Optical Fiber Project: Design, Implementation, and Communication System Development

Aritra Mukhopadhyay
National Institute of Science Education and Research Bhubaneswar, Odisha 751005, India 3rd year, Integrated M.Sc. Physics Roll No.: 2011030
(Dated: March 8, 2024)

This report summarizes our optical fiber project, encompassing fiber preparation, parameter measurements, and the development of a novel fiber optic communication system.

Our journey involved overcoming challenges in fiber preparation and light coupling, culminating in the successful development of a robust communication system. We introduced a Pulse Width Modulation (PWM) scheme to ensure adaptability to varying transmission adversaries.

This project sheds light on various aspects of optical technology, offering valuable insights into the challenges and complexities associated with optical fiber experiments and communication systems. Our work has led to a deeper understanding of optical fibers and their practical applications.

I. THEORY

In the realm of optical communication, fiber optic systems play a pivotal role in transmitting data over long distances with exceptional speed and reliability. These systems harness the unique properties of optical fibers to carry data in the form of light pulses. In this section, we delve into the theory behind the design and operation of a fiber optic communication system.

A. Fiber-end Preparation and Light Coupling

Fiber-end preparation and light coupling represent the critical initial steps in any optical fiber experiment or communication system. The quality of these preparations directly impacts the efficiency of light transmission through the optical fiber. This section discusses the significance of fiber-end preparation and the essential conditions that must be met.

The optical fiber's core function is to transmit light efficiently from one end to another while minimizing losses. Achieving this requires careful attention to the preparation of the fiber's end faces. The primary reasons for meticulous fiber-end preparation are as follows:

1. **Minimizing Light Losses:** The cut face of the optical fiber must be smooth and free from imperfections. Any irregularities or roughness on the cut surface can scatter and absorb light, leading to signal attenuation and decreased transmission efficiency.
2. **Maximizing Light Coupling:** Light coupling refers to the efficient transfer of light from an external source, such as a laser or LED, into the optical fiber. Properly prepared fiber ends ensure that a significant portion of incident light is coupled into the fiber, enhancing signal strength.
3. **Reducing Signal Distortion:** Irregularities in the fiber-end face can cause signal distortion, leading to data errors. Ensuring a flat and smooth end face helps maintain signal integrity during transmission.

1. *Conditions for Fiber-end Preparation*

To achieve optimal light transmission and coupling efficiency, certain conditions must be met during fiber-end preparation:

1. **Smooth Cut Face:** The cut face of the optical fiber must be smooth and free from any visible imperfections or damage. Any roughness or irregularities on this surface can scatter and attenuate light.
2. **Perpendicularity to Fiber Axis:** The end face of the optical fiber should be perfectly perpendicular to the fiber's axis. This alignment ensures that light enters and exits the fiber without deviation, reducing the risk of signal loss due to misalignment.
3. **Cleanliness:** The fiber-end face should be thoroughly cleaned to remove any contaminants or debris that could obstruct light transmission or damage the fiber surface.
4. **Flatness:** The end face should be flat and free from warping to prevent light distortion during propagation.

In summary, fiber-end preparation and light coupling are crucial steps in optical fiber experiments and communication systems. Proper preparation ensures minimal light losses, maximum light coupling efficiency, and reduced signal distortion, contributing to the overall success of optical fiber-based applications.

B. Numerical Aperture Measurement

Numerical Aperture (NA) is a fundamental optical parameter that characterizes the light-gathering ability and light-confining capacity of an optical system, such as an optical fiber. A higher NA fiber generally allows us to transfer a bigger bandwidth of data.

NA is a dimensionless number that defines the cone of light acceptance and the ability to capture light from a wide range of angles. NA is a crucial factor in determining the performance of optical components, including optical fibers, microscopy objectives, and lenses. Mathematically, the numerical aperture (NA) is defined as:

$$NA = n \cdot \sin(\theta)$$

Where:

- NA is the numerical aperture.
- n is the refractive index of the medium in which the optical system operates (typically air or a specific medium).
- θ is the half-angle of the maximum cone of light that the optical system can accept.

A higher numerical aperture indicates a greater ability to capture light from a wider range of angles, resulting in better light-gathering efficiency and increased resolution in optical systems.

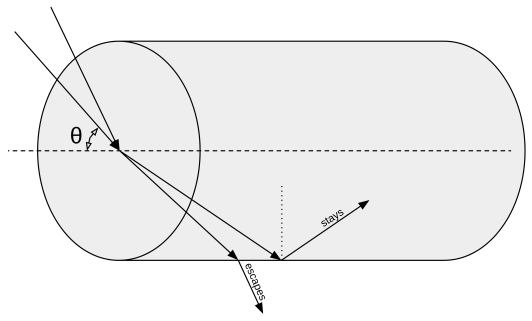

FIG. 1: Illustration of Numerical Aperture

To measure the numerical aperture of the optical fiber, we didn't have proper measurement equipments. So we had to improvise on some pre-existing methods. We passed laser light through on end of the fiber, so a we got a cone of light at the other end. Rays falling at a greater angle had already escaped the fiber through it's wall (as described in Figue 1). Then we cut the cone of light perpendicular to it's principle axis. We get a circle. If we get the diameter (D_1 and D_2) of two such circles and the distance between their taking points (Z) on the cone principle axis, we can calculate the numerical aperture of the fiber. The formula is:

$$NA = n \cdot \sin\left(\tan^{-1}\left(\frac{D_1 - D_2}{2Z}\right)\right) \tag{1}$$

By precisely measuring the numerical aperture, we gained insights into the optical fiber's bandwidth capacity and its suitability for specific applications. Numerical aperture measurements are vital in designing optical communication systems and selecting optical components that optimize light transmission.

C. Fiber Optic Communication and Different Modulation Techniques

Fiber optic communication systems revolutionize the way data is transmitted, offering advantages in terms of bandwidth, signal quality, and immunity to electromagnetic interference. At the core of such a system is the optical fiber, a medium that guides light through total internal reflection.

The basic components of a fiber optic communication system include a light source, typically a laser or light-emitting diode (LED), that generates optical signals. These signals travel through optical fibers, which serve as the transmission medium. Optical detectors, such as photodiodes, are used at the receiver end to convert the optical signals back into electrical signals for processing and decoding.

Fiber optic communication systems transmit data as optical signals through optical fibers. To carry information effectively, these systems employ modulation techniques, a fundamental concept in communication engineering. Modulation is the process of varying one or more properties of a carrier signal, such as amplitude, frequency, or phase, in accordance with the information to be transmitted. It serves two essential purposes:

1. **Compatibility**: Optical fibers primarily transmit light signals, which cannot directly represent binary data (0s and 1s). Modulation allows data to be superimposed onto the carrier signal in a compatible format for transmission.

2. **Noise Immunity**: Modulation helps mitigate the effects of external noise and interference during signal transmission. By varying specific signal properties, it becomes easier to distinguish between different signal levels and retrieve the original data accurately.

In our exploration of modulation techniques for our fiber optic communication system, we considered several options, including Pulse Width Modulation (PWM), Pulse Position Modulation (PPM), Pulse Amplitude Modulation (PAM), Manchester encoding and Differential Manchester encoding.

1. Manchester Encoding and Differential Manchester Encoding

Manchester encoding [3] and its derivative, Differential Manchester encoding, are popular modulation techniques used in digital communication systems. They operate by altering the polarity of the carrier signal within each bit interval to represent binary data. In Manchester encoding, a transition from a high to a low voltage (or vice versa) within the middle of a bit interval signifies a '1', while the absence of such a transition represents a '0'. Differential Manchester encoding, on the other hand, utilizes transitions at the start or end of a bit interval to encode '1' bits, with the lack of transitions denoting '0' bits.

These techniques offer several advantages, including self-clocking properties that aid in bit synchronization and robust noise immunity. However, they entail complex encoding and decoding processes due to the requirement for accurate transition timing. Consequently, implementing Manchester encoding and Differential Manchester encoding can be challenging, making them less suitable for our project, where simplicity and ease of implementation were prioritized. The Manchester encoding waveform is shown in Figure 2 [8].

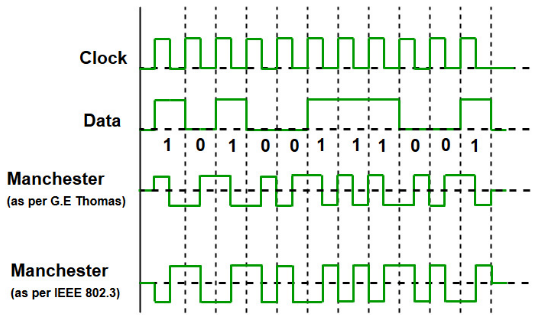

FIG. 2: Manchester Encoding Waveform

2. *Pulse Position Modulation (PPM) and Pulse Amplitude Modulation (PAM)*

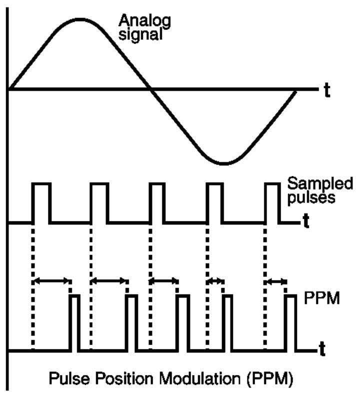

FIG. 3: Pulse Position Modulation (PPM) Waveforms

Pulse Position Modulation (PPM) [4] relies on precise positioning of pulses within a bit interval to convey binary data. Each unique pulse position corresponds to a specific data value. Conversely, Pulse Amplitude Modulation (PAM) [5] involves varying the amplitude of signal pulses to represent data. Higher amplitudes indicate '1' bits, while lower amplitudes denote '0' bits.

Both PPM and PAM necessitate high precision in pulse timing or amplitude detection, respectively. Achieving such precision can be challenging, particularly with the equipment at our disposal. Additionally, PPM and PAM are more susceptible to noise interference, potentially degrading signal quality. These factors led us to opt for the simplicity and noise immunity offered by PWM for our fiber optic communication system. The PPM signal is shown in Figure 3 and the PAM signal is shown in Figure 4.

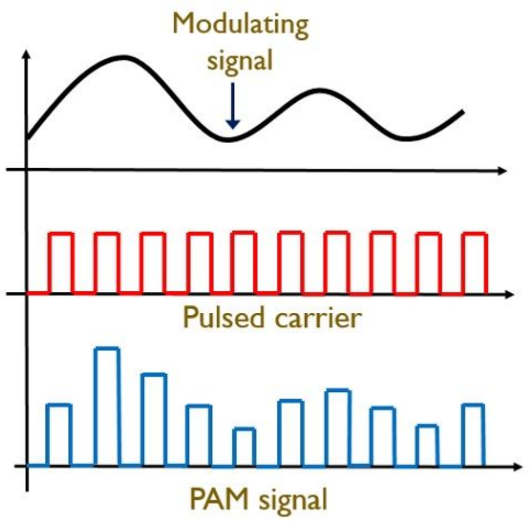

FIG. 4: Pulse Amplitude Modulation (PAM) Waveforms

3. *Pulse Width Modulation (PWM)*

Pulse Width Modulation (PWM) [6] is a modulation technique that varies the width or duration of pulses within a bit interval to convey binary data. In PWM, a '1' bit is represented by a pulse with a longer duration, while a '0' bit corresponds to a shorter pulse duration. This modulation technique offers simplicity and noise resilience, making it well-suited for our project.

PWM's effectiveness lies in its ability to provide clear distinctions between '1' and '0' bits, even in the presence of noise. It is relatively straightforward to implement, as encoding and decoding involve measuring pulse durations. For our fiber optic communication system, PWM was chosen as the modulation technique due to its ease of implementation and robust performance. The PWM waveform is shown in Figure 5.

At the receiver end, the demodulation process extracts the original data from the modulated optical signal. The choice of modulation scheme impacts the receiver's ability to accurately interpret the transmitted data.

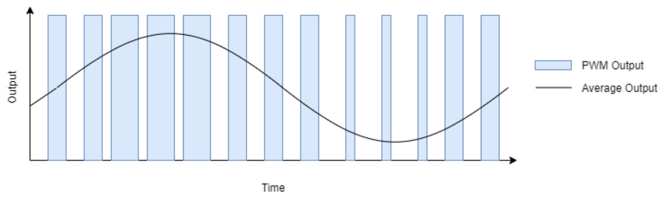

FIG. 5: Pulse Width Modulation (PWM) Waveform

In our experiments, we explored the design and implementation of a fiber optic communication system, including the modulation and demodulation of optical signals. Our goal was to gain hands-on experience in creating a functional optical communication link, using a combination of optical components and electronic control to transmit and receive data reliably.

By developing a working fiber optic communication system, we aimed to bridge the gap between theoretical knowledge and practical application, gaining insights into the intricacies of optical signal transmission.

D. Technique we used

The optical fiber communication system consists of two main components:

- **Bob:** The sender endpoint responsible for encoding a user-provided message into binary format and modulating the signal for transmission. Bob's program shall prompt the user for a message, which is a string of characters. The message provided by the user shall be converted into a binary representation (ASCII), where each character in the message contributes 8 bits (1 byte) to the binary message. Bob shall modulate the binary message into a format suitable for transmission through the optical fiber. The modulated signal shall be sent through the optical fiber for transmission.

- **Alice:** The receiver endpoint responsible for receiving the modulated signal, demodulating it, decoding the binary message, and displaying it on the screen. Alice's program shall receive the modulated signal from the optical fiber. The received signal shall be demodulated to extract the encoded binary message. Alice shall decode the binary message to reconstruct the original string of characters. The decoded message shall be displayed on the screen for the user to view.

Initially, our plan for encoding and transmitting messages in the optical fiber communication system was to turn an LED on for a time t to represent a `'1'` and keep it off for the same time t to represent a `'0'`. However, we encountered a significant flaw in this approach. When there were multiple consecutive `'0'`s or `'1'`s in the message, Alice, the receiver, could not distinguish them without knowing the exact value of t. Unfortunately, due to errors and noise in the system, the practical value of t could vary throughout the transmission. This inconsistency made it challenging for Alice to decode the message correctly.

To address this issue, we devised a new, more robust modulation scheme that might even allow Bob to change the transmission speed during the signal without Alice needing to know the precise value of t.

nitially, our plan for encoding and transmitting messages in the optical fiber communication system was to turn an LED on for a time t to represent a `'1'` and keep it off for the same time t to represent a `'0'`. However, we encountered a significant flaw in this approach. When there were multiple consecutive `'0'`s or `'1'`s in the message, Alice, the receiver, could not distinguish them without knowing the exact value of t. Unfortunately, due to errors and noise in the system, the practical value of t could vary throughout the transmission. This inconsistency made it challenging for Alice to decode the message correctly.

To address this issue, we devised a new, more robust modulation scheme that allows Bob to change the transmission speed during the signal without Alice needing to know the precise value of t.

Among all the techniques explored by us, digital data encoded in PWM was the most suitable to us. Only it does not rely upon the precise position of the pulse, but the relative duration of the pulse.

II. THE MODULATION SCHEME

In the new method, Bob sets the speed, represented by a value t, which denotes the time length of one bit of the message. Alice doesn't even need to know the length of one bit, she will figure it out on her own.

At the beginning of each bit, Bob always turns the LED on. Thus transition from `'0'` to `'1'` marks the start of a bit. For a `'0'`, he turns the LED off after $t/4$ time and keeps it off for the remaining $3t/4$ time of the bit. For a `'1'`, he keeps the LED on for $3t/4$ time and keeps it off for the remaining $t/4$ time. The techniue is illustrated in Figure 6.

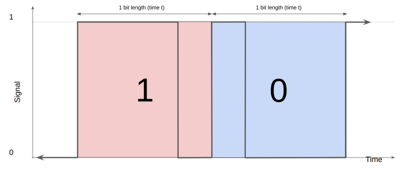

FIG. 6: Illustration of our technique

Thus, from Alice side, if LED is kept lit for a longer duration between two turning on of the LED, she understand it is a `'1'`, and if it is kept off for a longer duration, it signifies a `'0'`.

Alice samples the signal at a much higher rate, approximately every $^t/_n$ time (where n is a large number, such as 100 or 200). At first she continuously records the voltage values, classifying them as `'1'` if they exceed a predefined **threshold** else `'0'`.

She then processes these readings. Since she knows that a transition from `'0'` to `'1'` indicates the beginning of a bit, she segments the recorded data into chunks splitting them at a `"01"` . Each chunk conforms to the pattern `'1*0*'` (starting with one or more`'1'`s followed by `'0'`s). Alice then counts the number of `'1'`s and `'0'`s in each chunk. If there are more `'1'`s than `'0'`s, she decodes the bit as `'1'`; else `'0'`.

III. OBSERVATIONS AND CALCULATION OF NUMERICAL APERTURE

We used **multimode** fiber for all of our experiments. We cut a **25cm long fiber**, prepared the end, and couple it up with the microscope lenses.

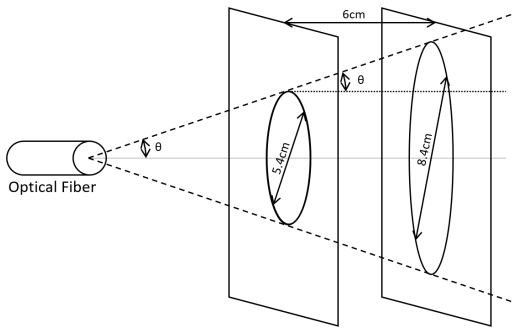

FIG. 7: Illustration of Numerical Aperture

We used the setup shown in Figure 7 to measure the numerical aperture of the fiber. We used a **He-Ne laser** as the light source. We then used a **microscope objective** to focus the laser on the fiber. We used a **graph paper** to mark the light cone formed on the other end of the fiber. This graph paper is shown in figure 8. We then used equation 1 to calculate the numerical aperture of the fiber.

IV. OPTICAL COMMUNICATION

A. Software Setup

For the experiment, we employed two Windows laptops, representing the endpoints of our communication system, Alice and Bob. Additionally, we utilized `'Python 3.10'` as our programming language. To control the LEDs used for signal modulation, we used `'Arduino UNO'` [1] microcontrollers connected to the computers on each side of the communication link.

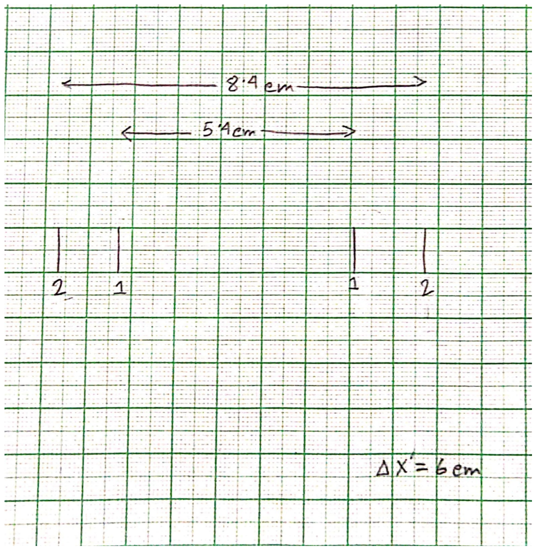

FIG. 8: Data marked on Graph Paper for easy measurement

1. **Upload of Standard Firmata:** We uploaded the `'StandardFirmata.ino'` [2] firmware code to each Arduino device using the Arduino IDE. This code enables the Arduino to receive and interpret commands sent from Python. But for finding the threshold value we use native Arduino Code. So we have to upload the Standard Firmata code to the arduino again everytime we upload the threshold code.

2. **Arduino IDE Setup:** We installed the `'PyFirmata'` [7] library using pip in a new virtual environment. This library allows Python to communicate with the Arduino boards.

3. **Serial Communication Protocol:** Python established communication with the Arduino boards using the 'Firmata' Serial communication protocol. The Standard Firmata program running on the Arduino boards facilitated the reception and decoding of commands transmitted by Python.

B. Hardware Setup

We used the same **25cm multimode** fiber here as well. We cut the fiber of required length, stripped it and prepared the ends of the fiber with appropriate equipments. We then mounted the fiber on the optical bed, and aligned the microscope objective lenses as shown in Figure 9.

At Bob's end, we attached an LED to the digital pin 13 of the Arduino board, through a 330Ω resistor. We had soldered the resistance to the Anode of the LED and put heat shrink on it for stable connection. Then we discovered that the LED perfectly fits at the intake hole of the objective lens, so we attached the LED to the lens

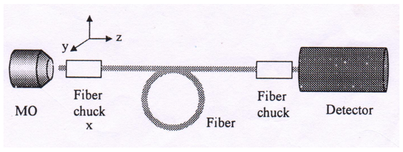

FIG. 9: Setup for launching light into optical fiber using microscope objective lenses

using tapes. We connected the Arduino to the laptop using a USB cable. The setup is shown in Figure 10.

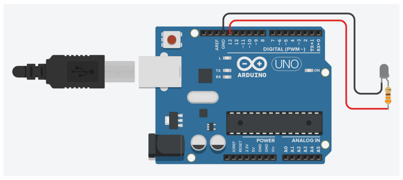

FIG. 10: Illustration of Hardware Setup at Bob's end

At Alice's end, we didn't have any holder to hold the photodiode at the correct height. So we had to improvise. We used a large breadboard, put the photodiode and the $10k\Omega$ resistor on it, conected the wires. Then we put the breadbard standing on it's short end. It kept standing pretty stable. Next we made the connection as shown in Figure 11.

The photoresistor changes it's resistance with the intensity of light falling on it. So to measure the change in resistance, we connected a comparable resistance to the photoresistor in series, biased the two ends, and measure the potential at the junction. In darkness the resistance of photoresistor is very high, so the junction potential is zero, (the potential drops accross the photoresistor). When light falls on the photoresistor, it's resistance decreases to $1.6k\Omega$, so the potential at the junction increases which we measure using the analog pin A0 of the Arduino.

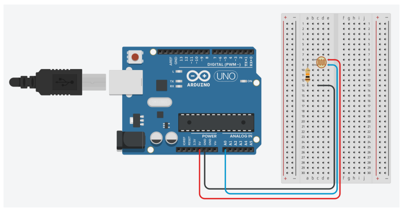

FIG. 11: Illustration of Hardware Setup at Alice's end

Images 10 and 11 have been prepared using TinkerCad.

C. Determining the Threshold

```
void setup() {
   pinMode(13, OUTPUT);
}

void loop() {
   digitalWrite(13, HIGH);
   delay(500);
   digitalWrite(13, LOW);
   delay(500);
}
```

Listing 1: Code to determine threshold

```
void setup() {
   pinMode(A0, INPUT);
   Serial.begin(9600);
}

void loop() {
   Serial.println(analogRead(A0));
   delay(30);
}
```

Listing 2: Code to determine threshold

We ran the arduino code 1 in the Bob device. It will generate a square wave of frequency 1Hz. We connect the LED to the pin 13, and the led will blink with that signal. We then ran the arduino code 2 in Alice device and opened the serial plotter in Arduino IDE. There we can see the voltage read by the analog pin A0. We then adjusted the distance between the LED and the photodiode, to maximise the maximum intensity read.

We observed that, at dark, the A0 reading sticks to 0, and when the LED is lit, the maximim reading comes to be about 30. (**Note:** this value is the 10bit integer value. Voltage reading varies from 0 (0V) to 1023 (5V) and linearly in between). But the photodiode had a slightly slow response curve, and took some time to grow to 30 so we decided to keep the threshold pretty low (4 used here) for best results. We didn't go below 4 that because in dark the reading was fluctuating between 0 to 1, and we didn't want to risk it.

Then we put the threshold value in the Alice's code and started communication. The Alice's code is shown in 4 and Bob's code is shown in 3. The codes have ample comments in them and are pretty self explanatory so we are not going to discuss them here.

```
from pyfirmata import Arduino
import time

board = Arduino('COM3')
LED = board.get_pin('d:13:o')  # led to pin 13
DURATION = 0.01 # length of one bit in seconds
PAUSE = DURATION/10  # length of pause to ensure uniformity


def send_one_bit(LED, bit:bool):
    print("1" if bit else "0", end = "")
    n = int(DURATION // PAUSE) + 1
    if bit is True:
        arr = [1]*(3*n//4)   + [0]*(n//4 + 1)
    else:
        arr = [1]*(n//4 + 1) + [0]*(3*n//4)
    for i in arr:
        LED.write(i)
        time.sleep(PAUSE)
```

```python

def string_to_binary(input_string):
    binary_string = "0001"
    # we found that Alice sometimes missed the first bit sent, so bob adds a "0001" at the start of the message to use the 0s as sacrifice bits
    for char in input_string:
        # Convert the character to its ASCII value and then to an 8-bit binary string
        binary_char = format(ord(char), '08b')
        binary_string += binary_char
    return binary_string

def send(string):
    for char in string:
        send_one_bit(LED, char == "1")

while True:
    text = input("\nEnter text to send: ")
    binary = string_to_binary(text)
    to = time.time()
    send(binary)
    print(f"\nTime taken: {(time.time() - to):.2f} s")
```

Listing 3: bob.py: code for Sender

```python
from pyfirmata import Arduino, util
import time
import numpy as np

board = Arduino('COM14')
it = util.Iterator(board)
it.start()

def binary_to_string(binary_string):  # decoder
    # Initialize empty string to store result
    result_string = ""

    # Iterate through the binary string in 8-character chunks
    for i in range(0, len(binary_string), 8):
        binary_char = binary_string[i:i + 8]
        # int(input, base) converts string of base=base to decimal
        char_code = int(binary_char, 2)
        # Convert the integer to its ASCII character and append it to the result string
        result_string += chr(char_code)
    return result_string

PIN = board.get_pin('a:0:i')  # A0 for input
LED = board.get_pin('d:13:o')  # not used
THRESHOLD = 4/1024  # arduino reads 10bit data, while pyfirmata typecasts to float... so to use 4 as threshold we need to do 4/1024
stop_n = 10_000  # stops after these many consecutive zeros
bits = ""  # initialising
PAUSE = 0  # time idle between two reads
started_reading = False  # flag

# board reads None untill it is ready
while PIN.read() is None: continue
print("Ready")

# Listening
while True:
    val = PIN.read()
    currentValue = val > THRESHOLD  # A2D
    if currentValue and not started_reading:
        started_reading = True
        print("\nStarted Recieving!\n")
    this_bit = "1" if currentValue else "0"
    print(this_bit, end="", flush=True)
    bits += this_bit
    time.sleep(PAUSE)
    if started_reading and bits[-stop_n:] == "0" * stop_n:
        print("\nCompleted Recieving!\n")
        break

# post processing
bits = bits[:-stop_n]  # remove last zeros
readings = bits.split("01")[1:]  # use "0011" to eleminate noisy bit flips of length 1
# readings is a list of strings who start with 1s followed by 0s
lengths = list(map(len, readings))
mean_length = sum(lengths) / len(lengths)
print("Mean length:", mean_length)
# fill last bit with zeros if it is too short
readings[-1] += "0" * (int(mean_length) - len(readings[-1]))
print(*readings, sep="\n")
# output of the above line if message is "a" = "01100001":
# 1111111100000000000000000000000000000000
# 111111111111111111111111111000000000000
# 1111111111111111111111111110000000000000
# 11111111111100000000000000000000000000
# 1111111111110000000000000000000000000000
# 1111111111110000000000000000000000000000
# 11111111111100000000000000000000000000000
# 1111111111111111111111111110000000000000

# demodulation
interprets = ""
for reading in readings:
    ones = reading.count("1")
    if ones > len(reading) / 2: interprets += "1"
    else: interprets += "0"
print("\n")


interprets = interprets.lstrip("0")[1:]

print(interprets)
# interprets = "01100001"
message = binary_to_string(interprets)
print(message)
```

Listing 4: alice.py: code for Reciever

V. EXPERIMENTS WITH ADVERSARIES

In this phase of the experiment, we subjected our optical fiber communication system to various adversarial conditions to assess its robustness and reliability. We tested the system's performance under two key adversarial factors: speed and intensity.

A. Speed Variation

To evaluate the system's resilience to speed variations, we made adjustments to both Alice and Bob's components.

1. Alice's Speed

We initially removed the sleep() function from Alice's code, allowing her to read at a speed limited by hardware. While this increased the number of readings per second significantly, the readings per second exhibited substantial variance. However, our modulation method demonstrated robustness in handling this variance effectively.

2. Bob's Speed

In contrast, we explored the possibility of increasing Bob's speed by decreasing the value of t, the time length of one bit. It's important to note that we could not entirely remove the sleep() function from Bob's code, as it was necessary to keep the LED on for the required time. We pushed the sleep() function to its limit, approximately 10ms, which roughly matches the time taken to send a single bit. Remarkably, Alice was still able to decode the message accurately under these extreme conditions. This experiment confirmed that our modulation method is robust enough to handle significant speed variations.

Results: We succeeded in communicating at a speed of **100 bits per second**.

B. Intensity Variation

We also examined how changes in LED intensity affected the communication system's performance.

1. LED Intensity Control

Initially, we attempted to programmatically control the voltage supplied to the LED using the `pyFirmata` alternative to the `analogWrite()` function in Arduino. However, this approach encountered complications because Arduino GPIO pins cannot provide true analog voltage; instead, they simulate it using PWM (Pulse Width Modulation) output. This simulated PWM signal substantially altered the intended waveform for our communication.

2. Using an External Potentiometer

To address this issue, we adopted a different approach. We used an external potentiometer to regulate the voltage across the LED. The LED initially operated at the full 5V, and by decreasing the voltage, we observed that the communication system encountered problems when the voltage dropped below 3V, even after adjusting the threshold value in Alice's code. Throughout these experiments, we maintained a 25cm long multimode optical fiber. Our observations revealed that the system demonstrated robustness in handling variations in LED intensity.

VI. CONCLUSION & DISCUSSION

- We learned about the various aspects of optical fibers, including their preparation, parameter measurements, and applications through this project. Although we couldn't do most of the experiments which were there in the manual due to lack of or broken equipments, we framed new experiments and learned a lot from them. We learned about the various types of optical fiber and their applications. We learned about the various parameters of optical fiber and how to measure them.As a whole we gained a lot of practical experience for working with optical fibers.
- For the communication system experiment, we developed a robust communication system using optical fibers. We explored a lot about modulation techniques. While doing that we didn't only learn about just signal transfer using optical fibers, we also gained a lot of knowledge on digital communication systems in general. We found immense joy in developing the system and seeing it work. We also learned a lot about the various challenges faced in optical communication systems and how to overcome them.
- Our experiments with adversaries have demonstrated the robustness and reliability of our optical fiber communication system, indicating its capability to function effectively under challenging conditions of speed and intensity variations.
- We successfully communicated at a speed of **100 bits per second** using our communication system. We believe that this speed can be increased further by using better hardware.

VII. POSSIBLE IMPROVEMENTS

- We can use a better LED and photodiode for better performance.
- We can use faster board for faster communication.
- If Alice and Bob's code (code 4 3) are written in Arduino language, we can acheive better speeds with the same hardware. This is because we were limited by the lower possible sleep time in Python. This

sleep time also depends on the Operating System used. We used Windows 10, and the minimum sleep time was about 10ms. In linux generally it is a bit lower. But in Arduino language, the sleep time can be as low as 1ms.

- We did not experiment with optical fibers of different lengths, we can do another experiment where we measure the top potential of the photodiode for different lengths of optical fibers, and plot a graph of top potential vs length of optical fiber. This will give us an idea about the attenuation of light in the fiber.
- We also didn't do any experiments with single mode fibers, because we didn't have equipments to do the experiments which used single mode fibers. Nevertheless, all the experiments can be tried with single mode fibers as well. We don't even know what the results will be. So we can do a lot of experiments with single mode fibers.

[1] Arduino (n.d.). Arduino uno. Microcontroller board.
[2] Contributors, F. P. (Year). Standardfirmata.ino. Accessed: Date.
[3] contributors, W. (n.d.a). Pulse-width modulation. Accessed: 2023-10-08.
[4] contributors, W. (n.d.b). Pulse-width modulation. Accessed: 2023-10-08.
[5] contributors, W. (n.d.c). Pulse-width modulation. Accessed: 2023-10-08.
[6] contributors, W. (n.d.d). Pulse-width modulation. Accessed: 2023-10-08.
[7] de Bruijn, T. and Sarlandie, T. (2009). Pyfirmata.
[8] GeeksforGeeks (2023). Manchester encoding in computer network.