Lego EV3 Segway

Bernardo Tiezzi
7036865
bernardo.tiezzi@stud.unifi.it

Edoardo Re
7042226
edoardo.re@stud.unifi.it

Abstract

Studio stabilizzazione Segway con software di programmazione Lego Mindstorm EV3 in posizione di equilibrio instabile attraverso legge di controllo ottimo. Il codice sorgente è implementato in linguaggio C con strumento di gestione RobotC.

1. Introduzione

Il soggetto in esame è un robot a due ruote, sistema non lineare, caratterizzato da due posizioni di equilibrio, una stabile l'altra instabile. La posizione di equilibrio stabile si presenta quando il corpo del robot è posizionato orizzontalmente su un piano; tale condizione risulta trascurabile ai fini dello studio condotto. L'altro stato di equilibrio è definito dal robot disposto in posizione verticale che risulta instabile poichè è sufficiente una piccola perturbazione per non consentire al corpo di riportarsi nello stato di equilibrio iniziale. L'obiettivo dello studio è implementare un sistema con legge di controllo ottima al fine di stabilizzare il robot attorno alla posizione di equilibrio instabile. Il corpo poggia inizialmente su un cavalletto che si solleva all'avvio del processo. Al termine dell'esecuzione trascorsi dieci secondi, il cavalletto viene abbassato in modo da ritornare in una condizione di equilibrio. Il codice sorgente implementato utilizza la matrice di controllo e tempo di campionamento descritti in [1]. Per la realizzazione del processo sono stati visionati e testati diversi linguaggi di programmazione imperativi, linguaggio C, e orientati agli oggetti, come Java e Python. Di seguito vengono spiegate le motivazioni che hanno condizionato il passaggio attraverso i vari linguaggi di alto livello:

1. **Implementazione Python**: Il primo codice sorgente del progetto è stato realizzato in Python utilizzando l'ambiente di sviluppo Visual Studio Code. Tuttavia lo script di Python presentava un tempo di campionamento troppo alto introducendo ritardo durante il processo.
2. **Implementazione Java**: Per implementare il sistema di controllo in linguaggio Java è stato necessario installare il framework leJOS EV3 su una microSD da utilizzare come strumento di memoria per il robot. Nonostante le migliorie apportate al tempo di campionamento, grazie al cambio di linguaggio, l'introduzione di latenza dovuta all'inserimento in coda dei valori per i motori del robot ha condotto all'impossibilità di proseguire il progetto utilizzando Java.
3. **Implementazione C**: L'implementazione funzionante del sorgente, per consentire il mantenimento della posizione di equilibrio, è stata infine realizzata utilizzando il linguaggio C. Nei capitoli successivi verranno analizzati struttura e funzionamento del robot e del relativo codice.

2. Componenti fisiche

Il robot è costituito da più blocchi componenti: "la mente e il cuore del robot" è rappresentato dal device Brick EV3 che può supportare comunicazioni via USB, Wifi, Bluetooth. Utilizzando una scheda di memoria microSD è possibile eseguire un download del codice dal computer al dispositivo. Il brick EV3 presenta varie porte per il collegamento di periferiche.

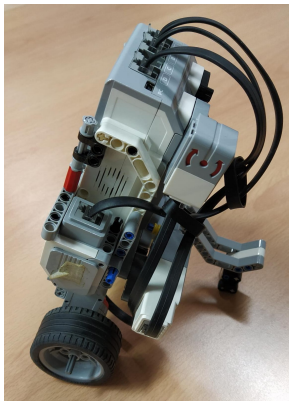

Figure 1. Vista laterale del setup LEGO EV3

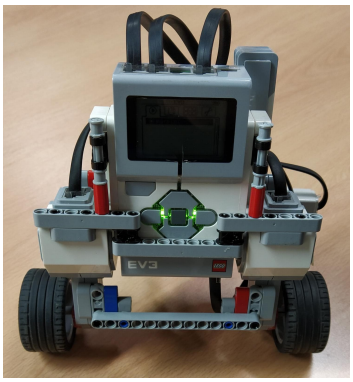

Figure 2. Vista frontale del setup LEGO EV3

Dalle figure 1 e 2 si individuano 4 periferiche collegate al device principale. Il cavalletto viene utilizzato per sorreggere il corpo del robot a riposo. La rotazione viene gestita collegando il cavalletto ad un MediumMotor tramite la porta B. La coppia delle ruote consente al robot di stabilizzarsi in posizione verticale ed è gestita da due LargeMotor distinti attraverso le porte A e D. Il giroscopio consente di ottenere real-time l'angolo di inclinazione del robot in modo da programmare il controllo in retroazione delle ruote. Il sensore è collegato alla porta S4 del dispositivo Lego.

3. Sistema di riferimento

Il sistema in esame è costituito da una struttura principale che può essere paragonata ad un pendolo inverso con ruote; le dimensioni del cavalletto possono essere trascurate.

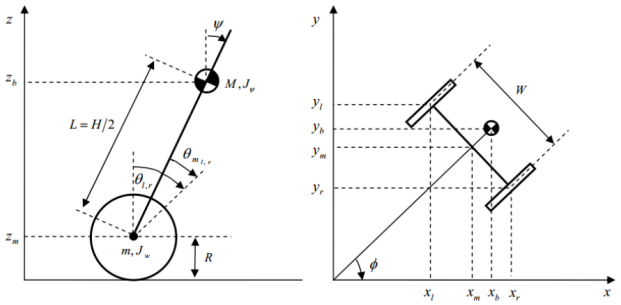

Figure 3. Sistema di riferimento adottato

In figura 3 viene mostrato il sistema di riferimento adottato durante lo studio condotto. Il corpo del robot è stato approssimato ad un punto materiale in cui è concentrata la sua massa e posizionato nel centro di massa ad altezza $H/2$ rispetto al centro dell'asse delle ruote. Per stabilizzare il dispositivo è necessario definire le seguenti variabili:

1. l'angolo (θ) dato dalla media tra l'angolo della ruota destra (θ_{mr}) e della ruota sinistra (θ_{ml});
2. l'angolo che la congiungente il centro di massa del corpo e il centro delle ruote forma con l'asse z (ψ);

Le variabili sovraelencate consentono di definire le velocità angolari $\dot{\theta}$ e $\dot{\psi}$ per l'asse delle ruote e giroscopio. La variabile θ_{int} è il valore dell'angolo θ integrato rispetto al tempo di campionamento ottenuto durante il processo. I valori assunti dalle variabili di stato sono ottenute tramite l'impiego di sensori quali gli encoder dei motori, che forniscono le misure degli angoli θ_{ml} e θ_{mr}, e il giroscopio, che riporta i valori della velocità di rotazione attorno all'asse y, $\dot{\psi}$. L'encoder del motore e il giroscopio presentano un grado di incertezza, rispettivamente pari a $\pm 1°$ e $\pm 1°/s$. Le relazioni esistenti tra le variabili sono riportate nelle formule (3.1)

$$\begin{cases} \theta_{\mathrm{l}} = \theta_{\mathrm{ml}} + \psi \\ \theta_{\mathrm{r}} = \theta_{\mathrm{mr}} + \psi \\ \theta = \frac{1}{2}(\theta_{\mathrm{ml}} + \theta_{\mathrm{mr}}) + \psi \end{cases} \tag{3.1}$$

3.1. Legge di controllo ottima

La legge di controllo adottata per stabilizzare il sistema è quella derivante dalla Teoria del Controllo Ottimo, che consiste in una retroazione statica dello stato del tipo:

$$u = -Kx \tag{3.2}$$

dove il vettore u è il vettore degli ingressi, definito come:

$$u = \begin{bmatrix} T_{\mathrm{l}} & T_{\mathrm{r}} \end{bmatrix} \tag{3.3}$$

con T_{l} e T_{r} le tensioni applicate ai motori delle ruote che consentono al robot di mantenere l'equilibrio. Il vettore dei parametri x:

$$x = \begin{bmatrix} \theta \\ \psi \\ \dot{\theta} \\ \dot{\psi} \\ \theta_{\mathrm{int}} \end{bmatrix} \tag{3.4}$$

è moltiplicato con la matrice dei guadagni K, trovata minimizzando il costo:

$$\int_0^\infty (x^T(t)Qx(t) + u^T(t)Pu(t))dt \tag{3.5}$$

La matrice di controllo comprende l'azione integrale che permette di migliorare le prestazioni del sistema ed impedire al robot di allontanarsi dalla posizione iniziale di partenza. La matrice K, ottenuta al termine del processo, è la seguente:

$$K = \begin{bmatrix} -0.8559 & -44.7896 & -1.2420 & -4.6061 & -0.5000 \\ -0.8559 & -44.7896 & -1.2420 & -4.6061 & -0.5000 \end{bmatrix} \tag{3.6}$$

4. Implementazione in C

Il codice in linguaggio C è eseguibile in pochi passi sul Lego EV3 tramite l'ambiente di sviluppo RobotC che permette, inserendo una micro SD all'interno del robot, di installare il kernel necessario all'esecuzione del codice in C. Il file da compilare è reperibile sul repository pubblico GitHub e si compone in un singolo file nominato LabRobotc.c. L'esecuzione del programma inizia con una fase calibrazione del giroscopio dove il robot deve essere posizionato perfettamente sulla verticale:

```
eraseDisplay();
displayCenteredBigTextLine(2, "Calibrate Zero");
float offset=calibrate_gyro();
//writeDebugStream("offset is: %f\n", offset);
resetGyro(Gyro);
sleep(4000);
playSound(soundBeepBeep);
```

al termine si esegue un suono che notifica di poter appoggiare il robot sul cavalletto concedendo un tempo di pausa di cinque secondi prima di avviare la fase di controllo. La calibrazione del giroscopio descritta precedentemente si compone di due fasi: la prima il calcolo dell'offset per eliminare la deriva nel valore della velocità angolare e la seconda nella chiamata alla funzione di libreria resetGyro che imposta a zero l'angolo di partenza. La calibrazione dell'offset si esegue nella funzione calibrate_gyro() che integra i valori del giroscopio per 5 secondi accumulando i valori in angle e moltiplicando per un guadagno di 1/5 ottiene una media della velocità angolare di deriva.

```
float calibrate_gyro(){
    float angle=0;
    float rate=0;
    time1[T1]=0;
    while(time1[T1]<5000){
     angle=angle+(-getGyroRate(Gyro)*0.01);
     sleep(10);
    }
    rate=angle/5;
    return rate;
}
```

Dopo aver regolato correttamente il giroscopio è possibile porre sul cavalletto il robot e successivamente si resettano gli encoder dei motori. Infine si mostrerà su display del robot la scritta "Balancing".

```
eraseDisplay();
displayCenteredBigTextLine(2, "Put on rack");
sleep(5000);
resetMotorEncoder(rightMotor);
resetMotorEncoder(leftMotor);
eraseDisplay();
displayCenteredBigTextLine(2, "Balancing");
```

La parte di stabilizzazione tramite controllo inizia con un ciclo while true che si interrompe nel momento in cui il tempo trascorso supera dieci secondi e lo Psi, ovvero l'angolo di inclinazione del robot rilevato integrando i valori del giroscopio è minore o uguale a zero che è il valore verticale. Quando il ciclo di interrompe si abbassa istantaneamente il cavalletto e si stoppano i due motori.

```
time1[T1]=0;
while(true){
    if(time1[T1]>10000 && Psi<= 0){
        moveMotorTarget(motorB,
        -getMotorEncoder(motorB),-100);
        setMotorSpeed(rightMotor, 0);
        setMotorSpeed(leftMotor, 0);
        sleep(1000);
        break;
}
```

Il codice all'interno del while esegue ogni dieci millisecondi un aggiornamento di Psi, Psi_dot, Theta, Theta_dot e Theta_int. L'aggiornamento di Psi_dot si esegue con la chiamata a funzione:

```
Psi_dot=-getGyroRate(Gyro)-offset;
```

La funzione `getGyroRate(Gyro)` recupera il valore della velocità angolare di rotazione del giroscopio, a cui viene sottratto il valore dell'offset, calcolato in precedenza. In modo analogo si ottiene il valore dell'angolo ψ.

```
Psi=-getGyroDegrees(Gyro);
```

Come si può notare sia `Psi` sia `Psi_dot` hanno segno invertito rispetto ai valori riportati dal sensore del giroscopio; tale scelta è dovuta alla discordanza tra quest'ultimo e gli encoder dei motori. Il valore della variabile θ viene ricavato applicando le formule 3.1 del capitolo 3.

```
angleDx=getMotorEncoder(rightMotor);
angleSx=getMotorEncoder(leftMotor);
ThetaM=(angleDx+angleSx)/2;
Theta=ThetaM+Psi;
```

La velocità angolare `Theta_dot`, così come l'azione integrale definita dalla variabile `Theta_int`, si ottengono applicando la definizione di rapporto incrementale e la regola del punto medio per l'integrazione matematica.

```
ThetaList[0]=ThetaList[1];
ThetaList[1]=Theta;
    if (count>1){
        Theta_dot=(ThetaList[1]-ThetaList[0])/t;
        Theta_int=Theta_int+(Theta*(t));
    }
```

Derivata ed integrale dell'angolo `Theta` sono calcolate rispetto al tempo di campionamento `t=0.01s` [2]. La variabile `count` assicura che le celle dell'array `ThetaList` non siano vuote. I dati forniti dai sensori vengono utilizzati per definire il vettore u di stato del sistema.

```
ThetaRad=Theta*PI/180;
PsiRad=Psi*PI/180;
Theta_dotRad=Theta_dot*PI/180;
Psi_dotRad=Psi_dot*PI/180;
Theta_intRad=Theta_int*PI/180;
u=(-0.8559*ThetaRad)+(-44.7896*PsiRad)+(-0.9936*Theta_dotRad)
+(-4.6061*Psi_dotRad)+(-0.500*Theta_intRad);
u=-u*100/getBatteryVoltage();
setMotorSpeed(rightMotor, u);
setMotorSpeed(leftMotor, u);
```

Il vettore x (3.4), dopo averne portato le componenti da gradi in radianti, viene moltiplicato per la matrice dei guadagni K, ridotta ad un vettore per ottimizzare il tempo di esecuzione del processo. La variabile di ingresso ai

motori `u` viene portata in percentuale e passata come parametro alla funzione `setMotorSpeed(Motor, u)`, per entrambi i motori delle ruote; la funzione statica `getBatteryVoltage()` viene utilizzata per prelevare il voltaggio effettivo del dispositivo. Affinchè il robot possa stabilizzarsi è necessario che, dopo aver avviato il programma, il cavalletto venga alzato, così da non destabilizzare la struttura.

```
count=count+1;
if (count==15){
            //rackUp
            moveMotorTarget(motorB, 100, 10);
        }
sleep(10);
```

Il `count` viene incrementato ad ogni iterazione del ciclo `while` iniziale. Al termine della quindicesima iterazione, dopo circa $1.5s$, il cavalletto viene alzato, in modo da coincidere con la spinta iniziale data alle ruote. Il metodo `moveMotorTarget()` solleva il cavalletto regolando la velocità per ridurre il rumore meccanico. Poichè il tempo di campionamento applicato al calcolo integrale e differenziale è $0.01s$ ed ogni iterazione impiega in media $0.0009s$ si applica una `sleep(10)` al termine del ciclo `while`.

5. Esecuzione e analisi delle prestazioni

L'esecuzione del programma si divide in più fasi:

1. **Calibrazione del giroscopio**:

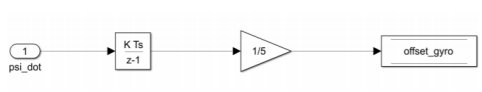

Figure 4. fase di calibrazione

   in fase di avvio, il robot deve essere mantenuto nella posizione di equilibrio instabile per consentire al giroscopio di calibrarsi sulla posizione $\psi = 0$. Durante l'operazione sul display del Brick EV3 device viene visualizzata la stringa **Calibrate Zero**.

2. **Avvio dell'esecuzione**: Al termine della fase precedente viene emesso un suono dal dispositivo e sul display viene mostrato il comando **Put on rack**. L'utente deve quindi riporre il corpo nella condizione di riposo sul cavalletto. Il tempo stimato affinche il robot sia pronto per entrare nella fase di controllo è di circa $5s$.

3. **Stabilità nel punto d'equilibrio**:

   durante l'esecuzione del programma il corpo è in grado di mantenersi stabile, azionando il movimento delle ruote secondo l'inclinazione dell'asse del centro di massa.

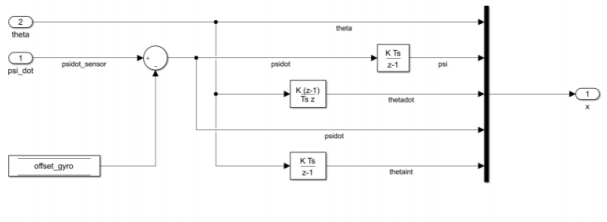

Figure 5. Stima dello stato

4. **Terminazione**: il processo si conclude dopo essere trascorsi almeno $10s$ dall'inizializzazione. Il cavalletto viene abbassato, portando il robot ad arrestarsi e tornare nella posizione di riposo.

Nel caso si verifichino problemi durante l'esecuzione del codice sorgente è consigliabile controllare che le porte siano correttamente collegate al device principale. In fig. 4, 5, 6 sono mostrati i modelli simulink del sistema ad anello chiuso.

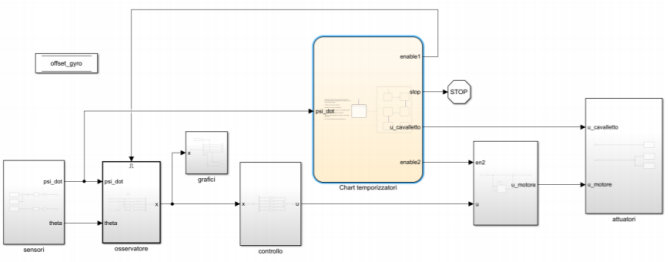

Figure 6. Fasi del processo

Durante l'esecuzione del file sorgente, i dati ottenuti dai sensori del robot vengono presentati attraverso l'utilizzo di funzioni grafiche proprie della piattaforma **RobotC**.

```
datalogDataGroupStart();
datalogAddValue(0, Psi_dot);
datalogAddValue(1, Psi);
datalogAddValue(2, Theta);
datalogAddValue(3, Theta_int);
datalogAddValue(4, Theta_dot);
datalogDataGroupEnd();
```

L'esecuzione del processo è disponibile su YouTube.

Figure 7. Video dell'esecuzione

I grafici sono descritti in figura 8, 9 e 10. In figura 8 si può osservare l'andamento dell'angolo θ, integrale e derivata durante la stabilizzazione del robot. Il valore iniziale di ogni variabile è 0. Il robot, per potersi sollevare necessita di una spita iniziale negativa; in seguito si può notare che tutte le funzioni, suppur con andamenti differenti, tendono a mantenersi attorno al valore 0. La velocità angolare $\dot{\theta}$ esegue rapide variazioni in modo da mantenere stabile il dispositivo in posizione verticale. Le variabili θ e θ_{int} tendono a ritornare nella posizione di partenza.

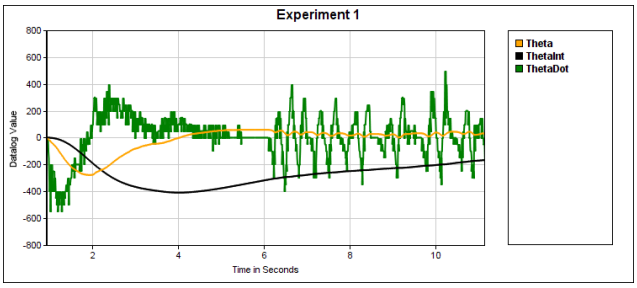

Figure 8. Evoluzione degli angoli Theta, integrale e derivata in una esecuzione

Il grafico di fig.9 evidenzia con maggior dettaglio l'andamento dell'angolo θ. È possibile notare come l'evoluzione valori ottenuti per ψ e $\dot{\psi}$ (figura 10) sia analoga al comportamento tenuto dalle variabili θ. Ciò è dovuto al controllo in retroazione che consente di direzionare la velocità delle ruote in base al valore fornito dall'angolo ψ. La variazione della velocità di rotazione $\dot{\theta}$, causerà una variazione della velocità misurata dal giroscopio $\dot{\psi}$.

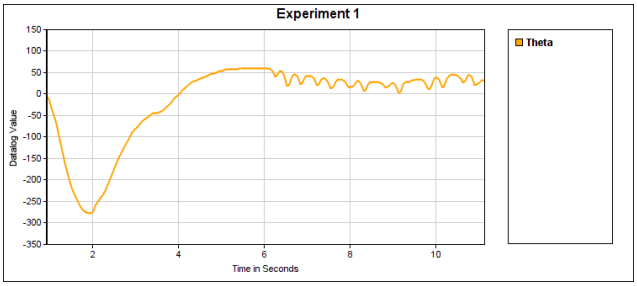

Figure 9. Evoluzione dell' angolo Theta in una esecuzione

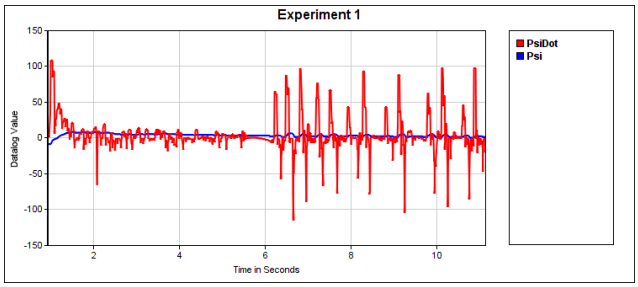

Figure 10. Evoluzione degli angoli Psi e PsiDot in una esecuzione

6. Conclusioni

Lo studio condotto aveva il compito di riprodurre il sistema di retroazione con legge di controllo ottima [1], utilizzando un linguaggio di programmazione software. Le difficoltà maggiori sono state riscontrate nella scelta del linguaggio da adottare. Un programma per la stabilizzazione di un dispositivo richiede tempi di esecuzione molto rapidi; l'introduzione di un ritardo tra la variazione dello stato e la percezione effettiva del cambiamento, da parte dei sensori, può portare al fallimento del processo. Linguaggi come `Python` e `Java` si sono dimostrati inadatti a causa di tempi di esecuzione non ottimali. La rapidità di calcolo è un altro fattore di rilevanza; le funzioni utilizzate non devono nè introdurre latenza nei tempi di risposta ad una chiamata nè risultare bloccanti durante l'esecuzione del programma. Altre difficoltà riguardano il movimento delle ruote: l'attivazione della trazione del robot genera rumore meccanico che influisce sia sulla rilevazione del giroscopio, sia sui collegamenti delle periferiche sulle varie porte. Se necessario è possibile eseguire un'operazione di smoothing del segnale utilizzando un `Low Pass Filter`; tuttavia un segnale filtrato potrebbe causare un ritardo nell'esecuzione, compromettendo la stabilità del corpo in esame. Per migliorare l'algoritmo di controllo potrebbe essere interessante aggiungere un `osservatore dello stato`, in modo da monitorare l'evoluzione del sistema ed eseguire una stima dello stato più accurata.

References

[1] S. C. Davide Martini. Controllo di un robot a due ruote realizzato con lego mindstorms ev/3, 2020.

[2] Y. Yamamoto. Nxtway-gs (self-balancing two-wheeled robot) controller design.