0.1 Programma arduino

Questo programma si occupa di acquisire i dati dai sensori attraverso l'I2C, elaborarli e scriverli sulla seriale, in modo che il modulo bluetooth li invii al pc. 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 l'I2C.
Nel "setup" si trova soltanto l'inilizializzazione alla seriale per comunicare con il modulo "HC-05", il baud rate impostato è lo stesso che è stato settato nel modulo tramite la modalità AT:

```
1 void setup()
2 {
3   Serial.begin(115200);
4 }
```

Listing 1: funzione "setup"

Nella funzione "loop" si trovano le varie dichiarazioni alle altre classi, un while 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:

```
1  void loop()
2  {
3    // chiamo i costruttori delle 2 classi che gestiscono i sensori e provvedo al loro settaggio.
4    QMC5883 mm;
5    MPU_6050 aa;
6    aa.setting();
7
8    // accendo un led per il debug
9    digitalWrite(13, HIGH);
10
11   // aspetto finchè il pin "state" del modulo "HC-05" non diventa alto
12   while (!digitalRead(3))
13   {
14     delay(10);
15   }
16
17   // quando il bluetooth è connesso posso procedere ad inizializzare e sincronizzare la seriale
18   serial ser;
19   ser.sinc();
20
21   // spengo il led
22   digitalWrite(13, LOW);
23
24   while (true)
25   {
26     // acquisisco i dati del magnetometro e li normalizzo
27     mm.get_data();
28     mm.normalize(100);
29
30     // acquisisco i dati dell'accelerometro
31     aa.get_data();
32
33     // li sposto in degli array
34     float acc_xyz[] = {aa.get_AcX(), aa.get_AcY(), aa.get_AcZ()};
35     float g_xyz[] = {aa.get_GyX(), aa.get_GyY(), aa.get_GyZ()};
36     float magn[] = {mm.getX(), mm.getY(), mm.getZ()};
37
38     // li scrivo sulla seriale in modo che il modulo bluetooth li trasmetta al pc.
39     ser.send_data(acc_xyz, g_xyz, magn, aa.get_temp());
40   }
41 }
```

Listing 2: funzione "loop"

Qui sono state usate le classi "MPU_6050", "QMC5883" e "serial" (TODO da rifare la frase).
Le classi "MPU_6050" e "QMC5883" si occupano di impostare i registri dei 2 moduli ed inoltre hanno metodi che permettono l'estrazione delle misurazioni da loro effettuate.
L'impostazione dei moduli è stata eseguita seguendo i rispettivi datasheet e (TODO!) copiando spudoratamente esempi su internet.

```
1 class MPU_6050
2 {
```

```
  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 che ci metto qua in mezzo? :(

```
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(){...}
    float getZ_rescaled(){...}

    // funzione che provvede a leggere i regisrti del modulo per acquisire i dati
    // spostandoli nelle variabili di classe
    void get_data(){...}

    // funzioni per calcolare la norma e ritornare i valori normalizzati
    float norma(float x, float y, float z){...}
    float get_norma(){ return norma(nowX, nowY, nowZ); }
    void normalize(int Max){...}
};
```

Listing 4: classe "QMC5883"

Infine la classe serial si occupa di scomporre i dati di tipo float restituiti dai moduli in array di 4 byte che poi vengono trasmessi al modulo bluetooth tramite la seriale.

```
class serial
{
  public:

    // funzione che permette di sincronizzarsi con il pc
    void sinc(){...}

    // union per poter scompattare un float in 4 byte
    union Scomp_float{...};

    // funzione che permette di inviare un intero float scompattandolo in
    // 4 byte che vengono trasmessi serialmente
    void send_float(float n){...}

    // funzione che permette di riceve un float come sopra
    float receive_float(){...}

    // funzione per l'invio di un solo carattere
    void send_char(char ch){...}

    // funzione per la ricezione di un singolo carattere
    char receive_char(){...}

    // funzione che provvede ad inviare i dati necessari.
    void send_data(float* acc_xyz, float* g_xyz, float* magn, float temp){...}
};
```

Listing 5: classe "serial"

Per effettuare l'operazione di serializziopne dei float in array di byte abbiamo fatto ricorso all'uso di una struttura dati chiamata "union":

```
union Scomp_float
{
  float f;
  int i;
  unsigned char byte_s[4];
};
```

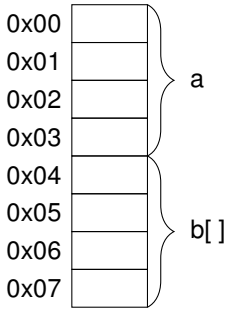
Listing 6: classe "serial"

Questa struttura dati ha la particolarità che a differenza di una struct, in cui i dati sono memorizzati in modo contiguo nella memoria, l'union memorizza tutti i dati della struttura nelle stesse locazioni di memoria.

TODO sistemare la formattazione alla fine

```c
struct es_struct
{
  int a;
  unsigned char b[4];
}
```

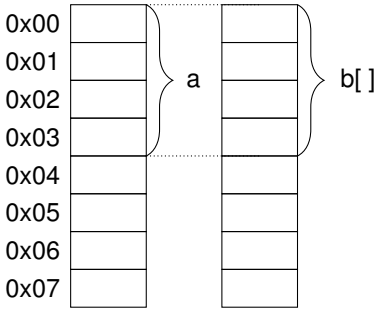
(a) Struct generico



(b) Rappresentazione dei dati dello struct memorizzati in ram

```c
union es_union
{
  int a;
  unsigned char b[4];
}
```

(c) Union generico



(d) Rappresentazione dei dati dell'union memorizzati in ram

Figura 1: Confronto tra struct ed union

Come si può vedere dalla figura 1 lo struct (figura 1a) memorizza i dati in ram facendo partire l'intero dall'indirizzo 0x00 e l'array di byte dall'indirizzo 0x04 (figura 1c), mentre l'union (figura 1d) sovrappone i 2 dati nelle stesse locazioni di memoria (figura 1). Cioò permmette ad esempio di leggere un intero (o un qualunque altro tipo di dato) come un insieme di byte: con riferimento alla figura 1d se pongo:

```
es_union U;
U.a = 0x0514FFAA;
```

Listing 7: classe "serial"

e successivamente vado a vedere il valore di b[ ] ottengo i valori:

```
U.b[0] -> 0x05
U.b[1] -> 0x14
U.b[2] -> 0xFF
U.b[3] -> 0xAA
```

Listing 8: classe "serial"

che sono esattamente i byte costituenti U.a. Ciò ci ha permesso di inviare un float come sequenza di byte al pc, nella classe serial sul pc (TODO inserire ref) c'e' esattamente lo stesso tipo di struttura per ricostruire il float.