
**UNIVERSITA' POLITECNICA DELLE MARCHE**

FACOLTA' DI INGEGNERIA

---

Corso di Laurea triennale in Ingegneria Informatica e dell'Automazione

# Programmazione di microcontroller per la gestione e il controllo del robot Ballbot

# Programming of microcontrollers for navigation and control of the Ballbot robot

Relatore:

Prof. Andrea Bonci

Tesi di Laurea di:

Christian Ascani

A.A. 2020 / 2021

# *Indice*

# *Introduzione*

Un Ballbot è un robot situato al di sopra di una sfera, come una palla, in grado di bilanciarsi attraverso il suo unico punto di contatto con il terreno ed evitare quindi la naturale caduta. Esso è omnidirezionale e non ha un raggio di sterzata minimo, a differenza di altri robot a due ruote, rendendolo eccezionalmente agile e manovrabile. Il suo funzionamento si basa sul principio di un pendolo invertito, per cui è tenuto in posizione verticale attorno al suo punto di equilibrio instabile dal controllo. Il sistema è costituito da tre parti: il corpo, la propulsione e la palla. I motori forniscono l'azionamento alla sfera, la quale dà propulsione al sistema permettendone il movimento. Il corpo non è legato in alcun modo ad essa, è libero di muoversi e il contatto tra le due parti avviene attraverso tre ruote, che ruotando ne garantiscono il movimento. Il sottosistema propulsivo, essendo composto dalla sfera mossa dalle tre omniwheels azionate da motori, rappresenta la parte più complessa.

L'aspetto controintuitivo del Ballbot è che per muoversi in avanti, il corpo deve andare in avanti, e per muovere il corpo in avanti la palla deve ruotare all'indietro. Inoltre, il Ballbot deve inclinarsi per compensare le forze centripete, che si traducono in movimenti eleganti e aggraziati.

Attraverso il diagramma a blocchi (Figura 1), è possibile identificare il funzionamento generale del Ballbot. Si individuano i tre ingressi rappresentanti gli angoli (espressi in radianti):

- $\boldsymbol{\phi}$ angolo di Roll (rollio), ovvero la rotazione rispetto all'asse x
- $\boldsymbol{\theta}$ angolo di Pitch (beccheggio), ovvero la rotazione rispetto all'asse y
- $\boldsymbol{\psi}$ angolo di Yaw (imbardata), ovvero la rotazione rispetto all'asse z

Essi saranno sommati alla retroazione del sistema in modo da ottenere i valori di errore per ogni ingresso, così da essere processati dai rispettivi PID. Ciascun PID restituirà in uscita il valore di tensione necessario a compensare lo squilibrio introdotto dal Ballbot; grazie ad una matrice adibita all'elaborazione delle tensioni si ottiene la corrente di

armatura da fornire ai rispettivi PID, la quale è considerata come il riferimento della coppia, essendo proporzionale ad essa. In seguito, la corrente necessaria a ripristinare la stabilità del Ballbot verrà fornita ai rispettivi motori.

Invece, per quanto riguarda la retroazione, si possono considerare due parti, ovvero l'IMU (Unità di Misura Inerziale) e il filtro di Kalman; in ingresso dell'IMU ad uno spostamento effettuato con una determinata accelerazione e una specifica posizione viene associato un numero, che ne rappresenta l'angolo, il quale verrà passato al filtro di Kalman per eliminare eventuali rumori e restituire un segnale pulito.

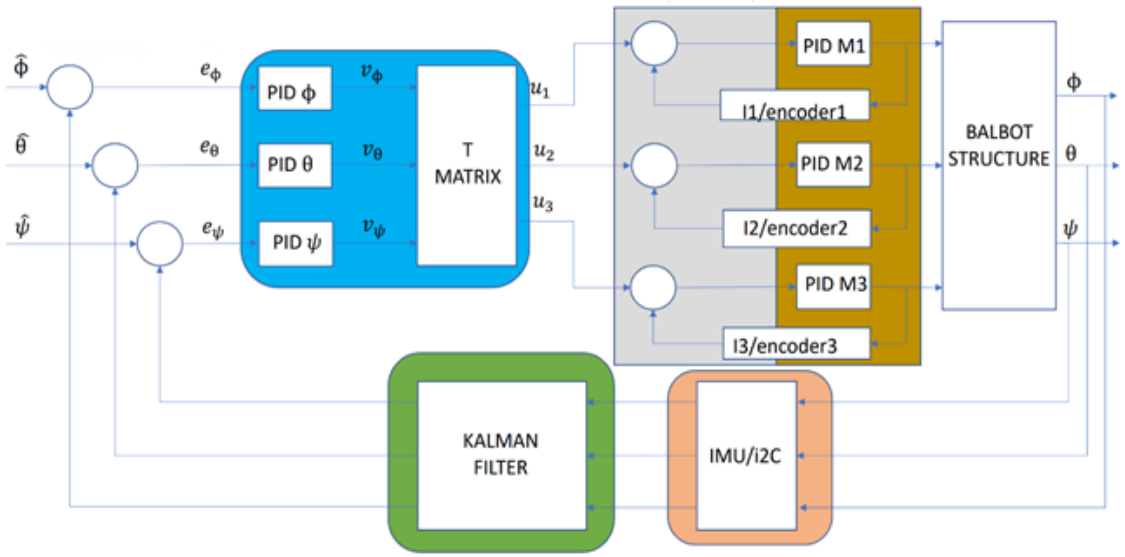



***Figura 1*** *– Diagramma a blocchi*

Focalizzando l'attenzione sul controllo in coppia (Figura 2), nel sistema a catena chiusa la regolazione è automatica: l'uscita viene confrontata con l'ingresso di riferimento, in modo da produrre, ogni qual volta si verifichi una differenza tra il segnale di uscita e il segnale di riferimento, un'azione correttiva che riporti l'uscita al valore desiderato.

Si vuole fornire al motore il valore della corrente necessaria a mantenere in equilibrio il Ballbot, ricavato dalla combinazione tra il valore letto dal sensore di corrente ($I_{att}$) e il termine di riferimento ($I_{des}$), ovvero la corrente ricavata in base all'angolo di riferimento. Perciò, ad ogni percorrenza del loop di controllo è necessario conoscere il valore di coppia attuale in modo da essere confrontato con quello desiderato così che il PID generi un segnale di controllo adeguato.

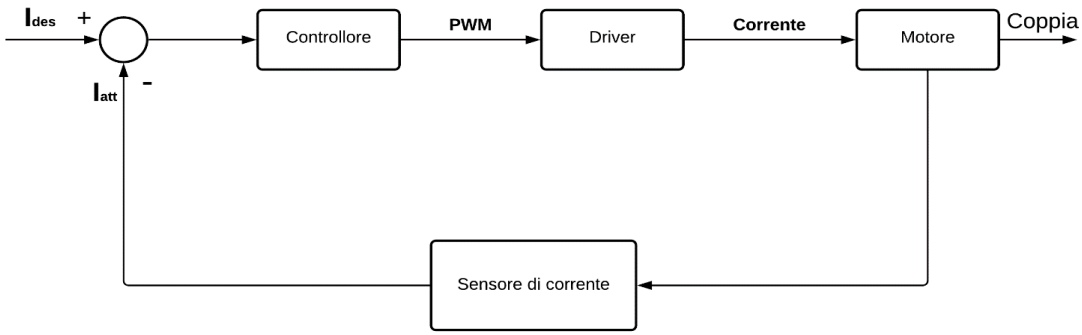



***Figura 2*** *– Schema del controllo in coppia*

Poiché non è possibile disporre di strumenti atti alla misura della coppia motrice sviluppata dai singoli motori, si può ricavare tale valore a partire da altre grandezze. Infatti, la coppia in uscita, acquisita ad ogni ciclo di clock, è proporzionale alle correnti che attraversano il circuito di armatura e il circuito di eccitazione, secondo la legge meccanica:

$$C_m(t) = K \cdot I_e(t) \cdot I_a(t)$$

Nel Ballbot la corrente di eccitazione e il coefficiente di coppia sono costanti fisse (fornite dal costruttore), quindi l'unica grandezza modificabile, nonché unica da misurare, è la corrente di armatura. Perciò l'equazione precedente può essere riscritta come:

$$C_m(t) = K_m \cdot I_a(t)$$

Dalla formula si evince che il motore, essendo a collettore, può essere controllato agendo sulla corrente del circuito di armatura ($I_a$); in generale, questa modalità di controllo è utilizzata prevalentemente per azionamenti di piccola potenza.

# CAPITOLO I – HARDWARE DEL BALLBOT

Il “cervello” del Ballbot è dato dal µ-controller RX63N ad alte prestazioni, montato sulla Renesas Demonstration Kit YRDKRX63N a 100 pin, che include: 32-bit MCU capace di operare oltre i 100 MHz, FPU (Floating Point Unit) per i calcoli, più di 21 canali per ADC a 12-bit e oltre 2 canali per DAC, Unità Timers MTU2 con funzioni di input capture / output compare / counter clearing per generazione di segnali PWM e controllo motori, watchdog timer indipendente e funzione CRC, svariate funzioni di comunicazione (I2C, Ethernet, CAN, ecc).

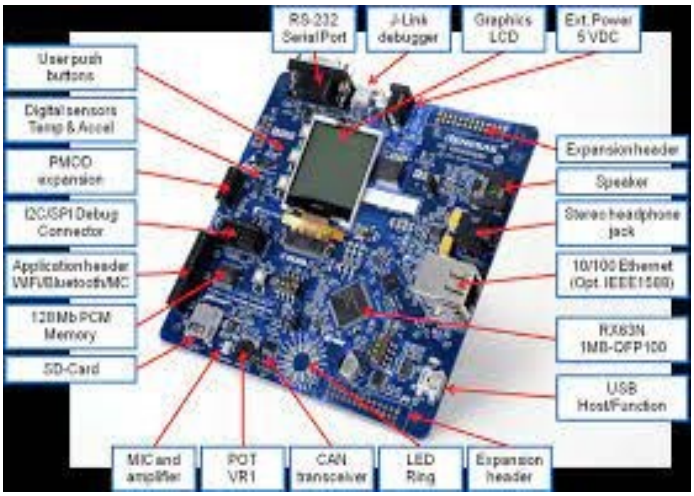



***Figura 3*** *– Renesas Demonstration Kit*

Dal punto di vista funzionale, il sistema può essere suddiviso in tre macro-blocchi, che verranno analizzati in dettaglio nei capitoli successivi: il primo è dato dagli attuatori, il secondo è rappresentato dai sensori e il terzo è espresso dal controllo.

## 1.1 Motore

Il motore utilizzato nel Ballbot è un DC fabbricato dalla Pololu con spazzole da 12V con un cambio in metallo 131.25:1 ed un encoder in quadratura integrato che fornisce una risoluzione di 64 conteggi per giro dell'albero motore (Figura 4). Dalle specifiche senza carico esso, alimentato con 12 V e 200 mA, compie 76 giri per minuto, mentre allo stallo sviluppa una coppia di 45 kg·cm con una corrente pari a 5.5 A; naturalmente il funzionamento allo stallo può portare il motore ad un deterioramento strutturale e/o a

rottura. Ad ogni albero motore si fissa una ruota omnidirezionale; questa tipologia di ruote è formata da tanti piccoli dischi, disposti lungo la circonferenza principale e orientati perpendicolarmente alla direzione di rotazione. L'effetto che si ottiene è che la ruota, oltre che sviluppare una forza lungo la direzione di movimento, può anche scorrere lateralmente se viene applicata qualche forza longitudinale.

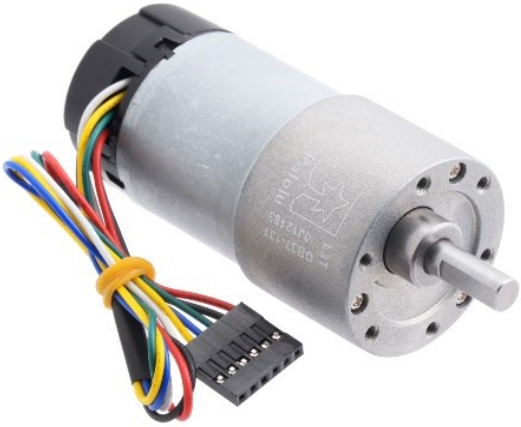


***Figura 4*** *– Motore montato sul Ballbot*

Riguardo al pilotaggio dei motori, il semplice controllo ON/OFF non risultava essere idoneo poiché esso prevedeva solamente di comandare il motore alla massima velocità oppure di fermarlo completamente. Perciò è stato utilizzato l'azionamento Cytron MD10C R3 il quale presenta al suo interno un ponte a H che consente di far girare il motore in entrambi i versi di rotazione, invertendo il segno della corrente che passa all'interno del motore stesso.

## 1.2 Convertitore

Nell'assetto del robot sono presenti tre convertitori DC-DC Power Module da 25 W (Figura 5); tali strumenti elettronici spesso contengono diversi circuiti, in cui ognuno necessita di un livello di tensione differente da quella fornita dalla batteria. I convertitori DC-DC offrono un metodo per generare diversi livelli di tensione controllati a partire da una batteria a tensione variabile, risparmiando in tal modo spazio ed evitando di utilizzare molte batterie per fornire energia alle diverse parti dello strumento. Questi convertitori sono in grado di convertire qualsiasi tensione continua tra 3,6-25V ad una tensione selezionabile da 3,3-25V. In realtà, il primo dispositivo è utilizzato per

alimentare la scheda Renesas a 5 V, mentre il secondo è deputato all'alimentazione degli encoders con tensione d'uscita di 3.3 V; il terzo è utilizzato esclusivamente per l'alimentazione dei motori. Grazie all'interruttore a slitta è possibile scegliere la tensione di uscita da 5 V, adottata proprio per l'alimentazione del µ-controller; poi, è possibile scegliere fra tre diverse interfacce di uscita, due morsetti con tensione pari a quella di ingresso ed un pulsante per accensione-spegnimento del convertitore.

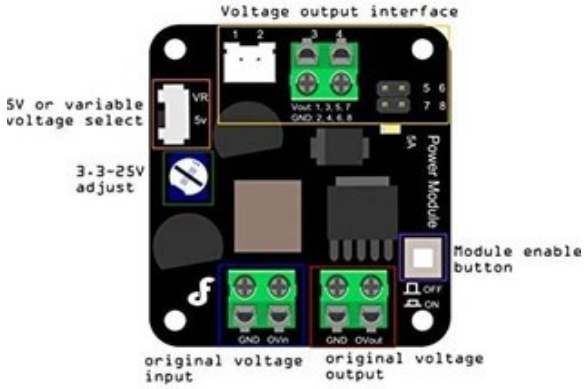



*Figura 5 – Convertitore DC-DC*

| Specifiche tecniche |
| --- |
| Tensione di ingresso: 3.6 – 25V |
| Tensione di uscita: 3.3 – 25 V |
| Corrente di uscita: 5 A (a 5 V) |
| Potenza massima: 25 W |
| Frequenza di commutazione: 350 kHz |

## 1.3 Sensoristica

La stima dell'inclinazione del robot è uno dei principali problemi da risolvere per riuscire a bilanciarlo al meglio. Un piccolo errore nell'identificazione di questo parametro potrebbe portare il robot a oscillare e, successivamente, a cadere. Perciò di fondamentale importanza è l'IMU (Inertial Measurement Unit) che consiste in un accelerometro a 3 assi e un giroscopio a 3 assi al suo interno, finalizzati a misurare l'accelerazione, la velocità, l'orientamento, lo spostamento e molti altri parametri relativi al movimento di un sistema. In particolare, il sensore scelto è un GY-80, il quale ha molte funzioni sul singolo chip. In breve, è costituito da un accelerometro, un giroscopio e un sensore di temperatura; tale modulo è molto preciso durante la conversione di valori analogici in digitale perché ha un hardware convertitore analogico-digitale a 16 bit per ogni canale.

Invece, per il rilevamento della corrente viene utilizzato un sensore INA219, prodotto dalla Texas Instruments, il quale è un circuito integrato in grado di misurare l'assorbimento di corrente con 1% di precisione attraverso la resistenza di shunt,

trasmettendo i dati tramite protocollo $I^2C$. Tale trasduttore fornisce letture digitali di corrente, tensione e potenza necessarie per un processo decisionale accurato in sistemi controllati con precisione, come il Ballbot. Inoltre, su di esso si trovano dei registri programmabili, i quali consentono una configurazione flessibile per la risoluzione delle misurazioni e un funzionamento continuo.

## 1.4 Struttura ed organizzazione

Ogni elemento che costituisce il robot è posizionato in una particolare posizione in base ad alcuni criteri, dettati da caratteristiche funzionali e strutturali. Per primo, la scheda Renesas è fissata su di un supporto piano che si trova a sua volta sopra una struttura esagonale, solidale con il piano inferiore, il quale ospita sulla parte terminale anche gli alloggi per i tre motori. Su tre dei sei lati della struttura centrale esagonale sono fissate le schede di pilotaggio, mentre sugli altri tre lati sono fissati i convertitori DC-DC; in uno spigolo è stato messo un interruttore. Poi, i sensori di corrente sono posizionati sotto i corrispondenti driver, sempre fissati al telaio. Al suo interno, si trova l'alloggio per la batteria, che dunque può essere rimossa semplicemente alzando la base superiore. Invece, sulla base inferiore sono presenti quattro schede millefori: una è utilizzata come riferimento a massa (ground), una rappresenta una tensione di 3.3 V, due sono utilizzate per la comunicazione $I^2C$ (SCL ed SDA); inoltre, in questa sezione è presente l'IMU. Per quanto riguarda il cablaggio, si possono distinguere due fasci di cavi: il primo è diretto verso i connettori JN2 e J8 della Renesas (usati per PWM e fasi encoder), mentre il secondo è diretto verso il connettore JN1 (usato per $I^2C$). Invece, i cavi dell'alimentazione sono diretti ai convertitori e – poiché i sensori di corrente sono disposti in serie fra motore e driver – il positivo del motore è collegato a $V_{in}$- del sensore, mentre $V_{in}$+ è collegato con A del driver. Il negativo del motore viene collegato con B del driver.

# CAPITOLO II – ATTUATORI DEL SISTEMA

## 2.1 Identificazione

L'identificazione del motore è stata eseguita in due passaggi: per primo, con il "mydaq" di National Instruments grazie al software Labview è stato realizzato un campionamento su 20 ms, poi con l'utilizzo di Matlab si è proceduto alla vera e propria identificazione. In dettaglio, attraverso il "System Identification Toolbox" messo a disposizione da Matlab, dopo aver inserito i campioni di input e di output, si è in grado di ricavare il modello matematico del processo (Figura 6).

```
Process model with transfer function:
                Kp
     G(s) = ----------
              1+Tp1*s

          Kp = 0.69307
         Tp1 = 0.1016
```

***Figura 6*** *– Modello matematico del processo*

```
% A questo punto si ricava la resistenza di armatura allo stallo, Ra=V/I
% Dal datasheet allo stallo V è pari a 12 V mentre la corrente è 5.5 A
% Ras = 12/5.5 = 2.18 Ohm;
Va = 12;                % [V]
% Si misura anche la resistenza, tenendo ferma una ruota solidale
% all'albero, e si ricava una resistenza pari a 2.4
% Facendo una media si ottiene
Ra = 12/4.8;            %2.3 con un errore di 0.1 Ohm

% Sappiamo che Cm = Ke*Ia (Ia = corrente di armatura, Ke = costante di
% eccitazione, dato che la corrente di eccitazione è costante)
% Misuriamo inoltre allo stallo la costante del motore Kt, leggendo dal
% datasheet i valori della corrente allo stallo (4.8 A) e del
% Momento meccanico (450 Kg*mm = 4.41 N*m)
T = 4.41;               % [N*m]
Is = 4.8;               % [A]

% La costante allo stallo è:
Kt0 = T/Is;
% Ora per ricavare la costante di eccitazione Ke (quella che sarà
% nell'anello di retroazione che serve per ricavare la forza
% controelettromotrice) si deve sperimentalmente ricavare velocità, forza
% controelettromotrice e costanti (per poi fare una media).
% Dalla proporzione TensMax : velAngMax = TensX = VvelAngX , sapendo che la
% velocità angolare che raggiunge il motore (no-load) è circa 76 RPM (cioè
% 8 rad/s, si può ricavare le seguenti velocità
vAngMax = 8.1;       % [rad/s] misurata

vAng1 = 3/12*vAngMax;   % 3V
vAng2 = 5/12*vAngMax;   % 5V
vAng3 = 6/12*vAngMax;   % 6V
vAng4 = 8/12*vAngMax;   % 8V
vAng5 = 10/12*vAngMax;  % 10V
% Sperimentalmente vengono uniti gli alberi di due motori grazie a due
% ruote, alimentandone solo uno e misurando la tensione ai capi dell'altro
% (si usa come dinamo) e dunque si ricava invertendo la formula Ke*vAng=e
e1 = 1.5; e2 = 3; e3 = 3.8; e4 = 5.2; e5 = 6.66;
Ke1 = e1/vAng1;
Ke2 = e2/vAng2;
Ke3 = e3/vAng3;
Ke4 = e4/vAng4;
Ke5 = e5/vAng5;     % Calcolo la media pesata con le fcem

% Ke = (Ke1+Ke2+Ke3+Ke4+Ke5)/5;
% Trovata calcolando sperimentalmente (probabile errore del calcolo di Ke1,
% Ke2, ecc.
Ke = 1.4;
% Kt calcolato sperimentalmente (non allo stallo)

Kt = 0.64;

% Sappiamo che l'attrito B è
att_B = Kt*(1-Kp*Ke)/(Kp*Ra);

% Dalla formula si ha che l'attrito J è
att_J = (Tp/Ra)*(Kt*Ke + Ra*att_B);
```

Ricavata la funzione di trasferimento, si utilizza un motore come dinamo e – grazie anche ai dati ricavati dal datasheet ufficiale – si calcolano i parametri caratteristici, come la costante del motore $K_t$ e gli attriti. Avendo ora a disposizione tali informazioni, con Simulink è possibile costruire il modello del sistema (Figura 7) in cui – però – è stato trascurato il polo elettrico, poiché è molto più piccolo del polo meccanico (non influisce nel funzionamento a regime).

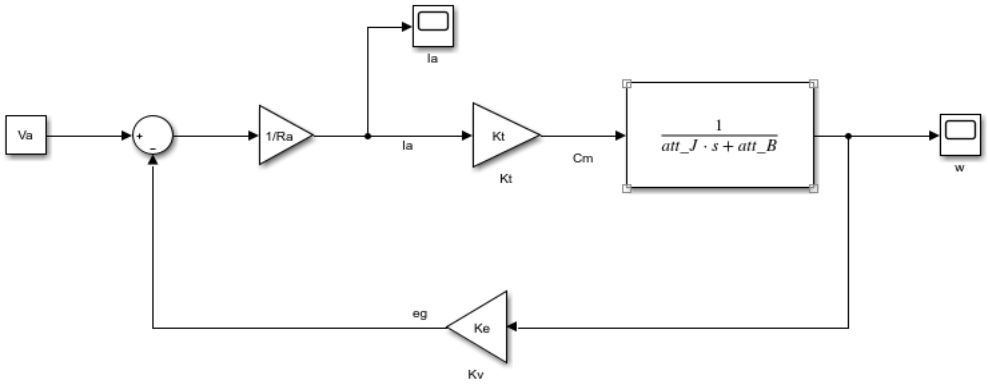



*Figura 7 – Modello del sistema*

Entrando nel dettaglio, la costante di eccitazione è presente nell'anello di retroazione poiché dalla formula $C_m(t) = K_e \cdot \omega$ si ottiene la velocità angolare, dove $K_e = K \cdot I_e(t)$ la quantità risulta essere costante. Nel funzionamento a regime, questa retroazione intrinseca del motore a collettore permette di avere una certa stabilità in quanto – nel caso in cui variasse il carico – tramite la coppia motrice in uscita dalla catena diretta si otterrebbe nuovamente l'equilibrio. Ponendo la coppia resistente a zero (nel modello essa è stata omessa), si ottengono dei risultati coerenti, in quanto la velocità angolare a regime – come indicato nel datasheet del motore – è pari a 8.3 rad/s e la corrente a regime tende ad un offset di circa 0.15 A (Figura 8). Dopo aver completato l'identificazione del motore, si esamina in dettaglio il suo funzionamento.

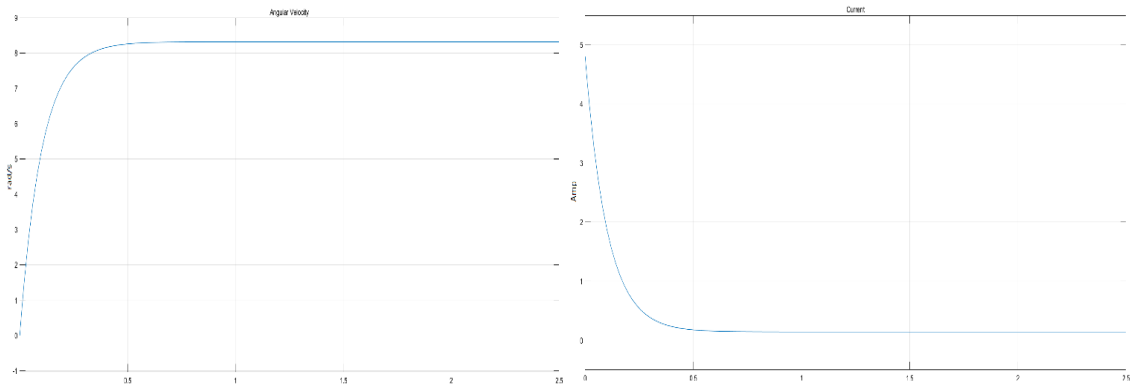


*Figura 8 – Grafici della velocità angolare e della corrente*

## 2.2 Pilotaggio dei motori

Oltre a rendere possibile il moto bidirezionale dei motori, la scheda Cytron MD10C R3 (Figura 9) consente di controllare il motore in velocità attraverso un segnale PWM (Pulse Width Modulation, cioè Modulazione a Larghezza di Impulso) in modalità SN

(Sign Magnitude). Essa prevede l'utilizzo di due segnali: il duty cycle (dal PWM) e un segnale costante che determina il verso di rotazione (DIR); essa permette di controllare la potenza assorbita dal motore, modulando il duty cycle. Il segnale è caratterizzato da una frequenza fissa e da un duty cycle variabile; questo è il rapporto tra il tempo in cui l'onda assume valore alto e il periodo T (inverso della frequenza), con possibili valori tra 0 e 100. In assenza di segnale, ossia con un duty cycle pari a 0, il motore risulta fermo, mentre con duty cycle a 100 il motore ruota con velocità massima, nel verso determinato in base al piedino DIR. Inoltre, l'altro pin presente, ovvero quello di GND, deve essere sempre collegato a massa per disporre di un riferimento logico, altrimenti non sarebbe possibile far ruotare il motore in entrambe le direzioni.

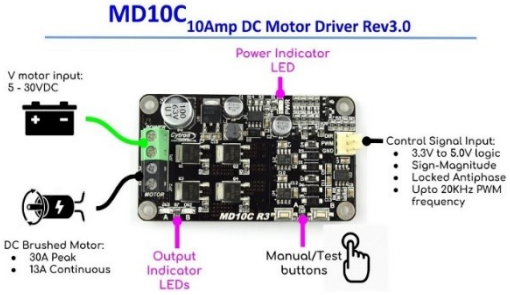



***Figura 9*** *– Driver del motore*

| **Specifiche tecniche** |
|---|
| Tensione di ingresso: 5 – 30V |
| Corrente continua massima del motore: 13 A |
| Corrente di picco del motore: 30 A |
| Tensione di bordo: 11 – 14 V |
| Tensione degli ingressi logici, alto: 3 – 5 V |
| Tensione degli ingressi logici, basso: 0.5 V |
| Massima frequenza PWM: 20 kHz |

## 2.3 Analisi del codice

Analizzando – invece – il codice atto al controllo dei motori, contenuto nel file "pwm.c", dopo l'inizializzazione è presente la procedura **Volt_to_duty** la quale converte la tensione in ingresso in un valore numerico che verrà in seguito inserito nel registro corrispondente al motore considerato. Infatti, un altro argomento è proprio il canale che serve a impostare i nuovi valori di duty cycle nei registri a disposizione (MTU3.TGRB, MTU3.TGRD, MTU4.TGRB, MTU4.TGRD); la variabile "tgr_reg" è d'appoggio e serve ad effettuare calcoli di tipo numerico.

```
/*
*********************************************************************************************************
*                                   PWM SIGNAL GENERATION
*
* Description   : This convert the voltage value to a register value in order to produce the pwm.
*
* Argument      : the desired voltage and the selected channel.
*
* Returns       : None
*********************************************************************************************************
*/
void Volt_to_duty (float control_output, unsigned char channel)
{
 unsigned short int tgr_reg;    /* Valore per il registro TGR (B o D) */

    if(channel == 1 || channel == 2)
        tgr_reg = (unsigned short)(((float)TGRA_3_VAL * control_output) / (float)V_MAX);
    else if(channel == 3 || channel == 4)
        tgr_reg = (unsigned short)(((float)TGRA_4_VAL * control_output) / (float)V_MAX);


    if(tgr_reg == 0 || control_output <= 0.0) /* Se lo sforzo di controllo si approssima a zero oltre il minimo valore di precisione */
      {
        tgr_reg = TGRA_3_VAL+1;  /* max value for TGR */
      }
    else if(control_output >= V_MAX)    /* Max value 3.3 volt */
            {
             tgr_reg = 0x0000;  /* corrisponde al min valore di TGR */
            }
        else if(channel == 1 || channel == 2) tgr_reg = TGRA_3_VAL - tgr_reg; /* valore per TGRB_3 or TGRD_3 */
        else if(channel == 3 || channel == 4) tgr_reg = TGRA_4_VAL - tgr_reg; /* valore per TGRB_4 or TGRD_4 */

  if(channel == 1) /* Coppia TGRA-TGRB 3 */
   {
    /* imposta il nuovo valore del duty cycle */

    MTU3.TGRB = tgr_reg;  /* new TGRB_3 => change Duty Cycle */

   } else if(channel == 2) /* coppia TGRC-TGRD 3 */
          {
            /* imposta il nuovo valore del duty cycle */

            MTU3.TGRD = tgr_reg;  /* new TGRD_3 => change Duty Cycle */

          } else if(channel == 3) /* coppia TGRA-TGRB 4 */
                {
                /* imposta il nuovo valore del duty cycle */

                MTU4.TGRB = tgr_reg;  /* new TGRB_4 => change Duty Cycle */

                } else if(channel == 4)
                    {
                    /* imposta il nuovo valore del duty cycle */
                    MTU4.TGRD = tgr_reg;  /* new TGRD_4 => change Duty Cycle */
                    }
}
```

La procedura **Init_Port_Dir** inizializza le porte deputate a stabilire la direzione del motore, con il corrispondente pin sulla scheda; in generale, si indica come verso di riferimento quello antiorario, appunto identificato con lo 0.

```
/* Inizializzazione delle porte di Direzione */
void Init_Port_Dir()
{
    // Motor 1 (Connector JN2 - PIN 19)
    PORTD.PDR.BIT.B6 = 1; //OUTPUT
    PORTD.PMR.BIT.B6 = 0; //PD6
    PORTD.PODR.BIT.B6 = 0; //antiorario

    // Motor 2 (Connector JN2 - PIN 16)
    PORTD.PDR.BIT.B3 = 1; //OUTPUT
    PORTD.PMR.BIT.B3 = 0; //PD3
    PORTD.PODR.BIT.B3 = 0; //antiorario

    // Motor 3 (Connector JN2 - PIN 7)
    PORTA.PDR.BIT.B7 = 1; //OUTPUT
    PORTA.PMR.BIT.B7 = 0; //PA7
    PORTA.PODR.BIT.B7 = 0; //antiorario
}
```

Per quanto riguarda la funzione che si occupa di controllare ed eventualmente cambiare la direzione del motore, **Rescale** presenta in ingresso il valore della tensione e il canale corrispondente al motore considerato; se “volt_signal” è negativo, il verso di rotazione viene cambiato in orario e viene restituito il modulo del valore di tensione.

```
float Rescale(float volt_signal, unsigned char channel)
{
    float output = 0.0;

    if(volt_signal < 0.0)
    {
        switch(channel)
        {
        case (1):
        PORTD.PODR.BIT.B6 = 1; //orario
        break;

        case (2):
        PORTD.PODR.BIT.B3 = 1; //orario
        break;

        case (3):
        PORTA.PODR.BIT.B7 = 1; //orario
        break;

        }

        volt_signal *= -1;
    }

    output = (volt_signal/Max_Volt)*V_MAX;
    return output;
}
```

# CAPITOLO III – SENSORI DEL SISTEMA

## 3.1 Unità di Misura Inerziale

Come già accennato, l'IMU utilizzata nel Ballbot è una GY-80 a 9 assi (Figura 10) e comprende – tra l'altro – un giroscopio, un accelerometro, un magnetometro e un barometro. Richiede un'alimentazione di 3.3 V e viene utilizzato il protocollo $I^2C$ per la comunicazione (infatti sono presenti i pin per SCL ed SDA, collegati rispettivamente al JN1 pin 26 e 25 della scheda).

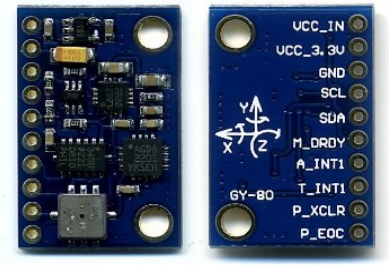



***Figura 10** - GY-80*

In generale, un corpo può ruotare nello spazio in riferimento ai tre assi (Figura 11). Nel dettaglio, i giroscopi misurano le variazioni di posizione angolare e quindi i cambiamenti di orientamento, che in genere sono espresse in gradi al secondo. Il giroscopio classico 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. Per quanto riguarda l'accelerometro, d'altro canto, questo misura le accelerazioni, dunque le variazioni di velocità; quando un corpo si muove a partire dalla propria posizione di riposo in modo proporzionale all'accelerazione, lo spostamento della massa, rilevato tramite un apposito sensore, viene convertito in un'accelerazione a sua volta convertita in un segnale elettrico. Di lì, i valori ricavati da questi due sensori andranno uniti dato che - per poter mantenere allineati gli accelerometri lungo le loro direzioni preferenziali – si necessita di una tipologia di sensori che permetta di calcolare le velocità angolari, ovvero le variazioni che subiscono gli angoli compresi tra la direzione attuale degli accelerometri e quella preferenziale. Partendo dai dati provenienti da questi due sensori, la stima dei tre angoli (Roll, Pitch e Yaw) è stata realizzata tramite un filtro di Kalman esteso.

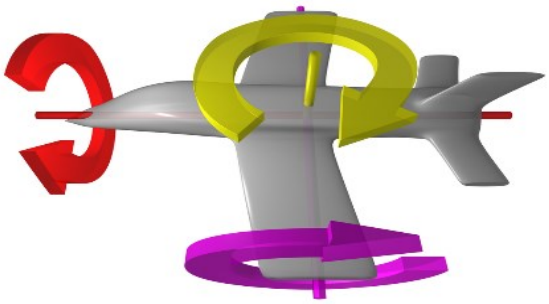


*Figura 11 – Angoli di riferimento*

In realtà, l'orientamento tridimensionale del Ballbot è fornito dall'AHRS (Attitude and Heading Reference System), talvolta chiamato MARG (Magnetic, Angular Rate, and Gravity), che integra e fonde i dati dei giroscopi con quelli dell'accelerometro e del magnetometro. Infatti, combinando le informazioni su velocità angolare ed accelerazione provenienti dall'unità di misura inerziale, la deriva dell'integrazione dei giroscopi è compensata da vettori di riferimento, cioè la gravità e il campo magnetico terrestre. Perciò, l'AHRS rappresenta un sistema più affidabile e preciso rispetto alla singola unità di misura inerziale, tanto da permettere di calcolare una posizione relativa nel tempo.

## 3.2 Sensore di corrente

Per quanto riguarda il rilevamento della corrente per il controllo in coppia, il sensore di corrente scelto è stato l'INA219 (Figura 12), compatibile con il protocollo di comunicazione seriale $I^2C$.

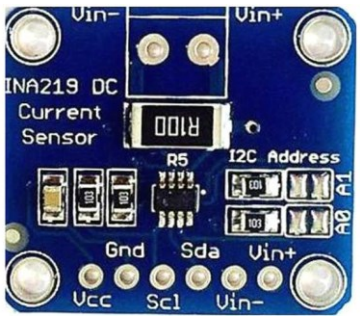



*Figura 12 – INA19*

| Specifiche tecniche |
| --- |
| Resistore di corrente: 0.1 Ω |
| Tensione di alimentazione del carico max: 26 V |
| Range di misura della corrente: ± 3.2 A |
| Risoluzione della misura: 0.8 mA |
| Range della tensione di alimentazione: 3 – 5 V |

Analizzando lo schema presente nel datasheet (Figura 13), i due ingressi analogici *VIN+* e *VIN-* sono collegati ad una resistenza shunt nel bus, mentre *VCC* e *GND* sono relativi all'alimentazione di 3.3 V. La tensione del bus rilevata può variare da un minimo di 0 a

un massimo di 26 V e si misura la piccola caduta di potenziale grazie alla *R1-SHUNT*, rispetto a *VIN-* (a massa).

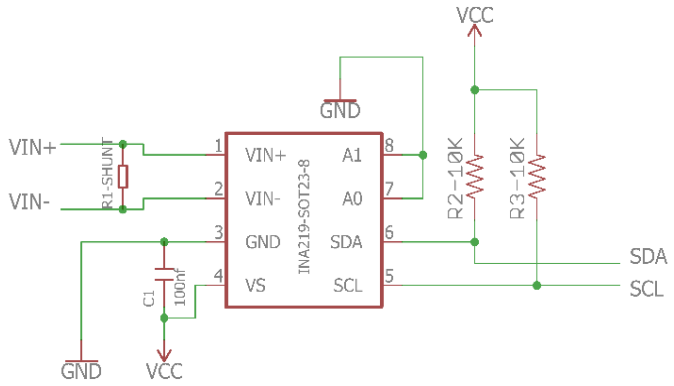



***Figura 13*** *– Schema logico del sensore*

Quando il sensore è in modalità operativa normale, l'INA219 converte continuamente la tensione di shunt fino al numero impostato nella funzione di media della tensione di shunt (registro di configurazione). Il dispositivo converte quindi la tensione del bus fino al numero impostato nella media della tensione del bus (registro di configurazione). Tutti i calcoli di corrente e potenza vengono eseguiti in background e non influiscono sul tempo di conversione. La modalità *ADC Off* (impostata dal registro di configurazione) interrompe tutte le conversioni. Sebbene INA219 possa essere letto in qualsiasi momento e i dati dell'ultima conversione rimangano disponibili, viene fornito il bit pronto per la conversione (registro di stato) per facilitare il coordinamento delle conversioni one-shot o attivate. Riguardo alla misura della potenza, la corrente e la tensione del bus sono convertite in differenti punti del tempo, in base alla risoluzione e ad altre impostazioni. Dopo aver impostato il registro di calibrazione, vengono aggiornati il registro di corrente (04h) e il registro di potenza (03h) con i corrispondenti valori della tensione di shunt e del bus. Finché il registro di calibrazione non viene programmato, il registro di corrente e il registro di potenza rimangono a zero.

Il *Calibration Register* (registro di calibrazione) è calcolato grazie all'equazione

$$Calibration\ Register = trunc\left(\frac{0.04096}{Current_{LSB} \cdot R_{SHUNT}}\right)$$

dove "$Current_{LSB}$" è il valore predefinito dell'LSB per il registro di corrente ed è pari a

$$Current_{LSB} = \frac{Max\ Expected\ Current}{2^{15}}$$

Dopo aver impostato il registro di calibrazione, si può calcolare il valore nel *Current Register* (registro di corrente) tramite il registro di calibrazione e quello della tensione di shunt

$$Current\ Register = \frac{Shunt\ Voltage\ Register \cdot Calibration\ Register}{4096}$$

## 3.3 Protocollo di comunicazione I²C

I valori presenti nei vari registri d'interesse vengono trasmessi grazie al protocollo di comunicazione I²C, compatibile proprio con l'INA219. *Inter Integrated Circuit* (o IIC) è un protocollo che permette la comunicazione tra due o più dispositivi tramite l'uso di due fili differenti (Figura 14); infatti, le informazioni vengono inviate in modo seriale utilizzando una linea per i dati (*Serial Data Line*) e una per il clock (*Serial Clock Line*). Riguardo alla nomenclatura, si chiama *Master* il dispositivo che controlla il bus in un periodo di tempo; infatti, esso genera i segnali di start e di stop, regolando il segnale di clock (nel robot è il µ-controller). Invece, vengono chiamati *Slave* i dispositivi che, mettendosi in ascolto sul bus, ricevono o inviano i dati in base alle richieste del master (come IMU o INA219). A differenza di altri tipi di bus, le linee del bus I²C devono essere mantenute a livello alto usando due resistenze di pull-up.

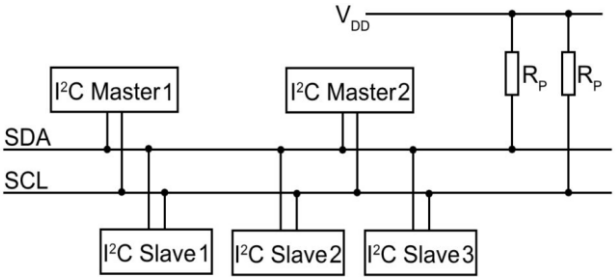



***Figura 14*** *– Protocollo di comunicazione utilizzato*

Il valore delle resistenze varia in base a certi parametri, come la tensione di alimentazione, la capacità del bus e il numero dei dispositivi collegati. Generalmente, nella trasmissione seriale dei dati il segnale SDA può cambiare stato solamente quando il segnale SCL è basso e – solo in seguito - il suo valore viene letto durante il fronte di salita/discesa dell'SCL. Per cui, all'inizio il master invia un bit di start, il quale consiste nel mandare a livello basso la linea SDA mentre quella SCL è a livello alto, abbassando

successivamente anche l'SCL. Dunque, il primo byte immesso nel bus dal master dopo lo start è quello dell'indirizzo dello slave (ADDR), identificato dai primi 7 bit (in quanto l'LSB indica se l'operazione da effettuare è di lettura o scrittura). Poiché il protocollo $I^2C$ richiede che ogni byte trasmesso debba essere concluso con la condizione di ACK, sono necessari 8 impulsi di clock per il sincronismo dei dati trasmessi dal master e solamente uno per sincronizzare l'Acknowledge dello slave. In seguito, avviene l'invio e la ricezione effettiva del byte di dati (DATA); per ogni bit immesso sulla linea SDA viene generato un impulso di clock sulla linea SCL (per cui i dati validi sono quelli cui corrisponde l'SCL a livello alto). In seguito all'attesa per l'invio e la ricezione dell'ACK (dopo ogni byte), il master invia un segnale di stop, realizzato mandando prima la linea SCL e poi la linea SDA al livello alto, grazie alle resistenze di pull-up. Il sensore ha due pin di indirizzo, A0 e A1, e vengono letti all'inizio di ogni evento di comunicazione con 16 possibili diverse configurazioni (Tabella 1). In particolare, lo stato dei pin viene campionato su ogni comunicazione bus e deve essere impostato prima che si verifichi qualsiasi attività sull'interfaccia.

| A1 | A0 | SLAVE ADDRESS |
|---|---|---|
| GND | GND | 1000000 |
| GND | $V_{S+}$ | 1000001 |
| GND | SDA | 1000010 |
| GND | SCL | 1000011 |
| $V_{S+}$ | GND | 1000100 |
| $V_{S+}$ | $V_{S+}$ | 1000101 |
| $V_{S+}$ | SDA | 1000110 |
| $V_{S+}$ | SCL | 1000111 |
| SDA | GND | 1001000 |
| SDA | $V_{S+}$ | 1001001 |
| SDA | SDA | 1001010 |
| SDA | SCL | 1001011 |
| SCL | GND | 1001100 |
| SCL | $V_{S+}$ | 1001101 |
| SCL | SDA | 1001110 |
| SCL | SCL | 1001111 |

***Tabella 1** – Configurazioni possibili*

L'accesso a un registro particolare del sensore si ottiene scrivendo il valore appropriato sul puntatore del registro e la scrittura in un registro inizia con il primo byte trasmesso dal master, ovvero l'indirizzo dello slave. Se questo è valido, il master invia un altro byte che – invece – contiene l'indirizzo su cui saranno scritti i dati, il quale aggiorna il puntatore al registro desiderato. Mentre avviene la lettura dal sensore, l'ultimo valore memorizzato nel puntatore del registro mediante un'operazione di scrittura determina quale registro viene letto durante un'operazione di lettura.

## 3.4 Analisi del codice

Considerando il codice relativo all'IMU, per primo, nel file "AHRS.c" si trovano le funzioni usate per rielaborare i dati provenienti dal sensore, grazie all'ausilio di una struttura dati contenente i dati filtrati degli angoli di Tait-Bryan e delle velocità angolari. La funzione **getAHRS**, che presenta in ingresso i riferimenti a Roll, Pitch e Yaw, è utilizzata per ricavare gli angoli e può essere chiamata con una frequenza minore rispetto a quella del filtro implementato da **madgwickFilterUpdate**.

```
void getAHRS(float* pitch, float* yaw, float* roll){
   *yaw   = atan2(2.0f * (q1 * q2 + q0 * q3), q0 * q0 + q1 * q1 - q2 * q2 - q3 * q3);
   *pitch = -asin(2.0f * (q1 * q3 - q0 * q2));
   *roll  = atan2(2.0f * (q0 * q1 + q2 * q3), q0 * q0 - q1 * q1 - q2 * q2 + q3 * q3);
   /*
   *yaw   *= 180.0f / PI;
   *yaw   -= 3.316666666; // Declination at Ancona, Italy is 3 degrees 19 minutes
   *yaw   *= PI / 180.0f;*/
}
```

La funzione **getYPR** riceve in ingresso le misurazioni eseguite in quaternioni e li trasforma in gradi e in radianti, appoggiandosi sulla struttura precedentemente illustrata.

```
void getYPR(MAG_data* mag_data ,IMU_temp* imu_temp, AHRS_data* ahrs_data){
    madgwickFilterUpdate(imu_temp->gyrRoll, imu_temp->gyrPitch, imu_temp->gyrYaw,
           imu_temp->accRoll, imu_temp->accPitch, imu_temp->accYaw, mag_data->x, mag_data->y, mag_data->z);

    getAHRS(&ahrs_data->PitchRad, &ahrs_data->YawRad, &ahrs_data->RollRad);
    ahrs_data->RollDeg = ahrs_data->RollRad * (180.0/PI);
    ahrs_data->PitchDeg = ahrs_data->PitchRad * (180.0/PI);
    ahrs_data->YawDeg = ahrs_data->YawRad * (180.0/PI);

    /* Calcolo delle velocità angolari degli angoli */
    ahrs_data->omegaRollRad = imu_temp->gyrRoll ;//* (PI/180.0);
    ahrs_data->omegaPitchRad = imu_temp->gyrPitch ;//* (PI/180.0);
    ahrs_data->omegaYawRad = imu_temp->gyrYaw ;//* (PI/180.0);

    ahrs_data->omegaRollDeg = ahrs_data->omegaRollRad * (180.0/PI);
    ahrs_data->omegaPitchDeg = ahrs_data->omegaPitchRad * (180.0/PI);
    ahrs_data->omegaYawDeg = ahrs_data->omegaYawRad * (180.0/PI);

    OmegaYawRad=ahrs_data->omegaYawRad;
    OmegaYawDeg=ahrs_data->omegaYawDeg;
}
```

Per rendere il sistema più organizzato, all'interno di "Setup_AHRS.c" è presente la procedura **Setup_MARG** la quale si occupa dell'inizializzazione dell'IMU e della calibrazione del magnetometro, con la possibilità di utilizzare valori predefiniti.

```
//Routine di setup dell'imu
void Setup_MARG(AHRS_out* ahrs)
{
    char msg[12];

    lcd_display(LCD_LINE1," IMU SETUP ");
    lcd_display(LCD_LINE2," IN CORSO ");

    imu_init(&ahrs->sens);
    mag_init(&ahrs->mag);

    lcd_display(LCD_LINE4,"Calibrazione");
    lcd_display(LCD_LINE5,"Magnetometro");

    /* Con SET impostato su 1 il magnetometro si setta con valori predefiniti. Per eseguire la calibrazione
       impostare nella define ad inizio file "#define SET 2"*/
    switch(SET)
    {
    case 1: setYPR(msg, &ahrs->mag);
            break;
    case 2: calibrationYPR(msg, &ahrs->mag);
            break;
    default: setYPR(msg, &ahrs->mag);
             break;
    }

}
```

Invece, tramite la funzione **Read_MARG**, che in ingresso presenta il puntatore alla struttura dati, si consente la lettura dei valori dell'IMU e del magnetometro, chiamando anche **getYPR**.

```
// Funzione di lettura dei dati dell' MARG filtrati
void Read_MARG(AHRS_out* ahrs)
{
    imu_read(&ahrs->raw, &ahrs->sens, &ahrs->temp);
    mag_read(&ahrs->mag);
    getYPR(&ahrs->mag, &ahrs->temp, &ahrs->ahrs_data);
}
```

Prima di analizzare il file sorgente relativo all'IMU, nel file di intestazione "imu.h" si individuano tre strutture dati importanti: *imu_raw*, che contiene i dati grezzi dell'IMU, *imu_sens*, che contiene i parametri caratteristici del sensore, *imu_temp*, che ha - invece - i dati rielaborati. In "imu.c" la procedura **imu_init** viene eseguita con il Ballbot in posizione di equilibrio così da rilevare i valori degli angoli e della velocità angolare di riferimento.

```
int imu_init(IMU_sens* imu_sens){
    // avvia i2c
    if (i2c_init())
        return 0x1;
    // avvia sensore con la libreria Inversense
    if (mpu_init(0))
        return 0x2;
    // abilita sensori Giroscopio e Accelerometro
    if (mpu_set_sensors(INV_XYZ_ACCEL | INV_XYZ_GYRO))
        return 0x3;
    // imposta frequenza la conversione dei dati inerziali
    if (mpu_set_sample_rate(200/*Hz*/)) //Hz
        return 0x4;

    // leggi sensibilità  del sensore
    mpu_get_accel_sens(&imu_sens->acc_sens);
    mpu_get_gyro_sens(&imu_sens->gyr_sens);
    return 0x0;
}
```

La procedura **imu_read**, che in ingresso ha i riferimenti alle tre strutture dati, legge i dati dal giroscopio e dall'accelerometro.

```
int imu_read(IMU_raw* imu_raw, IMU_sens* imu_sens, IMU_temp* imu_temp){

    short data_acc[3] = {imu_raw->accRoll,imu_raw->accPitch,imu_raw->accYaw};
    mpu_get_accel_reg(data_acc, NULL);
    imu_raw->accRoll = data_acc[0];
    imu_raw->accPitch = data_acc[1];
    imu_raw->accYaw = data_acc[2];

    short data_gyr[3] = {imu_raw->gyrRoll, imu_raw->gyrPitch, imu_raw->gyrYaw};
    mpu_get_gyro_reg(data_gyr, NULL);
    imu_raw->gyrRoll = data_gyr[0];
    imu_raw->gyrPitch = data_gyr[1];
    imu_raw->gyrYaw = data_gyr[2];

    imu_temp->accRoll=imu_raw->accRoll / imu_sens->acc_sens;
    imu_temp->gyrRoll = imu_raw->gyrRoll / imu_sens->gyr_sens * 0.01745329252;
    imu_temp->accPitch = imu_raw->accPitch / imu_sens->acc_sens;
    imu_temp->gyrPitch = imu_raw->gyrPitch / imu_sens->gyr_sens * 0.01745329252;
    imu_temp->accYaw = imu_raw->accYaw / imu_sens->acc_sens;
    imu_temp->gyrYaw = imu_raw->gyrYaw / imu_sens->gyr_sens * 0.01745329252;
    return 0;
}
```

Analizzando – ora – il codice dell'$I^2C$ e del sensore di corrente, tutte le funzioni responsabili alla comunicazione tra master e slave si trovano all'interno della libreria "i2c.c"; oltre all'inizializzazione, infatti, sono presenti le procedure utilizzare per la

lettura e la scrittura, **i2c_read** e **i2c_write**, che ricevono in ingresso l'indirizzo dello slave, il numero del registro, il numero di byte da leggero sullo slave e il puntatore al buffer su cui verranno memorizzati i dati.

```
riic_ret_t i2c_read (uint8_t slave_addr, uint8_t register_number, uint8_t num_bytes,
                     uint8_t *dest_buff)
{
     /* Storage for the slave address and target register. */
    uint8_t     addr_and_register[2];
    //riic_ret_t  ret = RIIC_OK;

    addr_and_register[0] = slave_addr<<1;
    addr_and_register[1] = register_number;

    ret |= R_RIIC_MasterTransmitHead(CHANNEL_0, addr_and_register, 2);

    /* Now read the data from the target register into the destination buffer. */
    ret |= R_RIIC_MasterReceive(CHANNEL_0, slave_addr<<1, dest_buff, num_bytes);


    return ret;
} //END - i2c_read(..)
riic_ret_t i2c_write(uint8_t slave_addr, uint8_t register_number, uint8_t num_bytes,
                     uint8_t *source_buff)
{
    /* Storage for the slave address and target register. */
    uint8_t     addr_and_register[2];


    addr_and_register[0] = slave_addr<<1;
    addr_and_register[1] = register_number;

    ret |= R_RIIC_MasterTransmitHead(CHANNEL_0, addr_and_register, 2);

    /* Now write the data from the source buffer into the target register. */
    ret |= R_RIIC_MasterTransmit(CHANNEL_0, source_buff, num_bytes);

    return ret;
} //END - i2c_write(..)
```

Per quanto riguarda l'INA219, per primo nel file di intestazione "ina219.h" è presente la struttura dati che contiene l'indirizzo del dispositivo, la corrente e la tensione di shunt. Nel file "ina219.c" si trova la procedura **sens_curr_init** responsabile dell'inizializzazione del sensore; essa riceve in ingresso la struttura dati e il canale corrispondete al sensore di corrente, in modo da impostare sia il registro di calibrazione sia quello di configurazione. Vengono eseguite diverse chiamate alle funzioni $I^2C$ e – nel caso in cui non ci siano errori – viene stampato a schermo la buona riuscita dell'inizializzazione.

```
/* Initialization function */
short sens_curr_init(struct sens_cur * SdC, short channel) {
    res = RIIC_OK;

        switch (channel) {
        case 0:
            SdC->ADDR = INA219_IIC_ADDRESS_1;
            break;
        case 1:
            SdC->ADDR = INA219_IIC_ADDRESS_2;
            break;
        case 2:
            SdC->ADDR = INA219_IIC_ADDRESS_3;
            break;
        }
```

```
    /*
      // By default we use a pretty huge range for the input voltage,
      // which probably isn't the most appropriate choice for system
      // that don't use a lot of power.  But all of the calculations
      // are shown below if you want to change the settings.  You will
      // also need to change any relevant register settings, such as
      // setting the VBUS_MAX to 16V instead of 32V, etc.
      // VBUS_MAX = 32V             (Assumes 32V, can also be set to 16V)
      // VSHUNT_MAX = 0.32          (Assumes Gain 8, 320mV, can also be 0.16, 0.08,
      // 0.04) RSHUNT = 0.1               (Resistor value in ohms)

      // 1. Determine max possible current
      // MaxPossible_I = VSHUNT_MAX / RSHUNT
      // MaxPossible_I = 3.2A

      // 2. Determine max expected current
      // MaxExpected_I = 2.0A

      // 3. Calculate possible range of LSBs (Min = 15-bit, Max = 12-bit)
      // MinimumLSB = MaxExpected_I/32767
      // MinimumLSB = 0.000061              (61uA per bit)
      // MaximumLSB = MaxExpected_I/4096
      // MaximumLSB = 0,000488              (488uA per bit)

      // 4. Choose an LSB between the min and max values
      //    (Preferrably a roundish number close to MinLSB)
      // CurrentLSB = 0.0001 (100uA per bit)

      // 5. Compute the calibration register
      // Cal = trunc (0.04096 / (Current_LSB * RSHUNT))
      // Cal = 4096 (0x1000)
    */
      uint16_t config, ina219_calValue = 4096;
      uint8_t cal8[2], config8[2];
      cal8[1]= ina219_calValue & 0xFF;
      cal8[0]=(ina219_calValue >> 8);

    /*

      // 6. Calculate the power LSB
      // PowerLSB = 20 * CurrentLSB
      // PowerLSB = 0.002 (2mW per bit)

      // 7. Compute the maximum current and shunt voltage values before overflow
      //
      // Max_Current = Current_LSB * 32767
      // Max_Current = 3.2767A before overflow
      //
      // If Max_Current > Max_Possible_I then
      //    Max_Current_Before_Overflow = MaxPossible_I
      // Else
      //    Max_Current_Before_Overflow = Max_Current
      // End If
      //
      // Max_ShuntVoltage = Max_Current_Before_Overflow * RSHUNT
      // Max_ShuntVoltage = 0.32V
      //
      // If Max_ShuntVoltage >= VSHUNT_MAX
      //    Max_ShuntVoltage_Before_Overflow = VSHUNT_MAX
      // Else
      //    Max_ShuntVoltage_Before_Overflow = Max_ShuntVoltage
      // End If

      // 8. Compute the Maximum Power
      // MaximumPower = Max_Current_Before_Overflow * VBUS_MAX
      // MaximumPower = 3.2 * 32V
      // MaximumPower = 102.4W

    */

      // Set Calibration register to 'Cal' calculated above
      res = i2c_write(SdC->ADDR, INA219_REG_CALIBRATION, 2, cal8);

      // Set Config register to take into account the settings above
      config = INA219_CONFIG_BVOLTAGERANGE_32V |
                       INA219_CONFIG_GAIN_8_320MV | INA219_CONFIG_BADCRES_12BIT |
                       INA219_CONFIG_SADCRES_12BIT_1S_532US |
                       INA219_CONFIG_MODE_SVOLT_CONTINUOUS;

      config8[1]=config & 0xff;
      config8[0]=(config >> 8);
      res = i2c_write(SdC->ADDR, INA219_REG_CONFIG, 2, config8);

      if (res == 0) {
            lcd_display(LCD_LINE4, "init SdC1 ok");
      } else {
            lcd_display(LCD_LINE4, "init SdC1 er");
      }

      return res;
    }
```

La funzione che effettivamente esegue la lettura dal sensore è **read_curr** che in ingresso ha la struttura dati corrispondente ad uno dei sensori; in questo caso – dopo l'invocazione della funzione $I^2C$ – la corrente viene misurata e il suo valore è immagazzinato nella struttura dati.

```
/* read and return load current in mA */
short read_curr(struct sens_cur * SdC) {
    int ina219_currentDivider_mA = 10; // Current LSB = 100uA per bit (1000/100 = 10)
    int res=0;
    float curr=0;
    unsigned char data[2];
    uint16_t tmp;
    int16_t tmp2;
    res = i2c_read(SdC->ADDR, INA219_REG_CURRENT, 2, data);
    if (res == 0) {
        if (data[0] > 0x7f) {
            data[0] = data[0] - 0x80;
            tmp = (((uint16_t)data[0] << 8) | (data[1]));
            tmp = tmp - 1;
            tmp = ~tmp;
            tmp = tmp - 0x8000;
            curr = -((float)tmp) / ina219_currentDivider_mA;
        } else {
            tmp = (((uint16_t)data[0] << 8) | (data[1]));
            tmp2 = (int16_t)tmp;
            curr = ((float)tmp2) / ina219_currentDivider_mA;
        }
    }
    SdC->curr = curr;
    return res;
}
```

# CAPITOLO IV – CONTROLLO DEL SISTEMA

## 4.1 Regolatori PID ed implementazione

I regolatori PID (Proporzionale-Integrale-Derivativo) sono dei sistemi in retroazione negativa che permettono di regolare i parametri di un sistema di controllo. Il controllore acquisisce in ingresso un valore da un processo e lo confronta con un valore di riferimento. La differenza (il segnale di errore) viene, quindi, usata per determinare il valore della variabile di uscita del controllore, che è la variabile manipolabile del processo. Il PID regola l'uscita in base a tre azioni diverse, disposte in parallelo: proporzionale (il valore del segnale di errore), integrale (i valori passati del segnale di errore), derivativa (quanto velocemente il segnale di errore varia). Infatti, la somma di questi tre contributi costituisce la variabile di controllo $u_{PID}$ usata per controllare il processo, secondo la formula

$$u_{PID}(kh) = P(kh) + I(kh) + D(kh)$$

dove – nel dominio del tempo discreto – $h$ rappresenta il periodo di campionamento. In dettaglio, l'azione proporzionale è ricavata moltiplicando il segnale di errore con la costante, come nella formula

$$P(kh) = K_G(b \cdot u(kh) - y(kh))$$

L'azione integrale è proporzionale all'integrale nel tempo discreto del segnale di errore, come indicato dalla formula, dove $T_t$ è la costante di antiwindup e $T_I$ è la costante di tempo integrale

$$I(kh + h) = I(kh) + \frac{K_G \cdot h}{T_I}(u(kh) - y(kh)) + \frac{h}{T_t}(u_{aw}(kh) - u_{PID}(kh))$$

Per migliorare le prestazioni del controllore si può aggiungere l'azione derivativa, data da

$$D(kh) = \frac{T_D}{T_D + Nh} D(kh - h) - \frac{K_G \cdot T_D \cdot N}{T_D + Nh}(y(kh) - y(kh - h))$$

dove $T_D$ è la costante di tempo derivativa ed N è il guadagno massimo per la parte derivativa.

Un problema particolare causato dalla presenza dell'azione derivativa è l'impossibilità teorica di realizzare un "derivatore puro": sarebbe necessario, infatti, misurare il valore del segnale di errore nel futuro. Per questo si calcola una derivata ingegneristica, che approssima il derivatore fino ad una certa frequenza. Nel Ballbot viene utilizzato il regolatore PID per il controllo in coppia dei motori DC (Figura 15): grazie ai dati forniti dai sensori (IMU e INA219), poiché si conosce la posizione finale desiderata (equilibrio), vengono calcolate le coppie motrici (con una certa direzione di rotazione) che devono essere applicate ai motori (PWM).

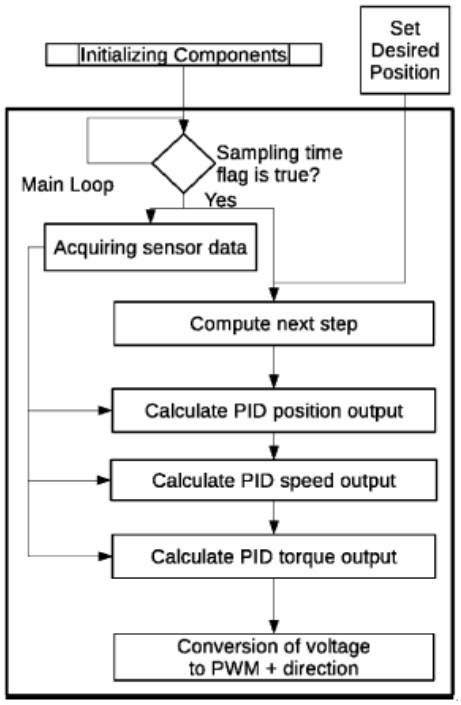



***Figura 15*** *– Schema per il controllo*

Per avere un PID affidabile e preciso, prima di implementarlo a livello di codice, è stata fatta una simulazione su Simulink (Figura 16): all'interno di una funzione Matlab, in cui si stabiliscono i parametri caratteristici del regolatore, si scrivono le formule illustrate; l'uscita che si ricava sarà proprio la corrente in ingresso al modello del motore DC, misurando quindi l'andamento della corrente e della velocità angolare del sistema.

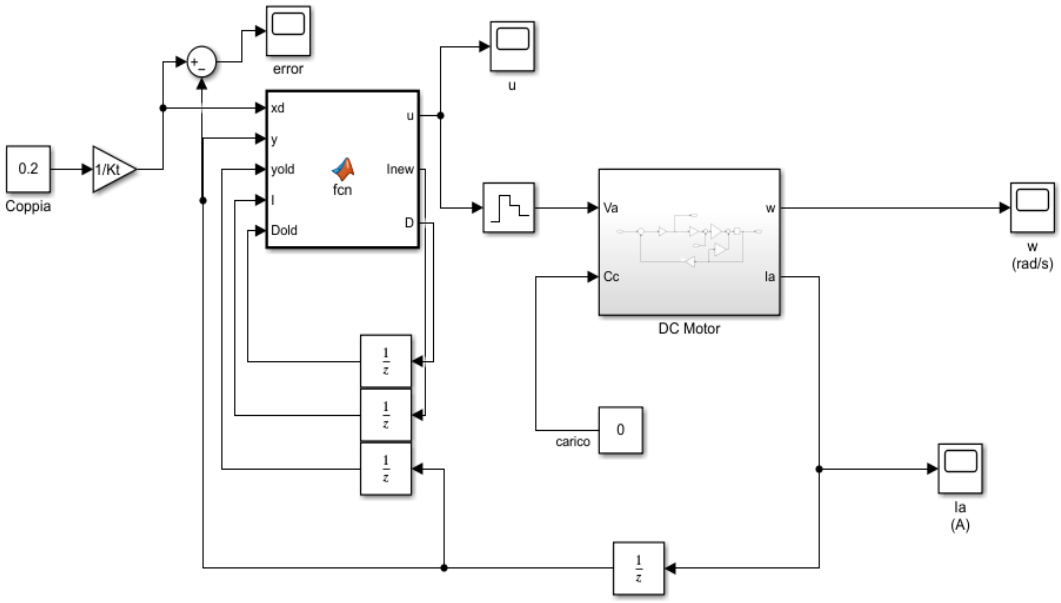



***Figura 16*** *– Schema di funzionamento del controllo PID nel motore DC*

In dettaglio, nel file d'intestazione "PID.h" la struttura dati è utilizzata per immagazzinare i parametri caratteristici, come il guadagno proporzionale, il tempo di campionamento, la parte integrale, ecc.

```
typedef struct PID {
    float k;        // Proportional gain
    float ti;       // Integral time
    float td;       // Derivative time
    float tt;       // Reset time
    float n;        // Maximum derivative gain
    float b;        // Fraction of set point in prop. term
    float ulow;     // Low output limit
    float uhigh;    // High output limit
    float h;        // Sampling period
    float bi;       // First term integral coefficient
    float ar;       // Second term integral coefficient
    float ad;       // First term derivative coefficient
    float bd;       // Second term derivative coefficient
    float partI;    // Integral part
    float partD;    // Derivative part
    float yold;     // Delayed measured

}pidSt;
```

Dopo aver inizializzato il PID di corrente tramite la procedura **init_pid_curr**, nel file "PID.c" la funzione **calcPID** riceve in ingresso il segnale desiderato, il segnale attuale e la struttura dati che lo identifica. In particolare, per primo si stima la parte derivativa e viene memorizzato il risultato; successivamente, si calcola l'uscita del segnale e – dopo una funzione antiwindup – viene aggiornata la parte integrale. Infine, si memorizza il valore finale dell'uscita, ritornando il segnale di controllo.

```
void init_pid_curr(pidSt * pid_curr) {
    pid_curr->k = 0.8;
    pid_curr->ti = 0.015;
    pid_curr->td = 0.001;
    pid_curr->tt = 0.01;
    pid_curr->n = 20;
    pid_curr->b = 1;
    pid_curr->ulow = -12;
    pid_curr->uhigh = 12;
    pid_curr->h = 0.02;
    pid_curr->bi = (pid_curr->k * pid_curr->h)/pid_curr->ti;
    pid_curr->ar = pid_curr->h/pid_curr->tt;
    pid_curr->ad = pid_curr->td/(pid_curr->td + pid_curr->n * pid_curr->h);
    pid_curr->bd = pid_curr->k*(pid_curr->n * pid_curr->ad);

    pid_curr->partI = 0;
    pid_curr->partD = 0;
    pid_curr->yold = 0;
}
float calcPID (float uc, float y, pidSt * pidVal) {
    /* P proportional part, v pid output, u final output */
    float P,v,u;
    P = pidVal->k * (pidVal->b * uc - y);

    /* Evaluating derivative part and store in struct */
    pidVal->partD = pidVal->ad * pidVal->partD - pidVal->bd * (y -  pidVal->yold);

    /* Pure pid output signal */
    v = P + pidVal->partD + pidVal->partI;

    /* Antiwindup function */
    if (v < pidVal->ulow) {
        u = pidVal->ulow;
    } else {
        if (v > pidVal->uhigh) {
            u = pidVal->uhigh;
        } else {
            u = v;
        }
    }
    /* Integral part update */
    pidVal->partI = pidVal->partI + pidVal->bi * (uc - y) + pidVal->ar * (u - v);

    /* Memorizing final output value */
    pidVal->yold = y;

    return u;
}
```

## 4.2 Modello dinamico e relativo codice

Dopo aver illustrato i regolatori PID nel controllo in coppia, ora si può considerare il modello dinamico del sistema. Per primo, si cerca di controllare il pendolo agendo tramite impulsi e forze esterne al fine di mantenerlo nella posizione denotata dall'angolo $\theta = 0$ che rappresenta un punto di equilibrio instabile, a causa delle forze che agiscono sul sistema. Come già detto in precedenza, il Ballbot è composto da una sfera e da un corpo, perciò si possono individuare tre sistemi di riferimento diversi: quello del terreno (*earth* o E), quello del pendolo (*pendulum* o P) e quello della palla (*ball* o B). I tre angoli di rotazione mostrati (rollio, beccheggio, imbardata) vengono assunti come descrizione della configurazione dell'orientamento del corpo rigido rispetto ad un sistema fisso di orientamento. Analizzando esclusivamente il modello del sistema rispetto alle coppie $T$ in ingresso, si indica con $U$ l'effettivo vettore coppia applicato

dalle omniwheels alla sfera e con $\bar{U}$ il vettore coppia disponibile alle ruote, associando le coppie fornite dai motori in ingresso al vettore di ingresso, per cui

$$\bar{\mathrm{U}} = \begin{bmatrix} u_x & u_y & u_z \end{bmatrix}^T$$

Per la conservazione della potenza rotazionale e per le leggi del moto circolare si può asserire che sono verificate sia la relazione $\omega_W \cdot \tau_W = \omega_B \cdot \tau_B$ sia l'equazione $R_w \cdot \omega_W = R_b \cdot \omega_B$ , perciò si può ottenere il parametro *k*, definito come il rapporto tra i momenti meccanici (o le velocità angolari) fra palla e ruote:

$$k = \frac{\omega_W}{\omega_B} = \frac{\tau_b}{\tau_w} = \frac{R_b}{R_w}$$

Dunque, il vettore coppia disponibile alle omniwheels è legato all'effettivo vettore coppia dalla relazione $U = k \cdot \bar{\mathrm{U}}$

Esprimendo con $T_i$ la coppia vettoriale fornita dalla i-esima ruota $W_i$ ed erogata dall'i-esimo motore, il vettore $\bar{U}$ deve avere le componenti legate a tali coppie, in base alla configurazione fisica (Figura 17)

$$\bar{\mathrm{U}} = R^P_{W_1}(\alpha, \beta) \begin{bmatrix} T_1 \\ 0 \\ 0 \end{bmatrix} + R^P_{W_2}\left(\alpha, \beta + \frac{2}{3\pi}\right) \begin{bmatrix} T_2 \\ 0 \\ 0 \end{bmatrix} + R^P_{W_3}\left(\alpha, \beta - \frac{2}{3\pi}\right) \begin{bmatrix} T_3 \\ 0 \\ 0 \end{bmatrix}$$

perciò si ottiene

$$\bar{\mathrm{U}} = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \begin{bmatrix} \cos(\alpha) \cdot \left[T_1 \cdot \cos(\beta) + T_2 \cdot \cos\left(\beta + \frac{2}{3\pi}\right) + T_3 \cdot \cos\left(\beta - \frac{2}{3\pi}\right)\right] \\ \cos(\alpha) \cdot \left[T_1 \cdot \sin(\beta) + T_2 \cdot \sin\left(\beta + \frac{2}{3\pi}\right) + T_3 \cdot \sin\left(\beta - \frac{2}{3\pi}\right)\right] \\ -\sin(\alpha) \cdot [T_1 + T_2 + T_3] \end{bmatrix}$$

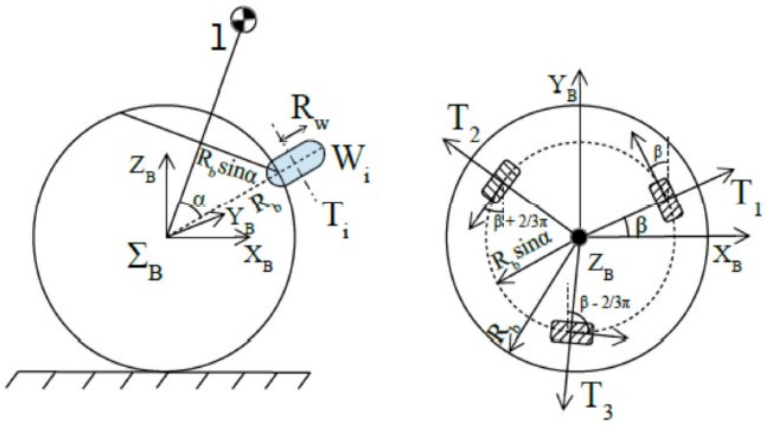



***Figura 17*** *– Modello fisico del Ballbot*

L’algoritmo di conversione è implementato nel file sorgente “matrix.c” ed – in particolare – nella funzione **TtoV** la quale riceve in ingresso le tre coppie dei rispettivi motori e il puntatore all’effettivo vettore coppia. Nel codice è stata utilizzata una variabile di appoggio e le funzioni trigonometriche sono state sostituite con i valori numerici corrispondenti. Infine, si memorizzano le singole componenti del vettore coppia effettivo applicato dalle omniwheels.

```
void TtoV(float T1, float T2, float T3, float* u) {
    float uMarked[3];   /* Coppia disponibile alle omniwheels */
    float U[3];         /* Coppia effettiva applicata */
    u = &U[3];
    //float R_b = 0.12;
    //float R_w = 0.053;
    float k = 2.26415094;   /* Rapporto tra le coppie (o tra le velocità angolari) k = R_b/R_w */

    /* cos(alpha)*(T1*cos(beta) + T2*cos(beta + 2/(3*PI_GREEK)) + T3*cos(beta - 2/(3*PI_GREEK))) */
    uMarked[0] = 0.7660*(T1*(-0.5000) + T2*(-0.6712) + T3*(-0.3064));
    /* cos(alpha)*(T1*sin(beta) + T2*sin(beta + 2/(3*PI_GREEK)) + T3*sin(beta - 2/(3*PI_GREEK))) */
    uMarked[1] = 0.7660*(T1*0.8660 + T2*0.7413 + T3*0.9519);
    /* -(sin(alpha)*(T1+T2+T3)) */
    uMarked[2] = -(0.6428*(T1+T2+T3));

    U[0] = uMarked[0]*k;
    U[1] = uMarked[1]*k;
    U[2] = uMarked[2]*k;

}
```

# CAPITOLO V – SOFTWARE DEL BALLBOT

## 5.1 Funzione main

Dopo aver analizzato il funzionamento del robot e dopo aver illustrato in dettaglio le librerie create, le procedure e gli algoritmi vengono inglobati nel “main.c”, che rappresenta il cuore del programma. Infatti, per primo si includono le librerie (di sistema e non), poi si dichiarano il buffer del display, la struttura esterna utilizzata per i dati dell’encoder e quella usata per la temporizzazione (timer).

```
Includes   <System Includes> , "Project Includes"
#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <machine.h>
#include "platform.h"
#include "r_switches.h"
#include "./I2C/CMT.h"
#include "./I2C/i2c.h"
#include "./I2C/ina219.h"
#include "./I2C/PrintOnScreen.h"
#include "./I2C/SetupAHRS.h"
#include "encoder_1.h"
#include "encoder_2.h"
#include "encoder_3.h"
#include "pwm.h"
#include "PID.h"
#include "filters.h"
#include "matrix.h"


/* Declare display buffer */
uint8_t  lcd_buffer[13];

/* Encoder */
unsigned char Enable_lcd_enc = 1;
extern struct encoder_data enc;

* extern struct timerClocks timers
extern struct timerClocks timers;
```

All’interno della funzione **main** si dichiarano la struttura dati necessaria all’algoritmo AHRS, le tre strutture utilizzate per memorizzare i dati dei rispettivi sensori di corrente e le tre strutture utilizzate per immagazzinare i dati del PID. Successivamente, si definiscono le variabili locali che vengono usate per identificare le coppie, il vettore coppia effettivo, i segnali PWM, le correnti, ecc. Poi, dopo aver inizializzato il display e il timer, si inizializzano il puntatore al vettore coppia effettivo e il segnale desiderato (il quale verrà utilizzato in seguito nel calcolo del PID).

In seguito, si prosegue – con questo specifico ordine - all'inizializzazione degli encoder, della periferica MTU2 utile per la generazione del PWM sui canali previsti, delle porte di direzione dei tre motori, dei parametri per il controllo PID. Poi, vengono inizializzati l'$I^2C$, i sensori di corrente e tutte le funzioni di impostazione dell'accelerometro, del magnetometro, del giroscopio e della calibrazione.

```
void main(void)
{
    /*Dichiarazioni strutture dati e variabili utilizzate.*/

    AHRS_out ahrs;
    struct sens_cur inas_1, inas_2, inas_3;
    pidSt pid_curr_1, pid_curr_2, pid_curr_3;
    float input, tmp;
    float* u;
    float U[3] = {0.0, 0.0, 0.0};
    float t1, t2, t3;
    float v_signal_1, v_signal_2, v_signal_3;
    float curr_1, curr_1f, curr_2, curr_2f, curr_3, curr_3f;

    lcd_initialize();
    lcd_clear();
    CMT_init();

    u = &U[3];
    input = 0.2/0.64;       // Segnale desiderato

    /* Funzioni di setup dell'encoder */
    encoder_1_init();
    encoder_2_init();
    encoder_3_init();

    /* Inizializzazione della MTU2 per generare la PWM sui canali (MTIOC3A, MTIOC3C, MTIO4A, MTIO4C) */
    PWM_Init(1);
    PWM_Init(4); // Set up PWM motor 1 JN2-PIN 22
    PWM_Init(2); // Set up PWM motor 2 J8-PIN 15
    PWM_Init(3); // Set up PWM motor 3 JN2-PIN 23

    /* Inizializzazione delle porte di direzione per motori */
    Init_Port_Dir();

    /* Inizializzazione dei parametri per il controllo PID in coppia */
    init_pid_curr(&pid_curr_1);
    init_pid_curr(&pid_curr_2);
    init_pid_curr(&pid_curr_3);

    /* Inizializzazione I2C e Sensori */
    i2c_init();

    sens_curr_init(&inas_1, 0);     /* Primo Sensore */
    sens_curr_init(&inas_2, 1);     /* Secondo Sensore */
    sens_curr_init(&inas_3, 2);     /* Terzo Sensore */


    /* Funzione di setup dello schermo, CMT, accelerometro, giroscopio, magnetometro e calibrazione magnetometro */
    Setup_MARG(&ahrs);
```

A seguire, nella parte centrale del programma, che corrisponde al ciclo while, si individuano quattro blocchi, i quali corrispondono a quattro timer differenti (10 ms, 20 ms, 50 ms, 100 ms). Per primo, ogni dieci millisecondi vengono invocate le tre funzioni caratteristiche dell'encoder per calcolare posizione e velocità del motore. Poi, ogni venti millisecondi vengono rilevate le correnti dai sensori, eseguendo una conversione e filtrando i dati, tramite cui si calcolano le coppie erogate dai motori e il vettore coppia effettivo applicato dalle omniwheels; successivamente, si prosegue con il calcolo del segnale di controllo da inviare ai motori (modulazione a larghezza di impulso),

utilizzando prima una variabile di appoggio per il PID. Ogni cinquanta millisecondi si processano le informazioni filtrate provenienti dal MARG, mentre ogni cento millisecondi vengono richiamate le funzioni di stampa dell'IMU o degli encoder, così da visualizzare sullo schermo le informazioni che si preferiscono.

```
    while (1)
    {

        if(timers.timer_10mS)
        {

            Query_Enc_1();
            Query_Enc_2();
            Query_Enc_3();

            timers.timer_10mS = 0;

        }

        if(timers.timer_20mS)
        {

            /* Corrente rilevata dai sensori */

            read_curr(&inas_1);
            curr_1 = inas_1.curr/1000;// - 0.015;
            curr_1f = MovingAvgFilter(curr_1, 0);

            read_curr(&inas_2);
            curr_2 = inas_2.curr/1000;// - 0.015;
            curr_2f = MovingAvgFilter(curr_2, 1);

            read_curr(&inas_3);
            curr_3 = inas_3.curr/1000;// - 0.015;
            curr_3f = MovingAvgFilter(curr_3, 2);


            /* Calcolo del segnale di controllo da inviare ai motori */

            tmp1 = calcPID(input, curr_1f, &pid_curr_1);
            v_signal_1 = Rescale(tmp1, 4);
            Volt_to_duty(v_signal_1, 4);

            tmp2 = calcPID(input, curr_2f, &pid_curr_2);
            v_signal_2 = Rescale(tmp2, 2);
            Volt_to_duty(v_signal_2, 2);

            tmp3 = calcPID(input, curr_3f, &pid_curr_3);
            v_signal_3 = Rescale(tmp3, 3);
            Volt_to_duty(v_signal_3, 3);


            t1 = curr_1f*0.64;          // Coppia erogata dal motore 1
            t2 = curr_2f*0.64;          // Coppia erogata dal motore 2
            t3 = curr_3f*0.64;          // Coppia erogata dal motore 3
            TtoV(t1, t2, t3, u);        // Vettore-coppia effettivo applicato dalle ruote

            timers.timer_20mS=0;
        }

        if(timers.timer_50mS)
        {

            /* IMU */
            Read_MARG(&ahrs);
            RealTimeChart(&ahrs);

            timers.timer_50mS=0;

        }

        if(timers.timer_100mS)
        {

             * Frequenza di stampa su schermo dei dati.

            /* Stampa a schermo IMU */

            Print_ABS(&ahrs);
            Print_Angoli(&ahrs);
            Print_VelAng(&ahrs);
            //Print_Temp(&ahrs);

            /* Stampa a schermo Encoder */

            Read_Enc_1();
            Read_Enc_2();
            Read_Enc_3();

            timers.timer_100mS = 0;
        }


        }

}   /* Fine ciclo acquisizione dati e controllo */
```

# *Conclusioni*

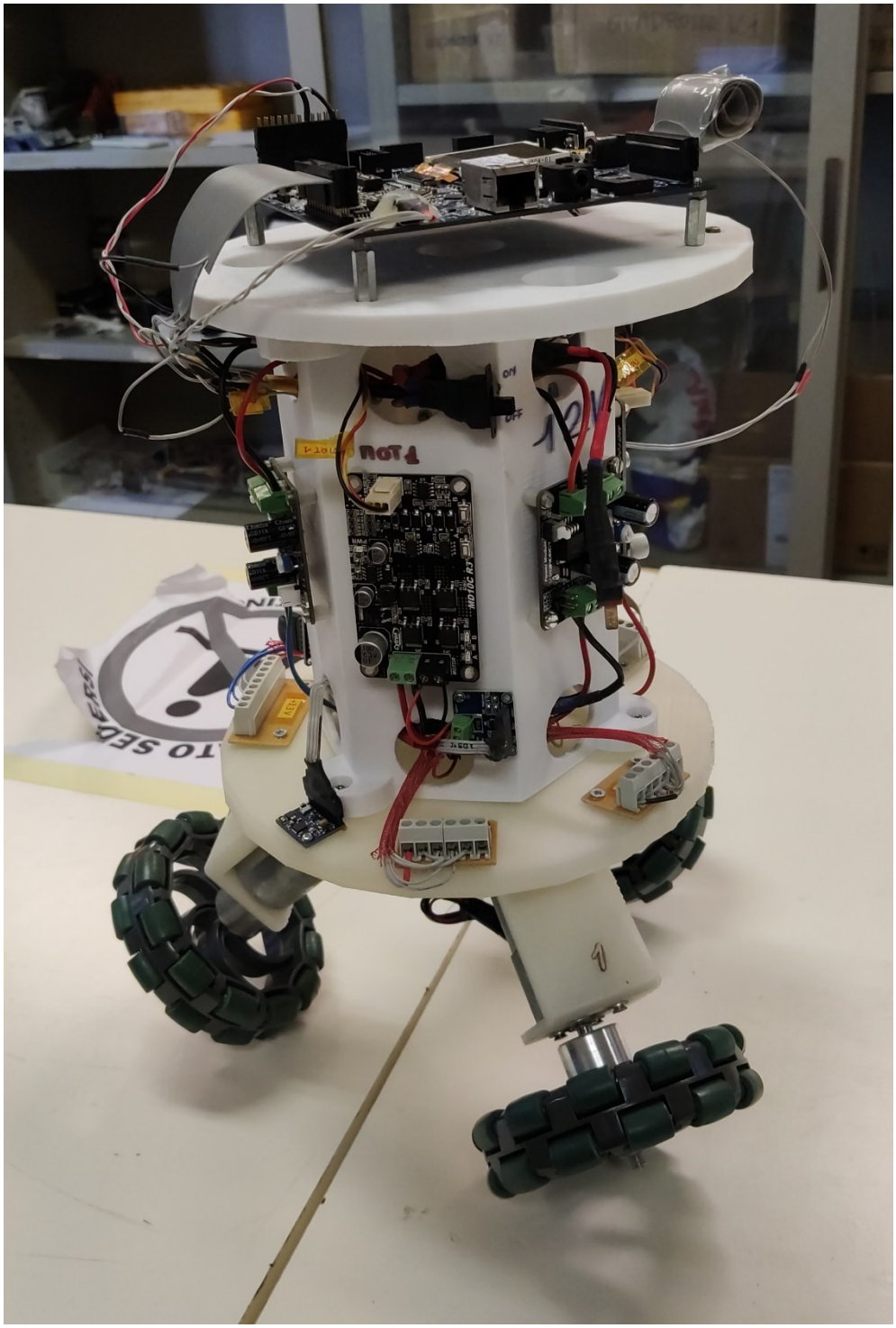
Dal lavoro svolto è stato possibile studiare il modello del sistema e – successivamente – procedere alla programmazione del μ-controller, organizzando le singole componenti del Ballbot coerentemente in base alle specifiche funzioni che esse ricoprono.

Nonostante l'analisi approfondita delle parti principali del robot, verso cui è stata dedicata maggiore attenzione per l'importanza che queste ricoprono nell'architettura generale (motori, sensori, scheda), è stata trascurata la spiegazione dettagliata sia dell'encoder sia del posizionamento dei pin nella scheda Renesas, riservando a tali sezioni solamente una panoramica generale.

Dopo aver assemblato le componenti del Ballbot, sono state eseguite alcuni test per verificarne il corretto funzionamento, le quali hanno portato alla luce alcune problematiche non riscontrabili dalle prove singole eseguite in precedenza. Per primo, poiché i sensori di corrente e l'IMU – che utilizzano il protocollo di comunicazione $I^2C$ – si trovano sul piano inferiore del robot, si registrano dei disturbi sulla linea SCL e sulla linea SDA a causa dell'alimentazione della scheda, dei driver e dei convertitori. Un altro problema che concerne tali sensori si verifica quando la corrente inizia a circolare nei motori, provocando un errore sia nella lettura della corrente sia nell'erogazione del segnale pwm e rendendo di fatto impossibile il controllo. Una possibile soluzione è data dall'utilizzo di sensori di corrente di tipo analogico, in particolare l'ACS712 da 5A di produzione Allegro MicroSystem; la tensione che esce dalla schedina è direttamente proporzionale alla corrente che attraversa il sensore. Poiché i sensori danno in uscita una tensione analogica e la Renesas richiede input in forma digitale, può essere utilizzato il convertitore analogico-digitale a 10 bit presente (Adb10). Tuttavia, nel caso del Ballbot devono essere implementati dei partitori di corrente fra driver e sensori, affinché si abbia una misura corretta da parte del dispositivo. In questo modo è possibile eliminare il problema legato ai sensori che utilizzano il protocollo di comunicazione $I^2C$ e – spostando l'IMU sul piano superiore del robot – si annullano i disturbi causati dal circuito di potenza.

Per quanto riguarda eventuali sviluppi futuri, è necessario procedere con la taratura del PID degli angoli per la stabilizzazione del Ballbot e per mantenerlo nella posizione di equilibrio desiderata. Questa corrisponde al valore nullo del rollio, del beccheggio e dell'imbardata, ovvero i dati filtrati provenienti dall'unità di misura inerziale. Similmente a quanto fatto su Simulink con quello di corrente, in uscita ciascun PID fornirà la legge di controllo per la correzione del valore delle variabili. In questo modo, è possibile chiudere la catena e si può procedere con le prove empiriche sulla sfera per verificarne il funzionamento, considerando anche la sensibilità ai disturbi del sistema. Una volta tarato il PID, è possibile installare un sistema WiFi per il pilotaggio del robot a distanza, così da aumentarne le possibilità di movimento.

Per concludere, grazie alla sua stabilità dinamica – pur limitata a superfici lisce – il Ballbot è ideale per essere un robot mobile personale (in uffici o in casa) fornendo, ad esempio, guide personali. Inoltre, avendo attirato l'attenzione dei media, tale tipo di robot ha una varietà di applicazioni nel settore dell'intrattenimento. Perciò, il Ballbot può trovare impiego anche nel territorio, come in aeroporti o stazioni, per accogliere i turisti, erogando un servizio di qualità.



## *Fonti bibliografiche e sitografia*


Bottoni A., Buccolini G., Elisei N., Luzi L., “Relazione controllo in coppia. Motori Ballbot”, Ancona, 2018

Bonci A., “New dynamic model for a Ballbot system”, published in “MESA” 2016 12th IEEE/ASME International Conference on 29-31, 2016

Giacomobono M., Kaur HD., Nociaro I., Olivieri D., “Sensoristica Ballbot”, Ancona, 2019

Grasselli A., Luciani I., Nkou L.D., Terramani S., “Simulazione e controllo di un sistema ballbot”, Ancona, 2017

Lauwers TB., Kantor GA., Hollis RL. “A Dynamically Stable Single-Wheeled Mobile Robot with Inverse Mouse-Ball Drive”, Paper, Carnegie Mellon University, USA, 2006.

Nagarajan U., Mampetta A., Kantor GA., Hollis RL. “State Transition, Balancing, Station Keeping, and Yaw Control for a Dynamically Stable Single Spherical Wheel Mobile Robot”, Paper, Carnegie Mellon University, USA, 2009.

User’s Manual: Hardware, Renesas 32-Bit MCU RX Family / RX600 Series

INA219 Zerø-Drift, Bidirectional Current With $I^2C$ Interface Datasheet, Texas Instruments
https://www.ti.com/lit/ds/symlink/ina219.pdf?ts=1624518539933&ref_url=https%253A%252F%252Fwww.google.com%252F

Pololu Metal Gearmotor 37Dx73L mm with 64 CPR Encoder,
https://www.pololu.com/product/2827

## *Ringraziamenti*

In primis vorrei ringraziare il prof. Andrea Bonci, che in questi sei mesi di lavoro, ha saputo guidarmi con suggerimenti pratici nelle varie fasi sia del tirocinio sia dell'elaborato. Al dottorando Giuseppe, che con i suoi consigli e con la sua esperienza mi ha spesso dato degli spunti di riflessione; è anche merito suo se ho imparato molto nei mesi in laboratorio.

Ai miei genitori e a mio fratello, che mi hanno sempre sostenuto, appoggiando ogni mia decisione, e che hanno sempre creduto in me, anche nei momenti più difficili. A mia sorella, che nonostante la lontananza, si è sempre dimostrata la più vicina, e che ha sempre rappresentato un punto di riferimento nelle scelte fatte in questi anni.

Al resto della mia famiglia, per avermi incoraggiato senza mai perdere fiducia in me e per avermi fatto capire quanto sono orgogliosi di me.

Ai miei amici, compagni di vita, per aver ascoltato i miei sfoghi; grazie per tutti i momenti di spensieratezza. A chi ho incontrato in Erasmus, per avermi fatto conoscere nuove culture e per avermi fatto crescere come persona, sentendomi fiero di essere sia italiano sia europeo.

Infine, vorrei dedicare questo piccolo traguardo a me stesso, che possa essere l'inizio di un percorso. Grazie.