



**Universidade do Minho**  
Escola de Engenharia

**Mestrado em Engenharia de Telecomunicações e Informática**

**Unidade Curricular de Projeto Integrado de Telecomunicações**

Docentes: José Augusto Afonso, Bruno Daniel Mestre Viana Ribeiro, Adriano Jorge Cardoso  
Moreira e Joaquim Melo Henriques Macedo.

# **Projeto Integrador em Telecomunicações e Informática**

## **Relatório Final**

Bárbara Fonseca PG53677

Camila Pinto PG53712

Eduarda Dinis PG53793

Miguel Pereira A94152

**Guimarães, maio de 2024**

# Índice de conteúdos

<b>Índice de conteúdos .....</b>	<b>ii</b>
<b>Lista de figuras .....</b>	<b>iv</b>
<b>Lista de acrónimos e siglas .....</b>	<b>v</b>
<b>1. Introdução .....</b>	<b>1</b>
<b>2. Arquitetura .....</b>	<b>2</b>
<b>3. Conceção do Sistema.....</b>	<b>4</b>
3.1 Implementação dos Circuitos.....	4
3.1.1 Circuito Driver.....	5
3.1.2 Circuito <i>Frontend</i> .....	6
3.2 Implementação da Comunicação entre módulos ESPs32 .....	9
3.2.1 Controlo de fluxo .....	10
3.3 Implementação das Aplicações .....	12
3.3.1 Aplicação Transmissão .....	12
3.3.2 Aplicação Recetor .....	13
3.3.3 Protocolo de Comunicação .....	13
3.3.4 Funcionamento .....	15
<b>4. Testes e Resultados .....</b>	<b>16</b>
4.1 Testes de Distância .....	16
4.2 Teste de tempo de transmissão .....	18
<b>5. Trabalho relacionado.....</b>	<b>20</b>
5.1.1 Transmissão de Dados de PC para PC usando Comunicação por Luz Visível.      20	

---

5.1.2	Comunicação Sem Fio Utilizando a Tecnologia Li-Fi .....	22
<b>6.</b>	<b>Conclusões .....</b>	<b>24</b>
	<b>Referências .....</b>	<b>26</b>

# Lista de figuras

Figura 1. Arquitetura geral do sistema.....	2
Figura 2. <i>Pinout</i> do Sistema.....	4
Figura 3. <i>Pinout</i> do circuito emissor.....	6
Figura 4. <i>Pinout</i> do recetor e do filtro.....	7
Figura 5. <i>Pinout</i> do comparador. ....	8
Figura 6. <i>Pinout</i> do retificador. ....	9
Figura 7. Trama da camada de ligação de dados. ....	9
Figura 8. Diagrama de controlo de fluxo. ....	10
Figura 9. Aplicação de Transmissão.....	12
Figura 10. Aplicação de receção.....	13
Figura 11. Formato da trama da camada aplicacional ....	14
Figura 12. Trama de inicialização.....	14
Figura 13. Cabeçalho da Trama.....	14
Figura 14. Resultado para uma distância de 14 cm. ....	16
Figura 15. Resultado para uma distância de 35 cm. ....	17
Figura 16. Resultado para uma distância de 60 cm. ....	17
Figura 17. Tempo de transmissão para o ficheiro de imagem. ....	18
Figura 18. Tempo de transmissão para um ficheiro de texto.....	19
Figura 19. A Transmitir uma string 'GOOD GRIEF' do Matlab usando VLC. ....	21
Figura 20. A receber uma string 'GOOD GRIEF' no Matlab ....	22
Figura 21. Arquitetura do sistema implementado.....	23

## Lista de acrónimos e siglas

ACK	Acknowledgment
ADC	Analog-to-Digital Converter
AGC	Automatic Gains Control
ESP32	Espressif Systems Platform 32
GPIO	General Purpose Input Output
LDR	Light Dependent Resistor
LED	Light Emitting Diode
Li-Fi	Low Fidelity
NACK	Negative Acknowledgment
PC	Personal Computer
RS	Reed-Solomon
SPI	Serial Peripheral Interface
UART	Universal asynchronous receiver-transmitter
USB	Universal Serial Bus
VLC	Visible Light Communication
Wi-Fi	Wireless Fidelity

# 1.Introdução

No âmbito da unidade curricular de Projeto Integrado em Telecomunicações e Informática, foi proposto aos alunos construir um protótipo de um sistema que utiliza a tecnologia de comunicação sem fio com luz visível ou infravermelho, como base para uma aplicação de *chat*, que permita a conversação, em tempo real, entre dois computadores pessoais (PC - *Personal Computer*), permitindo que além de mensagens, seja possível a transferência de ficheiros.

Este projeto foi dividido em três fases distintas, cada uma com objetivos específicos. Na fase A foram implementados o circuito *driver* e *frontend*, utilizados para converter o sinal elétrico num sinal ótico e o sinal ótico de volta para um sinal elétrico, respetivamente.

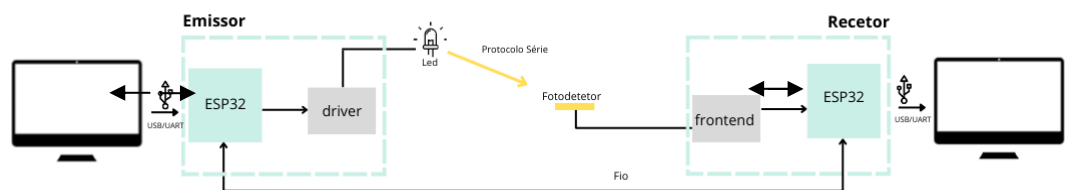
Para a fase B, o objetivo centrou-se na implementação da camada de ligação de dados (camada 2), especificamente no estabelecimento de uma transferência fiável de dados entre as duas placas ESP32 (*Espressif Systems Platform 32*) pelo *link* ótico, tendo em conta as tarefas de controlar a troca de informação na ligação local e controlar o acesso ao meio.

Para a fase C, o objetivo principal foi o desenvolvimento e implementação da camada de aplicação (camada 7), focando-se no protocolo das aplicações desenvolvidas, e proporcionando uma interface gráfica ao utilizador.

Desta forma, o presente relatório aborda uma visão geral e objetiva do sistema final desenvolvido, descrevendo a implementação realizada e os testes executados. Serão apresentados os resultados obtidos da solução final, bem como as principais dificuldades enfrentadas e as soluções encontradas. Além disso, serão discutidas as contribuições e possíveis melhorias do sistema, levando em consideração as aplicações práticas e os desafios encontrados ao longo do processo de desenvolvimento.

## 2.Arquitetura

A arquitetura geral do sistema final implementado é apresentada na Figura 1. Esta é composta pelos computadores, as placas de desenvolvimento, o circuito *driver* (implementado no lado do emissor) e o circuito *frontend* (implementado no lado do recetor).



**Figura 1. Arquitetura geral do sistema.**

Os computadores estabelecem ligação com a placa de desenvolvimento, ESP32, através de uma ligação USB (*Universal Serial Bus*), utilizada para alimentar as placas com energia, permitindo a comunicação entre o computador e a placa de desenvolvimento com recurso ao protocolo UART (*Universal asynchronous receiver-transmitter*). Este protocolo é um assíncrono de transmissão e receção de dados e é implementado em ambos os dispositivos. Como o serviço é *half-duplex*, não é permitido receber mensagens ao mesmo tempo, então a ligação será bidirecional não simultânea.

No estabelecimento da comunicação através do *link* ótico, foi desenvolvida a camada de ligação de dados (camada 2), que inclui mecanismos para controlo de fluxo, através da implementação do protocolo *Stop-and-Wait*.

Para permitir a visualização dos dados, foi desenvolvida a camada de aplicação (camada 7), com a implementação do protocolo de comunicação entre os PC's e a criação de uma interface gráfica. Isto permite ao utilizador escrever uma mensagem na interface, sendo que o texto é escrito num PC e enviado para outro PC, que recebe e apresenta a informação difundida através do canal ótico. Além disso foram desenvolvidas as funcionalidades de transferência de ficheiro de texto e de imagem,

permitindo aos utilizadores continuar a troca de mensagens *chat* em simultâneo com a transferência de um ficheiro.

Os detalhes sobre a implementação dos circuitos *driver* e *frontend*, assim como os protocolos de comunicação utilizados nas diferentes camadas, serão abordados em maior detalhe nas secções seguintes.



## 3. Conceção do Sistema

### 3.1 Implementação dos Circuitos

Nesta secção, abordar-se-á a implementação dos circuitos emissor e recetor, responsáveis pela comunicação ótica. Na Figura 2 é possível observar o *pinout* do sistema final, composto por ambos os circuitos (*driver e frontend*).

Uma característica importante do UART é que, quando está em estado *idle* (inativo), a linha de dados é mantida em nível lógico alto (3,3 V). Desta forma, o LED (*Light Emitting Diode*) estaria constantemente aceso quando a ESP32 não estivesse a transmitir dados, interferindo na capacidade de detetar corretamente os sinais de comunicação. Para resolver esse problema, usamos circuitos integrados NOT (inversores), que invertem o estado lógico do sinal. Isto significa que, quando a linha de dados UART está em nível lógico alto (3,3 V), a saída do inversor estará em nível lógico baixo (0 V), e vice-versa.

No circuito emissor, ao passar o sinal UART da ESP32 por um inversor antes de acionar o LED, garantimos que o LED só acende quando há dados a serem transmitidos. No circuito recetor, o sinal recebido pelo fotodetetor também passa por um inversor antes de ser enviado à ESP32. Isto inverte o sinal, garantindo que os dados recebidos correspondam exatamente aos dados enviados.

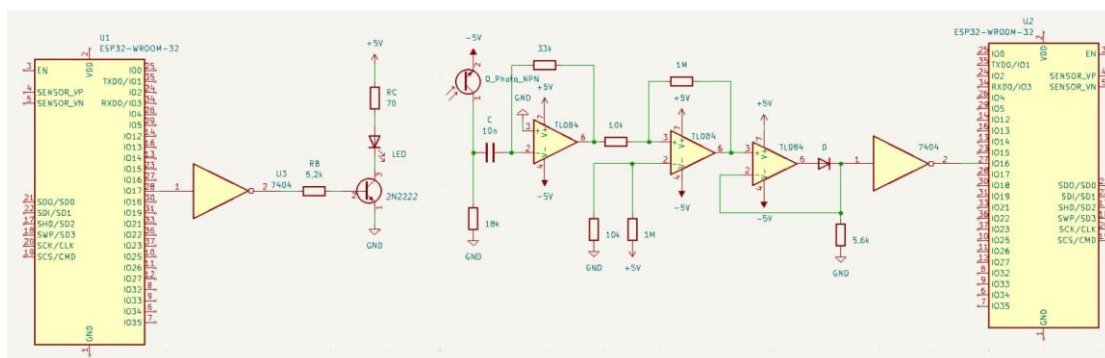


Figura 2. Pinout do Sistema.

### 3.1.1 Circuito Driver

O circuito emissor é composto por um transistor do tipo NPN (*Negative-Positive-Negative*), um LED e resistências. O LED infravermelho é utilizado para gerar um sinal ótico, enquanto a resistência do coletor,  $R_C$ , é utilizada para limitar a corrente que flui para o LED e o transistor, tal como demonstrado na Figura 3. O transistor controla a corrente que atravessa o LED: quando uma corrente suficiente é aplicada à base (por meio do resistor de base), o transistor conduz e permite a passagem de corrente do coletor para o emissor, ativando o LED. Quando a corrente na base é reduzida, o transistor fica em estado de corte, interrompendo o fluxo de corrente e desligando o LED. A resistência de base,  $R_B$ , controla a corrente base do transistor, determinando a corrente necessária para ativar o transistor e, por conseguinte, ligar o LED.

Com base no *datasheet* do componente fotodiodo L-53F3BT [1], a corrente máxima suportada pelo LED é 50 mA, ou seja, a corrente do coletor do transistor será 50 mA.

De acordo com o *datasheet* do transistor 2N2222A, o ganho é de 100 a 300, e considerou-se que  $\beta=100$ , então calcula-se o  $I_B$ .

$$I_B = \frac{I_C}{\beta} = \frac{50 \text{ mA}}{100} = 0,5 \text{ mA}$$

Como no transistor o tem uma queda de tensão do base para o emissor de 0,7 V e como a tensão fornecida pela placa ESP32 é de 3,3 V, temos que:

$$R_B = \frac{V_{in} - V_{BE}}{I_B} = \frac{3,3 - 0,7}{0,5 \text{ m}} = 5,2 \text{ k}\Omega$$

Para calcular a  $R_C$  usa-se o recurso à Lei das Malhas:

$$V_{CC} - V_{LED} - V_C - V_{CE} = 0$$

E temos que,  $V_{CC}=5V$ . De acordo com a *datasheet* do LED, a queda de tensão é de 1,2V e, por fim, com base no *datasheet* do transistor a queda de tensão entre o coletor e o emissor é de 0,3V.

$$5 - 1,2 - V_C - 0,3 = 0 \Rightarrow V_C = 3,5 \text{ V}$$

Utilizando a Lei de Ohm,

$$R_C = \frac{V_C}{I_C} = \frac{3,5}{50 \text{ m}} = 70 \Omega$$

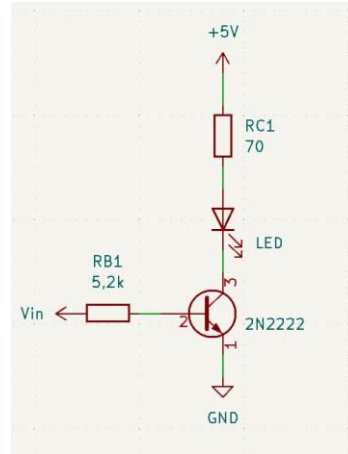


Figura 3. Pinout do circuito emissor.

### 3.1.2 Circuito Frontend

O circuito recetor é constituído por um fotodetetor, um amplificador, um filtro passa-alto, um comparador e um retificador, tal como demonstrado na Figura 4.

Devido ao facto de o sinal vindo do fotodetetor ser muito fraco, optámos por colocar um amplificador pela sua capacidade de amplificar esse sinal. O comparador atua como um limiar de tensão. Assim, se na entrada positiva do amplificador operacional estiver um valor de tensão superior à tensão da entrada negativa (GND no nosso circuito), a saída será Vcc. Caso contrário, a saída será -Vcc, representando os estados high e low de forma clara, possibilitando obter uma onda quadrada mais precisa. O uso do retificador surge da necessidade de a ESP32 não aceitar valores de tensão negativos nos seus pinos I/O. Assim, quando à saída do comparador temos valores negativos, na saída do retificador teremos tensão zero.

Com base no *datasheet* do PL-51P3C, verificou-se que a corrente do fototransistor ( $I_{ON}$ ) está entre 0,1 mA e 0,5 mA. E sabe-se que a tensão que tem de chegar à placa ESP32 é 3,3 V.

$$R = \frac{V_{out}}{I_{ON}} = \frac{3,3}{0,1 \text{ m}} = 33 \text{ k}\Omega$$

## Filtro

O fotodetetor deteta todas as frequências de luz, incluindo a luz ambiente e a luz das lâmpadas. Por esta razão, foi necessário construir um filtro com uma frequência de corte superior a 100 Hz, visto que esta é a frequência da luz ambiente. Para a construção do filtro foi utilizado uma resistência de 18kΩ e um condensador de 10nF.

Frequência de corte:

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2 \times 10 \times 18 \times 10^3 \times 10^{-9}} = 884,19 \text{ Hz}$$

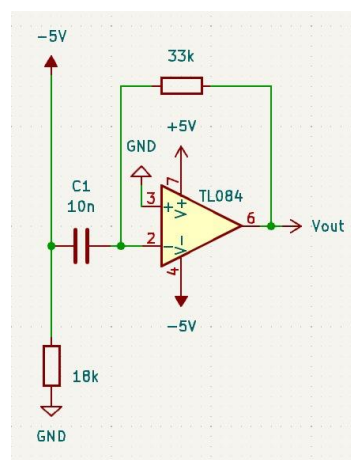


Figura 4. Pinout do recetor e do filtro.

## Comparador

O comparador, tal como exemplificado na Figura 5, é responsável por produzir uma saída que varia entre +5V e -5V. Em ambientes onde a luz ambiente e outras fontes de interferência podem introduzir ruído no sinal, o comparador ajuda a rejeitar variações pequenas e rápidas, mantendo a estabilidade do sinal de saída. A histerese ( $\Delta h$ ), adicionada ao circuito do comparador, ajuda a evitar mudanças erráticas na saída devido ao ruído, proporcionando um funcionamento mais estável.

$$\Delta h = 2 \times V_{sat} \times \frac{R1}{R2}$$

$$R2 = 100 \times R1$$

Sendo  $V_{sat} = 5V$  e  $\Delta h = 100V$ :

$$V_{CEN} = \frac{V_{REF}}{2} = 50mV$$

$$V_{CEN} = V_{REF} \times \frac{R1 + R2}{R2}$$

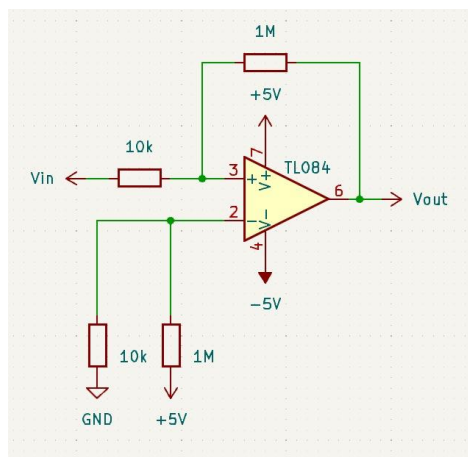
$$V_{REF} = 49mV \cong 50mV = V_{CEN}$$

Para o  $V_{REF}$  ser negativo à entrada do Ampop:

$$V_{REF} = 5 \times \frac{R4}{R4 + R3}$$

$$R3 = 100 \times R4$$

$$R1 = R3 \text{ e } R2 = R4$$



**Figura 5. Pinout do comparador.**

### Retificador de meia onda

O retificador, demonstrado na Figura 6 é responsável por retirar as tensões negativas, por isso é utilizado um retificador de meia onda. Na saída do retificador, é adicionado um inversor alimentado com um VCC de +3,3V e um GND de 0V, pois são as tensões que a placa ESP32 é capaz de detetar.

$$R = 5,2k\Omega \text{ ou } 5,6k\Omega$$

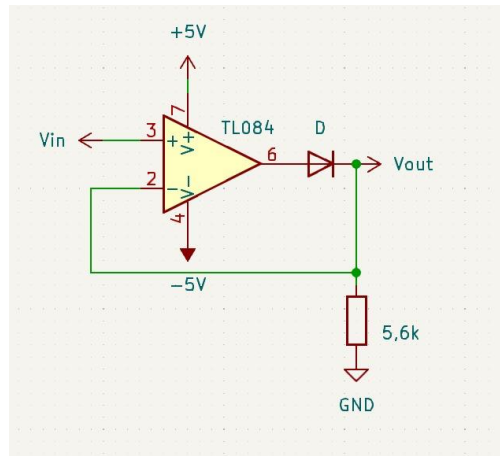


Figura 6. Pinout do retificador.

## 3.2 Implementação da Comunicação entre módulos ESPs32

A comunicação entre os dois módulos ESP32, o emissor e o recetor foi implementada com recurso à linguagem de programação C++ e ao *software* Arduino IDE (*Integrated Development Environment*).

O protocolo de comunicação desenvolvido para esta camada, apresenta a trama com os seguintes campos, o bit de início, o endereço de origem e destino, o tamanho do payload e os dados, tal como demonstrado na Figura 7.

Start bit	Endereço origem	Endereço Destino	Tamanho payload	Payload	FEC
1 byte	1 byte	1 byte	1 byte	1-128 bytes	2-256 bytes

Figura 7. Trama da camada de ligação de dados.

O *start bit* permite que haja sincronismo entre o emissor e recetor, e que o recetor identifique o início da trama, sendo que este é representado pela sequência 1110 (0x0E). Para identificar a origem e destino da informação, utilizamos o endereço de origem (0x01) e o endereço de destino (0x08). O número de *bytes* de correção de erros varia de acordo com o tamanho do *payload* da trama. A técnica utilizada acrescenta redundância, deste modo o fator de multiplicação é três, triplicando o tamanho do *payload* da trama. Quando o computador envia para a placa de desenvolvimento os dados, através da porta série, esta envia os dados encapsulados nesta trama, para a placa recetora, *byte a byte*.

### 3.2.1 Controlo de fluxo

Para implementar o controlo de fluxo, utilizamos o protocolo *Stop-and-Wait*. Desta forma, o emissor envia um pacote ao recetor e aguarda pela confirmação. Se não receber essa confirmação dentro de um período de *timeout* de 5 segundos, o pacote é reenviado. Se a confirmação for recebida dentro desse intervalo, um ACK (*Acknowledgment*) (0x06) é enviado, indicando sucesso, e o próximo pacote é enviado em seguida. Se um NACK (*Negative Acknowledgment*) (0x07) for recebido, a trama é reenviada imediatamente. É possível visualizar este processo no diagrama de sequência apresentado na Figura 8.

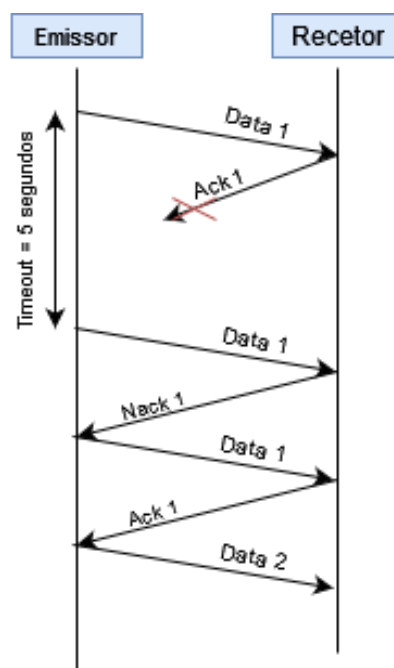


Figura 8. Diagrama de controlo de fluxo.

Foi implementado um método de deteção de erros na camada de ligação de dados para garantir a integridade dos dados transmitidos entre o emissor e o recetor. Utilizamos a biblioteca *zlib* do Python, que inclui uma variedade de funções para compressão de dados e cálculo do *CRC*. Especificamente, utilizamos o método `crc32()` desta biblioteca para calcular um valor de *CRC* para cada trama de dados antes de ser enviada pelo emissor.

Este método de detecção de erros baseado em *CRC* proporciona uma camada adicional de segurança para garantir que os dados transmitidos sejam recebidos de forma confiável e sem corrupção. Ao combinar este mecanismo de detecção de erros com o protocolo Stop-and-Wait para controlo de fluxo, podemos assegurar uma comunicação robusta e eficiente entre o emissor e o recetor.

De modo geral, a camada de ligação de dados, recebe a trama proveniente da camada 7 e encapsula-a no seu *payload*. Assim, os serviços implementados pela camada 2 são disponibilizados à camada superior através de um conjunto de funções, designadas APIs (*Application Programming Interfaces*), sendo estas a função *send\_data* e *receive\_data*, que permitem também a comunicação bidirecional.

A função *send\_data* é responsável pelo envio de dados do emissor para o recetor utilizando uma conexão série. Esta define cada campo da trama, calcula o *CRC* do *payload* e adiciona-o à trama. Posteriormente a trama é enviada e aguarda-se pela receção de um ACK ou NACK. Se um ACK (b'\x06') for recebido dentro do tempo limite, a função considera que a trama foi recebida com sucesso e pode enviar o próximo pacote. Se um NACK for recebido, a trama é reenviada imediatamente. Caso o tempo limite expire sem receber uma resposta, a trama é reenviada. Este processo garante a integridade dos dados transmitidos, pois qualquer falha na transmissão resulta no reenvio da trama.

A função *receive\_data* é responsável por receber dados do emissor e enviar uma confirmação de receção (ACK) ou uma notificação de erro (NACK). Lê o *byte* inicial (*start\_byte*) e verifica se corresponde ao *byte* esperado (b'\x0E'), se for válido a função lê os *bytes* seguintes que contêm o endereço de origem, endereço de destino, o tamanho do *payload*, e o *payload* propriamente dito e o *CRC* recebido. Após calcular o *CRC* dos dados recebidos, a função compara com o *CRC* recebido. Se os valores do *CRC* corresponderem, um ACK é enviado ao emissor, confirmando a receção bem-sucedida dos dados. Caso contrário, um NACK é enviado, indicando que a trama está corrompida e precisa ser reenviada. Este método assegura que apenas os dados corretos são aceites e processados.



### 3.3 Implementação das Aplicações

Na camada aplicacional, foram desenvolvidas duas interfaces gráficas, uma correspondente ao emissor e outra ao recetor. Foram implementadas recorrendo a linguagem de programação Python com recurso à biblioteca *tkinter* e *customtkinter*.

#### 3.3.1 Aplicação Transmissão

A aplicação de transmissão é responsável por permitir o utilizador gerar os dados que serão transmitidos pelo sistema de modo simples através do computador pessoal, sendo estes dados processados e posteriormente escritos na *serial port* para a placa ESP32 conectada ao circuito *driver*.

A interface de transmissão permite ao utilizador seleccionar o tipo de dados a ser enviado, ficheiro ou mensagem de texto. A interface possui campos para entrada de texto, um botão para o envio de ficheiros, e uma caixa de texto para exibir as mensagens enviadas e recebidas. A Figura 9 exemplifica a interface gráfica de transmissão.

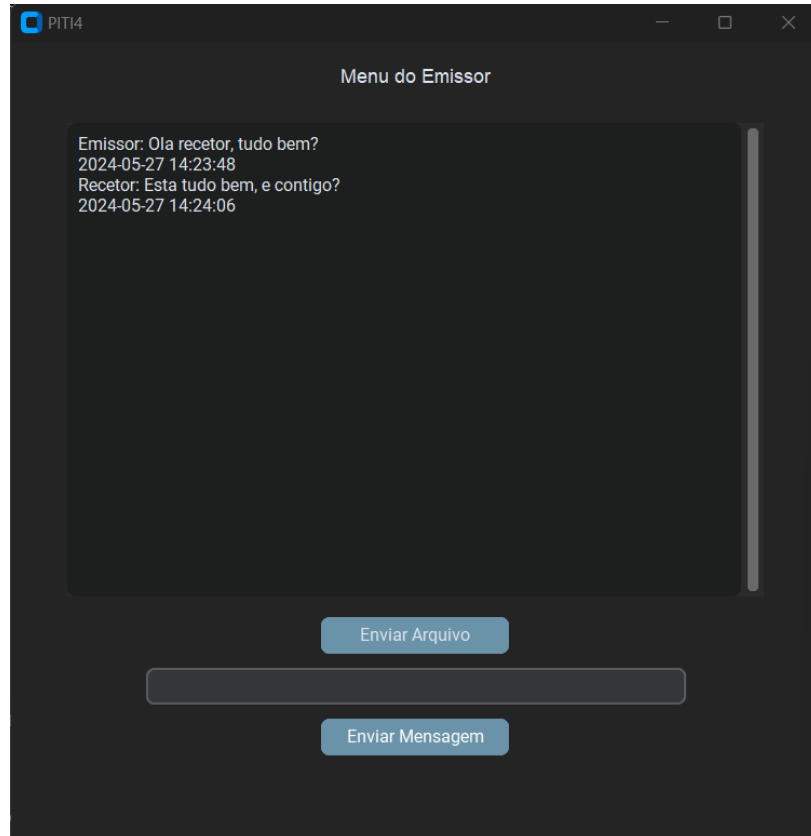


Figura 9. Aplicação de Transmissão.

### 3.3.2 Aplicação Recetor

A aplicação no lado do recetor é responsável por receber os dados enviados pelo emissor e exibi-los para o utilizador. A interface gráfica permite a visualização das mensagens recebidas, bem como a reconstrução e exibição dos ficheiros de imagem.

A interface de receção possui um campo para entrada de texto e uma caixa de texto para a exibição das mensagens recebidas e enviadas. Quando uma imagem é recebida e reconstruída é exibida numa nova janela. A Figura 10 exemplifica a interface gráfica da aplicação de receção.

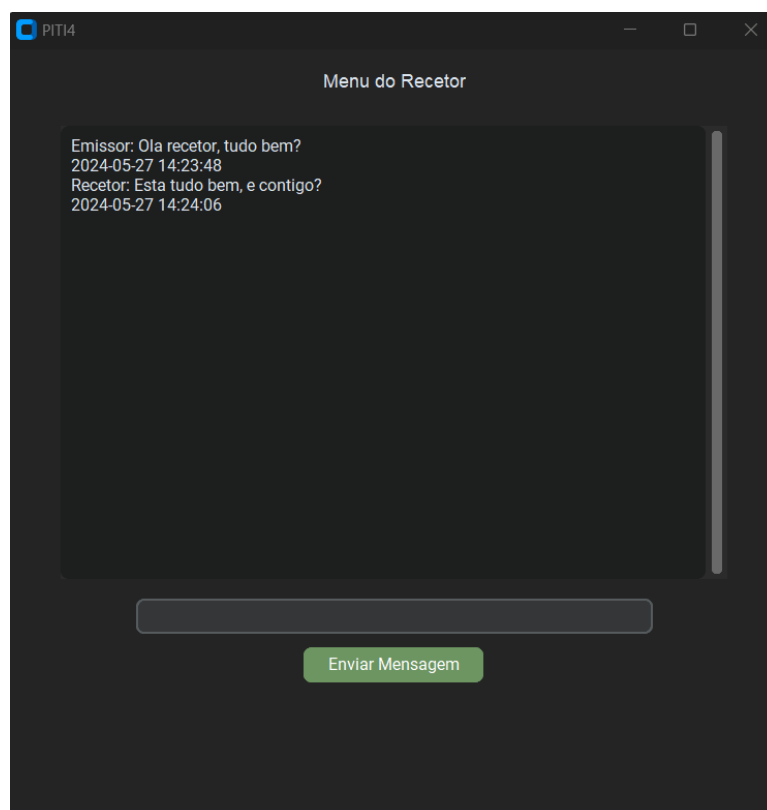


Figura 10. Aplicação de receção.

### 3.3.3 Protocolo de Comunicação

Para a comunicação e interpretação dos dados entre o remetente e o destinatário será definido um conjunto de mensagens. A Figura 11, exemplifica o formato da trama implementada para esse protocolo de comunicação.

Header (3 bytes)	Payload (1-128 bytes)
------------------	-----------------------

**Figura 11. Formato da trama da camada aplicacional**

Dado os diferentes tipos de dados que serão enviados, a criação da trama de inicialização tem como objetivo iniciar a comunicação, fornecendo ao recetor algumas informações sobre o que será enviado, sendo possível usar essas informações para reconstruir os ficheiros. Esta é apresentada na Figura 12

Tipo de dados (1 byte)	Total de pacotes (1 byte)	Tamanho do nome de ficheiro (1 byte)	Nome do ficheiro (1 byte)
------------------------	---------------------------	--------------------------------------	---------------------------

**Figura 12. Trama de inicialização.**

O campo *Tipo de dados* assume o valor 0, já que se trata da trama de inicialização. O campo *Total de pacotes* será utilizado para controlar o número de tramas enviadas (número de sequência) para realizar a reconstrução dos ficheiros de texto e imagem. O tamanho define o tamanho do nome do ficheiro. O campo *Nome ficheiro*, define o nome do ficheiro que será armazenado em disco.

Na Figura 13 encontra-se especificado o conteúdo do *header* da trama da camada 7.

Tipo de dados (1 byte)	Número de Sequência (1 byte)	Tamanho payload (1 byte)
------------------------	------------------------------	--------------------------

**Figura 13. Cabeçalho da Trama**

O campo *Tipo de dados* pode assumir diferentes valores para representar os diferentes tipos de mensagens:

- 1 – Ficheiro de texto/imagem;
- 2 – Mensagem de texto simples.

O campo *Número de sequência* representa o número de sequência das tramas. O campo *Tamanho payload* assume o tamanho da informação enviada.

### 3.3.4 Funcionamento

No modo emissor, é solicitado que o utilizador introduza se pretende enviar um ficheiro ou uma mensagem (*chat*), e escreva o conteúdo correspondente. Em seguida, a trama da camada aplicacional é construída. Quando é para o envio de um ficheiro, o mesmo é dividido em pacotes de até 128 *bytes*, e a trama construída é precedida pela trama de inicialização.

Para o envio de dados, é inicializada uma *thread* dedicada, enquanto para o chat são inicializadas duas *threads*: uma para o envio e outra para a receção de dados. Durante o processo de envio, o protocolo *Stop-and-Wait* é utilizado para garantir a entrega correta das tramas, enviando um ACK (0x06) em caso de sucesso ou um NACK (0x07) em caso de erro, o que provoca a retransmissão da trama.

No modo recetor, é inicializada uma *thread* para receção de dados. Ao receber uma mensagem, o recetor pode responder à mesma. Quando um ficheiro é recebido, a trama de inicialização é utilizada para extrair as informações necessárias para a reconstrução do ficheiro de texto ou imagem.

O campo Nome do ficheiro é utilizado para determinar o nome do ficheiro a ser armazenado em disco, enquanto o campo Total de pacotes é comparado com o número de sequência de cada trama para verificar se todas as tramas foram recebidas. Uma vez recebidas todas as tramas, o ficheiro é reconstruído. No caso de imagens, a reconstrução é seguida da exibição da imagem numa nova janela.

A funcionalidade de *chat* permite a comunicação em tempo real entre o emissor e o recetor. No modo de *chat*, ambas as interfaces (emissor e recetor) possuem uma *ScrolledText* onde as mensagens enviadas e recebidas são exibidas. Cada mensagem é precedida por uma identificação do autor.

Durante a comunicação, o emissor escreve uma mensagem e envia-a. Esta mensagem é encapsulada numa trama e transmitida ao recetor. A interface do recetor então exibe a mensagem na sua *ScrolledText*. Simultaneamente, o recetor pode responder às mensagens, seguindo o mesmo processo de encapsulamento e transmissão.

## 4. Testes e Resultados

Nesta secção serão abordados todos os testes efetuados, bem como os resultados obtidos. Foram efetuados testes que avaliam a distância máxima que se podia atingir sem que houvesse grande distorção do sinal e testes para analisar o tempo de transmissão de ficheiros.

### 4.1 Testes de Distância

Para este tipo de testes, foram capturadas as formas de onda, com o auxílio do osciloscópio, à entrada do circuito driver e na saída do circuito *frontend*, a distâncias de 14 cm, 35 cm e 60cm.

As imagens que se seguem são fotos tiradas ao osciloscópio, em que o *channel 1* (onda de cima) corresponde ao sinal do emissor e o *channel 2* ao sinal recetor (onda de baixo).



Figura 14. Resultado para uma distância de 14 cm.

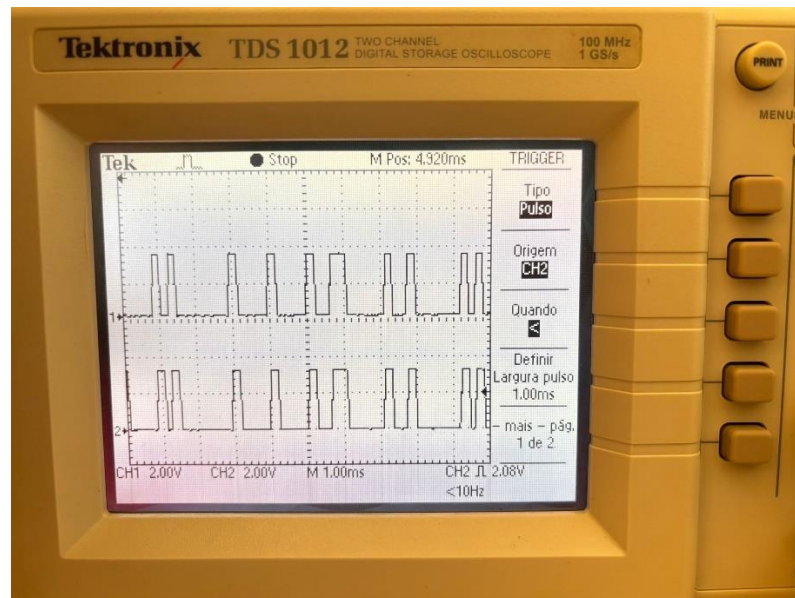


Figura 15. Resultado para uma distância de 35 cm.

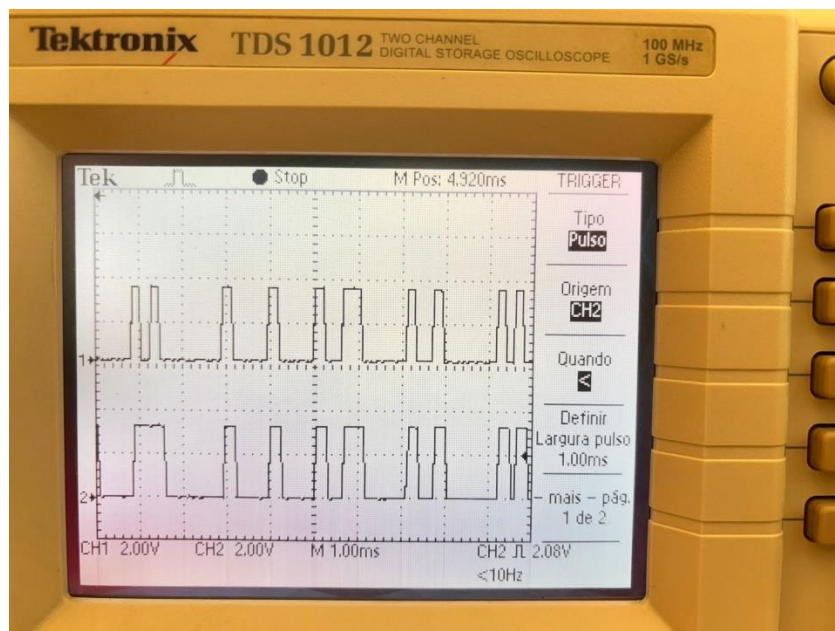


Figura 16. Resultado para uma distância de 60 cm.

Comparando as imagens, conseguimos ver que até esta distância não ocorre distorção de sinal, porém ao aumentar para 1 m o mesmo fica distorcido, pelo que 60 cm é a nossa distância máxima. Isto podia ser otimizado modificando o *threshold* do comparador.

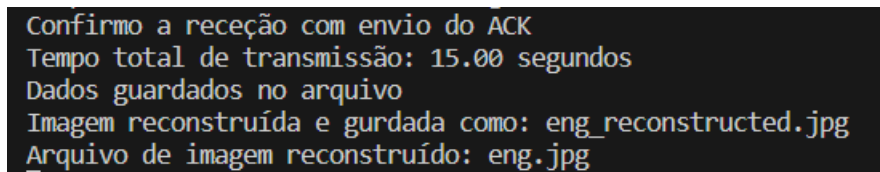
## 4.2 Teste de tempo de transmissão

No caso de estudo testado, utilizámos uma imagem e um ficheiro de texto, em formato .txt. O tempo de transmissão no recetor foi estimado, começando a contar a partir da leitura do *start\_byte*, que indica o início de uma trama, e terminando após a leitura da trama e o envio do ACK. O tempo total é a soma dos tempos de transmissão de cada trama. A taxa de transmissão é 9600 bps.

O cálculo teórico do tempo de envio é dado pela expressão:

$$T = \frac{n^{\circ} \text{ total de bits}}{\text{taxa de transmissão}} (s)$$

Tendo em conta uma imagem (em formato .jpg), com 4 632 bytes, o tempo de transmissão obtido é 15 segundos, tal como é comprovado na Figura 17.



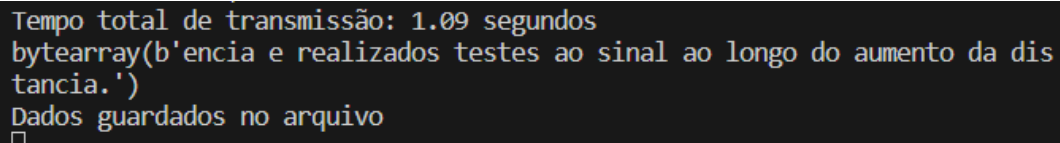
```
Confirmo a receção com envio do ACK
Tempo total de transmissão: 15.00 segundos
Dados guardados no arquivo
Imagem reconstruída e gurdada como: eng_reconstructed.jpg
Arquivo de imagem reconstruído: eng.jpg
```

Figura 17. Tempo de transmissão para o ficheiro de imagem.

Utilizando este exemplo, sabemos que o número total de *bytes* da imagem é 4632, como cada trama contém no máximo 128 *bytes* de informação útil, são geradas aproximadamente 36 tramas  $\frac{4632}{128} \approx 36$ . A cada uma destas tramas, a nível aplicacional, são acrescentados 3 *bytes* (tipo de dados, número de sequência, tamanho *payload*), 128 *bytes* de informação que resulta num total de 131 *bytes*. A trama de início acrescenta e 4 *bytes* (tipos de dados, total de pacotes, tamanho do nome do ficheiro, nome do ficheiro). No nível dos módulos ESP32 também é introduzido *overhead*, sendo acrescentados 4 *bytes* (*start bit*, endereço origem, endereço destino e tamanho *payload*) e 256 *bytes* para o FEC, resultando num total de 260 *bytes*. Deste modo, para 36 tramas,

será enviada uma com 395 *bytes*, correspondente à trama de *start*, e as restantes enviam 391 *bytes*. O número total de *bytes* a enviar para 36 tramas é 14080 *bytes*, passando para *bits* temos 112640 *bits*. Aplicando a fórmula, o tempo teórico resulta em 11,73 segundos.

Tomando o caso do ficheiro de texto, com 324 *bytes*, obtivemos 1,03 segundos, tal como demonstrado na Figura 18. O tempo teórico obtido é 0,122604 segundos.



```
Tempo total de transmissão: 1.09 segundos
bytearray(b'encia e realizados testes ao sinal ao longo do aumento da dis
tancia.')
Dados guardados no arquivo
□
```

**Figura 18.** Tempo de transmissão para um ficheiro de texto.



## 5. Trabalho relacionado

### 5.1.1 Transmissão de Dados de PC para PC usando Comunicação por Luz Visível.

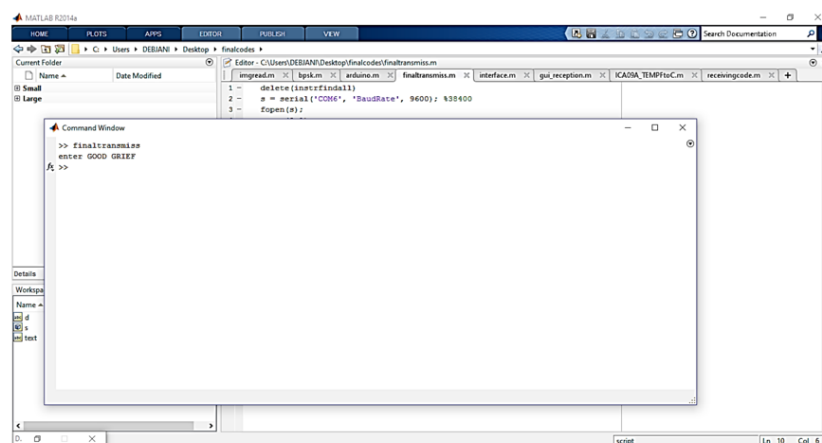
O projeto descrito em [2] construiu um sistema de VLC (*Visible Light Communication*) sem fios, capaz de transmitir dados de texto entre dois computadores usando luz visível. Um Díodo Emissor de Luz (LED) é utilizado como transmissor, o ar como meio de transmissão, e um Resistor Dependente de Luz (LDR - *Light Dependent Resistor*) como componente recetor. Os dados de texto são transmitidos como sequências de 1s e 0s enquanto o LED pisca rapidamente, a uma taxa indetectável para o olho humano. Estes dados binários são recebidos pelo LDR, convertidos para o formato apropriado e exibidos no ecrã de saída.

O modelo projetado é composto por dois computadores, um no extremo transmissor e outro no extremo recetor, dois microcontroladores (Arduino UNO) em ambos os extremos, um LED branco no extremo transmissor e um LDR no extremo recetor. No extremo transmissor, o computador está ligado ao Arduino UNO, que está conectado ao LED branco. O Arduino faz com que o LED pisque rapidamente a uma velocidade indetectável para o olho humano, transmitindo dados simultaneamente. Estes rápidos piscar de LED criam pulsos com diferentes durações. No extremo recetor, o LDR está ligado ao Arduino, que está conectado ao computador. O LDR deteta o rápido piscar do LED. Usando uma função especial no Arduino IDE, a duração dos pulsos, criados pelo piscar, é calculada. Dependendo das durações, pulsos ALTO e BAIXO são distinguidos e uma sequência de bits binários é gerada. Esta sequência de bits é enviada para o Matlab para processamento adicional e geração de saída. O Arduino IDE enfrenta muitos problemas ao lidar com grandes quantidades de dados, que o Matlab pode lidar

facilmente. Além disso, certas funções predefinidas do Matlab facilitam o processo de geração de saída.

O sistema concebido conseguiu corretamente demonstrar a comunicação serial entre dois PCs. O objetivo era estabelecer um sistema de comunicação por luz visível que transmitisse e recebesse texto. Na fase inicial, foi utilizado o microcontrolador Arduino UNO e o Arduino IDE para transmitir e receber um único caractere. Mais tarde, um processo passo a passo foi utilizado para a transmissão de texto usando o Matlab. No final, até 18 caracteres foram transmitidos e recebidos com sucesso. A distância entre o LED e o LDR foi definida em 6 centímetros.

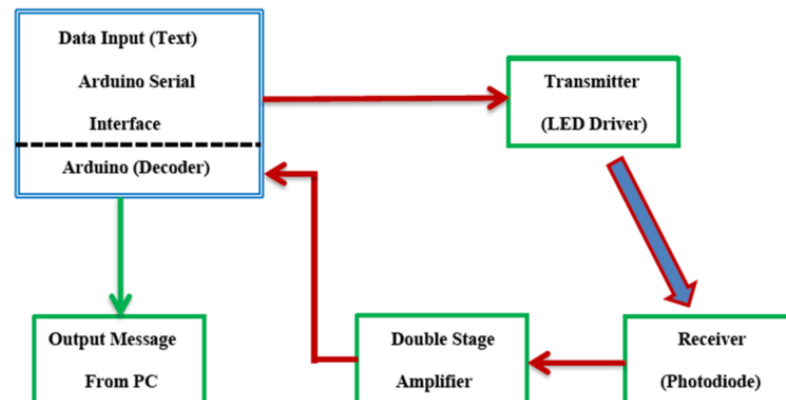
Uma representação visual de representações da interface Matlab, demonstrando a transmissão e receção de dados, está incluída nas figuras seguintes.



**Figura 19. A Transmitir uma string 'GOOD GRIEF' do Matlab usando VLC.**

A Figura 19 mostra a consola do Matlab á saída do transmissor. Após executar o código, a consola solicita uma *string* como entrada do usuário, que neste caso é 'GOOD GRIEF'.





**Figura 21. Arquitetura do sistema implementado.**

Observando a imagem identificamos os blocos do emissor e recetor, o emissor é responsável por converter dados digitais em luz visível, este recebe um sinal do computador via USB e, em seguida, dos pinos de Entrada e Saída de Propósito Geral (GPIO - *General Purpose Input Output*) no Arduino. Por outro lado, o recetor é responsável por converter a luz recebida em corrente usando um fotodíodo, seguido de um amplificador operacional para filtrar e amplificar o sinal.

Na implementação do circuito recetor, como a distância entre este e o recetor pode variar e de modo a evitar sinal muito pequeno ou muito alto, um Controlador Automático de Ganho (AGC - *Automatic Gains Control*) podia ser projetado; no entanto, optaram por uma resistência variável.

Além do circuito emissor e recetor, foi desenvolvida uma aplicação em Java, com o intuito de decodificar a mensagem enviada pelo emissor e a exibir na janela de saída, permitindo ao utilizador visualizar as mensagens.

Em conclusão, o sistema demonstrou um desempenho satisfatório na transmissão de mensagens, a implementação foi económica e serviu o propósito de demonstrar o funcionamento de um protótipo Li-Fi e apoiar as vantagens de Li-Fi sobre o Wi-Fi. No entanto, o mesmo apresenta limitações, não suportando o acesso multiutilizador, a velocidade alcançada é de apenas 11, 520 bps e não é bidirecional, servindo apenas para o propósito e transmissão.

## 6. Autoavaliação/Reflexão

**Camila Pinto:** Na fase A, auxiliei a minha colega na montagem dos circuitos eletrônicos. Na fase B implementei o protocolo de comunicação da camada 2, juntamente com a comunicação entre as placas ESP32 e o controlo de fluxo. Na fase C implementei o protocolo de comunicação da camada 7 e a correção do link ótico. Participei na elaboração de todos os relatórios apresentados.

**Eduarda Dinis:** Na fase A, implementei a montagem dos circuitos eletrônicos. Na fase B implementei o protocolo de comunicação da camada 2, juntamente com a comunicação entre as placas ESP32 e o controlo de fluxo. Na fase C corriji a comunicação do link ótico. Participei na elaboração de todos os relatórios apresentados.

**Bárbara Fonseca:** Participei de todos os relatórios apresentados.

**Miguel Pereira:** Na fase A, ajudei as minhas colegas na montagem dos circuitos. Na fase B implementei a correção de erros. Na fase C implementei a interface gráfica e a funcionalidade de chat e melhorei o controlo de fluxo. Participei na elaboração dos relatórios apresentados, exceto no de especificação da fase b.

## 7. Conclusões

O objetivo do projeto consistiu na construção um protótipo de um sistema que utiliza a tecnologia de comunicação sem fio com luz visível ou infravermelho, como base para uma aplicação de *chat*, que permite a conversação, em tempo real, entre dois computadores pessoais.

No geral, considerámos que implementámos o projeto de forma bem-sucedida. Para tal utilizamos o nosso conhecimento em eletrónica de forma a desenvolver o circuito *driver* e *frontend* capaz de comunicação, demonstrando a capacidade dos circuitos óticos de modular a intensidade luminosa de um LED infravermelho para transmitir informações e de reconstruir o sinal do lado do recetor. Além disso, empregamos o nosso conhecimento em informática e redes para desenvolver o protocolo de comunicação para a transmissão pelo *link* ótico, bem como para desenvolver a camada aplicacional que proporcionou ao utilizador uma interface para envio e receção de dados.

Apesar do sucesso na implementação do projeto, é importante destacar algumas limitações e desafios encontrados ao longo do caminho. Infelizmente, não conseguimos otimizar os nossos circuitos para funcionar a uma distância superior, algo que poderia ser alcançado com uma melhor gestão de tempo ou trabalho futuro. Adicionalmente, embora inicialmente tenhamos implementado um mecanismo de CRC, as suas limitações tornaram-se evidentes ao utilizar a ligação infravermelha. Por conseguinte, optámos apenas pela deteção de erros em vez da correção. Reconhecemos que teria sido mais apropriado utilizar códigos como *Hamming* ou *Reed-Solomon* para uma deteção e correção mais eficazes.

## Referências

- [1] K. Corporation. L-53F3BT datasheet. [Online]. Available: <https://pdf1.alldatasheet.com/datasheetpdf/download/89553/KINGBRIGHT/L-53F3BT.html>. [Accessed: 08-Feb-2024].
- [2] S. Das, A. Chakraborty, D. Chakraborty and S. Moshat, "PC to PC data transmission using visible light communication," *2017 International Conference on Computer Communication and Informatics (ICCCI)*, Coimbatore, India, 2017, pp. 1-5.
- [3] E. Ifada, N. T. Surajudeen-Bakinde, N. Faruk, A. Abubakar, O. O. Mohammed and A. O. Otuoze, "Implementation of a Data Transmission System using Li-Fi Technology," *2019 2nd International Conference of the IEEE Nigeria Computer Chapter (NigeriaComputConf)*, Zaria, Nigeria, 2019, pp. 1-7.