VAB - Veicolo auto bilanciato

Relazione di progetto

Laboratorio di Sistemi Meccatronici II
Università degli Studi di Bergamo
Kilometro rosso
A.A. 2019/2020

Calegari Andrea - 1041183
Piffari Michele - 1040658

September 10, 2020

Contents

List of Figures

1

Introduzione

L'approccio seguito per la stesura del modello dinamico del veicolo autobilanciato ha da subito preso una via meno *tradizionale* rispetto al classico metodo risolutivo: abbiamo infatti preferito, dato il nostro *background* informatico, approcciare il problema direttamente in ambiente Matlab, sfruttando sin da subito le potenzialità di calcolo offerte dal software di *Mathworks*.

Nello specifico, per la parte di stesura e definizione della dinamica, abbiamo inzialmente seguito una via risolutiva duale, portando avanti sia un'analisi letterale, sfruttando le potenzialità del **calcolo simbolico** messe a disposizione delle funzionalità di **live scripting**, sia uno studio numerico (considerando quindi le varie grandezze fisiche con i valori definiti delle specifiche di progetto).

In linea di massima lo sviluppo del progetto ha seguito un andamento a step graduali, cadenziati da incontri settimanali in cui poter confrontare e consolidare lo *stato di avanzamento dei lavori*: nello specifico, il lavoro ha seguito uno sviluppo in questa direzione step by step, rappresentabile in linea di massima da queste *pietre miliari*:

- **Dinamica di ogni singolo corpo rigido**: abbiamo impostato il problema della dinamica andando a considerare il veicolo auto bilanciato come un insieme di corpi rigidi di cui poterne studiare la dinamica in maniera separata;
- **Dinamica completa del VAB**: siamo andati poi a considerare il sistema nella sua completezza, unendo i contributi dei corpi rigidi considerati in prima battuta singolarmente. Questo ci ha permesso di ottenere le equazioni del moto, in forma non lineare, le quali hanno permesso una completa descrizione del sistema che abbiamo modelizzato;
- **Linearizzazione**: siamo poi andati a linearizzare queste equazioni dinamiche (appunto non lineari) nell'intorno dell'equilibrio;
- **Definizione del controllo**: tramite la tecnica di *pole placement*, siamo andati a definire il controllore più adatto per questo sistema;
- **Modellizzazione del motore**: introduzione del modello del motore sulla base delle caratteristiche reali contenute nella specifica di progetto;
- **Discretizzazione**: passaggio a tempo discreto per i segnali derivanti dal mondo analogico, ovvero per tutto ciò che concerne la parte di sensoristica;
- **Non idealità**: siamo andati a modellizzare anche la presenza di disturbi, di natura stocastica e legati alla quantizzazione;
- **Trasformazioni di blocchi in *interpreted function***: conversione delle funzionalità di controllo del motore e filtraggio dei segnali provenienti dai sensori in blocchi di codice *Matlab* per favorirne poi la successiva conversione ed implementazione a bordo del controllore;

2

Dinamica

2.1 Scomposizione del VAB

2.1.1 Considerazioni iniziali

Per il calcolo delle equazioni dinamiche del sistema siamo andati a considerare ogni singolo corpo rigido componente il sistema, calcolandone le grandezze fisiche di posizione e velocità, seguendo un approccio cartesiano. Nello specifico abbiamo considerato il sistema composto da:

- Asta
- Utente a bordo dello chassis
- Chassis (nel corso della trattazione sarà chiamata talvolta anche base)
- Ruota (che poi sarà considerata con un contributo doppio, essendo il VAB composto da due ruote)

Ognuno di questi corpi rigidi separati è individuato da un punto, che ne rappresenta il centro di massa (o baricentro del corpo stesso): avremo quindi il seguente insieme di punti caratterizzanti il sistema (figura 2.1)

- P_a
- P_b
- P_c
- P_r

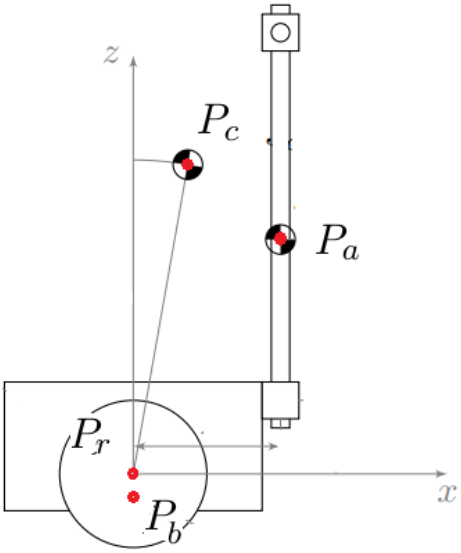

Figure 2.1: Baricentri dei singoli corpi rigidi

2.1.2 Grandezze di supporto

Prima di andare a definire le componenti di energia potenziale e cinetica di ogni singolo corpo, siamo andati ad introdurre alcune grandezze geometriche di supporto che definiremo qui di seguito.

Nello specifico abbiamo introdotto i seguenti parametri, specificati anche in figura 2.2:

- $\mathbf{l_a}$: rappresenta la congiungente tra il centro del sistema di riferimento XZ e il centro dell'asta, utilizzata appunto come manubrio, individuato come

$$l_a = \sqrt{(\frac{h_a}{2} + \frac{h_b}{2})^2 + (\frac{w_b}{2})^2}$$

- $\mathbf{l_c}$: questa grandezza rappresenta per noi l'altezza del baricentro del corpo dell'utente, la quale ovviamente andrà a dipendere dal valore di inclinazione del corpo stesso. Considerando il corpo inzialmente in posizione verticale, avremo che questa lunghezza corrisponde alla congiungente dal centro del sistema di riferimento al punto P_c, che equivale a dire:

$$l_c = 0.55h_c + \frac{h_b}{2}$$

- $\mathbf{l_b}$: rappresenta lo spostamento verso il basso, lungo l'asse z, del baricentro dello chassis. Da specifiche del progetto sappiamo che questa grandezza ha valore (con segno negativo) di

$$l_b = 0.1m$$

- β: angolo formato con la verticale dalla congiungente tra il centro del sistema di riferimento e il punto P_a. Si ricava, con un semplice approccio trigonometrico che, l'angolo in questione, ha questa forma:

$$\arctan\left(\frac{\frac{w_b}{2}}{\frac{h_a}{2}+\frac{h_b}{2}}\right)$$

2.1.3 Calcolo componenti dinamiche e potenziali per ogni corpo rigido del sistema

Per ognuno dei corpi rigidi definiti in precedenza siamo andati appunto a calcolare:

- **Coordinate spaziali P** espresse nel sistema di riferimento XZ. A queste due coordinate cartesiane ne va aggiunta una terza, relativa alle coordinate angolari (per poter tener così conto dei contirbuti inerziali dei corpi);

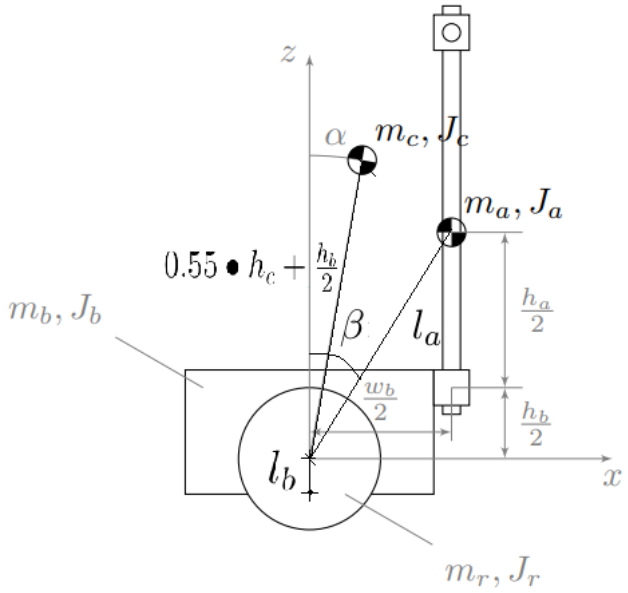

Figure 2.2: Grandezze di supporto

- **Vettore delle velocità V** $\rightarrow$ vettore 3×1
- **Matrice delle masse M** $\rightarrow$ matrice 3×3
- **Energia cinetica T** $\rightarrow \frac{1}{2} V^T M V$
- **Energia potenziale U**
- **Lagrangiana *parziale* L del singolo corpo rigido**

Asta

- $P_a = \begin{pmatrix} r\,\phi(t) + l_a \sin(\beta + \theta(t)) \\ l_a \cos(\beta + \theta(t)) \\ \theta(t) \end{pmatrix}$
- $V_a = \begin{pmatrix} r\,\dot{\phi}(t) + l_a \cos(\beta + \theta(t))\ \dot{\theta}(t) \\ -l_a \sin(\beta + \theta(t))\ \dot{\theta}(t) \\ \dot{\theta}(t) \end{pmatrix}$
- $M_a = \begin{pmatrix} m_a & 0 & 0 \\ 0 & m_a & 0 \\ 0 & 0 & m_a\,{l_a}^2 + J_a \end{pmatrix}$
- $T_a = m_a\,{l_a}^2 \left(\dot{\theta}(t)\right)^2 + \frac{m_a\,r^2\left(\dot{\phi}(t)\right)^2}{2} + \frac{J_a\left(\dot{\theta}(t)\right)^2}{2} + m_a \cos(\beta + \theta(t))\ l_a\,r\,\dot{\theta}(t)\,\dot{\phi}(t)$
- $U_a = g\,l_a\,m_a \cos(\beta + \theta(t))$
- $L_a = m_a\,{l_a}^2 \left(\dot{\theta}(t)\right)^2 + \frac{m_a\,r^2\left(\dot{\phi}(t)\right)^2}{2} + \frac{J_a\left(\dot{\theta}(t)\right)^2}{2} + m_a \cos(\beta + \theta(t))\ l_a\,r\,\dot{\theta}(t)\,\dot{\phi}(t) - g\,m_a \cos(\beta + \theta(t))\ l_a$

Chassis (base)

- $P_b = \begin{pmatrix} r\,\phi(t) - l_b \sin(\theta(t)) \\ -l_b \cos(\theta(t)) \\ \theta(t) \end{pmatrix}$
- $V_b = \begin{pmatrix} r\,\dot{\phi}(t) - l_b \cos(\theta(t))\ \dot{\theta}(t) \\ l_b \sin(\theta(t))\ \dot{\theta}(t) \\ \dot{\theta}(t) \end{pmatrix}$
- $M_b = \begin{pmatrix} m_b & 0 & 0 \\ 0 & m_b & 0 \\ 0 & 0 & m_b\,{l_b}^2 + J_b \end{pmatrix}$
- $T_b = m_b\,{l_b}^2 \left(\dot{\theta}(t)\right)^2 + \frac{m_b\,r^2 \left(\dot{\phi}(t)\right)^2}{2} + \frac{J_b \left(\dot{\theta}(t)\right)^2}{2} - m_b \cos(\theta(t))\ l_b\,r\,\dot{\theta}(t)\,\dot{\phi}(t)$
- $U_b = -g\,l_b\,m_b \cos(\theta(t))$
- $L_b = m_b\,{l_b}^2 \left(\dot{\theta}(t)\right)^2 + \frac{m_b\,r^2 \left(\dot{\phi}(t)\right)^2}{2} + \frac{J_b \left(\dot{\theta}(t)\right)^2}{2} - m_b \cos(\theta(t))\ l_b\,r\,\dot{\theta}(t)\,\dot{\phi}(t) + g\,m_b \cos(\theta(t))\ l_b$

Utente

- $P_c = \begin{pmatrix} r\,\phi(t) + l_c \sin(\alpha + \theta(t)) \\ l_c \cos(\alpha + \theta(t)) \\ \alpha + \theta(t) \end{pmatrix}$
- $V_c = \begin{pmatrix} r\,\dot{\phi}(t) + l_c \cos(\alpha + \theta(t))\ \dot{\theta}(t) \\ -l_c \sin(\alpha + \theta(t))\ \dot{\theta}(t) \\ \dot{\theta}(t) \end{pmatrix}$
- $M_c = \begin{pmatrix} m_c & 0 & 0 \\ 0 & m_c & 0 \\ 0 & 0 & m_c\,{l_c}^2 + J_c \end{pmatrix}$
- $T_c = m_c\,{l_c}^2 \left(\dot{\theta}(t)\right)^2 + \frac{m_c\,r^2 \left(\dot{\phi}(t)\right)^2}{2} + \frac{J_c \left(\dot{\theta}(t)\right)^2}{2} + m_c \cos(\alpha + \theta(t))\ l_c\,r\,\dot{\theta}(t)\,\dot{\phi}(t)$
- $U_c = g\,l_c\,m_c \cos(\alpha + \theta(t))$
- $L_c = m_c\,{l_c}^2 \left(\dot{\theta}(t)\right)^2 + \frac{m_c\,r^2 \left(\dot{\phi}(t)\right)^2}{2} + \frac{J_c \left(\dot{\theta}(t)\right)^2}{2} + m_c \cos(\alpha + \theta(t))\ l_c\,r\,\dot{\theta}(t)\,\dot{\phi}(t) - g\,m_c \cos(\alpha + \theta(t))\ l_c$

Ruota

- $P_r = \begin{pmatrix} r\,\phi\,(t) \\ 0 \\ \phi\,(t) \end{pmatrix}$
- $V_r = \begin{pmatrix} r\,\phi\,\dot{(t)} \\ 0 \\ \phi\,\dot{(t)} \end{pmatrix}$
- $M_r = \begin{pmatrix} m_r & 0 & 0 \\ 0 & m_r & 0 \\ 0 & 0 & J_r \end{pmatrix}$
- $T_r = \frac{\left(m_r\,r^2+J_r\right)\left(\phi\dot{(t)}\right)^2}{2}$
- $U_r = 0$
- $L_r = \frac{\left(m_r\,r^2+J_r\right)\left(\phi\dot{(t)}\right)^2}{2}$

Veicolo completo

Una volta trovati le componenti dinamiche dei singoli corpi rigidi, possiamo andare a definire l'energia cinetica e potenziale totale del sistema, per poter poi calcolare l'equazione di Lagrange per l'intero sistema. In sostanza quindi avremo:

$$L = L_a + L_b + L_c + 2L_r$$

Quello che ne deriva è quindi la seguente equazione Lagrangiana, in grado di descrivere la dinamica dell'intero sistema (riportiamo il risultato letterale, ovvero non legato alla sostituzione di alcun valore numerico che descrive il sistema):

$$L = \left(m_a\,l_a^2 + m_b\,l_b^2 + m_c\,l_c^2 + \tfrac{J_a}{2} + \tfrac{J_b}{2} + \tfrac{J_c}{2}\right)(\theta\,\dot{(t)})^2 + \left(m_r\,r^2 + J_r + \tfrac{m_a\,r^2}{2} + \tfrac{m_b\,r^2}{2} + \tfrac{m_c\,r^2}{2}\right)(\phi\,\dot{(t)})^2 + \left(m_c\,\sigma_4\,l_c\,r + m_a\,\sigma_3\,l_a\,r - m_b\,cos\,(\theta\,(t))\;l_b\,r\right)\theta\,\dot{(t)}\,\phi\,\dot{(t)} + g\,m_b\cos\,(\theta\,(t))\;l_b - g\,m_c\,\sigma_4\,l_c - g\,m_a\,\sigma_3\,l_a$$
$$\sigma_3 = \cos\,(\beta + \theta\,(t))\;\sigma_4 = \cos\,(\alpha + \theta\,(t))$$

Note sul calcolo delle componenti dinamiche

- Nel calcolo della matrice di massa abbiamo considerato tre componenti:
  - Componente di massa lungo x;
  - Componente di massa lungo z;
  - Componente di massa rotazionale: dalla meccanica è noto che un corpo, con una certa massa che si trova in uno stato di rotazione, avrà un contirbuto inerziale che dipende dal braccio rispetto all'asse intorno al quale avviene la rotazione.

    Nello specifico abbiamo considerato il teorema di *Huygens-Steiner* per ogni singolo corpo rigido che è stato preso in esame.

    $$I_a = I_{cm} + md^2$$

    Esso permette di definire l'inerzia di un corpo come la somma di due diverse componenti:

* Momento d'inerzia definito rispetto all'asse passante per il centro di massa: questo parametro rappresenta il valore che è fornito dalle specifiche del progetto per gli specifici corpi;
* Prodotto tra la massa m del corpo preso in considerazione e la distanza tra l'asse rispetto a cui si riferisce la rotazione e quello passante per il centro di massa;

Questa componente inerziale è visibile nelle matrici di massa in posizione $(3,1)$: si sottolinea come invece, per la matrice di massa relativa alle ruote (matrice M_r 2.1.3), non sia presente la componente esplicitata dal teorema di *Huygens-Steiner* per il fatto che il centro di massa della ruota coincide con quello del sistema di riferimento XZ e quindi di conseguenza non è necessaria alcuna traslazione di asse;

- Nel calcolo simbolico in ambiente *Matlab* siamo andati ad utilizzare le funzionalità di *collect* e *simplify*, per permettere di ridurre e semplificare le equazioni. Nello specifico, con il comando simplify, tramite l'opzione *Steps*, siamo andati a settare il numero di step che l'algoritmo di calcolo simbolico andrà ad eseguire per poter ridurre e semplificare il maggior numero di termini possibili ($\uparrow$ *Steps*, $\uparrow$*Compattezza eq symb*);
- Le ruote, nel calcolo della dinamica completa del VAB, sono state considerate con un contirbuto doppio;
- Nello studio dello chassis (base), abbiamo seguito questo approccio: il baricentro della base stessa sappiamo essere posizionato ad una quota differente rispetto al centro del sistem di coordinate XZ preso come riferimento. Per questo motivo il suo contributo in termini cinetici e potenziali dipende dal valore dell'inclinazione dello chassis, ovvero dal valore dell'angolo *caratterizzante* il sistema θ: questo concetto è evidenziato in figura 2.3. L'aggiunta di π al valore di θ è necessaria per poter rendere sensibile i valori di energia cinetica e potenziale al *quadrante* in cui si trova ad essere posizionato il centro di massa della base stessa (P_b).

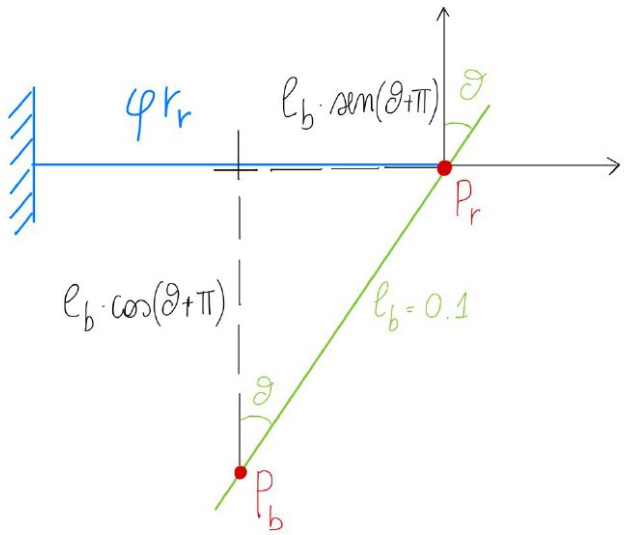

Figure 2.3: Ragionamento per il calcolo dei contributi dinamici dello chassis

2.2 Equazioni del moto

2.2.1 Lagrangiana

Una volta definita la dinamica del sistema ed ottenuta quindi l'equazione di Lagrange che ne caratterizza il comportamento, andiamo a ricavare le equazioni del moto, le quali consentono di definire

l'andamento delle *coordinate libere* (scelte in fase inziale di progetto) che sappiamo essere

- $\phi = q_1 =$ *angolo di rotazione delle ruote*
- $\theta = q_2 =$ *angolo di inclinazione dello chassis*

In particolare possiamo definire le cosidette *equazioni di Eulero-Lagrange*, ovvero un insieme di n equazioni differenziali (con n pari al numero di coordinate libere del sistema), la cui risoluzione fornisce le equazioni del moto del sistema.

$$\frac{d}{\mathrm{dt}}\left(\frac{\partial}{\partial \dot{q}_i}L\right) - \frac{\partial}{\partial q_i}L = \mathrm{Q_i} \;\rightarrow i = 1, ..., n$$

Sono da effettuare alcune annotazioni sugli addenti presenti nell'equazione precedente (eq 2.2.1):

- q rappresenta la coordinata libera (nel nostro caso saranno θ e ϕ);
- al secondo membro della lagrangiana troviamo le componenti generalizzate, relative alle forze attive esterne. Nel nostro sistema andiamo a modellizzare, come forze attive, quelle che sono le forze motrici immesse dai motori che gestiscono il movimento del V.A.B.: in particolare, nel nostro caso, per il legame matematico tra le componenti generalizzate e il concetto di lavoro virtuale, avremo che l'equazione lagrangiana sarà equiparata al valore di torque immessa dal motore nel sistema;
- molto importante, ai fini della correttezza delle equazioni del moto, è il valore da assegnare al secondo membro *torque*; infatti esso varia a seconda che il sistema agisca sulla coordinata libera q a valle o a monte della trasmissione.

  Nello specifico, si osserva che il motore è composto da due parti: lo statore e il rotore. Esse trasmettono una coppia uguale e inversa ai corpi a cui sono solidalmente collegati: lo statore, posizionato nella base del Segway, influenza la coordinata libera θ ed esercita una coppia sullo chassis pari ad una generica coppia $-2C_m$. Il fattore moltiplicativo 2 è dovuto al fatto che nella base sono presenti due motori.

  Nello stesso modo, il rotore, trasmette all'albero motore una coppia C_m: a valle della trasmissione si misura dunque una coppia $\frac{C_m}{\tau}$ dovuta alla presenza della trasmissione Questa coppia ridotta all'albero dell'utilizzatore è la coppia che in definitiva va ad agire sulle ruote e quindi sulla coordinata ϕ.

Si ottiene dunque il seguente sistema di equazioni, la cui risoluzione permette di ottenere due equazioni differenziali del secondo ordine nelle variabili θ e ϕ:

$$\begin{cases} \frac{d}{\mathrm{dt}}\left(\frac{\partial}{\partial\dot{\theta}}L\right) - \frac{\partial}{\partial\theta}L = -2\mathrm{C_m} \\ \frac{d}{\mathrm{dt}}\left(\frac{\partial}{\partial\dot{\phi}}L\right) - \frac{\partial}{\partial\phi}L = \frac{\mathrm{C_m}}{\tau} \end{cases}$$

dove τ è il rapporto di trasmissione il cui calcolo è riportato nella sezione 2.2.2. La soluzione di questo sistema di equazioni verrà ad essere utilizzata per rappresentare il comportamento del sistema reale, basandosi su una coppia di equazioni non lineare: questa tematica verrà poi approfondita direttamente nella sezione 4.2.

2.2.2 Rapporto di trasmissione

Come da specifiche del progetto, ognuno dei due motori è collegato alle ruote mediante una doppia sequenza di riduttori:

- Un primo riduttore epicicloidale con rapporto di trasmissione $\tau_1 = 0.1$;
- Un secondo riduttore, una cinghia dentata che presenta un rapporto di trasmissione facilmente calcolabile come rapporto tra il numero dei denti dei due alberi, essendo il passo uguale in entrambi gli alberi.

$$\tau_2 = \frac{Numero\_denti\_puleggia\_ingresso}{Numero\_denti\_puleggia\_uscita} = \frac{Z_{in}}{Z_{out}} = \frac{22}{26}$$

Il rapporto di trasmissione completo τ è quindi definito come:

$$\tau = \tau_1 \cdot \tau_2 = 0.085$$

2.2.3 Sistema non lineare

La risoluzione del sistema di equazioni sopra riportato permette di ottenere due equazioni differenziali del secondo ordine: queste soluzioni sono state ottenute sfruttando la potnezialità del calcolo simbolico messo a disposizione da Matlab, come si vede nel listato seguente.

```
theta2_diff = subs(theta2,{phi,vel,theta,vel_ang,C_m},{q_1 q_1_p q_2 q_2_p u})
phi2_diff = subs(phi2,{phi,vel,theta,vel_ang,C_m},{q_1 q_1_p q_2 q_2_p u })
```

Listing 2.1: Risoluzione del sistema di equazioni

Le soluzioni così ottenute rappresentano e governano la dinamica del sistema: esse verranno quindi utilizzate per andare ad analizzare il comportamento del sistema considerato reale, utlizzando il controllore che è stato progettato sul sistema linearizzato.

Un passaggio importante per la simulazione del sistema non lineare, è quello rappresentato dal salvataggio di queste due equazioni in esame: infatti esse sono, per forza di cose, richiamate all'interno della simulazione *Simulink*: per fare ciò abbiamo utilizzato il comando *matlabFunction()*, il quale va a scrivere in una funzione di matlab le due equazioni in esame.

```
matlabFunction(theta2_diff,'File','theta_secondo');
matlabFunction(phi2_diff,'File','phi_secondo');
```

Listing 2.2: Salvataggio in funzioni Matlab

Nello specifico queste funzioni avranno come input i valori da cui dipendono le equazioni differenziali e, componendoli algebricamente tra di loro, fornirà come output i valori di $\ddot{\theta}$ e $\ddot{\phi}$.

TODO (mettere le dipendende di theta pp e phi pp)

2.2.4 Linearizzazione

TODO linearizzazione attorno alla massa 70 kg e rileggere paragrafo

Nell'approccio alla modellistica dei sistemi dinamici, uno step molto importante è quello che concerne la linearizzazione del sistema, ovvero il passaggio da un insieme di equazioni non lineari ad un set di equazioni lineari definite nell'intorno di un punto specifico dello stato del sistema: questo nuovo sistema di equazioni andrà a definire un sistema lineare in grado di approssimare il comportamento dinamico del sistema non lineare vicino all'equilibrio.

Il punto in questione è quello in cui la variazione dello stato del sistema è nullo: deto quindi $\mathbf{x}(t)$ il vettore di stato, l'equilibrio sarà dato da $\dot{\mathbf{x}}(t)$.

Perciò, il passo successivo alla risoluzione del sistema di equazioni di Lagrange in θ e ϕ che è stato svolto riguarda la linearizzazione: nel contesto del controllo, linearizzare è importante poiché permette di progettare il controllore stesso sul sistema tangente a quello reale (ovvero il sistema lineare), potendo poi testarlo direttamente sul sistema reale (ovvero quello descritto da equazioni non lineari).

Calcolo della posizione di equilibrio

Per poter quindi procedere con la linearizzazione, è necessario innanzitutto calcolare l'equilibrio del sistema, cioè il punto in cui $\ddot{\theta} = \mathbf{0}$: questa condizione corrisponde ad una situazione in cui lo stato del sistema risulta essere in una condizione di equilibrio dinamico, ovvero quando la sua variazione nel tempo è nulla (nessuna accelerazione $\rightarrow$ velocità costante $\rightarrow$ posizione lineare):

$$\begin{pmatrix} -0.01174364 - 7.105427\text{e-}15\,\mathrm{i} \\ 3.129849 - 7.105427\text{e-}15\,\mathrm{i} \end{pmatrix}$$

Queste posizioni di equilibrio le abbiamo ottenute per mezzo della istruzione Matlab **??**, nella quale si vede come, partendo dall'equazione differenziale del sistema non lineare espressa in termini di $\ddot{\theta}$ si ricava la posizione di equilibrio del sistema.

```
equilibri = solve(subs(theta2_differenziabile,{q_1 q_1_p q_2_p u},{0,0,0,0})==0,q_2)
```

Listing 2.3: Calcolo posizione di equilbrio

Come si vede, siamo interessati solamente al valore di equilibrio relativo di θ: per questo motivo che poniamo a 0 (valore arbitrario) i valori di ϕ e $\dot{\phi}$, poichè non risultano essere di interesse per il calcolo dell'equilibrio.

I risultati ottenuti sono, con buona approssimazione, considerabili numeri naturali e rispondono a quanto ci aspettavamo: essendo presente un offset (w_b) tra quello che è il baricentro del sistema di riferimento e quello relativo al manubrio, non ci aspettavamo di ottenere due equilibri perfettamente di 0° (posizione verticale a "testa in sù") e di 180° (posizione verticale a "testa in giù").

Per completezza e per avere comunque un feedback più concreto, abbiamo provato ad azzerare il valore di w_b, andando a ricalcolare la posizione di equilibrio, ottenendo effettivamente i valori di

$$\begin{pmatrix} 0 \text{ deg} \\ 180 \text{ deg} \end{pmatrix}$$

Dal risultato ottenuto, considerando nulla la componente immaginaria, possiamo notare che (osservazioni comuni per i sistemi assimilabili al pendolo):

- il sistema ha due equilibri; il primo ($\theta = -0.01174364$ rad $= -0.67$ deg) ci si aspetta che sia instabile in quanto il baricentro del sistema V.A.B. (compreso di utente a bordo) risulta essere sopra all'asse delle ruote;
- il secondo ($\theta = 3.129849$ rad $= 179.32$ deg) è un equilibrio stabile, poiché il baricentro sta sotto l'asse delle ruote e, un eventuale disturbo dopo un certo tempo di transitorio, risulterebbe avere effetto nullo sullo stato del sistema, che ritornerebbe alla medesima posizione;

Calcolo del sistema lineare

Ovviamente, ai fini del controllo, si è linearizzato attorno al primo equilibrio; non avrebbe infatti senso controllare il sistema quando questo risulta essere capovolto (cosa che per di più risulta essere fisicamente impossibile, se il veicolo autobilanciato viene fatto muovere su una superficie).

I vettori che definiscono le **variabili di stato del sistema** e le **uscite del sistema** sono i seguenti:

$$x = \text{variabili di stato} = \begin{bmatrix} x_1 \to \phi \\ x_2 \to \dot{\phi} \\ x_3 \to \theta \\ x_4 \to \dot{\theta} \end{bmatrix}$$

$$y = \text{output del sistema} = \begin{bmatrix} y_1 \to \ddot{\theta} \\ y_2 \to \ddot{\phi} \end{bmatrix}$$

Per individuare il risultato finale della linearizzazione, abbiamo seguito un approccio matriciale: nello specifico abbiamo utilizzato la notazione matriciale nel caso di sistemi SIMO (*Single Input Multiple Output*), la cui stesura ha utilizzato le seguenti funzioni di supporto:

$$\begin{cases} \mathbf{f_1} \to \dot{x_1}(t) = \overline{x_2} = 0 \text{ (poichè all'equilibrio)} \\ \mathbf{f_2} \to \dot{x_2}(t) = \ddot{\phi} \\ \mathbf{f_3} \to \dot{x_3}(t) = \overline{x_4} = 0 \text{ (poichè all'equilibrio)} \\ \mathbf{f_4} \to \dot{x_4}(t) = \ddot{\theta} \\ \mathbf{g_1} \to y_1\ (t) = \phi = x_1 \\ \mathbf{g_2} \to y_2\ (t) = \theta = x_2 \end{cases}$$

Sfruttando le caratteristiche *matematiche* del sistema non lineare, ottenibili tramite lo sviluppo in serie di Taylor nell'intorno dell'equilibrio, possiamo definire i termini matriciali che permettono di definire il sistema, secondo le seguenti equazioni:

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases}$$

$$A = \begin{bmatrix} \frac{\partial}{\partial x_1} f_1 & \frac{\partial}{\partial x_2} f_1 & \frac{\partial}{\partial x_3} f_1 & \frac{\partial}{\partial x_4} f_1 \\ \frac{\partial}{\partial x_1} f_2 & \frac{\partial}{\partial x_2} f_2 & \frac{\partial}{\partial x_3} f_2 & \frac{\partial}{\partial x_4} f_2 \\ \frac{\partial}{\partial x_1} f_3 & \frac{\partial}{\partial x_2} f_3 & \frac{\partial}{\partial x_3} f_3 & \frac{\partial}{\partial x_4} f_3 \\ \frac{\partial}{\partial x_1} f_4 & \frac{\partial}{\partial x_2} f_4 & \frac{\partial}{\partial x_3} f_4 & \frac{\partial}{\partial x_4} f_4 \end{bmatrix} \quad B = \begin{bmatrix} \frac{\partial}{\partial u} f_1 \\ \frac{\partial}{\partial u} f_2 \\ \frac{\partial}{\partial u} f_3 \\ \frac{\partial}{\partial u} f_4 \end{bmatrix} \quad C = \begin{bmatrix} \frac{\partial}{\partial x_1} g_1 & \frac{\partial}{\partial x_2} g_1 & \frac{\partial}{\partial x_3} g_1 & \frac{\partial}{\partial x_4} g_1 \\ \frac{\partial}{\partial x_1} g_1 & \frac{\partial}{\partial x_2} g_1 & \frac{\partial}{\partial x_3} g_1 & \frac{\partial}{\partial x_4} g_2 \end{bmatrix}$$

$$D = \begin{bmatrix} \frac{\partial}{\partial u} g_1 & \frac{\partial}{\partial u} g_2 \end{bmatrix}$$

Nel specifico del veicolo autobilanciato, nel caso con utente con peso di *70 kg* e altezza *1.77 m*, siamo andati ad ottenere i seguenti valori numerici per le matrici in esame:

$$A = \begin{bmatrix} 0 & 1.0 & 0 & 0 \\ 0 & 0 & -15.2286 & 0 \\ 0 & 0 & 0 & 1.0 \\ 0 & 0 & 5.43924 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 2.7498 \\ 0 \\ -0.2612 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

Ricordando infine che un generico sistema dinamico può essere scritto come:

$$G(s) = C(\mathrm{sI} - A)^{-1}B + D$$

sostituendo i valori numerici matriciali ottenuti al passo precedente, possiamo ottenere il sistema di equazioni, le quali rappresentano l'insieme delle funzioni di trasferimento caratterizzanti il sistema:

$$\begin{pmatrix} \frac{2.75}{s^2} - \frac{1.749\mathrm{e}{+13}}{2.392\mathrm{e}{+13}\,s^2 - 4.398\mathrm{e}{+12}\,s^4} \\ \frac{2.75}{s} - \frac{1.749\mathrm{e}{+13}}{2.392\mathrm{e}{+13}\,s - 4.398\mathrm{e}{+12}\,s^3} \\ -\frac{1.149\mathrm{e}{+12}}{4.398\mathrm{e}{+12}\,s^2 - 2.392\mathrm{e}{+13}} \\ -\frac{1.149\mathrm{e}{+12}\,s}{4.398\mathrm{e}{+12}\,s^2 - 2.392\mathrm{e}{+13}} \end{pmatrix}$$

Si può notare come la prima e la seconda riga della matrice differiscano solo per un fattore derivativo, così come la terza e quarta riga; questo è ovvio ed atteso in quanto x_2 è a derivata di x_1 e x_4 la derivata di x_3.

Nello specifico, queste funzionalità di linearizzazione, sono state racchiuse all'interno del *live script* Matlab nominato *VAB_simulazioni_discrete.mlx*: in questo modo siamo stati in grado di rendere più efficace ed efficiente la prosecuzione della simulazione, non dovendo eseguire ogni volta anche questa parte di calcolo matematico relativo alla linearizzazione. Siamo andati infatti a sfruttare le funzionalità di salvataggio offerte da Matlab, per salvare in un file *.mat* l'intera *workspace* (file che abbiamo nominato *WS_VAB_wb_0_5.mat*), in maniera tale da permettere, agli script che ne avessero bisogno, di aprire il workspace e leggere tutte le variabili di interesse.

3

Controllo

3.1 Introduzione

Per rendere maggiormente leggibile il codice Matlab e, allo stesso tempo, ridurre i tempi necessari per l'esecuzione, abbiamo deciso di dividere le varie funzionalità in diversi *livescript*: in particolar modo la parte relativa allo studio ed alla definizione del controllore, siamo andati ad inserirla all'interno del file *GainCalculator.mlx* dove si può vedere che. come prima istruzione, si va a caricare dal workspace, risultante dalla linearizzazione, le matrici A-B-C-D che definiscono il sistema lineare modellizzato.

3.2 Analisi in anello aperto

Prima di procedere con l'individuazione del controllore più adatto per stabilizzare il V.A.B., siamo andati ad effettuare una breve analisi sul sistema ad anello aperto, per ottenere così alcuni spunti sulla correttezza del modello che era stato steso fino a quel punto.

Come noto dalla teoria, il sistema risulta essere asintoticamente stabile se gli autovalori della matrice A risultano avere tutti parte reale negativa; è instabile invece se è presente almeno un autovalore con parte reale strettamente positiva.

Abbiamo quindi calcolato gli autovalori con il comando seguente:

```
6 eig(A_real)
```

Listing 3.1: Calcolo autovalori matrice A (non simbolica)

La presenza di autovalori con parte reale strettamente negativa è stata confermata anche dalla visualizzazione del luogo delle radici (*rlocus*) del sistema in anello aperto: come si vede in figura 3.1, il polo che si trova nel semi-piano reale positivo non potrà, in alcun modo, essere stabilizzato, ovvero portato nel semi-piano sinistro.

Questa breve analisi ci ha permesso di avere una conferma numerica del fatto ovvio che, senza un controllo, il V.A.B. in esame non riesce a mantenere da solo la posizione verticale.

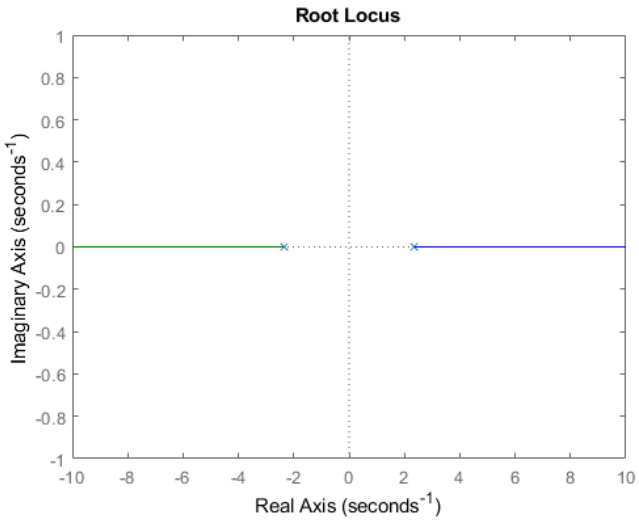

Figure 3.1: Luogo delle radici del V.A.B. modellizzato in anello aperto

3.3 Controllore

Per quanto riguarda il sistema del veicolo autobilanciato, siamo andati a far riferimento a quella classe di problemi definiti come *problemi di regolazione*, in cui il sistema si presenta inizialmente con una condizione iniziale dello stato non nulla, condizione che si intende di riportare a zero con una velocità di convergenza assegnata.

Nello specifico, questo tipo di problema relativo alla scelta del controllore, è stato risolto mediante l'utilizzo dell'assegnazione degli autovalori ottenuti tramite retroazione dello stato, in cui il moto del sistema è composto completamente dal moto libero che si vuole controllare e annullare in un tempo a piacere.

Il posizionamento dei poli, e quindi la scelta del guadagno del regolatore, va eseguita sul solo sistema lineare: questo a conferma sia di quanto riportato in figura 3.2 (nel blocco di colore blu infatti è riportato il sistema lineare), sia di quanto detto in precedenza in merito al fatto che il controllore venga ad essere definito lavorando sul sistema linearizzato.

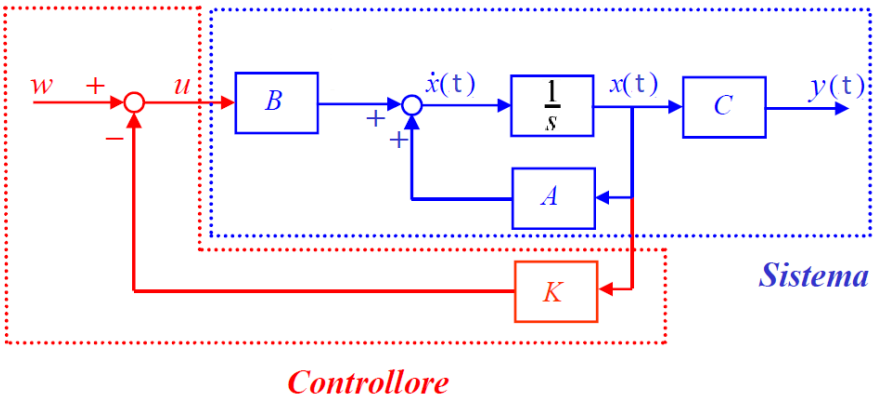

Figure 3.2: Schema concettuale del controllo ([1])

Ciò che è riportato graficamente in figura 3.2, può essere riscritto matematicamente come segue:

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases}$$

$$u(t) = -Kx(t) + w(t) = \text{ legge di controllo}$$

Dunque, sostituendo la legge di controllo nell'equazione di stato del sistema, possiamo ottenere le seguenti equazioni che ci permettono di definire il legame del sistema in anello chiuso con la matrice K:

$$x(t) = Ax(t) + Bu(t) = Ax(t) + B\big(-Kx(t) + w(t)\big)$$

$$= Ax(t) - BKx(t) + Bw\,(t) = (A - BK)x(t) + Bw(t)$$

$$A_{cl} = A - BK = \text{ matrice di stato in anello chiuso (cl} \rightarrow \text{closed loop)}$$

3.3.1 Posizionamento dei poli continui

Per scegliere il valore da assegnare a K, e quindi definire il guadagno del controllore, è necessario scegliere la posizione desiderata dei poli, in base a quelle che sono le specifiche di velocità desiderate.

$$G(S) = \frac{\omega_n^{\,2}}{s^2 + 2\xi\omega_n s + \omega_n^{\,2}}$$

$$\xi = 0.7$$

$$\omega_n = 2\pi f_{\text{propria}}$$

Dovendo posizionare due coppie di poli complessi coniugati si sono scelte due frequenze ad una decade di distanza, in maniera tale da poter considerare le variabili che sono controllate disaccoppiate in frequenza: essendo il limite di banda (ovvero entro quali limiti il segnale passa senza essere attenuato e/o modificato) del sistema interno molto più alto del limite di banda del sistema esterno, possiamo considerarli disaccoppiati in frequenza.

$$f_{\text{propria}\theta} = 0.2 \text{ Hz}$$

$$f_{\text{propria}\phi} = 0.02 \text{ Hz}$$

TODO Nello specifico, una prima osservazione, è il fatto che il polo più veloce (ovvero quello con frequenza pari a 0.2 Hz) è stato assegnato alla parte relativa al controllo dell'angolo θ, essendo che è auspicabile un controllo maggiormente reattivo per quanto riguarda la stabilizzazione della base, piuttosto che il rapido raggiungimento del set point di posizione spaziale (e quindi angolare ϕ delle ruote).

Una seconda osservazione è che, motivi esterni, il motore risulta avere una coppia massima molto limitata (per via delle limitazioni di corrente): è stato dunque necessario posizionare i poli ad una frequenza tale che permettesso di non saturare per molto tempo la coppia fornita dal motore. Questa scelta ha inevitabilmente rallentato la risposta del sistema.

Definita quindi la frequenza a cui posizionare i poli, siamo andati a risolvere l'equazione al denominatore della forma generica della f.d.t (G(s)) con due poli complessi: la risoluzione di questa equazione, come è noto dalla teoria, permette di ottenere i valori della variabile di Laplace s che azzerano il denominatore stesso, che corrisponde a risolvere questa equazione (in forma generica):

$$s^2 + 2\,\xi\,s\,w_n + {w_n}^2 = 0$$

Nella nostra simulazione i valori numerici ottenuti per i poli sono stati i seguenti:

$$polo_\theta = \begin{pmatrix} -\frac{7\,\pi}{250} + \frac{\pi\sqrt{51}\,\mathrm{i}}{250} \\ -\frac{7\,\pi}{250} - \frac{\pi\sqrt{51}\,\mathrm{i}}{250} \end{pmatrix}$$

$$polo_\phi = \begin{pmatrix} -\frac{7\,\pi}{25} + \frac{\pi\sqrt{51}\,\mathrm{i}}{25} \\ -\frac{7\,\pi}{25} - \frac{\pi\sqrt{51}\,\mathrm{i}}{25} \end{pmatrix}$$

Con il comando *place* di Matlab si ottiene dunque:

$$K = [-0.0023, -0.0278, -28.1397, -7.7022]$$

Ricordando a questo punto quanto detto nella prima parte di questo capitolo in merito alla definizione della matrice A in anello chiuso, siamo andati a definirla appunto come

$$A_{cl} = A - BK$$

utilizzando il valore di K appena trovato: una volta trovata la nuova matrice A, siamo andati a ri-effettuare le stesse analisi portate avanti anche per il sistema in anello aperto. Come si vede in figura 3.3, grazie al posizionamento degli autovalori, siamo stati in grado di spostare i nuovi poli nel semi-piano sinistro: questo significa quindi che siamo riusciti a stabilizzare il nostro sistema dinamico.

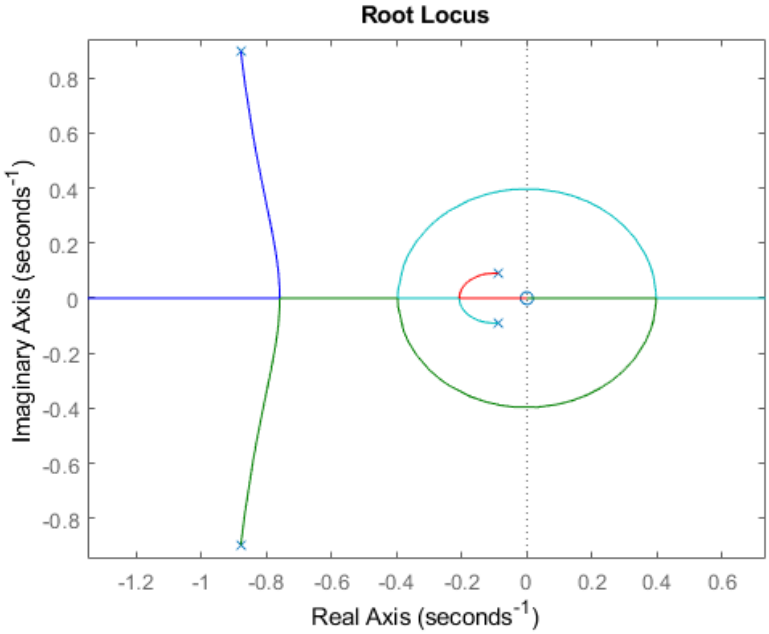

Figure 3.3: Root locus del sistema in anello chiuso

3.3.2 Posizionamento dei poli discreti

TODO

Nello sviluppo del sistema, come presenteremo poi nella sezione di simulazioni (sezione TODO), siamo andati anche ad inserire il passaggio a tempo discreto, a seconda di quelle che sono le specifiche temporali di campionamento del controllore.

Nel definire quindi un controllo a tempo discreto è stato necessario anche andare a riportare il posizionamento dei poli in ambito discreto.

3.3.3 Setpoint di velocità

Oltre al controllo del moto libero del sistema è stata implementata anche la possibilità di inserire un setpoint sulla velocità a fine transitorio del Segway. In questo modo si apre la possibilità per l'utente finale di impostare la velocità desiderata e di mantenerla nel tempo nonostante i vari disturbi che in un sistema reale influiranno sulla macchina(vento, salita, discesa..)

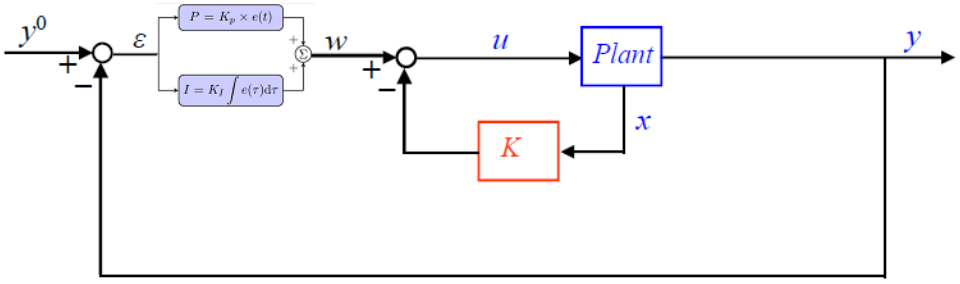

Figure 3.4: chiusura dell'anello di controllo di $\dot{\phi}$ con un controllore I [1]

In Fig.3.4 si nota come l'anello di $\dot{\phi}$ sia esterno rispetto all'anello di retroazione dello stato; solitamente si procede dunque nel progettare il controllore avendo cura di lasciare una decade di spazio tra la frequenza del polo dell' integratore e la coppia di poli più lenta del controllore della retroazione dello stato. In questo caso, per ragioni puramente pratiche, non è stato possibile : il sistema presenta una risposta particolarmente lenta agli input viste le limitazioni di coppia di cui si approfondirà successivamente; sarebbe quindi stato necessario quindi troppo tempo per raggiungere il setpoint di velocità se si fosse proceduto a lasciare una decade di distanza. Si è invece proceduto a posizionare il polo dell'integratore tramite la funzione di Tuning offerta da Matlab Simulink stesso:

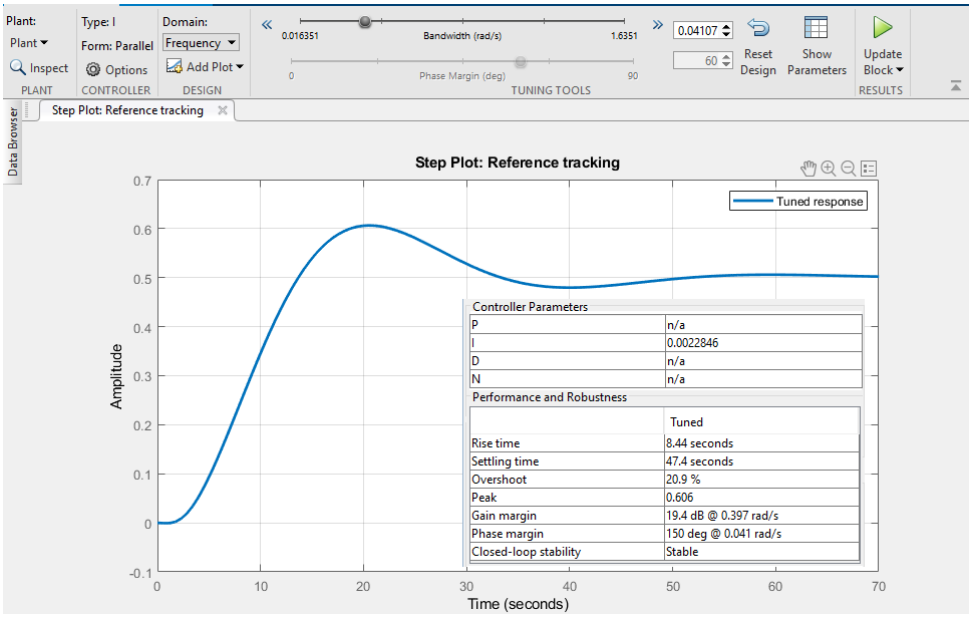

Figure 3.5: Taratura del controllore

Il tuning è stato effettuato sul sistema lineare in quanto Matlab richiede un sistema lineare per questo tipo di tecniche. Da notare inoltre, in Fig.3.5 che la frequenza $f_3 = \dfrac{0.041 rad/s}{2\pi} = 0.0065 Hz$ che è circa $\dfrac{1}{3} f_{\text{propria}\phi}$. Sempre in Fig.3.5 si osserva come comunque, non avendo rispettato la decade di distanza, il sistema sia comunque molto lento e arrivi a regime in circa 70 s. TODO: perchè arriva a 0.5? per la retroazione interna?

3.4 OPC - UA

Per quanto riguarda la parte di controllo, il sistema presenta un articolato insieme di controllori inter-operanti tra di loro: nello specifico ad ogni singolo controllore sono affidate delle mansioni ben specifiche, tutte ovviamente volte al controllo e alla stabilizzazione del *veicolo auto bilanciato*.

In questa fase dello sviluppo del progetto, siamo andati ad implementare parte del codice che verrà installato, in un secondo momento, a bordo del raspberry: esso infatti svolge, all'interno del sistema

(come si vede in figura 3.6) una comunicazione a due direzioni, che ne determinano due comportamenti differenti:

- Come **server** per la parte di comunicazione *OPC-UA* (per il settaggio dei guadagni);
- Come **master** nella comunicazione seriale verso Arduino (per quanto riguarda invece la gestione dell'algoritmo di controllo);

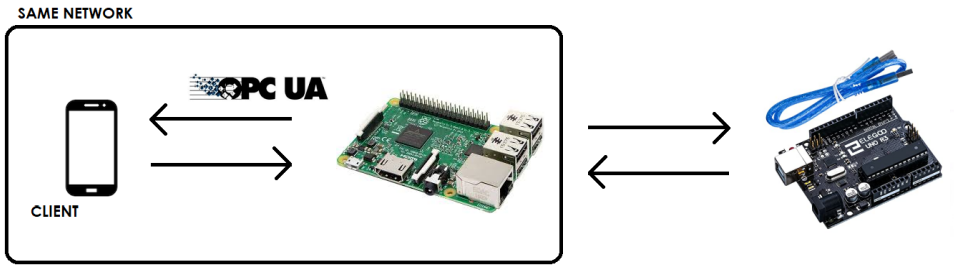

Figure 3.6: Schema di massima dell'utilizzo di Raspberry Pi 3

In questa fase abbiamo quindi sviluppato la parte relativa all'utilizzo di *Raspberry Pi 3* come *server OPCUA*.

3.4.1 Idea base OPC-UA

L'*Open Platform Communications Unified Architecture* (OPC UA) è un protocollo di comunicazione automatico per l'automazione industriale.

OPC UA sostituisce il protocollo *OPC Classic*, conservando tutte le funzionalità del predecessore. Poiché OPC Classic era stato costruito su una tecnologia Microsoft detta modello a oggetti per componenti distribuiti, era vincolato a Microsoft, ma questa caratteristica è diventata sempre più limitante.

OPC UA risulta completamente interoperabile tra i diversi sistemi operativi usati, aggiungendosi a Windows e alle tecnologie industriali come i PLC, inoltre comprende Linux, iOS e anche sistemi operativi per dispositivi mobili come Android. Consentire al maggior numero di dispositivi possibile di comunicare contribuisce al progresso dell'IoT.

Queste caratteristiche di **interoperabilità** sono state sfruttate al massimo in questo contesto, potendo così creare, in maniera semplice e veloce, una comunicazione tra differenti tipologie e famiglie di dispositivi.

3.5 OPC-UA e V.A.B.

Nel contesto del progetto del *veicolo auto bilanciato*, siamo andati ad utilizzare il protocollo di comunicazione *OPC-UA* come supporto per il tuning dei parametri relativi al **gain** del controllore, ovvero ai parametri del vettore K, che abbiamo chiamato (all'interno dello script di Python):

- **K_phi**
- **K_phi_p**
- **K_theta**
- **K_theta_p**

Nello specifico, lo scambio di parametri tra server e controllore avviene tramite unfile *.txt*, che permette, in maniera semplice e immediata, di implementare uno scambio di informazioni tra il server *OPC-UA* strutturato in Python e l'ambiente *real-time* introdotto a bordo del controllore *Raspbery Pi 3*.

In particolare abbiamo due files:

- **Un file temporaneo** *("GainParametersToController.txt")* utilizzato come pipeline per il passaggio dei parametri tra server e controllore. Questo risulta essere un file temporaneo che viene ad essere cancellato e ricreato ogni qualvolta che il server viene spento e successivamente riaccesso. Nello specifico, ad ogni riaccensione, i valori iniziali di questo file, vengono settati con gli stessi valori presenti nel file *definitivo* (qualora quest'ultimo esista già: in caso contrario si procede con un inizializzazione dei parametri a 0);
- **Un file definitivo** *("GainParametersConfirmed.txt")* il quale invece viene creato una e una sola volta e sul quale poi vengono salvati i parametri che saranno poi letti all'accensione successiva del server ed utilizzati come parametri iniziali per il controllore.

Questi file possono essere settati con i parametri presenti nel server che sono visibili in figura 3.7, nello specifico:

- **Submit change to controller** permette di scrivere i valori dei gains sul file *"GainParametersToController.txt"*;
- **Store definitively in file** permette di scrivere i valori dei gains sul file *"GainParametersConfirmed.txt"*;
- **SHUT DOWN SERVER** permette invece di spegnere il server e di cancellare successivamente il file temporaneo.

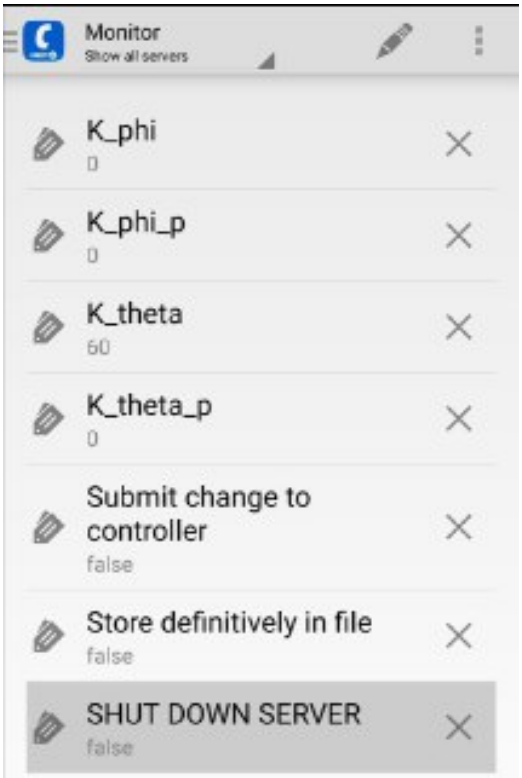

Figure 3.7: Parametri impostabili lato client

Abbiamo racchiuso in figura 3.8 il funzionamento di massima del codice lato server: codice che siamo andati a testare utilizzando un'apposita app per smartphone Android (OPC-UA Android client); per quanto riguarda invece l'effettiva implementazione, abbiamo riportato nell'Appendice A l'intero codice prodotto, per la gestione della comunicazione OPC-UA.

3.6 Rumore

Piattaforma inerziale –> PSD Questione anti windup –> coppia già limitata quindi nessun vincolo di saturazione

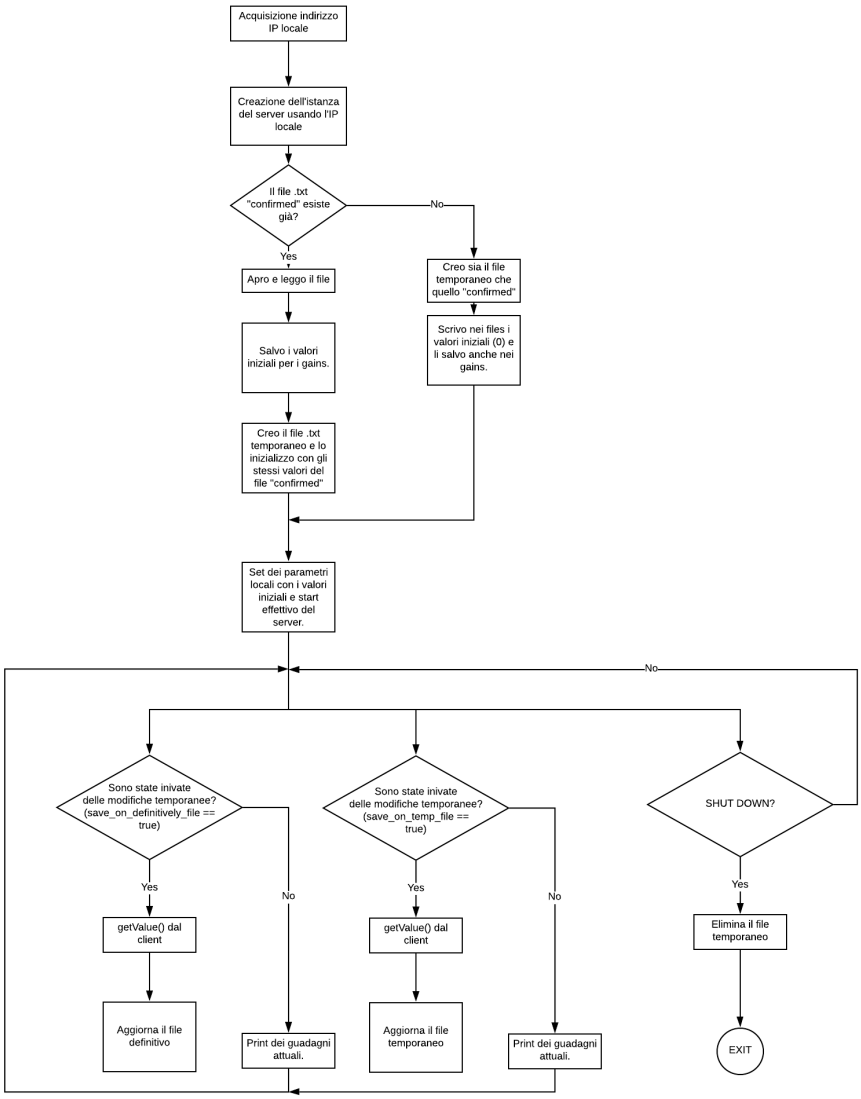

Figure 3.8: Diagramma rappresentante le funzionalità del server

4

Simulink

4.1 Introduzione

TODO: rivdere questa introduzione (io la rivedrei)

Data la straordinarietà degli eventi che erano in corso dal punto di vista sanitario nel nostro paese si è reso necessario svolgere gran parte del lavoro in modalità a distanza e quindi senza la possibilità di testare il modello matematico appena ottenuto e i successivi risultati dovuti all'azione di controllo sul V.A.B., il lavoro quindi è stato svolto per la maggior parte sfruttando il tool *Simulink* di Matlab che è, in poche parole, un risolutore di equazioni differenziali.

Abbiamo così adottato una metodologia di lavoro basata su prototipi sempre più simili a quello che dovrebbe essere il sistema reale.

Si va ora ad analizzare la risposta del sistema al controllo ottenuto nei punti precedenti, per assicurarci, che quanto scritto sopra valga oltre che nella teoria anche nella pratica:

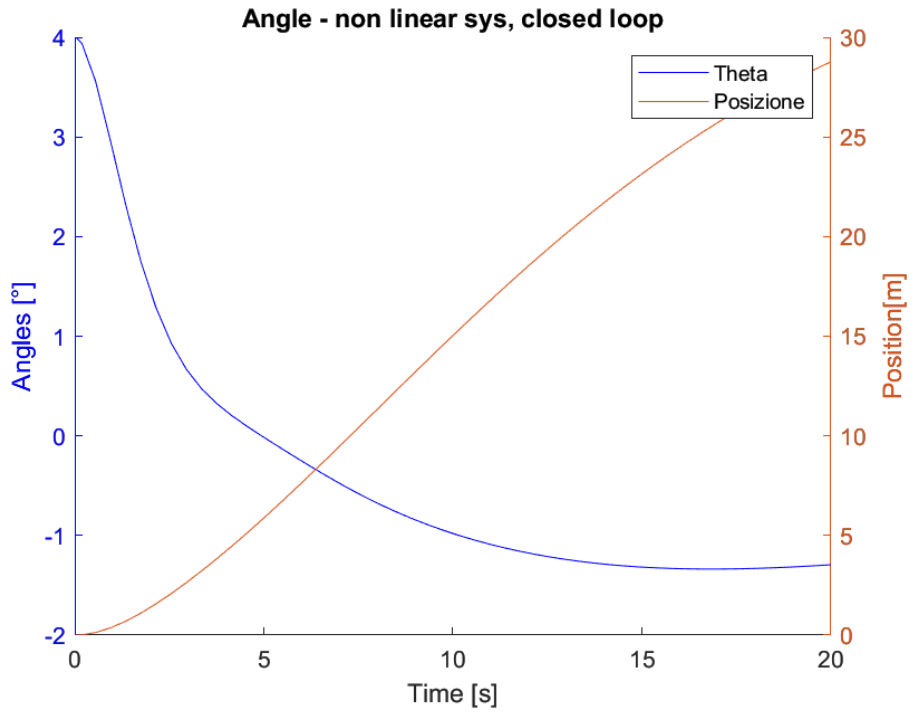

Figure 4.1: Risposta del sistema non lineare in anello chiuso

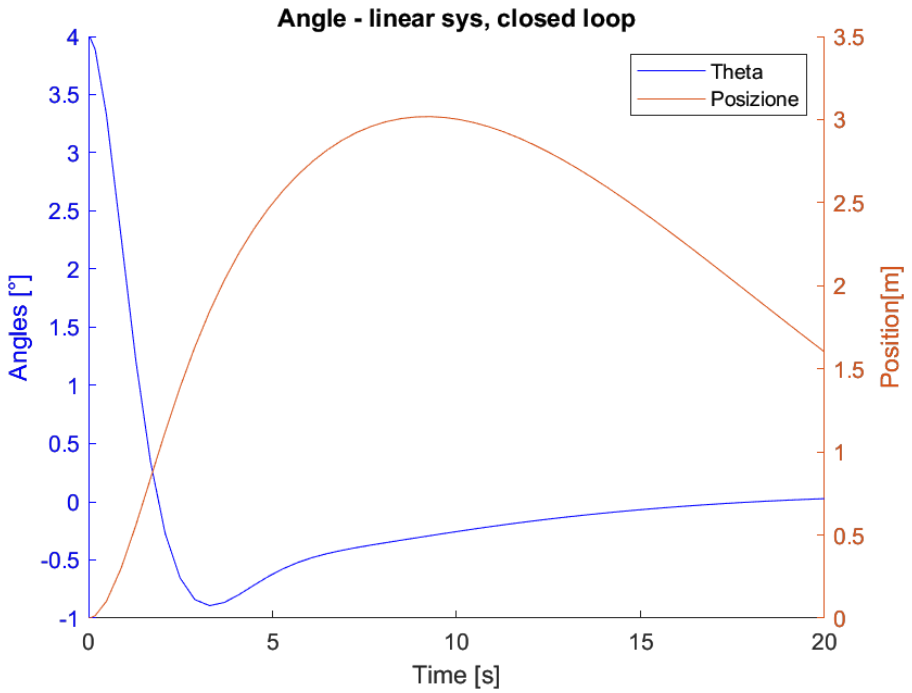

Figure 4.2: Risposta del sistema lineare in anello chiuso

TODO: come spieghiamo la differenza? rifare la simulazione?

4.2 Simulazione del sistema non lineare

Il primo compito che abbiamo risolto è stato quello di implementare le equazioni differenziali ottenute nel capitolo precedente:

- $\ddot{\phi} = f_{\ddot{\phi}}(M_c, \theta, \dot{\theta}, C_m)$
- $\ddot{\theta} = f_{\ddot{\theta}}(M_c, \theta, \dot{\theta}, C_m)$

Dove M_c sarebbe la massa del passeggero, $\theta e \dot{\theta}$ lo stato del sistema e C_m la coppia erogata dal motore. Si può notare come entrambe le equazioni differenziali siano indipendenti dalla coordinata libera ϕ.

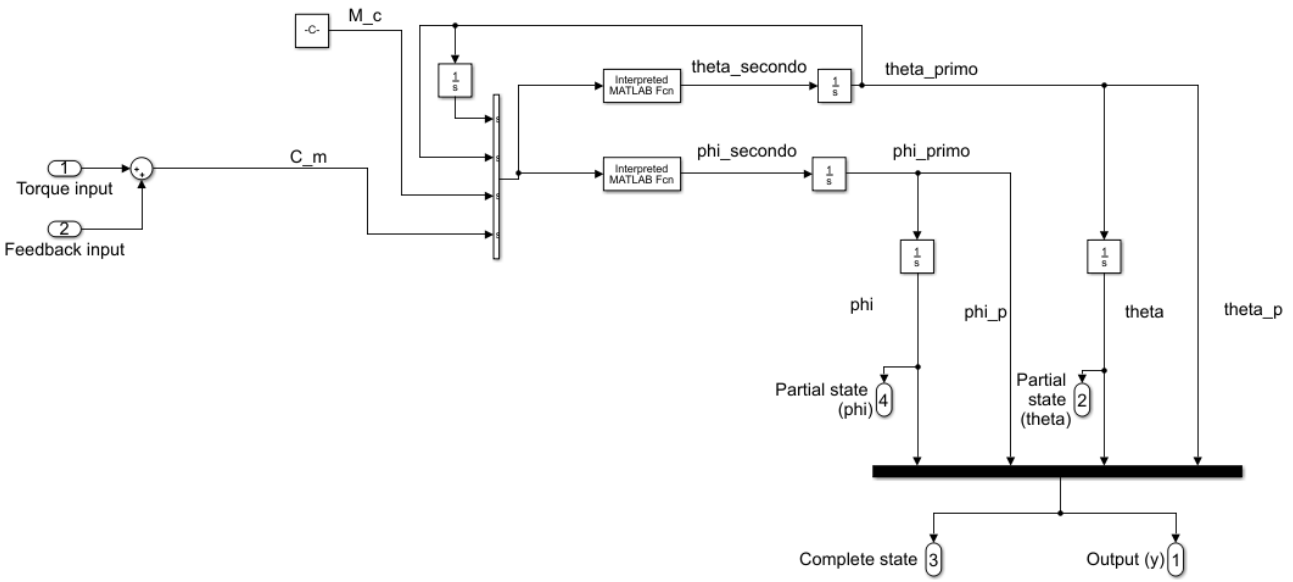

Figure 4.3: Implementazione simulink delle equazioni differenziali

In Fig.4.3 le *interpreted function* altro non sono che $f_{\ddot{\phi}}$ e $f_{\ddot{\theta}}$. A valle di esse sono presenti degli integratori che permettono di ottenere lo stato x completo del sistema. Si può facilmente notare come $\dot{\theta} e \theta$ siano collegate direttamente all'input delle *interpreted function*. Si è dunque proceduto a simulare il sistema per verificare la bontà di quanto ottenuto; in particolare, il sistema in anello aperto, dovrebbe oscillare all'infinito vista la mancanza di attriti nel modello.

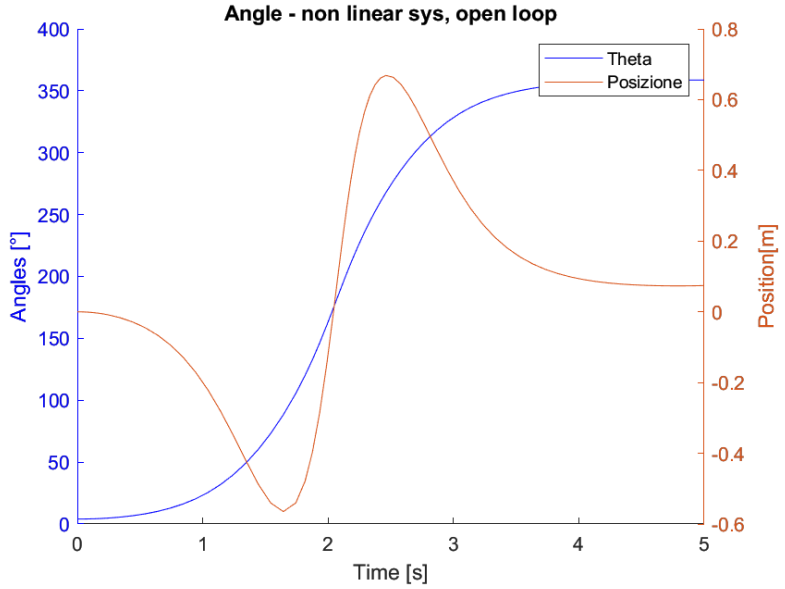

Figure 4.4: Risposta in anello aperto del sistema reale

La simulazione, il cui risultato è riportato in Fig.4.4 è stata svolta per 5 secondi e con un angolo iniziale di 4°; il grafico mostra dunque l'andamento di θ nel tempo e della posizione che in termini matematici si esprime come $posizione = \phi \cdot r_{ruota}$

4.3 Simulazione del sistema lineare

Si è inoltre creato un altro modello sfruttando il sistema lineare ottenuto prima con l'obbiettivo di semplificare il problema e di velocizzare le simulazioni con lo scopo di testare rapidamente nuove tecniche di controllo che se avessero dato esito positivo sul modello lineare sarebbero poi state testate sul simulink che imita il comportamento reale del sistema.

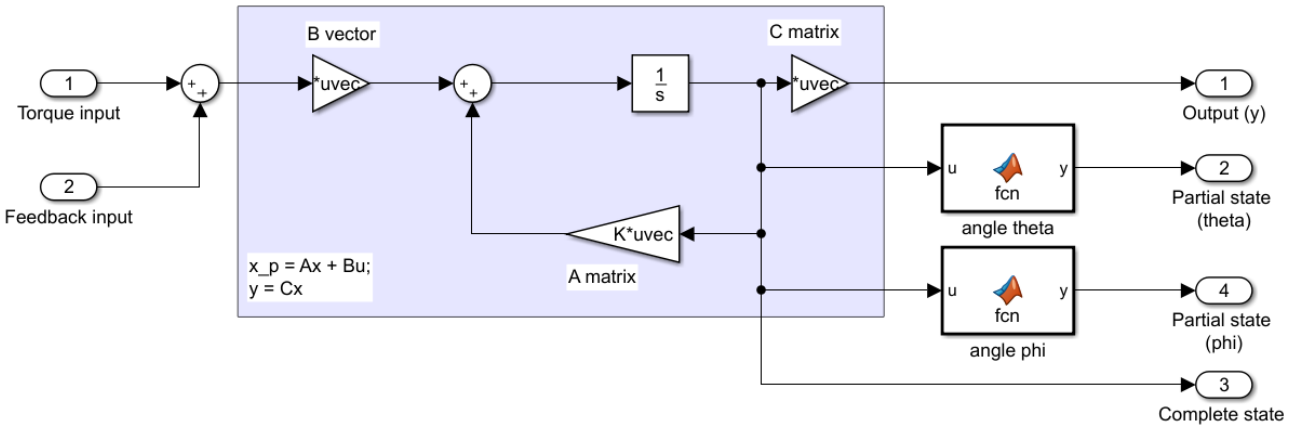

Figure 4.5: Implementazione simulink del sistema linearizzato

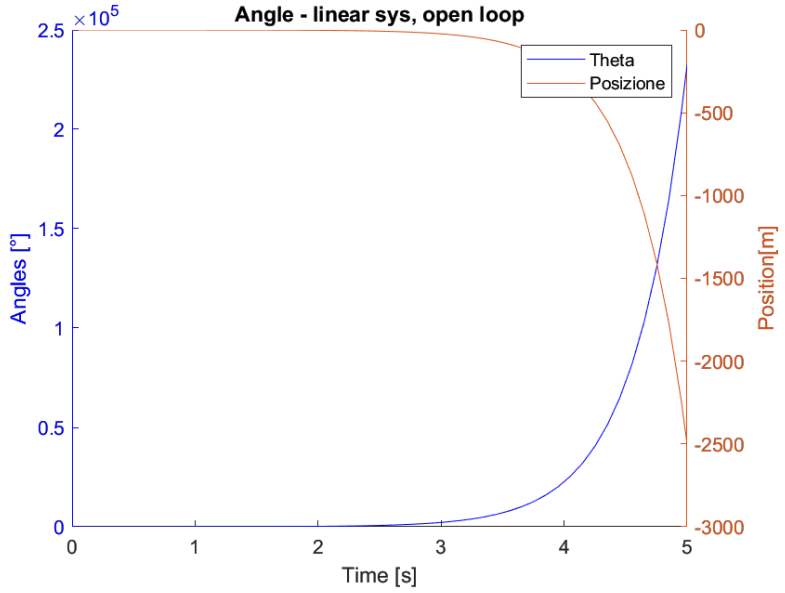

Figure 4.6: Risposta in anello aperto del sistema lineare

In questo caso, la simulazione in anello aperto mostra che il sistema diverge; questo perché la linearizzazione ha senso attorno al punto di equilibrio da cui è stata ottenuta, distante da quel punto il sistema lineare non approssima più il sistema reale ed anche un eventuale controllo ottenuto da esso non garantisce buone performance distante da quel punto. Si nota, in Fig.4.6 che il punto di partenza è 4° e il sistema, lineare, diverga quasi immediatamente; la differenza con la risposta del sistema non lineare in Fig.4.4

Come già verificato in precedenza (sezione 3.2) ci sono due poli reali, uno negativo e uno positivo; questo dimostra come il sistema sia instabile in anello aperto e necessiti di controllo.

4.4 Motore

Il passo successivo è stato quello di andare a modellizzare il motore presente a bordo dello chassis; questo è necessario in quanto si deve tenere conto, in primo luogo, del ritardo che gli attuatori (cioè il motore) introducono nel sistema e che per questo potrebbe diventare instabile.

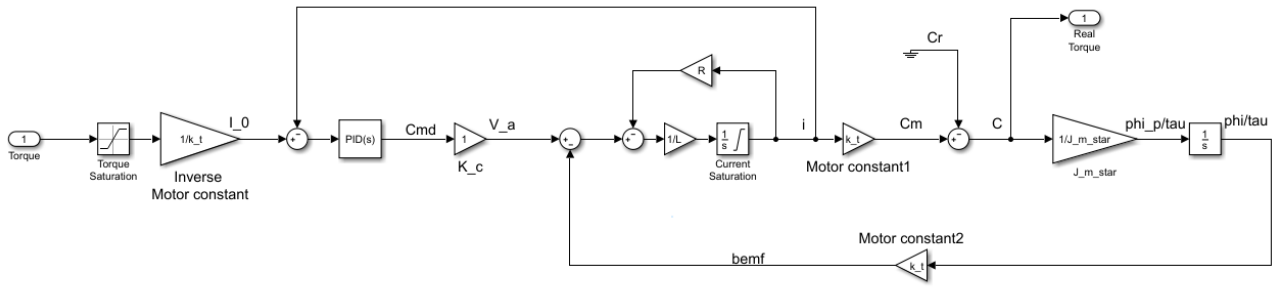

Figure 4.7: Schema a blocchi del motore

I valori delle componenti in Fig.4.7 sono presi dal datasheet del motore. Il controllo del motore DC in questione e stato ottenuto tramite una retroazione in corrente che permette quindi di definire un setpoint alla corrente presente nel motore. Questo si è reso necessario poiché il controllore, attraverso il vettore K e lo stato del sistema, definisce la coppia che il motore dovrebbe erogare. In un motore DC la correlazione tra coppia erogata e corrente esiste ed è ben definita e si tratta di k_t. Il controllore è stato realizzato seguendo metodi già noti in letteratura TODO: scrivere la formula o trovare un posto che la spieghi

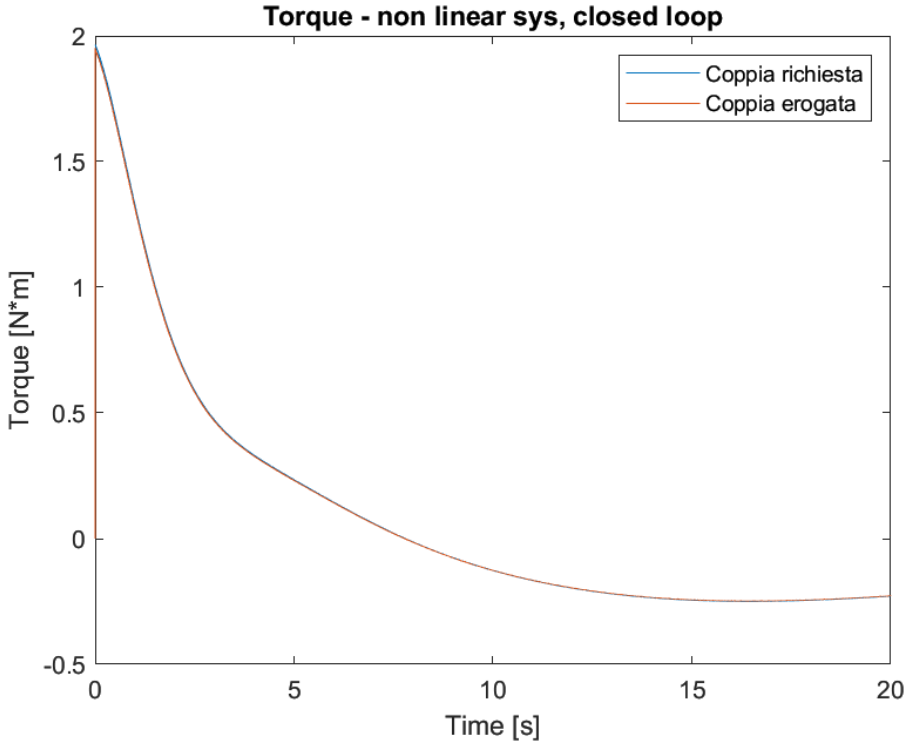

Figure 4.8: Transitorio del motore

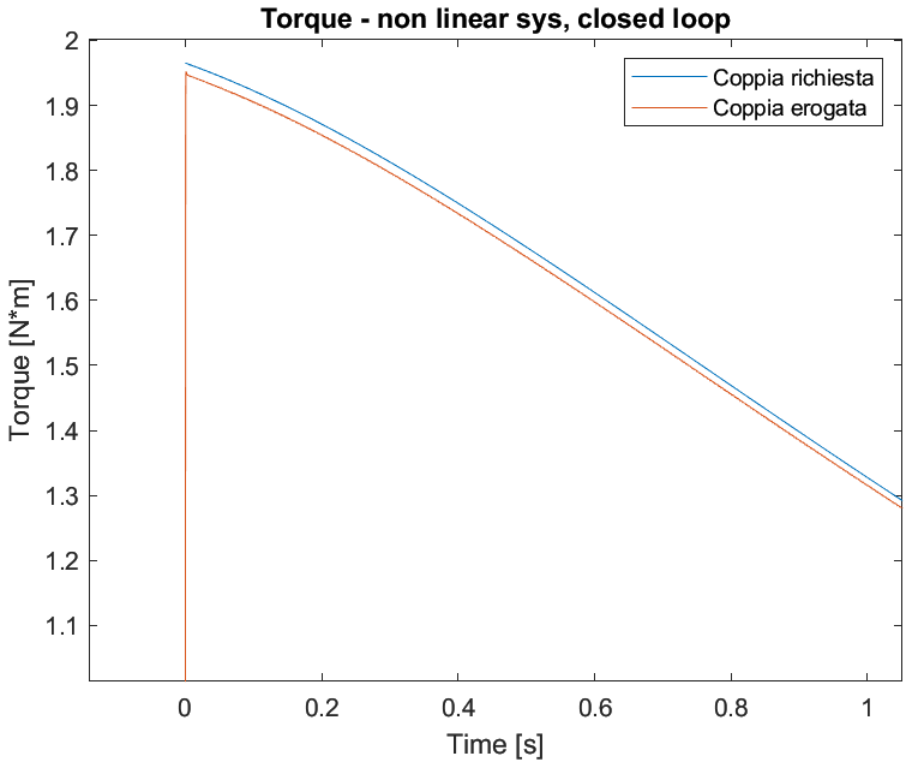

Figure 4.9: Zoom del grafico in figura Fig.4.8

Come si può notare il picco di coppia massimo è minore di 2; questo è dovuto al fatto che, per ragioni esterne, nel motore può scorrere ua corrente di massimo 20A e quindi:

$$C_{m,max} = I_{max} \cdot K_{motore} = 20A \cdot 10\frac{N \cdot m}{A} = 2N \cdot m$$

Un esempio in cui la coppia richiesta supera i $2N \cdot m$ è il seguente:

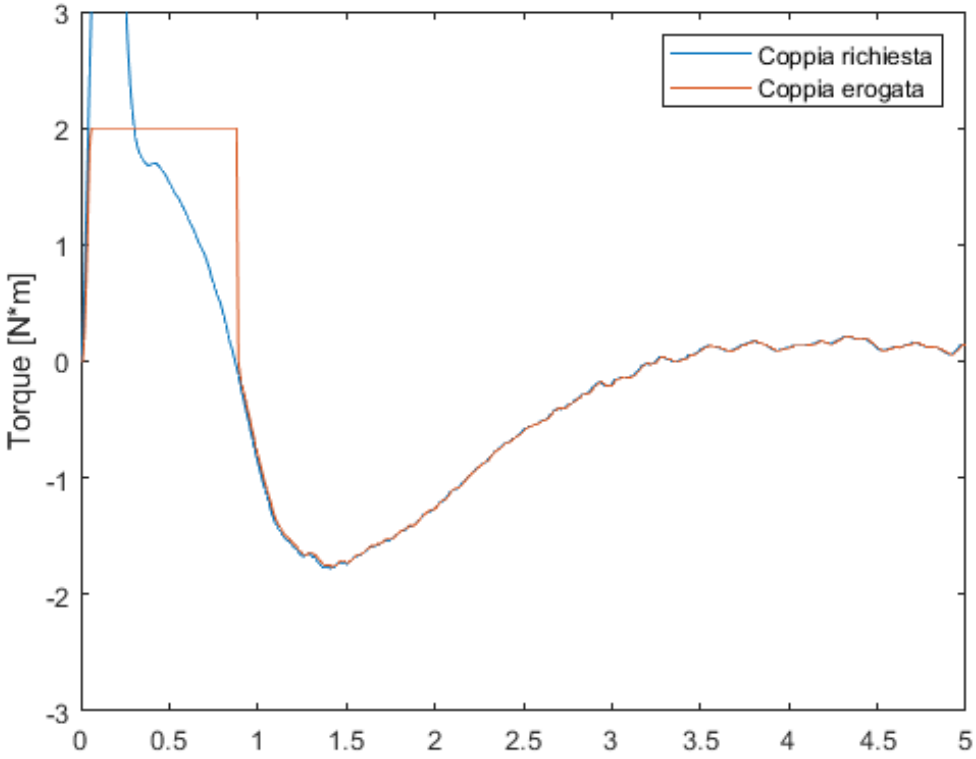

Figure 4.10: Clamp della coppia erogata da parte del motore

In Fig.4.10 è anche possibile notare come l'assenza del blocchetto denominato *Torque saturation*, presente in Fig.4.7, satura l'azione integrale dell'attuatore e inserisce un ritardo non secondario nell'azione di controllo.

4.5 Modello complessivo (al momento)

L'obbiettivo di questo paragrafo è quello di fare il punto della situazione del sistema sviluppato fino a questo punto e di sviluppare alcune considerazioni sul lavoro fatto.

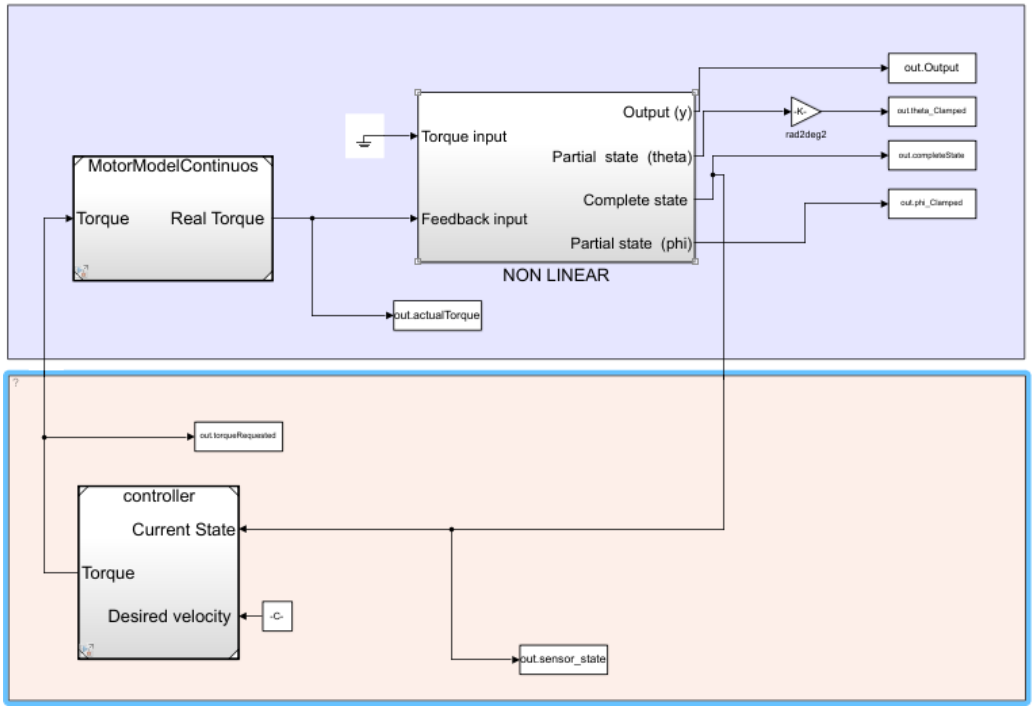

Figure 4.11: Simulink

Come si osserva in Fig.4.11 il sistema al momento comprende tre blocchi:

- Il blocco *non linear*: rappresenta quello che nella realtà sarebbe il sistema reale; si occupa durante la simulazione, dati gli input, di restituire output simili a quelli che si avrebbero in laboratorio utilizzando la macchina vera e propria.
- Il blocco *MotorModelContinuos*: si occupa di simulare la presenza e i transitori dovuti ai motori che nella realtà sono posti a bordo dello chassis.
- Il blocco *controller*: è, dei tre, l'unico blocco che effettivamente dovrebbe essere implementato su un calcolatore. Con i valori simulati dai due blocchi di cui sopra calcola il valore di coppia per il controllo e lo fornisce indietro ai suddetti blocchi per completare la retroazione.

Il sistema al momento è, in linea teorica, a tempo continuo. Nella realtà in un computer e in particolar modo su Simulink la possibilità di far operare i blocchi che simulano il sistema reale a tempo continuo è preclusa. Si é scelto dunque,per quanto fatto finora, di lasciare scegliere al software di Simulink il passo della simulazione ed in particolar modo utilizzare un passo variabile. Questo permette, nei punti in cui le variazioni sono spiccate(ad esempio quando il sistema si avvia) di utilizzare un passo di simulazione anche dell'ordine dei nanosecondi che approssima quasi perfettamente l'esecuzione a tempo continuo.

4.6 Discretizzazione

Come già detto sopra, il controllore è necessario che sia implementato su u calcolatore e quindi verrà eseguito a tempo discreto; per tenere conto di questa caratteristica è necessario discretizzare il controllore e acquisire gli input a tempo discreto:

- all'ingresso del blocco *controller* è stato posto un *Sample & Holder*; questo blocco si occupa di acquisire ogni tempo di sampling T_s il valore in input e mantenerlo inalterato fino alla lettura successiva
- all'uscita del *controller* si è posizionato un altro *Sample & Holder* per le stesse ragioni

- il *controller* è stato trasformato a tempo discreto; TODO inseriamo la cosa dei poli discreti o diciamo che sono uguali essendo un proporzionale? A differenza del controllore della retroazione dello stato, il controllore che chiude la retroazione in velocità va ricalcolato tenendo conto del T_s; fortunatamente Simulink fa da solo questa conversione se si setta il blocco simulink *Controllore PID* come integratore a tempo discreto fornendo il T_s e la costante moltiplicativa.

Il controllore a tempo discreto ha dunque questo aspetto nel modello simulink implementato:

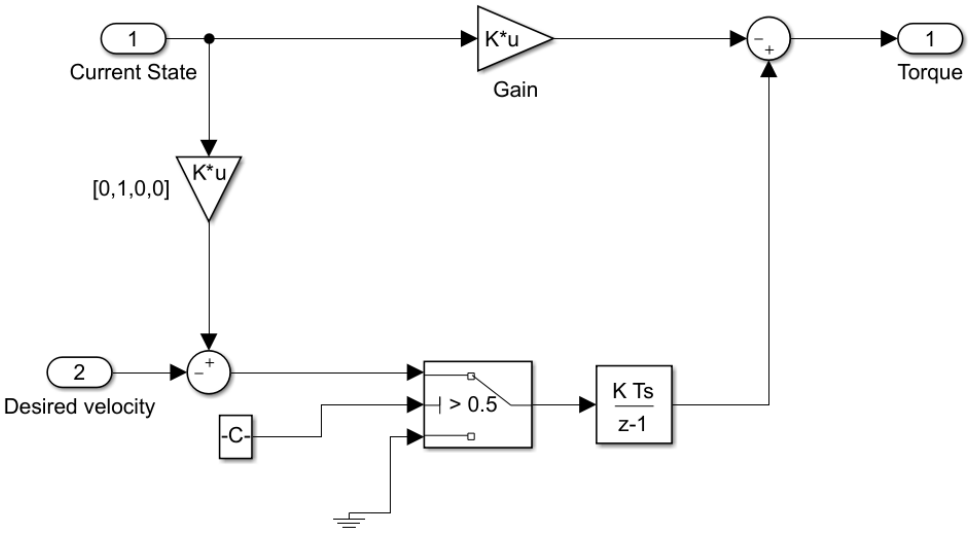

Figure 4.12: Modello Simulink del controllore

Per le stesse ragioni pratiche anche il modello del motore ha subito una conversione: il controllore dell'anello di retroazione in corrente sarà un software eseguito ciclicamente su di un calcolatore e va dunque implementato a tempo discreto. Ricordando che la funzione che la F.d.T. del controllore *PI* a tempo continuo era:

$$\frac{0.001216s + 0.9965}{0.00122s}$$

e applicando il metodo di Tustin da tempo continuo a discreto con un $T_s = 0.001$ si ottinene che:

$$\frac{1.405z - 0.5881}{z - 1}$$

4.7 Sensori

Sempre con l'obbiettivo di rendere la simulazione più raffinata e accurata possibile sono stati inseriti all'interno del file simulink anche dei modelli che simulano la presenza dei sensori. Questo è necessario ed importante poiché nella realtà non esiste un modello matematico che genera i dati che poi sono dati in pasto al controllore ma si devono invece usare dei sensori che danno una stima dello stato del sistema. Un sensore presenta due principali caratteristiche:

- Rumore: è stato modellizzato come un *Band-Limited White Noise* con *Noise power* = [0.00000001] e *Sample time* = 0.001.
- Quantizzazione: la lettura dei dati da un sensore, oltre ad avvenire ad un intervallo di tempo minimo e regolare, mostra anche un errore dovuto alla limitata sensibilità del sensore stesso o dell' *ADC* che effettua la lettura. Il blocco simulink *Quantizer* ha l'esatto scopo di simulare questo comportamento presente nei sensori reali.

TODO come spiego che mi arriva in ingresso theta e phi e non le derivate? Sul sistema sono presenti diversi sensori:

- Encoder incrementale: si occupa di misurare la rotazione della singola ruota

- I.M.U. : stima l'inclinazione dello chassis
- Sensore di corrente: misura la corrente presente nel motore

Per come si presenta il sistema, sia l'*encoder* incrementale sia la *I.M.U.* restituiscono rispettivamente una stima di ϕ e θ Per modellizzare quanto detto finora riguardante i sensori ed in particolare per l'*encoder* e l'*I.M.U.* è stato approntato il seguente blocco simulink:

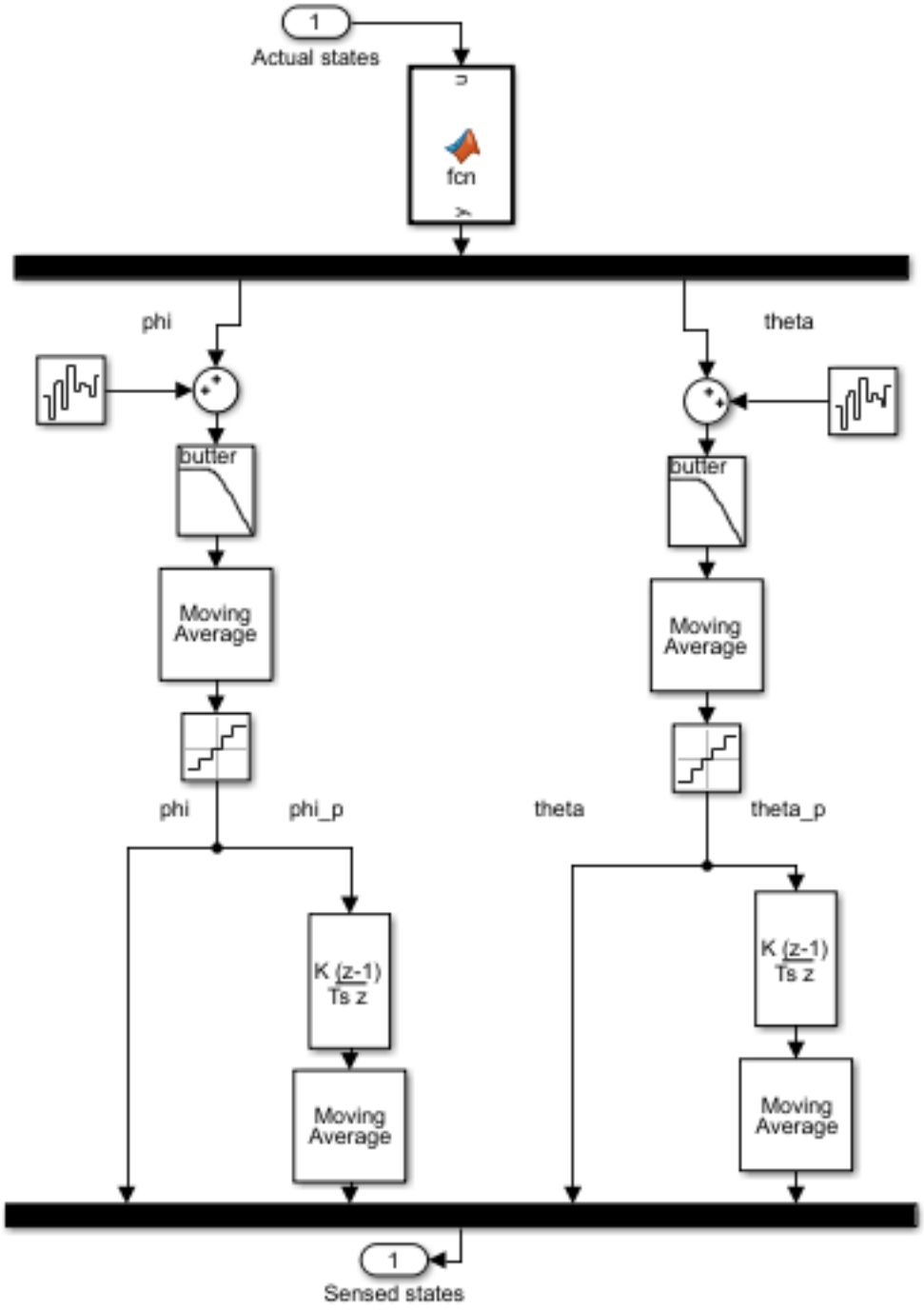

Figure 4.13: Modello Simulink dei sensori *encoder* e *I.M.U.*

Si può notare come in Fig.4.13 siano presenti due blocchi *Moving average* per ciascun sensore; questo si è reso necessario per due ragioni. La prima è che, presentando il sistema del rumore, era necessario rimuoverlo attraverso un filtro digitale; la seconda ragione è che, vista la presenza della quantizzazione, la derivazione a tempo discreto del segnale presenta molti picchi e molti valori nulli; è quindi necessario filtrare questi valori prima di passarli al controllore per evitare picchi di coppie

troppo alti oppure nulli in brevi intervalli di tempo. Si presenta ora la necessità di modellizzare la presenza del sensore di corrente nel motore; per fare ciò si è modificato solamente l'anello di retroazione esterno che permette il controllo del motore in corrente:

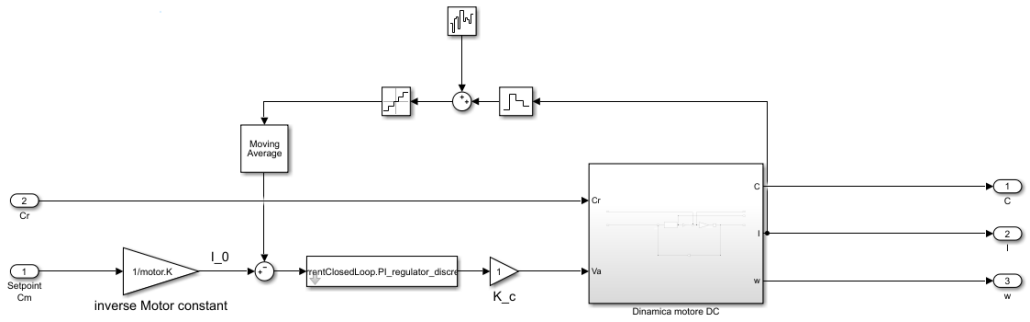

Figure 4.14: Modello Simulink discreto del motore

Si può notare in Fig.4.14 che il controllore è quello discreto presentato prima, mentre è stato inserito anche in questo caso un rumore sulla misura, una quantizzazione e un filtro sul segnale da passare al controllore. Da sottolineare che ognuno dei tre blocchi *Quantizer* ha un valore diverso dovuto al fatto che ognuno dei sensori ha sensibilità diverse:

- Encoder incrementale: presenta una sensibilità di $2^1 1bit = 2048$ diverse combinazioni e letture
- I.M.U. : sensibilità del giroscopio di $131\frac{s}{deg}$
- Sensore di corrente: il valore del sensore di corrente è un voltaggio letto da un Arduino Uno con un ADC con sensibilità di 10 bit

4.8 RTOS - XENOMAI

Riportiamo di seguito una breve descrizione del lavoro portato avanti durante questa settimana per quanto concerne la parte RTOS.

- Come prima cosa siamo andati a rivedere il file *.cpp* che avevamo prodotto nelle settimane scorse. In particolare abbiamo dovuto ritoccare la parte relativa alla lettura dei dati dal file, andando ad utilizzare le API presenti in ambiente c invece di quelle relative a cpp. Abbiamo quindi provato e testato la lettura da file tramite il tool DevC++, facilitando così il testing e il debug;
- Siamo andati ad installare una macchina virtuale con installato a bordo **debian 9.8.0** e la patch **Xenomai 3.0.8** (link);
- Essendo debian un OS interamente fruibile da linea di comando, siamo andati a prendere confidenza con l'ambiente, andando a capire l'organizzazione e chiamando gli update/upgrade necessari;
- Prima di andare ad importare il file completo (di cui abbiamo parlato al punto iniziale), siamo andati ad eseguire alcuni esempi iniziali (una sorta di HelloWorld):
  - Abbiamo per primo cosa studiato questo approccio iniziale (link)
  - Siamo andati ad impostare il funzionamento di un task (dummy) periodico, come se si trattasse appunto del main loop di Arduino (link)

- Lo step successivo è stato quello di importare il codice scritto e testato in ambiente DevC++ in xenomai, provandolo e introducendo alcuni dettagli aggiuntivi:

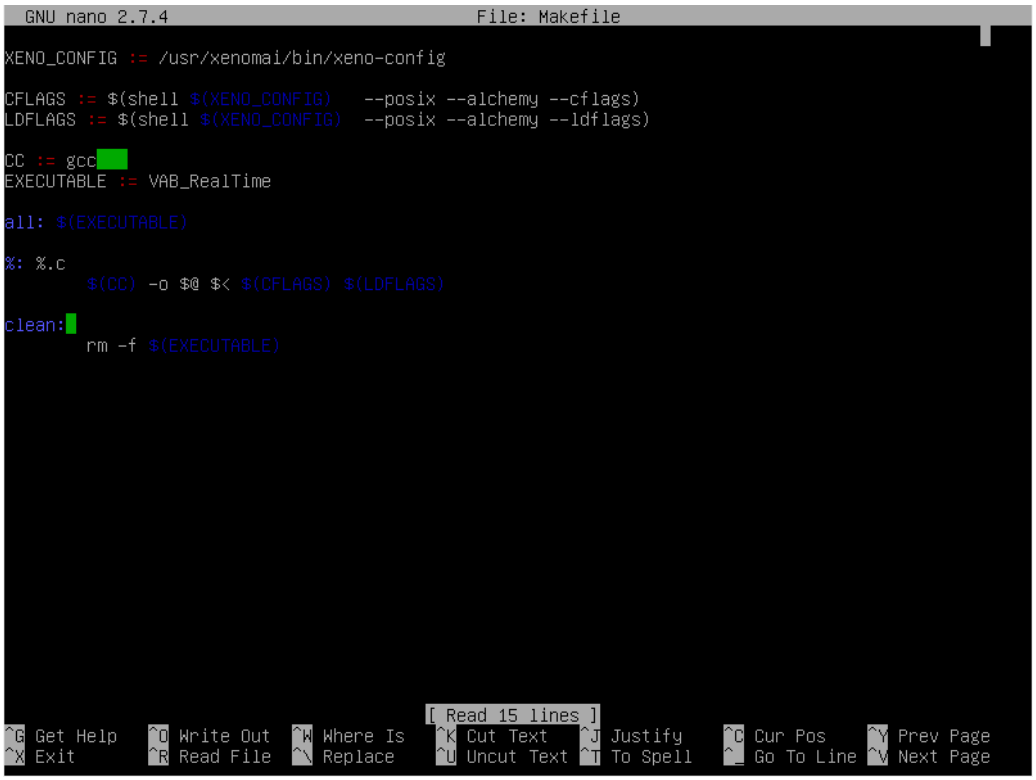

Figure 4.15: Per specificare quale file *.c* compilare

– Periodo di funzionamento definito tramite questi comandi

```
RTIME period  = 1000000000; % Expressed in [ns]
rt_task_set_periodic(NULL,TM_NOW,period);
```

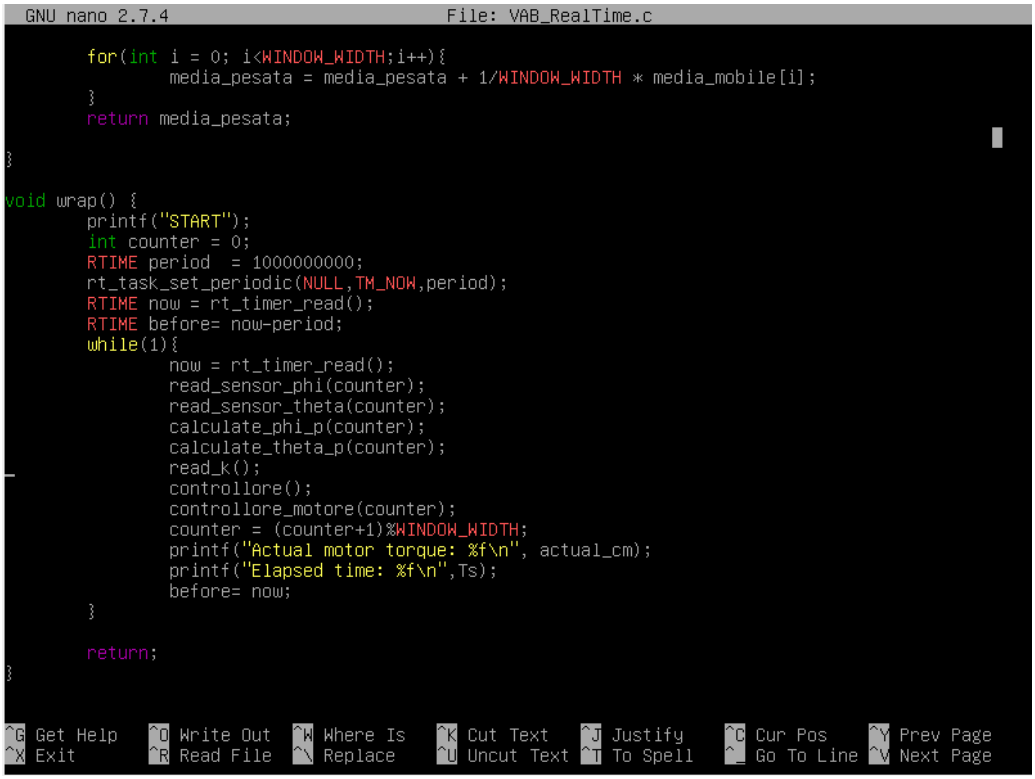

Figure 4.16: Definizione e temporizzazione del *WHILE*

– Salvataggio dei parametri temporali now ad inizio del ciclo e update del parametro before alla fine del ciclo: questo per facilitare, in qualsiasi punto del codice, l'accesso ai parametri temporali per eventuali necessità di calcolo; inoltre abbiamo utilizzato questi parametri per definitire l'intervallo di tempo su cui andare a calcolare la derivata;
– Abbiamo seguito questo approccio nel ciclo principale eseguito a frequenza fissa: lettura file e sensori → algoritmo/elaborazione → produzione dei risultati di interesse;

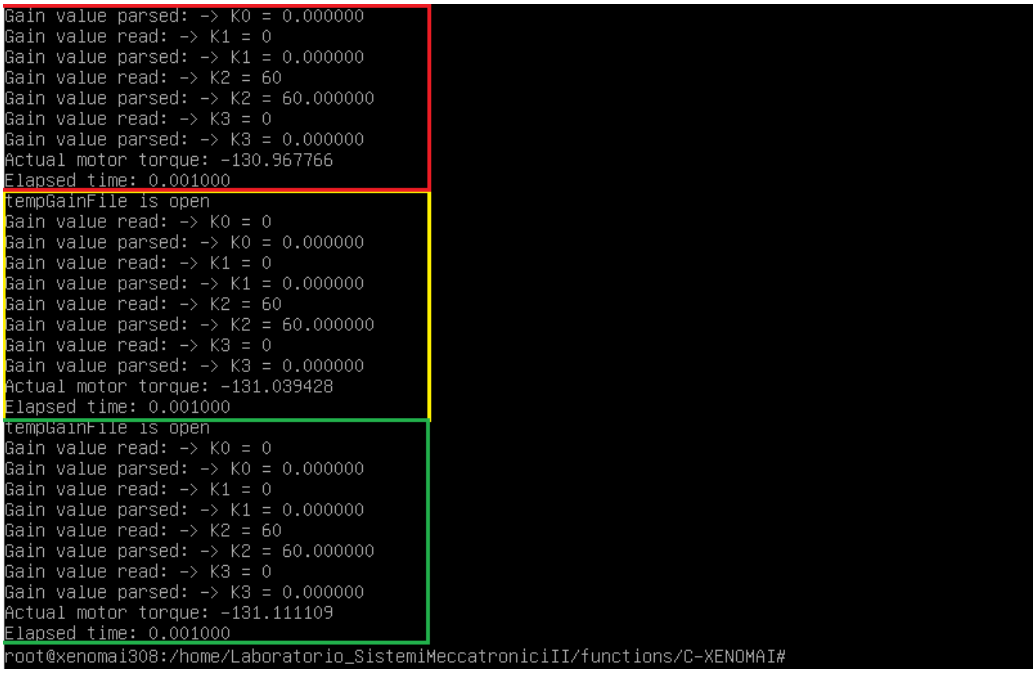

Figure 4.17: Esecuzione a tempo fissato delle funzioni di lettura e calcolo

– Il thread relativo al server opc - ua non è ancora stato gestito: esso sarà ovviamente lasciato in secondo piano rispetto al thread principale temporaizzato alla frequenza specifica;

Part I

Allegati

Appendix A

Appendice A

Riportiamo di seguito il codice prodotto per la gestione della comunicazione tramite protocollo OPC-UA.

```
########## LIBRARY IMPORT ########################
from opcua import Server
from random import randint
from datetime import datetime
import os
import time
import socket
##################################################

############# GET LOCAL MACHINE IP ################
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.connect(("8.8.8.8", 80))
id = s.getsockname()[0]
s.close()
##################################################

############ SET SERVER INSTANCE #################
server = Server()
url = "opc.tcp://"+id+":4840"

server.set_endpoint(url)

name = "V.A.B. - Server Raspberry PI"
addspace = server.register_namespace(name)

node = server.get_objects_node()
Param = node.add_object(addspace, "Gain parameters")
##################################################

############# FILE READING ########################
fileName_temp = "GainParametersToController.txt"
fileName_confirmed = "GainParametersConfirmed.txt"

lineHeaders = ["K_phi ", "K_phi_p ", "K_theta ", "K_theta_p "]
initialValueGains = [0, 0, 0, 0]

if os.path.exists(fileName_confirmed):
    print("Get initial values from file and create a temp one")
    file_object_def = open(fileName_confirmed, "r+")
    lines = file_object_def.readlines()
    count = 0
    # Strips the newline character
    for line in lines:
        print(line.strip())
        values = line.split()
        if count <= 3:
            initialValueGains[count] = int(values[1])
```

```
        else:
            timestamp = str(values[0]) + str(values[1])
        count = count + 1
    file_object_def.close()

    file_object_temp = open(fileName_temp, "w")
    for i, header in zip(initialValueGains, lineHeaders):
        print(header + str(i))
        file_object_temp.write(header + str(i) + "\n")

    # Time stamp from confirmed file
    file_object_temp.write(timestamp)
    file_object_temp.close()
else:
    print("Creating a new files and set parameters to zero")
    file_object_temp = open(fileName_temp, "w")
    file_object_def = open(fileName_confirmed, "w")

    for i, header in zip(range(4), lineHeaders):
        print(header + "0")
        file_object_temp.write(header + "0\n")
        file_object_def.write(header + "0\n")

    # current date and time
    now = datetime.now()
    timestamp = datetime.timestamp(now)
    dt_object = datetime.fromtimestamp(timestamp)
    file_object_temp.write(str(dt_object))
    file_object_def.write(str(dt_object))

    file_object_temp.close()
    file_object_def.close()
################################################

########## PARAMETERS INIT ######################
K_phi = Param.add_variable(addspace, "K_phi", initialValueGains[0])
K_phi_p = Param.add_variable(addspace, "K_phi_p", initialValueGains[1])
K_theta = Param.add_variable(addspace, "K_theta", initialValueGains[2])
K_theta_p = Param.add_variable(addspace, "K_theta_p", initialValueGains[3])
submit_to_controller = Param.add_variable(addspace, "Submit change to controller",
    False)
submit_to_file = Param.add_variable(addspace, "Store definitively in file", False)
shut_down = Param.add_variable(addspace, "SHUT DOWN SERVER", False)

K_phi.set_writable()
K_phi_p.set_writable()
K_theta.set_writable()
K_theta_p.set_writable()
submit_to_controller.set_writable()
submit_to_file.set_writable()
shut_down.set_writable()
################################################

server.start()

print("Server started at {}".format(url))

K1 = K_phi.get_value()
K2 = K_phi_p.get_value()
K3 = K_theta.get_value()
K4 = K_theta_p.get_value()

while True:
    save_on_temp_file = submit_to_controller.get_value()
    save_on_definitively_file = submit_to_file.get_value()
    exit = shut_down.get_value()
```

```
122
123     if save_on_definitively_file:
124         # Write on file that store gains considered stable
125         K1 = K_phi.get_value()
126         K2 = K_phi_p.get_value()
127         K3 = K_theta.get_value()
128         K4 = K_theta_p.get_value()
129
130         gains = [K1, K2, K3, K4]
131         file_object_conf = open(fileName_confirmed, "w")
132         for header, gain in zip(lineHeaders, gains):
133             file_object_conf.write(header + str(gain) + "\n")
134
135         # current date and time
136         now = datetime.now()
137
138         timestamp = datetime.timestamp(now)
139         dt_object = datetime.fromtimestamp(timestamp)
140         file_object_conf.write(str(dt_object))
141         file_object_conf.close()
142
143         submit_to_file.set_value(False)
144         print("Confirmed data")
145
146     if save_on_temp_file:
147         # Write on file that is read from controller on Raspberry
148         K1 = K_phi.get_value()
149         K2 = K_phi_p.get_value()
150         K3 = K_theta.get_value()
151         K4 = K_theta_p.get_value()
152
153         gains = [K1, K2, K3, K4]
154         file_object_temp = open(fileName_temp, "w")
155         for header, gain in zip(lineHeaders, gains):
156             file_object_temp.write(header + str(gain) + "\n")
157
158         # current date and time
159         now = datetime.now()
160
161         timestamp = datetime.timestamp(now)
162         dt_object = datetime.fromtimestamp(timestamp)
163         file_object_temp.write(str(dt_object))
164         file_object_temp.close()
165
166         submit_to_controller.set_value(False)
167         print("Submitted data to temp file that will be read from raspberry")
168
169     if not save_on_temp_file and not save_on_definitively_file:
170         # No change submitted
171         K = [K1, K2, K3, K4]
172         print(K)
173         time.sleep(2)
174
175     if exit:
176         print("Shut down server...")
177         break
178
179 if os.path.exists(fileName_temp):
180     print("Removing temp file...")
181     os.remove(fileName_temp)
182 else:
183     print("The temp file does not exist")
184 print("Server stopped")
```

Bibliography

[1] *Controllo tramite retroazione dello stato,Controlli automatici, Fabio Previdi* https://cal.unibg.it/wp-content/uploads/controlli_automatici/Lez06.pdf