**FACOLTÀ DI INGEGNERIA**

**UNIVERSITÀ POLITECNICA DELLE MARCHE**


***Trabajo Fin de Grado***

# Study and development of a Submarine Optical Communication: TCP Protocol

**Para acceder al Titulo de**

***Graduado en***
***Ingeniería de Tecnologías de Telecomunicación***

**Maria Jesús Pérez Saiz**

**Julio 2018**

## RESUMEN

Este trabajo pretende analizar si se pueden enviar mensajes mediante un medio acuático entre una base en la superficie y un submarino. Uno de los problemas importantes al que se debe de buscar una solución es que las ondas electromagnéticas se atenúan de forma considerable a través del agua. Se pretende comprobar si es posible transmitir datos mediante ondas luminosas procedentes de un diodo LED emisor y un fotodiodo receptor. Lo primero que se realizará es crear un programa basado en el protocolo TCP/IP que nos permita la transmisión de datos y que se adaptará a unas placas específicas que son LX200V20 para en ellas poder conectar el LED y el fotodiodo. Para finalizar se hará un estudio sobre cómo se transmiten los datos mediante el canal para poder elegir los componentes adecuados.

## *ABSTRACT*

*This study aims to analyze if messages can be sent through an aquatic environment between a base on the surface and a submarine. One of the problems that must be solved is that electromagnetic waves are dimmed when travelling though water. The goal is to verify if transmitting data using luminous waves from an LED diode and a photodiode receptor is possible. The first step consists in creating a program based on the TCP/IP protocol that allows us to transmit data and later adapt the components to specific LX200V20 boards. Finally, a study will be taken under way to understand how the data transmits though the channel to choose the correct components.*

# ÍNDICE

# ÍNDICE DE ILUSTRACIONES

## ÍNDICE DE GRÁFICOS



## ÍNDICE DE TABLAS



## ÍNDICE DE ESQUEMAS

# INTRODUCCIÓN.

En el entorno de transmisión inalámbrica tenemos una excepción debido a la fuerte atenuación de las ondas de radio en el agua, ya que generalmente se basa en ondas electromagnéticas en la parte de radiofrecuencia del espectro, pero en este caso no se pueden aprovechar. Por ello, las comunicaciones submarinas se basan en la modulación y la transmisión de ondas acústicas. Los módems acústicos disponibles poseen un ancho de banda disponible de unos pocos KHz y, por lo tanto, su velocidad binaria.

Los vehículos submarinos se basan principalmente en grabar imágenes de alta resolución, videos, sonar y otros datos, y de transmitirlos a una unidad central. Sin embargo, para este tipo de datos se requiere una tasa de bits alta y una transmisión de latencia baja.

Al combinar estos dos aspectos, alta transmisión inalámbrica y alta velocidad, las comunicaciones inalámbricas ópticas subacuáticas (UOWC) representan una alternativa válida. Gracias a los extraordinarios desarrollos en diodos emisores de luz (LED) con fines de iluminación ha sido posible generar dispositivos compactos comúnmente disponibles de bajo coste, alto brillo y ancho de banda de modulación relevante.

Gracias a que en el agua de mar hay una ventana de baja atenuación en la región visible, UOWC puede ofrecer tasas de bits mucho más altas en el orden de Mbits/s, pero a distancias más cortas si lo comparamos con los módems acústicos.

En este documento presentamos el sistema que se ha realizado en el marco del proyecto EU-FP7 SUNRISE, y es correcto para la integración en la red de proyectos submarinos. Los resultados muestran que los módems UOWC de OptoCOMM proporcionan una transmisión bidireccional estándar IEEE 10Base-T confiable (Ethernet 10 Mbit/s).

La sección óptica de los módems está compuesta por dos partes: la comunicación constituida por el transmisor óptico (TX) y el receptor (RX), que ayuda a estimar el nivel de potencia de luz en el RX. El TX está formado por un conjunto de LEDs de 7chips con una longitud de onda de emisión de 470 nm y un ancho de banda óptico de 20 nm. Por otro lado, el RX incluye un solo módulo de Avalanche Photo-Diode (APD) con un área activa de 100 mm2, un ancho de banda de 11 MHz y un amplificador de transimpedancia integrado (TIA).

La electrónica del módem incluye sensores auxiliares, alimentación y una placa de microcontrolador para la instalación de firmware/software y el control y gestión general del módem.

# CAPÍTULO 1. PROTOCOLO TCP/IP

## 1.1. INTRODUCCIÓN AL PROTOCOLO TCP/IP.

El software desarrollado en la placa tiene la función de gestionar los comandos recibidos de un usuario y enviar archivos a través de la capa de transmisión óptica. El usuario se comunica con el módem a través del protocolo TCP/IP Ethernet para cambiar las configuraciones del módem, recuperar información sobre el estado del módem, enviar y recibir datos a través de comunicación óptica.

Un protocolo se basa en un conjunto de normas y procedimientos mediante el cual un servidor y un cliente comparten información entre sí. Los protocolos TCP (Transmission Control Protocol) e IP (Internet Protocol) fueron creados a principios de 1980 y fueron adoptados por ARPANET (Advanced Research Projects Agency Network) en 1983. Ambos protocolos junto con el UDP (User Datagram Protocol) son los más importantes y más usados, los cuales se diferencian en la metodología de reconocer los paquetes de datos por parte del destino y hacer saber al emisor su llegada. En el caso de Internet, su éxito se debe a sus capacidades para soportar servicios de transferencia de datos de extremo a extremo confiables para una gran cantidad de aplicaciones que se ejecutan en un conjunto de sistemas finales.

A mediados de los años ochenta fue creado el protocolo TCP/IP con el objetivo de contar con un lenguaje común a todos los ordenadores que estén conectados a Internet sin importar las marcas y la tecnología diferente que cada uno posee, esto fue posible gracias a la unión de ARPANET, CSNET y MILNET.

La forma de transmitir los archivos en el protocolo TCP/IP se basa en dividir la información en paquetes de pequeño tamaño llamados “datagramas” o grupos de datos que son enviados como si se tratara de un mensaje. La función de enviar los paquetes de información de un sitio a otro se encarga el protocolo IP, mientras que el TCP se encarga de dividirlos en paquetes, ordenarlos en secuencia y añadir información para controlar si se produce un error.

Por el otro extremo, el protocolo TCP se ocupa de recibir los datagramas, revisar si existen errores y ordenarlos en el orden que fueron enviados. Cuando los archivos enviados son de gran tamaño se requieren sólo unos segundos, aunque los paquetes de información tengan que pasar de máquina a máquina hasta llegar a la computadora que los solicitó gracias a su dirección IP.

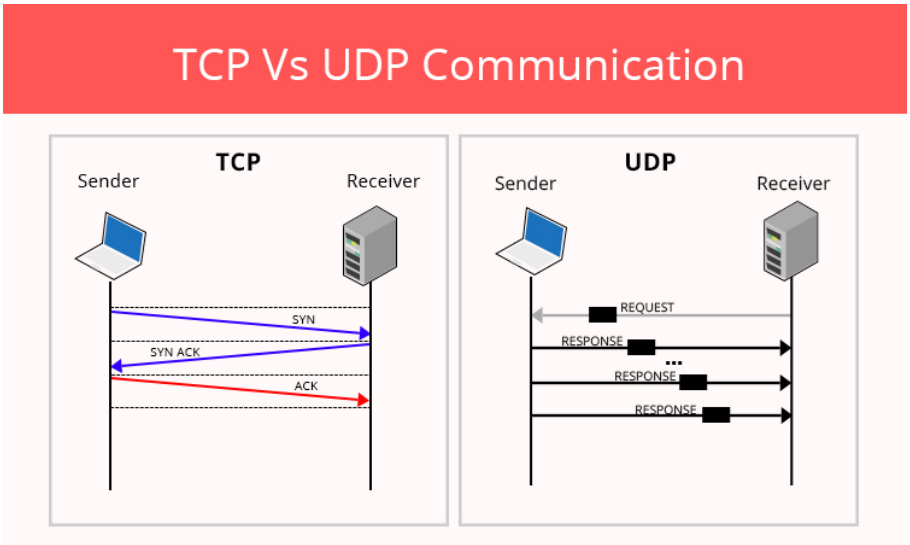
Los paquetes de información para llegar a su destino encuentran guías a lo largo del camino, llamadas ruteadores, que indican los tramos que deben de recorrer.

Los ruteadores son computadoras de gran tamaño que se ocupan de manejar el tráfico de información en la red hasta llegar al usuario que la solicitó.

El cliente necesita conectarse al módem óptico, servidor, en dos puertos diferentes (identificados como 1 y 2). En el puerto 1, el usuario envía los comandos y recibe la respuesta correspondiente mientras que en el puerto 2, el usuario recibe los datos de una transmisión óptica. En cuanto al protocolo de comunicación óptica, se informa al usuario sobre el resultado de la comunicación al recibir paquetes ACK y REC en el puerto 2.

A diferencia del TCP el UDP no está orientado a la conexión al no establecer una conexión segura y fiable, por otro lado, no establece una conexión antes de comenzar a transmitir datos. En la ilustración 1.1 se puede ver un esquema comparativo de estos dos protocolos.

**Ilustración 1.1. Comparación entre el protocolo TCP y UDP.**




*Fuente: Internet.*

## 1.2. CÓDIGO TCP/IP PARA LA TRANSMISIÓN DE DATOS.

El código se ha implementado en lenguaje C con el programa 'Visual Studio 2017' para permitir la transmisión de datos entre dos computadoras a través de un cable Ethernet siguiendo el protocolo TCP/IP.

Es de notable importancia saber que el TCP hace uso de la conexión y no del puerto del protocolo, lo que permite a un mismo puerto establecer varias

conexiones, sabiendo que una conexión completada consta de dos extremos (dos puertos cada uno en una computadora distinta a través de Internet).

En este caso, se utilizarán las librerías llamadas sockets que nos proporciona el fabricante al principio del programa tanto del servidor como del usuario, mostradas en la ilustración 1.2. Se crearán cuatro códigos distintos, uno para el servidor y cliente de cada computadora.

**Ilustración 1.2. Librerías para sockets en Windows.**

```
1   #include <iostream>
2   #include <string>
3   #include <WS2tcpip.h>
4
5    #pragma comment (lib, "ws2_32.lib")
```

*Fuente: Elaboración propia.*

Un socket es un punto de comunicación entre dos máquinas compuesto por la dirección IP de la máquina, que en muchos casos se refiere al nombre del socket y el número de puerto usado por el software TCP.

Hay muchas clases de socket diferentes, por un lado, se encuentran los que permiten la comunicación entre sistemas diferentes y, por otro lado, los que comunican procesos de un mismo sistema. Para diferenciarlos se utilizan los nombres o direcciones ya que a cada clase de sockets le corresponde un espacio de nombres diferente. Los dos espacios de nombres básicos son de ficheros e Internet. La diferencia entre ambos es que el de ficheros se utiliza en los sockets que comunican procesos con el mismo sistema y el espacio de nombres de internet se utiliza en los sockets que deben comunicar un proceso con otro de cualquier otro sistema.

Debido a que estamos usando el protocolo TCP nos interesa el espacio de nombres de Internet, que este mismo espacio se usaría en el caso del protocolo UDP. La dirección del socket en este caso consta de la dirección de red de nodo, es decir de la dirección de la máquina correspondiente, un número de puerto para distinguir los sockets de una máquina específica y el protocolo concreto que debe de usarse.

En este caso los sockets siguen el modelo cliente/servidor donde ambas computadoras tienen sockets. La máquina que se denomina cliente es la que inicia la conversación mientras que la máquina correspondiente al servidor espera a que el cliente inicie la conversación y responde.

Nos interesa profundizar sobre dos variantes de sockets de Internet, sockets de datagrama y sockets de secuencia de bytes. La diferencia entre ambos es el

protocolo de transporte de transmisión de datos usado. En este caso, utilizaremos el correspondiente al protocolo TCP, que utilizará un socket de secuencia de bytes, donde los datos se transmiten de extremo a extremo como una corriente o flujo ordenado de bytes.

Este estilo también es conocido como orientado a la conexión porque la comunicación consiste en establecer previamente una conexión con el socket remoto y después utilizarla para el intercambio de datos.

Los pasos que se deben de seguir para establecer una comunicación entre dos sockets son los siguientes:

Las operaciones que debe de efectuar el servidor son:

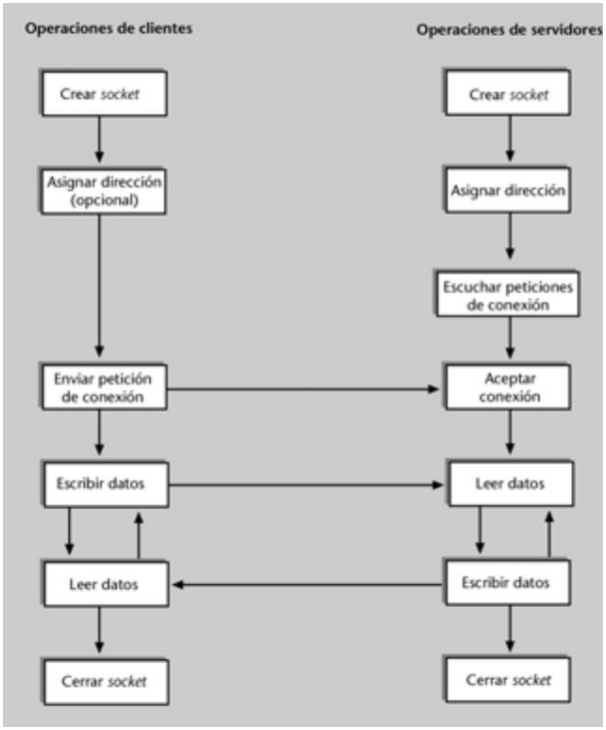
- Creación de un socket.
- Asignar una dirección IP al socket.
- Dejar el socket preparado para recibir conexiones.
- Activamente intentar establecer una conexión.
- Intercambiar datos con el cliente que ha solicitado la comunicación.
- Cerrar el socket liberando la conexión.

Por otro lado, las operaciones que debe seguir el cliente son:

- Creación de un socket.
- Asignar una dirección IP al socket.
- Enviar una petición de conexión al servidor.
- Intercambiar datos con el cliente que ha solicitado la comunicación.
- Cerrar el socket liberando la conexión.

En el esquema 1.1. se observa las operaciones correspondientes a los clientes y a los servidores.

**Esquema 1.1. Esquema de las operaciones correspondientes a los clientes y a los servidores**




*Fuente: Internet*

Se comenzará con el servidor, lo primero que hay que hacer es la creación e inicialización del socket una vez que ya se han introducido las librerías de la ilustración 1.2. En la ilustración 1.3 se observa como con la función *WSAStartup* se genera un canal de comunicación para la inicialización del socket.

El ciclo *if* que se muestra en la ilustración corresponde con un control de errores en el que la variable *wsOk* cuando la variable es distinta de cero escribe un mensaje por pantalla para que el usuario sepa que no se está inicializando con éxito el socket. El valor de esta variable es generado por la función *WSAStartup.*

**Ilustración 1.3. Inicialización de un socket para la comunicación.**

```
9   void main()
10  {
11      //Initialize winsock
12      WSADATA wsData;
13      WORD ver = MAKEWORD(2, 2);
14
15      int wsOk = WSAStartup(ver, &wsData);
16      if(wsOk != 0)
17      {
18          cerr << "Can't Inizialize winsock! Quitting" << endl;
19          return;
20      }
21
```

*Fuente: Elaboración propia.*

Una vez generado el canal de comunicación se debe crear el socket. Como el protocolo TCP está orientado a la conexión debe de estar preparado para recibir conexiones de manera pasiva, para ello utilizamos la llamada *listening* como se observa en la ilustración 1.4. El fin de utilizar esta llamada consiste en crear una cola donde se irán guardando las peticiones que lleguen. En el caso de que no se pudiera crear el socket aparecería un mensaje por pantalla ya que se ha introducido una parte de control de errores en las líneas de la 24 a la 28.

Además, se define el tipo de socket que se utiliza que en este caso se trata de *SOCK_STREAM* ya que se usa el protocolo TCP como se ha citado con anterioridad, en cambio si se usara el protocolo UDP sería del tipo *SOCK_DGRAM*. *SOCK_STREAM* representa el estilo de comunicación secuencia de bytes, es decir, está orientado a la conexión, mientras que *SOCK_DGRAM* corresponde con el estilo de comunicación datagrama.

**Ilustración 1.4. Creación del socket.**

```
22      //Create a socket
23      SOCKET listening = socket(AF_INET, SOCK_STREAM, 0);
24      if (listening == INVALID_SOCKET)
25      {
26          cerr << "Can't create a socket! Quitting" << endl;
27          return;
28      }
```

*Fuente: Elaboración propia.*

Cuando se ha creado el socket se debe asignar una IP a la dirección del socket. Para ello se utilizará la variable *hint* de tipo *sockaddr_in* como se observa en la ilustración 1.5. la cual se establece como *INADDR_ ANY*. *INADDR_ANY* representa una dirección IP indeterminada, se utiliza esta porque pueden recibir peticiones de conexión destinadas a cualquier dirección IP si el ordenador tiene más de una.

Por otro lado, se debe asignar un puerto al canal de comunicación como se observa es el canal 2001 para la conexión entre el servidor 1 y el cliente 2.

**Ilustración 1.5. Asignación de dirección IP**

```
30      //Bind the socket to an ip address and port
31      sockaddr_in hint;
32      hint.sin_family = AF_INET;
33      //hint.sin_port = htons(54000); //ports
34      hint.sin_port = htons(2001); //ports
35      hint.sin_addr.S_un.S_addr = INADDR_ANY; //IP address
36      //hint.sin_addr.S_un.S_addr = inet_pton(AF_INET, "192.168.254.1", &hint);
37
38      bind(listening, (sockaddr*)&hint, sizeof(hint));
39
40      //Tell winsock the socket is for listening
41      listen(listening, SOMAXCONN);   //maximum amount of connection we can have
```

*Fuente: Elaboración propia.*

A continuación, se debe preparar el socket para recibir conexiones como se muestra en la ilustración 1.6. Como se le ha aplicado la llamada *listen* se debe aplicar la llamada *accept* para establecer las conexiones con los clientes que las soliciten. Dicha llamada consiste en extraer la primera petición de la cola y crear un nuevo socket que tiene la misma dirección que el original y está conectado al creador de la petición.

**Ilustración 1.6. Preparar el socket para recibir conexiones.**

```
43      //Wait for a connection
44      sockaddr_in client;
45      int clientSize = sizeof(client);
46
47      SOCKET clientSocket = accept(listening, (sockaddr*)&client, &clientSize);  //socket that returns message in TCP
48
49      //Get information from the client such as IP in string of characters
50      char host[NI_MAXHOST]; //Client's remote name
51      char service[NI_MAXSERV]; //Service (i.e. port) the client is connect on
52
53      ZeroMemory(host, NI_MAXHOST);
54      ZeroMemory(service, NI_MAXSERV);
55
56
57      inet_ntop(AF_INET, &client.sin_addr, host, NI_MAXHOST);
58      cout << " IP: " << host << " " << "Connected on port " <<
59          ntohs(hint.sin_port) << endl;
60
61      //Close listening socket
62      closesocket(listening);
```

*Fuente: Elaboración propia.*

Antes de intentar establecer conexión con el cliente se crea un buffer de almacenamiento donde se guarda el mensaje recibido como se muestra en la ilustración 1.7. Todo está dentro de un ciclo que comprueba repetidamente si se ha recibido algún mensaje con la función *recv,* la cual permite leer datos e intentar de esa manera establecer conexión con el cliente. El primer parámetro de la función *recv* corresponde con el descriptor del socket, el parámetro *buf* es un puntero a la zona de memoria en la cual se dejan los datos leídos y el valor *4096* se refiere al número máximo de bytes que queremos leer. Al utilizar la función *recv* en vez de *read* disponemos de un cuarto parámetro, *flags*, pero como en este caso es 0 la función actúa exactamente igual que la función *read*.

Los ciclos que aparecen es un simple control de errores para comprobar si sigue conectado al cliente correspondientes con las líneas 73 a la 83 de la ilustración 1.7.

**Ilustración 1.7. Establecer conexión con el cliente.**

```
64      //While loop: accept and echo message back to client
65      char buf[4096];
66
67      while (true)
68      {
69          ZeroMemory(buf, 4096);
70
71          //Wait for client to send data
72          int bytesReceived = recv(clientSocket, buf, 4096, 0);
73          if (bytesReceived == SOCKET_ERROR)
74          {
75              cerr << "Error in recv(). Quitting " << endl;
76              break;
77          }
78
79          if (bytesReceived == 0)
80          {
81              cout << "Client disconnected " << endl;
82              break;
83          }
```

*Fuente: Elaboración propia.*

Por último, con la función *send* se escriben los datos. En el parámetro *buf* queda almacenada la dirección de memoria donde están los datos que se desean escribir y en el tercer parámetro se indica cuantos bytes se quieren escribir como se muestra en la ilustración 1.8. Al utilizar la función *send* en vez de *write* se dispone de un cuarto parámetro, *flags*, pero como en este caso es 0 la función actúa exactamente igual que la función *write*.

**Ilustración 1.8. Almacenamiento del mensaje recibido.**

```
84
85              cout << string(buf, 0, bytesReceived) << endl;
86
87              //Echo message back to client
88              send(clientSocket, buf, bytesReceived + 1, 0);
89
90          }
91
```

*Fuente: Elaboración propia.*

Para finalizar, lo que se tiene que hacer es liberar la conexión cerrando el socket como se muestra en la ilustración 1.9.

**Ilustración 1.9. Cierre del socket.**

```
92      //Close the socket
93      closesocket(clientSocket);
94
95      //Cleanup winsock
96      WSACleanup();
97      system("pause");
98
99  }
```

*Fuente: Elaboración propia.*

Se debe de implementar dos códigos que corresponden con el cliente y dos con el servidor como se ha citado con anterioridad, uno para cada computadora. La diferencia que existe entre el segundo código del servidor es en la parte de preparar al socket para recibir conexiones, el resto de código resulta exactamente igual que las ilustraciones mostradas.

La parte que debe de ser modificada es que corresponde con la declaración del puerto que sustituimos el puerto 2001 por el puerto 2002 para la conexión entre el servidor 2 y el cliente 1 como se muestra en la ilustración 1.10. en la línea 34.

**Ilustración 1.10. Asignación de direcciones IP de la computadora 2.**

```
30  //Bind the socket to an ip address and port
31  sockaddr_in hint;
32  hint.sin_family = AF_INET;
33  //hint.sin_port = htons(54000); //ports
34  hint.sin_port = htons(2002); //ports
35  hint.sin_addr.S_un.S_addr = INADDR_ANY; //IP address
36  //hint.sin_addr.S_un.S_addr = inet_pton(AF_INET, "192.168.254.3", &hint);
37
38  bind(listening, (sockaddr*)&hint, sizeof(hint));
39
40  //Tell winsock the socket is for listening
41  listen(listening, SOMAXCONN);   //maximum amount of connection we can have
42
```

*Fuente: Elaboración propia.*

Una vez finalizados los dos códigos para el servidor se debe implementar los códigos para el cliente. Se comenzará con el cliente de la computadora uno y de igual manera que el código del servidor se utilizaran sockets con sus respectivas librerías. Los pasos que hay que seguir son los descritos con anterioridad que se pueden recordar con el esquema 1.1.

De igual forma que el servidor, lo primero que hay que hacer es la creación e inicialización del socket como se ve en la ilustración 1.11 utilizando la función *WSAStartup* para la generación de un canal de comunicación. A diferencia del código del servidor se introduce la variable *ipAddress* que contiene la dirección

IP del servidor y la variable port que contiene el puerto del cliente de la segunda computadora que se conectará con el servidor de la computadora uno.

**Ilustración 1.11. Inicialización del socket.**

```
9   void main()
10  {
11      //string ipAddress = "127.0.0.1";  //IP Address of the server
12      string ipAddress = "192.168.254.1";  //IP Address of the server
13
14      //int port = 54000;             //Listening port # in the server
15      int port = 2002;
16
17      //Initialize WinSock
18      WSADATA data;
19      WORD ver = MAKEWORD(2, 2);
20      int wsResult = WSAStartup(ver, &data);
21      if (wsResult != 0)
22      {
23          cerr << "Can't start winsock, Err # " << wsResult << endl;
24          return;
25      }
26
```

*Fuente: Elaboración propia.*

Una vez creado el canal de comunicación se procede a la creación del socket como se observa en la ilustración 1.12., en el caso de que se produjera un error se mostraría un mensaje por pantalla informando de que no se ha podido crear el socket.

**Ilustración 1.12. Creación del socket.**

```
27      //Create socket
28      SOCKET sock = socket(AF_INET, SOCK_STREAM, 0);
29      if (sock == INVALID_SOCKET)
30      {
31          cerr << "Can't create socket, Err # " << WSAGetLastError() << endl;
32          WSACleanup();
33          return;
34      }
35
```

*Fuente: Elaboración propia.*

Cuando se ha creado el socket se debe asignar una IP a la dirección del socket. Para ello se utilizará una estructura denominada *hint* de tipo *sockaddr_in* como se observa en la ilustración 1.13.

**Ilustración 1.13. Creación de una estructura.**

```
36      //Fill in a hint structure
37      sockaddr_in hint;
38      hint.sin_family = AF_INET;
39      hint.sin_port = htons(port);
40      inet_pton(AF_INET, ipAddress.c_str(), &hint.sin_addr);
41
```

*Fuente: Elaboración propia.*

A continuación, se debe de enviar una petición de conexión al servidor. Esto se realiza mediante la función *connect* que sirve para que un cliente inicie de manera activa una conexión y en el caso de que no se pueda establecer alertar mediante un mensaje de error como se puede ver en la ilustración 1.14.

El primer parámetro que forma la función *connect* corresponde con la llamada que sigue los pasos necesarios para establecer una conexión con el socket servidor cuya dirección corresponde con el segundo parámetro. El tercer parámetro se refiere a al número de bytes que ocupa dicha dirección. Esta llamada no retorna nada hasta que el servidor no da una respuesta.

**Ilustración 1.14. Enviar una petición de conexión al servidor.**

```
42      //Connect to server
43      int connResult = connect(sock, (sockaddr*)&hint, sizeof(hint));
44      if (connResult == SOCKET_ERROR)
45      {
46          cerr << "Can't connect to server, Err # " << WSAGetLastError() << endl;
47          closesocket(sock);
48          WSACleanup();
49          return;
50      }
51
52      //Do-while loop to send and receive data
53      char buf[4096];
54      string userInput;
55
```

*Fuente: Elaboración propia.*

Una vez establecida la conexión se debe enviar el mensaje, para ello se utiliza un ciclo que entra en un bucle esperando recibir confirmación, como se muestra en la ilustración 1.15. De la misma manera que en el servidor, se escriben los datos con la función *send*.

**Ilustración 1.15. Intercambiar datos.**

```
56      do
57      {
58          //Prompt the user for some text
59          cout << "> ";
60          getline(cin, userInput);
61
62          if (userInput.size() > 0) //Make sure the user has typed in something
63          {
64              //Send the text
65              int sendResult = send(sock, userInput.c_str(), userInput.size() + 1, 0);
66              if (sendResult != SOCKET_ERROR)
67              {
68                  //Wait for response
69                  ZeroMemory(buf, 4096);
70                  int bytesReceived = recv(sock, buf, 4096, 0);
71                  if (bytesReceived > 0)
72                  {
73                      //Echo response to console
74                      cout << "SERVER> " << string(buf, 0, bytesReceived) << endl;
75
76                  }
77
78              }
79
80          }
81
82      } while (userInput.size() > 0);
```

*Fuente: Elaboración propia.*

Para finalizar el código hay que cerrar el socket con la función *closesocket* para liberar la conexión como se muestra en la ilustración 1.16.

**Ilustración 1.16. Cerrar socket liberando la conexión.**

```
84      //Close down everything
85      closesocket(sock);
86      WSACleanup();
87  }
88
```

*Fuente: Elaboración propia.*

De una forma análoga que en el caso del servidor se debe de crear un código cliente para la segunda computadora. El código seria exactamente el mismo con una única diferencia a la hora de definir la IP del servidor se pone la correspondiente con la computadora 2 al igual que el puerto que corresponde con el puerto del cliente de la primera computadora que se conectará con el servidor de la segunda computadora. Se pueden observar los cambios en la ilustración 1.17 en las líneas 12 y 15 respectivamente.

**Ilustración 1.17. Asignación de direcciones IP de la computadora 2.**

```
9   void main()
10  {
11      //string ipAddress = "127.0.0.1";  //IP Address of the server
12      string ipAddress = "192.168.254.3";  //IP Address of the server
13
14      //int port = 54000;             //Listening port # in the server
15      int port = 2001;
16
17      //Initialize WinSock
18      WSADATA data;
19      WORD ver = MAKEWORD(2, 2);
20      int wsResult = WSAStartup(ver, &data);
21      if (wsResult != 0)
22      {
23          cerr << "Can't start winsock, Err # " << wsResult << endl;
24          return;
25      }
26
```

*Fuente: Elaboración propia.*

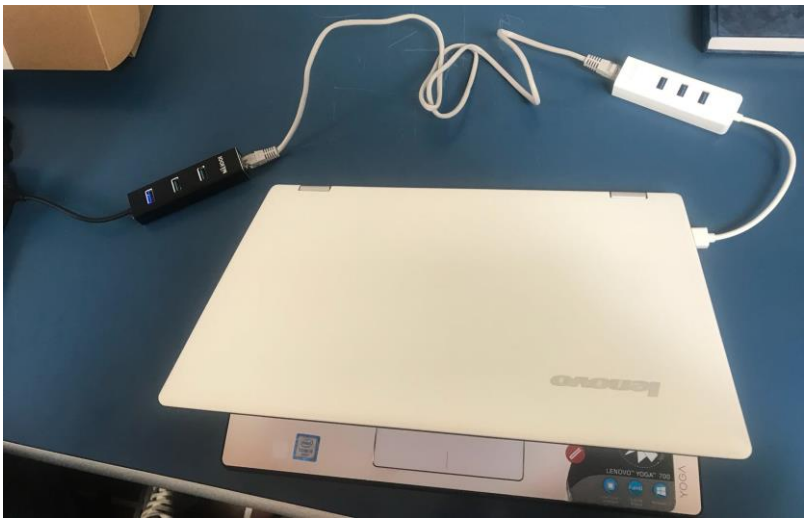
# CAPÍTULO 2. APLICACIONES.

## 2.1. TRANSMISIÓN POR CABLE ETHERNET.

Cuando se comprueba que los cuatro códigos creados, dos para el servidor y dos para el cliente, compilan correctamente y no tienen ningún error hay que ver las aplicaciones que tienen dichos códigos. En primer lugar, se va a comprobar su funcionamiento de una manera sencilla utilizando un cable Ethernet por donde se transmitirán los datos. Para ello, se conectará el cable entre la computadora uno y la computadora dos.

Como se ha visto en el capítulo uno hay que asignar una dirección IP a las computadoras para poder realizar la transmisión de datos, esto se realiza de forma manual. Una vez asignadas las direcciones IP correspondientes se debe de conectar el cable Ethernet a ambas computadoras como se muestra en la ilustración 2.1.

**Ilustración 2.1. Conexión de cable Ethernet entre las dos computadoras.**



*Fuente: Elaboración propia.*

Una vez conectado el cable Ethernet se debe comprobar el estado de la conexión del host local y el equipo remoto en una red, para ello se utilizará el comando *ping* como se muestra en la ilustración 2.2. Las direcciones IP que se muestran en dicha ilustración son las utilizadas en la primera prueba que corresponden con son 192.168.254.1 y 192.168.254.2. Al final las IP utilizadas son 192.168.254.1 y 192.168.254.3 como se ha visto en el capítulo anterior.

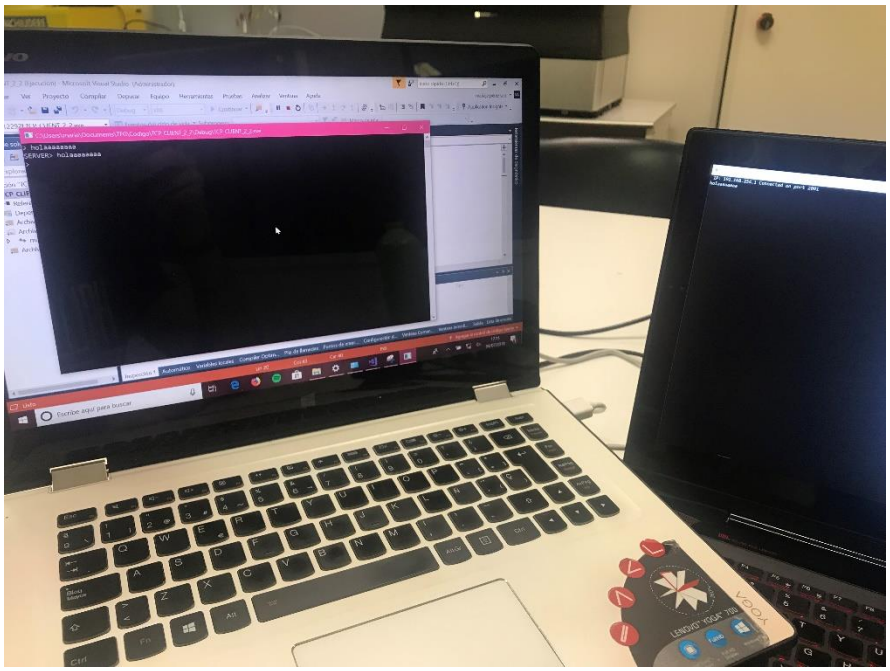
**Ilustración 2.2. Ejecución del comando ‘ping’.**

```
C:\Users\maria>ping 192.168.254.1

Haciendo ping a 192.168.254.1 con 32 bytes de datos:
Respuesta desde 192.168.254.1: bytes=32 tiempo<1m TTL=128
Respuesta desde 192.168.254.1: bytes=32 tiempo<1m TTL=128
Respuesta desde 192.168.254.1: bytes=32 tiempo<1m TTL=128
Respuesta desde 192.168.254.1: bytes=32 tiempo<1m TTL=128

Estadísticas de ping para 192.168.254.1:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
    (0% perdidos),
Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 0ms, Máximo = 0ms, Media = 0ms

C:\Users\maria>ping 192.168.254.2

Haciendo ping a 192.168.254.2 con 32 bytes de datos:
Respuesta desde 192.168.254.2: bytes=32 tiempo=1ms TTL=64
Respuesta desde 192.168.254.2: bytes=32 tiempo=1ms TTL=64
Respuesta desde 192.168.254.2: bytes=32 tiempo=1ms TTL=64
Respuesta desde 192.168.254.2: bytes=32 tiempo=1ms TTL=64

Estadísticas de ping para 192.168.254.2:
    Paquetes: enviados = 4, recibidos = 4, perdidos = 0
    (0% perdidos),
Tiempos aproximados de ida y vuelta en milisegundos:
    Mínimo = 1ms, Máximo = 1ms, Media = 1ms
```

*Fuente: Elaboración propia.*

A continuación, se procede a la ejecución del programa como se observa en la ilustración 2.3. en la que el ordenador de la izquierda corresponde con el cliente de la computadora dos, mientras que el de la derecha con el servidor uno.

**Ilustración 2.3. Prueba a través del cable Ethernet.**



*Fuente: Elaboración propia.*

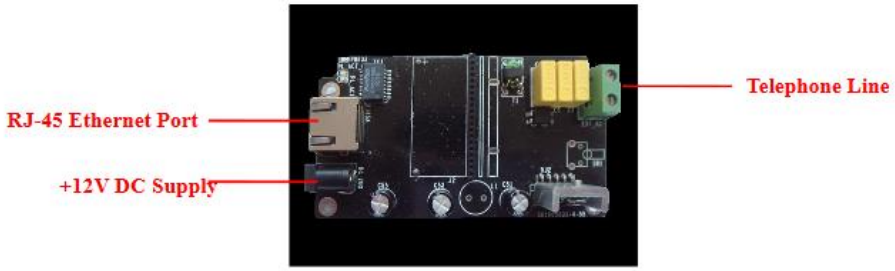
En la ejecución se ve como el cliente envía un mensaje con el texto ‘holaaaaaaaa’, el servidor lo recibe y aparece en su pantalla. Una vez que lo ha recibido, al cliente le aparece el mismo mensaje para saber que ha sido recibido por el servidor.

Con esta prueba se puede concluir que el trabajo realizado por el cliente y por el servidor es correcto y que se ha conseguido enviar correctamente un mensaje entre dos computadoras mediante el cable Ethernet.

## 2.2. TRANSMISIÓN MEDIANTE LA PLACA LX200V20.

Otra forma de comprobar su funcionamiento es mediante una placa, pero antes se debe de estudiar el funcionamiento de esta. La placa que se va a utilizar está formada principalmente por tres partes como se muestra en la ilustración 2.4. Se observa que una parte corresponde con la alimentación ya bien por medio de una entrada USB o por una fuente; los puertos de entrada Ethernet; y por último la parte correspondiente a la PowerLine que se encarga de transmitir la señal.

**Ilustración 2.4. Partes relevantes de la placa.**




*Fuente: Datasheet de la placa.*

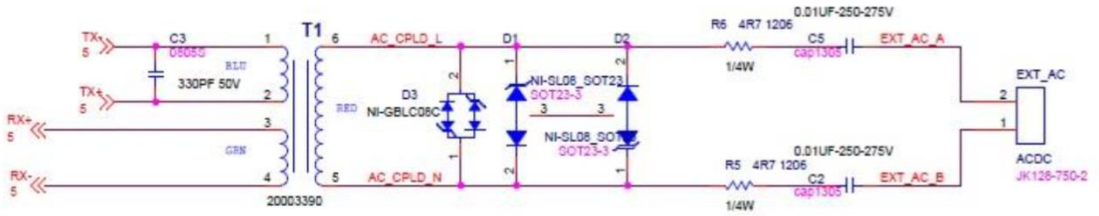
La parte correspondiente a la PowerLine hay que estudiarla con profundidad debido que para realizar la comunicación submarina mediante un LED hay que sustituir los dos cables que conectan las placas por un LED y un fotodiodo. No debería de tener ningún efecto a la hora de la transmisión de datos con la diferencia de que la señal viaja mediante la luz en vez de por cable.

Aparecen varios problemas, el primer problema que se observa es en el uso de dos cables para las dos placas, cuando se necesitan cuatro cables de salida de cada placa para conectar el LED y el fotodiodo. Para solucionar esto se debe de estudiar con ayuda de la hoja de características la placa para ver como convierte

dos cables de transmisión (TX) y los dos de recepción (RX) en uno de transmisión y uno de recepción.

Uno de los circuitos que se observan en la hoja de características es el mostrado en la ilustración 2.5. que corresponde con la parte de transmisión y recepción de la placa. Analizando el circuito se ve que mediante un transformador unifica los cables de transmisión y recepción.

**Ilustración 2.5. Circuito TX/RX.**

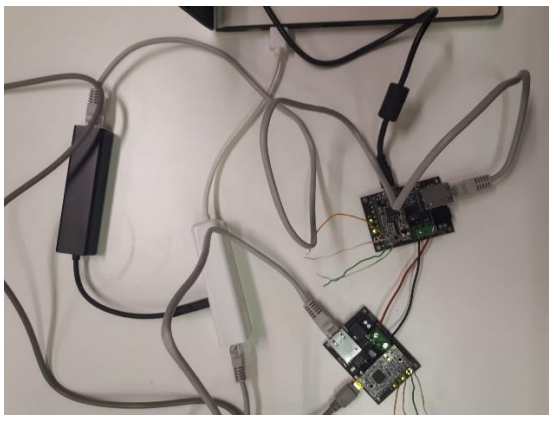



*Fuente: Datasheet de la placa LX200V20.*

Para comprobar que la placa funciona correctamente se realizara una primera prueba con la transmisión de las placas mediante dos cables. Para ello, se conectará dos cables directamente al puerto de salida de la placa y estos cables irán conectados al puerto de entrada de la segunda placa.

En la ilustración 2.6. se observa cómo se ha realizado la conexión entre las placas y las computadoras. Se puede ver como además de los cables que conectan las placas hay otros dos cables conectados a cada placa. Estos cables son los correspondientes con las tomas de alimentación que van directamente a la computadora a través de un cable USB.

**Ilustración 2.6. Conexión de las dos placas utilizando la configuración convencional.**



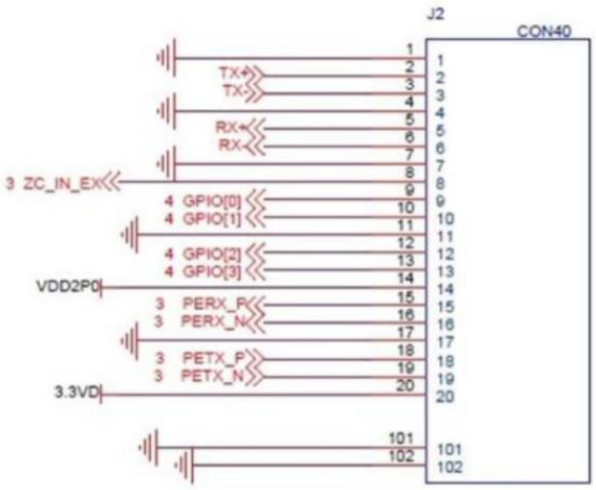
*Fuente: Elaboración propia.*

Se enviaron varios mensajes como se hizo con el cable Ethernet y se comprobó que los mensajes eran recibidos de forma correcta. A la hora del envío de un mensaje entre una computadora y otra no se deberían de observar cambios. Aunque la única diferencia que se puede dar es un retardo temporal con respecto a la transmisión directa mediante un cable Ethernet. Este retardo del que se habla no se puede apreciar a simple vista ya que es muy pequeño.

Analizando las pruebas que se han realizado se puede llegar a la conclusión que las placas funcionan bajo lo previsto y permiten modificar los datos para poder ser transmitidos a través de cables convencionales.

Una vez llegado a este punto, se procede a conseguir obtener dos cables de transmisión y dos cables de recepción. Con ese fin, se estudia de nuevo las hojas de características, pero en este caso se presta atención en los pines como se muestra en la ilustración 2.7.

**Ilustración 2.7. Pines de la placa LX200V20.**




*Fuente: Datasheet de la placa LX200V20.*

Se puede comprobar en la ilustración 2.7. que las entradas TX y RX aparecen como puertos de entrada y salida respectivamente de la placa. Los puertos que corresponden a la transmisión son 2 y 3, mientras los que se refieren a la recepción son los puertos 5 y 6.

Por tanto, al representarse como puertos de entrada/salida es posible soldar directamente los cables a dichos puertos de la placa al tener contactos metálicos como se puede apreciar en la ilustración 2.8.

**Ilustración 2.8. Conexión de dos cables TX y dos cables RX.**

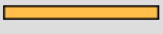
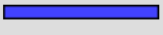
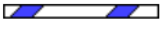
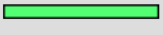
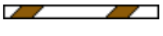
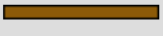

*Fuente: Elaboración propia.*

En la ilustración 2.8. se observa la placa LX200V20 que va montada encima de la placa general. En el lateral izquierdo de la placa LX200V20 corresponde con los puertos de la placa a la que van soldados los cuatro cables. Además, se observa el cable que va conectado al puerto USB que proporciona la alimentación de la placa.

Se puede ver que los cables soldados cada uno es de un color: naranja, naranja y blanco, verde y verde y blanco. Se han escogido estos colores para alterar de la menor manera posible la configuración inicial sin placas. Esto es porque dichos cables representan la utilidad de los cables de dentro del cable Ethernet general.

Los cables internos de un cable Ethernet vienen representados en la tabla 2.1. en la cual se puede ver una definición de cada uno de los cables según sus características.

**Tabla 2.1. Clasificación de cables internos de un cable Ethernet.**

| RJ45 Pin # | Wire Color (T568B) | Wire Diagram (T568B) | 10Base-T Signal 100Base-TX Signal | 1000Base-T Signal |
|---|---|---|---|---|
| 1 | White/Orange |  | Transmit+ | BI_DA+ |
| 2 | Orange |  | Transmit- | BI_DA- |
| 3 | White/Green |  | Receive+ | BI_DB+ |
| 4 | Blue |  | Unused | BI_DC+ |
| 5 | White/Blue |  | Unused | BI_DC- |
| 6 | Green |  | Receive- | BI_DB- |
| 7 | White/Brown |  | Unused | BI_DD+ |
| 8 | Brown |  | Unused | BI_DD- |

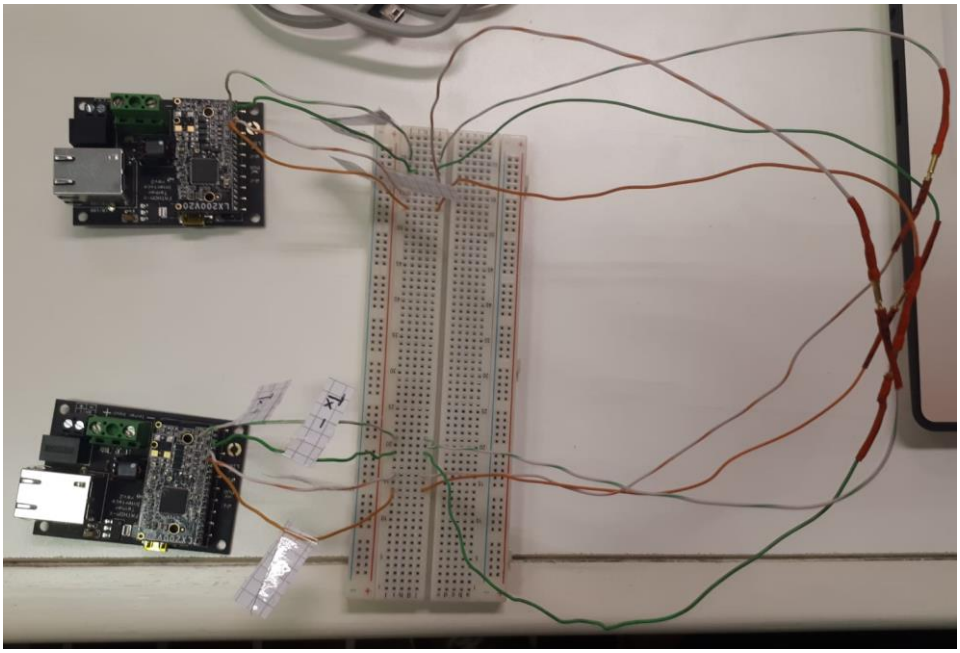
*Fuente: Internet.*

Una vez soldados los cables hay que comprobar que funciona de igual manera que con anterioridad. Para las dos verificaciones anteriores se creaban canales de transmisión de datos probados que deberían de funcionar. En cambio, en este caso, al conectarse los cuatro cables directamente a la placa se modifica el funcionamiento que desempeñan con normalidad las placas.

Como se está alterando el funcionamiento habitual es importante comprobar de forma cuidadosa el funcionamiento del canal a la hora de la transmisión de datos. En esta configuración no es necesaria la utilización de la placa general, bastaría solo con la placa LX200V20 con la utilización de los puertos para los cables convencionales. Pero por comodidad, no se decidió prescindir de la placa general ya que tiene una parte referida a la alimentación mediante el cable USB y el puerto a los cables Ethernet que resulta muy cómoda.

En los extremos finales de cada uno de los cuatro cables extraídos de la ilustración 2.8 se han acoplado unos conectores macho y hembra, para de esta forma minimizar las pérdidas a la hora de la conexión. Estos conectores se podían haber prescindido, pero para pruebas futuras serán más efectivas. La configuración final se muestra en la ilustración 2.9.

**Ilustración 2.9. Conexión a la placa mediante cuatro cables.**



*Fuente: Elaboración propia.*

Una vez que tenemos la configuración lista se debe comprobar que funciona de forma análoga que en los casos anteriores. De igual manera, se comprobó que el mensaje era enviado y recibido con éxito a través del canal.

Un efecto causado al extraer los cuatro cables de la placa LX200V20, es que resulta posible conectar un LED y un fotodiodo en los extremos de los cables para conseguir transmitir el mensaje mediante la luz.

# CAPÍTULO 3. ESTUDIO DE LA SEÑAL.

## 3.1. VISUALIZACIÓN DE LA SEÑAL.

Hasta este punto se ha logrado preparar el canal de transmisión de datos con una configuración la cual permite la conexión entre el diodo emisor y el fotodiodo receptor que funciona correctamente. Si se recapitula, el objetivo de este trabajo se basa en preparar un canal de comunicación de datos submarino que funcione utilizando la transmisión de datos mediante una señal luminosa en el medio del agua. Aunque por el momento, los cables por los que viaja la señal están conectados directamente.

El siguiente paso se basa en conectar el diodo y el fotodiodo directamente a los cables del canal, aunque antes de ello se debe de estudiar la señal que viaja por dicho canal. Esto es muy importante ya que en función de las características de la señal obtenida se deberá de introducir un diodo y un fotodiodo con unas características u otras. En el caso de saltarse el estudio de la señal y así poner un diodo y un fotodiodo con ciertas características puede llevar a que no funcionase bien o incluso estropear los diodos y fotodiodos que se utilizaran.

En el estudio de la señal, el parámetro más importante que hay que tener en cuenta es la referencia a los niveles de tensión que puede alcanzar la señal en el modo de transmisión. Para su estudio se utilizará un osciloscopio digital que ayudará a determinar los niveles de tensión que alcanza la señal tanto cuando se transmiten datos y cuando no.

Virtual Bench de National Instruments es el osciloscopio digital que se va a utilizar para realizar dicho estudio. El osciloscopio citado hay que conectarlo a una computadora donde se puedan visualizar las medidas, ya que no posee una pantalla propia. Aunque para su beneficio, el osciloscopio cuenta con una plataforma informática LabView la cual tiene un programa instalado que posibilita observar las señales del osciloscopio.

Una función que resulta beneficiosa que posee este programa es que incluye una opción que congela la señal en un instante de tiempo concreto para así poder observar con mayor detalle la señal ya que en su defecto la señal sino estaría en constante movimiento. Del mismo modo, posee unos marcadores para poder observar los niveles de tensión y tiene la opción de extraer los datos recogidos a lo largo de todo el eje temporal en un archivo de tipo Excel para así poder estudiar con mayor profundidad los datos y representarlos en un gráfico.

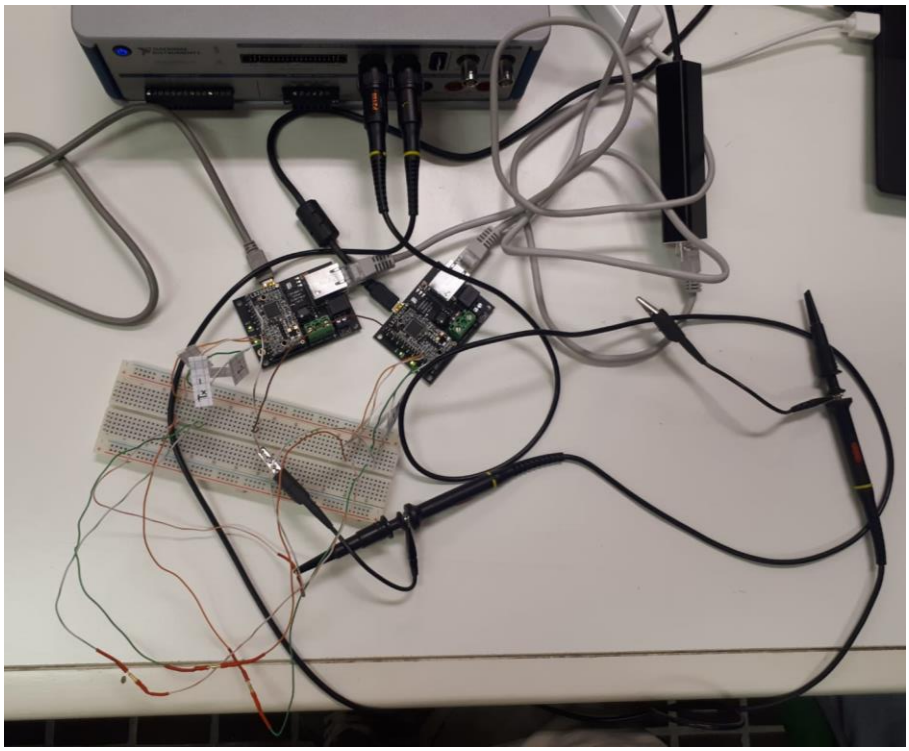
## 3.2. VISUALIZACIÓN DE LA SEÑAL EN EL CASO DE SIN TRANSMISIÓN.

Como ya se ha visto previamente en la configuración de los pines de la ilustración 2.7. los puertos de las placas se representan como $TX^+$ y como $TX^-$, para los puertos de transmisión y $RX^+$ y $RX^-$ para los puertos de recepción. Viendo esto las conexiones deben hacerse siguiendo la regla de que el puerto $TX^+$ de una de las placas esté conectado con el puerto $RX^+$ de la otra placa. Esto se verifica con las uniones mostradas en la ilustración 2.9. Y en la tabla 2.1.

Es de vital importancia hacer una clara diferencia entre el $TX^+$ y el $TX^-$ ya que el $TX^+$ viajará la señal mientras que $TX^-$ se puede tomar como tierra. Al igual que en el caso de la transmisión esta diferencia también es igual de importante con $RX^+$ y $RX^-$. La existencia de dos cables se debe a la diferencia de voltaje que existe entre ambos que debe de ser 0 o 1 en binario al transmitir datos.

Como por $TX^+$ del cliente viaja la señal es a este puerto donde se debe de conectar la sonda del osciloscopio para visualizar la señal y el cable de tierra de la sonda a $TX^-$ del cliente. Como se conecta la sonda al cliente hay que saber con claridad cuál es la computadora que actúa como cliente y cual como servidor. En la figura 3.1. se observa cómo se conectado la sonda a la configuración mostrada con anterioridad en la ilustración 2.9.

**Ilustración 3.1. Conexión de la sonda con el osciloscopio y la placa.**



*Fuente: Elaboración propia.*

Como se percibe en la ilustración 3.1. la sonda se encuentra conectada al cable blanco y naranja que corresponde con el cable de transmisión del cliente ($TX^+$), siendo la placa de la izquierda la conectada a la computadora que actúa como cliente. Por otro lado, la tierra de la sonda se encuentra conectada al cable naranja que corresponde con $TX^-$ de la misma placa.

Una vez realizada esta configuración se procede a ejecutar el programa y a observar la señal a través del canal sin enviar ningún mensaje. Lo que se espera es ver una señal plana, pero en su defecto se observa lo que se muestra en la ilustración 3.2.

**Ilustración 3.2. Observación de la señal sin transmisión.**


*Fuente: Osciloscopio digital.*

Aunque se contempla que en su mayor parte del tiempo la señal es plana como se esperaba al no estar transmitiendo nada, cada cierto tiempo aparece una señal de un breve periodo de tiempo como se puede percibir en la ilustración 3.2. Como se ha dicho con anterioridad, una de las aplicaciones que tenía la plataforma LabView es el programa que permite su visualización que nos permite congelar la señal con la función stop como se puede apreciar ya que sino la señal estaría en continuo movimiento.

Una de las ventanas que vemos en dicha ilustración corresponde con la señal a lo largo de todo el tiempo que está midiendo el osciloscopio, esto se ve en la ilustración 3.3.

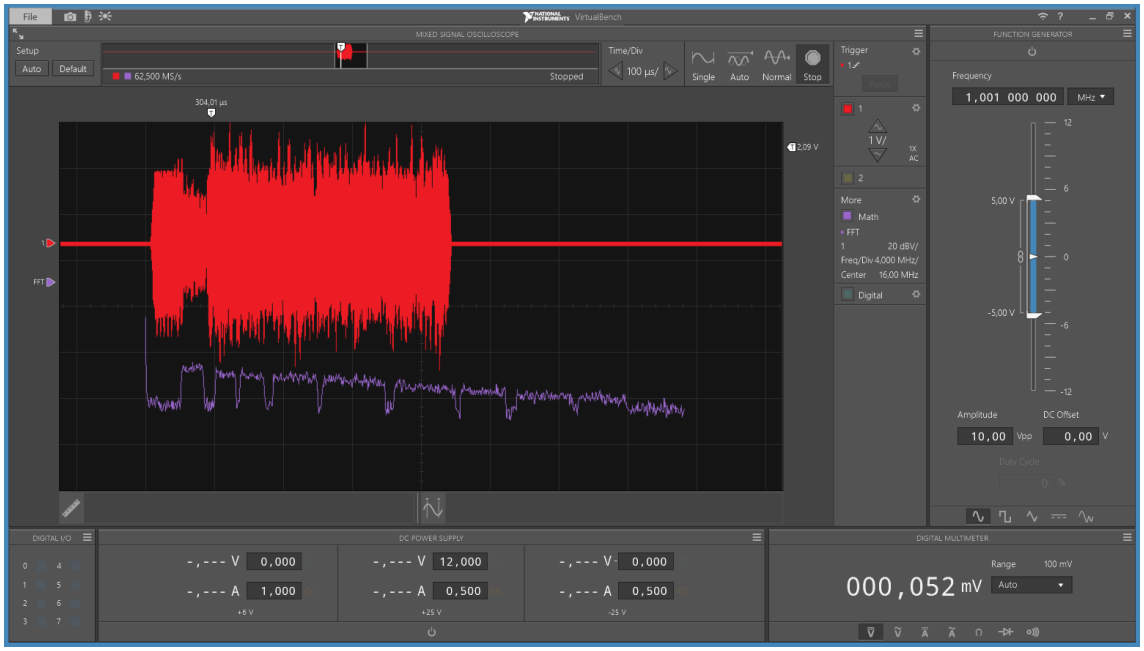
**Ilustración 3.3. Ventana de la gráfica completa del osciloscopio.**

*Fuente: Osciloscopio digital.*

En la ilustración 3.3 se ve una parte resaltada que corresponde con la ampliación vista en la ilustración 3.2. Con esta ilustración queda reflejado como el pulso se repite a lo largo del tiempo y con el mismo espacio temporal entre cada pulso.

Al aparecer esta señal no esperada hay que estudiar su señal y descubrir su utilidad. Para ello, se hizo un estudio en frecuencia como se comprueba en la ilustración 3.4. La aplicación con la que cuenta el programa permite que sobre la misma señal observar su transformada de Fourier y facilitar así el estudio.

**Ilustración 3.4. Señal sin transmisión en el dominio de la frecuencia.**

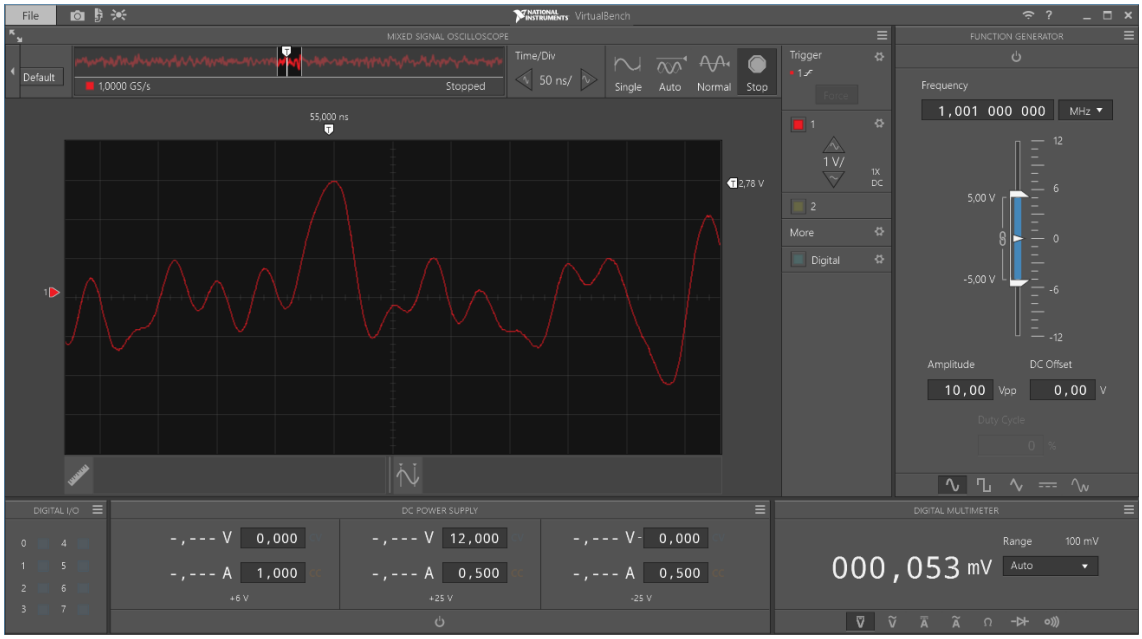



*Fuente: Osciloscopio digital.*

Examinando la transformada de Fourier obtenida en la ilustración 3.4. de un color violeta no se llegó a ninguna conclusión clara de cuál era su función y lo que representaba. Como su representación del tiempo no es algo concreto no se puede obtener una conclusión, pero se observa cómo hay diversas bandas de frecuencia que su amplitud disminuye a medida que se aumenta la frecuencia.

Como del análisis en frecuencia no se obtuvo nada claro, se procede a aumentar la señal mediante la herramienta de ‘zoom’ en el dominio del tiempo. El resultado obtenido se puede examinar en la ilustración 3.5. donde el espaciado del eje temporal corresponde con 50 ns.

**Ilustración 3.5. Señal sin transmisión ampliada en el dominio del tiempo.**

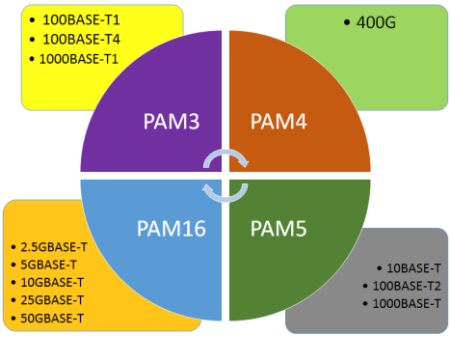



*Fuente: Osciloscopio digital.*

Mirando con profundidad el resultado obtenido en la ilustración 3.5, no se ve un patrón que siga cada cierto tiempo. Aunque realizando varias pruebas en distintos periodos de tiempo se pudo contemplar una conclusión.

La conclusión a la que se ha llegado es que la transmisión a través del canal tiene lugar a través de una modulación de tipo PAM5. La modulación PAM5 consiste en una modulación en amplitud a cinco niveles de tensión. Por lo tanto, este tipo de modulación encaja con lo esperado debido a que la utilización de las placas y el cable Ethernet es de tipo 10BASE-T. A continuación, en el gráfico 3.1. se puede hacer una comparación con el resto de las modulaciones en amplitud.

**Gráfico 3.1. Modulaciones PAM.**

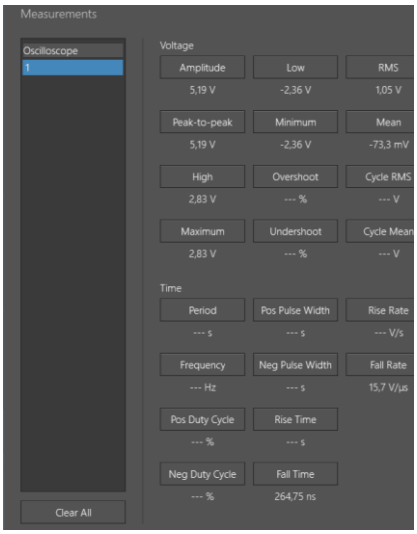



*Fuente: Internet.*

Con el conocimiento de que se está dando una modulación en amplitud se vuelve a estudiar el resultado obtenido en la ilustración 3.5. En la parte central de la señal aparece un pico de máxima tensión, mientras que el mínimo se aprecia en el extremo derecho de la gráfica. Estos picos que resaltan corresponden con los dos niveles extremos de la modulación. El resto de los picos hacen referencia a los niveles intermedios y, por último, el quinto corresponde con el nivel a cero voltios.

Para ver los niveles de tensión correspondientes a estos niveles de la modulación se vuelve hacer uso de una de las aplicaciones del programa. La aplicación que se utilizara en este caso es la mostrada en la ilustración 3.6 que permite mostrar datos numéricos en referencia a una tensión pico a pico, máximo, mínimo, etc.

**Ilustración 3.6. Aplicación del osciloscopio digital que muestra datos numéricos de la señal sin transmisión**.



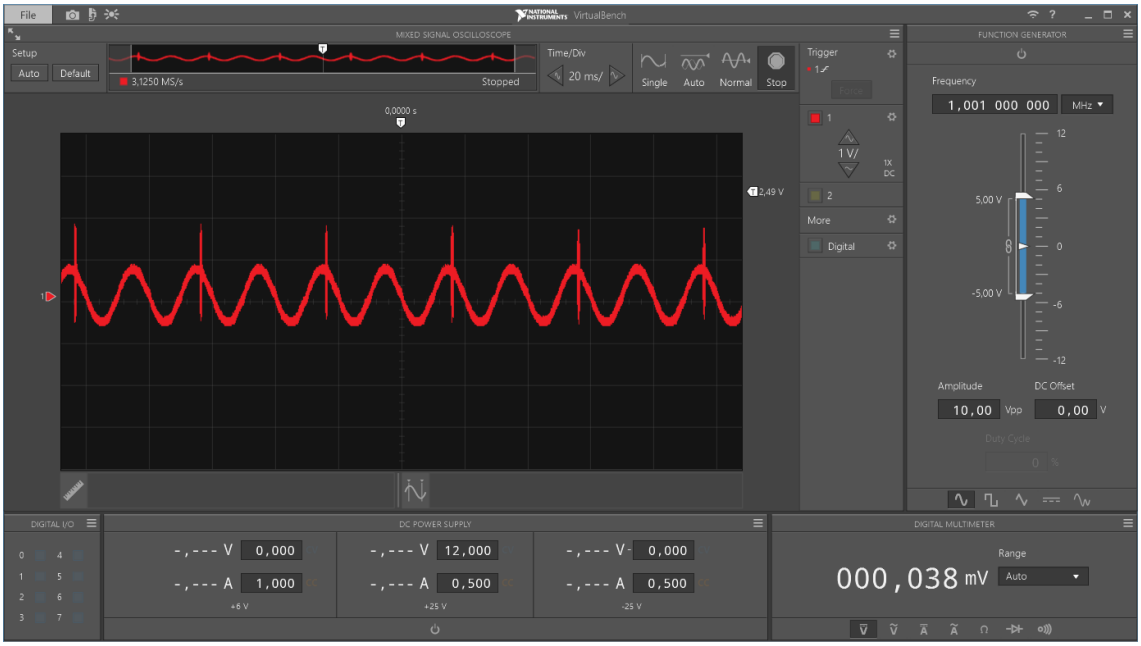

*Fuente: Osciloscopio digital.*

Como la media de la señal tiene un valor entrono a cero voltios demuestra que tiene una de sus componentes igual a cero. Por otro lado, si analizamos el valor máximo y mínimo vemos que son en torno a 3V y -3V respectivamente, aunque el valor mínimo algo más alejado a los -3 voltios.

La división que se realiza de la gráfica corresponde con 1 voltio, por lo que la señal se transmite a través del cable Ethernet utilizando una modulación a cinco niveles de amplitud que corresponden con: +3, +1, 0, -1 y -3 voltios.

Uno de los errores frecuentes que se suele producir es no conectar la sonda a tierra, el resultado del análisis es el que se muestra en la ilustración 3.7.

**Ilustración 3.7. Señal sin transmisión en el caso de no estar conectada la sonda a tierra.**




*Fuente: Osciloscopio digital.*

A pesar de no estar conectada la sonda a tierra se sigue apreciando la señal periódica cada cierto tiempo que corresponde con los picos que se observan en la ilustración 3.7. Pero en este caso, no se obtiene una onda plana cuando no hay transmisión, sino que aparece una señal sinusoidal.

Este fue uno de los problemas que se dieron a la hora de realizar varias pruebas, que al observar que el resultado era totalmente diferente a lo visto con anterioridad se investigó su causa. Entonces se retrocedió hasta el punto de la configuración donde se observó que la masa de la sonda se encontraba desconectada y en su defecto no estaba conectada al puerto $TX^{-}$.

Como se observa en la ilustración 3.7. se pierde la señal de referencia lo que provoca que la señal deje de ser plana y comience a tener un valor. Esto se debe a que la señal de referencia provoca una señal nula cuando no existe transmisión. De este fallo se puede obtener algo positivo ya que gracias a ello se ha podido observar la portante de la señal, es decir la señal a través de la cual se transmitirán los datos, que es la señal sinusoidal que se aprecia.

Aunque se ha resuelto con que correspondía el pulso que se obtenía cuando no se transmitía ningún dato, no se ha resuelto el problema inicial de por qué no solo aparece una onda plana.

Para intentar resolver la duda de por qué no aparece solo una onda plana se ha estudiado el protocolo utilizado (TCP/IP) y cómo se transmiten datos a través de cables Ethernet y se ha podido concluir con que la señal periódica corresponde con una señal de verificación automática. Esto quiere decir, que mientras no se estén enviando datos, el protocolo envía una señal muy breve para avisar al servidor de que el cliente sigue conectado.

## 3.3. VISUALIZACIÓN DE LA SEÑAL EN EL CASO DE QUE EXISTA TRANSMISIÓN.

Una vez que se ha visualizado la señal en el caso de que no exista transmisión, el siguiente paso es que exista transmisión y analizar la señal. De igual manera que sin conexión se analizará la señal utilizando el osciloscopio digital, pero en este caso se enviará un mensaje entre las dos computadoras.

En los primeros ensayos que se realizaron no se obtuvo ningún resultado que aportara información alguna. Esto se debió a que como la velocidad de transmisión es muy alta no es posible visualizar el envío del mensaje a simple vista.

Como no se puede visualizar el mensaje a simple vista se recurre a una de las aplicaciones que tiene el programa que se han citado que consiste en la exportación de datos a Excel. Sin embargo, no toma los datos anteriores durante un cierto tiempo, sino que toma los datos a lo largo del tiempo que se observan en la ilustración 3.3. Al tomar solo los datos de dicha ventana, sigue siendo demasiado rápido el envío del mensaje y no se puede obtener ninguna conclusión.

Otra solución que se intentó está basada en variar el código original para alargar este mensaje para que así el mensaje no fuera tan corto y produjera una señal mayor. La modificación del código consiste en meter el mensaje dentro de un ciclo infinito, ocasionando así que el mensaje se envíe repetitivamente, es decir,

la idea se basa que en vez de mandar un mensaje se envíe un mensaje repetitivas veces.

Esta solución tampoco tuvo el resultado que se esperaba obtener, ya que solo se apreciaba el pulso que corresponde con la señal de verificación. Por lo consiguiente, hay que realizar otro tipo de pruebas para observar la señal cuando se está transmitiendo un mensaje.

Pese a que no se ha podido observar la señal cuando existe transmisión, con las pruebas anteriores se obtuvieron datos necesarios para los que se realizaban dichas pruebas. Ya que el fin de realizar estos ensayos se basaba en adquirir las características de la señal para poder así montar el sistema de transmisión mediante luz.

Recordando los datos obtenidos se puede finalizar con que la tensión no pasa en ningún momento los 5 voltios. Por tanto, el siguiente paso sería seleccionar un diodo emisor de luz y un fotodiodo de acuerdo con estas características que no se exponen en este documento.

# CONCLUSIONES.

El presente trabajo ha pretendido analizar si se pueden enviar mensajes mediante un medio acuático entre una base en la superficie y un submarino. En primer lugar, se creó un programa en lenguaje C++ capaz de transmitir datos basado en el protocolo TCP/IP de tipo cliente/servidor como se muestra en el capítulo uno.

Una vez creados los códigos, estos se utilizaron para transmitir mensajes entre dos computadores mediante un cable Ethernet y una vez comprobado que funcionaba de manera esperada se sustituyó el cable por la placa LX200V20 como se hace constancia en el capítulo dos.

En el último capítulo, se estudió la visualización de la señal cuando existe o no transmisión de datos para poder seleccionar el LED emisor y el fotodiodo receptor adecuados para poder transmitir los datos por medio de la luz.

Para finalizar, resulta necesario subrayar que las conclusiones obtenidas en este análisis nos llevan a la conclusión de que se cree posible transmitir datos entre un submarino y una base en la superficie utilizando ondas luminosas en vez de electromagnéticas.

# BIBLIOGRAFÍA.

## PAGINAS WEB CONSULTADAS:


Protocolos TCP/IP de Internet revista unam: http://www.ru.tic.unam.mx/bitstream/handle/123456789/791/220.pdf?sequence=1&isAllowed=y (Consultada el 15 de mayo de 2018).

Protocolo TCP/IP de Internet IEEE Xplore: https://ieeexplore.ieee.org/abstract/document/4062840/ (Consultada el 23 de mayo de 2018).

Introduction to Sockets Programming in C using TCP/IP: https://www.csd.uoc.gr/~hy556/material/tutorials/cs556-3rd-tutorial.pdf (Consultada el 31 de mayo de 2018).

Computer hoy: https://computerhoy.com/noticias/internet/que-es-comando-ping-como-funciona-42607 (Consultada el 11 de junio de 2018).

Ethernet cables: https://www.ertyu.org/steven_nikkel/ethernetcables.html (Consultada el 15 de junio de 2018).


## REFERENCIAS BIBLIOGRÁFICAS:


**Alania Agüero José Carlos, institución JCA: "NAVEGADOR WEB DE CODIGO ABIERTO".** Disponible en: http://www.bvs.hn/cu-2007/ponencias/TELEDUC/TELEDUC53.pdf (Consultada el 31 de mayo de 2018).

**Revista Ibérica de Sistemas e Tecnologias de Informacion**. Disponible en: http://www.scielo.mec.pt/scielo.php?pid=S1646-98952011000200004&script=sci_arttext&tlng=en (Consultada el 5 de junio de 2018).

**Programación de sockets, Xavier Perramon Tornil.** Disponible en: http://redes.cointeres.es/material/redes/M3.Programacion_de_Sockets.pdf (Consultada el 7 de junio de 2018).

**Hoja de características LX200V20.** Disponible en: https://www.bluerobotics.com/downloads/LX200V20-Datasheet-v1.2.pdf?x68454