```
        / 1000000.0) * SONAR_SOUND_SPEED) / 2;
    //return (sonar_echo_count * SONAR_US_PER_COUNT) / 58.0;
}

#pragma interrupt (sonarEchoISR(vect = VECT(ICU, IRQ7)))
static void sonarEchoISR() {
    // Copy counter
    sonar_echo_count = MTU5U_GetTimerCounter();
    // Stop timer
    MTU5U_Stop();
    // Return to idle state
    sonar_state = SONAR_IDLE;
}
```

14 IMU

L'IMU (Inertial Mesurement Unit) è una scheda elettronica facente parte della famiglia dei MEMS (dall'acronimo inglese di sistemi micro elettromeccanici) che combina componenti elettriche e meccaniche in scala microscopica in grado di misurare specifiche caratteristiche ambientali quali temperatura o pressione, o di movimento come accelerazione e velocità angolare, trasformandole in variazioni di segnali analogici o digitali.
Nel nostro progetto è stata utilizzata una IMU Drotek 10Dof V2 costituita dalle seguenti componenti:

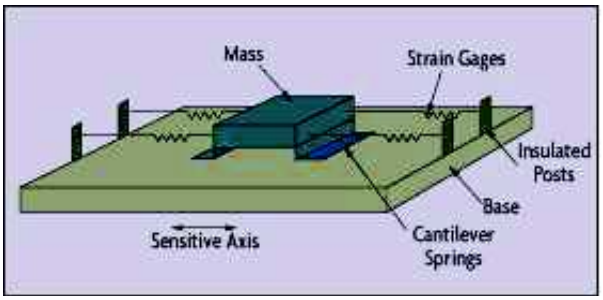
- Accelerometro: MPU-6050;
- Giroscopio: MPU-6050;
- Magnetometro: HMC-5883L.

Il Motion Processing Unit è la prima soluzione di elaborazione moto al mondo con sensore integrato di fusione "9-Axis". L'MPU-6050 integra, infatti, un Digital Motion Processor (DMP) che fornisce un'accelerazione hardware, mediante la quale è possibile esportare, contemporaneamente e in maniera unitaria, i dati provenienti da accelerometro, giroscopio e magnetometro, attraverso la porta I^2C denominata dal produttore "MotionFusion".

14.1 IMU: Accelerometro

14.1.1 Principio di Funzionamento

Un accelerometro è un dispositivo capace di misurare un'accelerazione lineare, che ha luogo in riferimento al suo sistema di coordinate.
Esistono diverse tipologie di questo componente, che differiscono per la possibilità di effettuare la misura in una, due o tutte e tre le direzioni dello spazio.

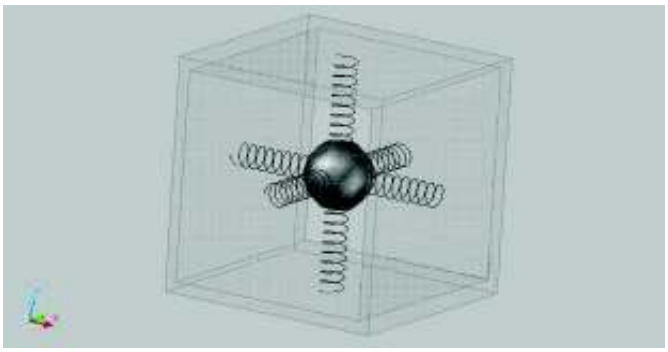


Il funzionamento é relativamente semplice: si misura la velocità di una massa vincolata tramite delle molle al centro di una parete ad ogni spostamento di tale oggetto nello spazio. Le molle, allungandosi e comprimensosi, permettono di stabilire l'accelerazione (variazione della velocità). I due principi su cui ci si basa sono:

- La **seconda legge della dinamica di Newton**: una massa m accelerata di a è sottoposta ad una forza F tale che: $F = m * a$;
- La **legge di Hooke**: per una molla elastica, forza F e posizione x sono direttamente proporzionali e legate da una costante k chiamata costante elastica della molla, per cui $F = k * x$.

Di conseguenza, risulta che: $F = m * a = k * x$ ovvero $a = \frac{k * x}{m}$.
Conoscendo massa, costante elastica e ricavando la posizione della massa, possiamo risalire all'accelerazione a cui è sottoposto il dispositivo o il corpo a cui esso è collegato.

L'accelerometro utilizzato in questo progetto è a tre assi, tuttavia il principio teorico rimane quello descritto sopra. La realizzazione è leggermente più complessa: si immagini una sfera all'interno di un cubo, sospesa mediante sei molle (tre coppie di molle), agganciata ognuna al centro di una faccia di tale solido.



Muovendo il cubo nello spazio la sfera si muoverà, andando ad allungare o comprimere le molle che la tengono sospesa. Anche in questo caso, il grado di compressione delle molle permetterà di andare a trovare l'accelerazione.

14.1.2 L'Accelerometro nell'MPU6050

Un'altra caratteristica dell'accelerometro di cui ci siamo serviti è la presenza di tre masse distinte, una per ogni asse.
L'accelerazione lungo ciascun asse causa uno spostamento di tali masse, il quale viene misurato mediante sensori capacitivi in modo differenziale.
Essenziale è notare come quando il dispositivo viene appoggiato su una superficie piatta, i valori misurati saranno pari a:

- X-axis: 0G
- Y-axis: 0G
- Z-axis: +1G

Il fattore di scala dell'accelerometro viene calibrato in fabbrica ed è indipendente dalla tensione di alimentazione.
Il full-scale range può essere scelto tra:

- ± 2G;
- ± 4G;
- ± 8G;
- ± 16G;

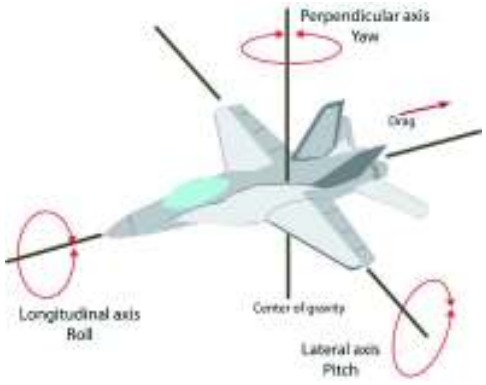
14.2 IMU: Giroscopio

14.2.1 Principio di funzionamento

Mentre l'accelerometro misura l'accelerazione di un oggetto lungo una direzione, il giroscopio va a determinare, invece, la rotazione attorno ad un asse fisso. Bisogna, innanzitutto, andare a sottolineare la netta differenza esistente tra un giroscopio elettronico (di nostro interesse) e giroscopio meccanico. Il primo sfrutta le forze di coriolis e misura la velocità angolare. Il secondo invece 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.
Quando un corpo di massa m è in moto rispetto ad un sistema di riferimento non inerziale (velocità V'), ed il sistema ha una propria velocità angolare ω, come in questo caso il sensore, su di esso agisce una forza fittizia chiamata forza di Coriolis. A tale forza apparente risulta soggetto un corpo quando si osserva il suo moto da un sistema di riferimento che sia in moto circolare rispetto a un sistema di riferimento inerziale.
Immaginiamo la struttura microscopica formata da una massa e delle molle che la mantengono vincolata, in analogia con quanto detto per l'accelerometro.
Nel momento in cui si ha una rotazione attorno all'asse perpendicolare al piano su cui giacciono le molle, la forza di Coriolis produce uno spostamento lungo l'asse X definito "asse di sense", ovvero l'asse dal quale è possibile ricavare il valore cercato.
La forza di Coriolis vale $F_c = -2m(\omega * V')$ e la forza di reazione della molla vale $F = k * x$.
Unendo le due relazioni trovate, conoscendo lo spostamento x possiamo risalire alla velocità angolare ω cercata, imponendo $F_c = F$.

14.2.2 Roll, Pitch, Jaw

Tutti i veicoli che sono liberi di muoversi nelle tre direzioni dello spazio (come gli elicotteri e come il Ducted Fan) possono cambiare la loro rotazione attorno ai tre assi ortogonali solidali con la struttura, il cui centro coincide con il centro di gravità del veicolo.



- L'angolo di Roll (chiamato anche angolo di rollio) corrisponde ad un trasferimento di peso trasversale della struttura attorno all'asse longitudinale;
- Il beccheggio (o angolo di Pitch) è il trasferimentro di peso longitudinale con rotazione attorno all'asse trasversale. Si verifica prevalenemente in fase di accelerazione e di frenata.
- Il moto di imbardata (angolo di Yaw) corrisponde alla rotazione attorno ad un asse verticale.

14.2.3 Il Giroscopio dell'MPU6050

Il modello appena presentato è molto semplificativo rispetto a quello reale.
Un giroscopio presente nell'IMU consiste in tre giroscopi MEMS indipendenti, in modo da realizzare la misurazione sui tre assi x, y, z. È formato da un sistema, definito "comb-drive", che sottopone le masse mobili ad una vibrazione. Nel momento in cui si ha una velocità angolare, a questa vibrazione, rilevata da un pickoff capacitivo, è sovrapposto uno spostamento causato dalla forza di Coriolis, che viene opportunamente rilevato e tradotto in un valore opportuno.
Il segnale risultante viene amplificato, demodulato e filtrato in modo da produrre una tensione proporzionale all'angolo di rotazione. Questo valore di voltaggio viene digitalizzato utilizzando un AD Converter a 16 bit, in modo indipendente per ogni asse.
Il full scale range del giroscopio può essere programmato in modo da corrispondere a:

- $\pm$ 250 gradi/secondo;
- $\pm$ 500 gradi/secondo;

- ± 1000 gradi/secondo;
- ± 2000 gradi/secondo;

14.3 MPU6050 Driver

Giroscopio e Accelerometro fanno parte, nel nostro progetto, dello stesso sensore inerziale. L'IMU da noi utilizzata è, infatti, la utilizzato nel nostro progetto è MPU6050, un accelerometro, come già accennato, a 3 assi.

In accordo con il principio seguito della logica a driver, è stata creata una

Tabella 1: Caratteristiche Principali

VDD	2.375V ÷ 3.46V
Protocollo di Comunicazione	I^2C

libreria chiamata, per l'appunto **MPU6050.h** che racchiude tutto l'elenco dei registri e il prototipo delle varie funzioni che hanno permesso l'acquisizione e l'elaborazione dei valori.

```
#define MPU6050_ADDRESS          0xD2 //Address with end
    write bit
#define MPU6050_RA_XG_OFFS_TC        0x00
#define MPU6050_RA_YG_OFFS_TC        0x01
#define MPU6050_RA_ZG_OFFS_TC        0x02
#define MPU6050_RA_X_FINE_GAIN       0x03
#define MPU6050_RA_Y_FINE_GAIN       0x04
#define MPU6050_RA_Z_FINE_GAIN       0x05
#define MPU6050_RA_XA_OFFS_H         0x06
#define MPU6050_RA_XA_OFFS_L_TC      0x07
#define MPU6050_RA_YA_OFFS_H         0x08
#define MPU6050_RA_YA_OFFS_L_TC      0x09
#define MPU6050_RA_ZA_OFFS_H         0x0A
#define MPU6050_RA_ZA_OFFS_L_TC      0x0B
#define MPU6050_RA_XG_OFFS_USRH      0x13
#define MPU6050_RA_XG_OFFS_USRL      0x14
#define MPU6050_RA_YG_OFFS_USRH      0x15
#define MPU6050_RA_YG_OFFS_USRL      0x16
#define MPU6050_RA_ZG_OFFS_USRH      0x17
#define MPU6050_RA_ZG_OFFS_USRL      0x18
#define MPU6050_RA_SMPLRT_DIV        0x19
#define MPU6050_RA_CONFIG        0x1A
#define MPU6050_RA_GYRO_CONFIG       0x1B
#define MPU6050_RA_ACCEL_CONFIG      0x1C
```

```
#define MPU6050_RA_FF_THR         0x1D
#define MPU6050_RA_FF_DUR         0x1E
#define MPU6050_RA_MOT_THR        0x1F
#define MPU6050_RA_MOT_DUR        0x20
#define MPU6050_RA_ZRMOT_THR          0x21
#define MPU6050_RA_ZRMOT_DUR          0x22
#define MPU6050_RA_FIFO_EN        0x23
#define MPU6050_RA_I2C_MST_CTRL       0x24
#define MPU6050_RA_I2C_SLV0_ADDR      0x25
#define MPU6050_RA_I2C_SLV0_REG       0x26
#define MPU6050_RA_I2C_SLV0_CTRL      0x27
#define MPU6050_RA_I2C_SLV1_ADDR      0x28
#define MPU6050_RA_I2C_SLV1_REG       0x29
#define MPU6050_RA_I2C_SLV1_CTRL      0x2A
#define MPU6050_RA_I2C_SLV2_ADDR      0x2B
#define MPU6050_RA_I2C_SLV2_REG       0x2C
#define MPU6050_RA_I2C_SLV2_CTRL      0x2D
#define MPU6050_RA_I2C_SLV3_ADDR      0x2E
#define MPU6050_RA_I2C_SLV3_REG       0x2F
#define MPU6050_RA_I2C_SLV3_CTRL      0x30
#define MPU6050_RA_I2C_SLV4_ADDR      0x31
#define MPU6050_RA_I2C_SLV4_REG       0x32
#define MPU6050_RA_I2C_SLV4_DO        0x33
#define MPU6050_RA_I2C_SLV4_CTRL      0x34
#define MPU6050_RA_I2C_SLV4_DI        0x35
#define MPU6050_RA_I2C_MST_STATUS     0x36
#define MPU6050_RA_INT_PIN_CFG        0x37
#define MPU6050_RA_INT_ENABLE         0x38
#define MPU6050_RA_DMP_INT_STATUS     0x39
#define MPU6050_RA_INT_STATUS         0x3A
#define MPU6050_RA_ACCEL_XOUT_H       0x3B
#define MPU6050_RA_ACCEL_XOUT_L       0x3C
#define MPU6050_RA_ACCEL_YOUT_H       0x3D
#define MPU6050_RA_ACCEL_YOUT_L       0x3E
#define MPU6050_RA_ACCEL_ZOUT_H       0x3F
#define MPU6050_RA_ACCEL_ZOUT_L       0x40
#define MPU6050_RA_TEMP_OUT_H         0x41
#define MPU6050_RA_TEMP_OUT_L         0x42
#define MPU6050_RA_GYRO_XOUT_H        0x43
#define MPU6050_RA_GYRO_XOUT_L        0x44
#define MPU6050_RA_GYRO_YOUT_H        0x45
#define MPU6050_RA_GYRO_YOUT_L        0x46
#define MPU6050_RA_GYRO_ZOUT_H        0x47
#define MPU6050_RA_GYRO_ZOUT_L        0x48
#define MPU6050_RA_EXT_SENS_DATA_00     0x49
#define MPU6050_RA_EXT_SENS_DATA_01     0x4A
#define MPU6050_RA_EXT_SENS_DATA_02      0x4B
#define MPU6050_RA_EXT_SENS_DATA_03     0x4C
#define MPU6050_RA_EXT_SENS_DATA_04     0x4D
```

```
#define MPU6050_RA_EXT_SENS_DATA_05     0x4E
#define MPU6050_RA_EXT_SENS_DATA_06     0x4F
#define MPU6050_RA_EXT_SENS_DATA_07     0x50
#define MPU6050_RA_EXT_SENS_DATA_08     0x51
#define MPU6050_RA_EXT_SENS_DATA_09     0x52
#define MPU6050_RA_EXT_SENS_DATA_10     0x53
#define MPU6050_RA_EXT_SENS_DATA_11     0x54
#define MPU6050_RA_EXT_SENS_DATA_12     0x55
#define MPU6050_RA_EXT_SENS_DATA_13     0x56
#define MPU6050_RA_EXT_SENS_DATA_14     0x57
#define MPU6050_RA_EXT_SENS_DATA_15     0x58
#define MPU6050_RA_EXT_SENS_DATA_16     0x59
#define MPU6050_RA_EXT_SENS_DATA_17     0x5A
#define MPU6050_RA_EXT_SENS_DATA_18     0x5B
#define MPU6050_RA_EXT_SENS_DATA_19     0x5C
#define MPU6050_RA_EXT_SENS_DATA_20     0x5D
#define MPU6050_RA_EXT_SENS_DATA_21     0x5E
#define MPU6050_RA_EXT_SENS_DATA_22     0x5F
#define MPU6050_RA_EXT_SENS_DATA_23     0x60
#define MPU6050_RA_MOT_DETECT_STATUS    0x61
#define MPU6050_RA_I2C_SLV0_DO      0x63
#define MPU6050_RA_I2C_SLV1_DO      0x64
#define MPU6050_RA_I2C_SLV2_DO      0x65
#define MPU6050_RA_I2C_SLV3_DO      0x66
#define MPU6050_RA_I2C_MST_DEL_CTRL     0x67
#define MPU6050_RA_SGNL_PATH_RESET      0x68
#define MPU6050_RA_MOT_DETECT_CTRL  0x69
#define MPU6050_RA_USER_CTRL        0x6A
#define MPU6050_RA_PWR_MGMT_1       0x6B
#define MPU6050_RA_PWR_MGMT_2       0x6C
#define MPU6050_RA_BANK_SEL         0x6D
#define MPU6050_RA_MEM_START_ADDR   0x6E
#define MPU6050_RA_MEM_R_W      0x6F
#define MPU6050_RA_DMP_CFG_1        0x70
#define MPU6050_RA_DMP_CFG_2        0x71
#define MPU6050_RA_FIFO_COUNTH      0x72
#define MPU6050_RA_FIFO_COUNTL      0x73
#define MPU6050_RA_FIFO_R_W         0x74
#define MPU6050_RA_WHO_AM_I         0x75
```

L'MPU6050 necessita di una inizializzazione (la funzione corrispondente è **Setup_MPU6050**, la quale viene richiamata nella fase di setup presente nel main:

```
void Setup_MPU6050()
{
    //Sets sample rate to 8000/1+7 = 1000Hz
    IMU_write(CHANNEL, MPU6050_ADDRESS, MPU6050_RA_SMPLRT_DIV
        , &RATE_1000, NUM_BYTES);
    //Disable FSync, 256Hz DLPF
```

```
    IMU_write(CHANNEL, MPU6050_ADDRESS, MPU6050_RA_CONFIG, &
       DISABLE, NUM_BYTES);
    //Disable gyro self tests, scale of 500 degrees/s
    IMU_write(CHANNEL, MPU6050_ADDRESS,
       MPU6050_RA_GYRO_CONFIG, &D_GYRO_SELF_TEST, NUM_BYTES);
    //Disable accel self tests, scale of +-2g, no DHPF
    IMU_write(CHANNEL, MPU6050_ADDRESS,
       MPU6050_RA_ACCEL_CONFIG, &DISABLE, NUM_BYTES);
    //Freefall threshold of |0mg|
    IMU_write(CHANNEL, MPU6050_ADDRESS, MPU6050_RA_FF_THR, &
       INITIAL, NUM_BYTES);
    //Freefall duration limit of 0
    IMU_write(CHANNEL, MPU6050_ADDRESS, MPU6050_RA_FF_DUR, &
       INITIAL, NUM_BYTES);
    //Motion threshold of 0mg
    IMU_write(CHANNEL, MPU6050_ADDRESS, MPU6050_RA_MOT_THR, &
       INITIAL, NUM_BYTES);
    //Motion duration of 0s
    IMU_write(CHANNEL, MPU6050_ADDRESS, MPU6050_RA_MOT_DUR, &
       INITIAL, NUM_BYTES);
    //Zero motion threshold
    IMU_write(CHANNEL, MPU6050_ADDRESS, MPU6050_RA_ZRMOT_THR,
        &INITIAL, NUM_BYTES);
    //Zero motion duration threshold
    IMU_write(CHANNEL, MPU6050_ADDRESS, MPU6050_RA_ZRMOT_DUR,
        &INITIAL, NUM_BYTES);
    //Disable sensor output to FIFO buffer
    IMU_write(CHANNEL, MPU6050_ADDRESS, MPU6050_RA_FIFO_EN, &
       DISABLE, NUM_BYTES);
```

Una volta completato il Setup, è possibile leggere i vari valori grezzi (in bit) mediante le funzioni **Get_Accel_Values** e **Get_Gyro_Value**, rispettivamente per l'accelerometro e per il giroscopio. I seguenti registri contengono i valori più recenti delle misurazioni dell'accelerometro. (pag.30 del manuale)

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
3B	59	ACCEL_XOUT[15:8]							
3C	60	ACCEL_XOUT[7:0]							
3D	61	ACCEL_YOUT[15:8]							
3E	62	ACCEL_YOUT[7:0]							
3F	63	ACCEL_ZOUT[15:8]							
40	64	ACCEL_ZOUT[7:0]							

Mentre per il giroscopio si hanno

Register (Hex)	Register (Decimal)	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
43	67	GYRO_XOUT[15:8]							
44	68	GYRO_XOUT[7:0]							
45	69	GYRO_YOUT[15:8]							
46	70	GYRO_YOUT[7:0]							
47	71	GYRO_ZOUT[15:8]							
48	72	GYRO_ZOUT[7:0]							

Per entrambi i sensori, i registri di misurazione sono composti di due set di registri:

- un set di registri interni aggiornati in base al tempo di campionamento.
- un set di registri di lettura user-facing che duplica i dati dell'altro set di registri tutte le volte che l'interfaccia seriale é nascosta.

Di seguito verrà presentata una spiegazione dettagliata del procedimento seguito per acquisire i dati reali del solo accelerometro, poiché per il giroscopio le funzioni sono analoghe. Da notare come i parametri ACCEL_*OUT siano valori a 16 bit in complemento a 2.

```
//Gets raw accelerometer data, performs no processing
void Get_Accel_Values(int* ACCEL_XOUT, int* ACCEL_YOUT, int*
    ACCEL_ZOUT)
{
    uint8_t ACCEL_XOUT_H = 0, ACCEL_XOUT_L = 0;
    uint8_t ACCEL_YOUT_H = 0, ACCEL_YOUT_L = 0;
    uint8_t ACCEL_ZOUT_H = 0, ACCEL_ZOUT_L = 0;


    IMU_read(CHANNEL, MPU6050_ADDRESS,
        MPU6050_RA_ACCEL_XOUT_H, &ACCEL_XOUT_H, NUM_BYTES);
    IMU_read(CHANNEL, MPU6050_ADDRESS,
        MPU6050_RA_ACCEL_XOUT_L, &ACCEL_XOUT_L, NUM_BYTES);
    IMU_read(CHANNEL, MPU6050_ADDRESS,
        MPU6050_RA_ACCEL_YOUT_H, &ACCEL_YOUT_H, NUM_BYTES);
    IMU_read(CHANNEL, MPU6050_ADDRESS,
        MPU6050_RA_ACCEL_YOUT_L, &ACCEL_YOUT_L, NUM_BYTES);
    IMU_read(CHANNEL, MPU6050_ADDRESS,
        MPU6050_RA_ACCEL_ZOUT_H, &ACCEL_ZOUT_H, NUM_BYTES);
    IMU_read(CHANNEL, MPU6050_ADDRESS,
        MPU6050_RA_ACCEL_ZOUT_L, &ACCEL_ZOUT_L, NUM_BYTES);

    *ACCEL_XOUT = ((ACCEL_XOUT_H<<8)|ACCEL_XOUT_L);
    *ACCEL_YOUT = ((ACCEL_YOUT_H<<8)|ACCEL_YOUT_L);
    *ACCEL_ZOUT = ((ACCEL_ZOUT_H<<8)|ACCEL_ZOUT_L);

}
```

A questo punto, avendo i valori grezzi, si rendono necessari due ulteriori passi per ottenere quelli reali:

```
//Accelerometer data in G
void Get_Accel_Gravity_Power(float* ACCEL_XPOWER, float*
    ACCEL_YPOWER, float* ACCEL_ZPOWER)
{
    int ACCEL_XOUT;
    int ACCEL_YOUT;
    int ACCEL_ZOUT;

    Get_Accel_Values(&ACCEL_XOUT, &ACCEL_YOUT, &ACCEL_ZOUT);

    *ACCEL_XPOWER = map(C2toD(ACCEL_XOUT), -QL/2, QL/2, -
        MPU6050_ACC_FULL_SCALE_RANGE,
        MPU6050_ACC_FULL_SCALE_RANGE);
//  *ACCEL_YPOWER = map(C2toD(ACCEL_YOUT), -QL/2, QL/2, -
    MPU6050_ACC_FULL_SCALE_RANGE, MPU6050_ACC_FULL_SCALE_RANGE
    );
//  *ACCEL_ZPOWER = map(C2toD(ACCEL_ZOUT), -QL/2, QL/2, -
    MPU6050_ACC_FULL_SCALE_RANGE, MPU6050_ACC_FULL_SCALE_RANGE
    );

    //Z=Y and Y=-Z due to imu rotation in positioning
    *ACCEL_ZPOWER = map(C2toD(ACCEL_YOUT), -QL/2, QL/2, -
        MPU6050_ACC_FULL_SCALE_RANGE,
        MPU6050_ACC_FULL_SCALE_RANGE);
    *ACCEL_YPOWER = -map(C2toD(ACCEL_ZOUT), -QL/2, QL/2, -
        MPU6050_ACC_FULL_SCALE_RANGE,
        MPU6050_ACC_FULL_SCALE_RANGE);
}
```

1. La conversione da bit a dati espressi in accelerazione di gravità, utilizzando, quindi come unitá di misura g ($1g = 9.8m/s^2$). Per questo motivo, si ricorre all'uso di **map** nella funzione **Get_Accel_Gravity_Power** Il valore di QL è settato a 65536, ovvero 2^{16} dove 16 corrisponde al numero di bit. La funzione **map**, basata sulla semplice proporzione

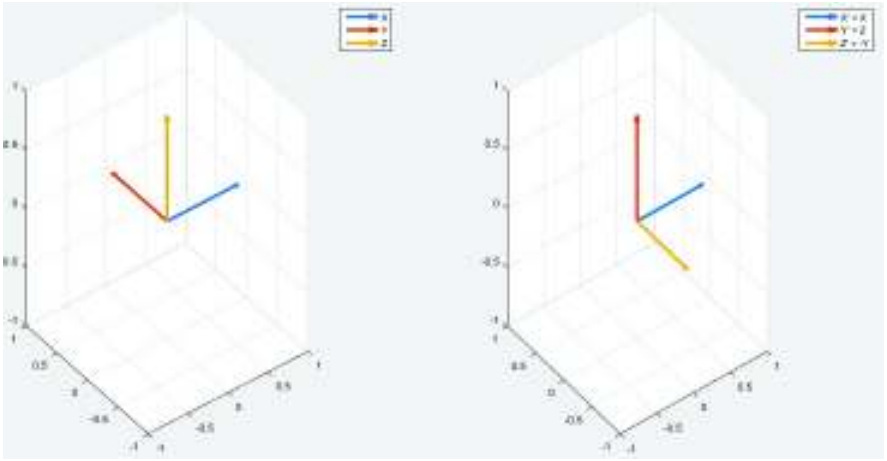
   $$\frac{x - FROM\_DST}{VAL - FROM\_SRC} = \frac{TO\_DST - FROM\_DST}{TO\_SRC - FROM\_SRC}$$

   Cioè:
   $x = \frac{TO\_DST - FROM\_DST}{TO\_SRC - FROM\_SRC} \cdot (VAL - FROM\_SRC) + FROM\_DST$

```
//Maps a value from a range to an other
inline float map (float VAL, float FROM_SRC, float TO_SRC, float FROM_DST, float TO_DST)
{
    return (((TO_DST-FROM_DST)/(TO_SRC-FROM_SRC))*(VAL-FROM_SRC)) + FROM_DST;
}
```

Il range di valori reali è settato mediante la variabile $MPU6050\_ACC\_FULL\_SCALE\_RANGE$, inizializzata a 2G nella libreria $MPU6050.h$: questa definisce, in sostanza, una sorta di sensitività dello strumento. (si rimanda a pag. 30 del manuale).

2. Il secondo passo è quello di applicare una sorta di cambio di coordinate, dovuto dalla posizione in verticale del sensore.



Le nuove corrispondenze saranno perciò (l'uso dell'apice sta ad indicare i nuovi assi):

- X' = X;
- Y' = -Z;
- Z' = Y;

Avendo, in questo modo, i dati reali rapportati all'accelerazione di gravità si calcolano gli angoli di inclinazione necessari per individuare l'angolo di Roll e di Pitch.
$AccX_{angle} = \arctan(-\frac{A_y}{A_z})$
$AccY_{angle} = \arctan(\frac{A_x}{\sqrt{A_y^2+A_z^2}})$
Tali calcoli restituiscono il risultato in radianti, che quindi dovranno essere convertiti in gradi mediante la proporzione:

$$360 : 2\pi = gradi : rad$$
$$gradi = \frac{360*rad}{2*\pi} = \frac{180*rad}{\pi} \simeq 57.295 * rad$$

Quindi, nel caso in esame si avrà

$$Acc*_{gradi} \simeq 57.295 * Acc*_{angle}$$

Tale compito spetta alla funzione **Get_Accel_Angles**.

```
//Converts the already acquired accelerometer data into 3D
   euler angles
void Get_Accel_Angles(float* ACCEL_XANGLE, float*
   ACCEL_YANGLE)
{
//  int ACCEL_XOUT;
//  int ACCEL_YOUT;
//  int ACCEL_ZOUT;
//
//  Get_Accel_Values(&ACCEL_XOUT, &ACCEL_YOUT, &ACCEL_ZOUT);
//
//  *ACCEL_XANGLE = 57.295*atan2(-(float)C2toD(ACCEL_YOUT), (
   float)C2toD(ACCEL_ZOUT));
//  *ACCEL_YANGLE = 57.295*atan2((float)C2toD(ACCEL_XOUT),
   sqrt(pow((float)C2toD(ACCEL_YOUT),2)+pow((float)C2toD(
   ACCEL_ZOUT),2)));

    float ACCEL_XOUT;
    float ACCEL_YOUT;
    float ACCEL_ZOUT;
```

```
    Get_Accel_Gravity_Power(&ACCEL_XOUT, &ACCEL_YOUT, &
        ACCEL_ZOUT);

    *ACCEL_XANGLE = 57.295*atan2(-ACCEL_YOUT, ACCEL_ZOUT);
    *ACCEL_YANGLE = 57.295*atan2(ACCEL_XOUT, sqrt(pow(
        ACCEL_YOUT,2)+pow(ACCEL_ZOUT,2)));
}
```

Come è stato accennato all'inizio di questo paragrafo, per il giroscopio il funzionamento nell'utilizzo delle funzioni **Get_Gyro_Value** e **Get_Gyro_Rates**. C'è tuttavia una particolarità: la necessitá del calcolo di un offset iniziale dovuto all'errore dello strumento o alla possibilità di un movimento già presente al momento dell'inizializzazione.

```
void Calibrate_Gyros()
{
    uint32_t GYRO_XOUT_OFFSET_GYRO_MEASURES_SUM = 0;
    uint32_t GYRO_YOUT_OFFSET_GYRO_MEASURES_SUM = 0;
    uint32_t GYRO_ZOUT_OFFSET_GYRO_MEASURES_SUM = 0;
    //uint8_t GYRO_XOUT_H = 0, GYRO_XOUT_L = 0;
    //uint8_t GYRO_YOUT_H = 0, GYRO_YOUT_L = 0;
    //uint8_t GYRO_ZOUT_H = 0, GYRO_ZOUT_L = 0;

    int GYRO_XOUT_init;
    int GYRO_YOUT_init;
    int GYRO_ZOUT_init;

    float GYRO_XOUT_OFFSET0 = 0;
    float GYRO_YOUT_OFFSET0 = 0;
    float GYRO_ZOUT_OFFSET0 = 0;

    float GYRO_XOUT_OFFSET_MAP;
    float GYRO_YOUT_OFFSET_MAP;
    float GYRO_ZOUT_OFFSET_MAP;

    int iteration = 1;
    do
    {
        GYRO_XOUT_OFFSET0 = GYRO_XOUT_OFFSET_MAP;
        GYRO_YOUT_OFFSET0 = GYRO_YOUT_OFFSET_MAP;
        GYRO_ZOUT_OFFSET0 = GYRO_ZOUT_OFFSET_MAP;

        for(int x = 0; x<GYRO_MEASURES; x++)
        {

            /*
```

```
            IMU_read(CHANNEL, MPU6050_ADDRESS,
               MPU6050_RA_GYRO_XOUT_H, &GYRO_XOUT_H,
               NUM_BYTES);
            IMU_read(CHANNEL, MPU6050_ADDRESS,
               MPU6050_RA_GYRO_XOUT_L, &GYRO_XOUT_L,
               NUM_BYTES);
            IMU_read(CHANNEL, MPU6050_ADDRESS,
               MPU6050_RA_GYRO_YOUT_H, &GYRO_YOUT_H,
               NUM_BYTES);
            IMU_read(CHANNEL, MPU6050_ADDRESS,
               MPU6050_RA_GYRO_YOUT_L, &GYRO_YOUT_L,
               NUM_BYTES);
            IMU_read(CHANNEL, MPU6050_ADDRESS,
               MPU6050_RA_GYRO_ZOUT_H, &GYRO_ZOUT_H,
               NUM_BYTES);
            IMU_read(CHANNEL, MPU6050_ADDRESS,
               MPU6050_RA_GYRO_ZOUT_L, &GYRO_ZOUT_L,
               NUM_BYTES);
            */

            Get_Gyro_Value(&GYRO_XOUT_init, &GYRO_YOUT_init,
               &GYRO_ZOUT_init);

            GYRO_XOUT_OFFSET_GYRO_MEASURES_SUM +=
               GYRO_XOUT_init;
            GYRO_YOUT_OFFSET_GYRO_MEASURES_SUM +=
               GYRO_YOUT_init;
            GYRO_ZOUT_OFFSET_GYRO_MEASURES_SUM +=
               GYRO_ZOUT_init;

        }

    GYRO_XOUT_OFFSET = (float)
       GYRO_XOUT_OFFSET_GYRO_MEASURES_SUM/GYRO_MEASURES;
    GYRO_YOUT_OFFSET = (float)
       GYRO_YOUT_OFFSET_GYRO_MEASURES_SUM/GYRO_MEASURES;
    GYRO_ZOUT_OFFSET = (float)
       GYRO_ZOUT_OFFSET_GYRO_MEASURES_SUM/GYRO_MEASURES;

    GYRO_XOUT_OFFSET_MAP = map(C2toD(GYRO_XOUT_OFFSET), -QL
       /2, QL/2, -MPU6050_GYROC_FULL_SCALE_RANGE,
       MPU6050_GYROC_FULL_SCALE_RANGE);
    GYRO_YOUT_OFFSET_MAP = map(C2toD(GYRO_YOUT_OFFSET), -QL
       /2, QL/2, -MPU6050_GYROC_FULL_SCALE_RANGE,
       MPU6050_GYROC_FULL_SCALE_RANGE);
    GYRO_ZOUT_OFFSET_MAP = map(C2toD(GYRO_ZOUT_OFFSET), -QL
       /2, QL/2, -MPU6050_GYROC_FULL_SCALE_RANGE,
       MPU6050_GYROC_FULL_SCALE_RANGE);
```

```
    char a[20];
    sprintf(a, "X: %4.2f", GYRO_XOUT_OFFSET_MAP);
    lcd_display(LCD_LINE4, (const uint8_t*)a);
    sprintf(a, "Y: %4.2f", GYRO_YOUT_OFFSET_MAP);
    lcd_display(LCD_LINE5, (const uint8_t*)a);
    sprintf(a, "Z: %4.2f", GYRO_ZOUT_OFFSET_MAP);
    lcd_display(LCD_LINE6, (const uint8_t*)a);
    sprintf(a, "iter: %2d", iteration);
    lcd_display(LCD_LINE7, (const uint8_t*)a);

    iteration++;

} while((abs(GYRO_XOUT_OFFSET0 - GYRO_XOUT_OFFSET_MAP) >
   OFFSET_ERROR) || (abs(GYRO_YOUT_OFFSET0 -
   GYRO_YOUT_OFFSET_MAP) > OFFSET_ERROR) || (abs(
   GYRO_ZOUT_OFFSET0 - GYRO_ZOUT_OFFSET_MAP) > OFFSET_ERROR))
   ;
}
```

Si calcola il valore medio di un certo numero di misurazioni (GYRO_MEASURES = 100) per ogni direzione. Supponendolo come offset, viene confrontato con il valore acquisito come offset nell'istante precedente. Fino a che la differenza tra i due non sarà minore di una certa soglia si prosegue con il calcolo di un nuovo offset e il confronto con quello precedente. Nel momento in cui verrà soddisfatta tale diseguaglianza si avrà un offset molto vicino a quello reale e quindi la calibrazione termina.

```
//Function to read the gyroscope rate data and convert it
   into degrees/s
void Get_Gyro_Value(int* GYRO_XOUT, int* GYRO_YOUT, int*
   GYRO_ZOUT)
{
    uint8_t GYRO_XOUT_H = 0, GYRO_XOUT_L = 0;
    uint8_t GYRO_YOUT_H = 0, GYRO_YOUT_L = 0;
    uint8_t GYRO_ZOUT_H = 0, GYRO_ZOUT_L = 0;

    IMU_read(CHANNEL, MPU6050_ADDRESS, MPU6050_RA_GYRO_XOUT_H
       , &GYRO_XOUT_H, NUM_BYTES);
    IMU_read(CHANNEL, MPU6050_ADDRESS, MPU6050_RA_GYRO_XOUT_L
       , &GYRO_XOUT_L, NUM_BYTES);
    IMU_read(CHANNEL, MPU6050_ADDRESS, MPU6050_RA_GYRO_YOUT_H
       , &GYRO_YOUT_H, NUM_BYTES);
    IMU_read(CHANNEL, MPU6050_ADDRESS, MPU6050_RA_GYRO_YOUT_L
       , &GYRO_YOUT_L, NUM_BYTES);
    IMU_read(CHANNEL, MPU6050_ADDRESS, MPU6050_RA_GYRO_ZOUT_H
       , &GYRO_ZOUT_H, NUM_BYTES);
    IMU_read(CHANNEL, MPU6050_ADDRESS, MPU6050_RA_GYRO_ZOUT_L
       , &GYRO_ZOUT_L, NUM_BYTES);
```

```
    *GYRO_XOUT = ((GYRO_XOUT_H<<8)|GYRO_XOUT_L) -
       GYRO_XOUT_OFFSET;
    *GYRO_YOUT = ((GYRO_YOUT_H<<8)|GYRO_YOUT_L) -
       GYRO_YOUT_OFFSET;
    *GYRO_ZOUT = ((GYRO_ZOUT_H<<8)|GYRO_ZOUT_L) -
       GYRO_ZOUT_OFFSET;
}
```

Nella funzione **Get_Gyro_Value** l'Offset grezzo (non mappato) viene poi sottratto dai dati acquisiti. Quindi, come per l'accelerometro, mediante **Get_Gyro_Rates** si possono ricavare tutti i valori reali espressi in $gradi/s$ tenendo presente anche in questo caso il discorso sul mapping (dal valore in bit a gradi/s utilizzando come $MPU6050\_GYRO\_FULL\_SCALE\_RANGE$ il valore 250) e quello riguardo al cambio di coordinate dovuto alla rotazione della IMU.

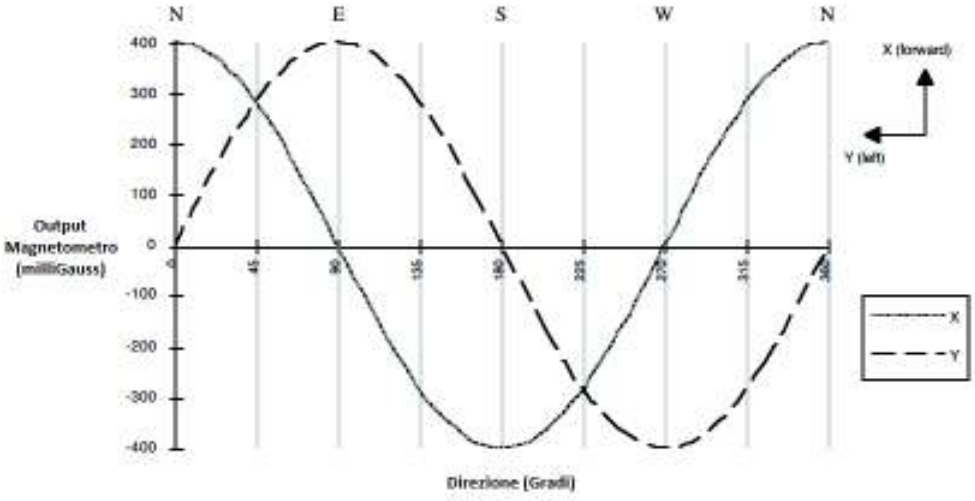
14.4 IMU: Magnetometro

14.4.1 Principio di funzionamento

I magnetometri, spesso definiti bussole elettroniche, sono quei dispositivi con i quali è possibile rilevare istantaneamente l'orientamento di un oggetto, prendendo a riferimento le direzioni Nord, Sud, Est, Ovest.
Il funzionamento è basato sul principio della variazione della resistenza di un materiale contenente elementi ferrosi quando è sottoposto a un campo magnetico ad esso perpendicolare, come può essere quello terrestre.
Nel caso dei magnetometri MEMS, essi presentano dimensioni dell'ordine dei micrometri e le variazioni sono infinitesime e appena percettibili. Il materiale sensibile al campo magnetico è disposto in modo opportuno e collegato per formare un classico circuito a ponte di Wheatstone, dal quale si rilevano con precisione le variazioni di resistenza.
Anche nel caso di questo componente, i dati che possiamo ricavare per mezzo del funzionamento sopra descritto, rappresentano le proiezioni di campo magnetico relative al riferimento terrestre, che sappiamo essere un campo con direzione Nord.
Chiamiamo i tre contributi M_x, M_y, M_z. Per eseguire una prova della correttezza del codice, è possibile basarsi sul riferimento del Nord magnetico terrestre, ruotando successivamente il dispositivo attorno al proprio asse longitudinale. Il risultato di tale operazione sarà la progressiva modifica delle componenti sopra descritte, grazie alle quali, dopo opportune operazioni matematiche, sarà possibile mappare la direzione ("heading") del dispositivo su di una circonferenza di 360°. La direzione del nostro drone può essere stimata analizzando le sole componenti M_x e M_y. Occorre precisare che questa

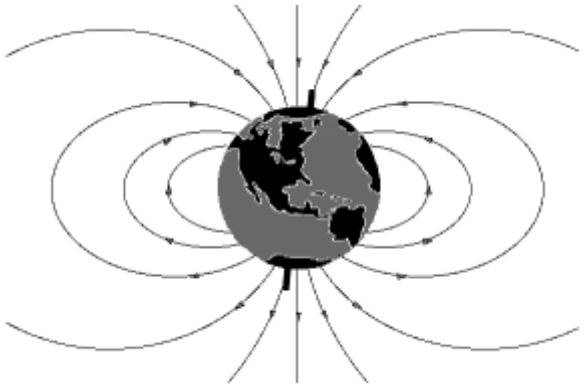
possibilità rappresenta un'eccezione, in quanto tale valore è veritiero solo nel caso in cui l'IMU sia parallela al terreno.

- $Direzione(y > 0) = 90 - 180\frac{arctan(\frac{x}{y})}{\pi}$
- $Direzione(y < 0) = 270 - 180\frac{arctan(\frac{x}{y})}{\pi}$
- $Direzione(y = 0, x < 0) = 180$
- $Direzione(y = 0, x > 0) = 0$



Al fine di ottenere una maggiore accuratezza nelle misurazioni, ovviando così alle varie anomalie che ogni dispositivo presenta in determinate circostanze, si è scelto di mixare i valori del campo magnetico con quelli delle accelerazioni ed ottenere così un risultato stabile e performante per mezzo di filtri complementari.

Il valore massimo di M_x e M_y dipende dalla forza del campo magnetico terrestre nel punto in cui viene calcolato. Il campo magnetico terrestre può essere rappresentato come un semplice dipolo magnetico.



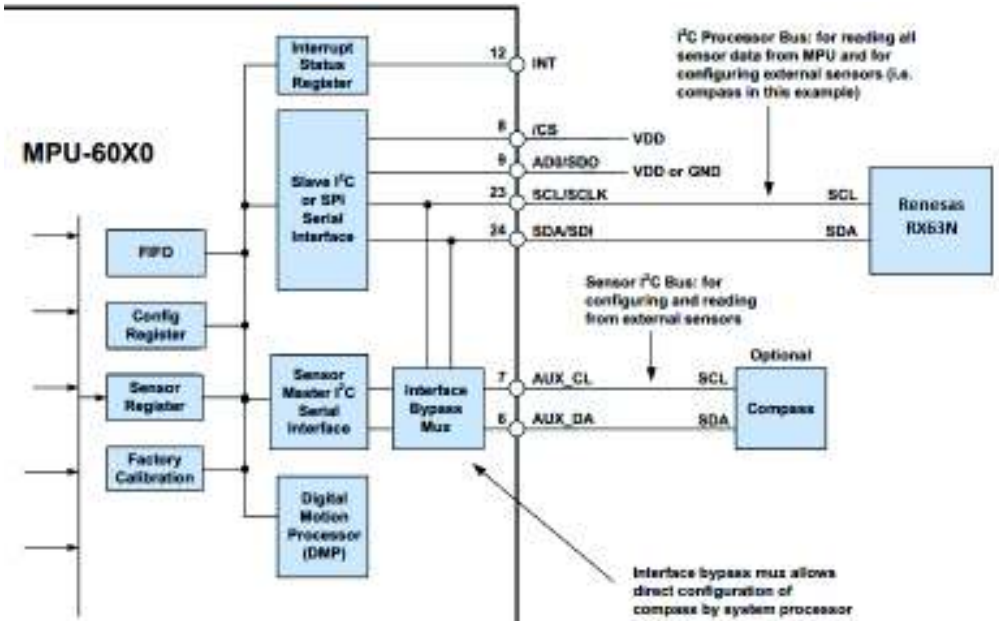
In quanto tale, effettuando una misurazione di tale campo, esso non sarà parallelo al terreno, perciò è necessario introdurre un angolo di declinazione, per correggere tale valore. Tramite il sito *www.magnetic-declination.com* abbiamo ricavato il valore dell'angolo correttivo da sommare o sottrarre per ottenere un valore corretto. Per quanto riguarda Ancona, l'angolo approssimativo è di +4.0°.
In generale l'angolo di PITCH e/o ROLL, del dispositivo, sono diversi da 0. Questa casistica ha richiesto una compensazione di tipo software, il **tilt compensation**. L'algoritmo è basato su calcoli trigonometrici, i quali offrono la possibilità di proiettare il campo magnetico rilevato, sul piano XY in maniera tale da ricondurci alla situazione precedentemente descritta.

14.4.2 Calibrazione Magnetometro

L'HMC-5883L possiede tre offsets, uno per ogni asse. Per annullare tali valori si é pensato di fissare il magnetometro in una determinata posizione e leggere i valori M_x, M_y e M_z. Successivamente é stata eseguita una rotazione, attorno al proprio asse Z, di 180° ed é stata effettuata una nuova lettura. Quello che ci si aspetta é che il dispositivo legga due valori dello stesso modulo, ma verso opposto, quindi tramite una semplice media aritmetica é stato possibile ricavare i valori di offsets da sottrarre ai dati letti ad ogni campionamento.

14.4.3 HMC-5883L Driver

Il magnetometro preso in considerazione é l'HMC-5883L, integrato nell'IMU Drotek V2. La comunicazione tra magnetometro e microcontrollore avviene tramite un'I^2C ausiliaria gestita dall'MPU-6050. Quest'ultimo agisce da master, mentre l'HMC-5883L da slave. Per questo motivo per effettuare una lettura dei registri del sensore è necessario leggere i registri dedicati ai "sensori esterni".



L'HMC-5883L possiede 13 indirizzi. Per il nostro lavoro sono stati indispensabili i primi tre per la fase di configurazione e i successivi sei per la lettura dei dati.

Address Location	Name	Access
00	Configuration Register A	Read/Write
01	Configuration Register B	Read/Write
02	Mode Register	Read/Write
03	Data Output X MSB Register	Read
04	Data Output X LSB Register	Read
05	Data Output Z MSB Register	Read
06	Data Output Z LSB Register	Read
07	Data Output Y MSB Register	Read
08	Data Output Y LSB Register	Read
09	Status Register	Read
10	Identification Register A	Read
11	Identification Register B	Read
12	Identification Register C	Read

La prima sezione del codice riguarda il setting dei registri del magnetometro, rispettivamente i registri che definisco in numero di campionamenti effettuati ad ogni misura, la frequenza di campionamento, il guadagno dello strumento e la modalitá di lettura (si rimanda a pag. 12-14 del datasheet).

```
void Setup_HMC5883L()
{
    //I2C_MST_EN = 1    I2C master mode enabled (MPU6050
       Register Map pag. 38)
    IMU_write(CHANNEL, MPU6050_ADDRESS, MPU6050_RA_USER_CTRL,
        &CONFIG_6A, NUM_BYTES);
    //Set the MPU6050 as the master I2C rate (400kHz)
    IMU_write(CHANNEL, MPU6050_ADDRESS,
       MPU6050_RA_I2C_MST_CTRL, &CONFIG_MST_CTR, NUM_BYTES);

    IMU_write_AUXMASTER(HMC5883L_REG_A,CONFIG_REG_A);
    IMU_write_AUXMASTER(HMC5883L_REG_B,CONFIG_REG_B);
    IMU_write_AUXMASTER(HMC5883L_MODE_REG,CONFIG_MODE_REG);

    //Set the read mode
    //Configure the address used to specify the I2C slave
       address of Slave 0 (magnetometer): read mode
    IMU_write(CHANNEL, MPU6050_ADDRESS,
       MPU6050_RA_I2C_SLV0_ADDR, &CONFIG_SLV0_ADDR_R,
       NUM_BYTES);
    //Data transfer starts from this register within Slave 0
    IMU_write(CHANNEL, MPU6050_ADDRESS,
       MPU6050_RA_I2C_SLV0_REG , &CONFIG_SLV0_REG_DATA,
       NUM_BYTES);
    //Enable sub equipment operation
    IMU_write(CHANNEL, MPU6050_ADDRESS,
       MPU6050_RA_I2C_SLV0_CTRL, &CONFIG_SLV0_CTRL_2,
       NUM_BYTES);
    //Enable data ready interrupt
    IMU_write(CHANNEL, MPU6050_ADDRESS, MPU6050_RA_INT_ENABLE
       , &ENABLE_REG, NUM_BYTES);

}

void IMU_write_AUXMASTER(uint8_t CONFIG_SLV0_REG, uint8_t
   CONFIG_SLV0_DO)
{
    //Configure the address used to specify the I2C slave
       address of Slave 0 (magnetometer): write mode
    IMU_write(CHANNEL, MPU6050_ADDRESS,
       MPU6050_RA_I2C_SLV0_ADDR, &CONFIG_SLV0_ADDR_W,
       NUM_BYTES);
    //Data transfer starts from this register within Slave 0
    IMU_write(CHANNEL, MPU6050_ADDRESS,
       MPU6050_RA_I2C_SLV0_REG , &CONFIG_SLV0_REG, NUM_BYTES)
       ;
    //This register holds the output data written into Slave
       0 when Slave 0 is set to write mode
    IMU_write(CHANNEL, MPU6050_ADDRESS,
```

```
        MPU6050_RA_I2C_SLV0_DO, &CONFIG_SLV0_DO, NUM_BYTES);
    //Set the data operation number
    IMU_write(CHANNEL, MPU6050_ADDRESS,
        MPU6050_RA_I2C_SLV0_CTRL, &CONFIG_SLV0_CTRL_1,
        NUM_BYTES);
    //Enable sub equipment operation
    IMU_write(CHANNEL, MPU6050_ADDRESS,
        MPU6050_RA_I2C_SLV0_CTRL, &CONFIG_SLV0_CTRL_Enable,
        NUM_BYTES);
}
```

L'effettiva scrittura di questi registri viene gestita dall'MPU6050, tramite la funzione **IMU_write_AUXMASTER** (si rimanda al sito *http : //www.programering.com/a/M*

Il magnetometro, in base alla frequenza di output selezionata, ripone i dati negli appositi registri, i quali verranno letti dall'MPU6050 (pag. 32 MPU6050 Register Map and Description)

```
void Get_Mag_Value(uint16_t* MAG_XOUT, uint16_t* MAG_YOUT,
    uint16_t* MAG_ZOUT)
{
    uint8_t MAG_XOUT_H = 0, MAG_XOUT_L = 0;
    uint8_t MAG_YOUT_H = 0, MAG_YOUT_L = 0;
    uint8_t MAG_ZOUT_H = 0, MAG_ZOUT_L = 0;

    IMU_read(CHANNEL, MPU6050_ADDRESS, HMC5883L_DO_X_MSB_REG,
        &MAG_XOUT_H, NUM_BYTES);
    IMU_read(CHANNEL, MPU6050_ADDRESS, HMC5883L_DO_X_LSB_REG,
        &MAG_XOUT_L, NUM_BYTES);
    IMU_read(CHANNEL, MPU6050_ADDRESS, HMC5883L_DO_Y_MSB_REG,
        &MAG_YOUT_H, NUM_BYTES);
    IMU_read(CHANNEL, MPU6050_ADDRESS, HMC5883L_DO_Y_LSB_REG,
        &MAG_YOUT_L, NUM_BYTES);
    IMU_read(CHANNEL, MPU6050_ADDRESS, HMC5883L_DO_Z_MSB_REG,
        &MAG_ZOUT_H, NUM_BYTES);
    IMU_read(CHANNEL, MPU6050_ADDRESS, HMC5883L_DO_Z_LSB_REG,
        &MAG_ZOUT_L, NUM_BYTES);

    *MAG_XOUT = ((MAG_XOUT_H<<8)|MAG_XOUT_L);
    *MAG_YOUT = ((MAG_YOUT_H<<8)|MAG_YOUT_L);
    *MAG_ZOUT = ((MAG_ZOUT_H<<8)|MAG_ZOUT_L);

}

void Get_Mag_Value_Normalized(float* MAG_X_NORM, float*
    MAG_Y_NORM, float* MAG_Z_NORM)
{
    uint16_t MAG_X_OUT;
    uint16_t MAG_Y_OUT;
```

```
    uint16_t MAG_Z_OUT;

    Get_Mag_Value(&MAG_X_OUT, &MAG_Y_OUT, &MAG_Z_OUT);

    *MAG_X_NORM = (float)C2toD(MAG_X_OUT)*
       HMC5883L_MG_PER_DIGIT - MAG_XOUT_OFFSET;
    *MAG_Y_NORM = (float)C2toD(MAG_Y_OUT)*
       HMC5883L_MG_PER_DIGIT - MAG_YOUT_OFFSET;
    *MAG_Z_NORM = (float)C2toD(MAG_Z_OUT)*
       HMC5883L_MG_PER_DIGIT - MAG_ZOUT_OFFSET;

}
```

In base alle letture effettuate sará necessaria la compensazione sopra citata, la quale viene richiamata nel main e restituirá la direzione in gradi, sulla base del campo magnetico terrestre

```
void Get_Mag_Heading_Compensated (float* Heading, float
   RollAngle, float PitchAngle)
{
    float MAGX_OUT, MAGY_OUT, MAGZ_OUT;
    float cosRoll, sinRoll, cosPitch, sinPitch;
    float Xh, Yh;

    Get_Mag_Value_Normalized(&MAGX_OUT, &MAGY_OUT, &MAGZ_OUT)
       ;
    cosRoll = cos(RollAngle);
    sinRoll = sin(RollAngle);
    cosPitch = cos(PitchAngle);
    sinPitch = sin(PitchAngle);
    Xh = (MAGX_OUT * cosRoll) + (MAGY_OUT * sinPitch *
       sinRoll) - (MAGZ_OUT * cosPitch * sinRoll);
    Yh = (MAGY_OUT * cosPitch) - (MAGZ_OUT * sinPitch);


    *Heading = atan2(Yh, Xh);
//  float declinationAngle = (4.0 + (26.0 / 60.0)) / (180 /
   PI);
//  *Heading += declinationAngle;
    // Correct for heading < 0deg and heading > 360deg
    if (*Heading < 0)
    {
        *Heading += 2 * PI;
    }

    if (*Heading > 2 * PI)
    {
        *Heading -= 2 * PI;
    }
    *Heading *= 180/PI;
```

```
}
```

La funzione **Get_Mag_Heading_Compensated** necessita di tre parametri:

- Heading: variabile in cui verrá scritta la direzione;
- RollAngle: l'angolo di Roll ottenuto dall'accelerometro;
- PitchAngle: l'angolo di Pitch ottenuto dall'accelerometro.