Building a Visible Light Communication Device with Widely Available Materials

Andrew A. Benington

***Abstract*— Visible Light Communication (VLC) represents a band of electromagnetic frequency that is both unregulated and underused. One of the benefits of VLC is that transmitters and photodiodes are already widely available in the form of LEDs. In this project, I built and studied the performance of an LED-based visible light communication system to assess how such a system using cheap and readily available materials might work, especially in an environment with unpredictable interference and noise. The device built was ultimately able to transmit at 33.3 bps at a distance of 4 feet, with a negligible amount of bit flips. If the distance was increased past that there was still some success, but failures were much more common.**

I. Introduction

With much of the electromagnetic frequency space unavailable to the public for wireless communication, it is only wise to make the best use of those which are available in every possible way. And because many widely used devices can already transmit and/or perceive visible light, the visible light spectrum is a rapidly growing field of research in wireless communication.

Compared to traditional methods of wireless communication, visible light communication has a unique set of perks and obstacles. As stated earlier, light devices are already widely used, and are therefore widely available. Light also benefits from the fact that light-based communication largely cannot be read through walls, providing greater security and spatial reuse than most other means of wireless communication.

With these advantages come some drawbacks, however. Environmental sources of light can provide interference, especially those which oscillate rapidly, as it is much harder to account for interference from those than from constant sources like the sun. There is also a tradeoff between an LED's beam angle and the distance at which a signal can be read, as a smaller angle means there must be a more direct line of sight, but the receiver can be further away, and vice versa.

Another set of obstacles came with the use of cheap, widely available equipment. The Arduino and LED receiver did not have the advantage of hardware pre-processing of the signal, which placed a larger burden on the software that only had a stream of voltages and timestamps to work with.

1.1. Applications

The most popular uses being explored for visible light communication are those where lights are already present, and/or where traditional radio communication might not be practical, as noted by Khan [1]. One of those uses is for underwater applications. Radio frequencies don't travel as well as light does underwater, so it can be more efficient to use visible light as well as acoustic methods for these underwater communications.

Hospitals are another candidate for VLC applications. Radio frequencies can't always be used in hospitals as they can interfere with sensitive equipment, but a system that sends and receives information via ceiling lights would not have the same issue [1]. A hospital already has plenty of ceiling lights in patient areas, and those areas are protected from interference by walls and curtains. One proposal of VLC application in a hospital is for cleaning robots, as rooms or stairways could use their lights to tell the robots to stay out, without needing any sort of artificial intelligence on the part of the robot [6].

One other possible use is in traffic, as existing traffic signals and car lights could be put to use as communication devices to send information about traffic conditions. Messages sent this way would not have to compete with cellular communications for bandwidth, and in theory, would require less infrastructure as traffic signals are already present [1].

The reliability of visible light wireless communication using inexpensive materials is an important area to research, as many potential applications for VLC are in devices that might be mass-produced for public use. It is relevant whether devices constructed with these materials can work in a typical environment with the interferences such an environment might bring. This project's goal was to create a hardware design and a software protocol that allows information to be sent from one machine to another over visible light, using materials that are widely available and comparatively inexpensive.

II. Motivation and Background

2.1. Challenges

At its most basic level, the challenges that are uniquely present for VLC are at the physical layer. The ability to send information efficiently over light must involve modulating a light source very quickly, possibly at multiple distinct intensities of light, and possibly in sync with other light sources nearby sending the same message over a wide area.

The first issue is that existing light devices are designed to provide a constant source of light, not to be quickly modulated. If they are quickly modulated off and on over a significant period of time, they might break or burn out much sooner than they would otherwise. They might also consume more energy than is desirable.

If multiple light sources must be synchronized, then one of two things must be true: either they are all on the same circuit, or there must be an effort to synchronize the clocks of all the light sources. While clock synchronization presents all of the usual challenges, having the lights on the same circuit is not necessarily easy. Ceiling lights, for example, might be on the same internal circuit of a room, but switches on wires carrying such a current might not be designed to modulate as quickly as VLC requires.

If two-way communication is desired, there must be some sort of uplink. As mentioned by O'Brien (2008), there is an idea to use an infrared uplink with the VLC downlink, but a more practical idea would be to use RF communications like Wi-Fi as an uplink instead [8]. At this point, however, there must be some justification for using a VLC downlink at all, rather than just using RF for both.

2.2. Previous Research

In the experiments conducted by Little (2008), two models were built: a one-way receiver that read signals from a ceiling light, and a two-way setup with a more traditional network configuration, with reliability protocols [4]. The former was similar to the hospital robot idea, where smaller amounts of information are sent to devices, without the devices talking back. The second model is more in line with what is known as Li-Fi, named such because of its similarities to Wi-Fi as a network technology.

While Little seemed to have some success with his build, I found it troublesome that there was no encoding beyond On-Off Keying. OOK presents issues when data with long strings of 0s or 1s is used, as they can lead to clock desynchronization and flicker in the case of VLC. I did not find any justification for this choice, except for the fact that they reported no visible flicker in their build, which only means they did not use data with these long strings of bits.

Unlike Little's build, one of the Sengunthar (2013) implementations had no software processing the signal, instead directly translating the digital light signal from the photodiode to an audio output using a specialized transistor switching circuit [3]. While this build is interesting in its own right and has its own applications, my interest was in the application of sending more types of data between machines than just audio.

I mention here two separate implementations, one hardware-only and one using software, because Sengunthar was not clear. He claims that his receiver sends the signal directly to a speaker, but also claims that he sent a 14-byte file from one machine to another. This distinction between implementations is an assumption I made in the absence of a clear explanation.

Additionally, it seems that the best performance that was achieved in the software implementation was a 14-byte file over a distance of 2 centimeters. To prove the utility of VLC, a better performance is needed, both distance-wise and filesize-wise.

Despite these issues, I modeled my implementation after Segunthar's software implementation flow, with a few key differences. In their implementation, they had some hardware signal processing before the signal reached the software, because they used GPIO for input as well. GPIO pins are digital only, and because I wanted to process the actual voltage with my software, I had to use an Arduino with analog input.

Segunthar's software implementation used Manchester encoding, which they justified using to avoid complications from long strings of 0s or 1s in the data. I also used Manchester encoding for the same reasons, with the additional intention of reducing flicker. My approach was not fast enough to reduce flicker entirely, but the Manchester encoding made significant improvements.

2.3. Analog-Digital Conversion

When converting from an analog signal to a digital bit sequence, it is necessary sample the analog signal at such a rate that the original sequence can be reconstructed faithfully, without aliasing. This minimum rate is typically $\frac{1}{period}$ [5]. However the signal I will be sampling is not a simple sine wave, and the necessity for clock synchronization means I will have to sample more often.

In the Liu study, a more advanced method of decoding bit data from VLC signals was proposed. Their 'Sampling-Based Receiver' involved building a gaussian ML model to determine the log-likelihood ratio of the probability of a 1-bit symbol to the probability of a 0-bit, given a set of signals in a slot [8]. The model kept track of the overall mean (μ) and standard deviation (σ) for both a 0-bit symbol and a 1-bit symbol, assumedly using the preamble to calculate the values. Given the average voltage x for a given slot, the LLR formula they used was

$$LLR = ln\frac{\sigma_0}{\sigma_1} - \frac{(x-\mu_1)^2}{2\sigma_1^2} + \frac{(x-\mu_2)^2}{2\sigma_1^2}$$

Liu's second receiver model was based on dividing the stream into 'chips', a useful model for a stream where the waveform is not continuous but is instead a series of 'pulses'. By creating a model using a Poisson distribution, the algorithm could once again make an informed prediction on the bit value of a given symbol.

III. Approach/Implementation

For my approach, I decided to take aspects from both of the builds used by Little and his collaborators. Similar to his second model, I used widely available, inexpensive equipment for both the transmitter and the receiver. But unlike his second model, I simply built a one-way communication system without any guarantee of reliability, as in his first model.

3.1. Approach Overview

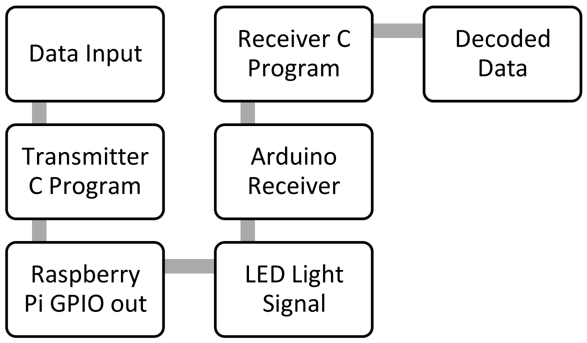

FIGURE 3.1

Figure 3.1 breaks down my sending and receiving process. Because of the limited processing power of the Arduino, I wanted the receiver's Arduino to devote all of its resources to sampling the voltage across the LED photodiode. Therefore I did the signal processing in a C program that communicated with the Arduino through its serial port.

Initially I tried using an Arduino to send the data as well. This worked well for the first 60 or so bytes, but that was as much data as the Arduino would receive from the transmitter C program. After struggling and failing to resolve this issue, I realized that the Raspberry Pi device that was sending data to the Arduino had its own controllable GPIO pins. By using system commands in the C program, I could directly control the LEDs without an intermediate Arduino.

The transmitter's circuit (Figure 3.2) was a set of three synchronized two LED lights with a 15° beam angle, controlled by the Raspberry Pi's GPIO pins as mentioned before. The receiver's circuit (Figure 3.3) was a single green LED light acting as a photodiode, connected to the Arduino that would constantly sample the voltage across the LED.

I chose green LEDS for a number of reasons. Green light is one of the three orthogonal visible light frequencies, being red, green and blue. Red LEDs tend to be less bright than their green and blue counterparts, which translates to a lower VLC range. Most “white” LED lights are actually blue LEDs with phosphors that convert some of the wavelength to that of yellow light. For these reasons, and in the interest of reducing interference from white LED ceiling lights, green LEDs were used in the circuit.

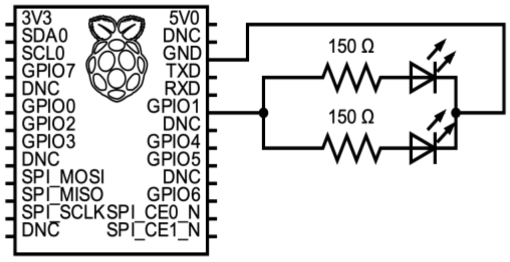

FIGURE 3.2

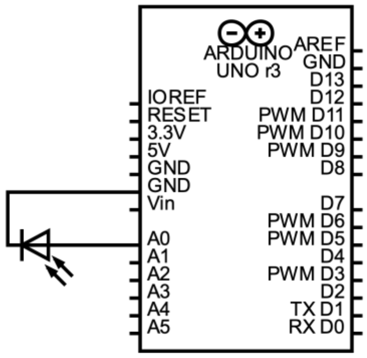

FIGURE 3.3

3.3. TRANSMITTER IMPLEMENTATION

The transmitter’s software used a standard Manchester encoding protocol. It began with a preamble and data beginning signal, which were necessary as detailed in section 3.4. It iterated through the data one byte at a time, and one bit at a time for each byte. For a given bit *b* it would turn on the LEDs for one slot and off for the next if *b* was 0, and vice versa if *b* was 1. After both slots were complete for *b* it would move on to the next bit, and once all 8 bits were sent it would move onto the next byte. When finished, it sent the byte 0x0, which worked because I was only sending ASCII data and so a 0x0 never occurred in the data itself.

3.4. RECEIVER IMPLEMENTATION

The protocol begins with the receiver constantly reading the voltages, calculating an average voltage and estimating the clock offset of the transmitter. When the transmitter starts transmitting, it begins with an alternating 1010101010… preamble, or repeated 0xff bytes in Manchester encoding, to help the receiver with these calculations. After a number of these signals, the byte 0x55 is sent, which lets the receiver know that the data transmission is about to begin. 0x55 was chosen because the Manchester encoding is 0110011001100110, a sequence that is easy to distinguish from the calibration preamble because of its consecutive double bits.

To actually translate the voltages back into bits, time is divided into ‘slots’ of a length agreed upon by both the transmitter and the receiver. In general, the smallest slot size I could manage while keeping an error-free bitstream was 15 ms. Over the course of a slot, I sum the differences of the measured voltage from the average voltage over the past 20 slots to get what I call the *SignalNet*:

$$SignalNet = \sum_{Slot} voltage - avg$$

A Manchester-encoded byte is 16 slots. Every 16 slots, I iterate through the past 16 *SignalNet* values two at a time, recording a 0 bit if the second value is larger than the first, and 1 otherwise. Because of the Manchester encoding it should almost always be the case that one of the two values is negative and the other one is positive, but in the case that the signal average abruptly changes, this method still gives the best guess as to which bit was received. It might seem that the summing the differences from the average is no different from just summing the signals themselves, but there are still some benefits, particularly when calculating the slot offset of the transmitter.

This implementation was significantly simpler than the one proposed by Liu [8]. As mentioned before, that model was intended to be used in low-illuminance environments, but the LEDs I used were very bright and far-reaching. In the interest of simplicity and comprehension, I used this simpler implementation.

3.5. Determining Slot Offset

When the voltage readings changed from consistently above average to consistently below average within the same time slot, the algorithm knew to adjust the timeslot offset. To avoid accidentally omitting or duplicating a slot after shifting the offset, the algorithm could only shift the offset a maximum of $\frac{slot\ length}{5}$ each slot.

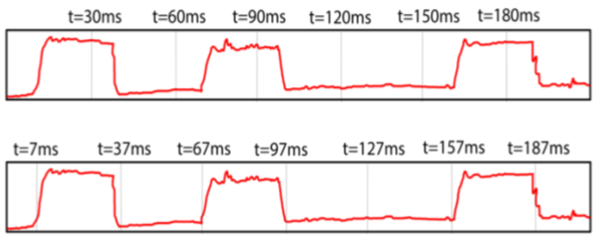

FIGURE 3.4

In the example shown in Figure 3.4, the significant edges were found at (t % 30 = 7), so the slots should ideally start at (t % 30 = 7). However, they would initially be shifted from (t % 30 = 0) to (t % 30 = 6), as the maximum shift size of $\frac{slot\ length}{5}$ means the full shift cannot happen until after 2 slots.

All of this computation was more than the Arduino could handle on its own, which is why the most the Arduino itself did as a receiver was constantly sample the voltage across the receiver LED and print those voltages, along with the time at which the voltage was sampled, to the Arduino's serial output. The signal processing described above was done in a C program that would read from the USB port connected to the Arduino, in order to access the reading printed to its serial output. At a baud rate of 115200 I found I could get a reading every millisecond, which is the highest granularity of time measurement the Arduino could measure.

The time slot I used was 15 milliseconds. With this length, I would get on average 15 readings per slot, enough to confidently decide each bit under ideal circumstances while giving some room for the offset to shift over time. For example, if the offset was 5 away from what it should be, and the algorithm shifted the slot offset up 5 milliseconds to compensate, the current slot might end 5 milliseconds earlier than anticipated but would still have around 10 voltage readings to accurately determine the bit of the slot.

This might mean that the offset might undergo a shift from 2->12, or 14->4, as the shift was calculated using modular arithmetic. If unchecked, this would mean the algorithm lets a slot go nearly twice as long, or cuts one extremely short respectively. I kept track of when the algorithm should move on to the next slot by keeping track of the index of the last slot, and checking the comparison

$$\frac{t - offset}{slotSize} > lastSlot$$

To account for the wrap-around shifts, I would follow a wraparound from low to high by incrementing *lastSlot*, and follow the same from high to low by decrementing *lastSlot*.

IV. Evaluation/Implementation

Initially, I was concerned that environmental light (from outdoors, ceiling lights, etc.) would be a major stumbling block. This was not the case. I get a similar quality signal in ambient light to when I am completely in the dark. This is likely for two reasons. The first is that because the range of frequencies the receiver's green LED reacts to is relatively narrow. The second is the fact that those wavelengths are green light, which is not as common in white LED ceiling lights as blue wavelengths.

There were factors that influenced the reliability of the communication, however. One metric I used to measure the stability of a communication session was what I called the "flip rate". After a slot ended, I would count the number of voltage readings in that slot that were on a different side of the average than they should have been for that bit. For example, if 2 out of the 15 voltage readings in a slot that was determined to be a 1 bit were below the average voltage, the flip rate for that slot would be 0.133. I averaged the flip rates over all the slots and printed it out at the end of a communication session.

For the following tests, I transmitted a 200-character-long string, specifically the first verse to *Oops!...I Did It Again* by Britney Spears. The string included lowercase and uppercase letters, as well as punctuation and newlines.

4.1. Effect of Slot Size on Error Rate

The shortest slot size I could get without any bit errors, or at least without any bit errors after 15 minutes of constant communicating, was 15 ms. For a purely reliable transmission, and for the experiments in the next sections, that is the slot size I used.

A single bit flip is not too worrisome, as some framing and error correction can fix that as long as the errors aren't too consistent. The more troubling errors I ran into as I reduced the slot size below 15 ms were consistent data corruption after the first bit flip, meaning an entire slot was omitted leading to garbage data. An example of this can be seen in Figure 4.1.

```
1010 0010 1110 0001 0011 1010
1010 0010 110 0001 0011 1010
```

FIGURE 4.1

After 10 transmissions of the sample sequence with a 14 ms slot size, the message was received with no errors 9 times. The other time, however, the message was almost entirely garbage data after the begin symbol, implying a slot was skipped over.

4.2. Effect of Distance Between Devices

Distance (feet)	Single Bit Errors (out of 1600)	Success Rate (Full Message)	Bytes Unti Garbage
4.5			
5.0	0.8	0.75	141
5.5	--	0	87.25

TABLE 4.1

The farther distance the devices could be from each other while consistently sending the entire message (i.e., no skipping slots that lead to garbage data) was 4 feet. Any farther than that, and only some of the sent messages would get to the receiver without part or all of them turning into garbage. The rate at which I found a message was received successfully is referred to in Table 4 as the Success Rate. In those messages that became garbage partway through, I found the average number of bytes before that happened and recorded it in Bytes Until Garbage. As suspected, the greater the distance between the devices, the shorter it would be able to send a stream of accurate bytes before a slot was lost.

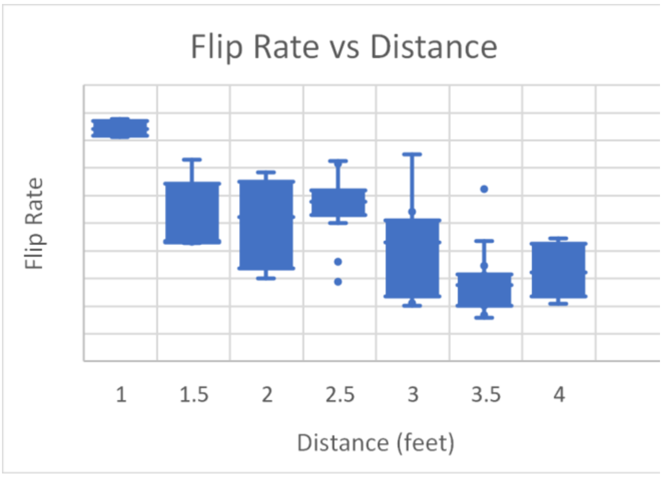

FIGURE 4.2

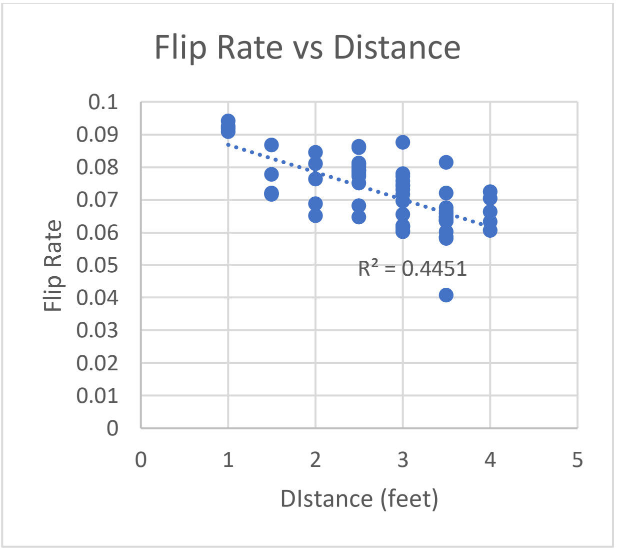

FIGURE 4.3

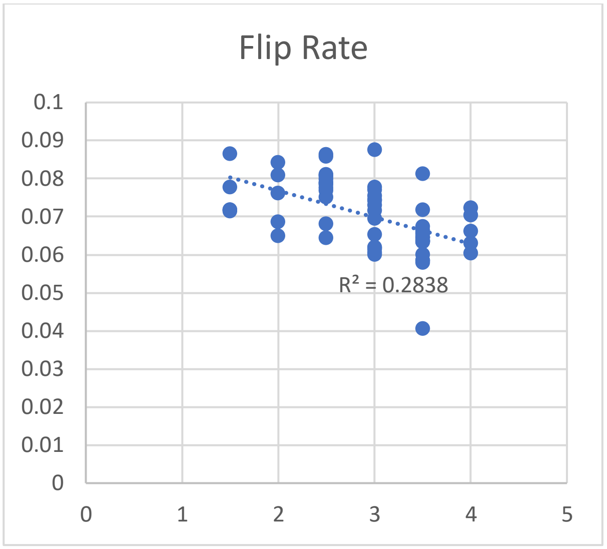

FIGURE 4.4

Interestingly enough, there was not a significant relationship between the flip rate and the distance between the transmitter and the receiver. As shown in Figure 4.2, the error bars for each distance significantly overlap, with the exception of the closest distance of 1 foot, which for some reason seemed to consistently have the highest flip rate.

After calculating a linear regression on the scatter plot shown in Figure 4.3, it was clear that any trend was weak, as the R^2 value was only 0.4451. If the data from 1 foot was omitted, the R^2 value was only 0.2838 (Figure 4.4), meaning there was likely little relationship between the distance and the flip rate as defined above.

When examining the individual signals during the debugging process, I found that voltages on the wrong side of the average were usually at the beginning or the end of a slot, meaning the offset calculation was slightly inaccurate. Because the flip rate was not significantly affected by distance, at least within the range of reliable transmission, it must be the case that differences in distance within this range do not have much effect on the waveform, and therefore offset calculation.

As for the higher flip rate at a distance of 1 foot, this likely has something to do with the reaction time of the LED as a photodiode. As shown in Figure 4.5, the voltage across the receiver's LED did not immediately go from high to low, instead gradually sloping back down to the minimum. At a shorter distance, and thus a higher intensity of light, it's likely that it took even longer for the LED to read the low value again, leading to more signals above the average towards the beginning of a 0-bit slot.

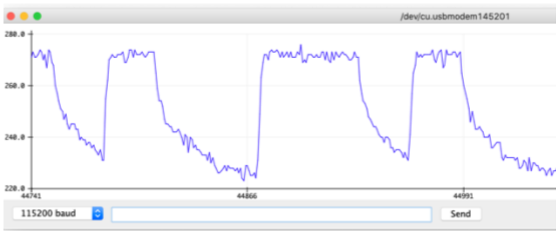

FIGURE 4.5

4.3. Environmental Interference

The most noticeable effects on the receiver's performance were when certain cables were carrying a current nearby. The two major culprits were my laptop charger and an ethernet cable, both of which caused significant additional oscillation in the voltage the Arduino read. These effects are likely because these cables are poorly grounded, and the magnetic field produced by these wires is inducing extra currents in my receiver.

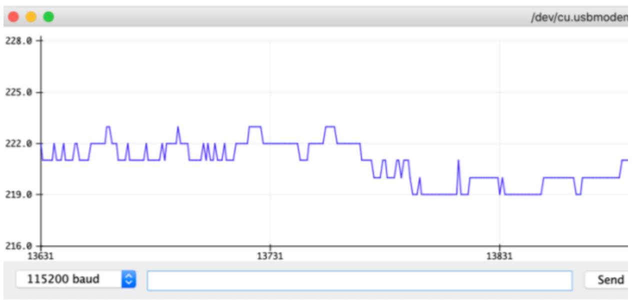

FIGURE 4.6

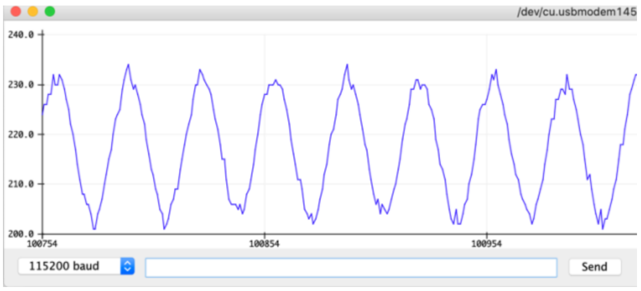

FIGURE 4.7

Figure 4.6 depicts the voltage readings on the receiver when the light is constantly on (not turning off at all), which induces a voltage that varies about 2-3V. Figure 4.7 is the that same reading, with the light still on, after a MacBook charger was plugged in nearby.

My receiver is made using cheap wires and a common LED, so it is possible that materials of higher quality would not have as much trouble. And if better quality and more expensive materials are necessary to deal with the type of environmental interference a typical household or workplace object like a computer charger provides, this may prove a challenge when developing technology for public use. At the same time, there could be more resource-savvy ways to construct an interference-proof receiver than my implementation.

V. Conclusions/Further Directions

5.1. Future Experimentation

While I was happy with the performance of the green LEDs, I would be more satisfied with concrete data weighing the performance of different colored LEDs against each other. It's possible that in a shorter-distance situation, red LEDs might be preferable, if the higher flip rate of the green LEDs at a short range is something to be concerned about.

One aspect of the devices that I believe contributed to the success at distances of up to

4 feet was the narrow angle of the LEDs, which focused the beam and let it travel further. Experiments with a different LED angle, or with the accuracy of the devices at an angle other than head on, might be useful. In a real life application, it can't be counted on that the transmitter and the receiver are lined up exactly.

5.2. Data Rate Improvements

There are a great many ways the devices could be modified to carry more data at once. The first is to introduce color and take advantage of the full visible light spectrum. Using color-shift keying or color division multiplexing would significantly increase the baud, and therefore the bitrate. It would be possible to use similar technology to that which I used in these devices, as LEDs only react to a narrow frequency range slightly under their emission frequency. With an additional N orthogonal color channels, the bitrate would be multiplied by 2^N.

Aside from the use of color, multiplexing could also be achieved with a carefully constructed set of LED arrays. An array of LEDs acting independently would run the risk of cross-contamination of signals, but this could be curbed with enough spacing between LEDs, by using lasers or LEDs with significantly narrower viewing angles, or mirrors and lenses to focus the light. If color is included, the array could be constructed so that neighboring LEDs are of orthogonal colors, further reducing the risk of interference.

In regard to encoding, it would be worth investigating how 4B/5B encoding performs compared to Manchester encoding. I chose to use Manchester for three main reasons: to maintain an average unskewed by long strings of the same bit, to better maintain clock synchronization, and to reduce flicker. There is obvious flicker as the device works right now, so that didn't matter in the end. If the device were to approach the speed necessary to eliminate visible flicker, it would need to be 1.5x faster using 4B/5B, because of the possibility for 3 of the same bit in a row instead of just 2. This possibility for 3 in a row might also influence the accuracy of the running average or the clock synchronization, so the only way to tell is through testing and comparison.

5.3. Reliability Improvements

As mentioned in section 2.2, Liu et. all developed two formulas for signal decoding using statistical models, whereas I simply took a sum of the signals over one slot and compared it to the sum of the next to determine the Manchester signal [8]. If implemented, one or both of these algorithms might alleviate the reliability issues I had with a slot size of less than 15 ms.

Different applications of VLC require different degrees of reliability. One-way communications from ceiling lights to objects below rely on a bitstream similar to the one implemented here, while "Li-Fi" requires the reliability one would expect from Wi-Fi. In both cases, reliability could be improved with a framing scheme and error detection/correction.

For even more reliability, framing could be using in conjunction with a back-channel. This could be accomplished with another set of visible light LEDs, or it could be accomplished with infrared LEDs, which would be more useful and convenient in a real-world setting where the receiver is a personal device [7].

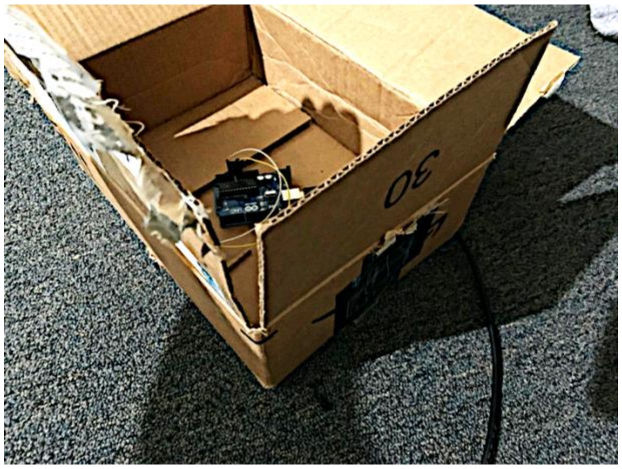

FIGURE 5.1

As discussed in section 4.3, the clarity of the signal was muddled by environmental noise caused by nearby devices and their electromagnetic fields. My build used relatively long and unshielded wires to connect the receiver's LED to the Arduino (Figure 5.1), and I believe these wires acting as "antennas" are why the interference was picked up. If a receiver could be built without wires like these, or perhaps using a Faraday cage, it is possible that the weaker signal of a further distance could still be read without error.

5.4. Miscellaneous Improvements

Because implementation was only for ASCII data, it was only guaranteed to work for files without 0x0 bytes. By changing the terminating symbol, it would be possible to send any kind of file over these devices. One way might be to simply turn the light completely off, as that would never happen in Manchester encoded data.

If instead I used 4B/5B encoding, I could reserve one of control symbols not only for the terminating symbol, but for the beginning symbol as well. This means that multiple files could be sent in the same session, without the need for another preamble.

5.5. Conclusion

At a distance of 4 ft, and a slot size of 15 ms, which is 33.3 bps under Manchester encoding, my device consistently could send a message over light without bit errors. This was unaffected by normal environmental light, natural or artificial, although it was affected by electromagnetic fields nearby induced by other devices.

Unfortunately, I was unable to reach a frequency that would be imperceptible to the human eye. If the device as it exists now used a ceiling light instead of LEDs, it would cause eye strain and generally unpleasant conditions for anyone nearby. But as a directed, smaller device, it would work just fine, perhaps comparable to IR communication on early cell phones.

References

[1] Khan, Latif Ullah. “Visible Light Communication: Applications, Architecture, Standardization and Research Challenges.” *Digital Communications and Networks*, vol. 3, no. 2, May 2017, pp. 78–88. *DOI.org (Crossref)*, doi:10.1016/j.dcan.2016.07.004.

[2] Seeber, R., Ulrici, A. Analog and digital worlds: Part 1. Signal sampling and Fourier Transform. *ChemTexts* **2,** 18 (2016). https://doi.org/10.1007/s40828-016-0037-1

[3] Sengunthar, R. Patil, S. Bhosale, R. Khodaskar and S. N. Talbar, "Visual Light Communication Using LED," *2013 Texas Instruments India Educators' Conference*, Bangalore, 2013, pp. 121-124, doi: 10.1109/TIIEC.2013.28.

[4] T. D. C. Little, P. Dib, K. Shah, N. Barraford and B. Gallagher, "Using LED Lighting for Ubiquitous Indoor Wireless Networking," *2008 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, Avignon, 2008, pp. 373-378, doi: 10.1109/WiMob.2008.57.

[5] Gudino, Miguel. “How Analog-to-Digital Converters (ADCs) Work.” *Arrow.com*, Arrow Electronics, 17 Apr. 2018, www.arrow.com/en/research-and-events/articles/engineering-resource-basics-of-analog-to-digital-converters.

[6] R. Murai *et al.*, "A novel visible light communication system for enhanced control of autonomous delivery robots in a hospital," *2012 IEEE/SICE International Symposium on System Integration (SII)*, Fukuoka, 2012, pp. 510-516, doi: 10.1109/SII.2012.6427311.

[7] D. C. O'Brien, L. Zeng, H. Le-Minh, G. Faulkner, J. W. Walewski and S. Randel, "Visible light communications: Challenges and possibilities," 2008 IEEE 19th International Symposium on Personal, Indoor and Mobile Radio Communications, Cannes, 2008, pp. 1-5, doi: 10.1109/PIMRC.2008.4699964.

[8] X. Liu, C. Gong, S. Li and Z. Xu, "Signal Characterization and Receiver Design for Visible Light Communication Under Weak Illuminance," in *IEEE Communications Letters*, vol. 20, no. 7, pp. 1349-1352, July 2016, doi: 10.1109/LCOMM.2016.2558498.