be found from the sorted data and an index number. The output of a BWT isn't simply a string in ascending or descending order and not all similar letters are necessarily grouped together. BWT is constrained by the fact that the original data must be recoverable from sorted data.

For example, let's say we were to use the BWT to sort the word "POTATO". The BWT method first shifts the data one step and creates a new row and then shifts again until it has cycled through all the characters. Hence, for this example, we will have the six rows below:



Next the algorithm sorts these rows in ascending order. Hence for our example we have:



Now, the last column of the table is the set of characters we send to the compressor before it is transmitted. In addition to this, the index of the row which has the original data we started with is also transmitted. Hence for this example the two outputs of the BWT algorithm are 'TTPOOA' and index 4. These two pieces of data (i.e. the sorted characters and the index of the row with the original data) are all we need to reconstruct the original data from the sorted data. One can now use a suitable compression method on the sorted data before transmission. The bzip2 compression algorithm makes use of BWT and Huffman coding for compression.

2.6 Arithmetic Compression

Arithmetic coding [29, 30, 31] works by performing arithmetic operations on the data to be coded. The number line between 0 and 1 is typically divided up into segments where each segment represents each of the unique characters in the data set to be compressed. To make the coding more efficient, the number line is not divided evenly. Rather it is divided according to the frequency of occurrence of the characters [32]. More frequent characters are assigned larger segments of the number line. This way, fewer bits end up being used to encode more frequent characters.

The procedure for arithmetic coding is:

1. divide the number line between 0 and 1 into the requisite number of segments according to the frequency of appearance of the characters.
2. select the segment for the first character to be encoded.
3. divide this selected segment in (2) above into the same number of segments as in (1) above using the proportions used in (1) above.
4. select the segment for the next character to be transmitted.
5. Repeat the process until we have used all characters for transmission.
6. Transmit a value which lies in the segment of the last character for transmission. This value will have all the information in it about all characters which were in the original data. Simply transmitting this value makes it possible to reconstruct the original data.

For example, if the string ‘JOHN’ is to be transmitted, assuming the characters have equal probability in our entire data stream and the entire stream does not have any other characters other than J, O, H and N, we can divide up the 0 to 1 interval as shown in Fig. 1. If the value 0.106 is now transmitted, we can reconstruct the string JOHN from this as we know that this solution only exists within the fourth quarter of the third quarter of the second quarter of the first quarter of 1. The algorithm is a little more complex than this but this will suffice to present the basic concept behind arithmetic coding.

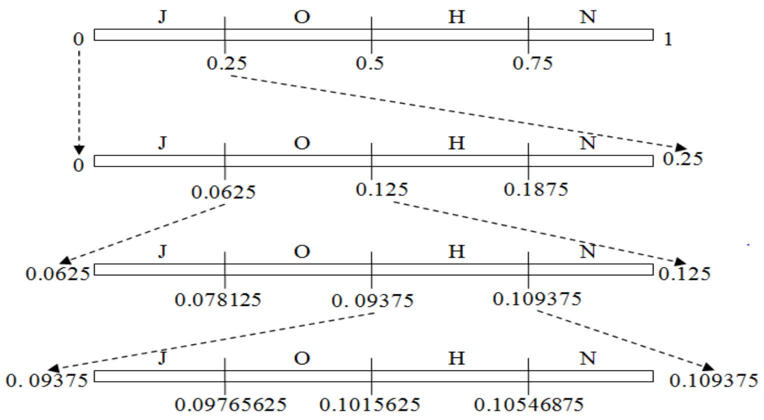


Fig. 1. Arithmetic Coding to encode JOHN

2.7 Comparison of the Above

There is no single best compression method for all forms of data. Each method has its comparative advantages. The standard most frequently used as a benchmark for data compression methods up until the 1990s was the Calgary Corpus [33, 34]. Table 3 shows compression capabilities of the methods presented above when tested with the Calgary Corpus as reported in [35, 36]. BWT methods tend to do better than LZW which in turn is better than both LZ77 and Huffman in terms of compression ratios. The compression ratio of arithmetic coding is dependent on the implementation of the algorithm but it tends to provide very good compression ratios. In terms of speed of compression and decompression, LZW executes faster than BWT and LZ77. BWT tends to be slow because sorting is done before compression. The decompression of BWT takes even longer because it involves decompression and then sorting the decompressed data and swapping rows over and over.

Table 3. Compression performance of compression algorithms

Year	Compression Scheme	Bits per Character
1967	ASCII	7.00
1950	Huffman	4.70
1977	Lempel-Ziv 1977 (LZ77)	3.94
1985	Lempel-Ziv-Welch (LZW)	2.93
1990s to Date	Arithmetic coding	Depends on the programmatic implementation but very good compression is achieved.
1995	Burrows-Wheeler Transform	2.29

Data compression coding methods can be roughly divided into three: static models, dynamic models and adaptive models. Static models use the same code table for all data to be compressed. The code table is often created to be somewhat optimal based on the language they are written for e.g. Morse code was optimized for the English language based on the fact that vowels like ‘e’ and ‘i’ appear fairly frequently. Examples of the static model are Morse code and ASCII. Dynamic models get their name from the fact that their code tables are not “set in stone”. The code tables are often determined when compression is to be done. These methods look at the data and try to determine the best code table which would compress the particular data best. Examples of dynamic model codes are Huffman codes and arithmetic coding. The adaptive coding model however does not create its code table for the entire data set before it encodes it but rather creates a code table while encoding it. This way it is not necessary to transmit the code table from the source to the receiver in addition to the data being encoded. The receiver can reconstruct the code table from the received data and use it to decode the received data. Table 4 presents a comparison of these three models.

Table 4. Comparison of Lossless Compression Models

	Static Model	**Dynamic Model**	**Adaptive Model**
Code table	Same code table for all data to be compressed	Code table is generated based on the data to be compressed so that it is optimized for that particular data	Create code table on the fly while coding (or decoding)
Compression and decompression Speeds	Very fast	*Code table must be generated before compression is done i.e. slow	*Faster than dynamic models
Compression ratio	Quite poor	Close to entropy	Better compression than both other models
Code table availability	Everyone has the code table already	Code table must be transmitted to decoder	Length of each code word must be transmitted & decoding must start at beginning
Example	E.g. ASCII, Morse Code, etc.	E.g. Huffman code, BWT, arithmetic coding	e.g. LZ, LZW, LZMA, etc.

* Specific dynamic models may be faster than specific adaptive models but the table presents the general case

3 Our Method

For remote laboratories applications where data is to be streamed live between the user and the lab we propose that the LZW algorithm or an algorithm similar to it be used for data compression. The reason for this is the fact that the code table is created on the fly without prior knowledge of the probability of occurrence of individual data characters and no need to transmit the code table to the decompressor. Hence, LZW is well-suited for compressing a live stream of data whose statistics we have no prior knowledge of. A typical goal in a remote laboratory is to give the users the freedom to make mistakes and freedom to explore but protect the lab equipment by preventing harmful specifications. Typically, the user should be allowed to submit even ridiculous specifications before the online platform tells him his specifications are faulty. Hence it would be impossible to tell the exact probabilities of specific data being sent by users to the lab and create an optimal code table from this.

The main disadvantage of the LZW algorithm is the fact that the decompressor must start decompressing the received data from the very beginning or else its code table will not match that of the compressor. Hence it is essential to ensure that any app using LZW does some handshaking before it begins compression to ensure that the decompressor is receiving its data.

3.1 The System Under Test

The experiment chosen for this work is the ball and beam system experiment, a Control Engineering position control experiment. The ball and beam system is a system in which a beam is tilted back and forth in order to keep the ball located at a desired position. When the ball is stationary at the desired position the beam is kept horizontal. When the ball however gets displaced from this position the beam is tilted accordingly to bring it back to the desired position. The ball and beam system is shown in Figure 2.

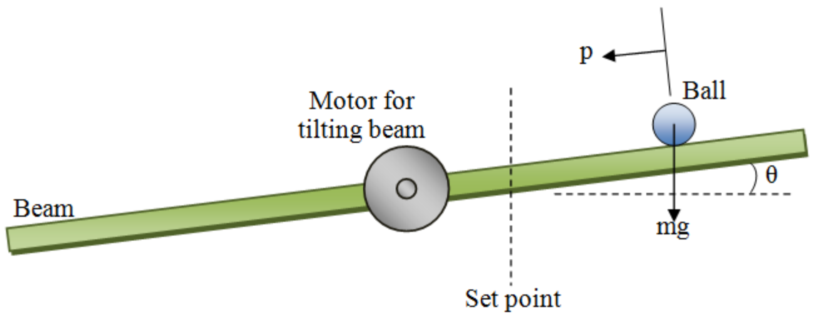


Fig. 2. A ball and beam system

A number of mathematical models for the ball and beam system have been computed [37, 38, 39, 40, 41]. A number of works focus on the use of the voltage supplied to the DC motor as the control signal. When this is done, the dynamics of the beam are very essential. For this work however, the control signal was the beam's angle. Hence the only dynamics to be concerned with, since the beam was rigid, were the ball's angular position and velocity, and the ball's translational displacement. The angular position, α of the ball is directly related to the translational displacement of the ball, p.

$$p = r\alpha \tag{1}$$

where p is the translational displacement of the ball, r is the radius of the ball and α is the angle by which the ball has rotated. Hence, from equation 1, the ball's angular acceleration can be written in terms of its translational displacement.

$$\ddot{\alpha} = \ddot{p}/r \tag{2}$$

Now, assuming that the ball rotates along the beam with zero slip and zero friction, the torque on the ball, τ can be expressed in terms of the force due to rotation acting on the ball, F_r.

$$\tau = F_r \times r \tag{3}$$

The torque on the ball can also be expressed in terms of the moment of inertia of the ball J and the angular acceleration of the ball.

$$\tau = J\ddot{\alpha} = \frac{J}{r}\ddot{p} \tag{4}$$

From equations 3 and 4, we get the force due to rotation acting on the ball as

$$F_r = \frac{J}{r^2}\ddot{p} \tag{5}$$

Now, balancing the forces on the ball, assuming no external forces acting on the ball, we have

$$mgsin\theta = m\ddot{p} + \frac{J}{r^2}\ddot{p} \tag{6}$$

where m is the mass of the ball and g is the acceleration due to gravity. Linearizing this equation for small deviations of θ, we have

$$mg\theta = \left(m + \frac{J}{r^2}\right)\ddot{p} \tag{7}$$

Given that the moment of inertia of the ball (a sphere) is given as $J = \frac{2}{5}mr^2$, we obtain the transfer function of the ball and beam system from equation 7, with the ball's translational displacement as the output and the beam's angle as the input, as

$$\frac{P(s)}{\theta(s)} = \frac{5g}{7} \times \frac{1}{s^2} \tag{8}$$

Thus, without a controller, the system is undamped. The controller used in this work is a PID controller. The configuration used in this work is shown in Figure 3. The transfer function of the closed loop system is

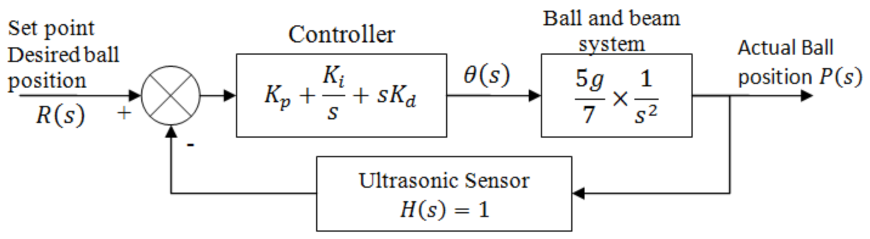


Fig. 3. Control system configuration used

$$T(s) = \frac{5g(s^2K_d + sK_p + K_i)}{7s^3 + 5g(s^2K_d + sK_p + K_i)} \tag{8}$$

The controller was implemented on an Arduino Uno R3 board. The built ball and beam system is shown in Figure 4.

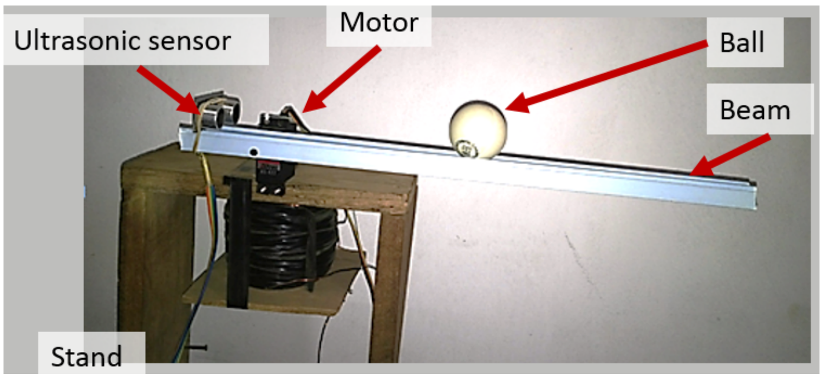


Fig. 4. Built ball and beam system

The task of the student in the ball and beam experiment is to design a controller to make the controlled system meet some desired performance characteristics. For example, the student is required to design a PID controller which will cause the ball the settle as the specified set point within 2 seconds of its disturbance and with no more than a 50 % overshoot. The student is to do the math and come up with proportional, derivative and integral gains for the PID controller which would ensure that the system meets these performance characteristics. Having come up with these, the student supplies these gains to the physical system and observes / measures the system's response.

3.2 The Remote Lab Client

The remote laboratory client was created using Construct 2. Construct 2 is a game development system which develops applications using HTML5, JavaScript and jQuery [42]. Hence applications developed using Construct 2 can be easily exported to various mobile platforms. The laboratory server, at the back-end, was written in C#. WebSocket methods as well as data compression methods were built into the lab server.

The remote lab client application developed had two purposes. The first purpose was to enable the student to enter his desired controller gains, and the second purpose was to display the response of the physical system to the student. The algorithm of the remote lab client developed is shown in Figure 5. The laboratory functions as a remote laboratory when a connection between the lab client, and the lab server and experiment setup exists. The laboratory defaults into the virtual lab mode once this connection is lost. In the virtual lab mode, it uses the mathematical models of the ball and beam system and the controller to compute what the response of the system should be. In remote lab mode, however, the user's specifications are sent to the physical ball and beam system setup which responds to these specifications and sends its response to the lab client. Data compression was done on all data sent between the client and the server over the internet. The LZW algorithm was used.

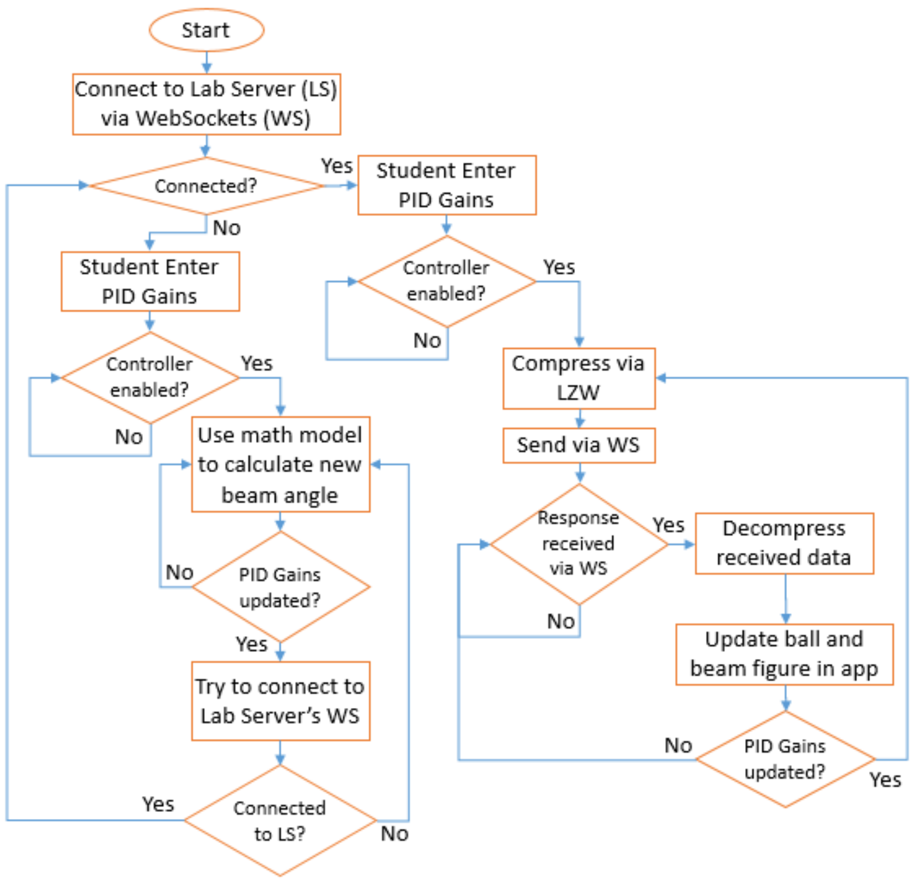


Fig. 5. Algorithm of laboratory client on the user's device

The data compression and decompression in the client were done by using the LZW algorithm. JavaScript plugins were written for Construct 2 to handle the compression and decompression. Figures 6 and 7 show the compression and decompression algorithms used respectively.

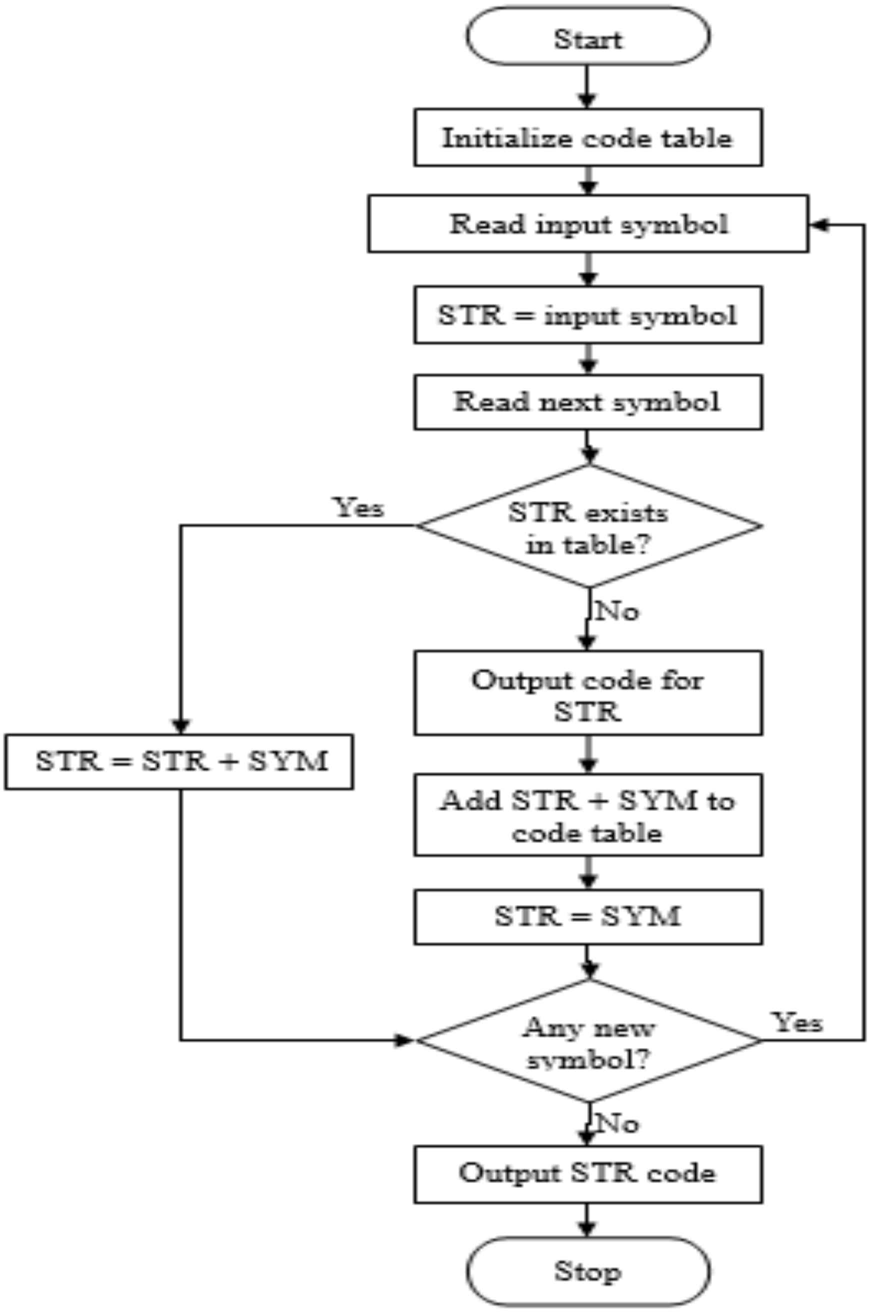


Fig. 6. Data compression algorithm used

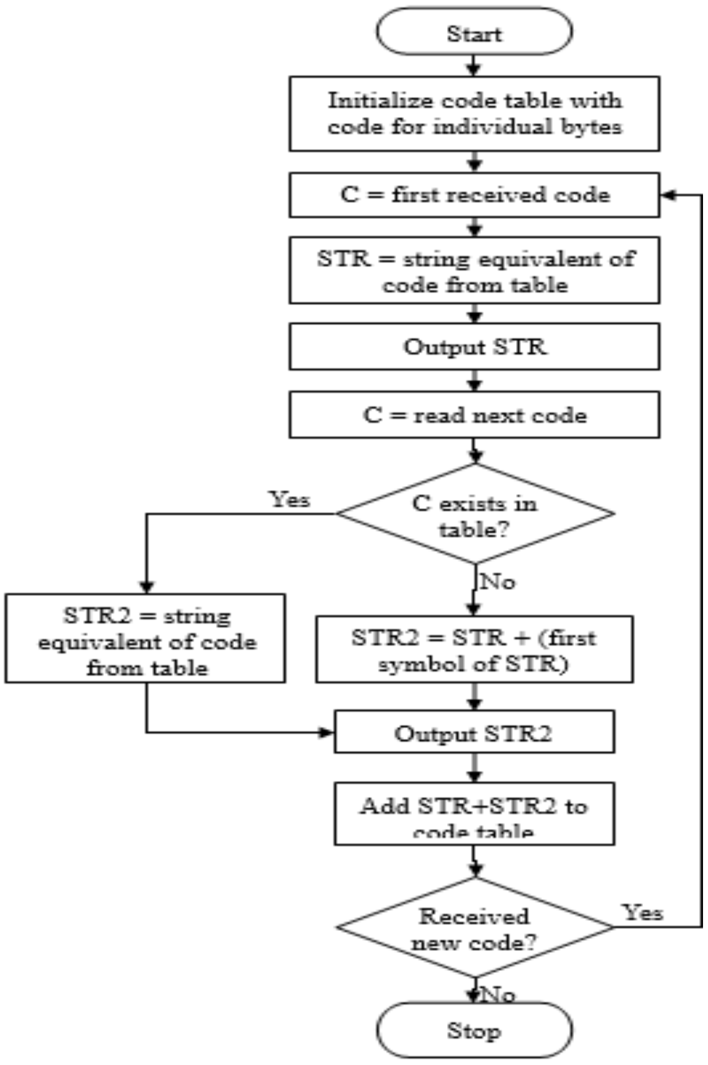
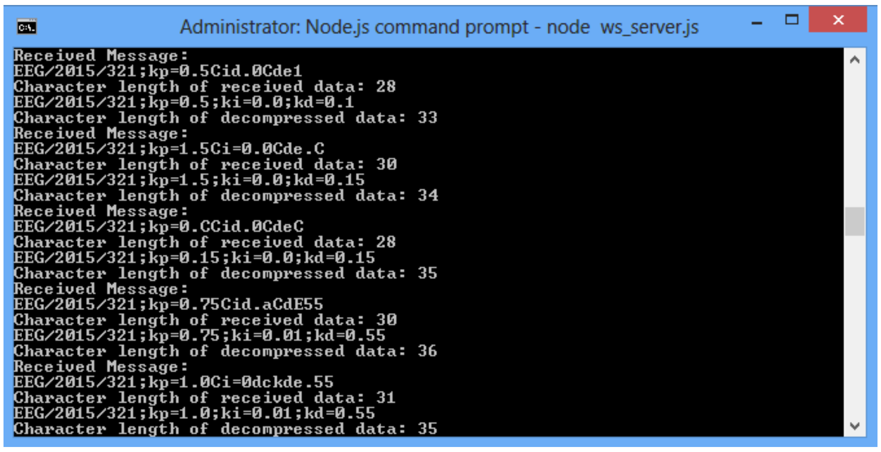

Start
Initialize code table with code for individual bytes
C = first received code
STR = string equivalent of code from table
Output STR
C = read next code
C exists in table?
Yes
No
STR2 = string equivalent of code from table
STR2 = STR + (first symbol of STR)
Output STR2
Add STR+STR2 to code table
Received new code?
Yes
No
Stop

Fig. 7. Data decompression algorithm used

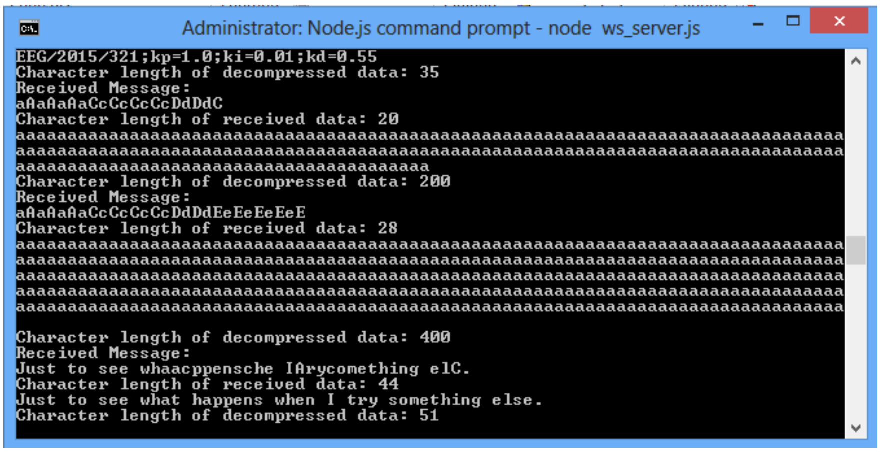
4 Results and Discussion

The data compression system was tested using a Node.js server [43] on the backend by making use of the WebSocket package [44] and the node-lzw package [45]. The tests showed that the data was correctly decompressed each time. Figure 8 and Figure 9 present the results. Figure 8 show that compression ratios of 0.82 to 0.88 were obtained for the ball and beam remote lab. Hence, the use of this remote lab client would save the student more than 10% of his mobile data use when using a similar remote lab client but which does not employ LZW.



```
Received Message:
EEG/2015/321;kp=0.5Cid.0Cde1
Character length of received data: 28
EEG/2015/321;kp=0.5;ki=0.0;kd=0.1
Character length of decompressed data: 33
Received Message:
EEG/2015/321;kp=1.5Ci=0.0Cde.C
Character length of received data: 30
EEG/2015/321;kp=1.5;ki=0.0;kd=0.15
Character length of decompressed data: 34
Received Message:
EEG/2015/321;kp=0.CCid.0CdeC
Character length of received data: 28
EEG/2015/321;kp=0.15;ki=0.0;kd=0.15
Character length of decompressed data: 35
Received Message:
EEG/2015/321;kp=0.75Cid.aCdE55
Character length of received data: 30
EEG/2015/321;kp=0.75;ki=0.01;kd=0.55
Character length of decompressed data: 36
Received Message:
EEG/2015/321;kp=1.0Ci=0dckde.55
Character length of received data: 31
EEG/2015/321;kp=1.0;ki=0.01;kd=0.55
Character length of decompressed data: 35
```

Fig. 8. Data received by Node.js WebSocket server showing sizes before and after decompression by LZW



```
EEG/2015/321;kp=1.0;ki=0.01;kd=0.55
Character length of decompressed data: 35
Received Message:
aAaAaAaCcCcCcCcDdDdC
Character length of received data: 20
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Character length of decompressed data: 200
Received Message:
aAaAaAaCcCcCcCcDdDdEeEeEeEeE
Character length of received data: 28
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

Character length of decompressed data: 400
Received Message:
Just to see whaacppensche IArycomething elC.
Character length of received data: 44
Just to see what happens when I try something else.
Character length of decompressed data: 51
```

Fig. 9. Response from SMS experimentation server to specifications sent.

Figure 9**Error! Reference source not found.** shows that the LZW is not optimal for sending small non-repetitive data from the client to server if the code table is reinitialized each time data is to be sent (Compression ratios of 0.82 to 0.86). For highly repetitive data however, it does very well, the larger the amount of data to be compressed. Other compression schemes could be used for different laboratories with code tables better suited for each lab. However, the system developed here is generic and can be used with any lab without prior knowledge of the characters the student would be expected to send to the server at the back end.

It is necessary to point out that the choice of whether to host an HTML5 lab client in a web page or whether to deploy it as apps which students can install on their phones does also significantly impact the student's data costs while using the lab. Hosting the lab client on the lab server is advantageous as it is very easy to update or modify the client. Once any user loads the client from your server, the most recent version is what he sees. On the other hand, each time the user launches the lab in his browser, the entire client (or just some part of it if cookies and intelligent coding is done) is downloaded to the user's browser. Hence his cost of use of the lab client is significant higher. If, on the other hand, the user has the client installed on his phone his only data cost of use of the client would be the data interchanged between the app and the server e.g. handshaking and then his lab specifications. The initial data cost of downloading the app from the lab server or online store can also be possibly mitigated for Android, Windows Phone and Blackberry 10 users if a copy is made available on their campus. For example, the lab client's apk (for Android), bar (for Blackberry 10) or xap (for Windows 10) can be downloaded by the lecturer and his students can copy this from him via Bluetooth for example. iOS users would have to download the app from the Apple appstore.

5 Conclusion

This work has demonstrated that "Cost of Use" of remote laboratories can be reduced by the use of data compression when sending data between the students and the laboratory. Data use savings of up to 18% were obtained for the remote laboratory developed in this work. There is probably a discussion to be had as to which compression algorithm ought to be adopted as the default data compression algorithm for remote laboratories.

The discourse in this work has been only about remote laboratories. It should be said that it can also be employed in virtual laboratories whenever data is to be sent between the user and the servers at the back end.

6 Proposed Further Work

Dropped packets during a live transmission can be a problem when using the LZW algorithm. It would therefore be reasonable to have the decompressor periodically send back a decompressed piece of data to the sender to ensure that the compressor and decompressor are still synchronised and using the same data tables. This could be done in addition to the presently used system of reinitializing the code table periodically to ensure that the code table remains a manageable size and doesn't get too large. We also hope to implement a number of data compression schemes in the near future in order to create a guideline of suggested compression schemes for various kinds of remote and virtual labs.

7 References

[1] H. M. Al-Otaibi, R. A. AlAmer and H. S. Al-Khalifa, "The next generation of language labs: Can mobiles help? A case study," Computers in Human Behavior, vol. 59, pp. 342-349, 2016. https://doi.org/10.1016/j.chb.2016.02.028

[2] W. Rochadel, J. Shardosim Simao, J. da Silva and A. Vaz da Silva Fidalgo, "Application of mobile devices and remote experiments for physics teaching in elementary education," in Global Engineering Education Conference (EDUCON), 2013 IEEE, Berlin, 2013. https://doi.org/10.1109/educon.2013.6530210

[3] O. S. Oyediran, O. B. Akinwale, K. P. Ayodele and L. O. Kehinde, "Development of a Remote Operational Amplifier iLab Using Android-based Mobile Platform," Computers in Eduction Journal, vol. XXIII, no. 4, pp. 100-110, 2013.

[4] B.-A. Deaky, D. G. Zutin and P. H. Bailey, "The First Android Client Application for the iLab Shared Architecture," International Journal of Online Engineering (iJOE), vol. 8, no. 1, pp. 4-7, February 2012. https://doi.org/10.3991/ijoe.v8i1.1946

[5] M. A. Guerra, C. M. Francisco and R. N. Madeira, "PortableLab: Implementation of a Mobile Remote Laboratory for the Android Platform," in 2011 IEEE Global Engineering Education Conference (EDUCON) – "Learning Environments and Ecosystems in Engineering Education", Amman, Jordan, 2011. https://doi.org/10.1109/educon.2011.5773266

[6] J. Bermudez-Ortega, E. Besada-Portas, J. A. Lopez-Orozco, J. A. Bonache-Seco and J. M. d. l. Cruz, "Remote Web-based Control Laboratory for Mobile Devices based on EJsS, Raspberry Pi and Node.js," in IFAC-PapersOnLine, 2015.

[7] J. B. da Silva, W. Rochadel, R. Marcelino, V. Gruber and S. M. S. Bilessimo, "Mobile remote experimentation applied to education," in IT Innovative Practices in Secondary Schools: Remote Experiments, O. Dziabenko and J. García-Zubía, Eds., Bilbao, University of Deusto, 2013, pp. 281-302.

[8] O. O. Satope, L. O. Kehinde, I. O. Boboye, O. B. Akinwale and O. I. Asubiojo, "Development of a suite of virtual experiments for Physics and Chemistry undergraduate laboratories," Computers in Education Journal, vol. XXV, no. 2, 2015.

[9] A. Böhne, N. Faltin and B. Wagner, "Self-directed Learning and Tutorial Assistance in a Remote Laboratory," in Interactive Computer Aided Learning (ICL) - International Workshop 2002, Villach, Austria, 2002.

[10] L. Hesselink and B. Wagner, "Internet Assisted Laboratories (I-Labs)," 2001.

[11] L. O. Kehinde, X. Chen, K. P. Ayodele and O. B. Akinwale, "Developing Remote Labs for Challenged Educational Environments," in Internet Accessible Remote Laboratories: Scalable E-Learning Tools for Engineering and Science Disciplines, A. Azad, M. Auer and V. Harward, Eds., 2012, pp. 432-452. https://doi.org/10.4018/978-1-61350-186-3.ch022

[12] O. B. Akinwale, L. O. Kehinde, K. P. Ayodele, A. M. Jubril, O. P. Jonah, O. Ilori and X. Chen, "A labview-based on-line robotic arm for students' laboratory," in Proc., ASEE Annual Conference & Exposition, Austin, Tx, 2009.

[13] O. B. Akinwale, K. P. Ayodele, L. O. Kehinde and O. Osasona, "Implementing Remote Laboratories with the ILab Architecture: Three Case Studies from Obafemi Awolowo University, Nigeria," Computers in Education Journal, pp. 86-98, 2011.

[14] L. O. Kehinde, K. P. Ayodele, O. B. Akinwale and O. Osasona, "Remote Labs in Education. The Obafemi Awolowo University Experience," in Using Remote Labs in Education: Two Little Ducks in Remote Experimentation, J. G. Zubia and G. R. Alves, Eds., University of Deusto, 2012, pp. 81-111.

[15] University of Colorado, "Phet Interactive Simulations," 2016. [Online]. Available: https://phet.colorado.edu/. [Accessed 22 April 2016].

[16] K. Sayood, "Mathematical Preliminaries for Lossless Compression," in Introduction to Data Compression, 4th ed., Elsevier Inc., 2012, pp. 13-41. https://doi.org/10.1016/B978-0-12-415796-5.00002-8

[17] I. Synclair, "ASCII Codes," in Electronics Demystified, 3rd ed., Elsevier Inc., 2011, pp. 335-336.

[18] J. A. Farrell, "IBM PC Extended ASCII Codes," in From Pixels to Animation, Elsevier Inc., 1994, pp. 655-659.

[19] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," in Proceedings of the IRE, 1952.

[20] U. Nandi and J. K. Mandal, "Windowed Huffman Coding with Limited Distinct Symbols," Procedia Technology, vol. 4, pp. 589-594, 2012. https://doi.org/10.1016/j.protcy.2012.05.094

[21] H.-C. Huang and J.-L. Wu, "Windowed Huffman Coding Algorithm with Size Adaptation," in IEE Proceedings I - Communications, Speech and Vision, 1993.

[22] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," IEEE Transactions on Information Theory, Vols. IT-23, no. 3, pp. 337-343, May 1977. https://doi.org/10.1109/TIT.1977.1055714

[23] J. Ziv, "Coding Theorems for Individual Sequences," IEEE Transactionson Information Theory, Vols. IT-24, no. 4, pp. 405-412, July 1978. https://doi.org/10.1109/TIT.1978.1055911

[24] J. Ziv and A. Lempel, "Compression of Individual Sequences via Variable-Rate Coding," IEEE Transactions on Information Theory, Vols. IT-24, no. 5, pp. 530-536, September 1978. https://doi.org/10.1109/TIT.1978.1055934

[25] M. Mahoney, Data Compression Explained, Dell Inc., 2013.

[26] T. A. Welch, "A Technique for High-Performance Data Compression," Computer, vol. 17, no. 6, pp. 8-19, 1984. https://doi.org/10.1109/MC.1984.1659158

[27] H. N. Dheemanth, "LZW Data Compression," American Journal of Engineering Research (AJER), vol. III, no. 2, pp. 22-26, 2014.

[28] M. Burrows and D. J. Wheeler, "A Block-sorting Lossless Data Compression Algorithm," Digital Equipment Corporation, Palo Alto, Carlifornia, 1994.

[29] F. Rubin, "Arithmetic Stream Coding Using Fixed Frecision Registers," IEEE Transactions in Information Theory, Vols. IT-25, pp. 672-675, November 1979. https://doi.org/10.1109/TIT.1979.1056107

[30] G. D. Langdon, "An Introduction to Arithmetic Coding," IBM Journal of Research and Development, vol. 28, no. 2, pp. 135-149, March 1984. https://doi.org/10.1147/rd.282.0135

[31] J. J. Rissanen and G. G. Langdon, "Arithmetic Coding," IBM Journal of Research and Development, vol. 23, no. 2, pp. 149-162, March 1979. https://doi.org/10.1147/rd.232.0149

[32] P. G. Howard and J. S. Vitter, "Arithmetic Coding for Data Compression," in Encyclopedia of Algorithms, M. Kao, Ed., Springer US, 1994, pp. 65-68.

[33] T. Bell, I. H. Witten and J. G. Cleary, "Modeling for Text Compression," ACM Computing Surveys, vol. 21, no. 4, pp. 557-591, December 1989. https://doi.org/10.1145/76894.76896

[34] J. Abe, "Calgary Corpus," 2015. [Online]. Available: http://www.data-compression.info/Corpora/CalgaryCorpus/ #CalgaryCorpus. [Accessed 31 December 2015].

[35] D. Salomon and G. Motta, Handbook on Data Compression, 5th ed., London: Springer-Verlag, 2010. https://doi.org/10.1007/978-1-84882-903-9

[36] G. Blelloch, "Introduction to Data Compression," 31 January 2013. [Online]. Available: http://www.cs.cmu.edu/afs/cs/project/pscico-guyb/realworld/www/compression.pdf. [Accessed 31 December 2015].

[37] C. G. Bolivar-Vincenty and G. Beauchamp-Baez, "Modelling the Ball-and-Beam System from Newtonian Mechanics and from Lagrange Methods," in 12th Latin American and Caribbean Conference for Engineering and Technology (LACCEI'2014), Guayaquil, Ecuador, 2014.

[38] D. Colon, Y. S. Andrade, A. M. Bueno, I. S. Diniz and J. Balthazar, "Modeling, Control and Implementation of a Ball and Beam System," in 22nd International Congress of Mechanical Engineering - COBEM 2013, Ribeirão Preto, Brazil, 2013.

[39] M. Keshmiri, A. F. Jahromi, A. Mohebbi, M. H. Amoozgar and W.-F. Xie, "Modeling and Control of Ball and Beam System using Model Based and Non-model Based Control Approaches," International Journal on Smart Sensing and Intelligent Systems, vol. 5, no. 1, pp. 14-35, March 2012.

[40] M. Virseda, "Modeling and Control of the Ball and Beam Process," Lund Institute of Technology, Lund, Sweden, 2004.

[41] W. Wang, "Control of a Ball and Beam System," Adelaide, Australia, 2007.

[42] Scirra Ltd, "Create Games with Construct 2," 2015. [Online]. Available: https://www.scirra.com. [Accessed 13 April 2015].

[43] Node.js Foundation, "Node.js," 2015. [Online]. Available: https://nodejs.org/en/. [Accessed 12 November 2015].

[44] B. McKelvey, "Websocket," npm Inc., 28 September 2015. [Online]. Available: https://www.npmjs.com/package/websocket. [Accessed 15 November 2015].

[45] M. Murdocca, "node-lzw," September 2015. [Online]. Available: https://www.npmjs.com/package/node-lzw. [Accessed 10 November 2015].

8 Authors

Olawale B. Akinwale is lecturer in the Department of Electronic and Electrical Engineering and a member of the Remote Laboratories Developers Group of the Obafemi Awolowo University, Ile-Ife, Nigeria. Olawale majors in Instrumentation and Remote Laboratories with a keen interest in enhancing pedagogy, mobile devices and machine learning.

Lawrence O. Kehinde, a Professional Engineer, is a Professor of Electronic and Electrical Engineering and the Coordinator of the Remote Lab Development Group of the Obafemi Awolowo University (OAU), Ile-Ife, Nigeria. He worked in Techno-Managerial position as the Director of ICT at OAU for years. His major field is Instrumentation Designs and has designed various equipment. He was the founding Principal Investigator of the University's iLab research.

Article submitted 08 February 2017. Published as resubmitted by the authors 15 March 2017.