



IMU 2018-2019
UNIVERSITÁ POLITECNICA DELLE MARCHE
Laboratorio di Automazione

Studenti:
Laura Fratini
Luca Mezzanotti
Alex Giacomini

Professore:
Andrea Bonci

8 giugno 2019

Indice

1	Introduzione	3
2	Hardware	4
2.1	Renesas YRDKRX63N	4
2.2	GY-86	4
2.2.1	MPU6050	4
2.2.2	Accelerometro MEMS	5
2.2.3	Giroscopio MEMS	7
2.2.4	Combinazione accelerometro MEMS e giroscopio MEMS	9
2.3	Collegamenti circuitali	9
3	Software	10
3.1	<i>e² studio</i>	10
3.2	Architettura	10
3.3	Inizializzazione IMU	11
3.3.1	Descrizione MAIN	12
3.4	Calibrazione IMU	13
3.5	AHRS - Spiegazione teorica	14
3.6	AHRS - Spiegazione pratica	14
4	Test IMU su e-bike	16
5	Modello Solid-Works e-bike	18
6	Conclusioni	20

1 Introduzione

Scopo di tale progetto é quello di studiare come, sfruttando una tecnologia a microcontrollore, sia possibile ricavare da ulteriori periferiche, alcuni valori utilizzabili nella stabilizzazione di un qualsiasi oggetto preso in esame. Nello specifico, ci siamo basati su una board RENESAS YRDKRX63N dotata di microcontrollore a 32bit. Come periferica esterna abbiamo utilizzato una board modello GY-86, dotata di accelerometro, giroscopio e magnetometro. Combinando appropiatamente i valori ottenuti dalle singole componenti della GY-86, si riescono ad ottenere gli angoli di roll, pitch, yaw, le accellerazione e le velocitá angolari nelle tre direzioni.

Tutto il progetto é stato sviluppato partendo da codici pre-esistenti come ad esempio le librerie fornite direttamente dalla casa produttrice della board GY-86 e da progetti sviluppati nello stesso laboratorio da studenti di anni passati. Il nostro compito é stato quello di sviluppare un pacchetto con delle strutture ben definite, utilizzabili da chiunque voglia inserire all'interno del proprio progetto una IMU ovvero Inertial Measurement Unit.

2 Hardware

2.1 Renesas YRDKRX63N

È il controller su cui è basato l'intero progetto. Su tale scheda, montato un μ -controller RX63N ad alte prestazioni. La board incorpora un core della famiglia RX631 che include:

- 32-bit MCU capace di operare oltre i 100 MHz;
- FPU (Floating-PointUnit) per i calcoli aritmetici;
- Oltre 21 canali per ADC a 12-bit e oltre 2 canali per DAC;
- Unit Timers MTU2 [Multi Timer Unit Function], con funzioni di:
 - Input capture;
 - Output compare;
 - Counter clearing per generazione di segnali PWM;
 - Controllo motori;
- Watchdog timer Indipendente e funzione CRC per lo standard di applicazioni domestiche (IEC 60730) for Europe;
- Molte funzioni di comunicazione: Ethernet, SCI, RSPI, CAN, and *I2C*.

L'elemento della scheda che abbiamo maggiormente sfruttato è la tipologia di comunicazione attraverso il bus. In particolare si è utilizzato il protocollo di comunicazione *I2C*.

2.2 GY-86

Piccola scheda basata sul sensore della Invensense MPU-6050 (accelerometro a 3 assi e giroscopio a 3 assi), sulla bussola digitale a 3 assi HMC5883L della Honeywell e sul sensore di pressione ad alta risoluzione MS5611 della MEAS. Sfutta il protocollo di comunicazione standard *I2C*.

Pu essere utilizzata per vari scopi tra cui:

- Rilevazione movimenti
- Realtà aumentata
- Stabilizzazione elettronica delle immagini
- Stabilizzazione ottica dell'immagine

Nello specifico il nostro utilizzo è stato quello di rilevare gli spostamenti della board stessa nello spazio. Non abbiamo, per motivi di necessità, dovuto utilizzare il sensore di pressione e temperatura.

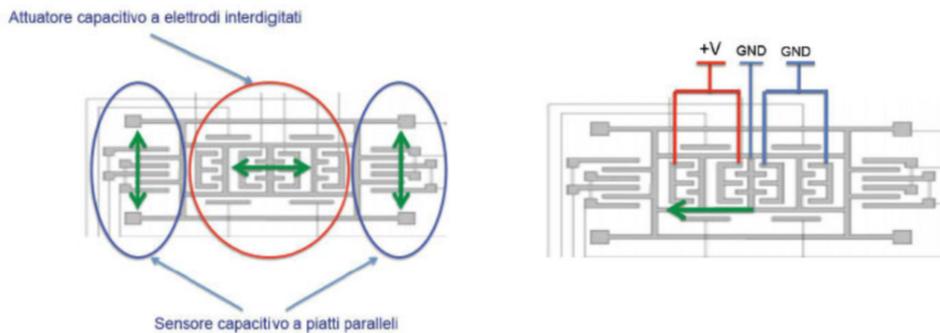
2.2.1 MPU6050

È un chip costituito dal giroscopio e accelerometro. È un dispositivo MotionTracking a 6 assi integrato che combina un giroscopio a tre assi, un accelerometro a 3 assi, e un Digital Motion Processor. Tutti questi sensori sono contenuti in un chip di dimensioni ridotte quali 4x4x0.9mm.

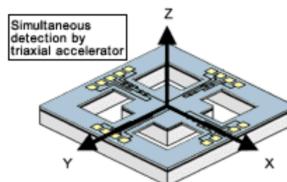
La famiglia di MPU-60X0 dispone di:

- Tre convertitori analogici - digitali (ADC) a 16 bit per la digitalizzazione delle uscite del giroscopio
- Tre convertitori analogici - digitali (ADC) a 16 bit per la digitalizzazione delle uscite dell'accelerometro

Inoltre possibile specificare quale deve essere la sensibilità del giroscopio pari $\pm 250, \pm 500, \pm 1000, \pm 2000$ [gradi]/[secondi] ed anche quale deve essere la sensibilità dell'accelerometro pari a $\pm 2g, \pm 4g, \pm 8g, \pm 16g$. La comunicazione con tutti i registri del dispositivo, viene eseguita utilizzando *I2C* a 400kHz . All'interno del chip **MPU6050**, possiamo trovare un giroscopio di tipo **MEMS** il cui principio di funzionamento si basa su delle masse vibranti che sfruttano le accelerazioni d'inerzia, le quali nascono per effetto del moto del sensore rispetto ad un sistema di riferimento non inerziale. In particolare il sensore è costituito da due "pettini" opportunamente incastriati, su cui scorre della corrente. Il funzionamento si basa sul fatto che la distanza tra le due superfici dei "pettini" cambia, e dunque varia la capacità tra di essi e di conseguenza la differenza di potenziale che si viene a creare.



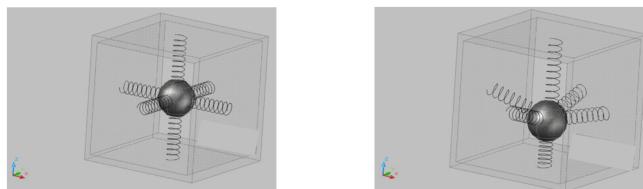
2.2.2 Accelerometro MEMS



È un particolare tipo di trasduttore, in grado di **misurare accelerazioni** lungo la direzione di un proprio asse. Nel nostro caso, si ha a disposizione un dispositivo **triaxiale**. L'accelerometro può essere utilizzato per vari scopi tra cui:

- Misure di accelerazione in una o più direzioni
- Misure di posizione, dedotte dalle accelerazioni per integrazione
- Sensori di orientamento: tilt sensing
- Sensori di vibrazione

Per compiere tali misurazioni, ci sono diverse tipologie, la più diffusa quella a tecnologia **MEMS (Micro Electro Mechanical System)**. Essa si basa sul principio della *massa – molla*.

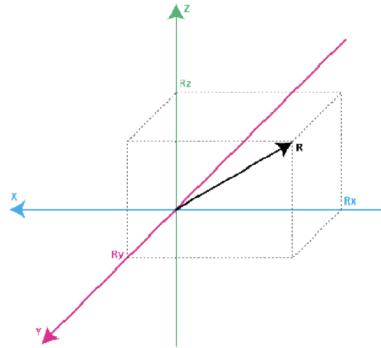


Si può notare come nella figura a destra, la massa risulta essere centrata all'interno del cubo e dunque le molle sono a riposo. Nell'immagine a sinistra invece, si vede come la massa è spostata verso il basso e dunque la molla

sottostante risulta essere compressa e quella sovrastante risulta in tensione. Misurando l'allungamento della molla Δx si riesce a determinare l'accelerazione posseduta dalla massa.

In pratica, il sensore, considera lo stimolo di accelerazione sotto forma di forza inerziale, su tutti e tre gli assi. Rappresentando ora tali forze su di un sistema di riferimento solidale con il sensore stesso, riusciamo a scomporre il vettore forze inerziale R , nelle sue componenti R_x, R_y, R_z come in figura sottostante.

È necessario utilizzare delle formule per convertire i valori letti direttamente dal sensore tramite il **modulo ADC**,



in valori espressi in **g** ossia in (m/s^2) . Nello specifico:

$$R_x \approx \frac{\left(\frac{ADCR_x \cdot V_{ref}}{QL - V_{0G}} \right)}{SensitivityAcc} \quad (1)$$

$$R_y \approx \frac{\left(\frac{ADCR_y \cdot V_{ref}}{QL - V_{0G}} \right)}{SensitivityAcc} \quad (2)$$

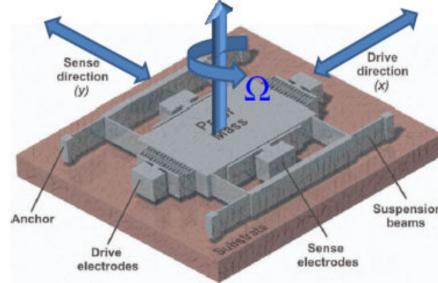
$$R_z \approx \frac{\left(\frac{ADCR_z \cdot V_{ref}}{QL - V_{0G}} \right)}{SensitivityAcc} \quad (3)$$

Dove:

- ADC_{Acc} : valori che escono dal giroscopio, rappresentano le accelerazioni lungo gli assi X, Y, Z
- V_{ref} : tensione di riferimento del modulo ADC, nel nostro caso (GY-86) 3.3V
- QL : livello di quantizzazione dell' ADC, è l'utente ad impostarlo e può valere da 8 a 16 bit.
- V_{0G} : è la tensione che esce dal accelerometro quando non è sottoposto ad accelerazioni, in genere si ha valori del tipo 1.65V.
- $SensitivityAcc$: è la sensibilità del accelerometro. Valore reperibile nel datasheet.

Il sensore stesso, può essere utilizzato, come nel nostro caso, per misurare l'inclinazione del dispositivo sfruttandolo in modalità **tilt sensing**. In questo particolare tipo di utilizzo si misura non l'accelerazione dinamica, ma bensì quella statica.

2.2.3 Giroscopio MEMS



In generale il giroscopio, misurando un'accelerazione rileva una tensione secondo l'equazione:

$$V = S \cdot (\hat{n}^T \omega) + O \quad (4)$$

Dove:

- S = sensibilitá del giroscopio
- O = valore di offset
- \hat{n} = versore che individua l'asse sensibile
- ω = vettore delle rotazioni attorno ai tre assi del riferimento assoluto

Definendo z l'asse soggetto alla rotazione, si ha che:

$$V = S \cdot (n_x \ n_y \ n_z) \cdot \begin{pmatrix} 0 \\ 0 \\ \omega_z \end{pmatrix} + O \quad (5)$$

Inoltre se l'asse del giroscopio é allineato con l'asse Z_0 la formula (2) diventa:

$$V = S \cdot \omega_z + O \quad (6)$$

tale formula **rappresenta il segnale rilevato dal giroscopio MEMS**.

Conoscendo il segnale V , é possibile risalire alla **velocitá angolare** ω_z come:

$$\omega_z = \frac{V - O}{S} \quad (7)$$

Per il calcolo degli angoli di rotazione del giroscopio, basta integrare opportunamente la velocitá di rotazione angolare che é stata ricavata nella formula (4).

$$\theta_z = \int_{t_1}^{t_2} \omega_z dt \quad (8)$$

dove t_1 e t_2 sono gli istanti di inizio e fine acquisizione dei dati.

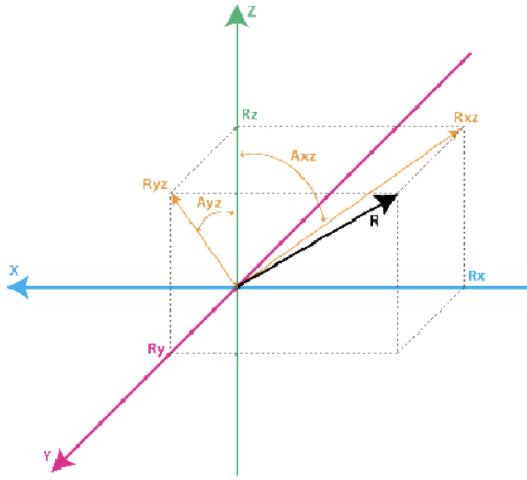
Se ora si sostituisce il valore di ω_z nell'espressione sovrastante e si integra, si ottiene l'angolo desiderato.

$$\theta_z = \int_{t_1}^{t_2} \frac{V - O}{S} dt = \frac{1}{S} \int_{t_1}^{t_2} (V - O) dt \quad (9)$$

Data la presenza di integrali, si va incontro al cosiddetto **fenomeno della deriva**. La presenza di rumore o interferenze, viene considerata all'interno dell'integrale stesso, che pertanto risulta non veritiero, generando un angolo che non rispecchia la realtá. Per questo motivo si combina l'uso di un accelerometro, il quale viene utilizzato come **inclinometro o tilt sensing**.

In pratica, se sottoponiamo il giroscopio ad una rotazione, su di esso sar applicato un vettore **forza inerziale** R , centrato sull'origine del sistema di riferimento non inerziale. Volendo ora trasformare tale vettore in valori numerici di velocit angolari e di conseguenza angoli consideriamo tale procedimento:

Dove:



- R_{xy} proiezione del vettore forza inerziale R sul piano XY
- R_{yz} proiezione del vettore forza inerziale R sul piano YZ
- A_{xz} angolo compreso tra R_{xz} e l'asse Z
- A_{yz} angolo compreso tra R_{yz} e l'asse Z

Se consideriamo che ogni uscita del giroscopio misura le velocità angolari $\omega_x, \omega_y, \omega_z$, intorno agli assi coordinati X, Y, Z ; tali velocità possono essere espresse come variazione dei rispettivi angoli, in particolare:

$$\omega_x = \frac{dA_{yz}}{dt} \approx \frac{\Delta A_{yz}}{\Delta t} = \frac{A_{yz}(t_i) - A_{yz}(t_{i-1})}{(t_i - t_{i-1})} \quad (10)$$

$$\omega_y = \frac{dA_{xz}}{dt} \approx \frac{\Delta A_{xz}}{\Delta t} = \frac{A_{xz}(t_i) - A_{xz}(t_{i-1})}{(t_i - t_{i-1})} \quad (11)$$

$$\omega_z = \frac{dA_{xy}}{dt} \approx \frac{\Delta A_{xy}}{\Delta t} = \frac{A_{xy}(t_i) - A_{xy}(t_{i-1})}{(t_i - t_{i-1})} \quad (12)$$

I valori delle uscite non ritornano mai sotto forma di velocità, ma necessitano di una conversione. Va acquisito il dato proveniente dall' ADC-Converter(ADC), già presente all'interno della scheda Renesas RDKYRX63N e portato in valore di velocità angolare ω ($^{\circ}/s$ o rad/s) attraverso le seguenti formule:

$$\omega_x \approx \frac{\Delta A_{yz}}{\Delta t} = \frac{\left(\frac{ADCGyroYZ \cdot V_{ref}}{QL - V_{0Rate}} \right)}{SensitivityGyro} \quad (13)$$

$$\omega_y \approx \frac{\Delta A_{xz}}{\Delta t} = \frac{\left(\frac{ADCGyroXZ \cdot V_{ref}}{QL - V_{0Rate}} \right)}{SensitivityGyro} \quad (14)$$

$$\omega_z \approx \frac{\Delta A_{xy}}{\Delta t} = \frac{\left(\frac{ADCGyroXY \cdot V_{ref}}{QL - V_{0Rate}} \right)}{SensitivityGyro} \quad (15)$$

Dove:

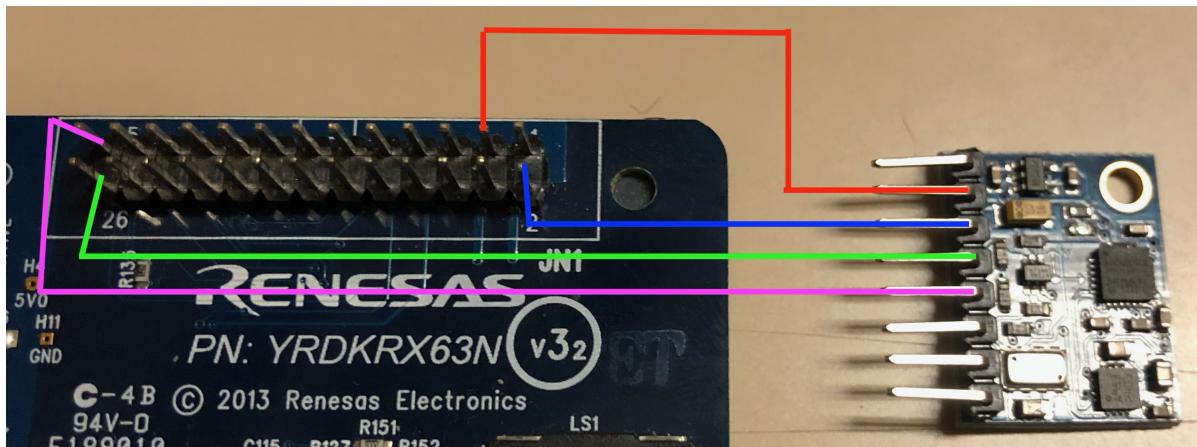
- $ADCGyro$: valori che escono dal giroscopio, rappresentano gli angoli attorno agli assi X, Y, Z
- V_{ref} : tensione di riferimento del modulo ADC, nel nostro caso (GY-86) $3.3V$
- QL : livello di quantizzazione dell' ADC, è l'utente ad impostarlo e può valere da 8 a 16 bit.
- V_{0Rate} : è la tensione che esce dal giroscopio quando non è sottoposto a rotazioni, in genere si ha valori del tipo $1.23V$.
- $SensitivityGyro$: è la sensibilità del giroscopio. Valore reperibile nel datasheet.

2.2.4 Combinazione accelerometro MEMS e giroscopio MEMS

Essendo la scheda stessa sottoposta alla forza di gravità, la misura dell'accelerometro, non risulta essere veritiera. Inoltre l'accelerometro risente costantemente di disturbi dovuti all'ambiente esterno, per questi motivi un modulo IMU, utilizza in combinazione giroscopio e accelerometro per ovviare a tali problematiche. Per combinare opportunamente tali letture, si devono applicare determinati **algoritmi di filtraggio**. Nel nostro caso **AHRS**. (Si veda il capitolo dedicato.)

2.3 Collegamenti circuituali

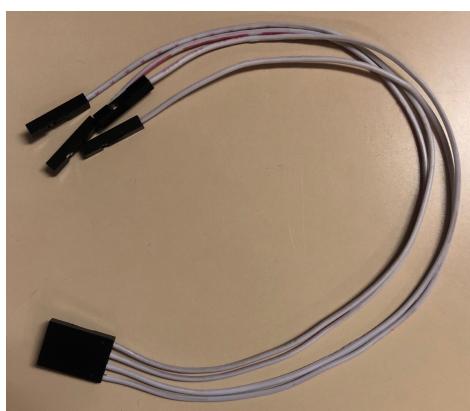
Di seguito lo schema di alimentazione e trasferimento dati tra la board Renesas e l'unità inerziale GY-86.



Dove:

- ROSSO : Alimentazione 3.3V
- BLU : massa GND
- VERDE : Clock line, SCL
- SDA : Data Line, SDA

Per il collegamento abbiamo utilizzato dei fili morsettati maschio femmina, come quelli in figura.



3 Software

3.1 *e² studio*

É un IDE (Integrated Development Environment) basato su *Eclipse* utile per creare firmware per poi eseguirli sulle schede μ -*controllers* di *RENESAS*, nel nostro caso la *YRDKRX63N*.

Scaricando determinati pacchetti relativi alla board, si puó, con lo stesso IDE:

- Compilare
- Assemblare il codice
- Sfruttare una GUI per editare codice
- Fare il debug sia sull'IDE, sia su una board

L'installazione del software *e² studio*, puó essere fatta scaricando dal sito di *Renesas*, <https://www.renesas.com/in/en/products/software-tools/tools/ide/e2studio.html>, direttamente l'eseguibile. L'IDE é disponibile solo per macchine con *Microsotf Windows* ©.

3.2 Architettura

Il codice dell'IMU é strutturato in maniera tale da poter essere utilizzato indipendentemente dal contesto su cui l'IMU viene montata. É composto da un main, con determinate librerie incluse, da integrare con il resto del progetto. Ció che l'utente deve fare é inserire tutte le istruzioni del nostro main sul proprio e includere le nostre librerie sui propri file. Nel seguito l'utente potrà prelevare i valori desiderati dalla struttura *AHRSdata* qui sotto rappresentata.

```
typedef struct{
    float RollRad;           // Corrisponde all'uscita in radienti dell'asse x di AHRS
    float PitchRad;          // Corrisponde all'uscita in radienti dell'asse y di AHRS
    float YawRad;            // Corrisponde all'uscita in radienti dell'asse z di AHRS
    float RollDeg;           // Corrisponde all'uscita in gradi dell'asse x di AHRS
    float PitchDeg;          // Corrisponde all'uscita in gradi dell'asse y di AHRS
    float YawDeg;             // Corrisponde all'uscita in gradi dell'asse z di AHRS
    float omegaRollRad;      // Corrisponde all'uscita in radienti dell'asse x di AHRS (velocità angolare)
    float omegaPitchRad;     // Corrisponde all'uscita in radienti dell'asse y di AHRS (velocità angolare)
    float omegaYawRad;       // Corrisponde all'uscita in radienti dell'asse z di AHRS (velocità angolare)
    float omegaRollDeg;      // Corrisponde all'uscita in gradi dell'asse x di AHRS (velocità angolare)
    float omegaPitchDeg;     // Corrisponde all'uscita in gradi dell'asse y di AHRS (velocità angolare)
    float omegaYawDeg;        // Corrisponde all'uscita in gradi dell'asse z di AHRS (velocità angolare)
}AHRS_data;
```

All'interno di tale struttura sono presenti tutti i valori di angoli, accelerazioni e velocitá nelle rispettive direzioni. Dichiарando dunque sul proprio main una struttura di tipo *AHRSdata* si ha l'accesso a tali campi.

Questi valori vengono ricavati da altre strutture presenti all'interno del codice. Come si nota sotto abbiamo una struttura che elabora i valori dell'IMU e una struttura che elabora quelli del magnetometro. xt

```
typedef struct {
    short raw[3];
    float x,y,z;
    float sens;
    float bias[3];
    float scale[3];
    float ABS;
} MAG_data;
```

```

typedef struct {
    float accRoll;
    float accPitch;
    float accYaw;
    float gyrRoll;
    float gyrPitch;
    float gyrYaw;
} IMU_raw;

typedef struct {
    uint16_t acc_sens;
    float gyr_sens;
} IMU_sens;

typedef struct {
    float accRoll;
    float accPitch;
    float accYaw;
    float gyrRoll;
    float gyrPitch;
    float gyrYaw;
} IMU_temp;

typedef struct{
    float Roll_Rad_Ref;
    float Pitch_Rad_Ref;
    float Yaw_Rad_Ref;
} IMU_rif;

```

3.3 Inizializzazione IMU

Il codice é stato implementato con l'idea di creare un "pacchetto" funzionante, indipendentemente dall'utilizzo finale. Per questo motivo ci siamo basati su una struttura principale **AHRSdata**, dove vi sono immagazzinate tutte le informazioni ricavabili dalla board. L'utente finale non dovrá far altro che leggere i valori dei campi in tale struttura. La vera e propria inizializzazione stá dunque nell'integrare il nostro "pacchetto", con la restante parte di codice.

Per quanto riguarda l'installazione del pacchetto software all'interno di un progetto, si deve:

1. Copiare tutti i file allegati qui, all'interno della propria cartella **src**.
2. Inserire le MACRO necessarie al dispositivo all'interno delle *proprietá* del progetto, come in figura.
3. Includere i file *.h* del pacchetto, su tutti i file del progetto. Si veda figura sottostante con le librerie necessarie.
4. Nel proprio *main* vanno dichiarate e richiamate le strutture e funzioni come da codice sorgente. Nella seconda figura di questo capitolo, vi é riportato il main con tutte le istruzioni necessarie al corretto funzionamento del dispositivo.



```

#include <stdint.h>
#include <stdio.h>
#include "platform.h"
#include "CMT.h"
#include "imu.h"
#include "mag.h"
#include "AHRS.h"
#include "switches.h"

```

```

switches_initialize();
lcd_initialize();
lcd_clear();
CMT_init();
setAHRSFrequency(1000.0/(float)PERIOD);           //Set frequenza di calcolo AHRS
switch (imu_init(&imu_sens)){                     //Test per verificare il corretto funzionamento IMU
    case (0x0):
        lcd_display(LCD_LINE1,"-IMU OK-");
        break;
    case (0x1):
        lcd_display(LCD_LINE1,"i2c ERR");
        break;
    case (0x2):
        lcd_display(LCD_LINE1,"mpu ERR 1");
        break;
    case (0x3):
        lcd_display(LCD_LINE1,"mpu ERR 2");
        break;
    case (0x4):
        lcd_display(LCD_LINE1,"mpu ERR 3");
        break;
}
switch (mag_init(&mag_data)){                   //Test per verificare il corretto funzionamento MAG
    case (0x0):
        lcd_display(LCD_LINE2,"-MAG OK-");
        break;
    default:
        lcd_display(LCD_LINE2,"mag ERR");
}
calibrationYPR(&msg, &mag_data);                //Calibrazione MAG

while(1){                                         //Ciclo infinito. Lettura valori IMU-MAG
    if(imu_read(&imu_raw, &imu_sens, &imu_temp)){
        lcd_display(LCD_LINE3,"imu_read ERR");
    }
    if(mag_read(&mag_data)){
        lcd_display(LCD_LINE4,"mag_read ERR");
    }
    getYPR(&mag_data, &imu_temp, &ahrs_data);

    errore=mag_data.x*mag_data.x+mag_data.y*mag_data.y+mag_data.z*mag_data.z;
    sprintf(errorestr, "%f", (errore-mag_data.ABS)/mag_data.ABS);
    lcd_display(LCD_LINE5,(uint8_t*)errorestr);
}

```

3.3.1 Descrizione MAIN

La prima parte l'inclusione delle librerie necessarie alla gestione del codice. Nello specifico si ha:

1. *"stdint.h"*; libreria c standard per la gestione degli input.
2. *"stdio.h"*; libreria c standard per la gestione degli I/O.
3. *"platform.h"*; libreria contenente le funzioni per la comunicazione tra IMU GY-86 e la board Renesas.
4. *"CMT.h"*; libreria per la gestione dei timers.
5. *"imu.h"*; libreria contenente le funzioni di gestione dell'IMU.
6. *"mag.h"*; libreria contenente le funzioni di gestione del magnetometro.
7. *"AHRS.h"*; libreria contenente le funzioni dell'algoritmo AHRS.

8. "switches.h"; libreria per la gestione e l'utilizzo degli switches.

Subito dopo la dichiarazione degli *include*, vanno richiamate le funzioni di inizializzazione dei vari dispositivi, tra i quali switches, lcd, CMT. Inoltre é fondamentale settare la frequenza di lavoro dell'algoritmo AHRS con l'apposita funzione. Nel caso preso in esame, l'algoritmo lavora con una frequenza di:

$$AHRS_{frequency} = \frac{1000.0}{PERIOD} \quad (16)$$

dove PERIOD é una costante che puó essere impostata nel main.c; nel nostro caso é inizializzata a 5 cosí da ottenere una frequenza pari a $200Hz$. É da notare che la frequenza a cui l'algoritmo lavora, influenza la coerenza in real-time dei risultati. Infatti una frequenza troppo bassa potrebbe ritardare l'aggiornamento dei dati e dunque produrre dati non coerenti. Al contrario non è possibile aumentare in modo illimitato tale frequenza, piú della frequenza a cui il micro - controller lavora.

Sotto si trovano due switch case. Entrambi servono alla gestione degli errori che i dispositivi possono riportare. In particolare, il primo gestisce gli errori prodotti dall'IMU, il secondo dal magnetometro. Tali errori possono essere di tipo fisico, ad esempio, un cavo scollegato o con malfunzionamento di contatto, porta un errore di tipo *mpu ERR* o di tipo *mag ERR*. In assenza di errori, a display, si vedrá le scritte *-IMU OK-* e *-MAG OK-*.

Dopo la porzione di codice che gestisce gli errori, troviamo la funzione di calibrazione la quale é discussa nel capitolo successivo.

Sotto tale funzione inizia il ciclo infinito che rappresenta l'evoluzione real-time di tutta la scheda. All'interno di questo, troviamo due condizioni le quali restituiscono il valore FALSE, se la lettura dei valori é andata a buon fine, viceversa, in caso di esito negativo la condizione é TRUE e dunque a schermo troveremo le scritte *imu.read ERR* o *mag.read ERR*.

La funzione *getYPR* valorizza i vari campi della struttura *AHRS_data* con i valori appena letti. Le ultime tre istruzioni stampano sul LCD la variabile ***errorestr*** la quale, per ottenere buoni risultati, deve essere prossima a 0. In particolare si ha:

$$errorestr = \frac{[(mag_data.x^2 + mag_data.y^2 + mag_data.z^2) - mag_data.ABS]}{mag_data.ABS} \quad (17)$$

3.4 Calibrazione IMU

Questa fase é fondamentale per ottenere delle buone misurazioni. In particolare si deve procedere con la calibrazione, ogni qual volta si installa la board, all'interno di un progetto fisico. Questo perché ogni struttura é costruita sfruttando materiali diversi e quest'ultimi, specie se ferrosi o conduttori, possono modificare i campi magnetici che il magnetometro va a leggere.

In particolare la calibrazione consiste nel ruotare in **tutte le direzioni** l'IMU stesso cosí da permettergli di rilevare **tutti** i campi elettromagnetici presenti nelle vicinanze.

Nello specifico, analizziamo i passi che si devono svolgere per ottenere una buona calibrazione, e di conseguenza risultati coerenti.

1. Collocare e fissare saldamente, nella posizione finale, la board e l'*IMU*.
2. Eseguire il debug del software sulla scheda *Renesas*, dopo aver collegato opportunamente l'*IMU*.
3. Ruotare in **tutte le direzioni** l'*IMU* durante la scritta calibrazione.
4. Attendere di leggere a display una serie di valori i quali rappresentano i bias che verrano impostati.
5. Ripremere lo *switch* – 01 per uscire dalla fase di calibrazione, i valori sopra citati non saranno piú visibili.

La fase di calibrazione deve essere effettuata ogni qual volta si sposta fisicamente l' *IMU* all'interno del proprio progetto. Per non dover ricalibrare l' *IMU* ad ogni accensione, basta prendere i valori di **bias** che sono apparsi a display nella fase 5 sopra detta ed inserirli manualmente all'interno del codice sul file *mag.c*, dove si trova la valorizzazione della struttura *imu.data*. Cosí facendo la board avrà già all'interno i valori dei campi magnetici presenti nell'ambiente circostante e non vi sarà la necessità di una nuova ricalibrazione. Assicurarsi che alla prima accensione, sia non commentata l'istruzione di calibrazione. Una volta eseguito un primo avvio e conseguente inserimento dei valori all'interno della struttura sopra citata, é possibile commentare l'istruzione di calibrazione per non doverla ripetere ogni qual volta si avvia la board.

3.5 AHRS - Spiegazione teorica

L'Attitude Heading Reference System é un algoritmo capace di mantenere una stima accurata dell'assetto mentre un corpo é in movimento. Esso combina le funzioni dei sensori installati, fornendo gli angoli di roll, pitch e yaw. Il suo funzionamento é basato sull'integrazione delle velocitá angolari misurate dai giroscopi triassiali installati. Se idealmente si dispone di informazioni esatte sull'assetto iniziale e di giroscopi di elevata accuratezza l'integrazione dell'equazioni differenziali che descrivono l'evoluzione nel tempo dell'assetto in funzione delle velocitá angolari é sufficiente a fornire gli angoli desiderati con bassi errori per lunghi periodi di funzionamento. A causa però degli errori sull'assetto iniziale, degli errori di integrazione numerica e a causa dell'imprecisione dei giroscopi questo tipo di soluzione comporta crescenti errori di assetto che limitano il tempo di funzionamento entro il quale i dati forniti dal sistema sono sufficientemente accurati. L'AHRS impiega evoluti metodi di elaborazione primo fra tutti il Filtro di Kalman. Il suo principio di funzionamento é il seguente:

- Partendo da valori iniziali ottenuti dagli accelerometri, gli angoli di assetto vengono calcolati ed aggiornati ad alta frequenza mediante l'integrazione delle velocitá angolari. Gli angoli di beccheggio e di rollio cosí calcolati per via di errori presenti vengono mandati al filtro di Kalman, mentre si ignora l'angolo di imbardata.
- Ad opportuni intervalli di tempo, nelle misure degli accelerometri si effettua una rilevazione del vettore gravitá con cui si ricavano gli angoli di beccheggio e rollio in modo indipendente dall'integrazione delle velocitá angolari. Anche questi angoli vengono a loro volta inviati al filtro di Kalman.
- Il filtro di Kalman elabora i dati provenienti dal calcolo e dalla misura dell'assetto per ottenere degli angoli di beccheggio e di rollio aggiornati ad alta frequenza ed affetti da errori limitati. Le misure vengono impiegate dal filtro per stimare e correggere gli errori sviluppati nell'integrazione.

Per il miglioramento delle prestazioni nell'implementazione del filtro di Kalman, uso comune effettuare anche la stima degli errori di Bias dei giroscopi, i quali rientrano tra le cause principali di errori. La stima dei Bias utile per far fronte a situazioni in cui tali errori siano importanti fin dall'accensione. All'accensione tramite l'allineamento o calibrazione si assegnano le condizioni iniziali nell'integrazione delle velocitá angolari per il calcolo dell'assetto e si effettua velocemente la stima iniziale degli errori di Bias. Durante l'allineamento richiesto che il veicolo non sia soggetto ad accelerazioni perché il vettore gravitá si determina dalle sole misure degli accelerometri.

3.6 AHRS - Spiegazione pratica

Interamente al sistema l'assetto del veicolo viene descritto dai quaternioni, che rendono la trattazione del problema matematico meno intuitiva, ma portano vantaggi di carattere computazionale ed eliminano il problema delle singolaritá legate all'uso degli angoli di Eulero. Vengono usate l'equazioni differenziali non lineari di assetto integrate per formulare un modello matematico che fornisca la descrizione del sistema al Filtro di Kalman. Il modello del sistema deve essere arricchito con la descrizione matematica dei Bias. Le quantitá complessivamente da stimare sono sette, le quattro componenti del quaternione e i tre errori di Bias, dove solo le prime costituiscono le uscite del sistema. Le misure fornite al filtro sono le componenti del quaternione, ovvero una parte del sistema per cui le relazioni che descrivono le misure sono semplici e in particolar modo lineari.

Nel codice implementato nel file *AHRS.c* si tiene conto che per realizzare una corretta procedura di lettura dei magnetometri, si deve considerare anche tutte le condizioni ambientali. Anche se fosse disponibile un magnetometro perfetto, questo si baserebbe sull'accesso al campo magnetico terrestre, per produrre stime di orientamento accurate. Qualsiasi condizione che influenza la grandezza o la direzione del campo magnetico terrestre, puó influenzare la precisione della rotta. Le interferenze di tale campo possono avere sia proprietá statiche sia transitorie. Nel nostro caso **consideriamo solo quelle statiche**.

Le caratteristiche statiche possono essere suddivise a loro volta in due categorie:

- **hard iron**, rappresentano fonti di campo magnetico che si aggiungono (o sottraggono) al campo magnetico terrestre. Esempi di questo tipo di errore sono i magneti permanenti, le correnti di alimentazione.
- **soft iron**, rappresentano l'entitá e il cambio di direzione che il campo magnetico terrestre sperimenta quando si trovano vicino a oggetti ferromagnetici.

NOTA: Si noti che solo le sorgenti di distorsione che producono entrambi gli errori, possono essere compensate solo se rimangono **fisse e rigide rispetto al magnetometro** mentre si muove o ruota nello spazio.

Come si nota nel segmento di codice sotto riportato, nel file *ahrs.c*, è presente il calcolo di entrambi gli errori. Il valore ABS è lo stesso che viene utilizzato nel calcolo della variabile *errorestr* illustrata nei precedenti capitoli.

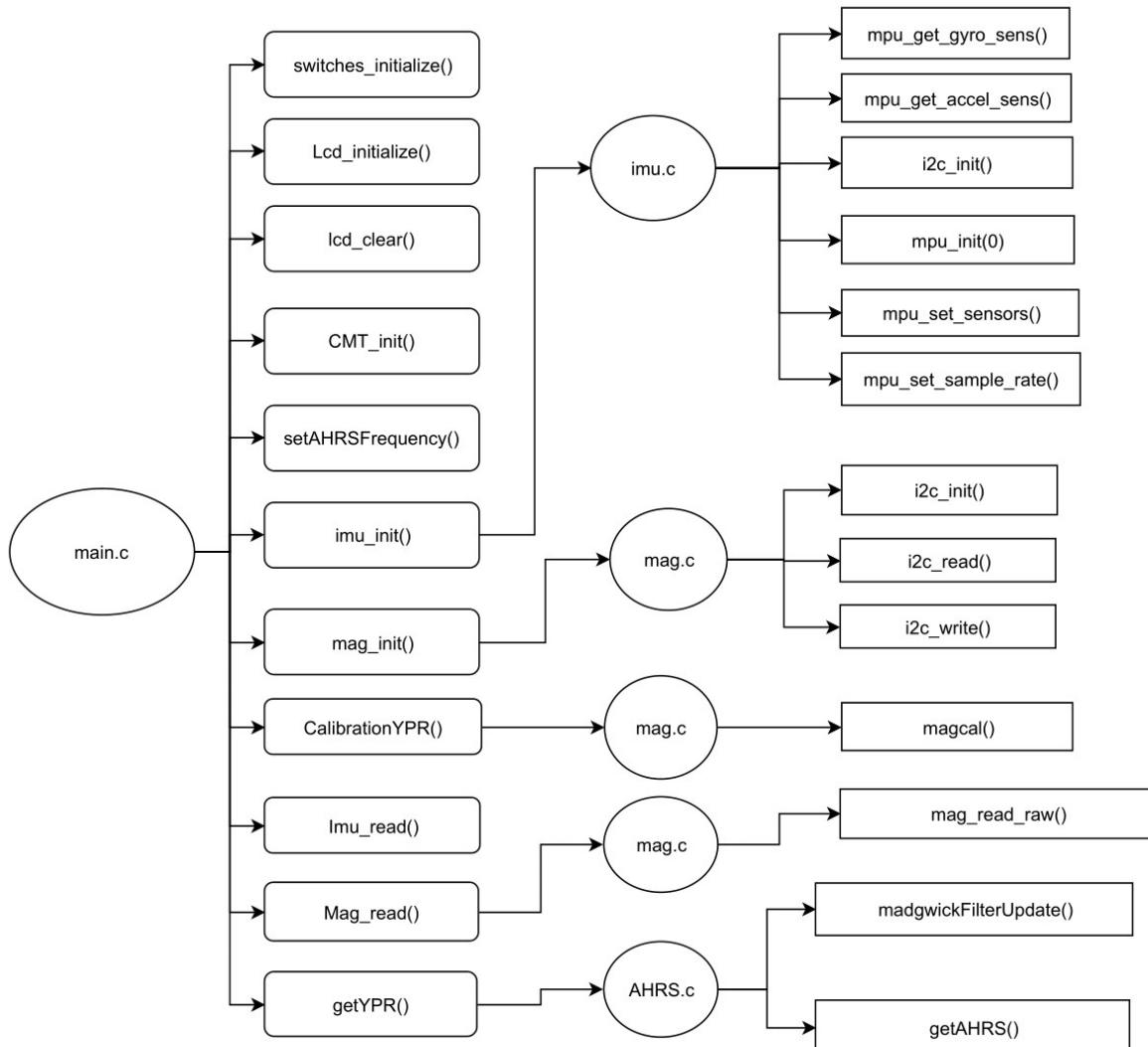
```
// Get hard iron correction
mag_data->bias[0] = (mag_max[0] + mag_min[0])/2; // get average x mag bias in counts
mag_data->bias[1] = (mag_max[1] + mag_min[1])/2; // get average y mag bias in counts
mag_data->bias[2] = (mag_max[2] + mag_min[2])/2; // get average z mag bias in counts

// Get soft iron correction estimate
mag_rad[0] = (mag_max[0] - mag_min[0])/2; // get average x axis max chord length in counts
mag_rad[1] = (mag_max[1] - mag_min[1])/2; // get average y axis max chord length in counts
mag_rad[2] = (mag_max[2] - mag_min[2])/2; // get average z axis max chord length in counts

float avg_rad = mag_rad[0] + mag_rad[1] + mag_rad[2];
avg_rad /= 3.0;

mag_data->scale[0] = avg_rad/((float)mag_rad[0]);
mag_data->scale[1] = avg_rad/((float)mag_rad[1]);
mag_data->scale[2] = avg_rad/((float)mag_rad[2]);
mag_data->ABS = avg_rad*avg_rad;
```

Di seguito un diagramma di flusso che illustra il funzionamento del codice.

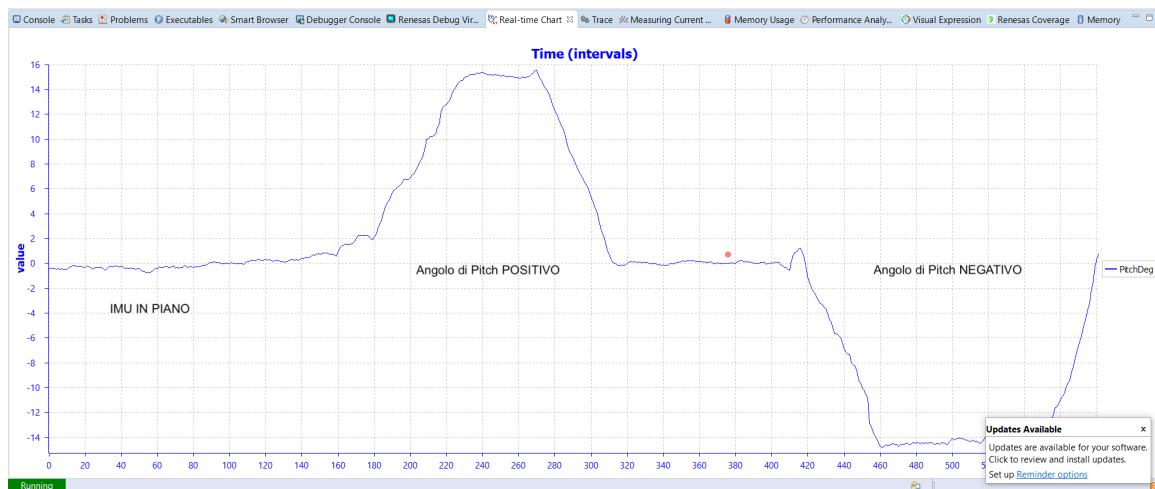
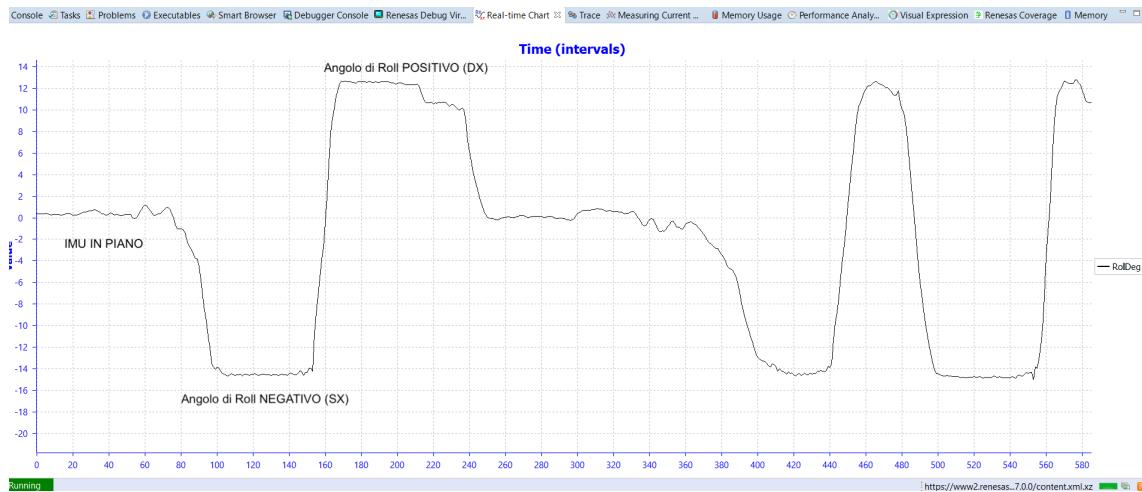


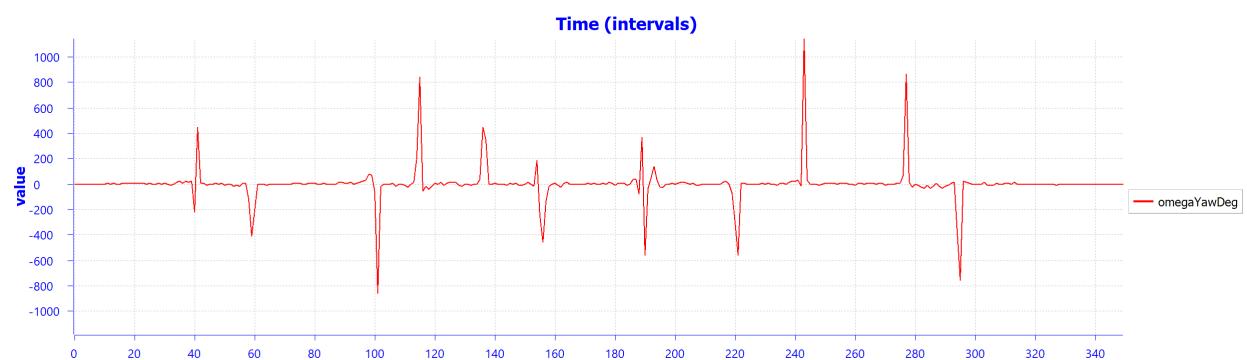
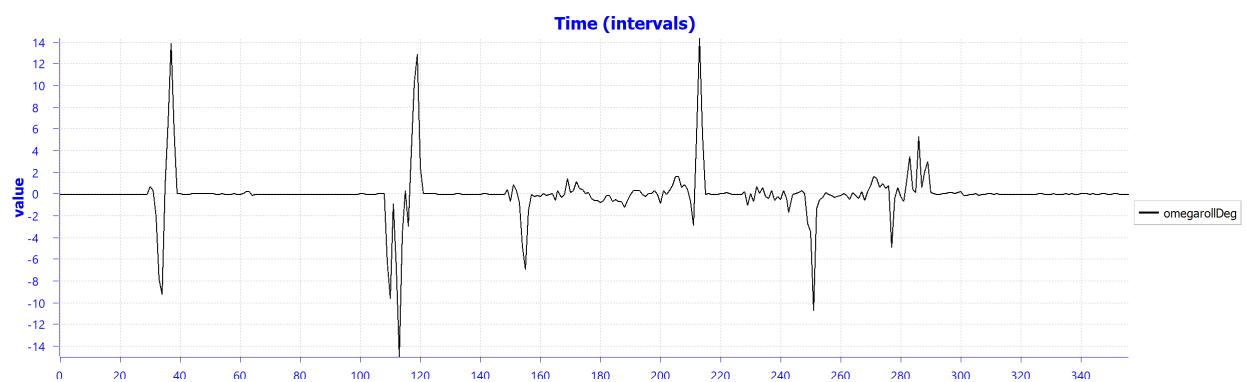
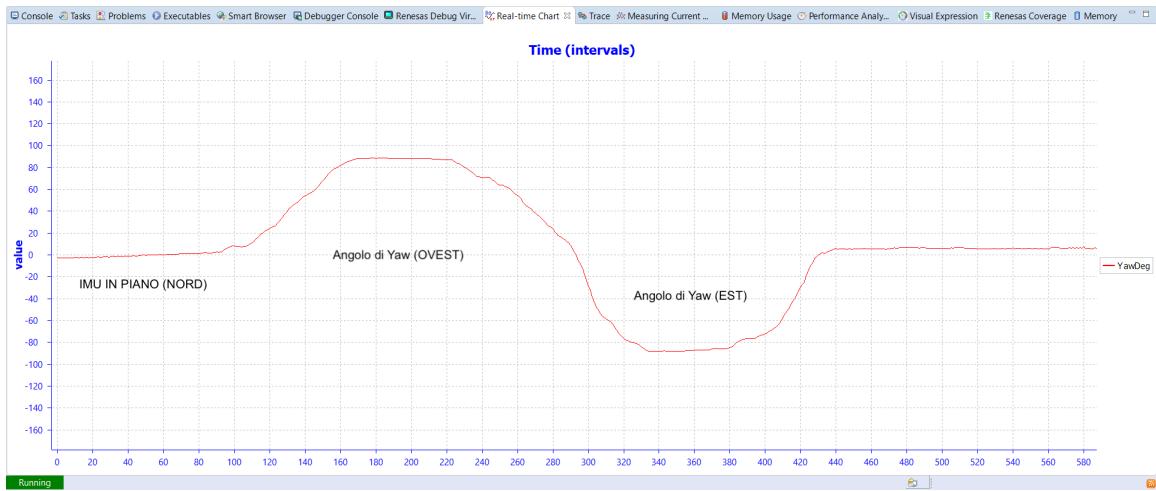
4 Test IMU su e-bike

In questo capitolo, abbiamo testato il codice e relativo funzionamento, installando sulla e-bike l'unità sensoriale. Per svolgere tali test, abbiamo lanciato il debug dall' IDE di Renesas e sfruttando la sua funzione di Real-Time abbiamo stampato a video i valori relativi agli angoli di roll, pitch, yaw e le rispettive velocità angolari. Qui sotto i grafici ottenuti.

Le prime tre figure rappresentano i test relativi agli angoli. Come si può notare, si ha una risposta veloce ad una variazione di direzione dell'IMU.

Le ultime tre riguardano le velocità angolari sui tre assi. Anche quiabbiamo una risposta veloce.





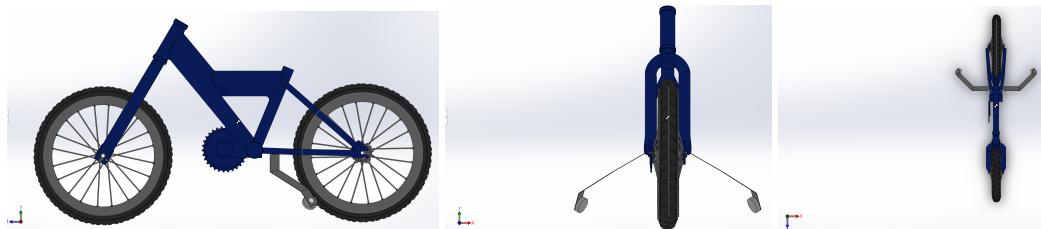
5 Modello Solid-Works e-bike

Sfruttando tale software per disegni CAD 3D, abbiamo ricreato il modello della e-bike presente in laboratorio. Per fare tale modello abbiamo misurato e pesato ogni componente della bici cos da ottenere un prototipo virtuale coerente sia in peso che in dimensioni con la realtà. Lo scopo di tale modello è stato quello di calcolare, con delle funzioni del software Solid-Works, i centri di massa dell'assieme complessivo e di specifiche parti. In allegato è possibile trovare una cartella contenente tutti i file delle singole parti e l'assieme complessivo. Di seguito sono riportate alcune immagini della bici e il corrispettivo modello 3D con i relativi centri di massa.



Al seguito, il modello realizzato su Solid Works. Abbiamo poi diviso il modello complessivo in tre assiemi.

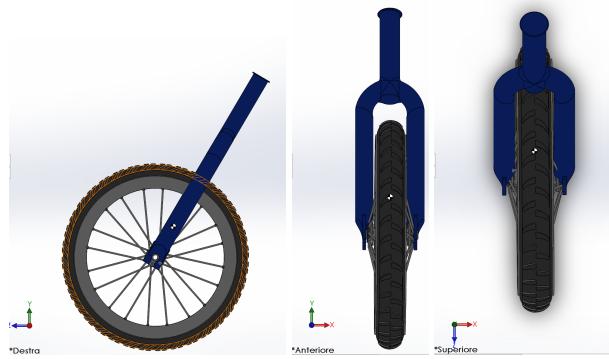
Primo assieme della bici complessiva.



Secondo assieme, rappresenta la bici e il relativo CdM tenendo conto della sola parte posteriore.



Terzo assieme, rappresenta la bici e il relativo CdM tenendo conto della sola parte anteriore, cioè mozzo e ruota.



Di seguito sono riportate le tabelle con i pesi delle singole componenti della bici e i valori dei momenti di inerzia dei vari assiemi. Questi sono presi rispetto al sistema di riferimento calcolato sul centro di massa.

Parte	Peso [Kg]
Telaio	3.582
Asta rotelle	0.555
Rotella singola	0.02
Corona anteriore	0.033
Mozzo centrale + piastra	0.05
Mozzo anteriore e supporto ruota anteriore	0.79
Ruota anteriore	0.991
Ruota posteriore	1.182
TOTALE	7.224

Momenti d'inerzia bici completa [Kg/m^2]

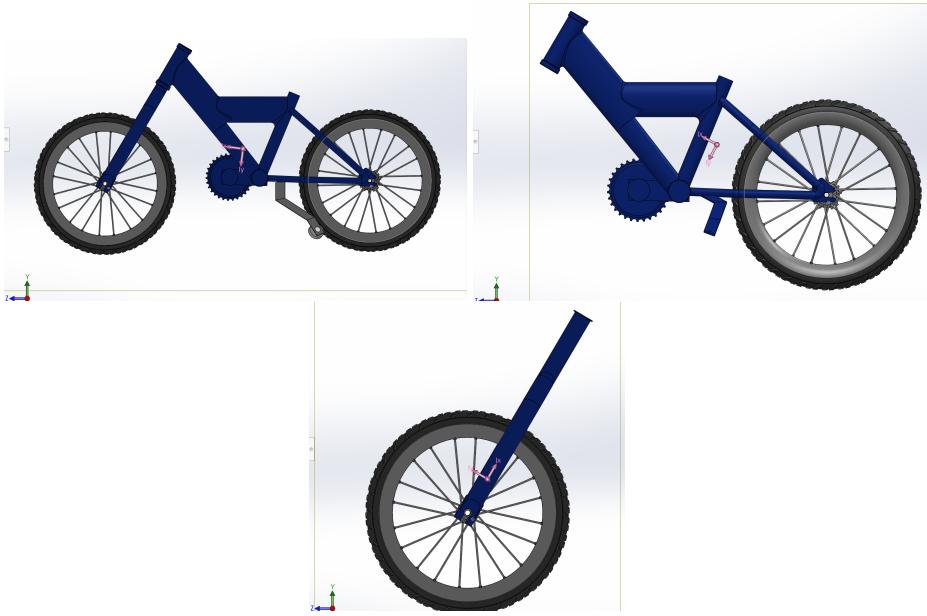
$I_{xx} = 0.000$	$I_{xy} = 0.123$	$I_{xz} = 0.992$
$I_{yx} = 0.000$	$I_{yy} = -0.992$	$I_{yz} = 0.123$
$I_{zx} = 1.000$	$I_{zy} = 0.000$	$I_{zz} = 0.000$

Momenti d'inerzia telaio posteriore [Kg/m^2]

$I_{xx} = -0.003$	$I_{xy} = 0.474$	$I_{xz} = 0.881$
$I_{yx} = -0.012$	$I_{yy} = -0.881$	$I_{yz} = 0.474$
$I_{zx} = 1.000$	$I_{zy} = -0.009$	$I_{zz} = 0.008$

Momenti d'inerzia telaio anteriore [Kg/m^2]

$I_{xx} = -0.002$	$I_{xy} = 0.862$	$I_{xz} = -0.508$
$I_{yx} = 0.010$	$I_{yy} = 0.508$	$I_{yz} = 0.862$
$I_{zx} = 1.000$	$I_{zy} = -0.003$	$I_{zz} = -0.009$



6 Conclusioni

Il lavoro da noi svolto è stato senza dubbio interessante ed innovativo. Partendo da un primo codice pre-esistente, il quale sfruttava un filtro di Kalman per il filtraggio del rumore, siamo riusciti ad analizzarlo e a capirne il funzionamento. Un secondo codice, dotato di filtro AHRS per combinare i valori raw (grezzi) ottenuti direttamente dalla GY-86, è stato analizzato e organizzato in modo tale da renderlo indipendente dal progetto di applicazione. Questo codice, se pur sfruttando metodi numerici molto più complessi, riesce a restituire all'utente dei valori coerenti con la realtà. Il nostro lavoro si è concentrato non tanto nello studio dell'algoritmo di filtraggio, ma nella costruzione di un'infrastruttura che potesse essere sfruttata a pieno in un sistema embedded come il nostro.

Altra parte del nostro lavoro è stata l'applicazione del codice e del rispettivo IMU, alla e-bike presente in laboratorio. Questo passaggio ha richiesto la determinazione del centro di massa della bici, il quale è stato calcolato realizzando il modello su un CAD 3D della bici stessa. Su tale modello abbiamo calcolato anche altri centri di massa relativi a delle parti specifiche della bici. Lo scopo di questi centri di massa "parziali" è la determinazione di altre eventuali posizioni sui cui allocare più IMU, le quali lavorando in sincronia restituirebbero dei valori sicuramente più efficaci e precisi rispetto ad un solo IMU, come nel nostro caso.

Uno sviluppo futuro della bici, potrebbe essere quello di dotarla di due motori, uno anteriore ed uno posteriore i quali opportunamente pilotati garantirebbero al veicolo una stabilità maggiore. Inoltre si potrebbe pensare di applicare due schede IMU, nei centri di massa "parziali" da noi calcolati e realizzare un software che mette insieme i dati delle singole IMU.

Riferimenti bibliografici

- [1] M. Camerlengo, F. di Girolamo, M. Simoni, F. Tesei. Relazione "Self Balancing Bicycle" - Professore: Andrea Bonci, A.A 2015/2016
- [2] Relazione Ballbot
- [3] Renesas Electronics Corporation. *RX63N Group, RX631 Group User's Manual: Hardware*. Rev.1.80 Apr 25, 2014
- [4] Renesas Electronics Corporation, Future Designs Inc. *YRDKRX63N Schematics*. January 24, 2012
- [5] Sebastian O.H. Madgwick. *An efficient orientation filter for inertial and inertial/magnetic sensor arrays*. April 30, 2010

- [6] *Hard and Soft Iron Correction for Magnetometer Measurements*,
<https://ez.analog.com/mems/w/documents/4493/faq-hard-soft-iron-correction-for-magnetometer-measurements/>