



**UNIVERSITA' DEGLI STUDI DI UDINE**

**DIPARTIMENTO POLITECNICO INGEGNERIA E ARCHITETTURA**

Corso di laurea triennale in

Ingegneria elettronica

**Tesi di Laurea**

Visualizzazione traiettorie 3D in tempo reale da accelerometro su dispositivo IoT

**Relatore:**

Prof. Pier Luca Montessoro

**Laureando:**

Lorenzo Brunato

---

ANNO ACCADEMICO 2018/2019

## **INDICE:**

<b>INTRODUZIONE .....</b>	<b>pag. 3</b>
 <b>CAPITOLO 1: Conoscenze preliminari e strumenti .....</b>	<b>pag. 4</b>
 <b>CAPITOLO 2: Sistema client-server .....</b>	<b>pag. 5</b>
2.1 Client-Server .....	pag. 5
2.2 Orange Pi PC 2 e accelerometro LIS3DH .....	pag. 6
2.3 La libreria multiplatforma QT .....	pag. 9
 <b>CAPITOLO 3: Software .....</b>	<b>pag. 11</b>
3.1 L'architettura del software .....	pag. 11
3.2 Main.cpp .....	pag. 12
3.3 Myserver.h .....	pag. 14
3.4 Myserver.cpp .....	pag. 16
3.5 Mygrafic.h .....	pag. 19
3.6 Mygrafic.cpp .....	pag. 21
 <b>CAPITOLO 4: Applicazione .....</b>	<b>pag. 27</b>
4.1 Collegamento client – server .....	pag. 27
4.2 Compilazione del programma .....	pag. 32
4.3 Esecuzione del programma .....	pag. 35
 <b>CAPITOLO 5: Conclusioni e applicazioni future .....</b>	<b>pag. 38</b>
 <b>APPENDICE A – Orange Pi Pc 2 .....</b>	<b>pag. 41</b>
<b>APPENDICE B – Accelerometro LIS3DH .....</b>	<b>pag. 43</b>
<b>APPENDICE C – Codice completo .....</b>	<b>pag. 44</b>
 <b>BIBLIOGRAFIA E SITOGRAFIA .....</b>	<b>pag. 57</b>
 <b>RINGRAZIAMENTI .....</b>	<b>pag. 58</b>

## INTRODUZIONE

La manipolazione dei dati in tempo reale rappresenta uno degli aspetti più importanti della robotica, poiché consente di ricevere degli input di controllo sullo stato di una macchina e di elaborarli per analizzarli o operare delle correzioni in un breve lasso di tempo.

Nella tesi si sono elaborati dati in tempo reale provenienti da un accelerometro.

L'accelerometro è un dispositivo molto diffuso nell'ambito della robotica e dell'automazione, poiché i suoi valori permettono di ricavare velocità e posizione con semplici formule di natura fisica.

Conoscere la posizione permette di controllare i movimenti della macchina con una elevata precisione. Il solo accelerometro però non è sufficiente a tale scopo, infatti esso viene sempre accoppiato con un giroscopio e un magnetometro.

Lo scopo di questa tesi sperimentale è quello di sviluppare un software che elabori i dati delle accelerazioni rilevati dall'accelerometro per ricavarne la posizione e plottare tali valori in un grafico tridimensionale in tempo reale, col quale è possibile interagire.

Nei successivi capitoli di questa tesi andremo a vedere quali sono stati, e perché, gli strumenti e le tecniche usate per lo sviluppo del software. Verrà ampiamente discusso il codice del software definendo le sue principali caratteristiche. Infine, verrà mostrata, passo per passo, l'esecuzione della tesi nel suo complesso e quali posso essere i suoi sviluppi futuri.

## CAPITOLO 1: Conoscenze preliminari e strumenti

I problemi affrontati durante lo svolgimento della tesi di laurea sono sia di natura fisico-matematica che di natura informatica legata allo sviluppo del software.

Le conoscenze preliminari per affrontare tale tesi derivano dai corsi di:

- Analisi 1
- Fisica 1
- Fondamenti di Programmazione
- Reti di Calcolatori

Oltre a queste conoscenze preliminari sono stati approfonditi i seguenti argomenti, che hanno portato allo sviluppo del software:

- L'uso del linguaggio di programmazione C++
- La libreria multiplatforma QT

Il C++ è un linguaggio di programmazione di alto livello orientato alla programmazione ad oggetti.

Un *oggetto* è una istanza della classe, ossia una rappresentazione concreta di una classe. Quando inizializzo una variabile definendola in una classe si crea un oggetto di quella classe rappresentato dal nome della variabile istanziata. Gli oggetti comunicano fra di loro attraverso dei messaggi.

Una *classe* è un insieme di oggetti che condividono struttura e comportamenti ed è in grado di memorizzare ed elaborare i dati.

Questo tipo di linguaggio è particolarmente utile quando si tratta di sviluppare interfacce grafiche.

La libreria multiplatforma *Qt* si è rivelata uno strumento molto versatile e utile per lo scopo di questa tesi, poiché possiede numerose librerie per la manipolazione dei dati e soprattutto per lo sviluppo di programmi con interfaccia grafica tramite l'uso di widget.

Gli strumenti fisici utilizzati sono l'*Orange Pi PC 2* e l'accelerometro *LIS3DH* che verranno analizzati in dettaglio nel capitolo 2.2.

## CAPITOLO 2: Sistema client - server

### 2.1 Client-Server

In informatica il termine client-server indica un'architettura di rete composta da due moduli che generalmente sono eseguiti su macchine diverse collegate in rete.

Il client è in grado di chiedere al server, tramite una richiesta, un servizio e di renderlo disponibile all'utente finale, mentre il server è un programma che resta in attesa sulla rete, fin quando non riceve una richiesta, per erogare un servizio che viene rispedito al client.

Affinché l'interazione tra client e server possa essere effettuata è necessario che entrambi utilizzino un linguaggio comune, ovvero un protocollo applicativo. Il protocollo su cui si basano è il TCP/IP.

Il TCP (Transfer Control Protocol) gestisce l'organizzazione dei dati e il controllo della trasmissione degli stessi. Ridimensiona la grandezza dei dati da inviare, spezzettandoli in pacchetti più piccoli. Questi vengono poi ricomposti nel momento in cui arrivano al computer di destinazione.

Per spostare i pacchetti di dati il protocollo TCP ha bisogno delle informazioni fornite dal protocollo IP (Internet Protocol). Ogni computer viene identificato da un indirizzo IP, rappresentato da un numero, composto da quattro blocchi numerici, che stabilisce con esattezza il dispositivo elettronico collegato alla rete e fornisce l'indirizzo di destinazione dei dati che quest'ultimo riceve.

Nella Figura 2.1.1 possiamo vedere l'architettura di tale protocollo, messa a confronto con il modello ISO/OSI (International Standards Organization / Open Systems Interconnection) che è un modello che definisce i vari livelli e dice loro cosa devono fare e come devono funzionare.

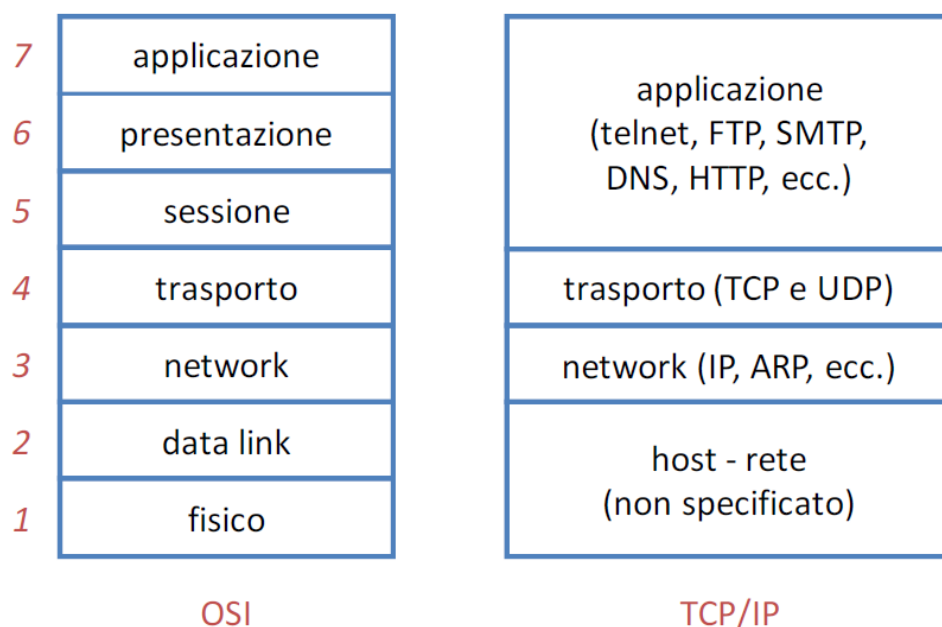


Figura 2.1.1

## 2.2 Orange Pi Pc 2 e accelerometro LIS3DH

L'*Orange Pi PC 2* è un computer single-board open-source capace di eseguire sistemi operativi come Android 5.1 e Ubuntu. Il dispositivo è illustrato in Figura 2.2.1. Con esso è possibile creare computer, server wireless e gestire suoni e video.

Nella tesi il ruolo del client, quindi di un computer, viene svolto dall'*Orange Pi PC 2*.

Il suo scopo è quello di connettersi al server, ossia il software che è stato sviluppato nella piattaforma *Qt*, tramite il protocollo TCP/IP e di inviargli una determinata quantità di dati, letti dall'accelerometro, con una determinata frequenza.

Nell'appendice A sono riportate le specifiche hardware e la definizione dell'interfaccia.

Un accelerometro è un dispositivo inerziale per la misurazione delle accelerazioni lineari.

Un dispositivo inerziale è in grado di fornire una misurazione senza bisogno di un riferimento esterno, a parte le condizioni iniziali fornite allo start-up.

Il LIS3DH è un accelerometro lineare a tre assi (XYZ), ad alte prestazioni e bassa potenza, con uscita standard digitale per interfaccia seriali I2C / SPI. Il dispositivo è dotato di modalità operative ultra-low power che consentono un risparmio energetico avanzato e funzioni integrate intelligenti.

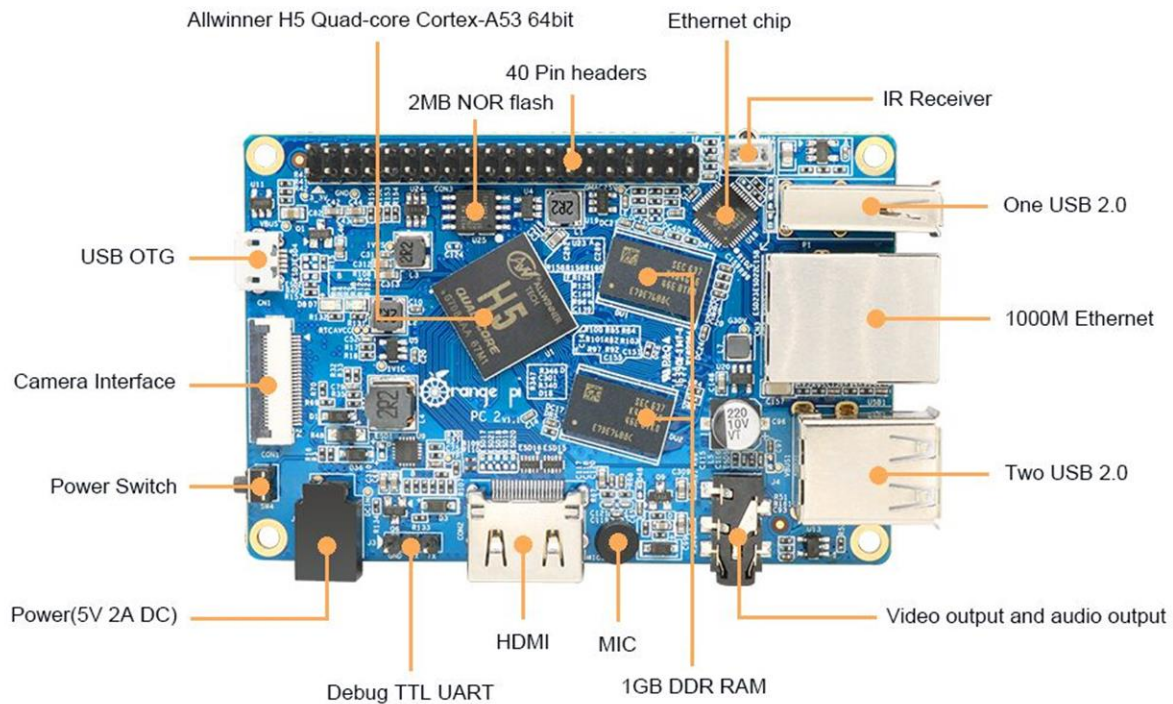
Il LIS3DH ha scale dinamiche selezionabili dall'utente di  $\pm 2g$  /  $\pm 4g$  /  $\pm 8g$  /  $\pm 16g$  ed è in grado di misurare le accelerazioni con velocità di trasmissione dati da 1 Hz a 5,3 kHz.

Le accelerazioni rilevate sono misurate in g e non in metri al secondo quadro [ $m/s^2$ ].

La Figura 2.2.2 è un'immagine del LIS3DH e si può notare l'orientazione dei tre assi.

Nell'appendice B è stata riportata la prima pagina del datasheet del dispositivo, se si vuole consultare il datasheet completo si faccia riferimento al sito web riportato nella sitografia.

## Top view



## Bottom view

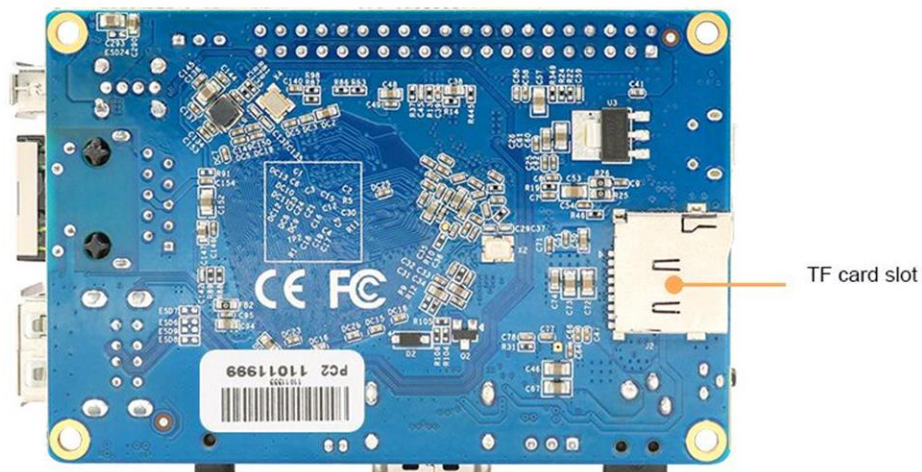


Figura 2.2.1





## 2.3 La libreria multiplatforma Qt

Qt è:

- una libreria, ossia è un insieme di funzioni o strutture dati predefinite e predisposte per essere collegate ad un programma software attraverso opportuni collegamenti,
- multiplatforma, ossia che possiede un linguaggio di programmazione che funziona su più di una piattaforma come Linux e Windows.

C, C++, Java e Python sono alcuni dei linguaggi di programmazione multiplatforma supportati dal programma Qt. Nell'applicazione software scritta per la tesi sono stati usati il C e il C++, quest'ultimo è il linguaggio standard usato da Qt. Inoltre, Qt è ottimo soprattutto per lo sviluppo di programmi con interfaccia grafica tramite l'uso di widget, il che l'ha reso il candidato ideale per lo sviluppo del server e della parte grafica della tesi.

Attraverso Qt si è creata una classe per la gestione del server e una classe per la gestione grafica, in entrambe le classi è stato possibile sfruttare le ampie librerie messe a disposizione da Qt.

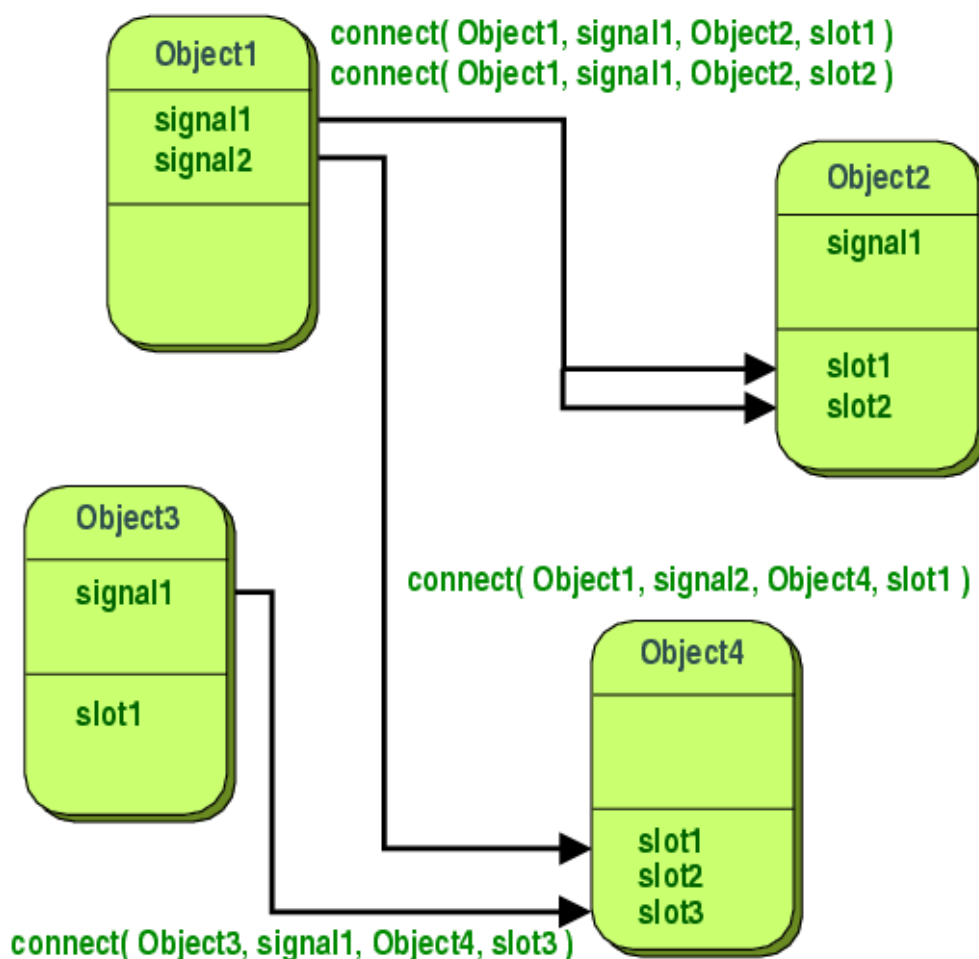


Figura 2.3.1

Qt mette a disposizione anche un importantissimo meccanismo per la comunicazione ad oggetti ossia il “signal-slot”.

Il meccanismo di “signal-slot” è utilizzato per la comunicazione tra oggetti ed è totalmente indipendente da qualsiasi loop di eventi.

I segnali vengono emessi da un oggetto quando il suo stato interno è cambiato, ad esempio quando il valore di una variabile commuta. I segnali sono funzioni di accesso pubblico e possono essere emessi da qualunque punto del programma. Il segnale viene emesso dalla dichiarazione “emit” durante l'esecuzione del codice.

Uno slot viene chiamato quando viene emesso un segnale ad esso collegato. Gli slot sono normali funzioni C ++ e possono essere chiamati normalmente; la loro unica caratteristica speciale è che i segnali possono essere collegati agli slot. È possibile che lo slot chiamato appartenga ad un'istanza diversa da quella del segnale emesso.

Qt mette a disposizione segnali e slot predefiniti, ma nulla vieta di crearne di nuovi per i propri programmi.

Il meccanismo di “signal-slot” garantisce che la firma di un segnale emesso corrisponda alla firma dello slot di ricezione, cosicché il collegamento risulti univoco. Questo ci permette di accoppiare liberamente i segnali e gli slot, inoltre una classe che emette un segnale non sa né si preoccupa di quali slot ricevono il segnale.

Quando un segnale si connette a uno slot, lo slot verrà eseguito immediatamente, proprio come una normale chiamata di funzione, e riceve, se presenti, qualsiasi tipo di parametri inviati dal segnale.

È possibile connettere tutti i segnali che si desidera su un singolo slot e un segnale può essere collegato a tutti gli slot di cui si ha bisogno.

Se più slot sono collegati a un segnale, gli slot verranno eseguiti uno dopo l'altro, nell'ordine in cui sono stati collegati durante l'esecuzione del codice.

La Figura 2.3.1 aiuta a chiarire questi concetti sopra espressi.

Nei capitoli 3.4 e 3.6 torneremo ad analizzare questo meccanismo e vedremo le sue applicazioni pratiche.

## CAPITOLO 3: Software

### 3.1 L'architettura del software

Prendendo a riferimento la Figura 3.1.1, adesso si vuole discutere l'architettura del software che è stato sviluppato.

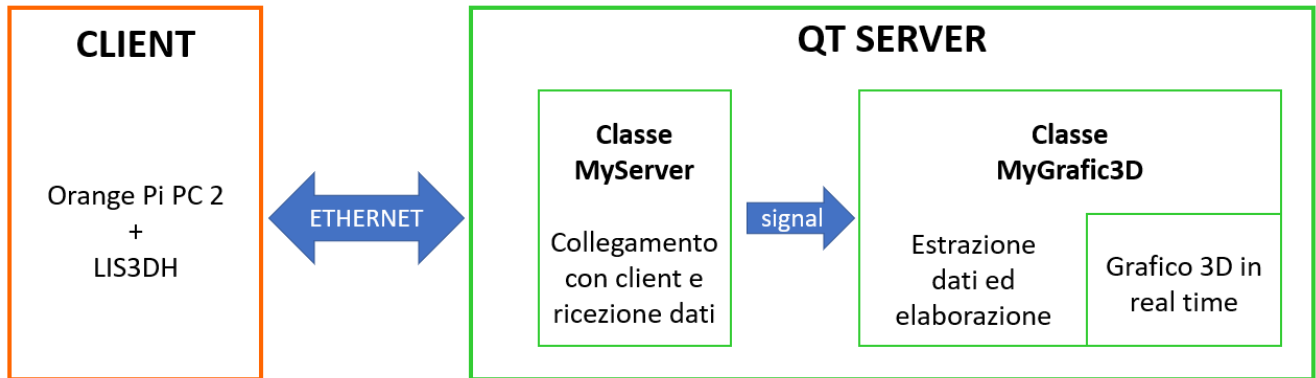


Figura 3.1.1

Notiamo immediatamente che client e server sono fisicamente separati, ciò non deve affatto sorprendere, infatti i due dispositivi comunicano tra loro attraverso la rete internet, tramite il protocollo TCP/IP.

Il client che è composto dall'*Orange Pi PC 2* e dall'accelerometro *LIS3DH*.

Il *LIS3DH* è il dispositivo di input, il quale rileva le variazioni di accelerazione, mentre l'*Orange Pi PC 2* elabora i dati dell'accelerometro e attraverso il programma "acc\_lis3dh.exe" crea un client e una connessione TCP/IP verso il server.

I dati delle proiezioni dell'accelerazione vengono inserite in una stringa e poi al server, il quale si occuperà della loro elaborazione.

Il server è composto da due classi:

1. la classe del server che si occupa:
  - della connessione col client,
  - di eseguire una prima estrazione dei dati,
  - di passare la stringa delle accelerazioni alla classe grafica.
2. la classe grafica che si occupa:
  - fare il set up del grafico tridimensionale,
  - plottare i punti in tempo reale,
  - estrarre le accelerazioni e convertirle in dati di tipo float,
  - calcolare le posizioni in funzioni delle accelerazioni.

Le classi comunicano fra loro tramite il meccanismo di signal-slot di Qt spiegato nel capitolo 2.3

### 3.2 main.cpp

Analizziamo il file main.cpp, all'inizio del file troviamo:

```
QApplication app(argc, argv);

Q3DScatter graph;
```

La prima è l'inizializzazione della **QApplication**, che gestisce il flusso di controllo dell'applicazione e le impostazioni principali dell'applicazione GUI. Mentre la seconda è la variabile che inizializza la classe che si occupa di tutta la grafica di tipo dispersione tridimensionale del programma.

Esistono vari tipi di classi grafiche, qui è stato usato il **Q3DScatter** ma Qt mette a disposizione altre classi, ad esempio: **Q3DBars** e **Q3DSurface**, tutte classi accessibili grazie a **<QtDataVisualization>**.

A questo punto viene eseguito un codice di controllo:

```
if (!graph.hasContext())
{
    qWarning() << QStringLiteral("Impossibile inizializzare
                                il contesto OpenGL!");
    return -1;
}
```

che restituisce "true" se il contesto OpenGL del grafico è stato inizializzato correttamente.

Ora viene impostata la finestra grafica:

```
graph.setFlags(graph.flags() ^ Qt::FramelessWindowHint);
graph.resize(800, 500);
graph.setTitle("TcpServerGraphic3D : Brunato Lorenzo");
```

La funzione **setFlags** imposta un flag che produce differenti caratteristiche alla finestra grafica in base al tipo di flag scelto. Quello nel programma produce una finestra senza bordi. È possibile consultare il sito di Qt per conoscere tutti i flag e le loro caratteristiche.

La funzione **resize** imposta manualmente le dimensioni, rispettivamente lunghezza e altezza, della finestra grafica.

La funzione **setTitle** imposta il titolo della finestra.

A questo punto è necessario chiamare le classi MyServer e mygrafic3d, a quest'ultima viene passata la variabile *graph*.

```
mygrafic3d mygrafic3d (&graph);  
MyServer server;
```

La funzione *show* che fa apparire la finestra grafica ed *exec* la blocca finché l'utente non deciderà di chiuderla.

```
graph.show();  
return app.exec();
```

### 3.3 Myserver.h

La classe `MyServer` è di tipo `QObject`.

All'interno della classe è possibile distinguere vari oggetti e metodi che analizziamo qui di seguito.

Nella sezione `public`:

```
QByteArray mykeypass;  
int controllo;  
char socketread[50];  
char messaggio[50];
```

- `mykeypass` è la variabile sulla quale verrà scritto ciò che viene letto dal socket.
- `controllo` assume solo due valori 0 o 1 in base a una condizione che verrà espressa nel capitolo 3.3.b.
- `socketthread` è una stringa nella quale saranno scritti i dati delle accelerazioni.
- `messaggio` è una stringa che mi indica che tipo di dati sono stati inviati dal client.

Nella sezione `public slots`:

```
void newConnection();  
void ascolto_client();  
void identifica_messaggio();
```

- La funzione `newConnection` crea una nuova connessione ogni volta che viene richiesto dal client.
- La funzione `ascolto_client` viene chiamata ogni volta che il client invia dei dati.
- La funzione `identifica_messaggio` riconosce il tipo di messaggio inviato dal client e ne estrae il contenuto.

Le prime due funzioni vengono chiamate dai rispettivi segnali mediante il meccanismo di signal-slot.

Nella sezione **signals**:

```
void signal_data(const QByteArray &);  
void signal_resetarray();
```

- Il segnale **signal\_data** viene emesso ogni volta che il messaggio ricevuto dal client contiene le accelerazioni ed è connesso allo slot **slot\_data** contenuto in mygrafic3d.cpp.
- Il segnale **signal\_resetarray** viene emesso ogni volta che il messaggio ricevuto dal client contiene il numero di campioni ed è connesso allo slot **slot\_resetarray** contenuto in mygrafic3d.cpp.

Nella sezione **private**:

```
QTcpServer *server;  
QTcpSocket *socket;
```

- Inizializzo il server chiamandolo server.
- Inizializzo il socket chiamandolo socket.

### 3.4 Myserver.cpp

Il primo metodo che troviamo nella classe myserver.cpp è il seguente

```
MyServer::MyServer(QObject *parent) : QObject(parent)
{
    controllo = 0;
    server = new QTcpServer(this);

    connect (server, SIGNAL(newConnection()), this,
             SLOT(newConnection()));

    if (!server->listen(QHostAddress::Any, PORTA))
    {
        qDebug() << "Il Server non può partire!";
    }
    else
    {
        qDebug() << "Server partito!";
    }
}
```

Questa funzione crea un nuovo server, che per l'appunto è stato chiamato `server`, e rimane in attesa finché non riceve il segnale `newConnection` che indica l'arrivo di una nuova connessione. La `connect` connette il segnale che è appena arrivato al server stesso ed esegue il metodo `newConnection`.

Questo è il primo esempio del meccanismo signal-slot di Qt descritto nel capitolo 2.3.

Più in generale la `connect` funziona nel seguente modo:

`QObject::connect(const QObject *mittente, const char *segnale, const QObject *ricevitore, const char *metodo).`

Infine, il server si mette in ascolto sull'indirizzo IP del computer che stiamo usando sulla `PORTA` che è stata definita all'inizio del file tramite la `#define`.

Il metodo `newConnection` aggancia il socket al server e aspetta che venga emesso il segnale `readyRead` che verrà connesso al metodo `ascolto_client` del server.

In questi due casi, dov'è stato utilizzato il meccanismo di signal-slot, i segnali che il server riceve sono definiti nelle librerie del Qt mentre nel prossimo metodo vediamo com'è possibile generare dei segnali personalizzati.



```

void MyServer::ascolto_client()
{
    if (controllo == 0)
    {
        controllo = 1;
        mykeypass = socket->readAll();
        [...]
    }
    else
    {
        mykeypass = socket->readAll();
        [...]

        identifica_messaggio();

        if (strcmp(messaggio, "C") == 0)
        {
            [...]
            emit signal_resetarray();
        }
        else if (strcmp(messaggio, "A") == 0)
        {
            emit signal_data(socketread);
        }
        else
        {
            qDebug() << "ERRORE: stringa non riconosciuta!!!";
        }
    }
}

```

Questo metodo memorizza nella variabile **mykeypass** ciò che viene scritto sul socket.

Se la variabile **controllo = 0** ignorerò la prima stringa di dati inviata al server poiché essa contiene dei parametri d'impostazione per l'accelerometro, l'indirizzo IP e la porta sulla quale il server è in ascolto. Finita quest'operazione avremo **controllo = 1** e in **mykeypass** saranno memorizzati o il numero di campioni o il valore delle accelerazioni.

Per capire il contenuto di **mykeypass** viene utilizzata la funzione **identifica\_messaggio** che prende la stringa, la elabora e ne separa il contenuto in due stringhe.

Analizziamo la gestione delle stringhe.

In programmazione una stringa è un vettore di caratteri il cui ultimo elemento è il codice ASCII '\0', in modo che il programma possa identificare la fine della stringa.

La stringa che viene creata ha un formato predefinito che deve essere conosciuto dal server in questo modo posso estrarne i dati in maniera corretta per poi trasformarli in variabili numeriche di tipo float, ossia un dato in virgola mobile rappresentato su almeno byte.

Se non si è a conoscenza di tale formato l'estrazione dei dati sarebbe errata generando così dati errati nel programma e portando a dei risultati finali non coerenti con la realtà.

Il formato che si è scelto di usare per le stringhe è il seguente:

- Per i valori delle accelerazioni: 

A	:		a	c	c	X		a	c	c	Y		a	c	c	Z		\	0
---	---	--	---	---	---	---	--	---	---	---	---	--	---	---	---	---	--	---	---
- Per il numero di campioni: 

C	:		n	u	m	e	r	o	c	a	m	p	i	o	n	i	\	0
---	---	--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Il primo elemento di entrambe le stringhe è un carattere:

- 'A' se la stringa contiene le accelerazioni;
- 'C' se la stringa contiene il numero di campioni.

Il secondo elemento di entrambe le stringhe è il carattere ':' il quale viene usato come carattere di controllo per determinare la fine del tipo di messaggio inviato dal client e di conseguenza il tipo di elaborazione da usare, in questo caso, per le due diverse stringhe.

Il terzo carattere è uno spazio che verrà ignorato in entrambi i casi.

Dal quarto carattere in poi c'è il valore inviato dal client, per l'accelerazione il valore scritto nella stringa **socketread** dove i tre valori delle accelerazioni sono sempre separati da uno spazio e l'ultimo elemento è il codice ASCII '\0'. Mentre per il campione il valore viene scritto nella stringa **messaggio**.

Quindi il metodo **ascolto\_client** emetterà due segnali personalizzati in base al tipo di **messaggio**:

- Se **messaggio** coincide con col carattere 'A' allora emetto il segnale **signal\_data** a cui passo la stringa **socketread**.
- Se **messaggio** coincide con col carattere 'C' allora emetto il segnale **signal\_resetarray**.
- Se **messaggio** non rientra nei casi precedenti il programma darà un messaggio di errore per l'utente

I segnali personalizzati sono facili da utilizzare infatti è sufficiente ricorrere alla funzione emit seguita dal nome del segnale e, eventualmente, scrivere il nome della variabile che si vuole passare al metodo collegato.

I metodi collegati si trovano nel file "mygrafic.cpp" e verranno analizzati nel capitolo 3.6.

Con questo termina l'analisi della classe myserver.

### 3.5 Mygrafic3d.h

Essa si occupa di tutta la gestione grafica del programma e di calcolare la posizione usando le formule del moto uniformemente accelerato.

Anche la classe `mygrafic3d` è di tipo `QObject`.

All'intero della classe è possibile distinguere vari oggetti e metodi che analizziamo qui di seguito, ma anche una struct. Partiamo da quest'ultima.

```
struct valori
{
    float accelerazione;
    float velocita;
    float posizione;
};
```

È stata definita una `struct valori` che contiene i valori, di tipo float, di accelerazione, velocità e posizione. Essa ci permette di tenere in memoria lo stato precedente in cui si trova l'accelerometro. Questo ci tornerà utile durante il calcolo della posizione poiché sappiamo dalla fisica che per calcolare la velocità e la posizione dello stato *i*-esimo dobbiamo conoscere le condizioni dello stato *i*-1. Le condizioni iniziali della `struct` per i valori di X, Y e Z sono tutti posti a zero.

Nella sezione `public`:

```
char dato[50];
float acc_x, acc_y, acc_z;
float xpos, ypos, zpos;

void estrai_dati();
void addData();

float calcolo_posizione_da_accelerazioni (float acc_letta,
                                           float acc_prec, float vel_prec, float pos_prec,
                                           int verso);
float moto_uniformemente_accelerato_velocita (float a,
                                              float v0);
float moto_uniformemente_accelerato_posizione (float a,
                                              float v0, float s0);
```

- In `dato` viene copiata la stringa contenente il valore delle accelerazioni.
- I valori di tipo float `acc_x`, `acc_y`, `acc_z` contengono le accelerazioni estratti dalla stringa.
- I valori di tipo float `xpos`, `ypos`, `zpos` contengono le posizioni ricavate dal moto uniformemente accelerato.
- La funzione `estrai_dati` separa la stringa `dato` in tre stringhe che verranno convertite in `acc_x`, `acc_y` e `acc_z`.

- La funzione `addData` aggiunge i dati `xpos`, `ypos`, `zpos` al grafico tridimensionale.
- La funzione `calcolo_posizione_da_accelerazioni` riceve tutti i parametri necessari per calcolare la posizione chiamando da sua volta le funzioni `moto_uniformemente_accelerato_velocita` e `moto_uniformemente_accelerato_posizione`, le cui funzioni sono facilmente intuibili.

Nella sezione `public slots`:

```
void slot_data(const QByteArray &);
void slot_resetarray();
```

- Il metodo `slot_data` è connesso al segnale `signal_data` emesso dalla classe del server.
- Il metodo `slot_resetarray` è connesso al segnale `signal_resetarray` sempre emesso dalla classe del server.

Nella sezione `private`:

```
QScatter3DSeries *m_series;
Q3DScatter *m_graph;
[...]
```

- `*m_series` contiene le serie di punti.
- `*m_graph` si occupa di tutte le funzioni relative alla grafica.

Questi sono i due parametri più importanti da analizzare.

### 3.6 Mygrafic3d.cpp

Il primo metodo che troviamo nella classe mygrafic3d.cpp è il seguente:

```
mygrafic3d::mygrafic3d(Q3DScatter *scatter)
    : [...]
{
    [...]

    m_graph->axisX()->setTitle("Y");
    m_graph->axisY()->setTitle("Z");
    m_graph->axisZ()->setTitle("X");

    [...]
    m_graph->addSeries(m_series);

    connect(&server, &MyServer::signal_resetarray, this,
            &mygrafic3d::slot_resetarray);
    connect(&server, &MyServer::signal_data, this,
            &mygrafic3d::slot_data);
}
```

Tale metodo inizializza una serie di variabili tra cui **m\_series** e **m\_graph** che troviamo nella sezione **private**, e nella prima parte del codice vengono eseguite delle istruzioni dedicate alla parte grafica ad esempio: viene scelto il tema del grafico, viene impostato il punto visivo da cui viene visto il grafico e vengono impostati gli assi.

Qt disposizione degli assi cartesiani tridimensionali secondo la regola della mano sinistra, ma solitamente noi rappresentiamo un sistema secondo la regola della mano destra.

Si possono osservare le differenze nella Figura 3.6.1.

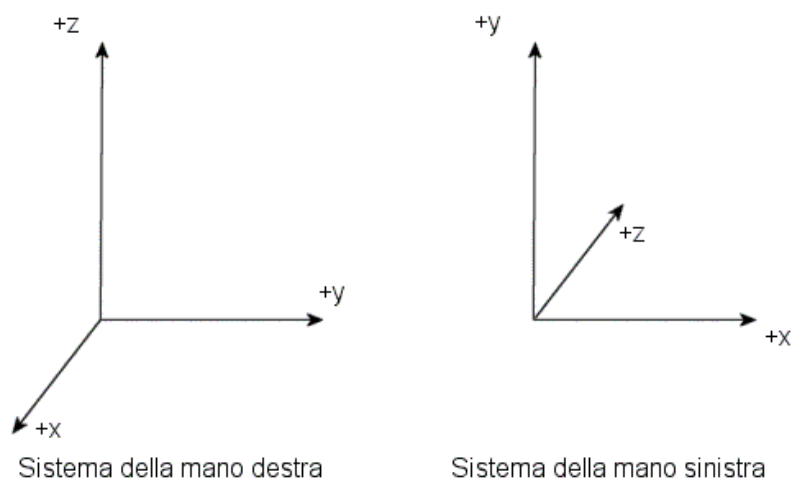


Figura 3.6.1

Per facilitare la visualizzazione grafica all'utente finale è stato sufficiente eseguire una rotazione degli assi in valore assoluto nel seguente modo:

$$\begin{aligned} X_{sx} &\rightarrow Y_{dx} \\ Y_{sx} &\rightarrow Z_{dx} \\ Z_{sx} &\rightarrow X_{dx} \end{aligned}$$

Dove a sinistra troviamo gli assi secondo la regola della mano sinistra mentre a destra troviamo gli assi secondo la regola della mano destra.

Questo giustifica il modo in cui sono state scritte le funzioni `setTitle`.

Questa rotazione della terna la ritroveremo più avanti quando usiamo la funzione per plottare i punti nel grafico, in coerenza col sistema di riferimento scelto ossia quello della mano destra.

La funzione `addSeries` aggiungerà al grafico gli elementi scritti nella variabile `m_series`.

Infine, troviamo le `connect` che, come spiegato nel capitolo 2.3, collegano i segnali `signal_data` e `signal_resetarray` provenienti dalla classe `myserver` ai loro rispettivi metodi appartenenti alla classe `mygrafic3d`. È in questo modo che avviene il passaggio dei dati fra le due classi.

Ora analizziamo i metodi eseguiti a seguito della ricezione dei segnali e incominciamo dal metodo `slot_resetarray`.

```
void mygrafic3d::slot_resetarray()
{
    m_series->dataProxy()->resetArray(new QScatterDataArray);
}
```

Il suo unico scopo è quello di creare una nuova `m_series`, eliminando l'eventuale suo contenuto precedente, ogni volta che dal client invia un numero di campioni.

Di questa funzione è possibile avere un riscontro visivo ogni volta che: finito un primo flusso di campioni, si decide di inviarne un secondo. Ciò significa che all'invio del numero di campioni del secondo flusso vedremo sparire dal grafico i punti del precedente flusso e apparire quelli del nuovo.

È molto più interessante analizzare le funzioni che contiene il metodo `slot_data`:

```
void mygrafic3d::slot_data(const QByteArray &stringa)
{
    strcpy(dato, stringa);
    estrai_dati();
    addData();
}
```

Esso viene chiamato ogni volta che il server emette il segnale `signal_data` contenente la stringa coi valori delle accelerazioni e copia questi valori nella variabile `dato` che sarà elaborata nella prima funzione `estrai_dati`.

```
void mygrafic3d::estrai_dati()
{
    [...]

    while (dato[i] != ' ')
    {
        asseX[j] = dato[i];
        j++;
        i++;
    }
    asseX[j] = '\0';
    i++;
    j = 0;

    while (dato[i] != ' ')
    {
        asseY[j] = dato[i];
        j++;
        i++;
    }
    asseY[j] = '\0';
    i++;
    j = 0;

    while (dato[i] != ' ')
    {
        asseZ[j] = dato[i];
        j++;
        i++;
    }
    asseZ[j] = '\0';
}
```

```

acc_x = std::stof(asseX);
acc_y = std::stof(asseY);
acc_z = std::stof(asseZ);

qDebug() << "Xacc: " << acc_x << "Yacc: " << acc_y <<
          "Zacc: " << acc_z;

xpos = calcolo_posizione_da_accelerazioni (acc_x,
      X.accelerazione, X.velocita, X.posizione, 1);
ypos = calcolo_posizione_da_accelerazioni (acc_y,
      Y.accelerazione, Y.velocita, Y.posizione, 1);
zpos = calcolo_posizione_da_accelerazioni (acc_z,
      Z.accelerazione, Z.velocita, Z.posizione, -1);

qDebug() << "Xpos: " << xpos << " Ypos: " << ypos <<
          "Zpos: " << zpos;
qDebug() << "-----";
}

```

Ricordiamo che il formato della stringa è il seguente:

accelerazione asse X, spazio, accelerazione asse Y, spazio, accelerazione asse Z, spazio, \0.

La prima parte della funzione estrae carattere per carattere i valori dentro **dato** e li copia nelle stringhe rispettive ad ogni accelerazione, **asseX**, **asseY** e **asseZ**, finché non trova uno spazio. Queste tre stringhe vengono poi terminate da '\0'.

A questo punto è possibile convertire i valori delle stringhe in variabili di tipo float tramite la funzione **std::stof()**. Le variabili prendono rispettivamente i nomi di **acc\_x**, **acc\_y** e **acc\_z**.

Da qui in poi tutti i valori numerici saranno di tipo float, questo perché la funzione che stamperà i punti richiede questo tipo di formato.

Per ricavare le posizioni **xpos**, **ypos** e **zpos**, tramite moto uniformemente accelerato, chiamiamo il metodo **calcolo\_posizione\_da\_accelerazioni** a cui passiamo i valori di **acc\_x**, **acc\_y** e **acc\_z** appena calcolati e i valori all'interno della **struct valori** che contengono i valori di accelerazione, velocità e posizione dello stato precedente e il verso degli assi, poiché la forza di gravità ha verso opposto rispetto all'asse Z per come l'abbiamo definita nel nostro software.



```

float mygrafic3d::calcolo_posizione_da_accelerazioni (float
acc_letta, float acc_prec, float vel_prec, float pos_prec, int
verso)
{

    [...]

    acc_nuova = acc_letta * G;
    delta_acc = (acc_nuova - acc_prec) * verso;
    acc_prec = delta_acc;

    if (fabs(delta_acc) > EPSILON)
    {
        vel_prec = moto_uniformemente_accelerato_velocita
                    (acc_prec, vel_prec);
        pos_prec = moto_uniformemente_accelerato_posizione
                    (acc_prec, vel_prec, pos_prec);
    }

    [...]

    return pos_prec;
}

```

I valori delle accelerazioni ricavate sono misurati dall'accelerometro in g, perciò la prima operazione da fare è convertirle in metri al secondo quadro [m/s<sup>2</sup>].

Dopodiché si calcola la variazione dell'accelerazione fra lo stato attuale e quello precedente. Se il valore assoluto di tale valore è superiore a **EPSILON**, un valore preimpostato all'inizio del programma, si passa al calcolo della velocità e posizione dello stato attuale. Altrimenti è possibile supporre che l'accelerometro e l'eventuale strumentazione ad esso collegato siano fermi.

Questi valori andranno poi a sostituire quelli dello stato precedente, per favorire l'aggiornamento dello stato si memorizzano i valori ricavati nelle variabili **vel\_prec** e **pos\_prec** che alla successiva chiamata della funzione rappresenteranno proprio lo stato precedente.

I metodi **moto\_uniformemente\_accelerato\_velocita** e **moto\_uniformemente\_accelerato\_posizione** calcolano la velocità e la posizione seconde le formule ben note della fisica meccanica:

$$v = v_0 + at$$

$$s = s_0 + v_0t + \frac{1}{2}at^2$$

La variabile “t” è stata definita all’inizio del file nel seguente modo: `#define t 1`, poiché stiamo lavorando con dimensioni misurabili in centimetri. Per applicazioni future tale parametro può essere modificato di conseguenza.

Essendo stato impostato nel client un tempo di campionamento di un decimo di secondo è stato possibile supporre l’intervallo sufficientemente piccolo tale da considerare l’accelerazione costante in essi e di conseguenza eseguire le operazioni sopra elencate.

Intervalli di tempo più grandi porterebbero ad errori evidenti, anche nella realtà, nel calcolo delle posizioni, mentre intervalli di tempo più piccoli non darebbero il tempo necessario al compilatore ad eseguire tutto il programma correttamente.

A questo punto abbiamo ricavato le posizioni `xpos`, `ypos` e `zpos` non resta che plottarle sul grafico tridimensionale e per farlo usiamo il secondo metodo `addData` del metodo chiamante `slot_data`.

```
void mygrafic3d::addData()
{
    QScatterDataArray *array = new
        QScatterDataArray(*m_series->dataProxy()->array());

    m_series->dataProxy()->resetArray(Q_NULLPTR);
    array->append(QScatterDataItem(QVector3D(ypos, zpos,
                                                xpos)) );
    m_series->dataProxy()->resetArray(array);
}
```

Questo metodo genera un nuovo array contenente i dati contenuti all’indirizzo di memoria dell’array e aggiunge alla fine di esso i nuovi dati tramite la funzione `append`.

I punti vengono stampati sul grafico tramite la funzione `QVector3D` che riceve come argomenti la tre posizioni in formato float. Anche qui, per i motivi spiegati a inizio capitolo, le posizioni sono state ruotate affinché l’utente finale possa visualizzare un grafico tridimensionale secondo il sistema della mano destra.

La funzione `resetArray` cancella l’array esistente se il nuovo array è diverso da esso. Se gli array sono uguali, questa funzione attiva semplicemente il segnale `arrayReset()`.

La classe `mygraic3d` eseguirà tutti i metodi sopra descritti ogni volta riceverà i segnali emessi dalla classe `MyServer`, tutto questo senza che il programma si blocchi o abbia un funzionamento anomalo.

È possibile consultare il codice completo nell’appendice C.

## CAPITOLO 4: Applicazione

### 4.1 Collegamento client – server

Innanzitutto, bisogna identificare se si ha a disposizione una connessione di rete con indirizzo IP pubblico o privato. Nel primo caso si consiglia di collegare all'*Orange Pi Pc 2* uno schermo tramite il cavo HDMI, una tastiera e un mouse tramite porte USB, in questo modo sarà più facile leggere l'indirizzo IP del dispositivo. Nel secondo caso invece è sufficiente usare la "Connessione Desktop remoto" del proprio computer, nel quale è installato Qt ed è presente l'applicazione scritta per questa tesi, poiché l'indirizzo IP verrà inserito manualmente.

Per prima si collega l'*Orange Pi Pc 2* al cavo di alimentazione ed eventualmente un monitor, una tastiera e un mouse se non si ha un IP privato. Quando sul dispositivo si accende una spia verde possiamo collegare l'accelerometro *LIS3DH* sui pin dell'*Orange Pi Pc 2*.

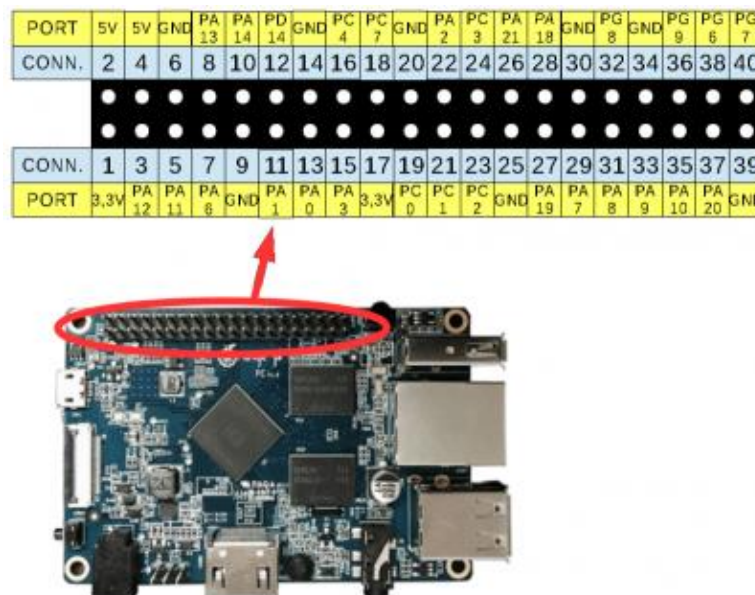


Figura 4.1.1

Facendo riferimento alle figure 4.1.1 e 2.2.2 colleghiamo i due dispositivi nel seguente modo:

- nel pin 1 collego l'SDO dell'accelerometro,
- nel pin 2 collego la  $V_{IN}$  dell'accelerometro,
- nel pin 3 collego l'SDA dell'accelerometro,
- nel pin 5 collego l'SCL dell'accelerometro,
- nel pin 6 collego la GND dell'accelerometro.

Questa configurazione permette di usare il protocollo di comunicazione I2C.

Infine, colleghiamo l'*Orange Pi Pc 2* al nostro computer tramite un cavo ethernet.

Dopo questa fase preliminare, andare sul proprio computer aprire il seguente percorso:  
pannello di controllo -> rete e internet -> connessioni di rete  
e si aprirà una pagina come quella riportata in Figura 4.1.2

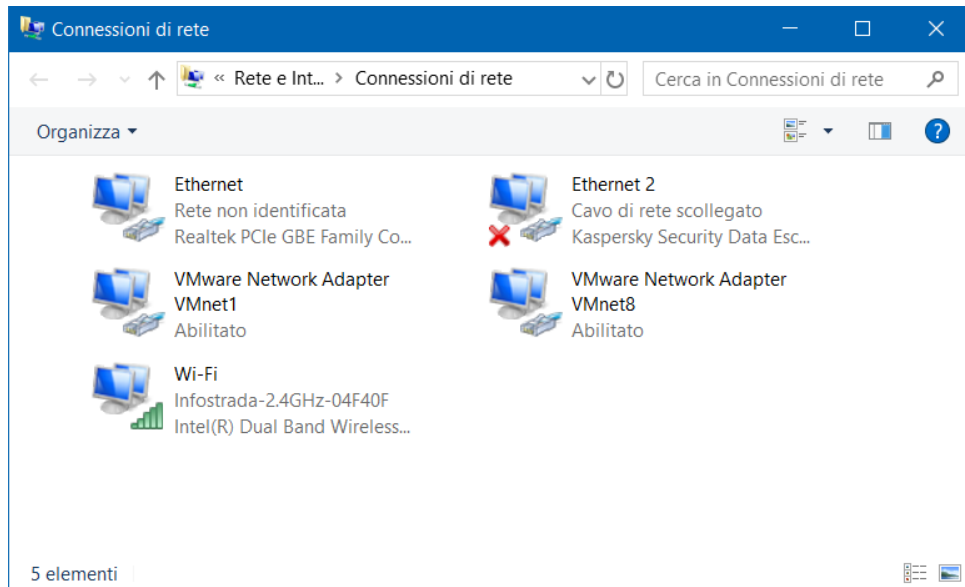


Figura 4.1.2

Ora selezionare Wi-Fi ed Ethernet e cliccando col tasto destro apparirà una finestra e bisogna selezionare l'operazione "Connessioni con bridging", ottenendo un "Bridge di rete" come in Figura 4.1.3.

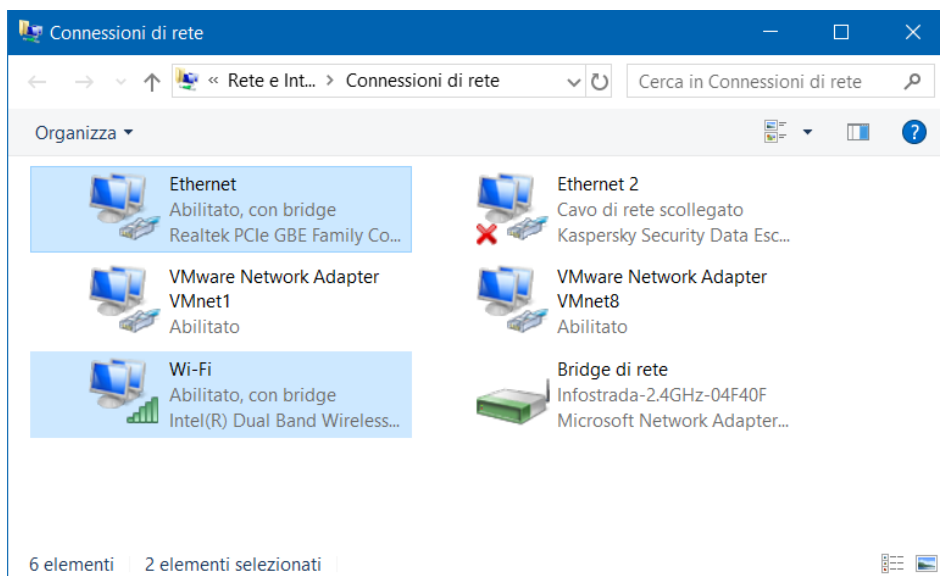


Figura 4.1.3

A questo punto cliccare col tasto destro sul “Bridge di rete” e selezionare Proprietà. Si aprirà una scheda come in Figura 4.1.4

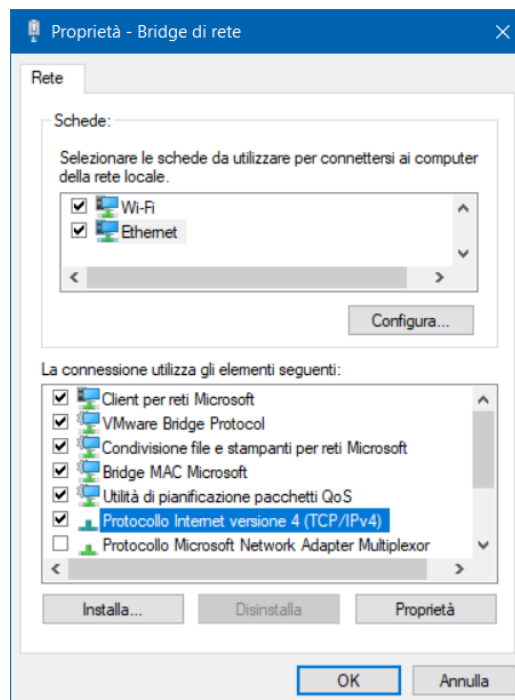


Figura 4.1.4

Qui controlliamo che Wi-Fi ed Ethernet siano entrambi fleggati, se così non fosse è sufficiente fleggarli, premere OK e riaprire la finestra in Figura 4.1.4 prima di andare avanti.

Ora, facendo doppio clic su “Protocollo Internet versione 4 (TCP/IPv4)” si aprirà una nuova scheda nella quale andremo a scrivere il nuovo indirizzo IP e altri parametri.

Per conoscere tutti i paramenti da inserire bisogna aprire il prompt dei comenti del computer e digitare l’istruzione `ipconfig/all` e leggere i dati relativi a “Scheda Ethernet Bridge di rete” come in Figura 4.1.5.

Quindi leggere i dati e inserirli come in Figura 4.1.6. Nella riga dell'indirizzo IP, inserisci l'indirizzo IPv4 del computer e nell'ultimo campo aumenta il valore di 1. Questo permette di avere due indirizzi IP diversi per l’*Orange Pi Pc 2* e il computer. Facciamo una cosa analoga per i server DNS, inserisco nella prima riga il numero letto dal prompt dei comandi, e inserisco nella seconda riga lo stesso numero aumentato di 1.

Infine, fleggere il riquadro “Convalida impostazioni all’uscita” e premere il tasto OK ed infine il tasto OK di Figura 4.1.4.

Se si apre la pagina “Diagnostica di rete windows” chiuderla per proseguire.

```
Prompt dei comandi

Scheda Ethernet Bridge di rete:

Suffisso DNS specifico per connessione:
Descrizione . . . . . : Microsoft Network Adapter Multiplexor Driver
Indirizzo fisico. . . . . : D4-25-8B-DA-83-74
DHCP abilitato. . . . . : Sì
Configurazione automatica abilitata : Sì
Indirizzo IPv6 locale rispetto al collegamento . : fe80::313a:1802:4367:8730%36(Preferenziale)
Indirizzo IPv4. . . . . : 192.168.1.13(Preferenziale)
Subnet mask . . . . . : 255.255.255.0
Lease ottenuto. . . . . : sabato 11 maggio 2019 11:47:55
Scadenza lease . . . . . : domenica 12 maggio 2019 11:47:55
Gateway predefinito . . . . . : 192.168.1.1
Server DHCP . . . . . : 192.168.1.1
IAID DHCPv6 . . . . . : 617883019
DUID Client DHCPv6. . . . . : 00-01-00-01-21-8F-4D-54-AC-E2-D3-7B-B5-A0
Server DNS . . . . . : 192.168.1.1
Server WINS primario . . . . . : 192.168.1.1
NetBIOS su TCP/IP . . . . . : Attivato
```

Figura 4.1.5

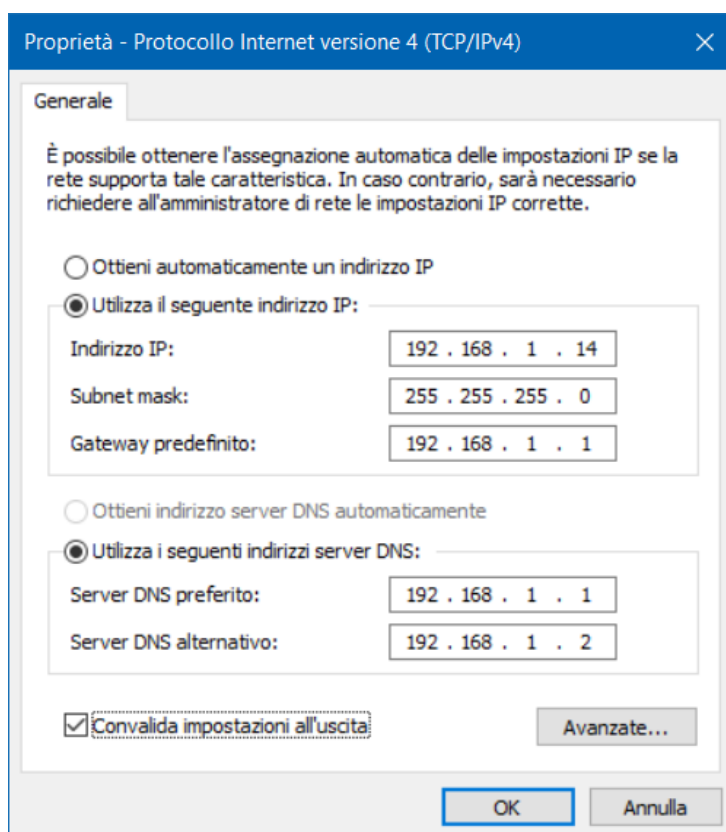


Figura 4.1.6

Ora che abbiamo creato il collegamento posso usare il “Desktop remoto” per avere un’interfaccia con l’*Orange Pi Pc 2*, come in Figura 4.1.7, nel caso di indirizzo IP privato, altrimenti se l’indirizzo IP è pubblico possiamo leggerlo nel desktop del client, cliccare sull’immagine della porta ethernet, in alto a destra, e cliccare su “Connection Information” come in Figura 4.1.8.

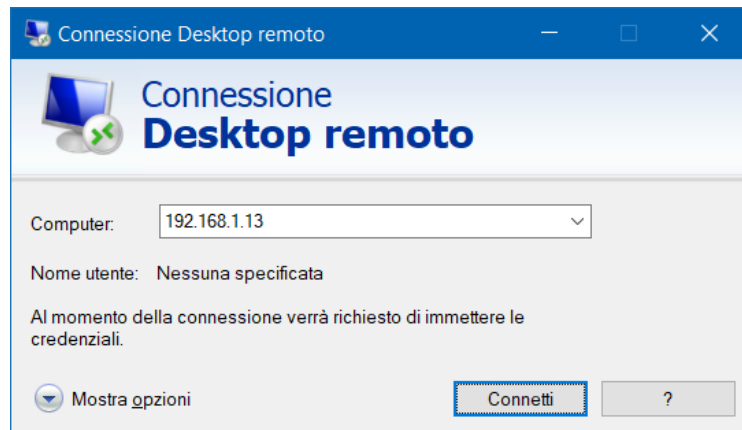


Figura 4.1.7

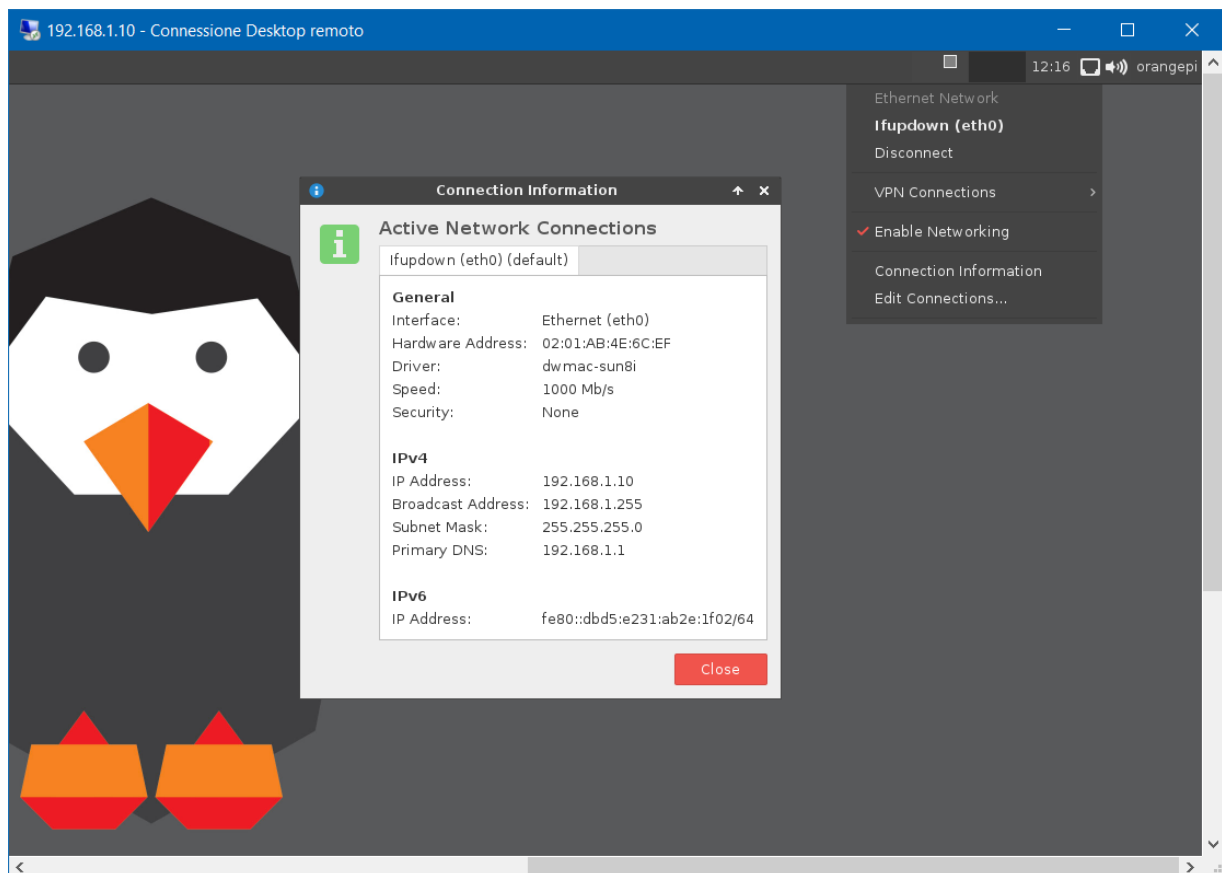


Figura 4.1.8

## 4.2 Compilazione del programma

Finita la procedura di connessione con bridging andiamo a vedere come si compila il programma sul lato client e poi sul lato server.

Il client è *case sensitive* ossia, che le lettere maiuscole e quelle minuscole vengono trattate come fossero caratteri diversi.

Quando c'interfacciamo per la prima volta col client ci verrà richiesto l'inserimento di user e password che sono riportate di seguito:

User: *orange*  
Password: *orange*



Figura 4.2.1

L'interfaccia che ci viene presentata dal client è mostrata in Figura 4.2.1 e andiamo ad aprire il terminale seguendo il percorso: Applications -> Terminal Emulator

Entrati nel terminal effettuiamo l'accesso alla modalità amministratore con comando:

*sudo su*

e reinseriamo la password:

*orange*

Verifichiamo che l'accelerometro sia stato collegato correttamente col comando:

*gpio i2cdetect*

Se il collegamento è corretto apparirà in una matrice a video il numero 19.



A questo punto accediamo alla cartella contenente il programma del client che nel nostro caso si trova nella cartella Client del Desktop perciò sul terminale andremo a scrivere:

```
cd Desktop
```

```
cd Client2
```

```
ls
```

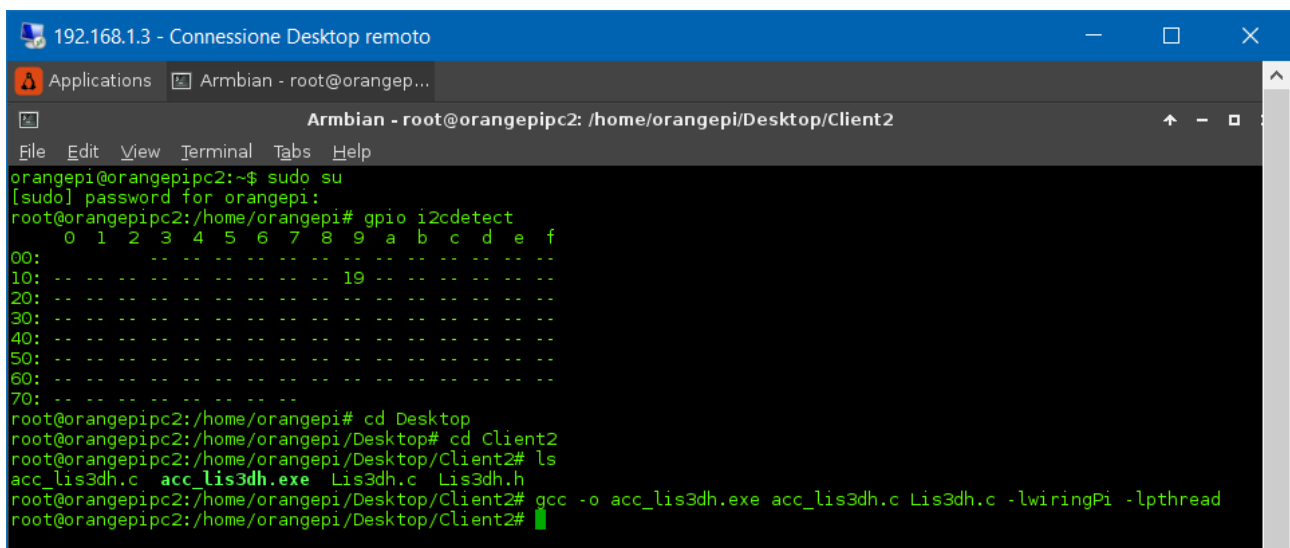
Quest'ultimo comando (elle esse) ci permette di vedere il contenuto della cartella Client2.

Adesso siamo pronti a compilare il programma col seguente comando:

```
gcc -o acc_lis3dh.exe acc_lis3dh.c Lis3dh.c -lwiringPi -lpthread
```

Se è stato eseguito tutto correttamente il compilatore non darà alcun tipo di errore ed è possibile eseguire il programma come vedremo nel capitolo 4.3.

Tutta la procedura è riportata nella Figura 4.2.2.



```
192.168.1.3 - Connessione Desktop remoto
Applications Armbian - root@orangeipc2: /home/orangepi/Desktop/Client2
File Edit View Terminal Tabs Help
orangeipi@orangeipc2:~$ sudo su
[sudo] password for orangeipi:
root@orangeipc2:/home/orangepi# gpio i2cdetect
0 1 2 3 4 5 6 7 8 9 a b c d e f
00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- 19 -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
root@orangeipc2:/home/orangepi# cd Desktop
root@orangeipc2:/home/orangepi/Desktop# cd Client2
root@orangeipc2:/home/orangepi/Desktop/Client2# ls
acc_lis3dh.c  acc_lis3dh.exe  Lis3dh.c  Lis3dh.h
root@orangeipc2:/home/orangepi/Desktop/Client2# gcc -o acc_lis3dh.exe acc_lis3dh.c Lis3dh.c -lwiringPi -lpthread
root@orangeipc2:/home/orangepi/Desktop/Client2#
```

Figura 4.2.2

Per quanto riguarda il server non serve compilare il programma poiché la compilazione e l'esecuzione vengono eseguiti in una sola volta col comando *Run* che approfondiremo nel capitolo 4.3.

Per ora ci limitiamo ad aprire il programma *Qt Creator* facendo doppio clic sulla sua icona, nella sezione “Welcome” andiamo nella sottosezione “Projects” e clicchiamo sul pulsante “Open Project” per aprire il progetto nella cartella in cui il software è salvato. Se è già stato aperto in precedenza, nella sottosezione “Projects” apparirà una lista dei progetti aperti di recente. Si veda Figura 4.2.3.

Il nome del software della tesi da aprire si chiama: **Tcp\_Server\_Graphics\_3D**.

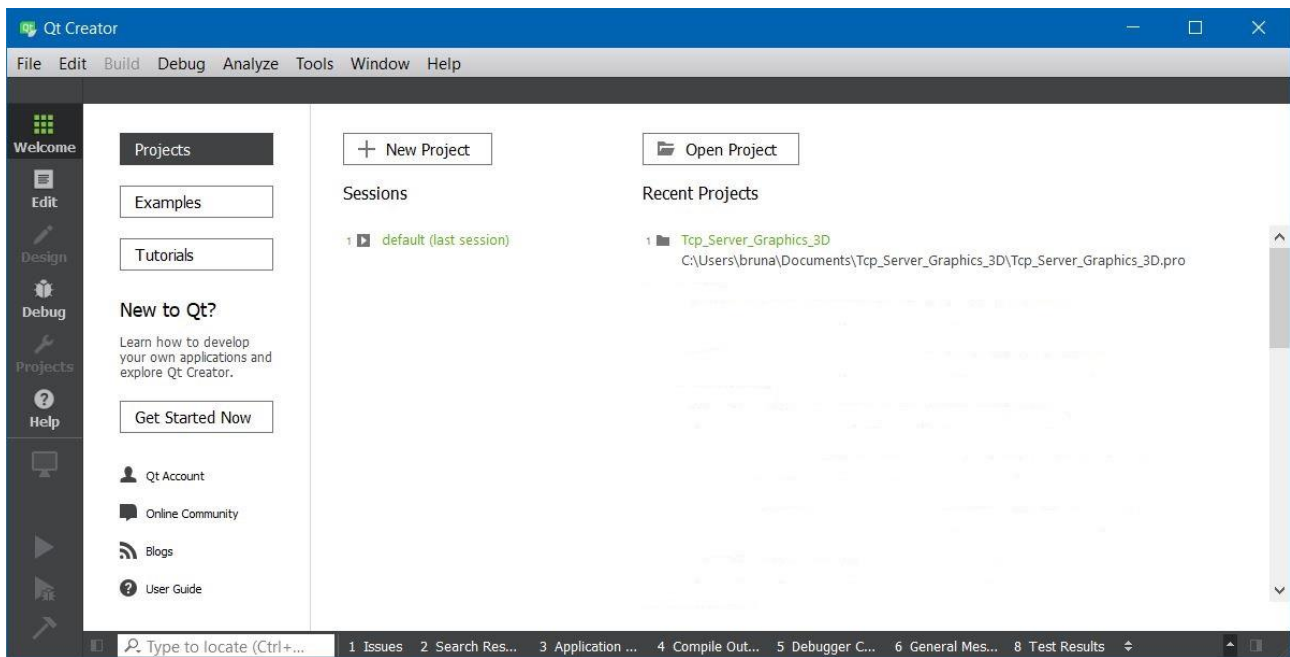


Figura 4.2.3

### 4.3 Esecuzione del programma

Ripartiamo dal client.

Una volta compilato il programma per eseguirlo usiamo l'istruzione:

`./acc_lis3dh.exe`

Questa non è l'istruzione completa, infatti vengono stampate a video le istruzioni successive che dobbiamo fornire al programma affinché venga eseguito.

La prima istruzione è specificare il protocollo di comunicazione se l'I2C o l'SPI. Ovviamente useremo il primo anche in funzione di come è stato collegato l'accelerometro. Vedi capitolo 4.1.

La seconda istruzione riguarda la scala dinamica del LIS3DH che può essere 2, 4, 8 o 16, come discusso nel capitolo 2.2. Per quest'applicazione è sufficiente una scala dinamica pari a 4.

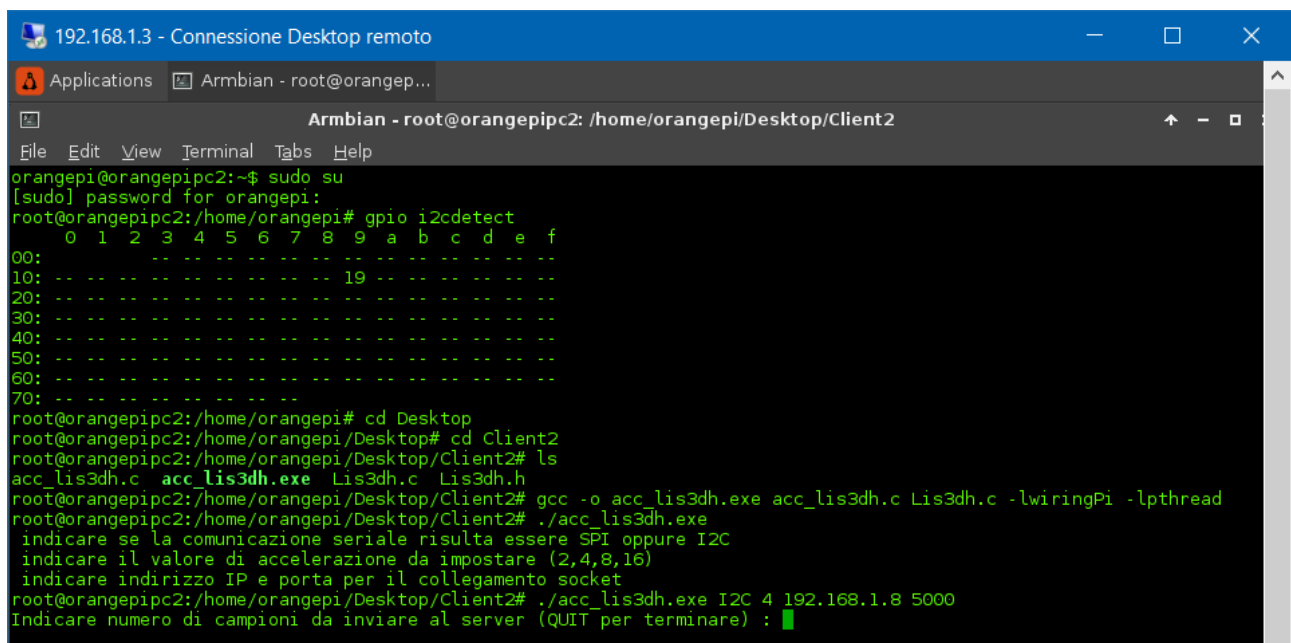
La terza istruzione è l'indirizzo IP del computer che esegue il server. Tale indirizzo può essere letto dal prompt dei comandi usando l'istruzione `ipconfig/all`.

La quarta istruzione è la porta sulla quale client e server si mettono in ascolto. La porta è la 5000, poiché è stata definita così tramite una `#define` nel file `myserver.cpp`. Vedi appendice C.

Quindi l'istruzione completa per eseguire il programma è, ad esempio:

`./acc_lis3dh.exe I2C 4 192.168.1.8 5000`

L'invio di questo comando dev'essere dato dopo aver eseguito il server, altrimenti la connessione fallisce. Una volta dato l'invio a quest'ultima istruzione il programma partirà, come in Figura 4.3.1, e sarà sufficiente seguire le istruzioni che appariranno a video.



```
192.168.1.3 - Connessione Desktop remoto
Applications  Armbian - root@orange...
Armbian - root@orangeipc2: /home/orangepi/Desktop/Client2
File Edit View Terminal Tabs Help
orangeipc2@orangeipc2:~$ sudo su
[sudo] password for orangeipc2:
root@orangeipc2:/home/orangepi# gpio i2cdetect
0 1 2 3 4 5 6 7 8 9 a b c d e f
00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- 19 -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
root@orangeipc2:/home/orangepi# cd Desktop
root@orangeipc2:/home/orangepi/Desktop# cd Client2
root@orangeipc2:/home/orangepi/Desktop/Client2# ls
acc_lis3dh.c  acc_lis3dh.exe  Lis3dh.c  Lis3dh.h
root@orangeipc2:/home/orangepi/Desktop/Client2# gcc -o acc_lis3dh.exe acc_lis3dh.c Lis3dh.c -lwiringPi -lpthread
root@orangeipc2:/home/orangepi/Desktop/Client2# ./acc_lis3dh.exe
indicare se la comunicazione seriale risulta essere SPI oppure I2C
indicare il valore di accelerazione da impostare (2,4,8,16)
indicare indirizzo IP e porta per il collegamento socket
root@orangeipc2:/home/orangepi/Desktop/Client2# ./acc_lis3dh.exe I2C 4 192.168.1.8 5000
Indicare numero di campioni da inviare al server (QUIT per terminare) :
```

Figura 4.3.1

Per quanto riguarda il server una volta aperto il programma del software è sufficiente premere il tasto *Run*, ossia il tasto play di colore verde in basso a sinistra. Se vengono rilevati errori il programma non verrà eseguito altrimenti si apriranno due finestre una del server e l'altra del grafico tridimensionale, come in Figura 4.3.2 e Figura 4.3.3.

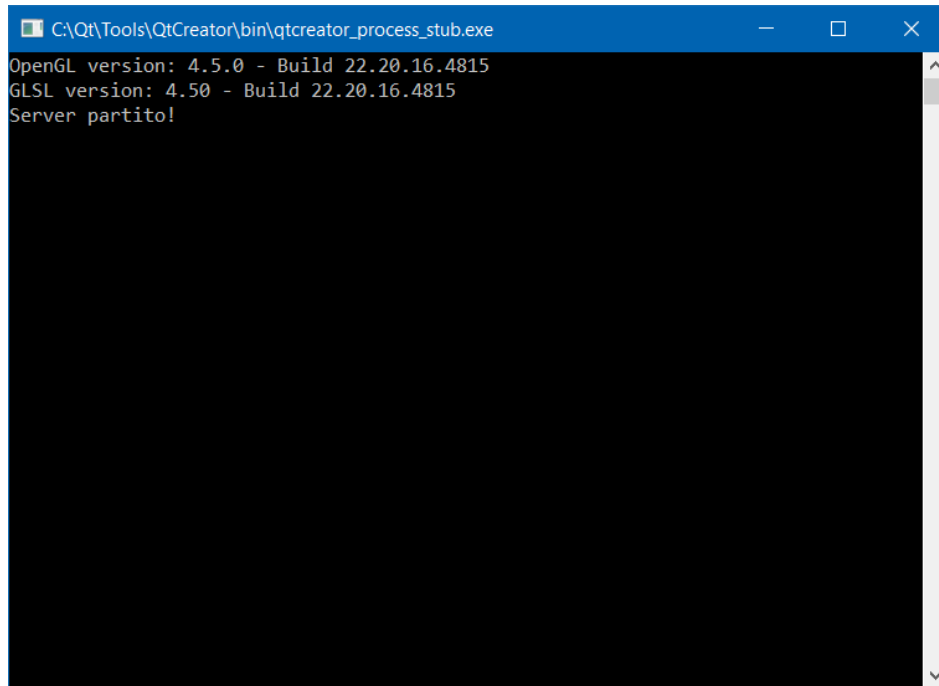


Figura 4.3.2

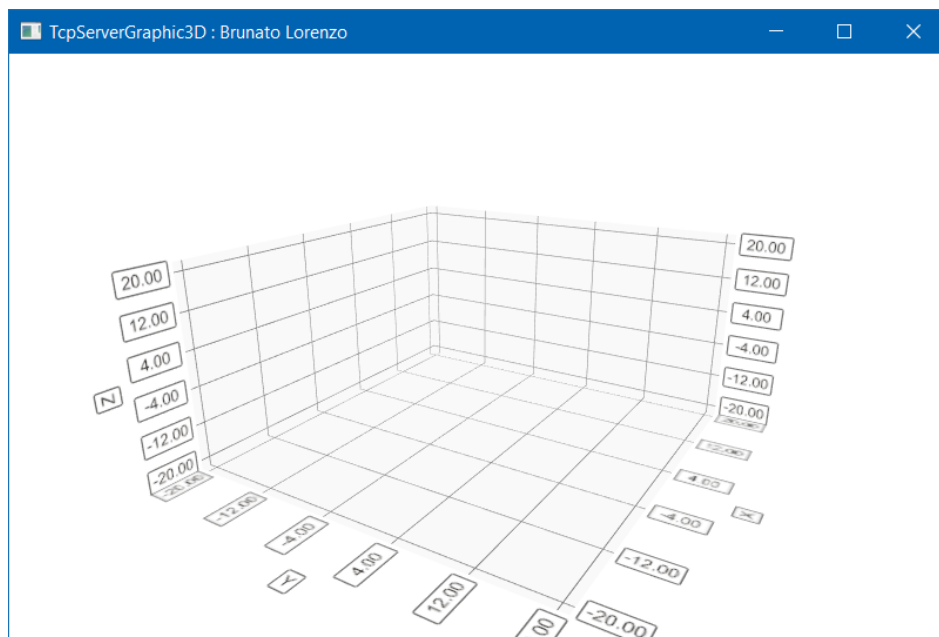


Figura 4.3.3

Adesso che client e server sono stati eseguiti e connessi possiamo indicare al client il numero campioni da inviare al server, inserendoli da tastiera, e muovendo l'accelerometro nello spazio potremmo vedere la sua traiettoria nel grafico tridimensionale, come in Figura 4.3.4.

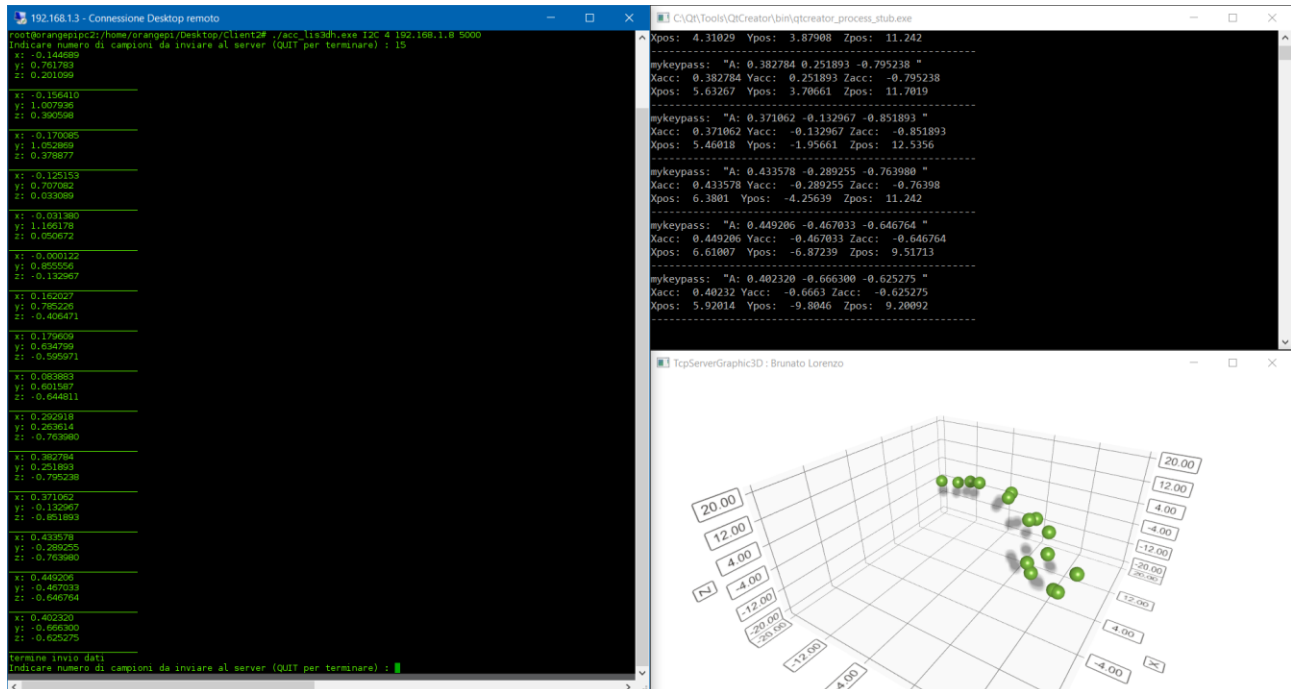


Figura 4.3.4

Sul lato client vengono stampati a video le accelerazioni in g rilevate.

Sul lato server vengono stampati a video, la stringa inviata dal client, le accelerazioni dopo che sono state convertite in dati di tipo float e le posizioni ricavate in centimetri.

Sul grafico troviamo i punti ricavati dal server. È possibile agire dinamicamente col grafico, infatti possiamo zoomare e ruotare la grafica a piacere e se si seleziona un punto vengono visualizzate le sue coordinate.

E con questo termina il capitolo sull'utilizzo del software.

## CAPITOLO 5: Conclusioni e applicazioni future

Alla luce di quanto analizzato e discusso nei capitoli precedenti, si può affermare che l'obiettivo principale della tesi è stato raggiunto.

Sfruttando l'applicazione client - server è stato possibile trasmettere i dati provenienti dall'acceleramento e tramite codici informatici, scritti in C++, è stato ricavare la posizione nello spazio e la possibilità di visualizzare le traiettorie, grazie al programma Qt Creator, in un grafico tridimensionale. Tutti questi dati messi assieme aiutano ad analizzare il comportamento dinamico del sistema ed eventualmente di apportare correzioni.

Tuttavia, il solo accelerometro non è sufficiente a ricavare la posizione nello spazio in maniera precisa perché è un dispositivo inerziale e non ha una terna cartesiana a cui fare riferimento. Perciò non riesce a distinguere una rotazione da una accelerazione lineare in mancanza di una terna di riferimento.

Essere sensibili alle rotazioni significa che l'*LIS3DH* rileva solo le accelerazioni esterne e non le accelerazioni angolari. L'accelerazione angolare è una grandezza vettoriale che rappresenta la variazione della velocità angolare al variare del tempo.

Le accelerazioni angoli rappresentano un nuovo problema fisico da studiare e successivamente da tradurre in una estensione del software già scritto.

Per rilevare tali variazioni abbiamo bisogno di un giroscopio. In fisica, il giroscopio è un dispositivo fisico rotante che, per effetto della legge di conservazione del momento angolare, tende a mantenere il suo asse di rotazione orientato in una direzione fissa.

Il giroscopio è un sensore che misura il moto rotazionale tramite la velocità angolare dell'asse. In un sistema, sottoposto a moto rotatorio, composto da tre assi differenti (X, Y e Z) avremo che ogni asse avrà una sua velocità angolare. Il calcolo della velocità angolare avviene grazie a piccole masse, tra 1 e 100 micrometri, che si muovono in funzione dei cambiamenti della velocità angolare. Queste minime variazioni vengono convertite in tensioni elettriche, amplificate ed elaborate da un microcontrollore.

Una soluzione a questo problema potrebbe essere la seguente. Sostituire il *LIS3DH* con un dispositivo che funga sia da accelerometro sia da giroscopio. La prima cosa che questo dispositivo dovrà fare è definire un'origine e una terna cartesiana a cui fare riferimento per gli spostamenti futuri. Da questo momento in poi sarà possibile determinare la posizione esatta del dispositivo.

Dal lato client, i dati rilevati dal giroscopio possono essere inseriti dentro una stringa, come è stato fatto per le accelerazioni nel capitolo 3.4, e inviare anch'essa al server insieme alla stringa delle accelerazioni rilevate nello stesso istante di tempo.

Mentre dal lato server, verrà eseguita una estrazione di questi dati e una loro conversione in posizioni angolari. Infine, in base allo studio fisico del problema, si scriverà un metodo che mette in relazione le posizioni e le posizioni angolari in modo da determinare in maniera precisa la posizione nello spazio di un oggetto, in questo caso del dispositivo accelerometro-giroscopio.

Sempre consultando il sito [www.st.com](http://www.st.com), come per l'accelerometro è stato possibile individuare un dispositivo adatto alla risoluzione a questo problema, ossia l'accelerometro e giroscopio LSM6DS3. Figura 5.1. Mentre in Figura 5.2 si trova la prima pagina del suo datasheet.

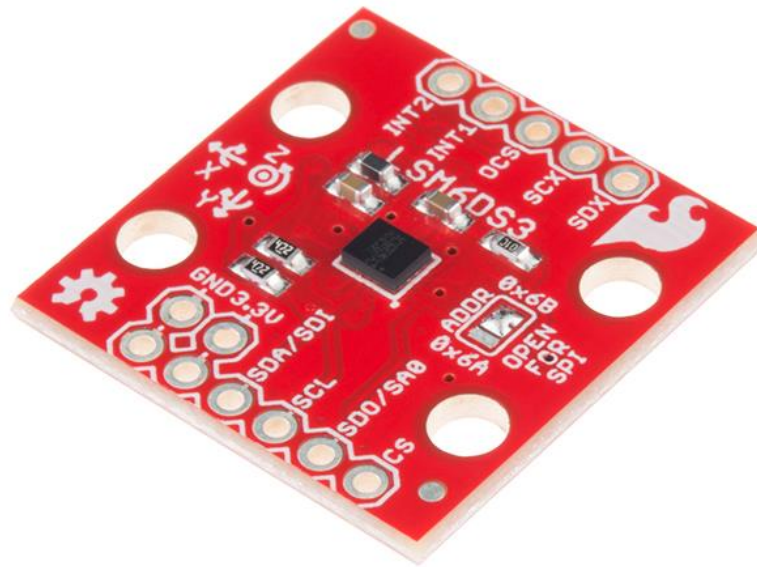


Figura 5.1

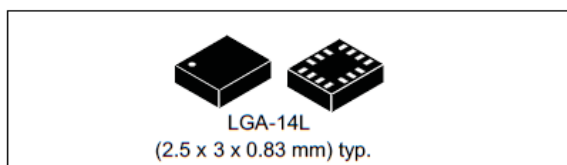
L'LSM6DS3 sembra avere tutte le caratteristiche per risolvere il problema sopra descritto, si consiglia comunque di valutarlo dopo aver studiato la natura fisica di tale problema. Inoltre, presenta caratteristiche simili all' LIS3DH.

Come si vede dalla Figura 5.1 il dispositivo possiede già un'orientazione propria degli assi cartesiani per l'accelerazione, che segue il sistema della mano destra, e le rotazioni di *pitch*, *roll* e *yaw* sui rispettivi assi X, Y e Z.

Sarà inoltre necessario verificare che quest'orientazione di assi concordi con quella dell'LIS3DH e con quella di Qt.

## iNEMO inertial module: always-on 3D accelerometer and 3D gyroscope

Datasheet - production data



### Features

- Power consumption: 0.9 mA in combo normal mode and 1.25 mA in combo high-performance mode up to 1.6 kHz.
- "Always-on" experience with low power consumption for both accelerometer and gyroscope
- Smart FIFO up to 8 kbyte based on features set
- Compliant with Android K and L
- Hard, soft ironing for external magnetic sensor corrections
- $\pm 2/\pm 4/\pm 8/\pm 16$  g full scale
- $\pm 125/\pm 250/\pm 500/\pm 1000/\pm 2000$  dps full scale
- Analog supply voltage: 1.71 V to 3.6 V
- Independent IOs supply (1.62 V)
- Compact footprint, 2.5 mm x 3 mm x 0.83 mm
- SPI/I<sup>2</sup>C serial interface with main processor data synchronization feature
- Embedded temperature sensor
- ECOPACK<sup>®</sup>, RoHS and "Green" compliant

### Applications

- Pedometer, step detector and step counter
- Significant motion and tilt functions
- Indoor navigation
- Tap and double-tap detection
- IoT and connected devices
- Intelligent power saving for handheld devices
- Vibration monitoring and compensation
- Free-fall detection
- 6D orientation detection

### Description

The LSM6DS3 is a system-in-package featuring a 3D digital accelerometer and a 3D digital gyroscope performing at 1.25 mA (up to 1.6 kHz ODR) in high-performance mode and enabling always-on low-power features for an optimal motion experience for the consumer.

The LSM6DS3 supports main OS requirements, offering real, virtual and batch sensors with 8 kbyte for dynamic data batching.

ST's family of MEMS sensor modules leverages the robust and mature manufacturing processes already used for the production of micromachined accelerometers and gyroscopes.

The various sensing elements are manufactured using specialized micromachining processes, while the IC interfaces are developed using CMOS technology that allows the design of a dedicated circuit which is trimmed to better match the characteristics of the sensing element.

The LSM6DS3 has a full-scale acceleration range of  $\pm 2/\pm 4/\pm 8/\pm 16$  g and an angular rate range of  $\pm 125/\pm 250/\pm 500/\pm 1000/\pm 2000$  dps.

High robustness to mechanical shock makes the LSM6DS3 the preferred choice of system designers for the creation and manufacturing of reliable products.

The LSM6DS3 is available in a plastic land grid array (LGA) package.

Table 1. Device summary

Part number	Temperature range [°C]	Package	Packing
LSM6DS3	-40 to +85	LGA-14L (2.5 x 3 x 0.83 mm)	Tray
LSM6DS3TR	-40 to +85		Tape & Reel



## APPENDICE A – Orange Pi Pc 2

Riportiamo dal sito ufficiale di Orange Pi le specifiche hardware e la definizione dell'interfaccia del dispositivo Orange Pi Pc 2.

Specifiche hardware	
Processore	<ul style="list-style-type: none"><li>• Cortex-A53 quad-core a 64 bit ad alte prestazioni H5</li><li>• Motore di accelerazione multimediale integrato</li><li>• Accelerazione Java hardware</li><li>• Coprocessore a virgola mobile hardware integrato</li></ul>
GPU	<ul style="list-style-type: none"><li>• Motore anti-aliasing 4X sovrastampato con scena completa senza utilizzo di larghezza di banda aggiuntiva</li><li>• Hexa-core Mali450 ad alte prestazioni</li><li>• OpenGL ES 2.0 / 1.1 / 1.0, OpenVG 1.1, EGL</li><li>• 40 GFlops, velocità di riempimento pixel maggiore di 2.7GPixel / s</li></ul>
Memoria (SDRAM)	DDR3 da 1 GB (condiviso con GPU)
Storage a bordo	TF card (Max. 32 GB) / NOR flash (2 MB)
Rete di bordo	Ethernet RJ45 1000M / 100M
Ingresso video	Un connettore di ingresso CSI Fotocamera: supporta interfaccia sensore CMOS YUV422 a 8 bit Supporta il protocollo CCIR656 per NTSC e PAL Supporta sensore per fotocamera con pixel SM Supporta soluzioni di acquisizione video fino a 1080p @ 30fps
Ingresso audio	MIC
Uscite video	Supporta l'uscita HDMI con HDCP Supporta HDMI CEC Supporta la funzione HDMI 3D CVBS integrata Supporta l'uscita simultanea di HDMI e CVBS
Uscita audio	Jack da 3,5 mm e HDMI

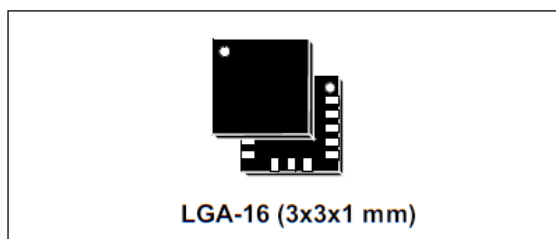
Fonte di potere	L'ingresso CC può fornire alimentazione, ma l'ingresso USB OTG non fornisce alimentazione
Porte USB 2.0	Tre HOST USB 2.0, un OTG USB 2.0
Pulsanti	Pulsante di accensione (SW4)
Periferiche a basso livello	40 pin Intestazione
PIN GPIO (1x3)	UART, terra.
GUIDATO	Led di alimentazione e led di stato
Chiave	Ingresso IR, POTENZA
Sistema operativo supportato	Android, Ubuntu, Debian

<b>Definizione dell'interfaccia</b>	
Taglia del prodotto	85 mm x 55 mm
Peso	70g



## MEMS digital output motion sensor: ultra-low-power high-performance 3-axis "nano" accelerometer

Datasheet - production data



### Features

- Wide supply voltage, 1.71 V to 3.6 V
- Independent IO supply (1.8 V) and supply voltage compatible
- Ultra-low-power mode consumption down to 2  $\mu$ A
- $\pm 2g/\pm 4g/\pm 8g/\pm 16g$  dynamically selectable full scale
- I<sup>2</sup>C/SPI digital output interface
- 16-bit data output
- 2 independent programmable interrupt generators for free-fall and motion detection
- 6D/4D orientation detection
- Free-fall detection
- Motion detection
- Embedded temperature sensor
- Embedded self-test
- Embedded 32 levels of 16-bit data output FIFO
- 10000 g high shock survivability
- ECOPACK<sup>®</sup>, RoHS and "Green" compliant

### Applications

- Motion activated functions
- Free-fall detection
- Click/double-click recognition
- Intelligent power saving for handheld devices
- Pedometers

- Display orientation
- Gaming and virtual reality input devices
- Impact recognition and logging
- Vibration monitoring and compensation

### Description

The LIS3DH is an ultra-low-power high-performance three-axis linear accelerometer belonging to the "nano" family, with digital I<sup>2</sup>C/SPI serial interface standard output. The device features ultra-low-power operational modes that allow advanced power saving and smart embedded functions.

The LIS3DH has dynamically user-selectable full scales of  $\pm 2g/\pm 4g/\pm 8g/\pm 16g$  and is capable of measuring accelerations with output data rates from 1 Hz to 5.3 kHz. The self-test capability allows the user to check the functioning of the sensor in the final application. The device may be configured to generate interrupt signals using two independent inertial wake-up/free-fall events as well as by the position of the device itself. Thresholds and timing of interrupt generators are programmable by the end user on the fly. The LIS3DH has an integrated 32-level first-in, first-out (FIFO) buffer allowing the user to store data in order to limit intervention by the host processor. The LIS3DH is available in small thin plastic land grid array package (LGA) and is guaranteed to operate over an extended temperature range from -40 °C to +85 °C.

Table 1. Device summary

Order codes	Temp. range [°C]	Package	Packaging
LIS3DHTR	-40 to +85	LGA-16	Tape and reel

## APPENDICE C – Codice completo

File: Tcp\_Server\_Graphic\_3D.pro

```
QT -= gui
QT += network
QT += core gui datavisualization

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++11 console
CONFIG -= app_bundle
TEMPLATE = app

# The following define makes your compiler emit warnings if you
# use
# any feature of Qt which as been marked deprecated (the exact
# warnings
# depend on your compiler). Please consult the documentation of
# the
# deprecated API in order to know how to port your code away from
# it.
DEFINES += QT_DEPRECATED_WARNINGS

# You can also make your code fail to compile if you use
# deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a
# certain version of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables
# all the APIs deprecated before Qt 6.0.0

SOURCES += \
    main.cpp \
    myserver.cpp \
    mygrafic3d.cpp

# Default rules for deployment.
qnx: target.path = /tmp/${TARGET}/bin
else: unix:!android: target.path = /opt/${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

HEADERS += \
    myserver.h \
    mygrafic3d.h
```

File: main.cpp

```
#include <QApplication>
#include <QtDataVisualization>
#include <QtWidgets>
#include <QtGui>

#include "myserver.h"
#include "mygrafic3d.h"

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    Q3DScatter graph;

    if (!graph.hasContext())
    {
        qWarning() << QStringLiteral("Impossibile inizializzare il
                                     contesto OpenGL!");
        return -1;
    }

    graph.setFlags(graph.flags() ^ Qt::FramelessWindowHint);
    graph.resize(800, 500);
    graph.setTitle("TcpServerGraphic3D : Brunato Lorenzo");

    mygrafic3d mygrafic3d (&graph);
    MyServer server;

    graph.show();

    return app.exec();
}
```

File: myserver.h

```
#ifndef MYSERVER_H
#define MYSERVER_H

#include <QDebug>
#include <QObject>
#include <QTcpServer>
#include <QTcpSocket>
#include <QtDataVisualization>
#include <QVector>

class MyServer : public QObject
{
    Q_OBJECT

public:

    explicit MyServer(QObject *parent = nullptr);
    QByteArray mykeypass;
    int controllo;
    char socketread[50];
    char messaggio[50];

public slots:

    void newConnection();
    void ascolto_client();
    void identifica_messaggio();

signals:

    void signal_data(const QByteArray &);
    void signal_resetarray();

private:

    QTcpServer *server;
    QTcpSocket *socket;
};

#endif // MYSERVER_H
```

File: myserver.cpp

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <QByteArray>
#include <QCoreApplication>
#include <QObject>
#include <QtDataVisualization>
#include <QThread>

#include "myserver.h"
#include "mygrafic3d.h"

#define PORTA 5000

MyServer::MyServer(QObject *parent) : QObject(parent)
{
    controllo = 0;
    server = new QTcpServer(this);

    //non appena arrivano nuove connessioni esegue newConnection()
    connect (server, SIGNAL(newConnection()), this,
SLOT(newConnection()));

    //mette in ascolto il server sulla porta definita sopra
    if (!server->listen(QHostAddress::Any, PORTA))
    {
        qDebug() << "Il Server non può partire!";
    }
    else
    {
        qDebug() << "Server partito!";
    }
}

void MyServer::newConnection()
{
    //Aggancio il socket al server (bind)
    socket = server->nextPendingConnection();

    //mando un messaggio di benvenuto al client
    socket->write("Ciao client! Inserisci una stringa:");
    socket->flush();

    connect(socket, SIGNAL(readyRead()), this,
        SLOT(ascolto_client()));
}
```

```

void MyServer::ascolto_client()
{
    if (controllo == 0)
    {
        controllo = 1;
        mykeypass = socket->readAll();
        socket->flush();
        qDebug() << "";
        qDebug() << "Client connesso e pronto a inviare i
campioni";
    }
    else
    {
        //Leggo e mostro cosa ha scritto il client
        mykeypass = socket->readAll();
        socket->flush();
        qDebug() << "mykeypass: " << mykeypass;

        identifica_messaggio();

        if (strcmp(messaggio, "C") == 0)
        {
            int campioni = atoi(socketread);
            qDebug() << "";
            qDebug() << "Numero di campioni inviati dal client: "
                << campioni;
            emit signal_resetarray();
        }
        else if (strcmp(messaggio, "A") == 0)
        {
            emit signal_data(socketread);
        }
        else
        {
            qDebug() << "ERRORE: stringa non riconosciuta!!!";
        }
    }
}

```



```

void MyServer::identifica_messaggio()
{
    strcpy(socketread, mykeypass);
    int i = 0;
    int j = 0;

    while (socketread[i] != ':')
    {
        messaggio[j] = socketread[i];
        i++;
        j++;
    }
    messaggio[j] = '\0';

    i++;
    i++;
    j = 0;

    while (socketread[i] != '\0')
    {
        socketread[j] = socketread[i];
        i++;
        j++;
    }
    socketread[j] = '\0';
}

```

File: mygrafic3d.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QtDataVisualization>
#include <QDebug>
#include <QObject>
#include <QTcpSocket>
#include <QThread>
#include <QtCore>

#include "myserver.h"

using namespace QtDataVisualization;

struct valori
{
    float accelerazione;
    float velocita;
    float posizione;
};

class mygrafic3d : public QObject
{
    Q_OBJECT

public:

    mygrafic3d(Q3DScatter *scatter);
    ~mygrafic3d();

    char dato[50];
    float acc_x, acc_y, acc_z;
    float xpos, ypos, zpos;

    void estrai_dati();
    void addData();

    float calcolo_posizione_da_accelerazioni (float acc_letta,
        float acc_prec, float vel_prec, float pos_prec, int verso);
    float moto_uniformemente_accelerato_velocita (float a, float
        v0);
    float moto_uniformemente_accelerato_posizione (float a, float
        v0, float s0);

    MyServer server;
```

```

public slots:

    void slot_data(const QByteArray &);
    void slot_resetarray();

private:

    QScatter3DSeries *m_series;
    Q3DScatter *m_graph;
    QScatterDataArray *m_array;
    QAbstract3DSeries::Mesh m_style;
    int m_fontSize;
    bool m_smooth;
};

#endif // MAINWINDOW_H

```

File: mygrafic3d.cpp

```
#include <stdio.h>
#include <stdlib.h>

#include <QtDataVisualization>
#include <QtWidgets>
#include <QDialog>
#include <QtCore>
#include <QObject>

#include "mygrafic3d.h"
#include "myserver.h"

#define DIM 10
#define EPSILON 0.002
#define G 9.81
#define RANGE 20
#define t 1

using namespace QtDataVisualization;

mygrafic3d::mygrafic3d(Q3DScatter *scatter)
    : m_graph(scatter),
      m_fontSize(40.0f),
      m_style(QAbstract3DSeries::MeshSphere),
      m_smooth(true),
      m_series(new QScatter3DSeries),
      m_array(new QScatterDataArray)
{
    //Modifiche visive
    m_graph->activeTheme()->setType(Q3DTheme::ThemeQt);
    QFont font = m_graph->activeTheme()->font();
    font.setPointSize(m_fontSize);
    m_graph->activeTheme()->setFont(font);
    m_graph-
>setShadowQuality(QAbstract3DGraph::ShadowQualitySoftLow);
    m_graph->scene()->activeCamera()-
>setCameraPreset(Q3DCamera::CameraPresetFront);

    m_graph->axisX()->setTitle("Y");
    m_graph->axisY()->setTitle("Z");
    m_graph->axisZ()->setTitle("X");

    m_graph->axisX()->setTitleVisible(true);
    m_graph->axisY()->setTitleVisible(true);
    m_graph->axisZ()->setTitleVisible(true);

    m_graph->axisX()->setRange(-RANGE, RANGE);
    m_graph->axisY()->setRange(-RANGE, RANGE);
    m_graph->axisZ()->setRange(-RANGE, RANGE);
```

```

//creo un QScatterDataProxy e il QScatter3DSeries associato
QScatterDataProxy *proxy = new QScatterDataProxy;
m_series = new QScatter3DSeries(proxy);
m_series->setItemLabelFormat(QStringLiteral("@zTitle: @zLabel
                                     @xTitle: @xLabel @yTitle: @yLabel"));
m_series->setMeshSmooth(m_smooth);
m_graph->addSeries(m_series);

connect(&server, &MyServer::signal_resetarray, this,
        &mygrafic3d::slot_resetarray);
connect(&server, &MyServer::signal_data, this,
        &mygrafic3d::slot_data);
}

mygrafic3d::~mygrafic3d()
{
    delete m_graph;
}

void mygrafic3d::slot_resetarray()
{
    m_series->dataProxy()->resetArray(new QScatterDataArray);
}

void mygrafic3d::slot_data(const QByteArray &stringa)
{
    strcpy(dato, stringa);
    estrai_dati();
    addData();
}

```

```

void mygrafic3d::estrai_dati()
{
    char asseX[DIM], asseY[DIM], asseZ[DIM];
    int i = 0;
    int j = 0;

    struct valori X = {0, 0, 0};
    struct valori Y = {0, 0, 0};
    struct valori Z = {0, 0, 0};

    while (dato[i] != ' ')
    {
        asseX[j] = dato[i];
        j++;
        i++;
    }
    asseX[j] = '\0';
    i++;
    j = 0;

    while (dato[i] != ' ')
    {
        asseY[j] = dato[i];
        j++;
        i++;
    }
    asseY[j] = '\0';
    i++;
    j = 0;

    while (dato[i] != ' ')
    {
        asseZ[j] = dato[i];
        j++;
        i++;
    }
    asseZ[j] = '\0';

    acc_x = std::stof(asseX);
    acc_y = std::stof(asseY);
    acc_z = std::stof(asseZ);

    qDebug() << "Xacc: " << acc_x << "Yacc: " << acc_y << "Zacc: "
              << acc_z;

    xpos = calcolo_posizione_da_accelerazioni (acc_x,
        X.accelerazione, X.velocita, X.posizione, 1);
    ypos = calcolo_posizione_da_accelerazioni (acc_y,
        Y.accelerazione, Y.velocita, Y.posizione, 1);
    zpos = calcolo_posizione_da_accelerazioni (acc_z,
        Z.accelerazione, Z.velocita, Z.posizione, -1);
}

```

```

qDebug() << "Xpos: " << xpos << " Ypos: " << ypos << " Zpos: "
        << zpos;
qDebug() << "-----"
        "-----";
}

void mygrafic3d::addData()
{
    QScatterDataArray *array = new QScatterDataArray(*m_series
        ->dataProxy()->array());
    m_series->dataProxy()->resetArray(Q_NULLPTR);
    array->append(QScatterDataItem(QVector3D(ypos, zpos, xpos)));
    m_series->dataProxy()->resetArray(array);
}

float mygrafic3d::calcolo_posizione_da_accelerazioni (float
acc_letta, float acc_prec, float vel_prec, float pos_prec, int
verso)
{
    /*se il valore la differenza è maggiore di EPSILON mi muovo
    quindi calcolo i nuovi valori altrimenti non mi muovo quindi
    mantengo i vecchi valori*/

    float acc_nuova, delta_acc;

    acc_nuova = acc_letta * G;
    delta_acc = (acc_nuova - acc_prec) * verso;
    acc_prec = delta_acc;

    if (fabs(delta_acc) > EPSILON)
    {
        vel_prec = moto_uniformemente_accelerato_velocita
            (acc_prec, vel_prec);
        pos_prec = moto_uniformemente_accelerato_posizione
            (acc_prec, vel_prec, pos_prec);
    }
    else
    {
        vel_prec = 0.0;
        pos_prec = pos_prec;
    }

    return pos_prec;
}

```

```

float mygrafic3d::moto_uniformemente_accelerato_velocita (float a,
float v0)
{
    float v;

    /*calcola la velocità*/
    v = v0 + a * t;

    /*aggiorno lo stato*/
    v0 = v;

    return v;
}

```

```

float mygrafic3d::moto_uniformemente_accelerato_posizione (float
a, float v0, float s0)
{
    float s;

    /*calcola la posizione*/
    s = s0 + v0 * t + 0.5 * a * t * t;

    /*aggiorno lo stato*/
    s0 = s;

    return s;
}

```



## BIBLIOGRAFIA E SITOGRAFIA

- H.M. Deitel, P.J. Deitel, C. *Corso completo di programmazione*, terza edizione, luogo, Apogeo, 2008.
- P. Mazzoldi, M. Nigro, C. Voci, *Elementi di fisica. Vol. 1: Meccanica, termodinamica*, seconda edizione, luogo, Edises, 2007.
- P. L. Montessoro, materiale didattico del corso: *Reti dei calcolatori*, Università degli studi di Udine, a.a. 2017/2018.
- P. L. Montessoro, D. Pierattoni, materiale didattico del corso: *Fondamenti di informatica. Linguaggio C*, Università degli studi di Udine, 2001.
- Sito ufficiale Qt: <https://www.qt.io/>
- Sito ufficiale Orange Pi: <http://www.orangepi.org/>
- Sito ufficiale St: [https://www.st.com/content/st\\_com/en.html](https://www.st.com/content/st_com/en.html)

## RINGRAZIAMENTI

Eccomi giunto alla fine di questa tesi sperimentale di laurea triennale: sono stati mesi intensi, nei quali ho approfondito argomenti già studiati nel mio percorso di studi e ho potuto scoprirne e studiarne di nuovi.

Voglio innanzitutto ringraziare il mio relatore di tesi, il professor Pier Luca Montessoro, senza di lui questo progetto non sarebbe stato possibile. Lo ringrazio per la sua professionalità, i consigli e gli strumenti che mi ha fornito durante il percorso di tesi e durante gli anni di studio all'Università degli studi di Udine.

Per lo sviluppo della tesi ringrazio anche il professor Luca Di Gaspero che ha consigliato la scelta del programma *Qt Creator* e per i consigli di gestione della parte grafica.

Ringrazio i miei genitori, Ginevra e Michele, mia sorella Sofia e i miei famigliari per il sostegno e il supporto che mi hanno dato in questi anni di studio.

Un particolare ringraziamento a mia zia Lucia, che ha reso possibile la stampa e la rilegatura di questa tesi di laurea.

Ringrazio tutti i miei amici che in questi anni si sono interessati ai miei studi e coi quali ho potuto scambiare consigli ed opinioni.

Ringrazio tutti i professori della laurea triennale di Ingegneria elettronica dell'Università degli studi di Udine che hanno contribuito alla mia formazione personale.

Infine, ringrazio il professor Luciano Picone per il prezioso supporto tecnico e morale che mi ha fornito in questi anni di studio.



