Relazione
controllo in coppia
Motori Ballbot

A cura di:

Alessandro Bottoni
(spa36@icloud.com)
Giacomo Buccolini
(buccolini.giacomo@gmail.com)
Nicola Elisei
(nico.elisei.96@gmail.com)
Luca Luzi
(lucaluzi@gmail.com)

INDICE:

“Un giorno le macchine riusciranno a risolvere tutti i problemi, ma mai nessuna di esse potrà porne uno.”
-Albert Einstein

Uno dei tre motori

Scheletro Ballbot UNIVPM

Primo Ballbot funzionante - 2005

Cos’è un Ballbot?

Un ballbot è un robot mobile dinamicamente stabile progettato per bilanciarsi su una singola ruota sferica (cioè una palla) attraverso il suo unico punto di contatto con il terreno. Esso è omnidirezionale e quindi eccezionalmente agile e manovrabile rispetto agli altri veicoli terrestri. La sua stabilità dinamica consente una migliore navigabilità in ambienti stretti e affollati.

Il suo funzionamento è basato sul principio di un pendolo invertito.

Renesas:

Per quanto riguarda il controllo e l’elaborazione dei dati, è stata utilizzata un’interfaccia prodotta dalla Renesas, demonstration Kit YRDKRX63N.

YRDKRX63N è la scheda di sviluppo su cui è montato il !-controller RX63N ad alte prestazioni, 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- unità Timers MTU2 [Multi Timer Unit Function], con funzioni di: input capture / output compare / counter clearing per generazione di segnali PWM, e controllo motori.
- watchdog timer Independent e funzione CRC per lo standard di applicazioni domestiche (IEC 60730)
- molte funzioni di comunicazione: Ethernet, SCI, RSPI, CAN, I2C.

Struttura generale di controllo di un Ballbot:

Al fine di controllare attivamente la posizione e l'orientamento del corpo di un ballbot attraverso un quadro sensore-attuatore-computer, accanto a un microprocessore adatto (Renesas demonstration kit nel nostro caso) per eseguire i necessari anelli di controllo, saranno necessari dei sensori che forniranno i dati per l'elaborazione. Attraverso il diagramma a blocchi rappresentato in figura 1.1, è possibile identificare in modo schematico, il funzionamento generale del ballbot.

I tre ingressi rappresentanti gli angoli di Pitch (□ - rotazione rispetto all'asse y), Roll (□ - rotazione rispetto all'asse x) e Yaw (□ - rotazione rispetto all'asse z) andranno sommati alla retroazione, che considereremo in seguito, in modo tale da ottenere i valori di errore per ogni angolo, per poi esser processati dai vari PID di competenza.

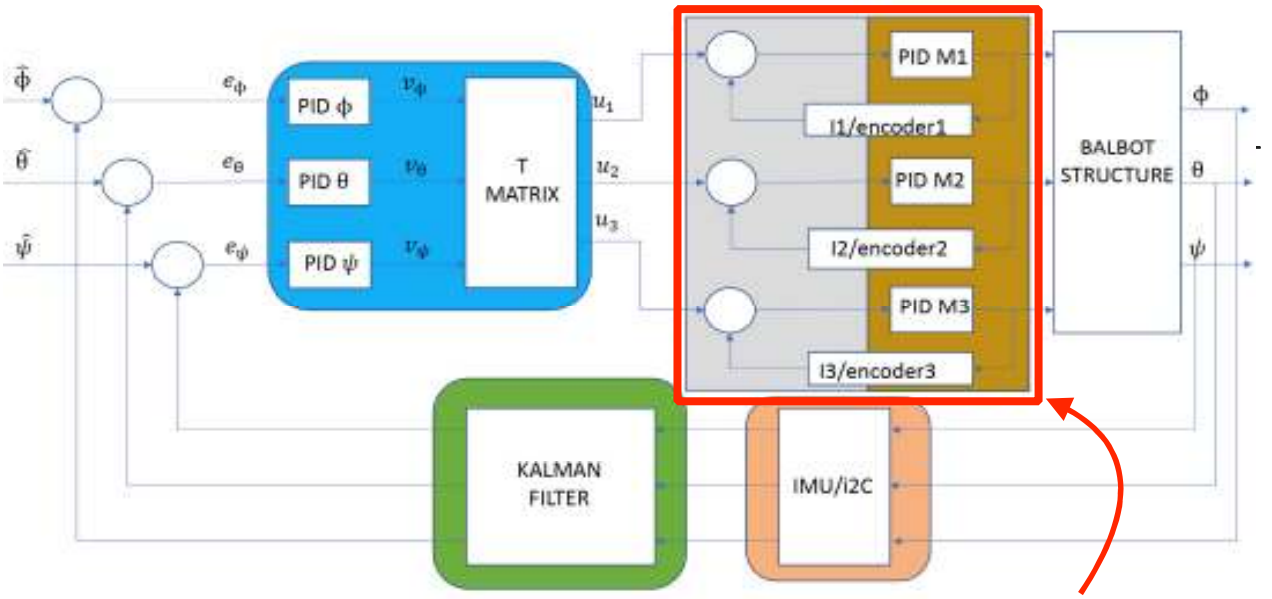


Figura 1.1

Una piccola osservazione riguarda il fatto che gli angoli saranno sempre rappresentati in radianti, piuttosto che in gradi, dato che nel momento in cui andremo ad effettuare più di un giro completo, avremmo modo di convertirli in gradi.

Ogni Pid adibito al controllo di ogni angolo, restituirà in uscita il valore di tensione necessario a compensare lo squilibrio introdotto dal ballbot.

A questo punto attraverso una matrice adibita all'elaborazione dei valori di tensione andremo ad ottenere la corrente di armatura da fornire ai rispettivi PID, tale corrente essendo proporzionale alla coppia può essere considerata come riferimento di coppia. Questo aspetto verrà approfondito esaustivamente in seguito, dato che il nostro contributo nel progetto fa riferimento proprio a ciò.

Successivamente alla nostra elaborazione in coppia, la corrente necessaria alla stabilità del ballbot verrà fornita ai rispettivi motori.

Riprendendo la retroazione precedentemente tralasciata, essa è composta da da IMU/I2C e dal filtro di Kalmann.

In ingresso dell'IMU avremo uno spostamento effettuato con una certa accelerazione e posizione al quale, attraverso il protocollo I2C, verrà

associato un numero che ne rappresenterà l'angolo, il quale, non essendo esente da rumore, verrà passato al filtro di Kalmann attribuendo così alla retroazione un segnale pulito.

Controllo in coppia:

Il processo fondamentale del controllo in coppia consiste nel fornire in INPUT al motore il valore della corrente necessaria al mantenimento in equilibrio del Ballbot.

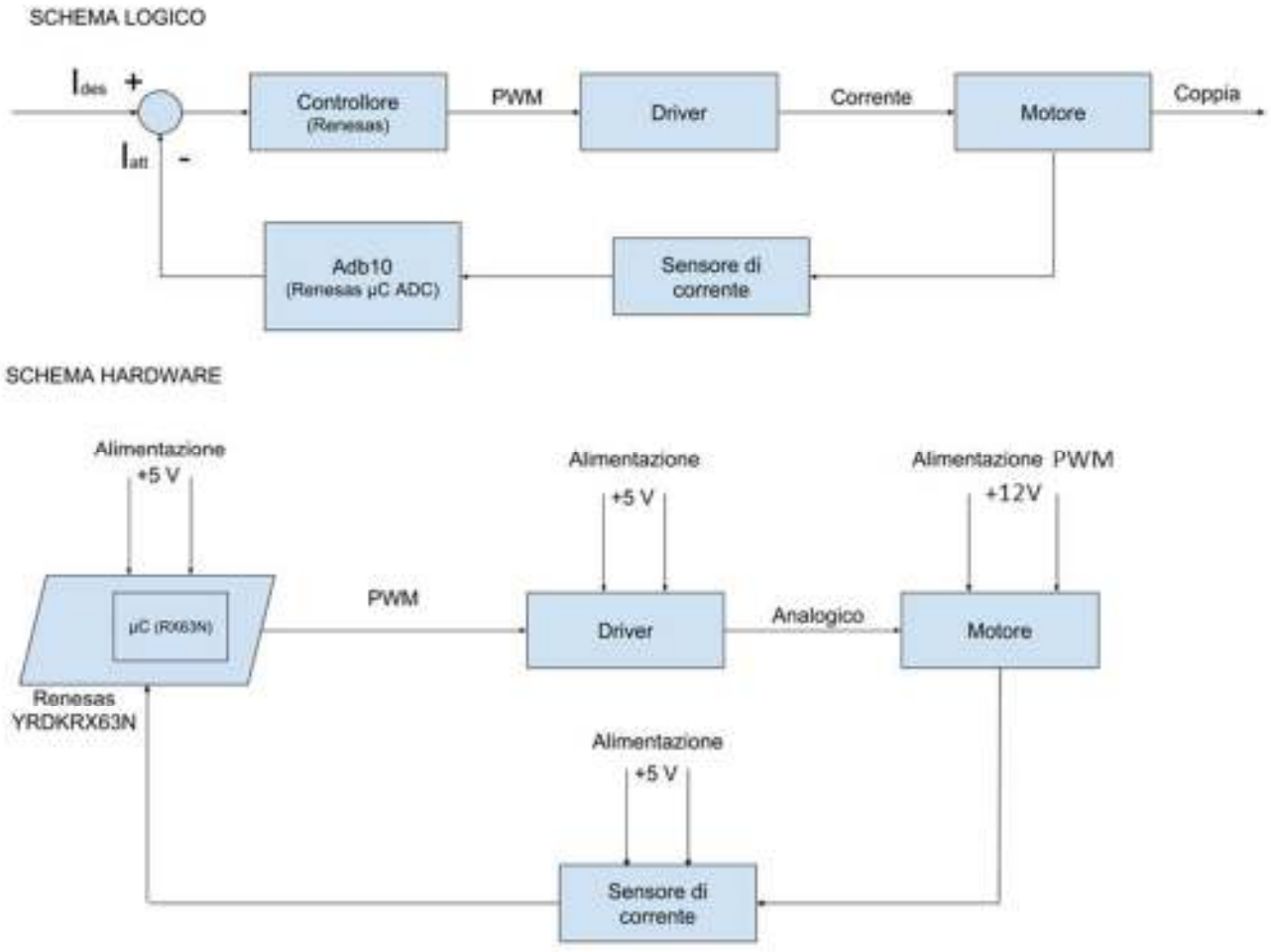


Figura 2.1

Quest'ultimo sarà dato dalla combinazione tra il valore letto dal sensore di corrente (che rappresenta la corrente assorbita dal motore) ed il termine di riferimento, cioè la corrente desiderata data in base all'angolo di sfasamento del robot.

Il controllo in coppia di un motore a collettore richiede che ad ogni ciclo del loop di controllo si conosca il valore di coppia attuale, da poter confrontare con quello desiderato e far sì che il PID in coppia generi un segnale di controllo adeguato.
Non essendo possibile dotare il robot di strumenti atti alla misura della coppia motrice sviluppata dai singoli motori, è necessario ricavarla a partire da altre grandezze, che vengono elaborate dal microcontrollore in maniera tale da fornire il dato richiesto.
Dalle leggi che descrivono le macchine in corrente continua sappiamo che il valore di coppia motrice di un azionamento elettrico è direttamente proporzionale alle correnti che attraversano il circuito di armatura e il circuito di eccitazione, secondo la legge:

$$C\,m(t) = K * Ie(t) * Ia(t)$$

Nel caso dei motori del Ballbot, la corrente di eccitazione $I_e(t)$ e il coefficiente di coppia K sono costanti fissate dal costruttore; pertanto l'unica grandezza modificabile, e che occorre misurare, è la corrente di armatura $I_a(t)$.
Raggruppando $I_e(t)$ e K in K_m, l'equazione della coppia motrice sviluppata può essere riscritta come:

$$C\,m(t) = Km * Ia(t)$$

In accordo con la teoria appena esposta, si è dotato il robot di tre sensori di corrente ad effetto Hall, uno per ciascun motore, in grado di misurare la corrente che li attraversa - ovvero quella di armatura che la batteria fornisce ad ogni azionamento - ed inviare il dato alla scheda.
(Schema di collegamento hardware Figura 3.1)

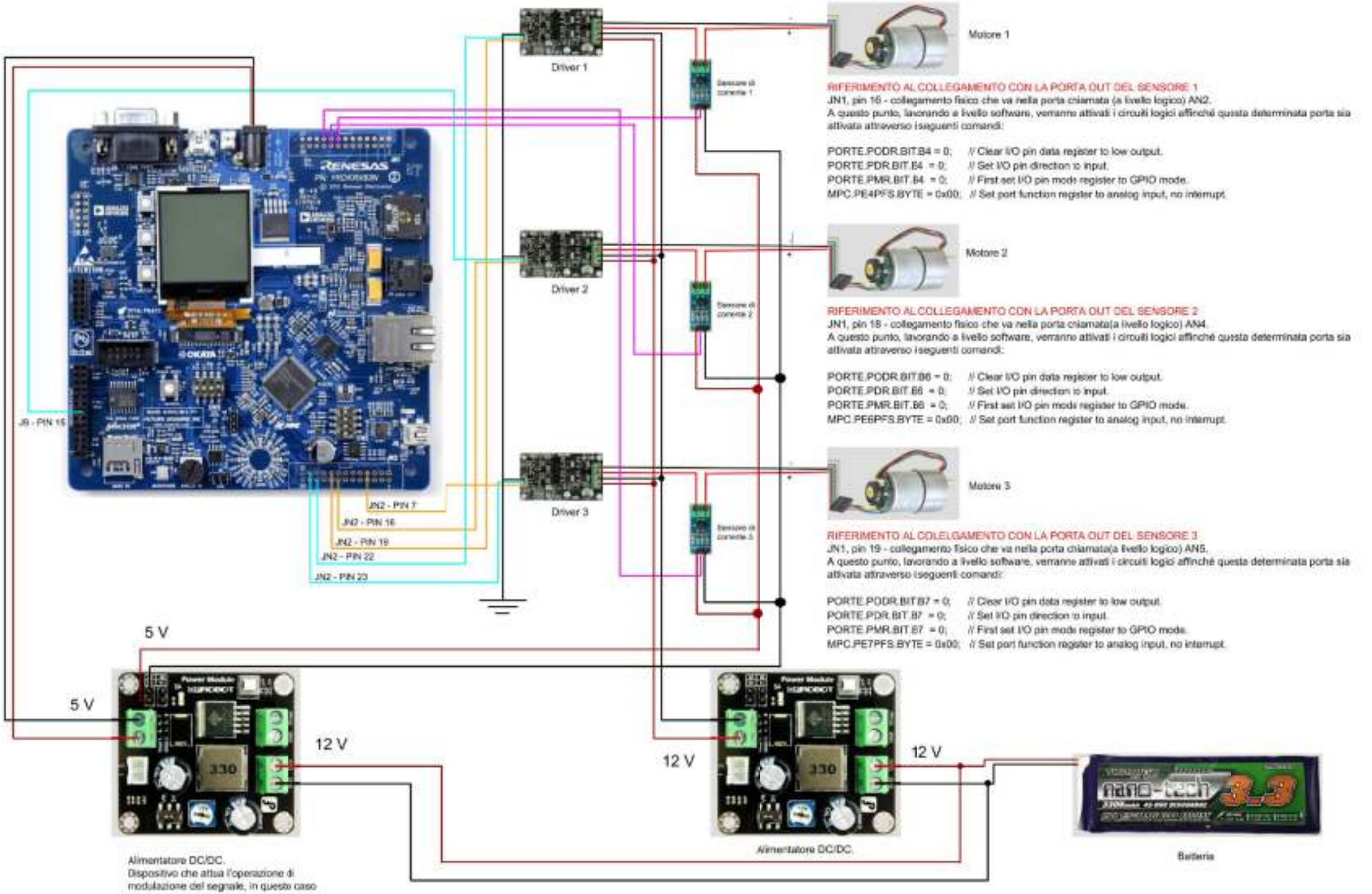

Driver 1
Driver 2
Driver 3
Motore 1
Motore 2
Motore 3
RIFERIMENTO AL COLLEGAMENTO CON LA PORTA OUT DEL SENSORE 1
JN1, pin 16 - collegamento fisico che va nella porta chiamata (a livello logico) AN2.
A questo punto, lavorando a livello software, verranno attivati i circuiti logici affinché questa determinata porta sia attivata attraverso i seguenti comandi:
PORTE.PODR.BIT.B4 = 0; // Clear I/O pin data register to low output.
PORTE.PDR.BIT.B4 = 0; // Set I/O pin direction to input.
PORTE.PMR.BIT.B4 = 0; // First set I/O pin mode register to GPIO mode.
MPC.PE4PFS.BYTE = 0x00; // Set port function register to analog input, no interrupt.
RIFERIMENTO AL COLLEGAMENTO CON LA PORTA OUT DEL SENSORE 2
JN1, pin 18 - collegamento fisico che va nella porta chiamata(a livello logico) AN4.
A questo punto, lavorando a livello software, verranno attivati i circuiti logici affinché questa determinata porta sia attivata attraverso i seguenti comandi:
PORTE.PODR.BIT.B6 = 0; // Clear I/O pin data register to low output.
PORTE.PDR.BIT.B6 = 0; // Set I/O pin direction to input.
PORTE.PMR.BIT.B6 = 0; // First set I/O pin mode register to GPIO mode.
MPC.PE6PFS.BYTE = 0x00; // Set port function register to analog input, no interrupt.
RIFERIMENTO AL COLELGAMENTO CON LA PORTA OUT DEL SENSORE 3
JN1, pin 19 - collegamento fisico che va nella porta chiamata(a livello logico) AN5.
A questo punto, lavorando a livello software, verranno attivati i circuiti logici affinché questa determinata porta sia attivata attraverso i seguenti comandi:
PORTE.PODR.BIT.B7 = 0; // Clear I/O pin data register to low output.
PORTE.PDR.BIT.B7 = 0; // Set I/O pin direction to input.
PORTE.PMR.BIT.B7 = 0; // First set I/O pin mode register to GPIO mode.
MPC.PE7PFS.BYTE = 0x00; // Set port function register to analog input, no interrupt.
J8 - PIN 15
JN2 - PIN 7
JN2 - PIN 16
JN2 - PIN 19
JN2 - PIN 22
JN2 - PIN 23
5 V
5 V
12 V
12 V
12 V
Alimentatore DC/DC.
Dispositivo che attua l'operazione di modulazione del segnale, in questo caso attenua la potenza trasmessa dalla sorgente (batteria) evitando che la Renesas sia alimentata con la stessa tensione richiesta dai motori.
Alimentatore DC/DC
Batteria

Figura 3.1

Sensore di corrente:

Il sensore utilizzato per la realizzazione del Ballbot, è un ACS712 di produzione da parte di Allegro MicroSystems (qui il *datasheet* (link), o vedi allegato cartella dei riferimenti). Di tale integrato esistono 3 versioni a seconda della massima corrente misurabile (5-20-30A); per la realizzazione del ballbot ed anche per le prove in laboratorio è stato utilizzato il modello da 5A.

Sul *datasheet* è riportato lo schema tipico:

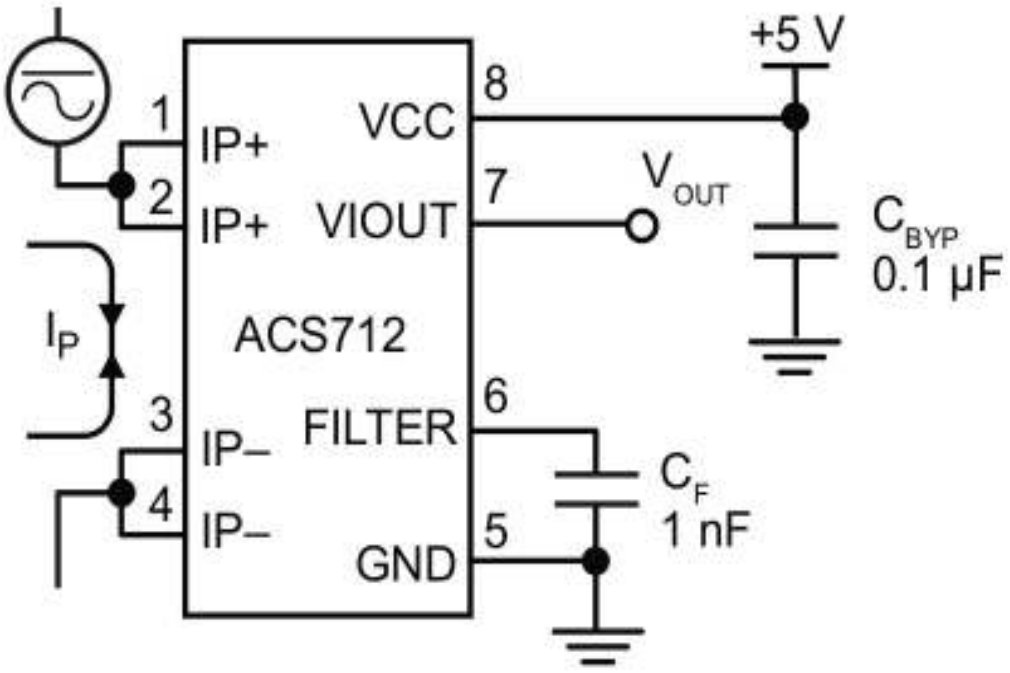


Figura 4.1 The ACS712 outputs an analog signal, V_{OUT}, that varies linearly with the uni- or bi-directional AC or DC primary sampled current, I_P, within the range specified. C_F is recommended for noise management, with values that depend on the application.

Ciascun sensore necessita di un'alimentazione a 5V, fornitagli dagli alimentatori DC/DC, e viene collegato in serie tra il driver del motore ed il motore stesso; dal connettore OUT sulla schedina esce una tensione di valore compreso tra 1,575V e 3,425V direttamente proporzionale all'intensità della corrente che attraversa il sensore. (Vedi schema su sensore di corrente)

L'ACS712 è un sensore bidirezionale, ciò significa che è in grado di misurare correnti in ingresso (A_{IN}) che circolano in entrambi i versi entro un range che va da -5 A a +5 A.

La costante di proporzionalità che lega la corrente A_{IN} alla tensione in uscita dal pin OUT del sensore, viene definita *Sensitivity (S).* Essa assume valore pari a 185 mV/A da costruttore.

È possibile verificare quanto detto, moltiplicando la nostra S per la variazione massima della corrente A_{IN} (10A, -5 A ÷ +5 A), ottenendo così 1,85 V, che è esattamente lo scostamento massimo (1,575 V ÷ 3,425 V) della tensione in uscita dal pin OUT del nostro sensore. Ci denota che per un valore di corrente A_{IN} uguale a 0, avremo una tensione di 2,5 V, o meglio denominata come tensione di Offset (Vo).

Le relazioni tra tensione d'uscita e corrente d'ingresso sono quindi date dalle seguenti espressioni:

1) $V'' out = Vo + S*Ain$

2) $A'' in = (Vout – Vo) / S$

I sensori danno in uscita una tensione analogica, mentre Renesas esige input in formato digitale.

Abbiamo dunque utilizzato il convertitore analogico-digitale a 10 bit già presente nella Renesas (ADb10 - trattato in seguito).

Alternative all'ACS712.

Durante l'esecuzione delle prove, abbiamo notato che la corrente assorbita dai motori non oltrepassava i 2 A, valore nettamente inferiore al range supportato dall'ACS712.
Ciò comporta uno scostamento limitato rispetto al valore di Offset, quindi una misura meno accurata per le correnti trattate.

Un'alternativa è data dal sensore prodotto dalla casa MAXIM, modello MAX471 GY-471 (qui il *datasheet(link)* o vedi allegato nella cartella dei riferimenti).
Infatti, dato il range minore (-3A ÷ +3A), sarà maggiormente adatto alle nostre esigenze.
Non essendo entrati nel dettaglio, si sono presentate le seguenti problematiche con il medesimo:

1) È possibile che l'uscita (in Volt) sia legata con un rapporto 1:1 all'ingresso (in Ampere)?
2) Nel caso fosse possibile, considerando che la variazione è di 6 Ampere, la nostra scala raggiungerà minimo i 6 Volt, non supportati da Renesas. Perciò occorre valutare l'implementazione di un circuito che mi adatti la tensione ad un valore massimo di 3,3 Volt attraverso l'uso di un amplificatore operazionale esterno.

Driver:

Per il controllo dei nostri motori non era sufficiente il semplice controllo ON/OFF, in quanto quest'ultimo può comandare il motore alla massima velocità di rotazione in un verso (interruttore ON), oppure fermarlo (interruttore OFF).

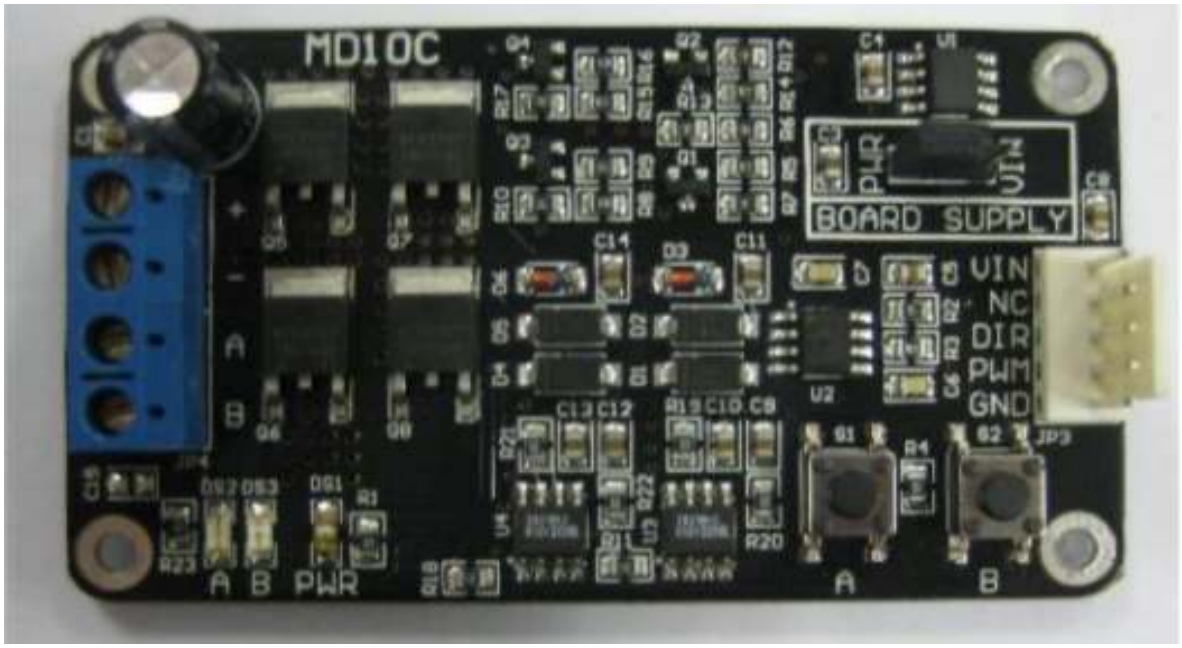


Figura 6.1

Pertanto si è optato per l'uso dell'azionamento Cytron MD10C "Figura 6.1" perché presenta al suo interno un ponte ad H che ci consente di far girare il motore in entrambi i versi di rotazione. In commercio esistono due tipi di ponti H: ponti H discreti, costituiti da componenti separati come transistor e diodi e ponti H integrati, in questo caso il circuito è racchiuso in un package plastico di tipo DIP (dual in-line package) o simile. La nostra scheda utilizza quello discreto. Un altro aspetto di grande rilevanza per cui si è scelta tale apparecchiatura è quello legato alla necessità di controllare il motore in velocità; infatti tale scheda presenta la possibilità di effettuare tale controllo attraverso un segnale PWM (PULSE WIDTH MODULATION), sia in modalità LAP (LOCKED ANTI PHASE) che in modalità SN (SIGN-MAGNITUDE). La tecnica da noi adottata è la SN. Questa consiste nell'inviare il segnale PWM all'ingresso di abilitazione del ponte e nel comandare la direzione di rotazione del motore tramite i due ingressi di controllo del ponte. Per il controllo SN sono necessari due segnali; il primo è un'onda quadra di duty cycle variabile fra 0-100%, che stabilisce la velocità di rotazione (PWM), il secondo è un segnale costante che determina il verso di rotazione (DIR).

N.B. È importante collegare SEMPRE il pin GND a massa per avere un riferimento logico altrimenti la scheda di pilotaggio non sarà in grado di abilitare la bi-direzionalità dei motori.

Caratteristiche principali:

Tensione di ingresso (tensione del motore) 3 – 25 V

Corrente continua massima del motore 10 A

Corrente di picco del motore1 15 A

Tensione di bordo 11 -14 V

Tensione degli ingressi logici – livello alto 3 – 5,5 V

Tensione degli ingressi logici – livello basso 0,5 V

Massima frequenza del PWM 10 kHz

Convertitore DC-DC:

Nell'impianto elettrico del robot sono stati inseriti due convertitori DC-DC Power Module da 25W, uno con uscita da +5V per l'alimentazione della scheda Renesas (microcontrollore), l'altro con una tensione di uscita di +3,3V per l'alimentazione degli encoders. Questo convertitore è in grado di convertire qualsiasi tensione continua tra 3,6-25V ad una tensione selezionabile da 3,3-25V. È possibile scegliere come tensione di uscita +5V

direttamente con l'interruttore a slitta presente nella scheda. Quest'ultima possibilità è stata adottata per l'alimentazione del microcontrollore. Inoltre è possibile scegliere tre diverse interfacce di uscita, ed è anche prevista la possibilità di avere due morsetti ulteriori con la tensione pari a quella di ingresso. Il pulsante ON/OFF sulla scheda può essere utilizzato per accendere o spegnere il convertitore. ATTENZIONE: la tensione di ingresso deve essere superiore alla tensione di uscita.

Caratteristiche principali:

Range tensione di ingresso 3,6-25V dc

Range tensione di uscita 3,3-25V dc

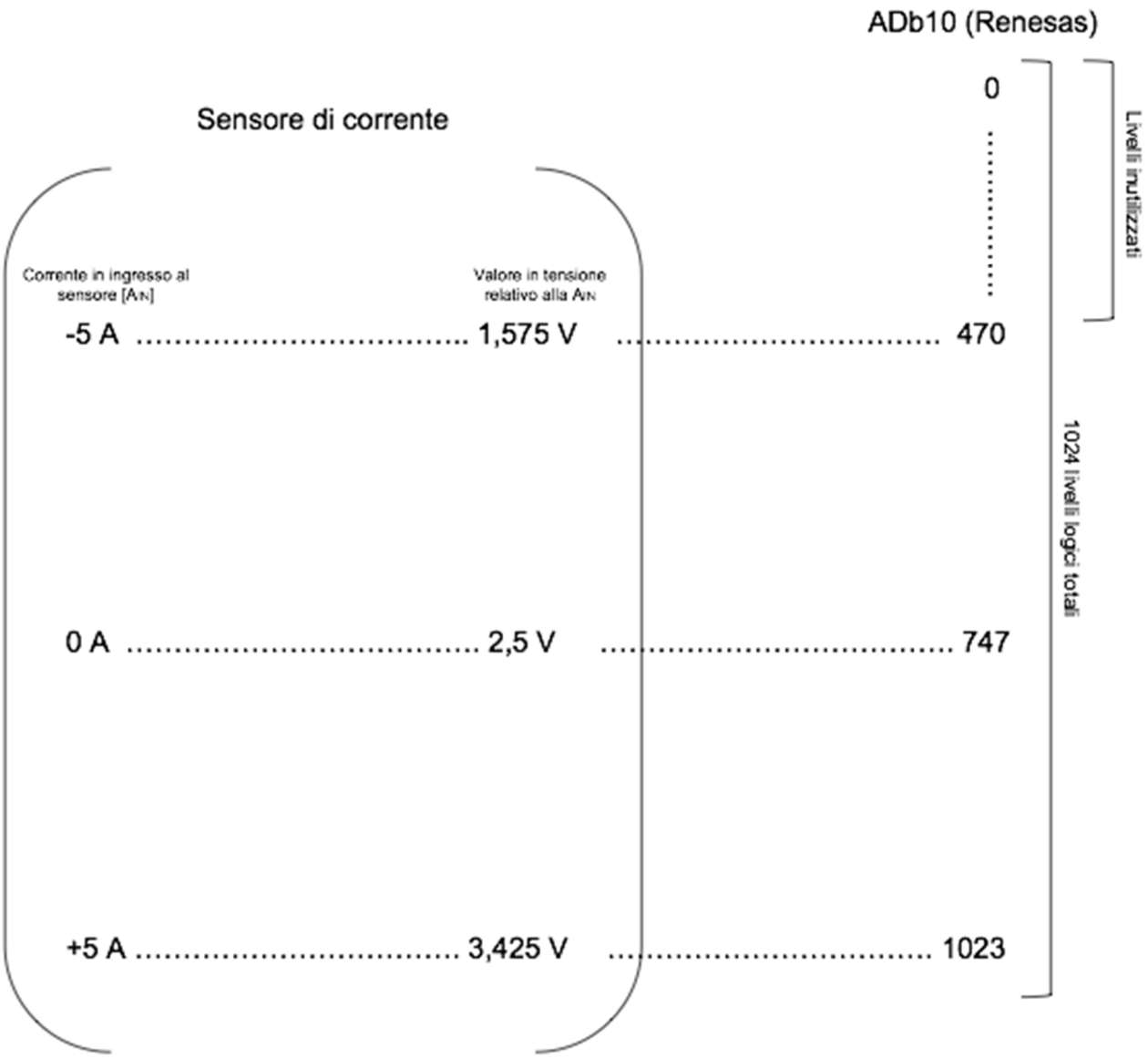
Corrente di uscita 5A a 5V

Potenza massima 25W

Frequenza di commutazione 350 kHz

ADb10:

È un convertitore analogico/digitale a 10bit integrato nella Renesas. Questo significa che possiamo lavorare su 1024 livelli (0 ÷ 1023) di quantizzazione del segnale. Nel nostro caso il convertitore viene utilizzato per la conversione del segnale analogico letto dal sensore di corrente che verrà poi trasformato in un segnale digitale in modo tale da poter essere elaborato dalla Renesas.
La correlazione tra il valore di tensione elaborato dal sensore di corrente ed il livello logico del convertitore è riportato nello "schema 7.1".



schema 7.1

Nella parte di codice relativo al convertitore, è presente un filtraggio FIR (finite impulse response - risposta infinita all'impulso) sui valori ricevuti dal sensore di corrente.

Questa tecnica di filtraggio a "media mobile ponderata", sostituisce ad ogni nuovo valore letto, la media di tutti i valori nella finestra di campionamento, pesati in ordine di lettura; quindi il più recente avrà un peso maggiore rispetto ai valori meno recenti.

La dimensione della finestra nella quale lavora è data dal parametro *“lenght”,* dichiarata nel file ADb10.h; diminuendo quest’ultima grandezza, avremo una risposta con valori che si allontanano maggiormente al valore atteso, degradando le prestazioni.
Il codice relativo al filtraggio è riportato in seguito nella “Figura 9.1”

Per quanto riguarda il nostro caso, la dimensione del parametro “lenght” è stata settata a 1000 dopo aver effettuato diverse prove. Per distinguere il comportamento del filtro in base alla lunghezza della variabile *“lenght”,* **abbiamo simulato il segnale in uscita dal sensore di corrente, generando un rumore bianco con varianza 0,4 e valor medio 0 e graficando successivamente i risultati ottenenuti**, attraverso l’uso della piattaforma Matlab, i casi in cui la finestra mobile avrebbe assunto il valore di 10, 100 e 1000 evidenziando la progressiva linearità dei valori in uscita dal filtro all’aumentare della dimensione del vettore.
In particolare abbiamo utilizzato 10 valori del rumore, riportati di seguito, inserendoli in un vettore che abbiamo replicato per ottenere i 1000 valori necessari per l’esperimento. Successivamente tramite la funzione “filter” di Matlab abbiamo simulato il comportamento del nostro filtro per poi replicare le operazioni software necessarie alla realizzazione della media ponderata. Quindi per verificare e confrontare il funzionamento dei tre diversi casi sopra citati (10,100,1000) abbiamo graficato i risultati ottenuti, i quali si trovano in “Figura 7.2”.

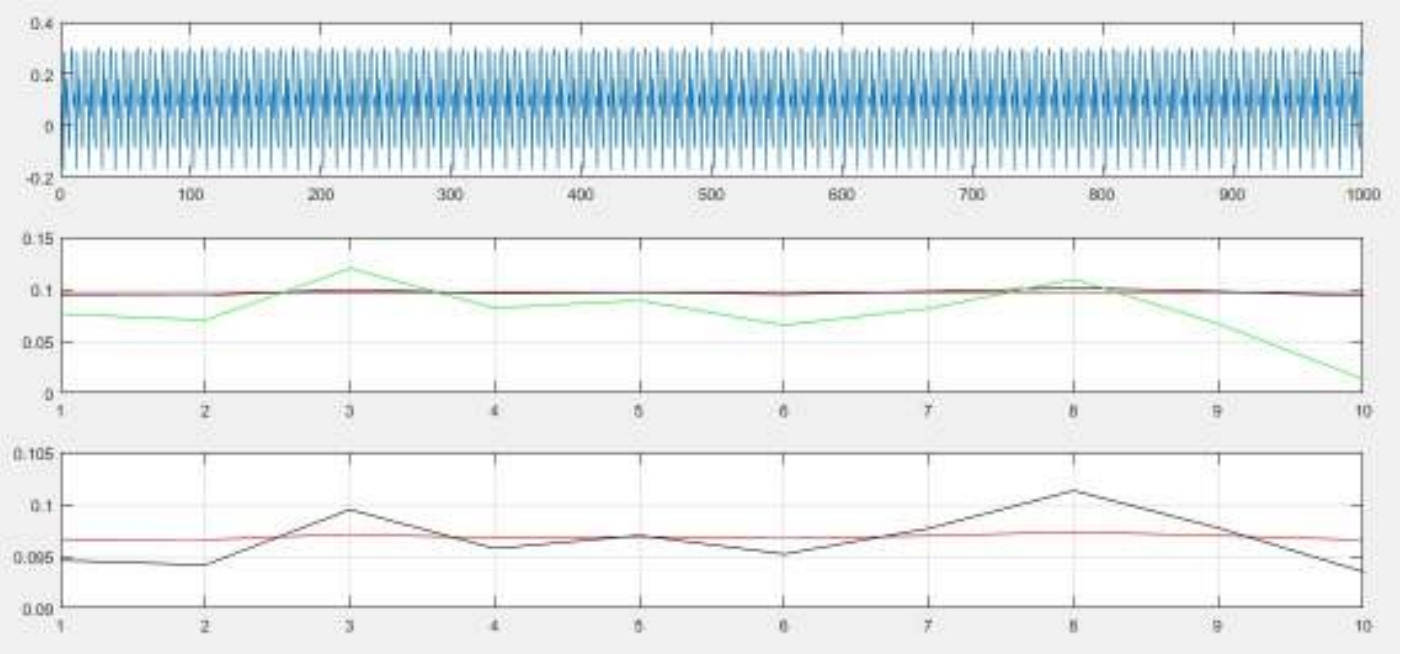


Figura 7.2

Nel primo grafico abbiamo riportato il rumore bianco (in ciano), utilizzato per effettuare il test, ottenuto a partire del seguente vettore di valori:

```
[0.07404 0.30020 0.27254 -0.08968 -0.03248 0.18343 0.03008 0.28408 -0.17169 0.11792]
```

Nel secondo grafico abbiamo messo in relazione i tre risultati ottenuti filtrando il rumore con una finestra di 10 valori (in verde), con una finestra di 100 valori (**in nero**) ed infine con una finestra di 1000 valori (in rosso). Da ciò possiamo notare come il filtraggio su 10 valori non sia adatto per il comportamento del ballbot, in quanto vi sono delle misure molto diverse tra di loro e perciò abbiamo escluso tale soluzione.
Nel terzo ed ultimo grafico, siamo entrati ancora più nel dettaglio per confrontare il filtraggio su 100 valori (**in nero**) ed il filtraggio su 1000 valori (in rosso). Come si può notare dal grafico, il risultato più adatto per l'applicazione al ballbot è quello che si ottiene con 1000 valori, dato che il segnale è più "smooth", ovvero con misure più correlate tra loro, rispetto agli altri due segnali filtrati.

PWM:

Per il controllo dei motori è stata utilizzata la modulazione PWM, la quale, integrata nella scheda Renesas, consente di fornire un'adeguata alimentazione a ciascuno dei 3 motori.
Questa tecnica prevede la parzializzazione della tensione costante data dalle batterie, impostando un valore opportuno del DutyCycle. Le funzioni adibite al corretto utilizzo della modulazione PWM sono le seguenti:
"PWM_Init" è la funzione che inizializza i registri della MTU2 per consentire la generazione del segnale PWM.
"Volt_to_duty" converte il valore di tensione rappresentante lo sforzo di controllo medio (parametro 'control_output') in un valore numerico per il registro di comparazione TGR del canale opportuno ('channel').
"Dutycycle" effettua la conversione del dutycycle (parametro 'duty_cycle') in un valore numerico per il registro di comparazione TGR ('channel').
"Init_port_dir" è la funzione che inizializza le porte di direzione per il pilotaggio dei motori in maniera bidirezionale.
"Chance_motor_dir" è la funzione che cambia la direzione di ciascun motore ('num_motor') attraverso un parametro opportuno ('direction').
"Duty_cycle_to_motor" calcola il valore percentuale del dutycycle per poi inoltrare tale valore al relativo motore mediante la funzione "Dutycycle" infine determina il verso di rotazione attraverso la funzione "change_motor_dir".

La funzione "Motor_direction" imposta il verso di rotazione del motore attraverso un'adeguata logica di controllo che prevede di inviare unicamente valori positivi (assoluti) indipendentemente dal verso di rotazione richiesto.

Schema generale logico/fisico:

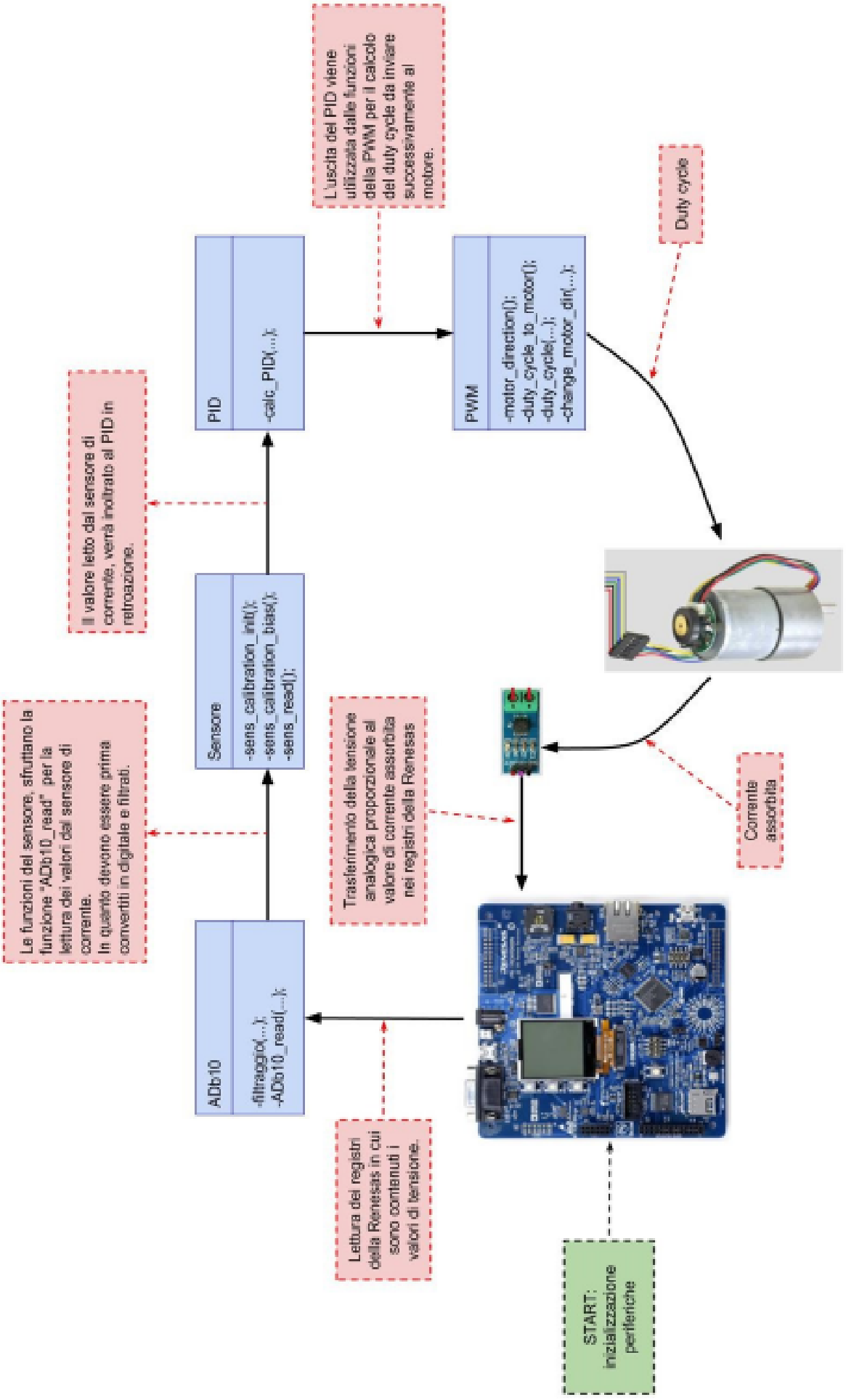


Figura 8.0

Codici principali con flowchart.

- Flowchart sensore di corrente/ADb10:

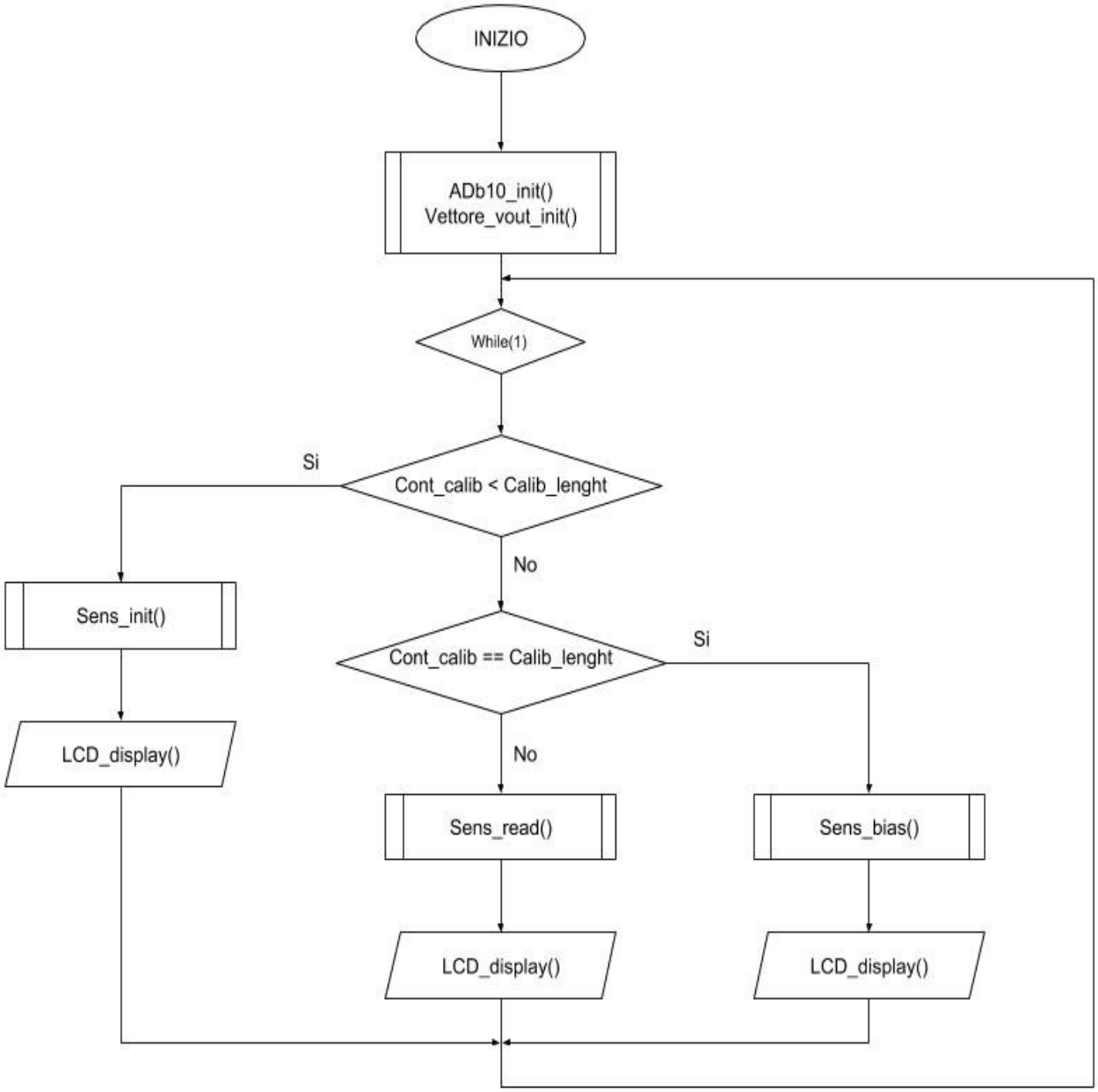


Figura 8.1

Relativo codice ADb10.c :

```
float filtraggio (float read,int contatore,float vettore_vout[])
    {
        int i=0;
        float somma = 0;
        float vout;
        if(contatore<length) //Ci dice se abbiamo raggiunto il limite lenght di valori
                    {
                        vettore_vout[contatore] = read; /*Nella variabile read è contenuto i valori
                                                          letti nei vari registri dal sensore di
                                                          corrente opportuno */
                        for(i=0; i<=contatore; i++)
                        {
                            somma += vettore_vout[i]*(i+1); //somma pesata di tutti i valori letti
                        }

                        vout =(float) somma / ((contatore+1)*(contatore+2)/2); /*assegniamo a vout la media calcolata ponderata */
                    }
                    else // Eseguiamo lo shift, dato che abbiamo già esaminato lenght valori
                    {
                        for(i=1; i<length; i++)
                        {
                            vettore_vout[i-1]=vettore_vout[i]; /*Scambia ultimo valore letto
                                                                 con il primo elemento del
                                                                 vettore */
                            somma += vettore_vout[i-1]*i;
                        }
                        vettore_vout[length-1] = read;
                        somma += vettore_vout[length-1]*(i+1);
                        vout = (float)somma/denom_tot;  //Guardare .h per spiegazione denom tot
                    }
                    return vout;
    }
```

Figura 9.1

Per implementare la funzione di filtraggio precedentemente introdotta, innanzitutto è stato implementato un primo IF per distinguere la fase di riempimento del vettore_vout da quella di “scorrimento” della finestra mobile grazie al confronto tra variabile contatore e “lenght” (definito 1000 su ADb10.h).

Nel primo caso, cioè quando la variabile contatore sarà minore della variabile “lenght”, verrà letto il valore contenuto nel registro opportuno attraverso “read”, per poi eseguire la somma pesata utilizzando il “FOR” ed infine sarà assegnato alla variabile “vout”, il risultato della media calcolata ponderata.

Nel secondo caso invece, quando contatore sarà maggiore di lenght, il vettore “vettore_vout” eseguirà uno “shift” grazie al ciclo “FOR”, tramite il quale il valore nella posizione i-esima verrà shiftato con quello nella posizione precedente i-esima - 1.

In questo modo, il valore meno recente all'interno del vettore verrà scartato, mentre il più recente prenderà la posizione "lenght - 1" del vettore "vettore_vout".

In questo caso, il denominatore per il calcolo della media ponderata, assumerà un valore preciso, dichiarato nel ADb10.h, chiamato denom_tot (*denomtot* = (*Lenght**(*Lenght* + 1))/2).

```
/* Function name : ADb10_read[]
float ADb10_read (int sens_channel)
{
    float read_register, vout_1, vout_2, vout_3; /*read_register è la variabile nella quale sono inseriti i valori
                                                   di tensione letti dai registri*/

    switch(sens_channel) //sens_channel è un intero che assume i valori 1, 2 o 3 a seconda del sensore di corrente considerato
    {
    // AD.ADDRC/E/F sono i registri dove vengono "salvati" i valori letti dal sensore di corrente
case 1:
            read_register = ((AD.ADDRC*VMAX)/MAX_COUNTS); /*legge il risultato del registro per la porta AN2
                                                            che corrisponde al pin 16 della 3N1*/
            vout_1 = filtraggio(read_register, contatore_1, vettore_vout_1);
            contatore_1++;
            return vout_1;
            break;
case 2:
            read_register = ((AD.ADDRE*VMAX)/MAX_COUNTS); /*legge il risultato del registro per la porta AN4
                                                            che corrisponde al pin 18 della 3N1*/
            vout_2 = filtraggio(read_register, contatore_2, vettore_vout_2);
            contatore_2++;
            return vout_2;
            break;
case 3:
            read_register = ((AD.ADDRF*VMAX)/MAX_COUNTS); /*legge il risultato del registro per la porta AN5
                                                            che corrisponde al pin 19 della 3N1*/
            vout_3 = filtraggio(read_register, contatore_3, vettore_vout_3);
            contatore_3++;
            return vout_3;
            break;

default: return 0;
            break;
        }
} /* End of function ADb10_read() */
```

Figura 9.2

La funzione "ADb10_read" è la responsabile della lettura dei valori di tensione nei vari registri.

Lo "switch" seleziona il canale n (n = {1, 2, 3}), in modo tale che verrà aggiornato il valore di "vout_n" eseguendo la funzione "filtraggio" in funzione del valore letto nel rispettivo registro.

- Flowchart pwm:

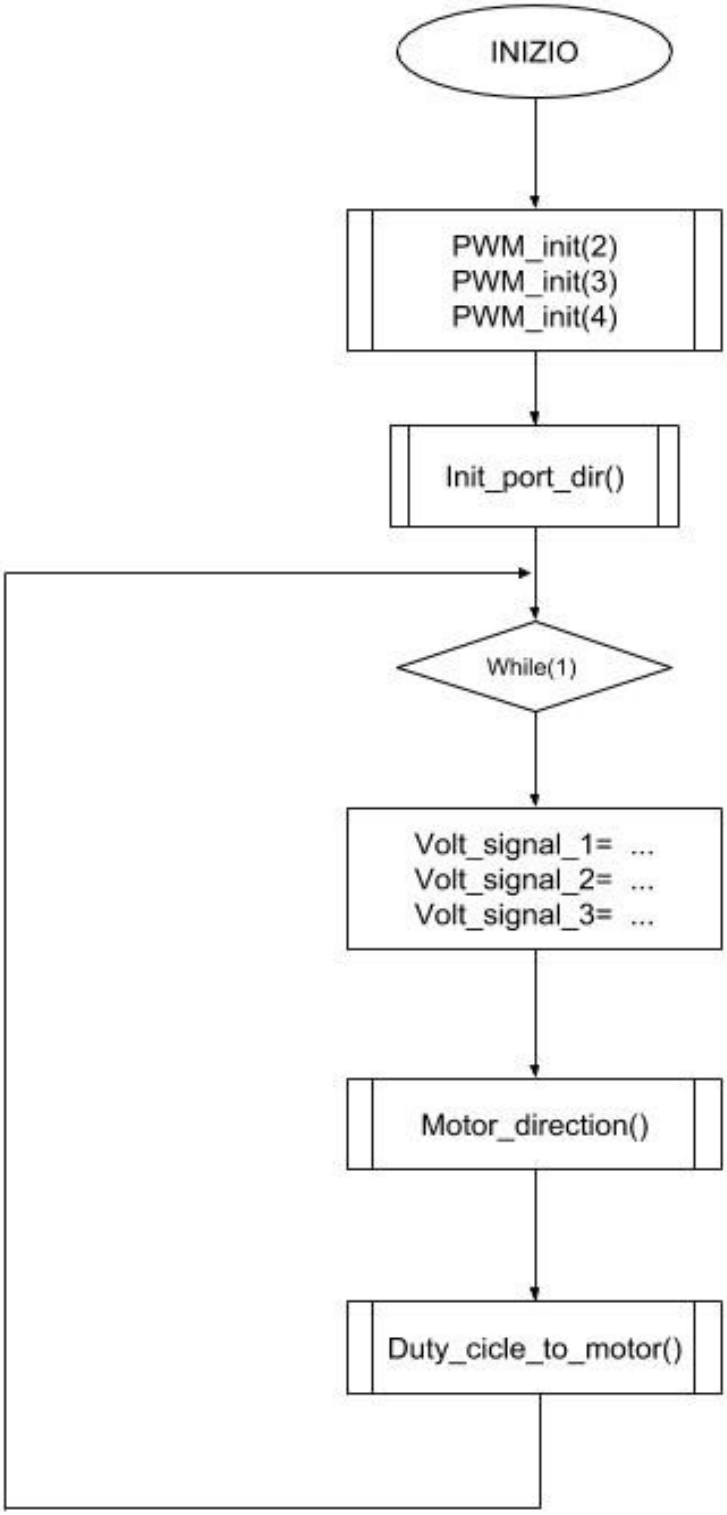


Figura 10.1

Relativo codice pwm.c :

```
493         * Function name: motor_direction
498         void motor_direction(void)
499           {
500               // Controllo del verso di rotazione del motore 1
501               // Se il valore della tensione è negativo viene reso positivo e impostato un flag a 1 per segnalare
502               // la rotazione antioraria del motore
503               if (volt_signal_1 < 0)
504               {
505                   volt_signal_1 = volt_signal_1*(-1.0);
506                   val1 = 1;
507               }
508               else val1 = 0;
509
510               // Controllo del verso di rotazione del motore 2
511               if (volt_signal_2 < 0)
512               {
513                   volt_signal_2 = volt_signal_2*(-1.0);
514                   val2 = 1;
515               }
516               else val2 = 0;
517
518               // Controllo del verso di rotazione del motore 3
519               if (volt_signal_3 < 0)
520               {
521                   volt_signal_3 = volt_signal_3*(-1.0);
522                   val3 = 1;
523               }
524               else val3 = 0;
525           }
```

Figura 10.2

La funzione “motor_direction” in questione, è la responsabile del controllo rotazione dei vari motori.

Essa infatti, setterà ad 1 la variabile “valn” (con n = {1, 2, 3} a seconda del motore selezionato) nel caso in cui “volt_signal_n” risulterà minore di 0. Altrimenti se dovesse risultare positiva, la variabile “valn” risulterà uguale a 0.

```
436   // Funzione per il cambio di direzione dei motori
437   void change_motor_dir(int num_motor, int direction)
438   {
439       switch(num_motor)
440       {
441           case(1):
442           PORTD.PODR.BIT.B6 = direction; // DIR Port Motor 1 - PD6 (Connector JN2 - PIN 19)
443           break;
444
445           case(2):
446           PORTD.PODR.BIT.B3 = direction; // DIR Port Motor 2 - PD3 (Connector JN2 - PIN 16)
447           break;
448
449           case(3):
450           PORTA.PODR.BIT.B7 = direction; // DIR Port Motor 3 - PA7 (Connector JN2 - PIN 7)
451           break;
452       }
453   }
```

Figura 10.3

Alla precedente funzione di “Figura 10.3” verranno passate le variabili “num_motor”, cioè sempre 1, 2 o 3 e la variabile “direction”, precedentemente settata come “valn” nella funzione “motor_direction” (0 = verso orario , 1 = verso antiorario).

```
456   * Function name: DutyCycle_to_Motor
461   void DutyCycle_to_Motor(void)
462   {
463       // Calcolo del Duty-Cycle per il motore 1
464       duty1 = ((volt_signal_1/Max_Volt)*100.0);   // Max Voltage : 100% = Read Current : x
465
466       if (duty1 > Max_Duty)
467           duty1 = Max_Duty;
468       DutyCycle(duty1, 4);    //Il duty1(in percentuale) viene inoltrato nel canale 4 ovvero verso il motore 1
469
470       change_motor_dir(1, val1);  // Determinazione del verso di rotazione del motore
471                                   // (0: verso orario, 1: verso antiorario)
472
473       // Calcolo del Duty-Cycle per il motore 2
474       duty2 = ((volt_signal_2/Max_Volt)*100.0);
475       if (duty2 > Max_Duty)
476           duty2 = Max_Duty;
477       DutyCycle(duty2, 2); //Il duty2(in percentuale) viene inoltrato nel canale 2 ovvero verso il motore 2
478
479       change_motor_dir(2, val2);  // Determinazione del verso di rotazione del motore
480                                   // (0: verso orario, 1: verso antiorario)
481
482       // Calcolo del Duty-Cycle per il motore 3
483       duty3 = ((volt_signal_3)/Max_Volt)*100.0;
484       if (duty3 > Max_Duty)
485           duty3 = Max_Duty;
486       DutyCycle(duty3, 3); //Il duty3(in percentuale) viene inoltrato nel canale 3 ovvero verso il motore 3
487
488       change_motor_dir(3, val3);  // Determinazione del verso di rotazione del motore
489                                   // (0: verso orario, 1: verso antiorario)
490   }
```

Figura 10.4

La funzione “DutyCycle_to_motor” è la responsabile del calcolo (sempre per ogni motore) del valore in percentuale del duty cycle per modulare PWM la tensione di ogni rispettivo motore, salvato nella variabile “dutyn”.

```
/*****************************************************************************************************
*                                    PWM SIGNAL GENERATION
*
* Description   : Converte il duty cycle in un valore numerico per il registro di comparazione TGR,
*
* Arguments     : 'duty_cycle' in percentuale (ad esempio 65.5) e numero del canale 'channel' 1,2,3,4
*
* Returns       : None
*****************************************************************************************************
*/

void DutyCycle (float duty_cycle, unsigned char channel) /* Duty in percentuale, ad esempio 65.5 */
{
 unsigned short int tgr_reg; /* Valore per il registro TGR-1-3-4 (B o D) */

  if(channel == 1 || channel == 2)
    {
     /* Puo' essere zero in relazione al valore di TGRA_3_VAL e di duty_cycle */
     tgr_reg = (unsigned short)(((float)TGRA_3_VAL * (float)duty_cycle) / (float)100);
    }

  else if(channel == 3 || channel == 4)
             {
              /* Puo' essere zero in relazione al valore di TGRA_4_VAL e di duty_cycle */
              tgr_reg = (unsigned short)(((float)TGRA_4_VAL * (float)duty_cycle) / (float)100);
             }

    if(tgr_reg == 0 || duty_cycle <= 0)  /* min 0% */
        tgr_reg = 0xFFFF;  /* max value for TGR */

    else  if (duty_cycle >= 100)/* max 100% */
            tgr_reg = 0x0000;  /* min value for 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)
    {
     /* imposta il nuovo valore del duty cycle*/
     MTU3.TGRB = tgr_reg;  /* new TGRB_3 => change Duty Cycle */
    }
    else if(channel == 2)
            {
             /* imposta il nuovo valore del duty cycle */
             MTU3.TGRD = tgr_reg;  /* new TGRD_3 => change Duty Cycle */
            }
            else if(channel == 3)
                    {
                     /* 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 */
                            }
}
```

Figura 10.5

La funzione “DutyCycle”, in sintesi, converte il duty cycle in percentuale in un valore numerico che verrà inserito nel registro di competenza di ogni motore, a seconda del canale della PWM analizzato. “tgr_reg” è una variabile di appoggio utilizzata per effettuare i calcoli numerici.
Infine, a seconda del canale selezionato *(channel)*, verranno impostati i nuovi valori di duty cycle nei vari registri (MTU3.TGRB/ MTU3.TGRD/MTU4.TGRB/MTU4.TGRD).

Relativo codice sensore.c (Per Flowchart fare riferimento alla figura 8.1):

```
28          * Function name: sens_calibration_init
33          void sens_calibration_init (void)
34            {
35              if(Gestisci_ADb10() == 1)
36              {
37                  vout1 = ADb10_read(1);
38                  vout2 = ADb10_read(2);
39                  vout3 = ADb10_read(3);
40
41                  sum_vout1 += vout1;
42                  sum_vout2 += vout2;
43                  sum_vout3 += vout3;
44
45                  cont_calibration++;
46              }
47            } /* Fine della funzione calibration_init */
48
50          * Function name: sens_calibration_bias
55          void sens_calibration_bias (void)
56            {
57                bias1 = (sum_vout1/calibration_length) - offset_vout;
58                bias2 = (sum_vout2/calibration_length) - offset_vout;
59                bias3 = (sum_vout3/calibration_length) - offset_vout;
60
61                cont_calibration++;
62            } /* Fine della funzione calibration_bias */
```

Figura 11.1

Le prime due funzioni contenute nel “sensore.c” sono *“sens_calibration_init”* e *“sens_calibration_bias”,* che, rispettivamente, serviranno per inizializzare il vettore che andremo ad utilizzare nel filtraggio, e per calcolare il valore del bias, cioè un disturbo sistematico che altera il valore di riferimento del sensore, per ogni sensore.

```
65          * Function name: sens_read
71          void sens_read (void)
72          {
73              if (Gestisci_ADb10() == 1)
74              {
75                  // Lettura dei valori di tensione legati alla corrente inviata ai motori
76                  vout1 = ADb10_read(1) - bias1;
77                  vout2 = ADb10_read(2) - bias2;
78                  vout3 = ADb10_read(3) - bias3;
79
80                  // Conversione dei valori di tensione letti dal sensore in valori di corrente
81                  sens_curr_1 = (vout1 - offset_vout)/sensitivity;
82                  sens_curr_2 = (vout2 - offset_vout)/sensitivity;
83                  sens_curr_3 = (vout3 - offset_vout)/sensitivity;
84              }
85          } /* Fine della funzione sens_read */
86
87          /* End of file sensore.c */
```

Figura 11.2

La funzione *“sens_read”* converte i valori di riferimento di tensione, calcolati mediante *“ADb10_read - bias”* in valori di riferimento in corrente, facendo il rapporto tra la differenza del valori di tensione precedentemente calcolato e l’offset del sensore (ricavato da datasheet) e la sensitivity.

Il valore di *“sensitivity”*, anch’esso dato da datasheet (0,185 V/A), ci indica quanto il segnale di output varia, in conseguenza alla variazione del segnale di input. (Da non confondere con la sensibilità, che invece rappresenta il valore minimo di corrente che il sensore è in grado di leggere.)

Prove effettuate in laboratorio:

1. Prova con oscilloscopio tensione in uscita dal sensore di corrente:

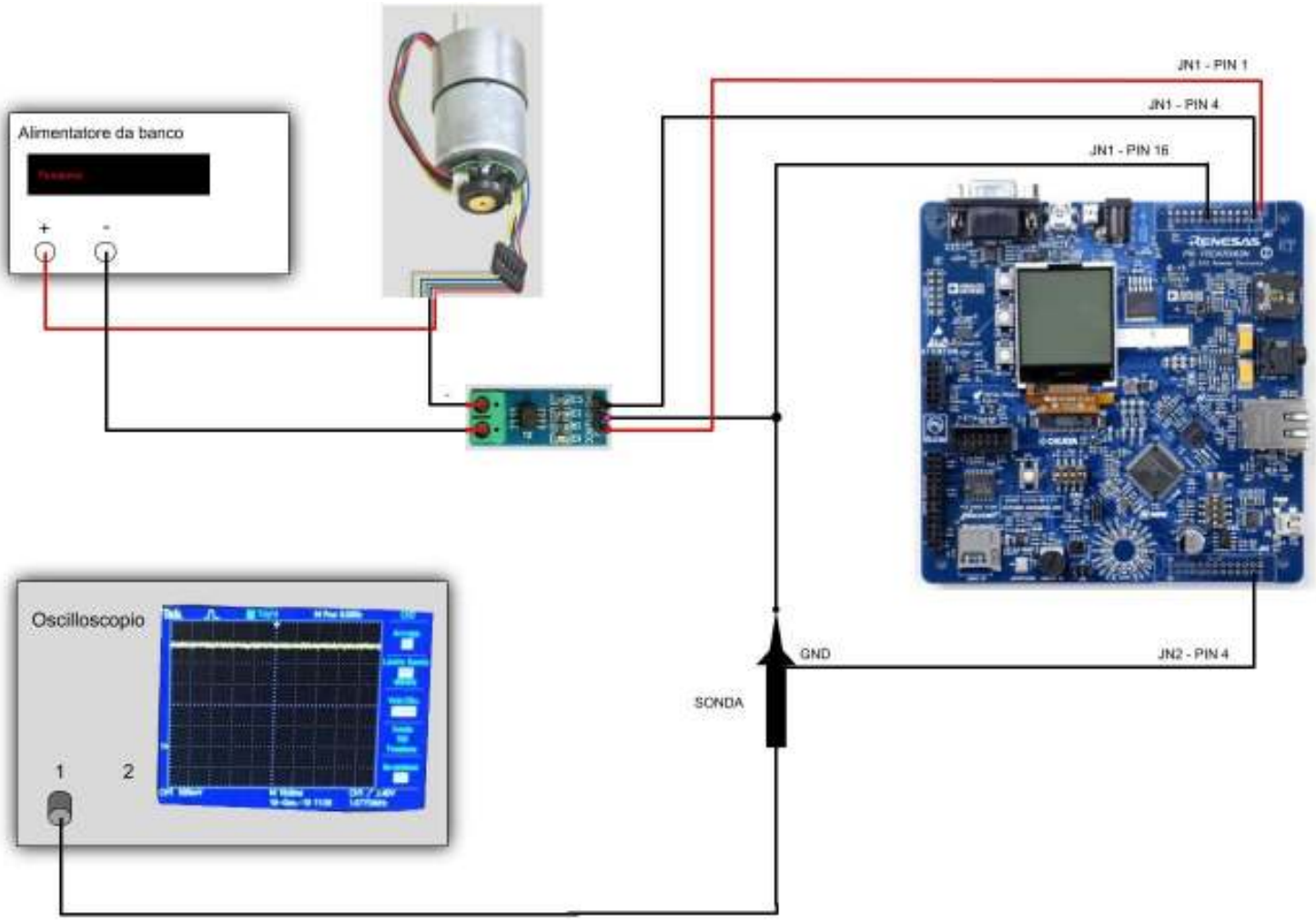


Figura 12.1

Una delle prove che abbiamo effettuato, consisteva nel verificare che il valore di tensione in uscita dal sensore di corrente fosse coerente con quanto ci aspettavamo, dato il valore di corrente in ingresso e le assunzioni fatte riguardo la proporzionalità ingresso/uscita. Per fare ci abbiamo utilizzato un oscilloscopio posizionandoci sul pin out del sensore di corrente attraverso una sonda visualizzando a schermo il valore atteso. (Schema collegamenti riferimento Figura 12.1)

2. Verifica tempo impiegato dalla Renesas per il completamento di un ciclo:

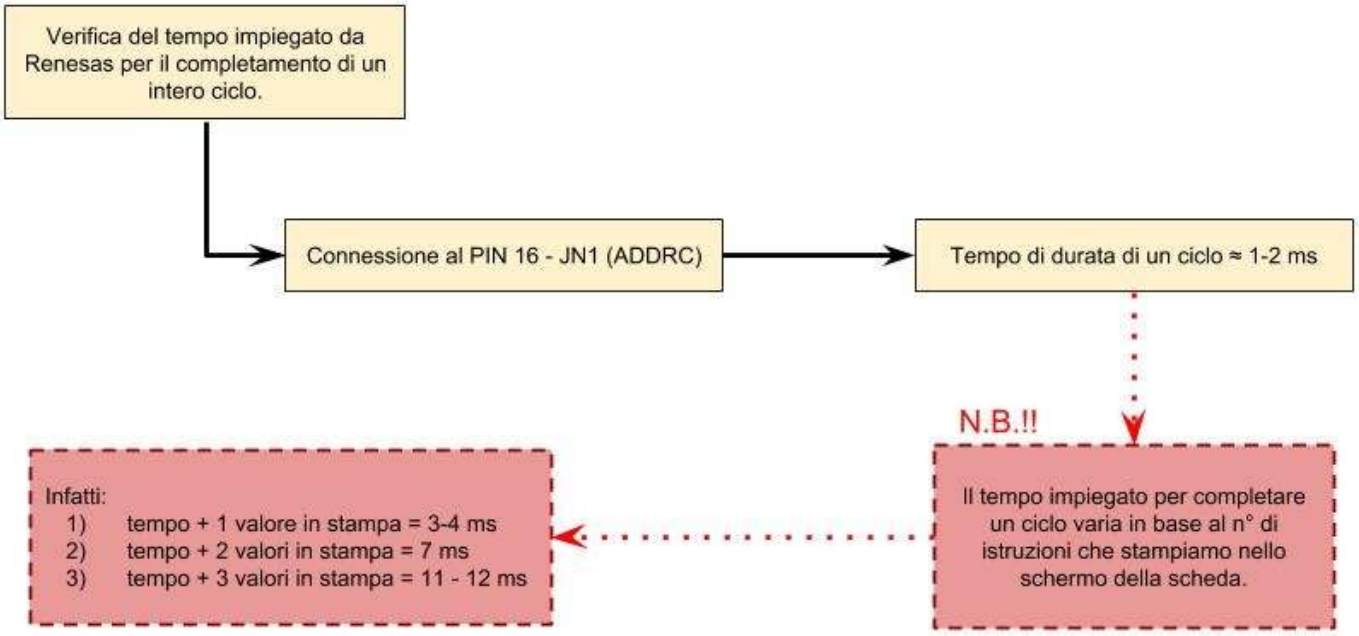


Figura 13.1

Per completare un intero ciclo, comprensivo quindi anche della funzione filtraggio, Renesas impiega un tempo ? 2 ms.
Come rappresentato in figura 13.1, la durata del ciclo aumenterà in base alle stampe a video che andremo ad effettuare.

3. Verifica funzionamento con ingresso costante ed ingresso dato da vettore di valori.

Per provare il corretto funzionamento del codice, sono state effettuate due prove: la prima attribuendo alla variabile *"PID_P_curr"* un valore costante, mentre la seconda, attribuendo sempre alla medesima variabile, un ingresso variabile nel tempo grazie all'utilizzo di un vettore e di un timer di 2ms.

```
70          //valori per la prova espressi in mA
71          PID_P_curr_1.uc=-100;
72          PID_P_curr_2.uc=200;
73          PID_P_curr_3.uc=150;
```

Figura 14.1

In figura 14.1 è possibile notare assegnazione di valori costante.

```
26          //vettore prova sensore
27          const int vettP[10]={10, 20 , 35, 20, 5, -10, -15, -20, -5, 15};
```

Figura 14.2

In figura 14.2 invece, notiamo l'assegnazione del vettore di prova. Per mezzo del timer, andremo a "scorrere" i valori del vettore in modo da "simulare" in maniera più simile alla realtà il funzionamento del ballbot.

```
90          // E' stato implementato un timer da 2ms per l'acquisizione della corrente assorbita dai motori
91          //timer_init=general_timer_mS;
92          if(timer_2mS!=oldtimer){
93              oldtimer = timer_2mS;
94              if (++timerCont>3){
95                  timerCont=0;
96                  if(++ptvettP>9)
97                      ptvettP=0;
98                  PID_P_curr_1.uc=vettP[ptvettP];
99                  PID_P_curr_2.uc=vettP[ptvettP];
100                 PID_P_curr_3.uc=vettP[ptvettP];
101             }
102         }
```

Figura 14.3

In figura 14.3, è possibile notare l'implementazione del timer che ci permetterà lo scorrimento del vettore. Nel nostro esempio, abbiamo simulato un cambiamento del segnale di ingresso ogni 6 ms.
Questa prova è stata sostenuta per sopperire alla mancanza del segnale di ingresso variabile dato dalla matrice di conversione dagli angoli Pitch, Roll e Yaw, nei corrispondenti valori di corrente (Competenza del gruppo precedente al nostro).