FACOLTÀ DI INGEGNERIA

UNIVERSITÀ POLITECNICA DELLE MARCHE

Tirocinio

Study and development of a Submarine Optical Communication: TCP Protocol

Per accedere al titolo di

Laureato in
Ingegneria della tecnologia delle telecomunicazioni

Maria Jesús Pérez Saiz

Luglio
2018

SOMMARIO

Questo lavoro intende analizzare se i messaggi possono essere inviati attraverso un mezzo acquatico tra una base sulla superficie e un sottomarino. Uno dei problemi importanti a cui si deve cercare una soluzione è che le onde elettromagnetiche si attenuano considerevolmente attraverso l'acqua. È inteso per verificare se è possibile trasmettere dati per mezzo di onde luminose da un LED emittente e un fotodiodo del ricevitore. La prima cosa che verrà fatta è creare un programma basato sul protocollo TCP/IP che ci consenta di trasmettere dati e che si adatti a piastre specifiche che sono LX200V20 in modo che possano collegare il LED e il fotodiodo. Infine, verrà effettuato uno studio su come i dati vengono trasmessi attraverso il canale al fine di scegliere i componenti appropriati.

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.

INDICE

INDICE DELLE ILLUSTRAZIONI

INDICE DI GRAFICOS

INDICE DELLE TABELLE

INDICE DI SCHEMI

INTRODUZIONE.

Nell'ambiente di trasmissione wireless abbiamo un'eccezione dovuta alla forte attenuazione delle onde radio nell'acqua, poiché generalmente si basa su onde elettromagnetiche nella parte di spettro radio dello spettro, ma in questo caso non possono essere utilizzate. Pertanto, le comunicazioni sottomarine si basano sulla modulazione e sulla trasmissione delle onde acustiche. I modem acustici disponibili hanno una larghezza di banda disponibile di alcuni KHz e, di conseguenza, la loro velocità in bit.

I veicoli sottomarini si basano principalmente sulla registrazione di immagini ad alta risoluzione, video, suoni e altri dati e la trasmissione a un'unità centrale. Tuttavia, per questo tipo di dati sono richiesti un bitrate elevato e una trasmissione a bassa latenza.

Combinando questi due aspetti, l'alta trasmissione wireless e l'alta velocità, le comunicazioni wireless ottici sottomarine (UOWC) rappresentano un'alternativa valida. Grazie agli straordinari sviluppi dei diodi a emissione luminosa (LED) per l'illuminazione è stato possibile generare dispositivi compatti comunemente disponibili a basso costo, alta luminosità e relativa larghezza di banda di modulazione.

Poiché esiste una finestra a bassa attenuazione nell'acqua di mare nella regione visibile, UOWC può offrire velocità in bit molto più alte nell'ordine di Mbits / s, ma a distanze più brevi rispetto ai modem acustici.

In questo documento presentiamo il sistema che è stato realizzato nell'ambito del progetto SUNRISE EU-FP7, ed è corretto per l'integrazione nella rete di progetti sottomarini. I risultati mostrano che i modem OptoCOMM UOWC forniscono una trasmissione bidirezionale standard IEEE 10Base-T affidabile (10 Mbit / s Ethernet).

La sezione ottica dei modem è composta da due parti: la comunicazione costituita dal trasmettitore ottico (TX) e dal ricevitore (RX), che aiuta a stimare il livello di potenza della luce nell'RX. Il TX è formato da un set di LED a 7 chip con una lunghezza d'onda di emissione di 470 nm e una larghezza di banda ottica di 20 nm. D'altra parte, la RX include un singolo modulo Avalanche Photo-Diode (APD) con un'area attiva di 100 mm2, una larghezza di banda di 11 MHz e un amplificatore di transimpedenza integrato (TIA).

L'elettronica del modem include sensori ausiliari, alimentazione e una scheda microcontrollore per l'installazione di firmware/software e il controllo generale e la gestione del modem.

CAPITOLO 1. PROTOCOLLO TCP / IP

1.1. INTRODUZIONE AL PROTOCOLLO TCP / IP.

Il software sviluppato sulla scheda ha la funzione di gestire i comandi ricevuti da un utente e inviare file attraverso il livello di trasmissione ottica. L'utente comunica con il modem tramite il protocollo Ethernet TCP / IP per modificare le impostazioni del modem, recuperare informazioni sullo stato del modem, inviare e ricevere dati tramite comunicazioni ottiche.

Un protocollo è basato su un insieme di regole e procedure mediante le quali un server e un client condividono le informazioni tra loro. I protocolli TCP (Transmission Control Protocol) e IP (Internet Protocol) sono stati creati all'inizio del 1980 e sono stati adottati da ARPANET (Advanced Research Projects Network Network) nel 1983. Entrambi i protocolli insieme all'UDP (User Datagram Protocol) sono i più importanti e più usati, che differiscono nella metodologia di riconoscere i pacchetti di dati dalla destinazione e consentono al mittente di conoscere il suo arrivo. Nel caso di Internet, il suo successo è dovuto alla sua capacità di supportare servizi di trasferimento dati end-to-end affidabili per un gran numero di applicazioni che girano su un set di sistemi finali.

A metà degli anni Ottanta il protocollo TCP / IP è stato creato con l'obiettivo di avere un linguaggio comune a tutti i computer connessi a Internet indipendentemente dalle marche e dalle diverse tecnologie possedute da ciascuno, ciò è stato possibile grazie al unione di ARPANET, CSNET e MILNET.

Il modo di trasmettere i file nel protocollo TCP / IP si basa sulla divisione delle informazioni in piccoli pacchetti chiamati "datagrammi" o gruppi di dati inviati come se fossero un messaggio. La funzione di inviare pacchetti di informazioni da un sito a un altro è gestita dal protocollo IP, mentre il TCP è responsabile della loro divisione in pacchetti, sequenziandoli e aggiungendo informazioni per controllare se si verifica un errore.

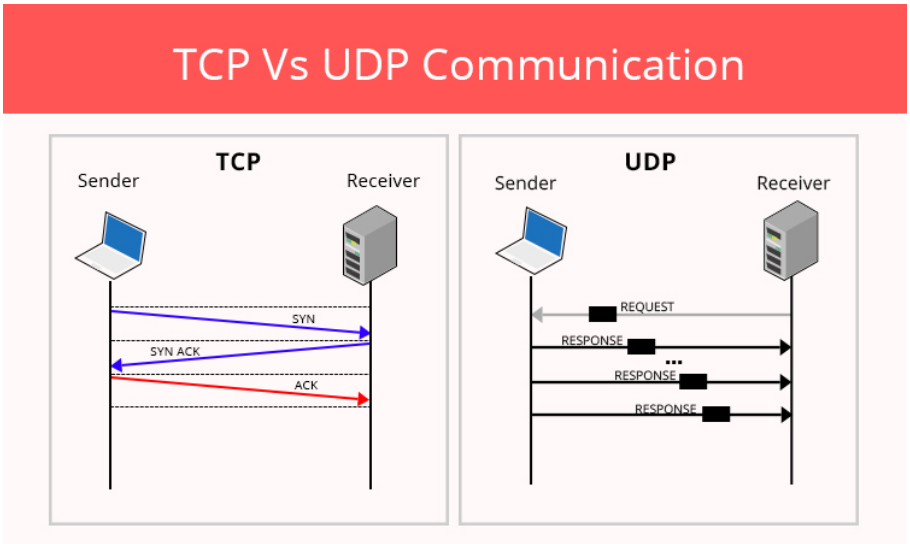
All'altro estremo, il protocollo TCP si occupa di ricevere i datagrammi, controllare gli errori e ordinarli nell'ordine in cui sono stati inviati. Quando i file inviati sono grandi, sono necessari solo pochi secondi, anche se i pacchetti di informazioni devono passare da una macchina all'altra finché non raggiungono il computer che li ha richiesti grazie al loro indirizzo IP.

I pacchetti di informazioni per raggiungere la loro destinazione trovano le guide lungo il percorso, chiamate router, che indicano le sezioni che devono viaggiare. I router sono computer di grandi dimensioni che gestiscono il traffico di informazioni sulla rete per l'utente che lo ha richiesto.

Il client deve connettersi al modem ottico, server, in due porte diverse (identificate come 1 e 2). Nella porta 1, l'utente invia i comandi e riceve la risposta corrispondente mentre si trova nella porta 2, l'utente riceve i dati da una trasmissione ottica. Per quanto riguarda il protocollo di comunicazione ottica, l'utente viene informato del risultato della comunicazione quando riceve pacchetti ACK e REC nella porta 2.

A differenza del TCP, UDP non è orientato alla connessione perché non stabilisce una connessione sicura e affidabile, d'altra parte non stabilisce una connessione prima di iniziare a trasmettere i dati. Nell'illustrazione 1.1 è possibile vedere uno schema comparativo di questi due protocolli.

Illustrazione 1.1. Confronto tra il protocollo TCP e UDP.



Fonte: Internet.

1.2. CODICE TCP/IP PER TRASMISSIONE DATI.

Il codice è stato implementato in linguaggio C con il programma 'Visual Studio 2017' per consentire la trasmissione di dati tra due computer tramite un cavo Ethernet seguendo il protocollo TCP / IP.

È molto importante sapere che TCP utilizza la connessione e non la porta del protocollo, che consente alla stessa porta di stabilire diverse connessioni, sapendo che una connessione completata è composta da due estremità (due porte ciascuna su un computer diverso dall'altra attraverso Internet).

In questo caso, verranno utilizzate le librerie chiamate socket che il produttore ci fornisce all'inizio del programma sia del server che dell'utente, mostrato nella figura 1.2. Verranno creati quattro codici diversi, uno per il server e il client di ciascun computer.

Illustrazione 1.2. Librerie per prese in Windows.

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

Fonte: propria elaborazione.

Un socket è un punto di comunicazione tra due macchine composte dall'indirizzo IP della macchina, che in molti casi si riferisce al nome del socket e al numero di porta utilizzato dal software TCP.

Esistono diverse classi di socket, da un lato, ci sono quelle che consentono la comunicazione tra sistemi diversi e, dall'altro, quelle che comunicano processi dello stesso sistema. Per differenziarli, vengono utilizzati nomi o indirizzi poiché ogni classe di socket corrisponde a un diverso spazio dei nomi. I due spazi dei nomi di base sono file e Internet. La differenza tra entrambi è che i file vengono utilizzati nei socket che comunicano i processi con lo stesso sistema e lo spazio dei nomi di Internet viene utilizzato nei socket che devono comunicare un processo con un altro di qualsiasi altro sistema.

Poiché stiamo usando il protocollo TCP, siamo interessati allo spazio dei nomi di Internet, che questo stesso spazio sarebbe utilizzato nel caso del protocollo UDP. L'indirizzo del socket in questo caso è costituito dall'indirizzo di rete del nodo, ovvero l'indirizzo della macchina corrispondente, un numero di porta per distinguere i socket di una macchina specifica e il protocollo specifico da utilizzare.

In questo caso i socket seguono il modello client / server in cui entrambi i computer hanno socket. La macchina chiamata client avvia la conversazione mentre la macchina corrispondente al server attende che il client avvii la conversazione e risponda.

Siamo interessati ad approfondire due varianti di prese Internet, socket datagramma e socket sequenza byte. La differenza tra entrambi è il protocollo di trasporto della trasmissione dati utilizzato. In questo caso, utilizzeremo quello corrispondente al protocollo TCP, che utilizzerà un socket di sequenza di byte, in

cui i dati vengono trasmessi da un estremo all'altro come un flusso o un flusso ordinato di byte.

Questo stile è anche noto come orientato alla connessione perché la comunicazione consiste nel stabilire in precedenza una connessione al socket remoto e quindi utilizzarlo per lo scambio di dati.

I passaggi che devono essere seguiti per stabilire una comunicazione tra due socket sono i seguenti:

Le operazioni che il server deve eseguire sono:

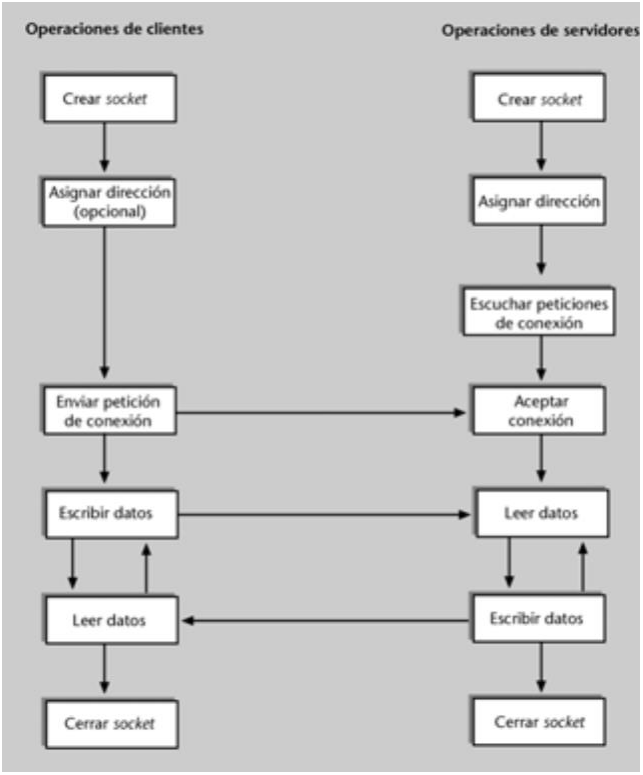
- Creazione di una presa.
- Assegna un indirizzo IP al socket.
- Lasciare la presa pronta per ricevere connessioni.
- Cerca attivamente di stabilire una connessione.
- Scambiare dati con il cliente che ha richiesto la comunicazione.
- Chiudere lo zoccolo rilasciando la connessione.

D'altra parte, le operazioni che il client deve seguire sono:

- Creazione di una presa.
- Assegna un indirizzo IP al socket.
- Invia una richiesta di connessione al server.
- Scambiare dati con il cliente che ha richiesto la comunicazione.
- Chiudere lo zoccolo rilasciando la connessione.

Nello schema 1.1. si osservano le operazioni corrispondenti ai client e ai server.

Schema 1.1. Diagramma delle operazioni corrispondenti ai client e ai server



Fonte: Internet

Comincerà con il server, la prima cosa da fare è la creazione e l'inizializzazione del socket una volta che le librerie dell' illustrazione 1.2 sono già state introdotte. Nell'illustrazione 1.3 si osserva come con la funzione *WSAStartup* viene generato un canale di comunicazione per l'inizializzazione del socket.

Il ciclo *if* mostrato nell'illustrazione corrisponde a un controllo degli errori in cui la variabile *wsOk* quando la variabile è diversa da zero scrive un messaggio sullo schermo in modo che l'utente sappia che il socket non viene inizializzato correttamente. Il valore di questa variabile è generato dalla funzione *WSAStartup*.

Illustrazione 1.3. Inizializzazione di un socket per la comunicazione.

```
 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
```

Fonte: propria elaborazione.

Una volta generato il canale di comunicazione, è necessario creare il socket. Poiché il protocollo TCP è orientato alla connessione, deve essere preparato a ricevere connessioni passivamente, per questo usiamo la chiamata *listening* come mostrato nella illustrazione 1.4. Lo scopo di utilizzare questa chiamata è creare una coda in cui verranno salvate le richieste che arrivano. Nel caso in cui non fosse possibile creare il socket, sullo schermo apparirà un messaggio poiché è stata introdotta una parte di controllo degli errori nelle righe da 24 a 28.

Inoltre, viene definito il tipo di socket utilizzato, che in questo caso è SOCK_STREAM, poiché il protocollo TCP viene utilizzato come accennato in precedenza, ma se il protocollo UDP fosse utilizzato, sarebbe di tipo SOCK_DGRAM. SOCK_STREAM rappresenta lo stile della sequenza di byte di comunicazione, cioè è orientato alla connessione, mentre SOCK_DGRAM corrisponde allo stile di comunicazione del datagramma.

Illustrazione 1.4. Creazione di 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      }
```

Fonte: propria elaborazione.

Quando il socket è stato creato, un indirizzo IP deve essere assegnato all'indirizzo del socket. A tale scopo, verrà utilizzato il suggerimento variabile *hint* di tipo *sockaddr_in* come mostrato nell'illustrazione 1.5. che è impostato su *INADDR_ ANY. INADDR_ANY* rappresenta un indirizzo IP indeterminato, questo è usato perché possono ricevere richieste di connessione destinate a qualsiasi indirizzo IP se il computer ne ha più di una.

D'altra parte, una porta deve essere assegnata al canale di comunicazione, come si può vedere nel canale 2001 per la connessione tra il server 1 e il client 2.

Illustrazione 1.5. Assegnazione dell'indirizzo 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
```

Fonte: propria elaborazione.

Successivamente, il socket deve essere preparato per ricevere connessioni come mostrato nella illustrazione 1.6. Poiché è stato applicato la chiamata *listen*, è necessario applicare la chiamata *accept* per stabilire le connessioni con i client che le richiedono. Tale chiamata consiste nell'estrarre la prima richiesta dalla coda e nella creazione di un nuovo socket che abbia lo stesso indirizzo dell'originale e sia connesso al creatore della richiesta.

Illustrazione 1.6. Preparare il socket per ricevere connessioni.

```
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);
```

Fonte: propria elaborazione.

Prima di tentare di stabilire una connessione con il client, viene creato un buffer di archiviazione in cui il messaggio ricevuto viene salvato come mostrato nella illustrazione 1.7. Tutto è all'interno di un ciclo che controlla ripetutamente se è stato ricevuto un messaggio con la funzione *recv*, che consente di leggere i dati e provare a stabilire una connessione con il client. Il primo parametro della funzione *recv* corrisponde al descrittore di socket, il parametro *buf* è un puntatore all'area di memoria in cui i dati vengono letti e il valore 4096 si riferisce al numero massimo di byte che vogliamo leggere. Quando si utilizza la funzione *recv* anziché *read* abbiamo un quarto parametro, *flags*, ma poiché in questo caso è 0, la funzione agisce esattamente come la funzione *read*.

I cicli visualizzati sono un semplice controllo degli errori per verificare se è ancora connesso al client corrispondente con le righe da 73 a 83 della illustrazione 1.7.

Illustrazione 1.7. Stabilire una connessione con il 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          }
```

Fonte: propria elaborazione.

Infine, con la funzione *send*, i dati vengono scritti. Nel parametro *buf*, l'indirizzo di memoria in cui devono essere scritti i dati viene memorizzato e nel terzo parametro viene indicato quanti byte devono essere scritti come mostrato nella illustrazione 1.8. Quando si usa la funzione *send* invece di *write*, abbiamo un quarto parametro, *flags*, ma dato che in questo caso è 0, la funzione agisce esattamente come la funzione *write*.

Illustrazione 1.8. Memorizzazione del messaggio ricevuto.

```
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
```

Fonte: propria elaborazione.

Per finire, ciò che devi fare è liberare la connessione chiudendo il socket come mostrato nella figura 1.9.

Illustrazione 1.9. Chiudi la presa.

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

Fonte: propria elaborazione.

È necessario implementare due codici che corrispondono al client e due al server come precedentemente menzionato, uno per ogni computer. La differenza tra il secondo codice server è nella parte di preparare il socket per ricevere connessioni, il resto del codice è esattamente lo stesso delle illustrazioni mostrate.

La parte che deve essere modificata è che corrisponde alla dichiarazione della porta che abbiamo sostituito la porta 2001 per la porta 2002 per la connessione tra il server 2 e il client 1, come mostrato nella illustrazione 1.10. sulla linea 34.

Illustrazione 1.10. Assegnazione di indirizzi IP del computer 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
```

Fonte: propria elaborazione.

Una volta terminati i due codici per il server, è necessario implementare i codici per il client. Comincerà con il client del computer e nello stesso modo in cui il codice del server userà i socket con le rispettive librerie. I passi che devono essere seguiti sono quelli descritti sopra che possono essere ricordati con lo schema 1.1.

Allo stesso modo del server, la prima cosa da fare è creare e inizializzare il socket, come mostrato nella illustrazione 1.11, usando la funzione *WSAStartup* per generare un canale di comunicazione. A differenza del codice server, vengono inseriti la variabile *ipAddress* contenente l'indirizzo IP del server e la

variabile *port* contenente la porta client del secondo computer che si collegherà al server del computer.

Illustrazione 1.11. Inizializzazione 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
```

Fonte: propria elaborazione.

Una volta creato il canale di comunicazione, il socket viene creato come mostrato nella illustrazione 1.12. Nel caso in cui si verifichi un errore, sullo schermo verrà visualizzato un messaggio che informa che non è stato possibile creare il socket.

Illustrazione 1.12. Creazione di prese.

```
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
```

Fonte: propria elaborazione.

Quando il socket è stato creato, un indirizzo IP deve essere assegnato all'indirizzo del socket. A tale scopo, verrà utilizzata una struttura chiamata *hint* suggerimento di tipo *sockaddr_in* come mostrato nella illustrazione 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.

Successivamente, è necessario inviare una richiesta di connessione al server. Questo viene fatto attraverso la funzione *connect* che viene utilizzata da un client per avviare una connessione in modo attivo e nel caso in cui non sia possibile impostare un avviso tramite un messaggio di errore come si può vedere nella illustrazione 1.14.

Il primo parametro che forma la funzione *conecct* corrisponde alla chiamata che segue i passi necessari per stabilire una connessione con il socket del server il cui indirizzo corrisponde al secondo parametro. Il terzo parametro si riferisce al numero di byte occupati da detto indirizzo. Questa chiamata non restituisce nulla finché il server non risponde.

Illustrazione 1.14. Invia una richiesta di connessione al server.

```
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
```

Fonte: propria elaborazione.

Una volta stabilita la connessione, il messaggio deve essere inviato, utilizzando un ciclo che entra in un ciclo in attesa di ricevere conferma, come mostrato nella illustrazione 1.15. Allo stesso modo del server, i dati vengono scritti con la funzione *send.*

Illustrazione 1.15. Scambia dati.

```
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);
```

Fonte: propria elaborazione.

Per completare il codice, è necessario chiudere il socket con la funzione *closesocket* per rilasciare la connessione come mostrato nella illustrazione 1.16.

Illustrazione 1.16. Chiudere la presa rilasciando la connessione.

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

Fonte: propria elaborazione.

In modo analogo, nel caso del server, è necessario creare un codice client per il secondo computer. Il codice sarebbe esattamente lo stesso con una singola differenza quando la definizione dell'IP del server è quella corrispondente con il computer 2 e la porta corrispondente alla porta client del primo computer che si connetterà al server con il secondo computer. È possibile visualizzare le modifiche nella illustrazione 1.17 sulle righe 12 e 15 rispettivamente.

Illustrazione 1.17. Assegnazione di indirizzi IP del computer 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
```

Fonte: propria elaborazione.

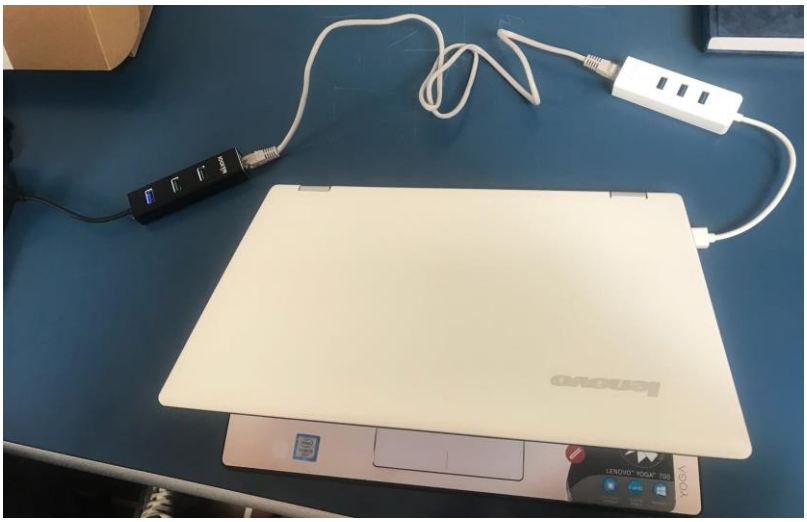
CAPITOLO 2. APPLICAZIONI.

2.1. TRASMISSIONE CAVO ETHERNET.

Quando viene verificato che i quattro codici creati, due per il server e due per il client, vengono compilati correttamente e non presentano errori, dobbiamo vedere le applicazioni che hanno questi codici. Innanzitutto, controllerà il suo funzionamento in modo semplice utilizzando un cavo Ethernet in cui verranno trasmessi i dati. Per questo, il cavo sarà collegato tra il computer uno e il computer due.

Come abbiamo visto nel primo capitolo, un indirizzo IP deve essere assegnato ai computer per effettuare la trasmissione dei dati, questo viene fatto manualmente. Una volta assegnati gli indirizzi IP corrispondenti, il cavo Ethernet deve essere collegato a entrambi i computer come mostrato nell' illustrazione 2.1.

Illustrazione 2.1. Connessione via cavo Ethernet tra i due computer.



Fonte: propria elaborazione.

Una volta collegato il cavo Ethernet, è necessario verificare lo stato della connessione dell'host locale e del computer remoto in una rete, per utilizzare il comando *ping* come mostrato nell' illustrazione 2.2. Gli indirizzi IP mostrati in questa illustrazione sono quelli usati nel primo test che corrispondono a 192.168.254.1 e 192.168.254.2. Alla fine, l'IP utilizzato è 192.168.254.1 e 192.168.254.3 come visto nel capitolo precedente.

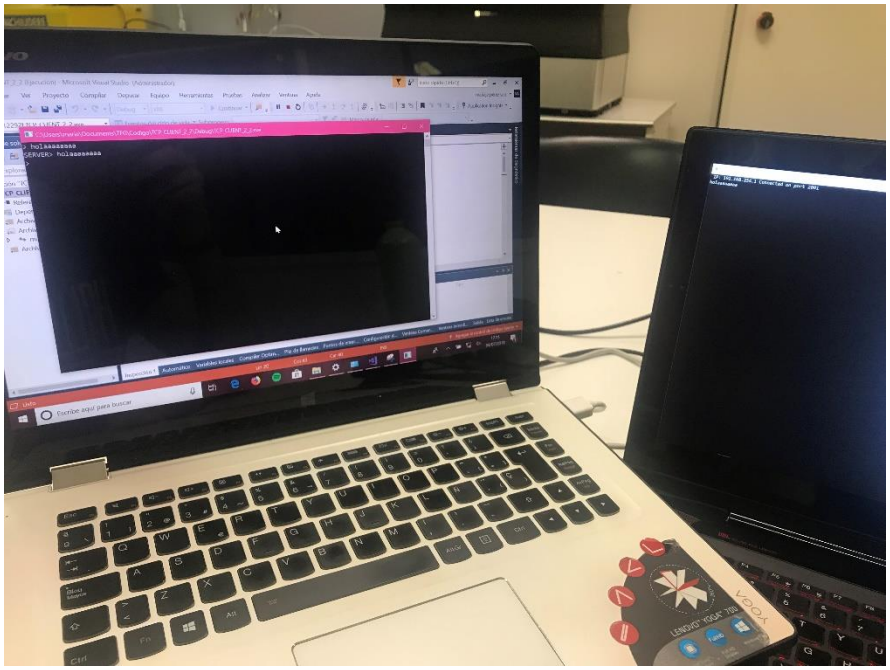
Illustrazione 2.2. Esecuzione del comando 'ping'.

```
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
```

Fonte: propria elaborazione.

Quindi, il programma viene eseguito come mostrato nell'illustrazione 2.3. in cui il computer a sinistra corrisponde al client del computer due, mentre quello a destra con quello del server.

Illustrazione 2.3. Test attraverso il cavo Ethernet.



Fonte: propria elaborazione.

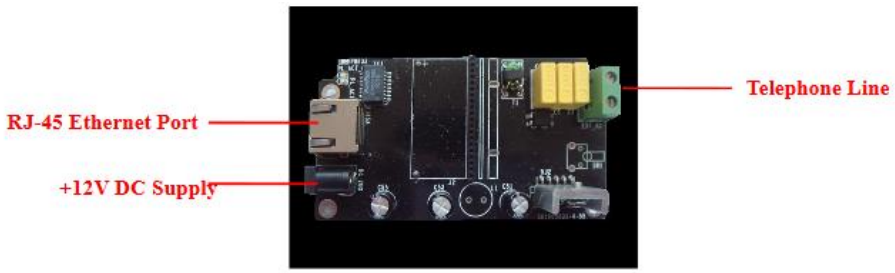
Nell'esecuzione si vede come il client invia un messaggio con il testo 'holoaaaaaaa', il server lo riceve e appare sullo schermo. Una volta ricevuto, il client vedrà lo stesso messaggio per sapere che è stato ricevuto dal server.

Con questo test si può concludere che il lavoro svolto dal client e dal server è corretto e che un messaggio è stato inviato con successo tra due computer tramite il cavo Ethernet.

2.2. TRASMISSIONE ATTRAVERSO LA PIASTRA LX200V20.

Un altro modo per verificarne il funzionamento è una piastra, ma prima dovresti studiarne il funzionamento. La piastra da utilizzare è composta principalmente da tre parti come mostrato nell'illustrazione 2.4. Si osserva che una parte corrisponde all'alimentazione sia per mezzo di un ingresso USB o da una sorgente; le porte di input Ethernet; e infine la parte corrispondente al PowerLine che è responsabile della trasmissione del segnale.

Illustrazione 2.4. Parti rilevanti della piastra.



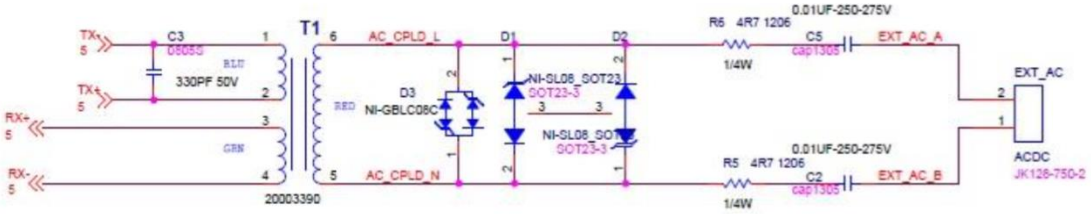
Fonte: Datasheet della piastra.

La parte corrispondente alla PowerLine deve essere studiata con profondità perché per eseguire la comunicazione sottomarina tramite un LED, i due cavi che collegano le piastre devono essere sostituiti da un LED e un fotodiodo. Non dovrebbe avere alcun effetto al momento della trasmissione dei dati con la differenza che il segnale viaggia alla luce invece che via cavo.

Appaiono diversi problemi, il primo problema che si osserva è l'uso di due cavi per le due piastre, quando sono necessari quattro cavi di uscita di ogni piastra per collegare il LED e il fotodiodo. Per risolvere questo, la piastra dovrebbe essere studiata con l'aiuto del datasheet per vedere come converte due cavi di trasmissione (TX) e due cavi di ricezione (RX) in una trasmissione e una ricezione.

Uno dei circuiti osservati nella scheda tecnica è mostrato nell'illustrazione 2.5. che corrisponde alla parte di trasmissione e ricezione della piastra. Analizzando il circuito si vede che per mezzo di un trasformatore unifica i cavi di trasmissione e ricezione.

Illustrazione 2.5. Circuito TX / RX.

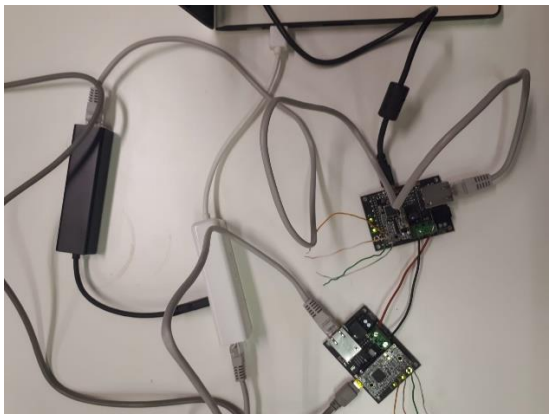


Fonte: Datasheet della piastra LX200V20.

Per verificare che la piastra funzioni correttamente, verrà effettuata una prima prova con la trasmissione delle lastre mediante due cavi. Per fare ciò, due cavi saranno collegati direttamente alla porta di uscita della scheda e questi cavi saranno collegati alla porta di ingresso della seconda scheda.

Nell'illustrazione 2.6. si osserva come è stata realizzata la connessione tra le schede e i computer. Puoi vedere come oltre ai cavi che collegano le piastre ci sono altri due cavi collegati a ciascuna piastra. Questi cavi sono quelli corrispondenti con le prese di corrente che vanno direttamente al computer attraverso un cavo USB.

Illustrazione 2.6. Collegamento delle due piastre usando la configurazione convenzionale.



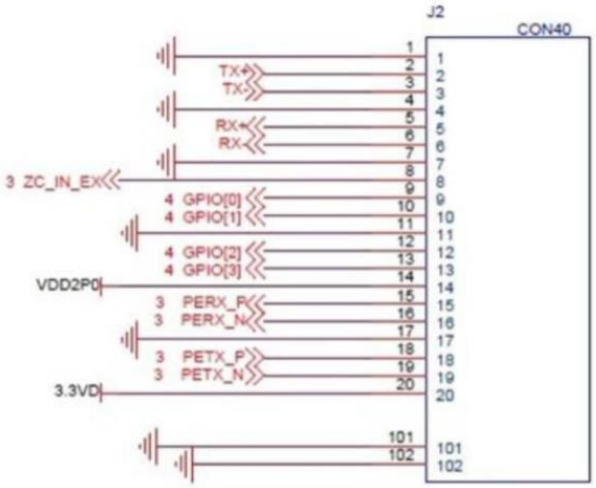
Fonte: propria elaborazione.

Sono stati inviati diversi messaggi come è stato fatto con il cavo Ethernet ed è stato verificato che i messaggi sono stati ricevuti correttamente. Quando si invia un messaggio tra un computer e un altro, non si devono osservare modifiche. Sebbene l'unica differenza che può essere data è un ritardo rispetto alla trasmissione diretta attraverso un cavo Ethernet. Questo ritardo che viene pronunciato non può essere visto a prima vista perché è molto piccolo.

Analizzando i test che sono stati effettuati, si può concludere che le lastre funzionano come previsto e consentono di modificare i dati in modo che possano essere trasmessi attraverso i cavi convenzionali.

Raggiunto questo punto, è possibile ottenere due cavi di trasmissione e due cavi di ricezione. A tale scopo, le schede tecniche vengono nuovamente studiate, ma in questo caso l'attenzione è rivolta ai pin come mostrato nell'illustrazione 2.7.

Illustrazione 2.7. Perni della scheda LX200V20.



Fonte: Datasheet della piastra LX200V20.

È possibile controllare nell' illustrazione 2.7. che gli ingressi TX e RX compaiano rispettivamente come porte di ingresso e di uscita della scheda. Le porte che corrispondono alla trasmissione sono 2 e 3, mentre quelle che fanno riferimento alla ricezione sono le porte 5 e 6.

Pertanto, quando sono rappresentati come porte di ingresso / uscita, è possibile saldare direttamente i cavi a dette porte della scheda mediante contatti metallici come mostrato nell'illustrazione 2.8.

Illustrazione 2.8. Collegamento di due cavi TX e due cavi RX.

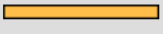
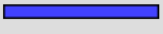
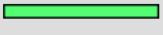
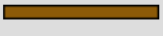
Fonte: propria elaborazione.

Nell'illustrazione 2.8. viene osservata la piastra LX200V20 montata sulla piastra generale. Sul lato sinistro della piastra LX200V20 corrisponde alle porte della scheda a cui sono saldati i quattro cavi. Inoltre, è possibile vedere il cavo collegato alla porta USB che fornisce alimentazione alla scheda.

Puoi vedere che i fili saldati sono di un colore: arancione, arancione e bianco, verde, verde e bianco. Questi colori sono stati scelti per modificare la configurazione iniziale senza piastre il meno possibile. Questo perché questi cavi rappresentano l'utilità dei cavi all'interno del cavo Ethernet generale.

I cavi interni di un cavo Ethernet sono rappresentati nella tabella 2.1. in cui è possibile vedere una definizione di ciascuno dei cavi in base alle loro caratteristiche.

Tabella 2.1. Classificazione dei cavi interni di un cavo 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-

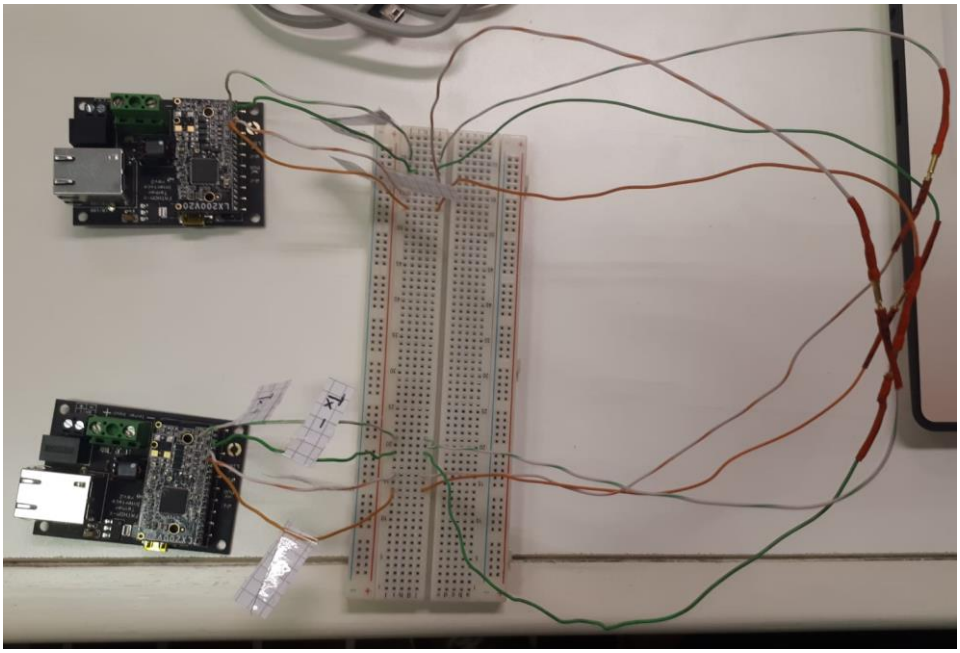
Fonte: Internet.

Una volta saldati i cavi, è necessario verificare che funzioni come prima. Per le due verifiche precedenti, sono stati creati canali di trasmissione dati testati che dovrebbero funzionare. D'altra parte, in questo caso, quando i quattro cavi sono collegati direttamente alla piastra, il funzionamento delle piastre viene modificato normalmente.

Poiché il normale funzionamento è stato modificato, è importante controllare attentamente il funzionamento del canale al momento della trasmissione dei dati. In questa configurazione non è necessario utilizzare la piastra generale, sarebbe sufficiente con la piastra LX200V20 con l'uso delle porte per i cavi convenzionali. Ma per praticità, non è stato deciso di rinunciare alla targhetta generale poiché ha una parte che si riferisce alla potenza attraverso il cavo USB e la porta ai cavi Ethernet che è molto confortevole.

Nelle estremità finali di ciascuno dei quattro cavi estratti dall' illustrazione 2.8, i connettori maschio e femmina sono stati accoppiati, al fine di minimizzare le perdite al momento della connessione. Questi connettori avrebbero potuto essere dispensati, ma per i test futuri saranno più efficaci. La configurazione finale è mostrata in illustrazione 2.9.

Illustrazione 2.9. Collegamento alla scheda tramite quattro cavi.



Fonte: propria elaborazione.

Una volta pronta la configurazione, è necessario verificare che funzioni analogamente nei casi precedenti. Allo stesso modo, è stato verificato che il messaggio è stato inviato e ricevuto con successo attraverso il canale.

Un effetto causato dall'estrazione dei quattro cavi della piastra LX200V20 è che è possibile collegare un LED e un fotodiodo alle estremità dei cavi per trasmettere il messaggio attraverso la luce.

CAPITOLO 3. STUDIO DEL SEGNALE

3.1. VISUALIZZAZIONE DEL SEGNALE.

Fino a questo punto è stato possibile preparare il canale di trasmissione dati con una configurazione che consente il collegamento tra il diodo emettitore e il fotodiodo ricevente che funziona correttamente. Se è ricapitolato, l'obiettivo di questo lavoro si basa sulla preparazione di un canale di comunicazione dati sottomarino che funziona utilizzando la trasmissione di dati attraverso un segnale luminoso nel mezzo dell'acqua. Sebbene al momento, i cavi attraverso i quali il segnale viaggia siano direttamente collegati.

Il passo successivo è quello di collegare il diodo e il fotodiodo direttamente ai cavi del canale, anche se prima deve essere studiato il segnale che viaggia attraverso il canale. Questo è molto importante poiché, a seconda delle caratteristiche del segnale ottenuto, è necessario introdurre un diodo e un fotodiodo con alcune caratteristiche o altri. Nel caso di saltare lo studio del segnale e quindi mettere un diodo e un fotodiodo con determinate caratteristiche può portare a non funzionare bene o addirittura danneggiare i diodi e i fotodiodi che sono stati utilizzati.

Nello studio del segnale, il parametro più importante che deve essere preso in considerazione è il riferimento ai livelli di tensione che il segnale può raggiungere nella modalità di trasmissione. Per il suo studio, verrà utilizzato un oscilloscopio digitale per determinare i livelli di tensione raggiunti dal segnale sia quando i dati vengono trasmessi sia quando non lo sono.

Virtual Bench of National Instruments è l'oscilloscopio digitale che verrà utilizzato per eseguire questo studio. L'oscilloscopio citato deve essere collegato a un computer in cui è possibile visualizzare le misure, poiché non ha il proprio schermo. Anche se a vostro vantaggio, l'oscilloscopio ha una piattaforma per computer LabView che ha un programma installato che consente di osservare i segnali dell'oscilloscopio.

Una caratteristica che è vantaggiosa per questo programma è che include un'opzione che congela il segnale in un momento specifico per essere in grado di osservare il segnale in modo più dettagliato perché altrimenti il segnale sarebbe in costante movimento. Allo stesso modo, ha alcuni indicatori per osservare i livelli di tensione e ha la possibilità di estrarre i dati raccolti lungo l'intero asse temporale in un file Excel per studiare in maggiore profondità i dati e rappresentarli in un chart.

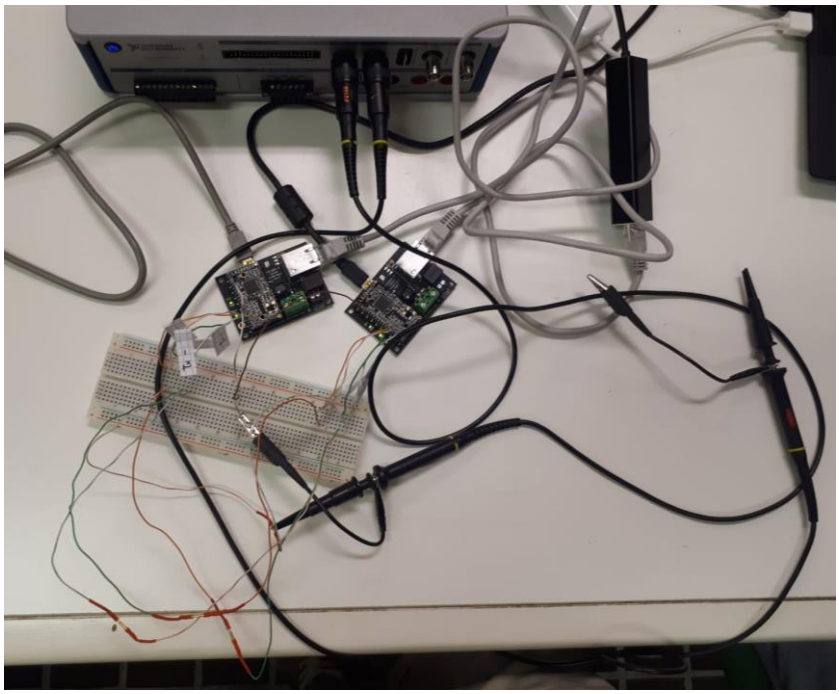
3.2. VISUALIZZAZIONE DEL SEGNALE IN CASO DI NESSUN TRASMISSIONE.

Come visto in precedenza nella configurazione dei pin nell'illustrazione 2.7 le porte delle schede sono rappresentate come TX^+ e come TX^-, per le porte di trasmissione e RX^+ e RX^- per le porte di ricezione. Vedendo questo le connessioni devono essere fatte seguendo la regola che la porta TX^+ di una delle schede è collegata alla porta RX^+ dell'altra scheda. Questo è verificato con i giunti mostrati nell'illustrazione 2.9. E nella tabella 2.1.

È fondamentale fare una chiara differenza tra TX^+ e TX^- poiché il TX^+ trasporterà il segnale mentre il TX^- può essere preso come terra. Come nel caso della trasmissione, anche questa differenza è altrettanto importante con RX^+ e RX^-. L'esistenza di due cavi è dovuta alla differenza di tensione che esiste tra di loro, che deve essere 0 o 1 in binario durante la trasmissione dei dati.

Come per TX^+ del client, il segnale viaggia verso questa porta dove la sonda dell'oscilloscopio deve essere collegata per visualizzare il segnale e il filo di terra della sonda su TX^- del client. Poiché la sonda è connessa al client, è necessario sapere chiaramente quale è il computer che funge da client e quale server. Nell'illustrazione 3.1. si osserva come la sonda sia collegata alla configurazione mostrata in precedenza nell'illustrazione 2.9.

Illustrazione 3.1. Collegamento della sonda con l'oscilloscopio e la piastra.



Fonte: propria elaborazione.

Come si vede nell'illustrazione 3.1. la sonda è collegata al cavo bianco e arancione che corrisponde al cavo di trasmissione del client (TX^+), con la piastra a sinistra collegata al computer che funge da client. D'altra parte, la terra della sonda è collegata al filo arancione corrispondente a TX^- della stessa piastra.

Una volta eseguita questa configurazione, il programma viene eseguito e il segnale viene osservato attraverso il canale senza inviare alcun messaggio. Quello che ci si aspetta è vedere un segnale piatto, ma in caso contrario, si osserverà quanto mostrato nell'illustrazione 3.2.

Illustrazione 3.2. Osservazione del segnale senza trasmissione.

Fonte: oscilloscopio digitale.

Sebbene sia contemplato che per la maggior parte del tempo il segnale è piatto come previsto quando non trasmette nulla, ogni tanto appare un segnale per un breve periodo di tempo, come si può vedere nell'illustrazione 3.2. Come è stato detto in precedenza, una delle applicazioni che avevano la piattaforma LabView è il programma che consente la sua visualizzazione che ci permette di congelare il segnale con la funzione di arresto come può essere visto poiché altrimenti il segnale sarebbe in continuo movimento.

Una delle finestre che vediamo in questa illustrazione corrisponde al segnale per tutto il tempo che l'oscilloscopio sta misurando, questo è mostrato nella illustrazione 3.3.

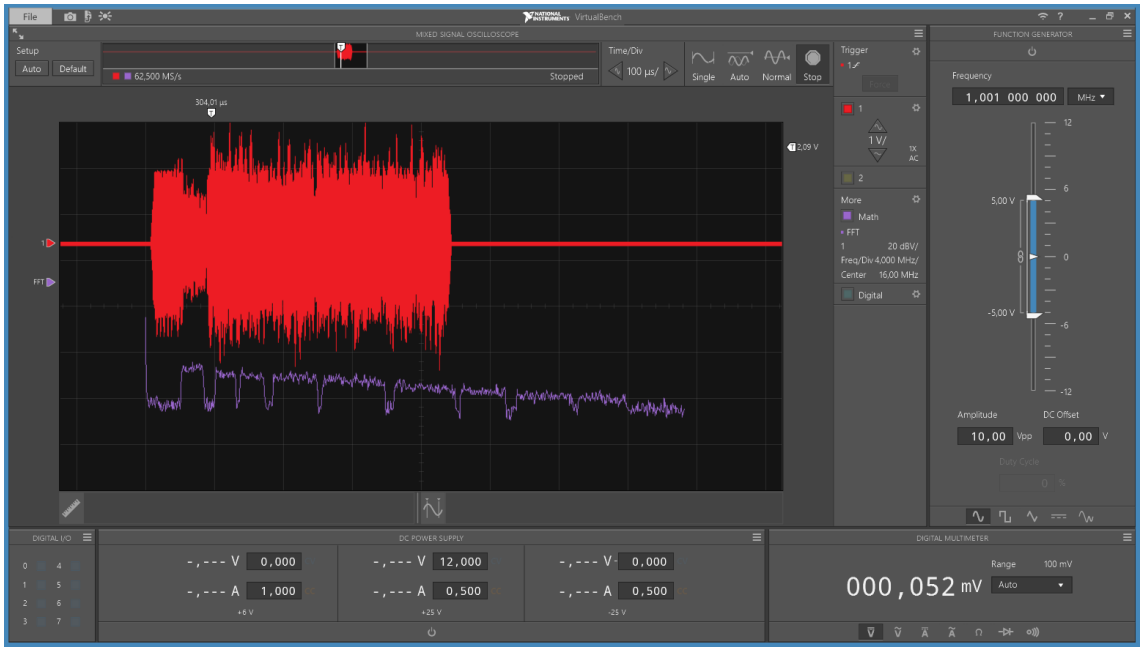
Illustrazione 3.3. Finestra del grafico completo dell'oscilloscopio.

Fonte: oscilloscopio digitale.

Nell’illustrazione 3.3 mostra una parte evidenziata che corrisponde all'ingrandimento mostrato nell'illustrazione 3.2. Con questa illustrazione, si riflette come il polso si ripete nel tempo e con lo stesso spazio temporale tra ogni impulso.

Quando appare questo segnale inaspettato, dobbiamo studiare il suo segnale e scoprire la sua utilità. Per fare ciò, è stato fatto uno studio di frequenza come mostrato nell'illustrazione 3.4. L'applicazione con cui il programma ha permesso allo stesso segnale di osservare la sua trasformata di Fourier e quindi facilitare lo studio.

Illustrazione 3.4. Segnale senza trasmissione nel dominio della frequenza.



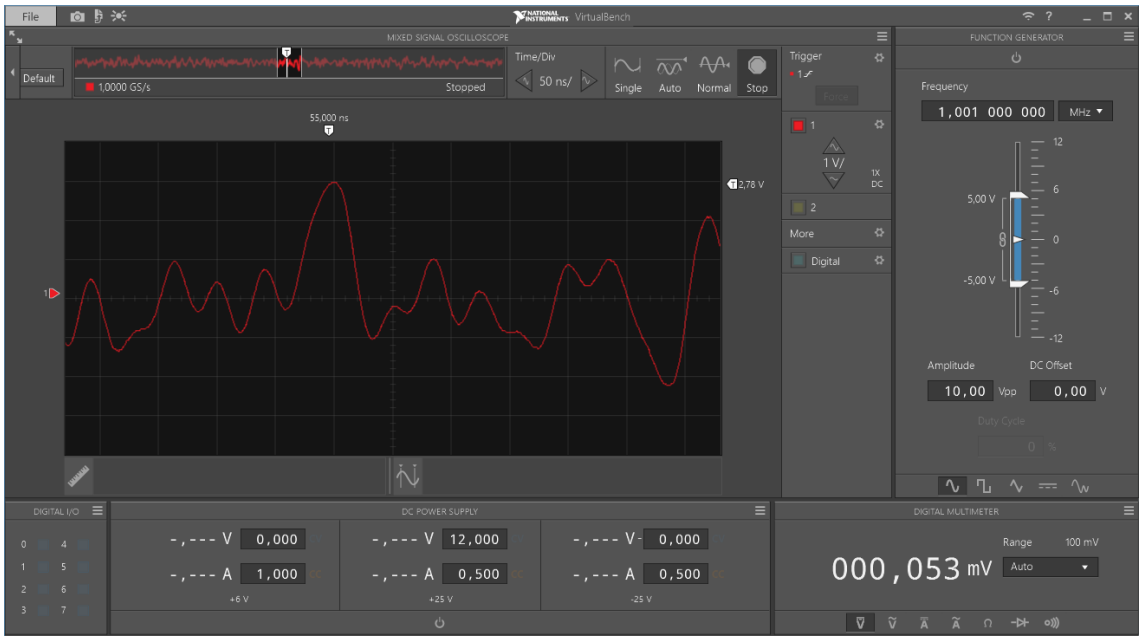
Fonte: oscilloscopio digitale.

Esaminare la trasformata di Fourier ottenuta nell'illustrazione 3.4. di un colore violetto, non è stata raggiunta una chiara conclusione su quale fosse la sua funzione e che cosa rappresentasse. Poiché la sua rappresentazione del tempo non è qualcosa di concreto, una conclusione non può essere ottenuta, ma si

osserva come ci siano diverse bande di frequenza la cui ampiezza diminuisce all'aumentare della frequenza.

Poiché non è stato ottenuto nulla dall'analisi della frequenza, il segnale viene aumentato mediante lo strumento "*zoom*" nel dominio del tempo. Il risultato ottenuto può essere esaminato nell'illustrazione 3.5. dove la spaziatura dell'asse temporale corrisponde a 50 ns.

Illustrazione 3.5. Segnale senza trasmissione estesa nel dominio del tempo.

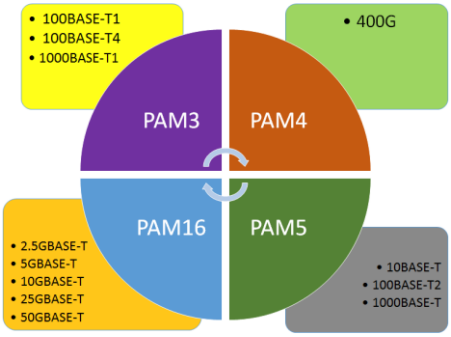


Fonte: oscilloscopio digitale.

Guardando da vicino il risultato ottenuto nell'illustrazione 3.5, non si vede uno schema che segue ogni tanto. Pur eseguendo numerosi test in diversi periodi di tempo, è possibile trarre una conclusione.

La conclusione che è stata raggiunta è che la trasmissione attraverso il canale avviene attraverso una modulazione di tipo PAM5. La modulazione PAM5 consiste in una modulazione di ampiezza a cinque livelli di tensione. Pertanto, questo tipo di modulazione si adatta alle aspettative poiché l'uso delle piastre e del cavo Ethernet è di tipo 10BASE-T. Successivamente, nel grafico 3.1. un confronto può essere fatto con il resto delle modulazioni in ampiezza.

Grafico 3.1. Modulazioni PAM

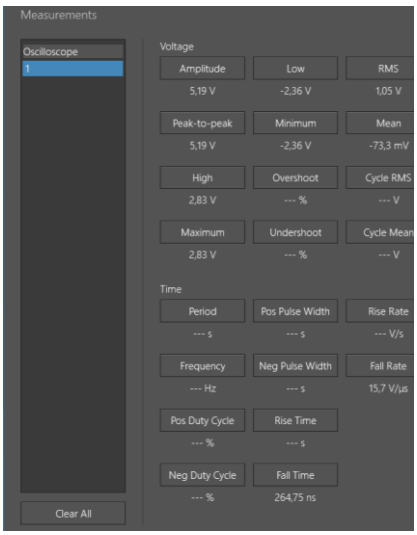


Fonte: Internet.

Con la consapevolezza che si sta verificando una modulazione di ampiezza, il risultato ottenuto nell'illustrazione 3.5 viene nuovamente studiato. Nella parte centrale del segnale appare un picco di tensione massima, mentre il minimo è visto all'estrema destra del grafico. Questi picchi sporgenti corrispondono ai due estremi livelli di modulazione. Il resto dei picchi si riferisce ai livelli intermedi e, infine, il quinto corrisponde al livello a zero volt.

Per vedere i livelli di tensione corrispondenti a questi livelli di modulazione, viene utilizzata nuovamente una delle applicazioni del programma. L'applicazione che verrà utilizzata in questo caso è quella mostrata in figura 3.6 che consente di visualizzare dati numerici in riferimento a una tensione picco-picco, massimo, minimo, ecc.

Illustrazione 3.6. Applicazione dell'oscilloscopio digitale che mostra i dati numerici del segnale senza trasmissione.



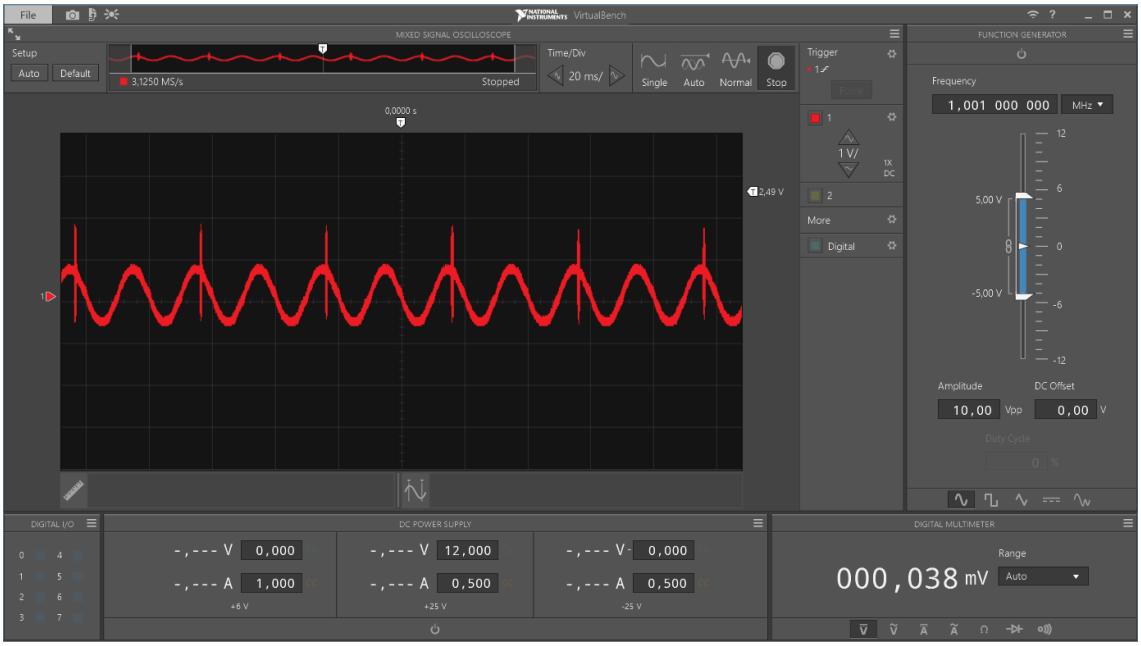
Fonte: oscilloscopio digitale.

Poiché la media del segnale ha un valore vicino a zero volt, mostra che ha uno dei suoi componenti uguale a zero. D'altra parte, se analizziamo il valore massimo e minimo, vediamo che si trovano rispettivamente a circa 3V e -3V, sebbene il valore minimo sia leggermente più lontano da -3 volt.

La divisione del grafico corrisponde a 1 volt, quindi il segnale viene trasmesso attraverso il cavo Ethernet utilizzando una modulazione a cinque livelli di ampiezza che corrispondono a: +3, +1, 0, -1 e -3 volt.

Uno degli errori frequenti che si verificano di solito non è quello di collegare la sonda a terra, il risultato dell'analisi è mostrato nell'illustrazione 3.7.

Illustrazione 3.7. Segnale senza trasmissione nel caso in cui la sonda non sia collegata a terra.



Fonte: oscilloscopio digitale.

Sebbene la sonda non sia collegata a terra, il segnale periodico è ancora apprezzato ogni tanto corrispondente ai picchi mostrati nell'illustrazione 3.7. Ma in questo caso non si ottiene un'onda piatta quando non c'è trasmissione, ma appare un segnale sinusoidale.

Questo è stato uno dei problemi che si sono verificati durante l'esecuzione di diversi test, che osservando che il risultato era totalmente diverso da quello che si è visto prima, la causa è stata studiata. Quindi è stato retratto al punto della configurazione in cui è stato osservato che la massa della sonda è stata scollegata e in sua assenza non era collegata alla porta TX^{-}.

Come mostrato nell'illustrazione 3.7. il segnale di riferimento è perso, il che fa sì che il segnale smetta di essere piatto e inizi ad avere un valore. Questo perché il segnale di riferimento causa un segnale nullo quando non c'è trasmissione. Da questo fallimento si può ottenere qualcosa di positivo poiché grazie a ciò è stato possibile osservare il vettore del segnale, cioè il segnale attraverso il quale i dati saranno trasmessi, che è il segnale sinusoidale che può essere visto.

Sebbene sia stato risolto con quello che corrispondeva all'impulso ottenuto quando non venivano trasmessi dati, il problema iniziale del perché non solo un'onda piatta appare non è risolto.

Per cercare di risolvere la domanda sul perché non appaia solo un'onda piatta, abbiamo studiato il protocollo utilizzato (TCP / IP) e il modo in cui i dati vengono trasmessi tramite cavi Ethernet ed è stato possibile concludere che il segnale periodico corrisponde a un segnale proveniente da verifica automatica Ciò significa che fino a quando i dati non vengono inviati, il protocollo invia un segnale molto breve per notificare al server che il client è ancora connesso.

3.3. VISUALIZZAZIONE DEL SEGNALE IN CASO DI TRASMISSIONE.

Una volta che il segnale è stato visualizzato nel caso in cui non ci sia trasmissione, il passo successivo è avere la trasmissione e analizzare il segnale. Allo stesso modo in cui senza collegamento il segnale verrà analizzato usando l'oscilloscopio digitale, ma in questo caso verrà inviato un messaggio tra i due computer.

Nelle prime prove effettuate, non sono stati ottenuti risultati che fornissero informazioni. Ciò è dovuto al fatto che poiché la velocità di trasmissione è molto elevata, non è possibile visualizzare l'invio del messaggio a occhio nudo.

Poiché il messaggio non può essere visualizzato a occhio nudo, una delle applicazioni in cui è stato citato il programma viene utilizzata per esportare i dati in Excel. Tuttavia, non prende i dati di cui sopra per un certo tempo, ma prende i dati nel tempo come mostrato nell’illustrazione 3.3. Prendendo solo i dati da quella finestra, l'invio del messaggio è ancora troppo veloce e non è possibile ottenere alcuna conclusione.

Un'altra soluzione che è stata provata si basa sulla variazione del codice originale per allungare questo messaggio in modo che il messaggio non sia così breve e produca un segnale più grande. La modifica del codice consiste nel mettere il messaggio in un ciclo infinito, causando l'invio ripetitivo del messaggio, cioè l'idea si basa sul fatto che invece di inviare un messaggio, un messaggio viene inviato ripetutamente.

Anche questa soluzione non aveva il risultato atteso, poiché solo l'impulso corrispondente al segnale di verifica era apprezzato. Pertanto, è necessario eseguire un altro tipo di test per osservare il segnale quando viene trasmesso un messaggio.

Sebbene non sia stato possibile osservare il segnale durante la trasmissione, con i test precedenti sono stati ottenuti i dati necessari per tali test. Poiché lo scopo di condurre questi test era basato sull'acquisizione delle caratteristiche del segnale per poter montare il sistema di trasmissione con la luce.

Ricordando i dati ottenuti si può concludere che la tensione non passa in qualsiasi momento 5 volt. Pertanto, il prossimo passo sarebbe selezionare un diodo ad emissione luminosa e un fotodiodo secondo queste caratteristiche che non sono discusse in questo documento.

CONCLUSIONI.

Il presente lavoro ha cercato di analizzare se i messaggi possono essere inviati per mezzo di un mezzo acquatico tra una base nella superficie e un sottomarino. Innanzitutto, è stato creato un programma in linguaggio C ++ in grado di trasmettere dati basati sul protocollo TCP / IP di tipo client / server come mostrato nel capitolo uno.

Una volta che i codici sono stati creati, sono stati utilizzati per trasmettere messaggi tra due computer utilizzando un cavo Ethernet e, una volta verificato che funzionava in modo prevedibile, il cavo è stato sostituito dalla scheda LX200V20 come mostrato nel capitolo due.

Nell'ultimo capitolo, la visualizzazione del segnale è stata studiata quando c'è o meno la trasmissione di dati per poter selezionare il LED di emissione appropriato e il fotodiodo ricevente per essere in grado di trasmettere i dati per mezzo della luce.

Infine, è necessario sottolineare che le conclusioni ottenute in questa analisi ci portano alla conclusione che si ritiene possibile trasmettere dati tra un sottomarino e una base sulla superficie utilizzando onde luminose anziché onde elettromagnetiche.

BIBLIOGRAFIA.

PAGINE WEB CONSULTE:

Protocolli TCP / IP della rivista Internet unam:http://www.ru.tic.unam.mx/bitstream/handle/123456789/791/220.pdf?sequence=1&isAllowed=y (Accessibile il 15 maggio 2018).

Protocollo TCP/IP de Internet IEEE Xplore: https://ieeexplore.ieee.org/abstract/document/4062840/ (Accessibile il 23 maggio 2018).

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

Computer hoy: https://computerhoy.com/noticias/internet/que-es-comando-ping-como-funciona-42607 (Accessibile il 11 giugno 2018).

Ethernet cables: https://www.ertyu.org/steven_nikkel/ethernetcables.html (Accessibile il 15 giugno 2018).

RIFERIMENTI BIBLIOGRAFICI:

Alania Agüero José Carlos, institución JCA: "NAVEGADOR WEB DE CODIGO ABIERTO". Disponibile in: http://www.bvs.hn/cu-2007/ponencias/TELEDUC/TELEDUC53.pdf (Accessibile il 31 maggio 2018).

Revista Ibérica de Sistemas e Tecnologias de Informacion. Disponibile in: http://www.scielo.mec.pt/scielo.php?pid=S1646-98952011000200004&script=sci_arttext&tlng=en (Accessibile il 5 giugno 2018).

Programación de sockets, Xavier Perramon Tornil. Disponibile in: http://redes.cointeres.es/material/redes/M3.Programacion_de_Sockets.pdf (Accessibile il 7 giugno 2018).

Hoja de características LX200V20. Disponibile in: https://www.bluerobotics.com/downloads/LX200V20-Datasheet-v1.2.pdf?x68454 (Accessibile il 15 giugno 2018).