1 Prefazione

1.1 scopo del progetto

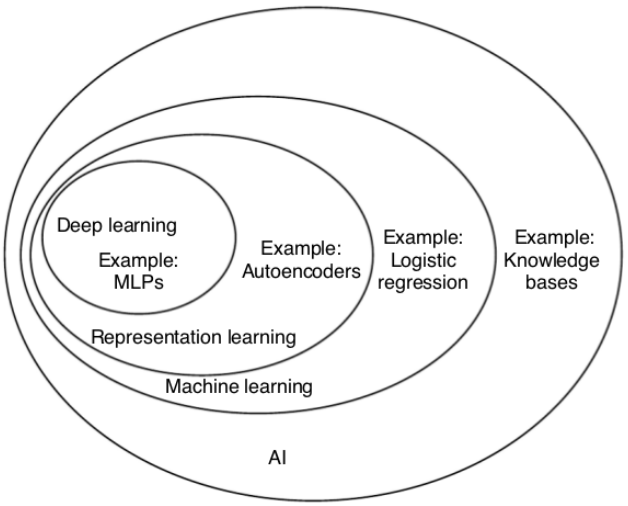
L'obbiettivo di questo progetto è la realizzazione di una modello di machine learning in grado di apprendere le relazioni tra i dati fruiti da una scheda embeded dotata di accelerometro, giroscopio e magnetometro, e la posizione spaziale di un braccio umano. Il problema che ci si è posti di risolvere poteva essere risolto in modo deterministico ma per lo scopo del progetto si voleva risolvere il problema tramite un algoritmo di ML, per poter verificare l'efficacia di questo metodo per la risoluzione di un problema con soluzione nota.

1.2 introduzione generale alle tecniche utilizzate

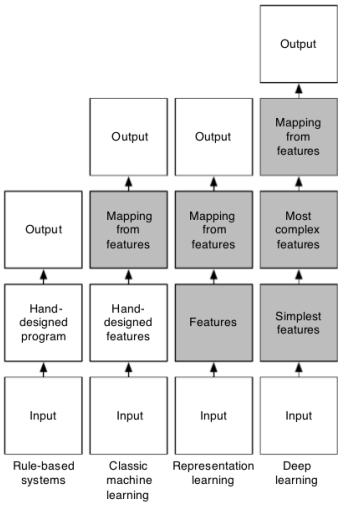
Per la realizzazione del progetto è stata realizzata una scheda con microcontrollore arduino, provvista di bluetooth, accelerometro, magnetometro, alimentata a batteria. É stata realizzata una libreria in C++/CUDA per la creazione, l'addestramento e l'utilizzo di una rete neuronale con supporto per CPU e GPU. É stato realizzato un secondo programma C++, per l'acquisizione dei dati della scheda e per l'acquisizione dei punti dello scheletro forniti dal kinect, lo stesso programma si occupa della sincronizzazione di tali dati e della creazione di un dataset necessario per il processo di addestramento della rete neuronale. In fine è stato realizzato un programma C# che sfrutta il game engine Unity per la realizzazione dell'ambiente grafico, che rende possibile la visualizzazione dell'output della rete, muovendo un manichino secondo gli input della scheda. Quest'ultima operazione è stata effettuata mettendo in comunicazione il programma C++ che si occuppa dell'acquisizione dei dati della scheda, e che ne esegue l'input nel modello addestrato, per poi passare i dati elaborati tramite comunicazione socket interna al motore grafico.

0.1 Dal machine learning al deep learning

Oggi l'intelligenza artificiale è un fiorente campo di ricerca, con l'obiettivo di risolvere una grande varietà di problemi, che per essere risolti attraverso la programmazione classica necessitano di una grande quantità di conoscenze pregresse, spesso non disponibili.
I Programmi basati sul paradigma dell'intelligenza artificiale si propongono di superare questi ostacoli acquisendo direttamente queste conoscenze dai dati grezzi, tale capacità è nota come machine learning. Sotto questa famiglia di algoritmi si trovano altri sottogruppi quali il representation learning e all'interno di quest'ultimo il deep learning.
Il deep learning rispetto ai metodi più classici, tipicamente in grado di riconoscere relazioni lineari(come ad esempio il noto SVM o support vector machine), si propone come alternativa per l'apprendimento di funzioni a molte variabili non lineari anche molto complesse. Le applicazioni più comuni comprendono il riconoscimento di oggetti nelle immagini, delle parole in tracce audio o anche per le traduzioni multilingue.
Il termine "deep learning" deriva proprio dalla capacità di questi modelli di riuscire a cogliere delle relazioni molto "profonde" tra i dati di ingresso e di uscita, approssimando con un certo errore ,dipendente dal caso specifico, la funzione che dati gli input restituisce l'output desiderato, purtroppo però presentano il grande problema di non rendere disponibile in maniera chiara le relazioni apprese.



famiglie di algoritmi



differenze generali tra
le diverse tipologie di algoritmi

0.1.1 Le reti feed-forward (MLP)

Uno dei modelli della famiglia del deep learning più comuni e semplici da utilizzare sono le reti neurali feed-forward o multi layer perceptron, le quali sono assimilabili a modelli di regressione statistica.
Le reti neurali sono un modello matematico molto semplificato del cervello, e per parlarne è necessario qualche riferimento al modello biologico.

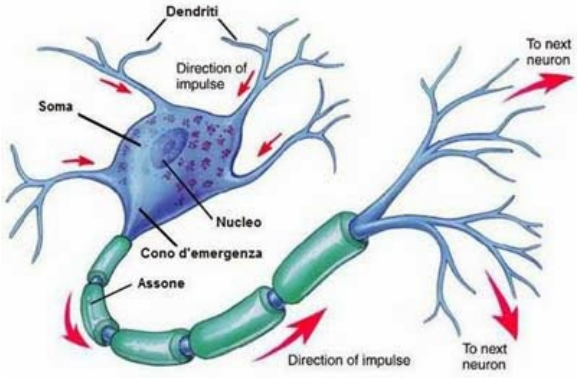


Figura 2: Il neurone biologico

Il neurone biologico

L'unità base del cervello è rappresentata dai neuroni (Figuara 2),
i quali a loro volta sono composti dai dendriti, dal soma, dall'assone e dalle sinapsi.
I dendriti sono l'apparato di input del neurone, attraverso il quale riceve i segnali dall'ambiente o da altri neuroni, tali segnali caricano il neurone, che accumula un potenziale elettrico all'interno del soma. In determinate condizioni o al raggiungimento di una certa soglia di potenziale il neurone emette un impulso lungo l'assone giungendo alle sinapsi, poste alla fine dell'assone, le quali sono connesse con altri neuroni o con altri apparati del corpo, come i muscoli.
Questa particolare cellula oltre a scambiare impulsi è in grado di accumulare informazioni attraverso l'ispessimento o l'assottigliamento della guaina mielinica, la quale è presente lungo l'assone. Le variazioni dello spessore della guaina comportano un aumento o una diminuzione della resistenza elettrica dell'assone determinando una variazione dell'impulso percepito dai neuroni connessi alle sinapsi a parità di potenziale rilasciato.

Il percettrone

Il neurone artificiale è composto da una serie di connessioni in ingresso, come i dendriti nel caso del neurone biologico, questa volta però il compito di immagazzinare informazione viene svolto da queste connessioni in input e non più dalle connessioni in output. Ad ogni input del neurone è associato un coefficiente che moltiplica il potenziale in ingresso, a questo punto gli ingressi pesati giungono al nodo sommatore, il corrispettivo del soma, che ne accumula la sommatoria. Lo step successivo è il blocco contenente la funzione di trasferimento, una funzione che prende in input il potenziale del neurone e rende disponibile il risultato come output finale o come input per i neuroni successivi. La funzione di trasferimento del neurone può essere di vari tipi, può variare infatti anche tra i neuroni della medesima rete, alcune delle più comuni sono:
(a) la funzione gradino.
(b) la funzione lineare con saturazione
(c) la funzione sigmoide tra 0 e 1
(d) la funzione sigmoide tra -1 e 1

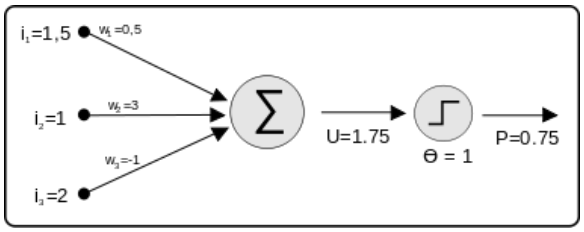


Figura 3: Il percettrone

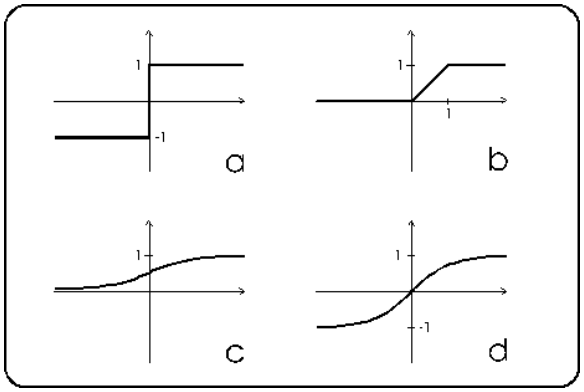


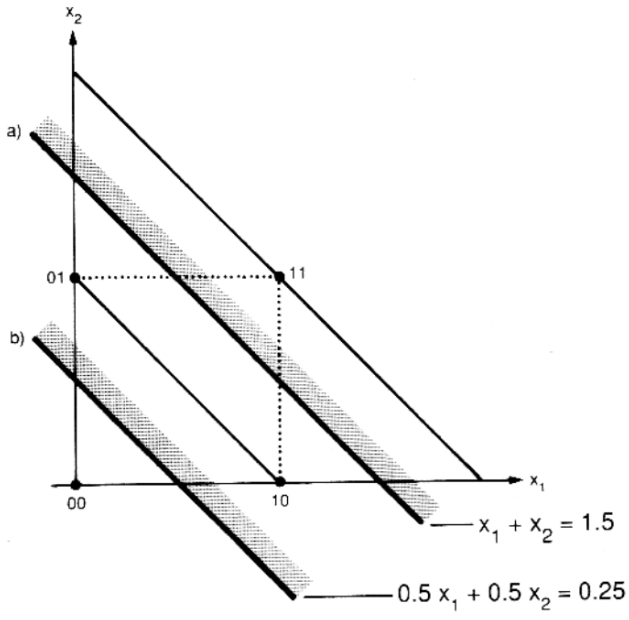
Figura 4: Le funzioni di trasferimento

Volendo esprimere il percettrone in termini matematici si ottiene la seguente equazione:

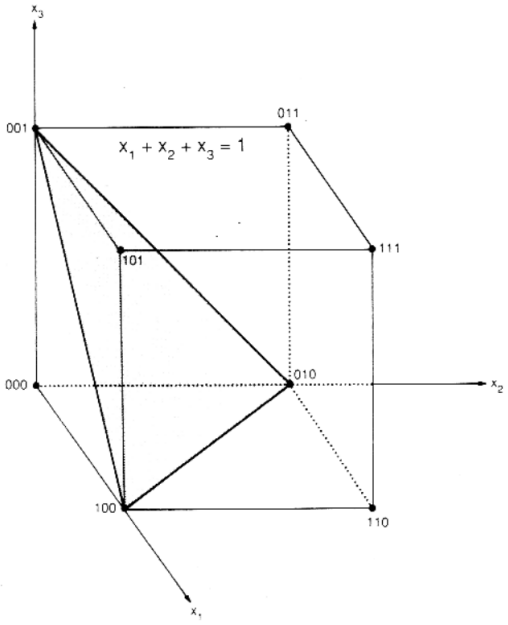
$$y = f(P) = f(\sum_i W_i \cdot X_i)$$

y - output del percettrone
f - funzione di trasferimento del percettrone
P - potenziale del neurone
W_i - peso o coefficiente dell'i-esimo ingresso
X_i - potenziale d'ingresso dell'i-esimo neurone

Una delle caratteristiche più importanti del percettrone è la sua capacità di compiere separazioni lineari. Tali separazioni avvengono in uno spazio a n dimensioni, dove n corrisponde al numero di variabili in ingresso al neurone, perciò si avrà ad esempio per un percettrone a due input la capacita di separare i valori su $\Re^2$ attraverso un retta, nel caso $\Re^3$ si potranno separare i valori dello spazio tridimensionale attraverso un piano, e così via.



esempio di linee di separazione per neurone a 2 ingressi binari
(a - funzione OR)
(b - funzione AND)



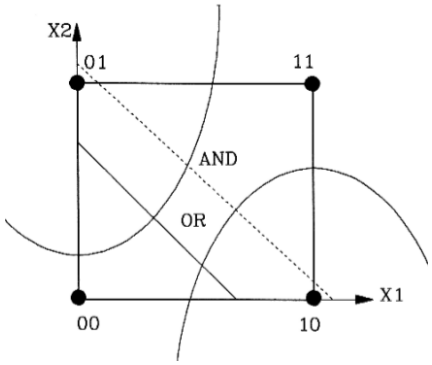
esempio di piano diseparazione lineare
a 3 ingressi per un neurone binario

Le reti multi-strato e la separazione non lineare

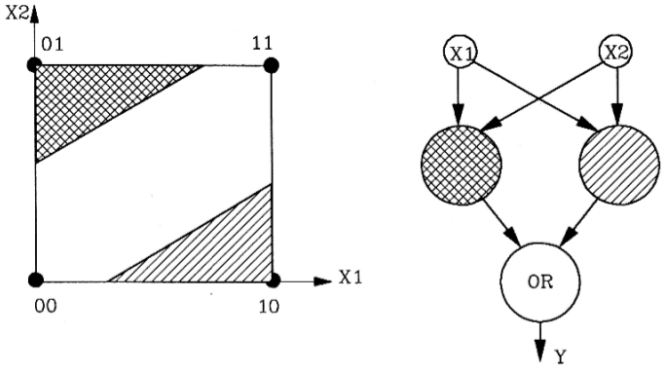
Le reti multi-strato sono composte da strati di percettroni, interconnessi tra loro in modo che gli output degli strati precedenti siano connessi con gli input dei percettroni successivi. Tale struttura "profonda" permette di rappresentare funzioni non linerai molto complesse, tanto più sono gli strati della rete.
Uno degli esempi che meglio chiarisce questa problematica è la funzione xor nel seguente esempio, la quale per semplicità è rappresentata con neuroni binari a soglia.

Il problema dello XOR

Lo xor è una delle funzioni non lineari più semplice, nella quale diventa evidente l'impossibilità di approssimare tale funzione con un solo strato di neuroni. Il grafico illustra come un neurone singolo possa effettuare la separazione lineare che riproduce la funzione and e or, ma come si nota lo xor necessità di due parabole per la separazione e non più di rette, tale separazione si effettua utilizzando una rete ad almeno due strati.



Esempio di linee di separazione per and or e xor



MLP in grado di rappresentare la funzione xor

0.1.2 La funzione di addestramento: Il back-propagation

Come già detto la struttura della rete neurale deve apprendere una data funzione da dei dati.
Tale operazione si effettua presentando degli input alla rete ai quali questa risponderà con degli output, i quali dovranno essere confrontati con degli output corretti per il set di input presentato, per calcolare l'errore nell'output della rete.
L'errore che la rete commette ad ogni tentativo viene utilizzato per calcolare attraverso una prescelta funzione, nel nostro caso il BP, dei coefficienti di correzione che vanno applicati ai pesi della rete per avvicinare il suo output a quello desiderato.
Come dice il nome "retropropagazione" in italiano, l'errore calcolato dall'output viene applicato ai neuroni precedenti e poi a quelli ancora precedenti fino ad arrivare al primo strato, tale procedura puo essere effettuata in diversi modi, calcolando gli errori su un determinato numero di esempi del dataset, accumulando le correzioni ed applicandole successivamente, tale tecnica è attualmente la più utilizzata ed è detta "mini-batch", ma è possibile anche effettuare la correzione esempio per esempio, e questo è detto addestramento in-line, in fine tra i metodi più comuni c'è anche la possibilità di eseguire tutti gli esempi del dataset e solo alla fine di un'intero ciclo di addestramento applicare le correzioni. questi tre metodi ovviamente risultano validi ma a seconda del problema da risolvere possono avere prestazioni migliori o peggiori in modo difficilmente prevedibile.
Dopo questa introduzione generale all'addestramento delle reti passiamo a dare una vista più accurata dell'algoritmo utilizzato nel progetto che è principalmente pensato per reti mlp con neuroni ad ingressi continui e con funzione di trasferimento sigmoidea o ad arcotangente.
Il BP cerca di minimizzare l'errore quadratico medio, relativo a un training set di esempi dati, scendendo lungo la superficie d'errore e seguendo la direzione di massima pendenza, in cerca di una "valle" sufficientemente profonda.
Consideriamo una rete con n input X_i (i = 1, 2,.. n), uno strato nascosto di q neuroni Z_k (k = 1, 2..., q) e uno strato output di m neuroni Y_j; (j = 1, 2... m). Il training set sia costituito da p esempi o casi C_r, (r = 1, 2,..,p). Tutti i neuroni abbiano la stessa funzione di trasferimento.
L'errore quadratico medio dello strato output è:

$$E = \frac{1}{2} \sum_j \sum_r (Y_{rj} - D_{rj})^2 \tag{1}$$

dove Y, è l'output del neurone Y_j alla presentazione dell'esempio C_r e D_{rj} è il suo valore desiderato. Per semplificare la notazione, eliminiamo l'indice r e minimizziamo l'errore quadratico medio ad ogni presentazione di esempio (modalità on-line), anziché dopo ogni ciclo completo di presentazione del training set o epoca (modalità batch). Avremo allora:

$$E = \frac{1}{2}\sum_j (Y_j - D_j)^2 \tag{2}$$

Si applica a questo punto la regola del gradiente (fig. Propagazione a):

$$\Delta W_{jk} = -\eta \frac{\partial E}{\partial W_{jk}} = -\eta \frac{\partial E}{\partial Y_j}\frac{\partial Y_j}{\partial P_j}\frac{\partial P_j}{\partial W_{jk}} = -\eta (Y_j - D_j) f'(P_j) Z_k \tag{3}$$

$$\frac{\partial E}{\partial Y_j} = (Y_j - D_j) \tag{4}$$

$$\frac{\partial Y_j}{\partial P_j} = f'(P_j) \tag{5}$$

$$\frac{\partial P_j}{\partial W_{jk}} = \frac{\partial(\sum W_{jk} Z_k)}{\partial W_{jk}} = Z_k \tag{6}$$

Ponendo a questo punto:

$$\delta_j = (Y_j - D_j) f'(P_j) \tag{7}$$

$$\Delta W_{jk} = -\eta\, \delta_j\, Z_k \tag{8}$$

La formula (**??**) viene impiegata per aggiornare i pesi sinattici delle connessioni tra strato nascosto e strato output, analogamente, per quanto riguarda le connessioni tra strato input e strato nascosto si utilizza la formula (fig. Propagazione b):

$$\Delta W_{ki} = -\eta \frac{\partial E}{\partial W_{ki}} = -\eta \frac{\partial E}{\partial Z_k}\frac{\partial Z_k}{\partial P_k}\frac{\partial P_k}{\partial W_{ki}} = -\eta \frac{\partial E}{\partial Z_k} f'(P_k) X_i \tag{9}$$

Confrontando questa formula con la precedente formula (**??**), al posto degli errori noti $(Y_j - D_j)$ troviamo le derivate $\frac{\partial E}{\partial Z_k}$, che dobbiamo ora calcolare retropropagando l'errore dallo strato output a ogni neurone nascosto:

$$\frac{\partial E}{\partial Z_k} = \sum_j \left(\frac{\partial E}{\partial Y_j}\frac{\partial Y_j}{\partial P_j}\frac{\partial P_j}{\partial Z_k} \right) = \sum_j (Y_j - D_j) f'(P_j) W_{jk} \tag{10}$$

o anche, applicando la formula (**??**) :

$$\frac{\partial E}{\partial Z_k} = \sum_j \delta_j\, W_{jk} \tag{11}$$

In fine otteniamo le equazioni:

$$\Delta W_{ki} = -\eta f'(P_k)(\sum_j \delta_j W_{jk}) X_i \tag{12}$$

che ponendo:

$$\delta_k = f'(P_k)(\sum_j \delta_j W_{jk}) \tag{13}$$

la quale assume la stessa forma della (**??**) :

$$\Delta W_{ki} = -\eta\, \delta_k\, X_i \tag{14}$$

Le formule (**??**) (**??**) (**??**) sono valide non solo per aggiornare i pesi sinattici tra strato nascosto (l'unico nella rete che abbiamo ipotizzato) e strato input ma anche, in reti più generali, per le connessioni tra due strati nascosti consecutivi. Se si adotta la funzione di trasferimento sigmoide si ha:

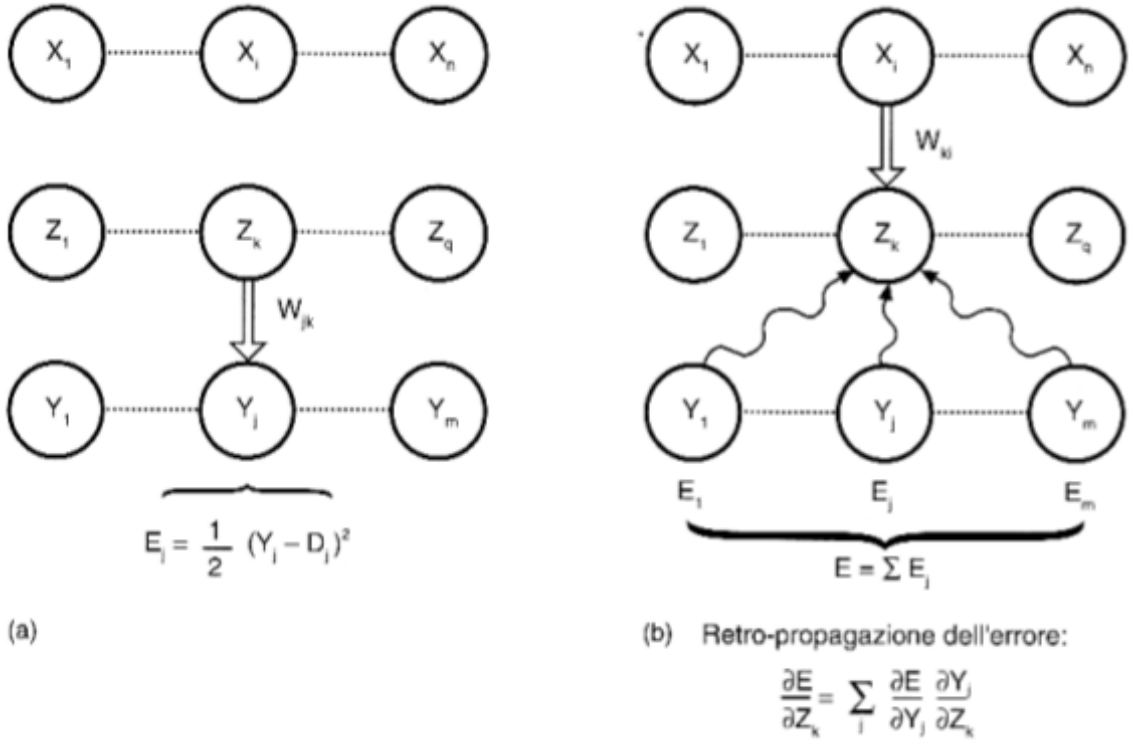
$$Y = f(P) = \frac{1}{1 + e^{kP}} \tag{15}$$

$$f'(P) = k f(P)(1 - f(P)) = kY(1 - Y) \tag{16}$$

Dove k è proporzionale alla pendenza della sigmoide nel suo punto di flessione, normalmente si pone k = 1. A questo punto otteniamo la rielaborazione delle formule (**??**) (**??**):

$$\Delta W_{jk} = -\eta k (Y_J - D_j) Y_j (1 - Y_j) Z_k \tag{17}$$

$$\Delta W_{ki} = -\eta k Z_k (1 - Z_k)(\sum_j \delta_j W_{jk}) X_i \tag{18}$$



Propagazione dell'errore nella rete

infine i pesi sinattici vengono aggiornati, ad ogni tempo t dell'intervallo 1, 2, ..., (t-1), t, (t+1), ...ecc con:

$$W_{jk}(t+1) = W_{jk}(t) + \Delta W_{jk} \tag{19}$$

$$W_{ki}(t+1) = W_{ki}(t) + \Delta W_{ki} \tag{20}$$

0.0.1 il computo parallelo

TODO

0.1 Il kinect

Il kinect è un dispositivo per la rilevazione del movimento, prodotto dall'azienda Microsoft e commercializzato insieme alla console videoludica Xbox. Esso può tuttavia essere interfacciato ad un comune pc tramite l'apposito connettore. Nel kinect è presente una fotocamera 1080p RGB, una fotocamera infrarossi per calcolare la profondità e 4 microfoni. Con questo set di sensori ed il suo SDK (software development kit), consente l'utilizzo dei dati prodotti per scopi più generici, una tra le quali l'utilizzo delle gesture per controllare il pc o la ricostruzione delle espressioni facciali. Questi dati sono resi disponibili attraverso l'apposito SDK e possono essere usati in generici programmi C++, C# ed altri linguaggi di programmazione, nello specifico per il linguaggio C++ si usa la libreria "Kinect.h".

Figura 1: Il kinect.

0.1.1 Modello dello scheletro

In particolare ci siamo interessati alla capacità del kinect di poter ricostruire la posizione delle parti del corpo nello spazio, infatti utilizzando la fotocamera infrarossi è in grado di rilevare 24 punti diversi nel corpo come indicato in figura 2.

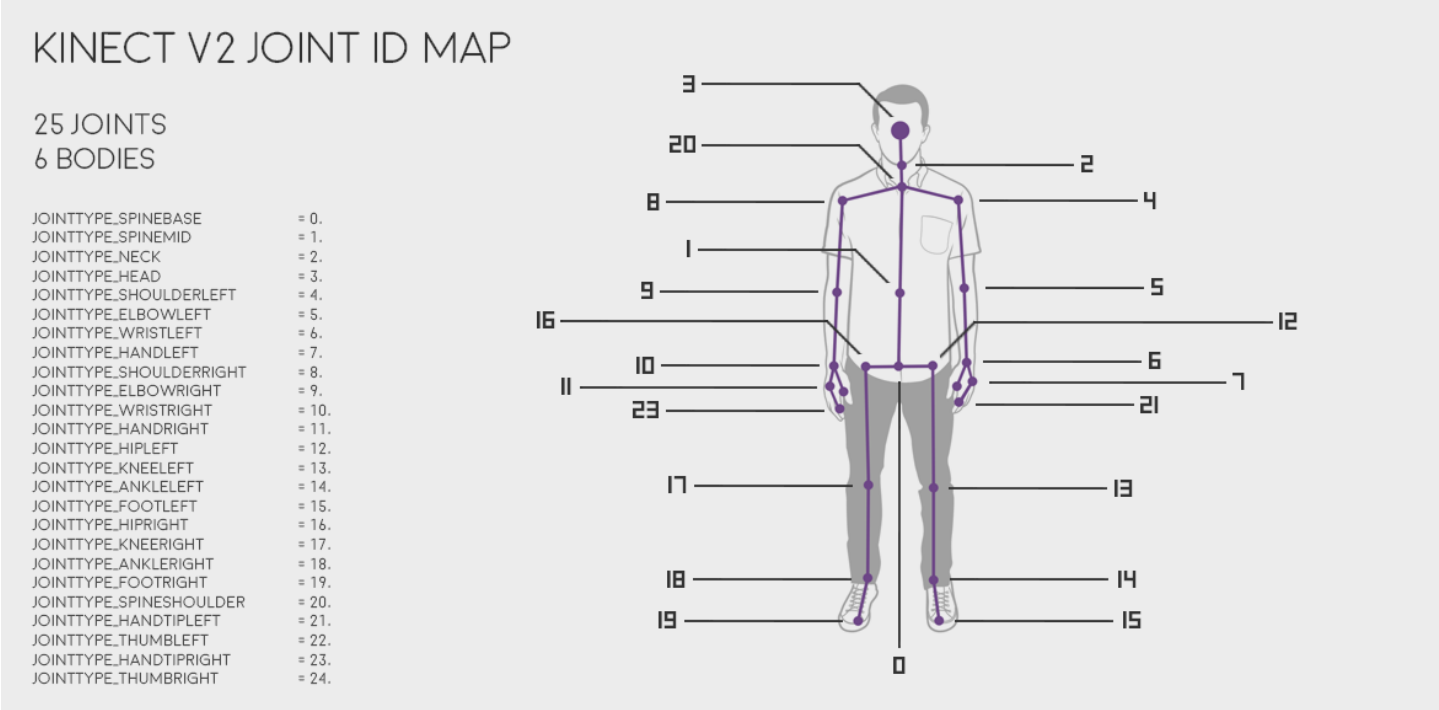


Figura 2: La mappa dei giunti che il kinect è in grado di rilevare.

I dati di posizione e profondità restituiti dal kinect fanno riferimento alla terna di orientazione indicata in figura 3. Per in nostri scopi abbiamo utilizzato delle procedure già definite dalla libreria che permettono di acquisire la posizione nello spazio di ogni singolo giunto dello scheletro e il suo stato di tracciamento, se la sua posizione è stata interpolata o è stato effettivamente tracciato.

Figura 3: Orientazione degli assi rispetto al kinect.

0.1 Scheda enbedded

Per questo progetto abbiamo realizzato una scheda enbedded che comprendesse l'imu, un modulo bluetooth, la batteria con il suo regolatore e un arduino.

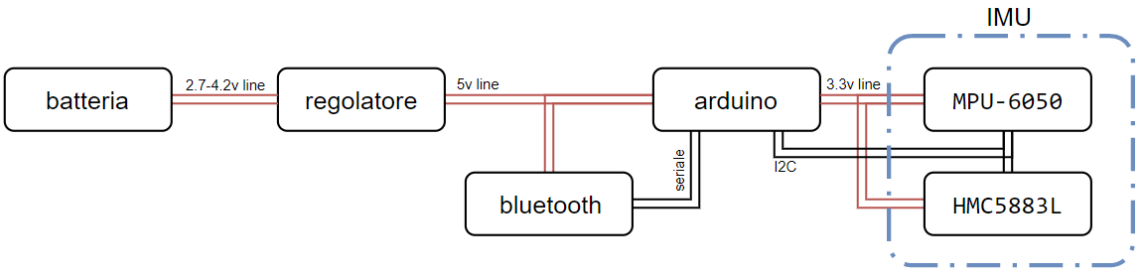


Figura 1: Schema scheda enbedded

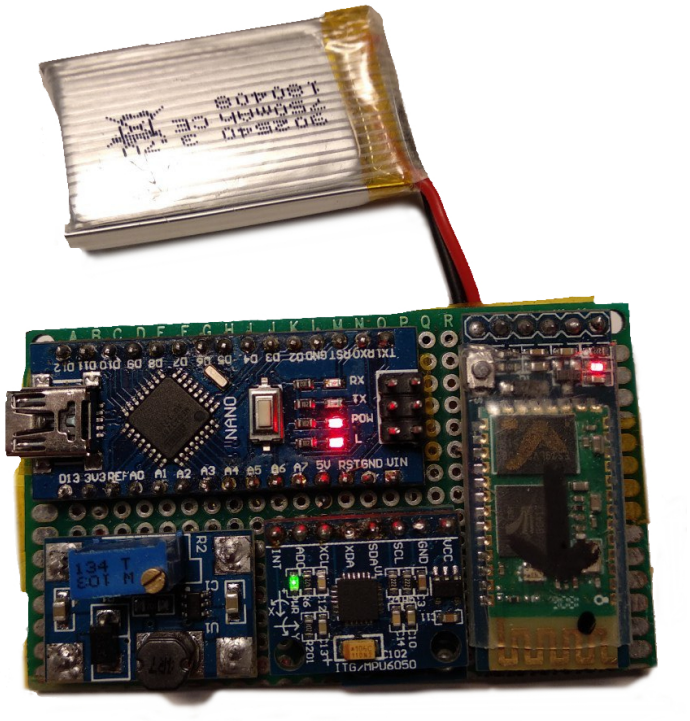


Figura 2: Scheda enbedded

TODO foto scheda
(..c'e' da dire altro?)

0.1.1 Arduino

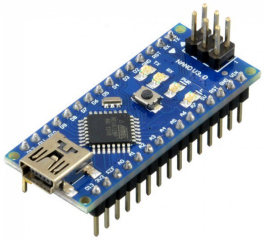


Figura 3: Arduino nano

Arduino è una piattaforma hardware open-source dotata di microcontrollore e tutto il suo ecosistema. Questo la rendono un ottima scheda per la prototipazione rapida (..da ristrutturare la frase). Nel nostro progetto abbiamo usato la scheda "Arduino nano" (fig.3). (..da aggiunge che e' alimentata a 5 V) La scheda si interfaccia al pc per essere programmata, tramite "Arduino IDE", e ha al suo interno un regolatore di tensione da 5 V a 3.3 V, utile per alimentare accelerometro e magnetometro. Inoltre ha, una sola seriale, l'I2C, ed altri pin GPIO di cui ne abbiamo usato uno per controllare il pin state del'HC-05.
TODO (altro ????????)

0.1.2 Bluetooth

Figura 4: HC-05

Il modulo bluetooth ulilizzato è "HC-05", ha 6 pin per interfacciarsi con le atre periferiche: 2 di alimentazione, 2 per la seriale, state e key. Questo modulo ha un regolatore da 5 V a 3.3 V integrato, quindi va alimentato a 5 V. Per comunicare con arduino usa la seriale (RS-232) con livello logico 3.3 V, ciò comporta la necessità di inserire un partitore sul pin rx del modulo (quindi il pin tx di arduino). Il pin state èstato usato per far conoscere ad arduino quando il modulo ha stabilito la connessione con il computer. Il pin key svolge una funzione particolare poichè permette di avviare il modulo in modalità "command mode", in questa modalità si possono usare gli AT commands per programmare il modulo (es. cambiare nome al dispositivo, oppure cambiare il baud rate della seriale). I comandi AT e la loro

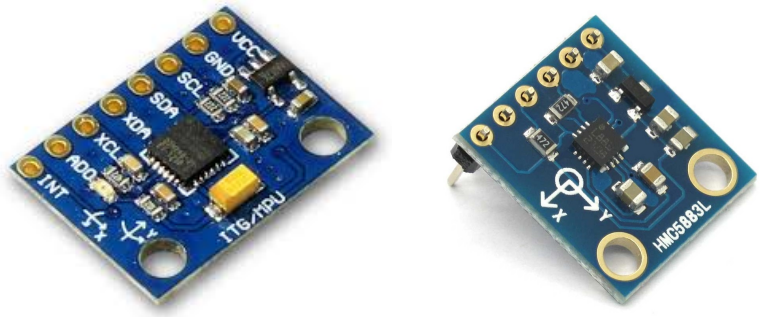
descrizione si trovano sul datasheet. Per avviare l'HC-05 in modalità AT si deve collegare il modulo all'alimentazione tenendo premuto il pulsante che si trova sulla scheda, dopodiche si può procedere ad inviare gli at command tramite seriale con un baud rate di 38400Bd. Per fare ciò abbiamo collegato la seriale del modulo con 2 pin di arduino con supporto PWM ed abbiamo usato la libreria software serial per poter avere un'altra seriale (la principale connessa al pc e la secondaria connessa al modulo). Dopodichè basta programmare arduino in modo che reindirizzi ciò che gli viene scritto dal pc al modulo, e viceversa. il programma usato è il seguente:

```
#include <SoftwareSerial.h>

SoftwareSerial BTSerial(10, 11); // RX | TX

void setup()
{
  Serial.begin(9600);
  Serial.println("Enter AT commands:");
  BTSerial.begin(38400);  // HC-05 default speed in AT command more
}

void loop()
{
  // Keep reading from HC-05 and send to Arduino Serial Monitor
  if (BTSerial.available())
    Serial.write(BTSerial.read());

  // Keep reading from Arduino Serial Monitor and send to HC-05
  if (Serial.available())
    BTSerial.write(Serial.read());
}
```

Così facendo abbiamo impostato il baud rate a 115 200 Hz.

0.1.3 L'imu

L'imu (inertial measurement unit) permette di misurare le forze ad esso applicate e l'orientazione dello stesso. Questo viene solitamente fatto combinando i dati di accelerometro, magnetometro e giroscopio. In particolare l'accelerometro misura le accelerazioni, da cui in condizioni di moto inerziale si può estrarre il vettore gravità sui 3 assi determinando quindi l'angolazione rispetto al suolo; Il magnetometro rileva invece il campo magnetico terrestre su 3 assi, dando così indicazione della direzione "nord"; Infine il giroscopio restituisce le accelerazioni angolari. Per questo specifico progetto si sono utilizzati i moduli commerciali "MPU-6050" (fig.5a) e "HMC5883L" (fig.5b), rispettivamente come accelerometro più giroscopio e magnetometro.



(a) MPU-6050 (b) HMC5883L

Figura 5: l'IMU utilizzata in questo progetto

Questi dispositivi comunicano con arduino attraverso il protocollo I2C, sono stati quindi connessi ai pin analogici 5 e 4 di arduino. In particolare l'MPU-6050 integra un accelerometro su 3 assi ed un giroscopio su 3 assi, che vengono convertiti in digitale da 6 ADC a 16 bits. Inoltre può essere programmato su diverse precisioni, il giroscopio tra ±250 °/s e ±2000 °/s, l'accelerometro tra ±2 g e ±16 g. Questo modulo ha anche un sensore di temperatura che per questo progetto non è stato usato. Supporta l'I2C fino a 400 kHz.

0.1.4 Regolatore di tensione

Questo regolatore di tensione è del tipo switching e permette di elevare la tensione da 2 V-24 V a 2 V-28 V con un picco massimo di 2 A. La tensione di uscita viene impostata girando il trimmer e qualunque sia la tensione in ingresso l'uscita rimarrà al valore impostato purchè si rispettino i limiti massimi e la tensione di ingresso sia sempre minore di quella di uscita. Nel nostro caso è stato molto utile perchè la betteria a litio ha una tensione che oscilla tra 2.7 V quando è scarica e 4.2 V quando è carica e questo integrato provvede a stabilizzare l'alimentazione a 5 V.

Figura 6: SX1308

0.1 Prefazione

Introduciamo i programmi con uno schema generale della relazione logico temporale tra i programmi: per prima cosa abbiamo dovuto acquisire un dataset ed usarlo per addesrare la rete neurale

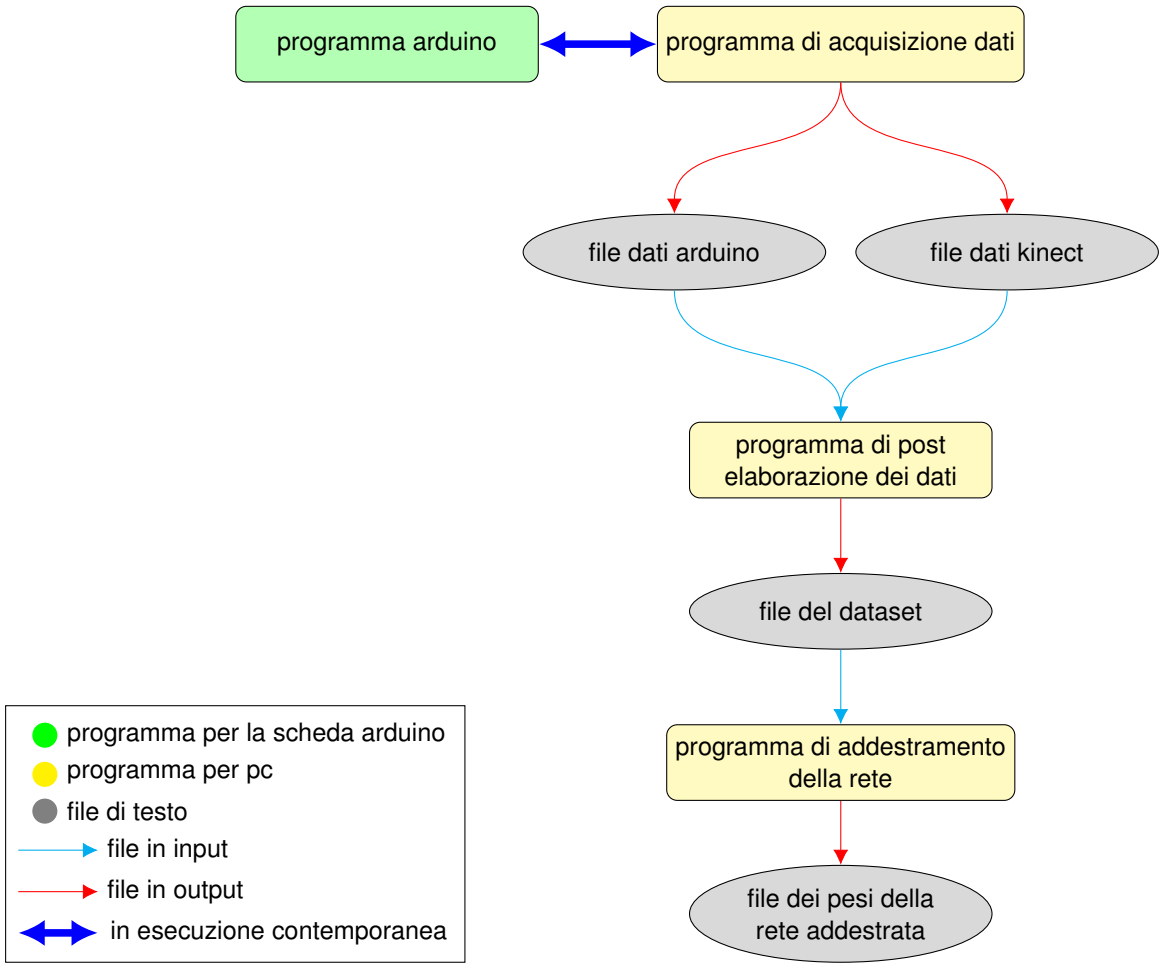


Figura 1: i programmi usati per ottenere i pesi della rete addestrata (?)

una volta addestrata la rete si possono usare i pesi così ottenuti per visualizzare in tempo reale il pupo che si muove(???)

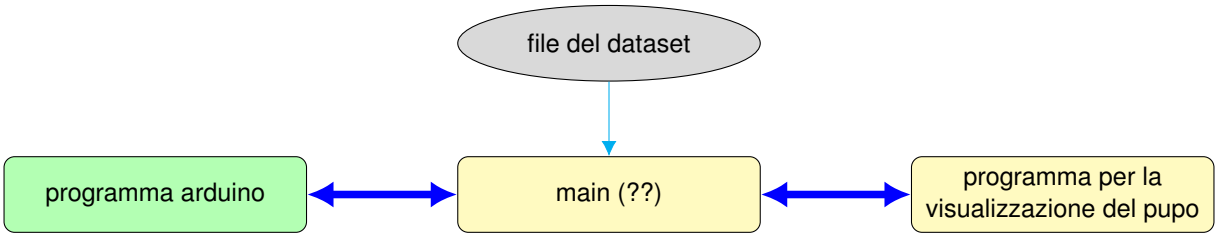


Figura 2: una volta addestrata la rete si può vedere il pupo muoversi in tempo reale

TODO aggiungere ciò che manca

0.1 Programma arduino

Questo programma si occupa di acquisire i dati dai sensori attraverso l'I2C, elaborarli e scriverli sulla seriale. Il programma si scompone nella classe "MPU0_6050" che gestisce l'accelerometro, nella classe "QMC5883" che gestisce il magnetometro, nella classe "serial" per gestire la seriale e nelle 2 funzioni standard di arduino "setup" e "loop". inoltre abbiamo usato la libreria "Wire.h" per la seriale. Nel "setup" si trova soltanto l'inilizializzazione alla seriale per comunicare con il modulo "HC-05":

```
void setup()
{
  Serial.begin(115200);
}
```

Listing 1: funzione "setup"

Nella funzione "loop" si trovano le varie dichiarazioni alle altre classi, un loop per aspettare che il modulo "HC-05" sia connesso al pc ed il ciclo infinito che provvede ad acquisire i dati dai sensori e scriverli sulla seriale:

```
void loop()
{
  // chiamo i costruttori delle 2 classi che gestiscono i sensori e provvedo al loro settaggio.
  QMC5883 mm;
  MPU_6050 aa;
  aa.setting();

  // accendo un led per il debug
  digitalWrite(13, HIGH);

  // aspetto finchè il pin "state" del modulo "HC-05" non diventa alto
  while (!digitalRead(3))
  {
    delay(10);
  }

  // quando il bluetooth è connesso posso procedere ad inizializzare e sincronizzare la seriale
  serial ser;
  ser.sinc();

  // spengo il led
  digitalWrite(13, LOW);

  while (true)
  {
    // acquisisco i dati del magnetometro e li normalizzo
    mm.get_data();
    mm.normalize(100);

    // acquisisco i dati dell'accelerometro
    aa.get_data();

    // li sposto in degli array
    float acc_xyz[] = {aa.get_AcX(), aa.get_AcY(), aa.get_AcZ()};
    float g_xyz[] = {aa.get_GyX(), aa.get_GyY(), aa.get_GyZ()};
    float magn[] = {mm.getX(), mm.getY(), mm.getZ()};

    // li scrivo sulla seriale in modo che il modulo bluetooth li trasmetta al pc.
    ser.send_data(acc_xyz, g_xyz, magn, aa.get_temp());
  }
}
```

Listing 2: funzione "loop"

Qui sono state usate le classi "MPU_6050", "QMC5883" e "serial". iniziando dalle 2 classi che gestiscono i sensori:

```
class MPU_6050
{
  public:
    int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;
    float normalized_AcX, normalized_AcY, normalized_AcZ, normalized_GyX, normalized_GyY,
        ↪ normalized_GyZ;
    float rescaled_AcX, rescaled_AcY, rescaled_AcZ, rescaled_GyX, rescaled_GyY, rescaled_GyZ;

```

```
    const uint8_t MPU = 0x68; // I2C address of the MPU-6050

    // costruttore della classe, accende l'MPU_6050
    MPU_6050(){...}

    // prede i dati dall'MPU_6050 e le sposta sulle variabili di classe.
    void get_data(){...}

    // funzioni per ritornare i valori dell'accelerometro
    float get_AcX(){ return (AcX*1.0); }
    float get_AcY(){...}
    float get_AcZ(){...}

    // funzioni per ritornare i valori del giroscopio
    float get_GyX(){ return (GyX*1.0); }
    float get_GyY(){...}
    float get_GyZ(){...}

    // funzioni per ritornare i valori normalizzati dell'accelerometro
    float get_normalized_AcX(){ return normalized_AcX; }
    float get_normalized_AcY(){...}
    float get_normalized_AcZ(){...}

    // funzioni per ritornar i valori normalizzati del giroscopio
    float get_normalized_GyX(){ return normalized_GyX; }
    float get_normalized_GyY(){...}
    float get_normalized_GyZ(){...}

    // funzioni per calcolare e ritornare la norma dell'accelerometro o del magnetometro
    float norma(float x, float y, float z){...}
    float get_norma_Gy(){ return norma(GyX, GyY, GyZ); }
    float get_norma_Ac(){...}

    // funzione per ritornare la temperatura
    float get_temp(){ return Tmp; }

    // funzioni per normalizzare l'accelerometro o il magnetometro
    void normalize_Ac(int Max){...}
    void normalize_Gy(int Max){...}

    // funzione che scrive le impostazioni all'MPU_6050
    void setting(){...}
};
```

Listing 3: classe "MPU_6050"

TODO

```
class QMC5883
{
  public:
    uint8_t add = 0x0D;
    int nowX = 0;
    int nowY = 0;
    int nowZ = 0;
    float rescaled_x = 0;
    float rescaled_y = 0;
    float rescaled_z = 0;

    // il costruttore provvede ad inizializzare il modulo e settarne i vari registri
    QMC5883(){...}

    // funzioni per ritornare i valori x-y-z del magnetometro
    float getX(){ return (nowX*1.0); }
    float getY(){...}
    float getZ(){...}

    // funzioni per ritornare i valori del magnetometro dopo averli riscalati
    float getX_rescaled(){ return rescaled_x; }
    float getY_rescaled(){...}
```

```
23      float getZ_rescaled(){...}
24
25      // funzione che provvede a leggere i regisrti del modulo per acquisire i dati
26      // spostandoli nelle variabili di classe
27      void get_data(){...}
28
29      // funzioni per calcolare la norma e ritornare i valori normalizzati
30      float norma(float x, float y, float z){...}
31      float get_norma(){ return norma(nowX, nowY, nowZ); }
32      void normalize(int Max){...}
33  };
```

Listing 4: classe "QMC5883"

TODO

```
1  class serial
2  {
3    public:
4
5      // funzione che permette di sincronizzarsi con il pc
6      void sinc(){...}
7
8      // union per poter scompattare un float in 4 byte
9      union Scomp_float{...};
10
11     // funzione che permette di inviare un intero float scompattandolo in
12     // 4 byte che vengono trasmessi serialmente
13     void send_float(float n){...}
14
15     // funzione che permette di riceve un float come sopra
16     float receive_float(){...}
17
18     // funzione per l'invio di un solo carattere
19     void send_char(char ch){...}
20
21     // funzione per la ricezione di un singolo carattere
22     char receive_char(){...}
23
24     // funzione che provvede ad inviare i dati necessari.
25     void send_data(float* acc_xyz, float* g_xyz, float* magn, float temp){...}
26 };
```

Listing 5: classe "serial"

TODO

0.1 programma per l'acquisizione dei dati dal kinect e da arduino

Questo programma è stato usato per acquisire i dati dai dispositivi e scriverli in dei file di testo. Per poter lavorare meglio abbiamo separato il programma principale in diverse librerie con scopi specifici, la lista completa delle librerie usate è:

```
// libreria usata da visual studio, da togliere in caso si usi un altro ide
#include "pch.h"

// librerie standard per i file stringhe e altro
#include <cstddef>
#include <cstdlib>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <ostream>
#include <string>
#include <math.h>

// librerie create da noi
#include "real_time.h"
#include "kin_file_manager.h"
#include "ard_file_manager.h"
#include "data_structure.h"

// libreria per usare i thread
#include <boost/thread.hpp>

// altre librerie per log, alcuni tipi di dato, un buffer FIFO
#include <boost/log/core.hpp>
#include <boost/call_traits.hpp>
#include <boost/circular_buffer.hpp>
#include <boost/container/vector.hpp>

// altre librerie per il logger
#include <boost/log/attributes.hpp>
#include <boost/log/attributes/scoped_attribute.hpp>
#include <boost/log/expressions.hpp>
#include <boost/log/sinks/sync_frontend.hpp>
#include <boost/log/sinks/text_ostream_backend.hpp>
#include <boost/log/sources/basic_logger.hpp>
#include <boost/log/sources/record_ostream.hpp>
#include <boost/log/sources/severity_logger.hpp>
#include <boost/log/utility/setup/common_attributes.hpp>
#include <boost/log/utility/setup/console.hpp>
#include <boost/log/utility/setup/file.hpp>

#include <boost/smart_ptr/make_shared_object.hpp>
#include <boost/smart_ptr/shared_ptr.hpp>

// altre librerie per i thread
#include <boost/thread/condition_variable.hpp>
#include <boost/thread/mutex.hpp>
#include <boost/thread/thread.hpp>

// librereria per il tempo
#include <boost/date_time/posix_time/posix_time.hpp>

// libreria fatta da noi per gestire la seriale
#include "Serial_handler.h"

// librerie di Windows e kinect
#include <Windows.h>
#include <Kinect.h>
#include <Shlobj.h>

// namespace standard
using namespace std;
```

Listing 1: librerie usate

Le librerie fatte da noi sono: "real_time.h", "kin_file_manager.h", "ard_file_manager.h", "data_structure.h", "Serial_handler.h". In "real_time.h" ci sono alcune funzioni per prendere il tempo e misurarlo:

```
#pragma once

#ifndef _REAL_TIME_H_
#define _REAL_TIME_H_

#include <boost/chrono.hpp>
#include <boost/timer/timer.hpp>

class real_time
{
private:
  boost::timer::cpu_timer timer;
  boost::timer::cpu_times t;

public:
  /// <summary>
  /// start the time counting
  /// </summary>
  /// <returns></returns>
  void start();

  /// <summary>
  /// return enlapsed time in millisecond
  /// </summary>
  /// <returns></returns>
  float stop();

  /// <summary>
  /// get the time in millisecond
  /// </summary>
  /// <returns></returns>
  uint64_t get_curr_time();

};

#endif // #ifndef _REAL_TIME_H_
```

Listing 2: librerie usate

In "kin_file_manager.h" ci sono le funzioni per gestire il file di testo relativo al kinect:

```
#pragma once

#ifndef _KIN_FILE_MANAGER_H_
#define _KIN_FILE_MANAGER_H_

#include <iostream>
#include <ostream>
#include <fstream>
#include <string>

// in questa libreria sono dichiarate le varie strutture dati
#include "data_structure.h"

class kin_file_manager
{
private:
  std::fstream f;
  std::ios::_Openmode mode;
  unsigned long int line_id = 0;

public:
  // il costruttore provvede a settare il nome del file e la modalit\'a di apertura
  kin_file_manager(std::string file_name, std::ios::_Openmode mode);

  // questa funzione provvede a scrivere un singolo blocco di dati nel file
  void write_data_line(kinect_data dat);
```

```
27
28    // funzione che legge un blocco di dati dal file e lo sposta nella struttura dati
29    // se si \'e raggiunta la fine del file la funzione ritorna 0, altrimenti ritorna 1.
30    bool read_data_line(kinect_data* dat);
31
32    // distruttore della classe, chiude il file
33    ~kin_file_manager()
34    {
35    f.close();
36    }
37
38    // funzione che chiude il file
39    void close()
40    {
41    f.close();
42    }
43
44 };
45
46 #endif // #ifndef _KIN_FILE_MANAGER_H_
```

Listing 3: librerie usate

Stessa cosa per "ard_file_manager.h" che contiene le funzioni per la gestione del file di testo relativo ai dati di arduino:

```
1  #pragma once
2
3  #ifndef _ARD_FILE_MANAGER_
4  #define _ARD_FILE_MANAGER_
5
6  #include <iostream>
7  #include <ostream>
8  #include <fstream>
9  #include <string>
10 #include "data_structure.h"
11
12
13 class ard_file_manager
14 {
15 private:
16   std::fstream f;
17   std::ios::_Openmode mode;
18   unsigned long int line_id = 0;
19
20 public:
21   ard_file_manager(std::string file_name, std::ios::_Openmode mode);
22
23   void write_data_line(arduino_data dat);
24
25   bool read_data_line(arduino_data* dat);
26
27   ~ard_file_manager()
28   {
29     f.close();
30   }
31
32   void close()
33   {
34     f.close();
35   }
36 };
37
38 #endif //#ifndef _ARD_FILE_MANAGER_
```

Listing 4: librerie usate

Questa libreria è strutturata in modo totalmente simile alla precedente con l'unica differenza che la struttura dati utilizzata è relativa ad arduino e non al kinect.
La libreria "data_structure.h" è fondamentale e definisce le strutture dati utilizzate nel resto del programma: TODO massiiiiii

```cpp
#pragma once


#ifndef _DATA_STRUCTURE_H_
#define _DATA_STRUCTURE_H_

#include <iostream>
#include <string>
#include <boost/array.hpp>

////////// massiiiiiiiiiiiiiiiii <-------------------------------------------------
// struttura dati per gestire i dati provenienti dal kinect
struct kinect_data
{
  static const int number_of_joints = 3;

  // sottostruttura che contiene i dati di ogni giunto: il suo numero, il giunto attaccato ad esso, la posizione e l'angolo rispetto all'altro giunto
  struct joit_data
  {
    int joint_name = -1;
    int to_joint = -1;
    float position[3] = { 0, 0, 0 };
    float angle[3] = { 0, 0, 0 };
  };

  // <-----------------------------------------------------massiiii
  int needed_joint[number_of_joints] = { 8, 9, 10 };
  int to_joint[24] = { -1, -1, -1, -1, -1, -1, -1, -1, 9, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 };
  int to_ref_joint[24] = { -1, -1, -1, -1, -1, -1, -1, -1, 0, 1, 2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 }; // tabella di accesso (chiave numero joint kinect) -> (restituisce l-indice del numero del giunto in needed_joint)
  float ref_Ang_x[number_of_joints] = { 0, 0, 0 }; //per  8, 9
  float ref_Ang_y[number_of_joints] = { 0, 0, 0 }; //per 8, 9
  float ref_Ang_z[number_of_joints] = { 180, 180, 0 }; //per 8, 9
  float mul_fctor_x[number_of_joints] = { 0, 0, 0 }; //per 8, 9
  float mul_fctor_y[number_of_joints] = { 1, 1, 0 }; //per 8, 9
  float mul_fctor_z[number_of_joints] = { 1, 1, 0 }; //per 8, 9

  //int needed_joint[number_of_joints] = { 4, 5, 6, 8, 9, 10, 1, 20 };
  //int to_joint[24] = { -1, 20, -1, -1, 5, 6, -1, -1, 9, 10, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, };
  //int to_ref_joint[24] = { -1, 6, -1, -1, -1, 1, 2, -1, 3, 4, 5, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 7, -1, -1, -1 }; // tabella di accesso (chiave numero joint kinect)
  //float ref_Ang_x[number_of_joints] = { 0, 0, 0, 0, 0, 0, -90, 0 }; //per 4, 5, 8, 9
  //float ref_Ang_y[number_of_joints] = { 0, 0, 0, 0, 0, 0, 90, 0 }; //per 4, 5, 8, 9
  //float ref_Ang_z[number_of_joints] = { 180, 180, 0, 180, 180, 0, 270, 0 }; //per 4, 5, 8, 9
  //float mul_fctor_x[number_of_joints] = { 0, 0, 0, 0, 0, 0, 0, 0 }; //per 4, 5, 8, 9
  //float mul_fctor_y[number_of_joints] = { 1, 1, 0, 1, 1, 0, 0, 0 }; //per 4, 5, 8, 9
  //float mul_fctor_z[number_of_joints] = { 1, 1, 0, 1, 1, 0, 0, 0 }; //per 4, 5, 8, 9

  joit_data needed_joints[number_of_joints];

  // tempo esatto in cui i dati sono stati acquisiti
  uint64_t frame_time = 0;

  // questo che era? <-----------------------------------
  uint64_t contatore = 0;

  // funzione per verificare se il frame acquisito \'e parziale o completo
  bool full_frame();

  // funzione per stampare i dati memorizzati
  void print_data();

  float jointAngleX(float *P1, float *P2);

  float jointAngleY(float *P1, float *P2);
```

```
64
65    float jointAngleZ(float *P1, float *P2);
66
67    void updateAngles();
68
69    // funzione per convertire il numero del giunto in una stringa
70    std::string joint_Enum_ToStr(int n, std::string language);
71  };
72
73  // struttura dati per gestire i dati provenienti da arduino
74  struct arduino_data
75  {
76    // variabili che rappresentano i dati: dell'accelerometro, del giroscopio, del magnetometro e
        ↪ la temperatura
77    float acc_xyz[3];
78    float gy_xyz[3];
79    float magn_xyz[3];
80    float temp;
81
82    // tempo esatto di acquisizione dei dati
83    uint64_t frame_time = 0;
84
85    // ?
86    uint64_t contatore = 0;
87
88    // funzione che permette di stampare il set di dati attualmente immagazzinato a schermo
89    void print_data();
90  };
91
92  // struttura template che gestisce un set di dati del dataset
93  template<typename type_in, std::size_t N, typename type_out, std::size_t M>
94  struct dataset_data
95  {
96    boost::array<type_in, N> in;
97    boost::array<type_out, M> out;
98
99    // stamapa i dati attualmente immagazzinati
100   void print_data();
101 };
102
103 #endif //#ifndef _DATA_STRUCTURE_H_
```

Listing 5: librerie usate

Infine l'ultima libreria fatta da noi ospita la classe che gestisce la seriale per la comunicazione tramite bluetooth con arduino:

```
1  #pragma once
2
3  #ifndef _SERIAL_HADLER_H_
4  #define _SERIAL_HADLER_H_
5
6  #include <iostream>
7  #include <string>
8  #include <boost/asio.hpp>
9  #include <boost/bind.hpp>
10 #include <boost/asio/serial_port.hpp>
11
12
13 class Serial
14 {
15 private:
16   boost::asio::serial_port* port;
17
18   union Scomp_float
19   {
20     float n_float;
21     uint32_t n_int;
22     uint8_t n_bytes[4];
23   };
24
```

```
  union Scomp_msg_tag
  {
    uint16_t n_16b;
    uint8_t n_bytes[2];
  };

public:
  Serial(std::string com);

  /// <summary>
  /// sincronizza arduino ed il pc
  /// </summary>
  void sinc();

  /// <summary>
  /// invia un floar ad arduino
  /// </summary>
  /// <param name="n"></param>
  void send_float(float n);

  /// <summary>
  /// riceve un float da arduino
  /// </summary>
  /// <returns></returns>
  float receive_float();

  void send_char(char ch);

  char receive_char();

  /// <summary>
  /// receive the data of acc, gy, magn from arduino
  /// </summary>
  /// <param name="acc_xyz"></param>
  /// <param name="g_xyz"></param>
  /// <param name="magn"></param>
  /// <param name="temp"></param>
  /// <returns>
  /// -> 0 if the trasmission is good
  /// -> 1 if it falied
  /// </returns>
  int receive_data(float* acc_xyz, float* g_xyz, float* magn, float* temp);

  ~Serial()
  {
    port->close();
  }
};

#endif // #ifndef _SERIAL_HADLER_H_
```

Listing 6: librerie usate

Oltre a quste classi/strutture integrate in delle librerie separate ci sono altre classi/altro nel main, infatti abbiamo: "namespace logger", "class bounded_buffer", "class kinect_class", "class arduino_class", "class ard_handler", "class kin_handler", "void start_aquire_data" e "void acquire_data".
Per prima cosa è bene spiegare il logger, che viene usato nel resto del programma. Questa libreria serve esclusivamente per il debug e ha attributi specifici per lavoare con più thread contemporaneamente in modo asincrono. Serve fondamentalmente a stampare un testo a schermo, ma una volta settate le impostazioni si può aggiungere a quasto testo il nome del thread il tempo esatto in cui il messaggio viene stampato ed evitare che 2 thread stampino contemporaneamente nella console (se lo facessero il testo si mischierebbe e non si capirebbe nulla). La documentazione approfondita su come funziona questa libreria si può trovare qui. Nel nostro caso i settaggi usati sono: un id che identifica il thread, un tag che identifica il nome del trhead, un tag che identifica il livello di severità, il tempo assoluto e la stringa da stampare. Ad esempio se questa riga è posta all'inizio del main e il main thread viene chiamato "main":

```
BOOST_LOG_SEV(slg, logger::normal) << "hello log";
```

Listing 7: librerie usate

Allora viene stampato a schermo un messaggio formattato in questo modo:

```
id->(0x24a8): main < normal >   time->[00:00:00.000404]  -> hello log
```

Listing 8: librerie usate

Ciò consente di debuggare facilmente operazioni asincrone su più trhead. Il codice relativo hai settaggi necessari per ottenere quel tipo di formattazione è il "namespace logger":

```
namespace logger
{
  // definizione dei divesti livelli di severit\`a
  enum severity_level
  {
    normal,
    warning,
    error,
    critical
  };

  // definizione degli attributi usati
  BOOST_LOG_ATTRIBUTE_KEYWORD(severity, "Severity", severity_level)
  BOOST_LOG_ATTRIBUTE_KEYWORD(tag_attr, "Tag", std::string)
  BOOST_LOG_ATTRIBUTE_KEYWORD(timeline, "Timeline", boost::log::attributes::timer::value_type)
  BOOST_LOG_ATTRIBUTE_KEYWORD(tread_id, "Tread_id", boost::thread::id)
  BOOST_LOG_ATTRIBUTE_KEYWORD(tread_name, "tread_name", std::string)

  // provvede a sostituire il numero del severity_level con la relativa stringa
  std::ostream& operator<< (std::ostream& strm, severity_level level){...}

  // inizializzazione della formattazione del logger
  void init(){...}

  // definizione dei vari parametri
  boost::log::sources::severity_logger< severity_level > f_init(){...}
  void attr_thread(...){...}
  void attr_time(...){...}
  void attr_tag(...){...}
  void attr_thread_name(...){...}
};
```

Listing 9: librerie usate

Inoltre per funzionare correttamente richiede che all'inizio del main siano posti alcuni settaggi:

```
// inizializzazione logger
logger::init();
auto slg = logger::f_init();
attr_thread(&slg);
attr_time(&slg);

// imposto il nome del main trhead in "main"
logger::attr_thread_name(&slg, "main");
```

Listing 10: librerie usate

Un altra classe usata nel resto del programma è "class bounded_buffer", questa classe ospita funzioni specifiche per immagazzinare temporaneamente dei dati gestendone l'ingresso e l'uscita dal buffer. Fondamentalmente costituisce un registro FIFO indipendente dal tipo di dato che si utilizza. Questo tipo di buffer viene comunemente usato in caso si ha la struttura thread che produce dati - thread che consuma dati: il produttore acquisisce i dati e li inserisce nel buffer, il consumatore estrae i dati e li processa. La dimensione del buffer viene specificata quando lo si inizializza e deve essere sufficentemente grande in modo da garantire la non perdita di dati e il non rallentamento del trhead produttore. Nel nostro caso abbiamo adottato questo tipo di buffer per poter garantire l'asincronicità delle acquisizioni tra arduino e il kinect, così facendo si garantisce la massima velocità tra le acquisizioni dei dati. Il codice relativo a questa classe è stato quasi interamente presto dagli esempi della libreria boost, qui ed è:

```
template <class T>
class bounded_buffer
{
public:

  typedef boost::circular_buffer<T> container_type;
  typedef typename container_type::size_type size_type;
```

```
  typedef typename container_type::value_type value_type;
  typedef typename boost::call_traits<value_type>::param_type param_type;

  // nel costruttore si imposta la capacit\'a del buffer
  explicit bounded_buffer(size_type capacity) : m_unread(0), m_container(capacity) {}

  // questa funzione permette di aggiungere un elemeto al buffer
  void push_front(param_type item){...}

  // questa funzione permette di estrarre un elemento dal buffer
  void pop_back(value_type* pItem){...}

  // restituisce la percentuale di riempimento del buffer
  float percentage_of_filling(){...}

  // svuota il buffer
  void flush(){...}

private:
  bounded_buffer(const bounded_buffer&);              // Disabled copy constructor
  bounded_buffer& operator = (const bounded_buffer&); // Disabled assign operator

  bool is_not_empty() const { return m_unread > 0; }

  bool is_not_full() const { return m_unread < m_container.capacity(); }

  size_type m_unread;
  container_type m_container;
  boost::mutex m_mutex;
  boost::condition_variable m_not_empty;
  boost::condition_variable m_not_full;
};
```

Listing 11: librerie usate

Di seguito troviamo i thread producer "kinect_class" e "arduino_class". La classe "kinect_class" provvede a leggere i dati dal kinect ed aggiungerli al suo corrispettivo registro FIFO. L'estrazione dei dati dal kinect è stata realizzata sulla base dell'esempio body basic D2D integrato nell'sdk browser fornito da microsoft. Il codice relativo a questa classe è:

```
class kinect_class
{
public:

  // Current Kinect
  IKinectSensor* m_pKinectSensor;
  ICoordinateMapper* m_pCoordinateMapper;

  // Body reader
  IBodyFrameReader* m_pBodyFrameReader;

  real_time t;

  bounded_buffer<kinect_data>* FIFO_kin;

  kinect_class(bounded_buffer<kinect_data>* data_FIFO){...}

  void start()
  {

    bool flag = false;
    while (true)
    {
      {...}

      // prendo il tempo
      uint64_t t1 = t.get_curr_time();

      // acquisisco i dati e li sposto in data
      flag = Update(&data);

```

```
32      // se l'acquisizione \'e andata a buon fine procedo
33      if (flag)
34      {
35        // prendo un altra volta il tempo
36        uint64_t t2 = t.get_curr_time();
37        // cos\'i da poter impostare che il tempo in cui sono stati acquisiti i dati sia
38        // la media tra il tempo prima di cominciare l'acquisizione ed il tempo dopo che essa \'
              ↪ e finita
39        data.frame_time = (t1 + t2) / 2;
40
41        // inserisco i dati cos\'i acquisiti nel registro fifo
42        FIFO_kin->push_front(data);
43
44        // faccio aspettare un po di tempo a questo trhead perch\'e altrimenti il kinect da
              ↪ errore,
45        // dato che non supporta un acquisizione a pi\'u di 30fps
46        boost::this_thread::sleep(boost::posix_time::milliseconds(35));  al kinect scoppia
47
48      }
49
50      {...}
51
52    }
53  }
54
55  // inizializza il kinect, ritorna 1 se ha avuto successo, 0 altrimenti
56  int InitializeDefaultSensor(){...}
57
58  // acquisisce i dati e li passa alla funzione per processarli
59  bool Update(kinect_data* data)
60  {
61    {...}
62
63    flag = ProcessBody(BODY_COUNT, ppBodies, data);
64
65    {...}
66
67    return flag;
68
69  }
70
71  // questa funzione provvede a spostare i dati da joints a data sistemandoli nella loro
        ↪ struttura dati,
72  // una spiegazione di come ci\'o viene fatto verra data nel capitolo successivo
73  bool handle_data(Joint* joints, kinect_data* data){...}
74
75
76  // quetsta funzione separa le diverse persone rilevate e trasferisce le informazioni
77  // relative hai giunti della prima persona alla funzione handle_data
78  bool ProcessBody(int nBodyCount, IBody** ppBodies, kinect_data* data)
79  {
80    {...}
81
82    for (int i = 0; i < nBodyCount; ++i)
83    {
84      {...}
85
86      Joint joints[JointType_Count];
87
88      hr = pBody->GetJoints(_countof(joints), joints);
89
90      return handle_data(joints, data);
91
92      {...}
93
94    }
95
96    {...}
97  }
```

```
98
99    // provvede a spegnere il kinect e deallocare le sue risorse nel caso la classe venga
          ↪ distrutta
100   ~kinect_class(){...}
101   template<class Interface>
102   inline void SafeRelease(Interface *& pInterfaceToRelease){...}
103
104 };
```

Listing 12: librerie usate

La classe "arduino_class" provvede a caricare continuamente i dati provenienti dal kineck nel registro fifo, infatti la funzione "start" viene lanciata come thread e continua a caricare i dati fino al termine del programma.

```
1  class arduino_class
2  {
3  public:
4    Serial* ser;
5    real_time t;
6    bounded_buffer<arduino_data>* FIFO_ard;
7
8    // il costruttore provvede ad inizializzare la seriale e sincronizzare i 2 dispositivi
9    arduino_class(bounded_buffer<arduino_data>* data_FIFO, string com_port)
10   {
11     ser = new Serial(com_port);
12     cout << "serial ok" << endl;
13     FIFO_ard = data_FIFO;
14     ser->sinc();
15   }
16
17   // questa funzione viene lanciata all'avvio del thread e provvede ad acquisire i dati ed
         ↪ inserirli nel registro fifo
18   void start()
19   {
20     while (true)
21     {
22       arduino_data data;
23
24       uint64_t t1 = t.get_curr_time();
25       ser->receive_data(data.acc_xyz, data.gy_xyz, data.magn_xyz, &data.temp);
26       uint64_t t2 = t.get_curr_time();
27
28     // per ottenere un accurata misurazione del tempo viene preso prima e dopo l'acquisizione e
           ↪ poi fatta la media dei valori ottenuti
29       data.frame_time = (t1 + t2) / 2;
30
31       FIFO_ard->push_front(data);
32     }
33   }
34 };
```

Listing 13: librerie usate

Dopo i 2 thread producer si hanno i 2 thread consumer: "class ard_handler" e "class kin_handler", che si occupano rispettivamente di salvare su file di testo le informazioni di arduino e del kinect.

```
1  /// <summary>
2  /// handle the data from arduino and write it on a file
3  /// </summary>
4  class ard_handler
5  {
6  private:
7    bounded_buffer<arduino_data>* ard;
8    ard_file_manager* f;
9
10 public:
11   // il costruttore provvede ad inizializzare la classe per gestire i file di testo di arduino
12   ard_handler(bounded_buffer<arduino_data>* ard, string file_name)
13   {
```

```
    this->ard = ard;
    f = new ard_file_manager(file_name, std::ios::out);
  }

  // questa funzione \'e quella che viene effettivamente lanciata all'avvio del thread e
      ↪ provvede a scaricare il buffer scrivendo i dati sul file fino al raggiungimento della
      ↪ quota predichiarata "tot_esempi"
  void start()
  {
  // inizializzazione logger
    auto slg = logger::f_init();
    attr_thread(&slg);
    logger::attr_thread_name(&slg, "ard_handler");

    BOOST_LOG_SEV(slg, logger::normal) << "hello log";

    for (int i = 0; i < tot_esempi; i++)
    {
      arduino_data data_ard;
      ard->pop_back(&data_ard);

      f->write_data_line(data_ard);

    }

  // segnalo sulla console qunado il thread ha finito il suo lavoro
    BOOST_LOG_SEV(slg, logger::normal) << "end";
  }
};
```

Listing 14: librerie usate

La classe "kin_handler" esegue esattamente le stesse operazioni della precedente con la sola differenza che usa "kin_file_manager" al posto di "ard_file_manager".

```
/// <summary>
/// handle the data from the kinect and write it in the file
/// </summary>
class kin_handler
{
private:
  bounded_buffer<kinect_data>* kin;
  kin_file_manager* f;

public:
  kin_handler(bounded_buffer<kinect_data>* kin, string file_name){...}

  void start(){...}
};
```

Listing 15: librerie usate

Dopo tutte queste classi si hanno 2 funzioni che si occupano di inizializzare le altre classi, lanciare tutti i thread neccessari ed acquisire dall'utente i nomi dei file etc.

```
void start_aquire_data(string out_ard, string out_kin, string com_port)
{
  // in inizializzo il logger per questa funzione
  auto slg = logger::f_init();
  attr_thread(&slg);
  logger::attr_thread_name(&slg, "main");

  // dichiaro i 2 buffer FIFO
  bounded_buffer<arduino_data> data_FIFO_ard(20);
  bounded_buffer<kinect_data> data_FIFO_kin(20);

  // chiamo i costruttori delle classi che gestiscono il kinect e arduino
  // al costruttore di arduino gli devo passare la com a cui il bluetooth \'e collegato
  arduino_class a(&data_FIFO_ard, com_port);
```

```
  kinect_class k(&data_FIFO_kin);

  // chiamo i costruttori delle classi per la scrittura nei file di testo
  ard_handler a_h(&data_FIFO_ard, out_ard);
  kin_handler k_h(&data_FIFO_kin, out_kin);

  BOOST_LOG_SEV(slg, logger::normal) << "costructor end" << std::endl;

  // lancio i thread che "producono i dati"
  boost::thread ard([&a]() { a.start(); });
  boost::thread kin([&k]() { k.start(); });

  BOOST_LOG_SEV(slg, logger::normal) << "producer thread launced" << std::endl;

  // lancio i thread che "consumnano dati"
  boost::thread ard_h([&a_h]() { a_h.start(); });
  boost::thread kin_h([&k_h]() { k_h.start(); });

  BOOST_LOG_SEV(slg, logger::normal) << "consumer thread launched" << std::endl;

  // aspetto all'infinito (finch\'e il programma non viene chiuso
  kin.join();
  ard.join();
  kin_h.join();
  ard_h.join();
}
```

Listing 16: librerie usate

infine c'è la funzione che si occupa di acquisire i dati dall'utente e passarli alla funzione appena vista.

```
void acquire_data()
{
  string ard_file_mane = "";
  string kin_file_mane = "";
  string arduino_com_port = "";

  // acquisisco il nome dei 2 file che conterranno i dati di arduino e i dati del kinect
  cout << "insert the file names: " << endl << "\t arduino -> ";
  std::getline(std::cin, ard_file_mane);
  cout << "\t kinekt -> ";
  std::getline(std::cin, kin_file_mane);

  // acquisisco il numero della com a cui \'e collegato il bluetooth del computer
  cout << "insert the arduino com port -> : ";
  std::getline(std::cin, arduino_com_port);

  // lancio la funzione passandogli i dati appena acquisiti
  start_aquire_data(ard_file_mane, kin_file_mane, arduino_com_port);
}
```

Listing 17: librerie usate

Infine il main chiama semplicemente la funzione appena trattata.

```
int main()
{
  logger::init();

  auto slg = logger::f_init();
  attr_thread(&slg);
  attr_time(&slg);
  logger::attr_thread_name(&slg, "main");

  BOOST_LOG_SEV(slg, logger::normal) << "hello log";

  acquire_data();

  return 0;
}
```

Listing 18: librerie usate

Riassumendo la struttura generale di questo programma si ha:

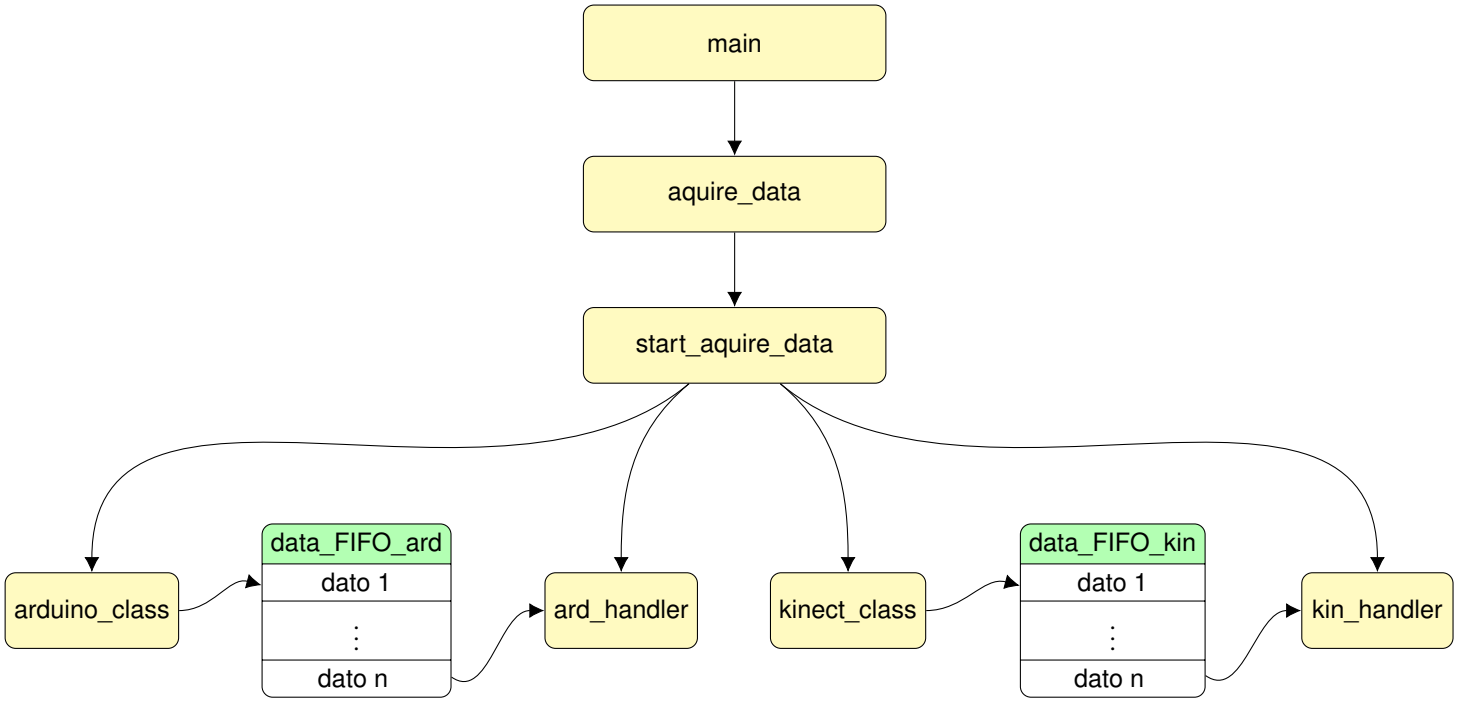


Figura 1: i programmi usati per ottenere i pesi della rete addestrata (?)

Cioè il "main" chiama "aquire_data" che chiama a sua volta "start_aquire_data". Dopodichè vengono lanciati i 4 thread ("arduino_class", "ard_handler", "kinect_class", "kin_handler"). I thread che si occupano di prelevare i dati dai dispositivi sono "arduino_class" e "kinect_class", che inseriscono i dati nel corrispettivo buffer FIFO. Infine i thread che si occupano di scrivere i dati su i file di testo sono "ard_handler" e "kin_handler", che scrivono i dati prelevati dal buffer nel corrispettivo file di testo.

0.1 programma che si occupa di post-elaborare i dati

Dopo aver acquisito i dati di arduino e del kinect bisogna effettuare una post elaborazione per ottenere coppie di valori riferiti allo stesso istante temporale. (altro da dire?) Il programma usato per fare ciò riprende molte cose dal codice del programma "prg_get_data", le librerie usate sono le stesse ed anche le classi "kin_file_manager", "kin_file_manager" e "data_structure.h". L'unica classe nuova è "make_dataset":

```
1  class make_dataset
2  {
3  private:
4    // dichiaro i puntatori alle classi per la gestione dei file cos\'i da renderle disponibili
       ↪ per tutta la classe
5    ard_file_manager *f_a;
6    kin_file_manager *f_k;
7
8    // delta di accettazione in millisecondi
9    const int range = 35;
10
11   // dichiaro 2 liste per conservare i dati
12   list<arduino_data> a_dat;
13   list<kinect_data> k_dat;
14
15   // devo dichiararare il numero di ingressi e di uscite della rete
16   static const std::size_t n_in = 9;
17   static const std::size_t n_out = 4;
18
19   // dichiaro una lista di tipo dataset_data che verr\'a riempita con i dati accoppiati
20   list<dataset_data<float, n_in, float, n_out>> dataset;
21
22   // questa funzione si occupa di scrivere il dataset sul file
23   void write_dataset(string dataset_file_name){...}
24
25   // restituisce 1 solo se i 2 tempi che gli vengono passati differiscono meno della soglia
       ↪ impostata sopra
26   bool check_sync(uint64_t t1, uint64_t t2){...}
27
28   // questo metodo sposta i dati dalle strutture "arduino_data" e "kinect_data" alla struttura "
       ↪ dataset" dichiarata globalmente nella classe
29   void transfer_data_to_dataset(arduino_data* ard_iter, kinect_data* kin_iter){...}
30
31   // restituisce la differenza temporale tra i tempi passati alla funzione
32   uint32_t time_distance(uint32_t t1, uint32_t t2){...}
33
34   // sincronizza i dati caricati in "a_dat", "k_dat" e li sposta in "dataset" attraverso la
       ↪ funzione "transfer_data_to_dataset". Questa funzione non tiene conto dei "cloni", nella
       ↪ sezione successiva vi \'e una spiegazione pi\'u dettagliata.
35   void sync_with_clones(){...}
36
37   // stessa cosa della funzione precedente ma eliminando i cloni.
38   void sync_without_clones(){...}
39
40   // carica i dati dai file usando "f_a" e "f_k", sposta i dati acquisiti in "a_dat" e k_dat"
41   void load_data(){...}
42
43 public:
44   // l'unica funzione accessibile \'e il costruttore che provvede ad inizializzare i  file
       ↪ manager, caricare i dati, sincronizzarli (con o senza cloni) e scrivere il file del
       ↪ dataset con i dati appena generati.
45   make_dataset(string ard_file_name, string kin_file_name, string dataset_file_name, bool
       ↪ want_clones)
46   {
47     f_a = new ard_file_manager(ard_file_name, std::ios::in);
48     f_k = new kin_file_manager(kin_file_name, std::ios::in);
49
50     load_data();
51
52     if (want_clones)
53     {
54       sync_with_clones();
```

```
    }
    else
    {
      sync_without_clones();
    }

    write_dataset(dataset_file_name);

  }

  // distruttore della classe
  ~make_dataset(){...}

};
```

Listing 1: librerie usate

La funzione che si occupa di acquisire i nomi dei file e altro dall'utente è "post_elaborate":

```
void post_elaborate()
{
  string ard_file_mane = "";
  string kin_file_mane = "";
  string dataset_file_name = "";
  string tmp = "";
  bool clones = false;

  // acquisisco i nomi dei file dei dati di arduino e del kinect
  cout << "insert the input file names: " << endl
    << "\t arduino -> ";
  std::getline(std::cin, ard_file_mane);
  cout << "\t kinekt -> ";
  std::getline(std::cin, kin_file_mane);

  // acquisisco il nome del file in cui verr\'a scritto il dataset
  cout << "insert the ouptut file name -> : ";
  std::getline(std::cin, dataset_file_name);

  // chiedo se si vuole il dataset con o senza cloni
  cout << "do you want clones in the data? [y/n] -> : ";
  std::getline(std::cin, tmp);

  if (tmp == "y")
  {
    clones = true;
  }

  // chiamo la classe make_dataset passandogli i dati appena acquisiti
  make_dataset(ard_file_mane, kin_file_mane, dataset_file_name, clones);
}
```

Listing 2: librerie usate

TODO