Ringrazio il Prof. Costanzo Manes che con competenza mi ha guidato nella tesi

Ringrazio la mia Famiglia

Ringrazio la Musica

Ringrazio gli amici che dopo una giornata di studio mi ricaricano le energie per il giorno dopo

Ringrazio l' Università dell'Aquila

“ [...] A voi, temerari della ricerca e del tentativo, e a chiunque si sia mai imbarcato
con ingegnose vele su mari terribili,
a voi, ebbri di enigmi e lieti alla luce del crepuscolo,
a voi, le cui anime suoni di flauto inducono a perdersi in voragini labirintiche:
giacchè voi non volete con mano vile seguir tentoni un filo, e dove siete in grado di
indovinare vi è in odio il dedurre
a voi soli racconterò l'enigma che io vidi,
la visione del più solitario tra gli uomini [...] “

“ Io ne ho viste cose che voi umani non potreste immaginarvi:
navi da combattimento in fiamme al largo dei bastioni di Orione,
e ho visto i raggi B balenare nel buio vicino alle porte di Tannhäuser.
E tutti quei momenti andranno perduti nel tempo, come lacrime nella pioggia.
È tempo di morire “

Contents

Chapter 1

Introduzione

L' obiettivo di questa tesi è il controllo della stabilità di una pallina, in moto su una piattaforma montata sull' organo terminale di un robot che, attraverso movimenti traslatori e rotatori, ne evita la caduta, riportandola alla condizione di quiete. La posizione della pallina è acquisita da un touchscreen integrato nella piattaforma, mentre la posizione e l' orientamento della piattaforma sono acquisiti dai sensori del robot. Il sistema touchscreen-pallina è comunemente detto sistema *ball-on-plate*: nella sua variante più classica è composto da una piattaforma attuata da motori a bassa potenza che le consentono di effettuare piccole rotazioni: nel nostro caso l' attuazione è effettuata dai motori ad alta potenza del robot su cui questa è montata, che favoriscono una stabilizzazione notevolmente più efficiente, eseguita per mezzo sia di rotazioni che di traslazioni. Il robot su cui verrà implementato l' algoritmo di controllo è un manipolatore antropomorfo, di cui si parlerà in seguito. Per ora il tutto è stato realizzato in simulazione nell' ambiente $MATLAB^{®}$ con approccio object-oriented, ma si prevede, in un futuro prossimo, un' implementazione anche su robot reale. È in corso nell' Università dell'Aquila un progetto parallelo a quello presentato in questa tesi, che ha l' obiettivo di creare un' interfaccia per impartire comandi al manipolatore direttamente da MATLAB; è prevista anche la risoluzione della cinematica inversa, con cui sarà possibile controllare da MATLAB direttamente la posizione cartesiana e l' orientamento dell' organo terminale. Grazie a questo progetto sarà presto possibile implementare il controllo realizzato finora in sim-

ulazione, anche su robot reale.

La prima parte della tesi riguarda la modellistica e la simulazione fisica del sistema: sono state trovate le leggi che governano il moto della pallina posta sopra una piattaforma rettangolare che effettua dei movimenti sia traslatori che rotatori; per rendere più intuitiva la comprensione dei risultati è stata programmata un' animazione che mostra il tutto. L' approccio matematico seguito è basato sull' analisi di tutte le accelerazioni a cui è sottoposta la pallina poggiata sulla piattaforma, dalle quali, tramite integrazione, si ricavano velocità e posizione.

È stato implementato un primo controllo di stabilità, dalle prestazioni discrete, attraverso un PD. L' uscita del regolatore, valutando l' errore di posizione e di velocità, restituisce l' angolo che deve assumere la piattaforma per portare la pallina alla condizione di equilibrio. Il piatto è portato all' orientamento scelto attraverso un profilo di velocità triangolare in un tempo prefissato relativamente breve.

Il vero controllo di stabilità è stato realizzato attraverso un regolatore lineare quadratico: è un controllore ottimo basato su una controreazione dello stato, in cui la matrice dei guadagni è scelta in modo da minimizzare un funzionale di costo quadratico nello stato e nell' ingresso del sistema, garantendo una risposta eccellente sia dal punto di vista della sovraelongazione e del tempo di assestamento, sia dal punto di vista dello sforzo di controllo. Tale controllore necessita di conoscere una realizzazione lineare nello spazio di stato del sistema da controllare; essendo questo non lineare, è stata effettuata una linearizzazione attorno al punto di equilibrio attraverso uno sviluppo di Taylor multidimensionale troncato al primo ordine. Sia la simulazione che il controllo sono stati realizzati a tempo discreto.

1.1 Il robot antropomorfo

Un robot, o manipolatore, può essere definito come una struttura meccanica multifunzionale riprogrammabile, progettata per spostare materiali, usufruire di appositi utensili, e in generale svolgere compiti tramite movimenti programmati: è composto di

- una **struttura meccanica** che consiste in un insieme di *bracci* intercon-

nessi tra loro attraverso dei giunti, che possono essere *prismatici* o *rotoidali:* un giunto prismatico effettua un moto relativo di traslazione tra i due bracci adiacenti, mentre uno rotoidale realizza un moto rotazionale

- degli **attuatori**, che imprimono il moto ai giunti e, conseguentemente, ai bracci. Nel nostro caso abbiamo dei motori elettrici
- dei **sensori** che rilevano lo stato attuale del manipolatore: nel nostro caso possiamo rilevare sia posizione che orientamento dell' organo terminale
- un' **unità di governo** che si preoccupa di realizzare il controllo e la supervisione dei movimenti del robot; l' algoritmo che si andrà a mostrare in questa tesi verrà applicato al manipolatore tramite questa unità

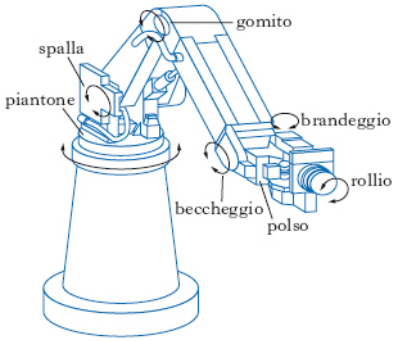

Figure 1.1: Configurazione dei giunti di un robot antropomorfo

Un parametro importante per caratterizzare i robot è il numero gradi di libertà, cioè la dimensione del sottospazio, detto spazio di lavoro, nel quale essi sono in grado di posizionare il loro organo terminale; tale dimensione coincide con il numero dei giunti. Pensiamo infatti ad un robot con un solo giunto: esso può posizionare il suo organo terminale su una varietà di ordine 1, cioè su una retta se il suo unico giunto è prismatico, o su una circonferenza se è rotoidale. Aggiungendo un giunto si guadagna un grado di libertà, cioè ci si può muovere su una varietà di ordine 2, quindi una superficie che può essere una sfera, un cilindro o un piano a seconda della configurazione dei giunti.

La catalogazione dei robot viene usualmente fatta in base al loro spazio di lavoro e alla loro configurazione dei giunti: una delle più diffuse è quella con

sei giunti rotoidali diposti come in figura 1.1, che si è dimostrata negli anni una delle più funzionali in ambito industriale: un robot con questa configurazione viene detto *antropomorfo,* e appare come in figura 1.2.

Figure 1.2: Un robot antropomorfo

Lo spazio di lavoro di tale robot è una varietà di ordine 6, con 3 gradi di libertà per la posizione e 3 per l' orientamento, garantendo massima versatilità per qualunque compito industriale. In questa tesi progetteremo l' algoritmo di controllo per un manipolatore antropomorfo prodotto dall' azienza italiana $COMAU^{®}$.

1.2 Il touchscreen e la pallina

Sull' organo terminale del robot verrà montato un sistema *ball-on-plate* simile a quello in figura 1.3, che consiste di una piattaforma touchscreen con una pallina posta sopra; il touchscreen è in grado di rilevare la posizione e la velocità della pallina ad ogni istante di campionamento, permettendo di avere le informazioni necessarie per effettuare il controllo di stabilità: tale controllo consiste nel mantenere sempre la pallina sulla piattaforma senza farla cadere e riportandola in tempo breve e con basse oscillazioni al centro, cioè nella sua posizione di equilibrio. Il movimento della piattaforma è interamente controllato dagli attuatori del robot che, avendo sei gradi di libertà, può eseguire un compito sia di traslazione che, cosa senza la quale il controllo sarebbe irrealizzabile, di rotazione. In particolare precisiamo che nel controllo PD la pallina viene stabi-

lizzata attraverso sole rotazioni della piattaforma, nel controllo LQR invece si combinanano traslazioni e rotazioni.

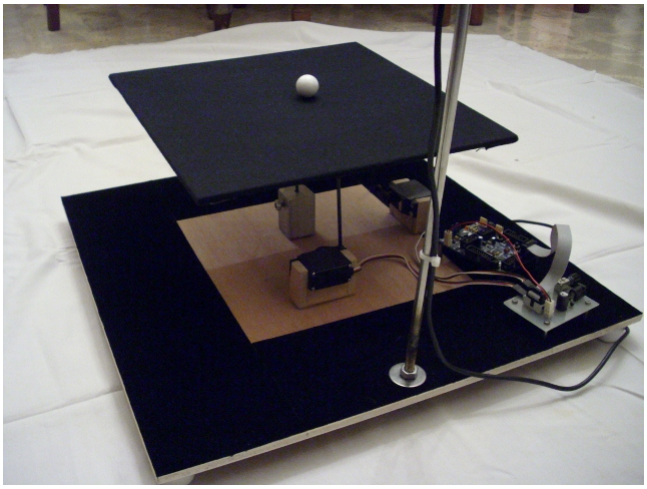

Figure 1.3: Un sistema ball on plate

Abbiamo tutti i sensori necessari per ricostruire lo stato del sistema che sarà definito come posizione, velocità, orientamento e velocità angolare della piattaforma, e posizione e velocità della pallina; non abbiamo bisogno di costruire alcun osservatore.

Un premessa importante da fare è che nella simulazione è stato necessario effettuare molti calcoli laboriosi che nell' implementazione reale non servono: le equazioni di moto della pallina, infatti, verrano definite nel sistema di riferimento solidale alla piattaforma; avvenendo però la simulazione in coordinate terrestri, è necessario trasformare le coordinate di terra in coordinate solidali alla piattaforma, applicare le leggi di moto, e infine ritornare alle coordinate terresti. Per il controllo, invece, non è necessario conoscere la posizione assoluta nelle coordinate terrestri, ma solo quella rispetto alla pattaforma, quindi si effettua anche lì una trasformazione di coordinate. Nell' applicazione reale il touchscreen rileva le coordinate della pallina direttamente nelle coordinate della piattaforma, quindi in quelle utili per il controllo: di conseguenza non deve essere effettuato alcun cambio di coordinate.

1.3 Alcune note

- Nella scrittura delle matrici sono state usate le parentesi quadre in contesti informatici, quando la matrice è una variabile MATLAB e in particolare quando rappresenta una struttura dati priva di significato algebrico, usata solo per memorizzare parametri. Sono state usate le parentesi tonde in ambiti matematici, quando la matrice ha un significato algebrico ben preciso di endomorifismo, di matrice Jacobiana e così via.

- Un' altra doppia notazione, introdotta per motivi di leggibilità, è quella per eprimere lo stato o l' ingresso in determinati istanti di campionamento: si troverà la notazione con il pedice $\mathbf{q}_{k+1}$ oppure quella con le parentesi $\mathbf{q}(k+1)$: le due notazioni sono *perfettamente equivalenti* e indicano il valore assunto lo stato $\mathbf{q}$ nell' istante $k+1$.

- In alcuni casi è stata usata una notazione molto usuale nella robotica, ambito in cui si lavora molto con le funzioni trigoniometriche: dato un angolo φ, si abbrevia con $c\varphi$ il coseno dell' angolo, $\cos\varphi$, e con $s\varphi$ il seno, $\sin\varphi$.

- Infine nel calcolo delle accelerazioni della piattaforma non viene inclusa l' accelerazione di compensazione gravitazionale. Se si volesse includere, basterebbe aggiungere il suo valore alla terza componente dell' accelerazione lineare, in ogni istante.

Chapter 2

La dinamica Newtoniana

Il modello dinamico Newtoniano è un modello fisico atto a studiare la traiettoria di un sistema di corpi soggetti a forze. È un approccio più antico di quello Lagrangiano, ma nonostante ciò conserva ancora la sua efficienza. Per sfruttare i principi di Newton è necessario calcolare la somma vettoriale, detta risultante, di tutte le forze agenti su un corpo, passare all' accelerazione, e ottenere per integrazione velocità e posizione: si dimostra infatti che da un punto di vista dinamico un corpo soggetto a più forze si comporta come se vi agisse una sola forza, la risultante. In questa tesi il sistema da analizzare è il sistema piatto-pallina di cui si troveranno le leggi per posizione e velocità; si partirà direttamente dalle accelerazioni poichè si trascurerà l' azione della forza peso della pallina sul piatto.

2.1 Il sistema piatto-pallina

Veniamo adesso a studiare le equazioni che regolano il moto della pallina e del piatto; sono state fatte le seguenti ipotesi:

- la pallina è *vincolata* al piatto: può muoversi solamente nel piano della piattaforma, non può "saltare"

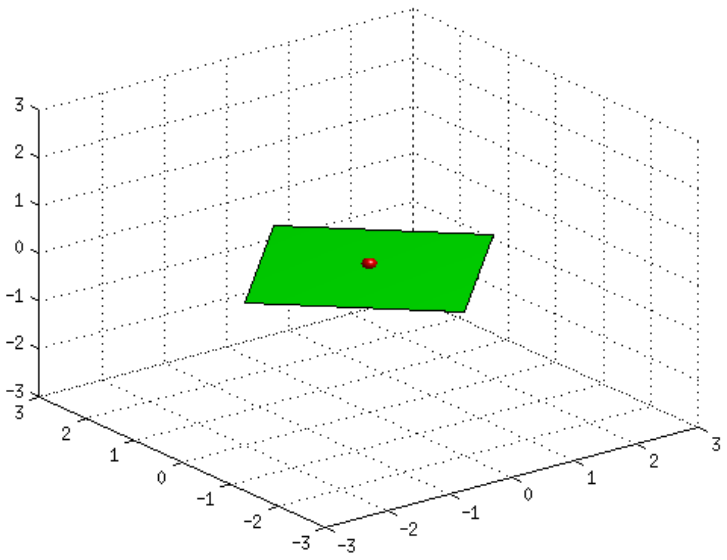

Figure 2.1: Sistema piatto-pallina

- la pallina è stata considerata un punto materiale: ciò vuol dire che non sono state modellizzate inerzie rotazionali che complicherebbero notevolmente il modello; inoltre poichè la pallina e la piattaforma sono fatte di un materiale a basso coefficiente di attrito, non è del tutto realistico considerare sempre il moto di puro rotolmento
- la massa della pallina è stata trascurata, poichè la sua forza peso ha un ordine di grandezza molto infieriore a quello delle forze attuate dal polso robotico

Per una rappresentazione agevole del sistema totale consideriamo il piatto e la pallina come due sistemi, ognuno dotato di un suo vettore di stato: lo stato del piatto è definito dal vettore:

$$\mathbf{q}_{piatto} = \begin{pmatrix} x \\ \dot{x} \\ \theta \\ \omega \end{pmatrix} \tag{2.1}$$

Ognuna delle componenti del vettore di stato è a sua volta un vettore di dimensione 3; lo spazio di stato ha complessivamente dimensione 12; x è la posizione del centro del piatto, $\dot{x}$ la sua velocità, θ l' orientamento espresso attraverso gli angoli di RPY, e ω la velocità angolare.

Per quanto riguarda la pallina, considerata come punto materiale, il vettore di stato è composto solamente da posizione e velocità lineare

$$\mathbf{q}_{ball} = \begin{pmatrix} p \\ \dot{p} \end{pmatrix} \tag{2.2}$$

e ha perciò una dimensione pari a 6; p è la posizione del suo centro, e $\dot{p}$ la sua velocità.

Possiamo definire lo stato complessivo del sistema come concatenazione dei due vettori:

$$\mathbf{q} = \begin{pmatrix} \mathbf{q}_{\text{piatto}} \\ \mathbf{q}_{\text{ball}} \end{pmatrix} \tag{2.3}$$

Bisogna sottolineare che la traiettoria della pallina dipende dallo stato del piatto, ma non è vero il viceversa: ricordando l' ipotesi di massa trascurabile, l' influenza della pallina sul moto del piatto viene trascurata; grazie a questa osservazione possiamo trovare le leggi di moto della piattaforma senza preoccuparci della pallina.

Le accelerazioni lineari e angolari a cui è sottoposta la piattaforma detemminano l' evoluzione del sistema; l' ingresso $\mathbf{u}$ viene definito come come concatenazione del vettore di accelerazione traslazionale a con quello di accelerazione angolare α

$$\mathbf{u} = \begin{pmatrix} a \\ \alpha \end{pmatrix} \tag{2.4}$$

È un vettore di dimensione 6. È necessario precisare che le accelerazioni traslazionali, quindi le forze, vengono applicate al baricentro del piatto, e perciò non generano alcun momento torcente della piattaforma, quindi alcuna accelerazione angolare. D' altra parte anche le accelerazioni rotazionali sono riferite al baricentro, e allo stesso modo non generano alcuna accelerazione traslazionale: da ciò deriva che la componente di modo traslazionale è indipendente da quella rotazionale, e le due possono quindi essere trattate in modo separato.

2.2 La dinamica della piattaforma

2.2.1 Parte traslazionale

Partiamo dalla parte traslazionale, e consideriamo una funzione di accelerazione nel tempo $a(t)$. Per calcolare la velocità a partire dall' accelerazione bisogna effettuare un' operazione di integrazione:

$$v(t) = \int_0^t a(t)dt \tag{2.5}$$

Per calcolare la posizione si deve effettuare la stessa operazione con la velocità, cioè

$$x(t) = \int_0^t v(t)dt \tag{2.6}$$

Avendo a disposizione stumenti digitali, per calcolare posizione e velocità bisogna utilizzare un' integrazione discreta; dato un tempo di campionamento Ts e una funzione di accelerazione $a(t)$, con un tempo di campionamento sufficientemente piccolo si può supporre che $a(t)$ sia costante all' intero di un periodo di campionamento: in tal caso, abbiamo

$$v(t + kTs) = v(kTs) + a\int_{kTs}^{t+kTs} dt = v(kTs) + a(kTs)t \tag{2.7}$$

Tale metodo di integrazione corrisponde a quello di Eulero. Considerando $v(t)$ sempre costante all' interno di un periodo di campionamento possiamo integrare una seconda volta per ottenere la posizione.

$$x(t + kTs) = x(kTs) + \int_{kTs}^{t+kTs} v(kTs) + atdt = x(kTs) + v(kTs)t + a(kTs)\frac{t^2}{2} \tag{2.8}$$

Considerando $t = Ts$, caso in cui si integra in un periodo, le espressioni di posizione e velocità diventano:

$$x((k+1)Ts) = x(kTs) + v(kTs)Ts + a(kTs)\frac{Ts^2}{2} \tag{2.9}$$

$$v((k+1)Ts) = v(kTs) + a(kTs)Ts \tag{2.10}$$

Abbiamo ottenuto due relazioni in grado di calcolare posizione e velocità all'

istante di campionamento successivo, conoscendo posizione e velocità all' istante di campionamento corrente e accelerazione, supposta costante all' interno dell' intervallo di campionamento.

2.2.2 Cambiamento del sistema di riferimento

Prima di passare ad analizzare la parte rotazionale e le leggi di moto della pallina, è importante introdurre il concetto di cambio di sistema di riferimento: d' ora in avanti in ogni vettore in cui ci sia ambiguità di interpretazione, verrà specificato il sistema di riferimento usando come notazione un apice sinistro: ad esempio con la notazione ${}^0p$ ci si riferisce al vettore p nelle coordinate del sistema 0, cioè quello di terra. Con la notazione ${}^pp$ invece ci si riferisce a p nelle coordinate del sistema di riferimento solidale al piatto. Verranno ricavate in seguito le formule per il cambio di sistema di riferimento.

2.2.2.1 Trasformazione di posizioni

Consideriamo inizialmente i due sistemi, ${}^0S$ e ${}^pS$ con origine coincidente, ma assi ruotati: il vettore ${}^pp$ ha come componenti le lunghezze delle proiezioni del vettore ${}^0p$ sui 3 versori ${}^p\hat{x}$ ${}^p\hat{y}$ e ${}^p\hat{z}$, assi del sistema ${}^pS$. Tali lunghezze si calcolano attraverso i tre prodotti scalari:

$$ {}^pp_x = p_0^{Tp}\hat{x} \tag{2.11} $$

$$ {}^pp_y = p_0^{Tp}\hat{y} \tag{2.12} $$

$$ {}^pp_y = p_0^{Tp}\hat{z} \tag{2.13} $$

Che scritto in forma compatta diventa un prodotto matriciale:

$$ {}^pp = {}^pR_0\,{}^0p \tag{2.14} $$

La notazione ${}^pR_0$ indica la matrice che porta dal sistema ${}^0S$ al sistema ${}^pS$. ${}^pR_0$, che può essere interpretata come operatore di rotazione, ha per colonne i 3 versori rappresentanti gli assi cartesiani del sistema ${}^pS$ espressi nelle coordinate di ${}^0S$, e appare in questo modo

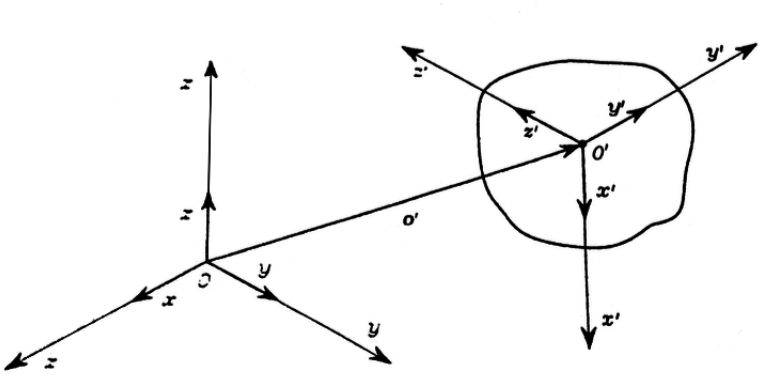

Figure 2.2: Sistema traslato e ruotato

$$ {}^{p}R_0 = \begin{pmatrix} {}^0x_x & {}^0y_x & {}^0z_x \\ {}^0x_y & {}^0y_y & {}^0z_y \\ {}^0x_z & {}^0y_z & {}^0z_z \end{pmatrix} $$

Una caratteristica peculiare di qualsiasi matrice di rotazione è l' ortogonalità, cioè la proprietà di avere l' inversa uguale alla trasposta. Data una rotazione ${}^{p}R_0$, la matrice ${}^{0}R_p$ che effettua la rotazione in senso inverso si ottiene facendo la trasposta, o analogamente l' inversa, della matrice di partenza. Vale quindi:

$$ {}^{0}R_p = ({}^{p}R_0)^T \tag{2.15} $$

Inoltre se si applicano ad un vettore più rotazioni in successione, la matrice che esprime la rotazione complessiva è il prodotto righe per colonne di tutte le matrici di rotazione, nell' ordine in cui avvengono.

Consideriamo adesso il caso di sistema traslato e ruotato, ma in cui l' origine sia sempre fermo: osservando la figura (2.2) si intuisce facilmente che la relazione è

$$ {}^{0}p = {}^{0}o_{touch} + {}^{0}R_p {}^{p}p \tag{2.16} $$

Dopo aver analizzato i vettori posizione p ci si deve porre il problema di come cambiano le velocità, quando si cambia sistema di riferimento. Nel nostro caso infatti la pallina si muove rotolando sul piatto, e dobbiamo poter esprimere la sua velocità nel sistema di riferimento solidale alla piattaforma; considerando il fatto che il sistema è in movimento, la matrice di rotazione $R(t)$ non sarà

costante come nel caso statico, ma varierà nel tempo.

Si effettua una derivata di ambo i membri dell' espressione (2.16) della posizione e di ottiene l' espressione per trovare il valore del vettore velocità in un sistema diverso.

$$^0\dot{p} = ^0\dot{o}_{touch} + {}^0R_p(t)^p\dot{p} + {}^0\dot{R}_p(t)^p p \tag{2.17}$$

La derivata del prodotto restituisce la somma di due termini: nel secondo appare la derivata della matrice di rotazione, che finora non è ancora stata trattata.

2.2.2.2 Derivata di una matrice di rotazione

Si parte dalla condizione di ortonormalità della matrice di rotazione $R(t)$, cioè il fatto che

$$R(t)R^T(t) = I \tag{2.18}$$

dove con I ci si riferisce alla matrice identità. Tenendo presente la formula di derivata di un prodotto, si derivano entrambi i membri dell' equazione (2.18) appena presentata:

$$\dot{R}(t)R^T(t) + R(t)\dot{R}^T(t) = O \tag{2.19}$$

con O matrice nulla. Ponendo

$$S(t) = \dot{R}(t)R^T(t) \tag{2.20}$$

l' espressione diventa

$$S(t) + S^T(t) = O$$

che mette in luce la proprietà di antisimmetria della matrice $S(t)$.

Moltiplicando a destra ambo i membri della (2.20) per $R(t)$ si ottiene

$$\dot{R}(t) = S(t)R(t) \tag{2.21}$$

che è una relazione molto importante, che consente di interpretare $S(t)$ come operatore di derivazione matriciale; ma questa non è l' unica interpretazione che si può dare a questa matrice: consideriamo un generico vettore rotante

$$p(t) = R(t)p_0 \tag{2.22}$$

indicando con p_0 il vettore costante a cui è applicata la rotazione tramite la matrice $R(t)$. Se ne facciamo la derivata otteniamo, avvalendoci della (2.21)

$$\dot{p}(t) = \dot{R}(t)p' = S(t)R(t)p'$$

Confrontando questa equazione con la nota relazione della meccanica

$$\dot{p}(t) = \omega(t) \times R(t)p'$$

si deduce che la matrice antisimmetrica $S(t)$ rappresenta un operatore di prodotto vettoriale per la velocità angolare. Infatti questa ha sempre forma

$$S = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{pmatrix} \tag{2.23}$$

si userà la notazione $[\ \omega \times\ ]$ o equivalentemente $S(\omega)$ per indicare la matrice costruita come in equazione (2.23).

Riassumento possiamo dire che la matrice S, per come è stata definita nella (2.20) ha duplice funzione: se posta davanti alla matrice $R(t)$ funge da operatore derivata. Se posta davanti ad un vettore rotante funge da operatore prodotto vettoriale per la velocità angolare ω.

Inoltre una relazione interessante da mostrare è quella del cambiamento di coordinate per la matrice prodotto vettoriale:

$$[\,(R\omega)\times\,] = R\,[\,\omega\times\,]\,R^T \tag{2.24}$$

dove R è una generica matrice di rotazione.

2.2.2.3 Trasformazioni di velocità

Dopo aver affrontato il problema della derivata di una matrice di rotazione possiamo effettuare i passaggi per ottenere la relazione di trasformazione di velocità; richiamiamo l' equazione (2.17)

$${}^0\dot{p} = {}^0\dot{o}_p + {}^0R_p(t)\,{}^p\dot{p} + {}^0\dot{R}_p(t)\,{}^pp$$

che sviluppando la derivata di matrice diventa

$$^0\dot{p} = {}^0\dot{o}_{touch} + {}^0R_p(t)\,{}^p\dot{p} + S({}^0\omega)\left[{}^0R_p(t)\,{}^pp\right] = {}^0\dot{o}_{touch} + {}^0R_p(t)\,{}^p\dot{p} + S({}^0\omega)\,{}^0r_b$$

$$^0\dot{p} = {}^0\dot{o}_{touch} + {}^0R_p(t)\,{}^p\dot{p} + {}^0\omega \times {}^0r_b \tag{2.25}$$

dove con ^0r_bsi indica la distanza del corpo in moto dall' origine del sistema ^pS.

La relazione è composta dalla somma di tre termini, che vale la pena analizzare singolarmente:

- $^0\dot{o}_p$ è la velocità dell' origine del sistema
- $^0R_p(t)\,^p\dot{p}$ è la velocità lineare dell' oggetto in moto all' interno del sistema ^pS
- $^0\omega \times {}^0r_b$ è la velocità generata dal moto angolare del sistema di riferimento

2.2.2.4 Gli angoli di Roll Pitch e Yaw

Affrontiamo adesso il problema di come esprimere l' orientamento di un corpo rigido: in questa tesi si è scelto di usare gli angoli di Roll Pitch e Yaw, prendendo come assi gli stessi relativi alla parte traslazionale;

Gli angoli di Roll-Pitch-Yaw sono tre angoli che consentono di parametrizzare in maniera minima l'orientamento di un sistema di riferimento ^pS, in questo caso il sistema solidale alla piattaforma, rispetto ad un sistema di terra ^0S. La regola che consente di associare in modo univoco ai tre angoli di Roll-Pitch-Yaw, indicati nel seguito con $(\vartheta_x, \vartheta_y, \vartheta_z)$, l'orientamento di ^pS rispetto a ^0S, è costituita da tre rotazioni attorno agli assi coordinati. Le tre rotazioni elementari sono definite come segue:

$$\Theta = \begin{pmatrix} \vartheta_x \\ \vartheta_y \\ \vartheta_z \end{pmatrix}, \quad \begin{array}{l} 1^o)\ rotazione\ \vartheta_x\ intorno\ a\ x_0 \\ 2^o)\ rotazione\ \vartheta_y\ intorno\ a\ y_0 \\ 3^o)\ rotazione\ \vartheta_z\ intorno\ a\ z_0 \end{array} \tag{2.26}$$

Interpretando la matrice ^0R_c come rappresentazione di un operatore di rotazione nelle coordinate di ^0S, rotazione che porta x_0 in x_c , y_0 in y_c e z_0 in z_c, si ha

$$^0R_c = R(\Theta) \tag{2.27}$$

dove

$$R(\Theta) = R_z(\vartheta_z) R_y(\vartheta_y) R_x(\vartheta_x), \tag{2.28}$$

in cui R_x , R_y e R_z sono le matrici assi coordinati. Si ha pertanto

$$R(\Theta) = \begin{pmatrix} c\vartheta_z & -s\vartheta_z & 0 \\ s\vartheta_z & c\vartheta_z & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c\vartheta_y & 0 & s\vartheta_y \\ 0 & 1 & 0 \\ -s\vartheta_y & 0 & c\vartheta_y \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & c\vartheta_x & -s\vartheta_x \\ 0 & s\vartheta_x & c\vartheta_x \end{pmatrix} \tag{2.29}$$

Svolgendo il prodotto tra le tre matrici si ottiene

$$R(\Theta) = \begin{pmatrix} c\vartheta_z & -s\vartheta_z & 0 \\ s\vartheta_z & c\vartheta_z & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} c\vartheta_y & s\vartheta_x s\vartheta_y & c\vartheta_x s\vartheta_y \\ 0 & c\vartheta_x & -s\vartheta_x \\ -s\vartheta_y & s\vartheta_x c\vartheta_y & c\vartheta_x c\vartheta_y \end{pmatrix} \tag{2.30}$$

$$R(\Theta) = \begin{pmatrix} c\vartheta_z c\vartheta_y & c\vartheta_z s\vartheta_y s\vartheta_x - s\vartheta_z c\vartheta_x & c\vartheta_z s\vartheta_y c\vartheta_x + s\vartheta_z s\vartheta_x \\ s\vartheta_z c\vartheta_y & s\vartheta_z s\vartheta_y s\vartheta_x + c\vartheta_z c\vartheta_x & s\vartheta_z s\vartheta_y c\vartheta_x - c\vartheta_z s\vartheta_x \\ -s\vartheta_y & c\vartheta_y s\vartheta_x & c\vartheta_y c\vartheta_x \end{pmatrix} \tag{2.31}$$

Si vuole ora mostrare come sia possibile calcolare i tre angoli di Roll-Pitch-Yaw associati ad una data matrice di rotazione

$$^0R_c = \begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ r_{2,1} & r_{2,2} & r_{2,3} \\ r_{3,1} & r_{3,2} & r_{3,3} \end{pmatrix} \tag{2.32}$$

per confronto con l'espressione (2.31).

Per prima cosa è possibile calcolare l'angolo ϑ_y in funzione del termine $r_{3,1}$, in quanto è evidente che $-s\vartheta_y = r_{3,1}$. E chiaro che, tranne che nei casi in cui $r_{3,1} = 1$ o $r_{3,1} = -1$ (in cui $\vartheta_y = -\frac{\pi}{2}$ o $\vartheta_y = \frac{\pi}{2}$), l'equazione

$$s\vartheta_y = -r_{3,1} \tag{2.33}$$

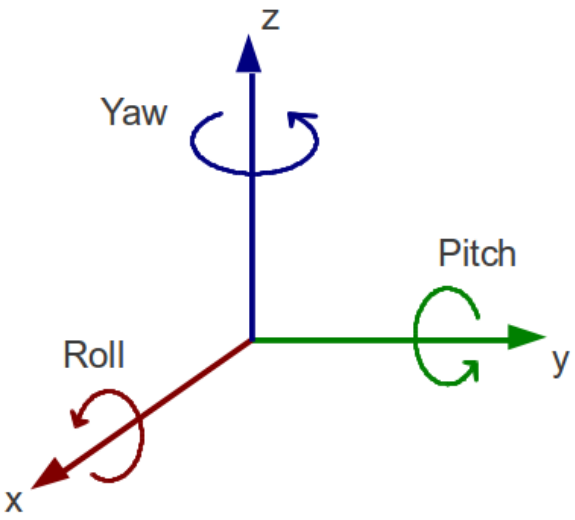

Figure 2.3: Angoli di Roll

ammette sempre 2 soluzioni nell'intervallo angolare $(-\pi, \pi]$, una con $c\vartheta_y > 0$ e l'altra con $c\vartheta_y < 0$. Chiameremo ϑ_y^1 la soluzione con coseno positivo, tale che $\vartheta_y^1 \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$, e con ϑ_y^2 la soluzione con coseno negativo, tale che $\vartheta_y^2 \in \left(-\pi, -\frac{\pi}{2}\right) \bigcup \left(\frac{\pi}{2}, \pi\right)$. Si ha quindi

$$\vartheta_y^1 = s\vartheta_y^1 = -r_{3,1}, \quad c\vartheta_y^1 = \sqrt{1 - r_{3,1}^2}, \qquad \vartheta_y^1 \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$$

$$\vartheta_y^2 = s\vartheta_y^2 = -r_{3,1}, \quad c\vartheta_y^2 = -\sqrt{1 - r_{3,1}^2}, \qquad \vartheta_y^2 \in \left(-\pi, -\frac{\pi}{2}\right) \bigcup \left(\frac{\pi}{2}, \pi\right)$$

Osservando la prima colonna della matrice si noti anche che

$$c\vartheta_y^1 = \sqrt{r_{1,1}^2 - r_{2,1}^2} \qquad c\vartheta_y^2 = -\sqrt{r_{1,1}^2 - r_{2,1}^2}$$

Pertanto, le due terne di angoli di RPY, Θ^1 e Θ^2, che risolvono il problema nel caso in cui $|r_{3,1}| < 1$ si calcolano come

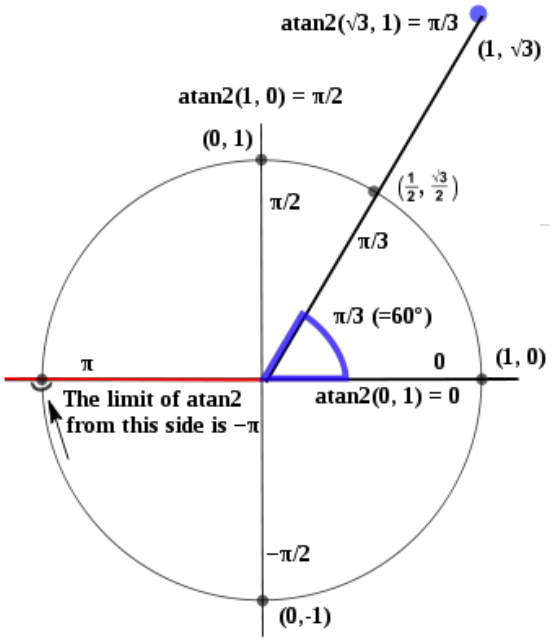

Figure 2.4: Arcotangente a quattro quadranti [$\arctan 2\,(\cdot,\cdot)$]

$$\Theta_1 = \begin{cases} \vartheta_x^1 = \arctan 2\,(r_{3,2},\, r_{3,3}) \\ \vartheta_y^1 = \arctan 2\left(-r_{3,1},\, \sqrt{r_{1,1}^2 + r_{2,1}^2}\right) \\ \vartheta_z^1 = \arctan 2\,(r_{2,1},\, r_{1,1}) \end{cases} \tag{2.34}$$

$$\Theta_2 = \begin{cases} \vartheta_x^1 = \arctan 2\,(-r_{3,2},\, -r_{3,3}) \\ \vartheta_y^1 = \arctan 2\left(-r_{3,1},\, -\sqrt{r_{1,1}^2 + r_{2,1}^2}\right) \\ \vartheta_z^1 = \arctan 2\,(-r_{2,1},\, -r_{1,1}) \end{cases} \tag{2.35}$$

dove la funzione $\arctan 2\,(\cdot,\cdot)$ è la cosiddetta "arcotangente a quattro quadranti", ed è tale che

$$\vartheta = \arctan 2\,(a,\, b) \quad \Longleftrightarrow \quad \begin{array}{l} a = \sqrt{a^2 + b^2}\,\sin\vartheta, \\ b = \sqrt{a^2 + b^2}\,\cos\vartheta. \end{array}$$

Va osservato che si ha anche

$$\arctan 2\,(\rho\sin\vartheta,\, \rho\cos\vartheta) = \vartheta, \qquad\qquad se \quad \rho > 0$$

$$\arctan 2\left(\rho \sin\vartheta,\, \rho\cos\vartheta\right) = \vartheta + \pi, \qquad se \quad \rho < 0$$

$$\arctan 2\left(-\rho \sin\vartheta,\, -\rho\cos\vartheta\right) = \vartheta, \qquad se \quad \rho < 0.$$

Nel caso in cui $r_{3,1} = \pm 1$, per l'angolo ϑ_y si ha una soluzione unica, data da

$$\vartheta_y = -\frac{\pi}{2} r_{3,1}. \tag{2.36}$$

A questo punto però è facile verificare che ci sono infinite soluzioni per gli angoli ϑ_x e ϑ_z. Infatti se $\vartheta_y = -\frac{\pi}{2}$ si ha $c\vartheta_y = 0$, e la matrice di rotazione ${}^0R_c\left(\Theta\right)$ diventa

$$\begin{aligned} {}^0R_c\left(\Theta\right)|_{\vartheta_y=\frac{\pi}{2}} &= \begin{pmatrix} 0 & c\vartheta_z s\vartheta_x - s\vartheta_z c\vartheta_x & c\vartheta_z c\vartheta_x + s\vartheta_z s\vartheta_x \\ 0 & s\vartheta_z s\vartheta_x + c\vartheta_z c\vartheta_x & s\vartheta_z c\vartheta_x - c\vartheta_z s\vartheta_x \\ -1 & 0 & 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 & \sin\left(\vartheta_x - \vartheta_z\right) & \cos\left(\vartheta_x - \vartheta_z\right) \\ 0 & \cos\left(\vartheta_x - \vartheta_z\right) & -\sin\left(\vartheta_x - \vartheta_z\right) \\ -1 & 0 & 0 \end{pmatrix}. \end{aligned} \tag{2.37}$$

Tutte le soluzioni ϑ_x, ϑ_y e ϑ_z sono date da

$$\vartheta_y = -\frac{\pi}{2}$$

$$\vartheta_x + \vartheta_z = -\frac{\pi}{2} = \arctan 2\left(-r_{1,2},\, r_{2,2}\right)$$

2.2.2.5 Legame tra $\dot{\theta}$ e ω

La relazione tra gli angoli di Eulero e la velocità angolare non è una relazione differenziale diretta, come nel caso di posizione e velocità: la grandezza $\dot{\theta}$ non è in generale uguale a ω. Intuitivamente ciò può essere compreso pensando al fatto che ω è un vettore la cui direzione corrisponde all' asse della rotazione, e il cui modulo è la velocità angolare; $\dot{\theta}$ invece è un vettore calcolato effettuando la derivata, componente per componente, degli angoli di Roll, Pitch e Yaw. Verrà dimostrato che l' endomorfismo che lega i due vettori $\dot{\theta}$ e ω è il prodotto per un'opportuna matrice, $G(\theta)$.

Per calcolare il legame tra la velocità angolare di un corpo rigido e la derivata

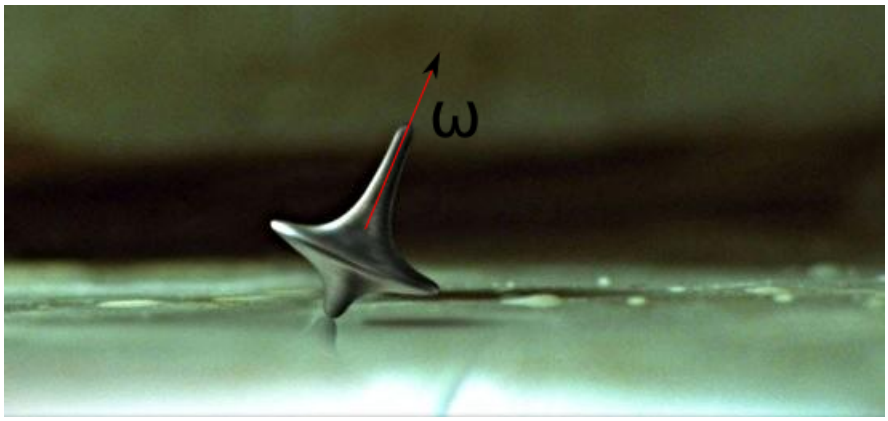

Figure 2.5: Velocità angolare in una trottola

degli angoli a di Roll-Pitch-Yaw, occorre ricordare la relazione

$$ {}^0\dot{R}_p\,{}^0R_p^T = \left[{}^0\omega\times\right] \tag{2.38} $$

dove con ${}^0\omega$ si intende la velocità angolare del sistema ${}^pS$ rispetto a ${}^0S$. Grazie alla regola della catena per la derivazione, possiamo scrivere:

$$ {}^0\dot{R}_p\,{}^0R_p^T = \left(\frac{\partial\,{}^0R_p}{\partial\vartheta_x}\dot{\vartheta}_x + \frac{\partial\,{}^0R_p}{\partial\vartheta_y}\dot{\vartheta}_y + \frac{\partial\,{}^0R_p}{\partial\vartheta_z}\dot{\vartheta}_z\right){}^0R_p^T\left(\Theta\right) \tag{2.39} $$

ed eguagliare le componenti omologhe della matrice $\left[{}^0\omega\times\right]$. Sviluppando la relazione (2.39) e ricordando che ${}^0R_c^T\left(\Theta\right) = R_x^T\left(\vartheta_x\right)R_y^T\left(\vartheta_y\right)R_z^T\left(\vartheta_z\right)$, si ha

$$ \begin{aligned} {}^0\dot{R}_p\,{}^0R_p^T = &\left(R_z\left(\vartheta_z\right)R_y\left(\vartheta_y\right)\tfrac{\partial R_x}{\partial\vartheta_x}\right)R_x^T\left(\vartheta_x\right)R_y^T\left(\vartheta_y\right)R_z^T\left(\vartheta_z\right)\dot{\vartheta}_x \\ &+\left(R_z\left(\vartheta_z\right)\tfrac{\partial R_y}{\partial\vartheta_y}R_x\left(\vartheta_x\right)\right)R_x^T\left(\vartheta_x\right)R_y^T\left(\vartheta_y\right)R_z^T\left(\vartheta_z\right)\dot{\vartheta}_y \\ &+\left(\tfrac{\partial R_z}{\partial\vartheta_z}R_y\left(\vartheta_y\right)R_x\left(\vartheta_x\right)\right)R_x^T\left(\vartheta_x\right)R_y^T\left(\vartheta_y\right)R_z^T\left(\vartheta_z\right)\dot{\vartheta}_z. \end{aligned} \tag{2.40} $$

Espandendo i prodotti e ricordando la proprietà di ortonormalità si ha

$$ \begin{aligned} {}^0\dot{R}_p\,{}^0R_p^T = &R_z\left(\vartheta_z\right)R_y\left(\vartheta_y\right)\left(\tfrac{\partial R_x}{\partial\vartheta_x}R_x^T\left(\vartheta_x\right)\right)R_y^T\left(\vartheta_y\right)R_z^T\left(\vartheta_z\right)\dot{\vartheta}_x \\ &+R_z\left(\vartheta_z\right)\left(\tfrac{\partial R_y}{\partial\vartheta_y}R_y^T\left(\vartheta_y\right)\right)R_z^T\left(\vartheta_z\right)\dot{\vartheta}_y \\ &+\left(\tfrac{\partial R_z}{\partial\vartheta_z}R_z^T\left(\vartheta_z\right)\right)\dot{\vartheta}_z. \end{aligned} \tag{2.41} $$

Si hanno poi le interessanti relazioni

$$\frac{\partial R_x}{\partial \vartheta_x} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix} R_x(\vartheta_x) = \left[{}^0x\times\right] R_x(\vartheta_x) \tag{2.42}$$

$$\frac{\partial R_y}{\partial \vartheta_y} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix} R_y(\vartheta_y) = \left[{}^0y\times\right] R_y(\vartheta_y) \tag{2.43}$$

$$\frac{\partial R_z}{\partial \vartheta_z} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} R_z(\vartheta_x) = \left[{}^0z\times\right] R_z(\vartheta_z) \tag{2.44}$$

facili da verificare, da cui si ricava

$$\frac{\partial R_x}{\partial \vartheta_x} R_x^T(\vartheta_x) = \left[{}^0x\times\right] \tag{2.45}$$

$$\frac{\partial R_y}{\partial \vartheta_y} R_y^T(\vartheta_y) = \left[{}^0y\times\right] \tag{2.46}$$

$$\frac{\partial R_y}{\partial \vartheta_y} R_y^T(\vartheta_y) = \left[{}^0z\times\right] \tag{2.47}$$

Ricordando inoltre che ${}^0R_p\,{}^0R_p^T = \left[{}^0\omega\times\right]$, ne consegue che

$$\begin{aligned}\left[{}^0\omega\times\right] = {} & R_z(\vartheta_z) R_y(\vartheta_y) \left[{}^0x\times\right] R_y^T(\vartheta_y) R_z(\vartheta_z)\dot{\vartheta}_x + \\ & + R_z(\vartheta_z) \left[{}^0y\times\right] R_z^T(\vartheta_z)\dot{\vartheta}_y + \left[{}^0z\times\right]\dot{\vartheta}_z\end{aligned} \tag{2.48}$$

È utile a questo punto richiamare la relazione (2.24) tra matrici di rotazione e matrici che e rappresentano prodotti vettoriali

$$\left[(R\,\omega)\times\right] = R\left[\,\omega\times\,\right]R^T$$

La equazione (2.48) diventa

$$\begin{aligned}\left[{}^0\omega\times\right] = {} & \left[\left(R_z(\vartheta_z) R_y(\vartheta_y)\,{}^0x\right)\times\right]\dot{\vartheta}_x + \\ & + \left[\left(R_z(\vartheta_z)\,{}^0y\right)\times\right]\dot{\vartheta}_y + \left[{}^0z\times\right]\dot{\vartheta}_z\end{aligned} \tag{2.49}$$

$$\begin{aligned}{}^0\omega = {} & R_z(\vartheta_z) R_y(\vartheta_y)\,{}^0x\dot{\vartheta}_x + \\ & + R_z(\vartheta_z)\,{}^0y\dot{\vartheta}_y + {}^0z\dot{\vartheta}_z\end{aligned} \tag{2.50}$$

Sviluppando i calcoli si ha

$$^0\omega = {}^0G(\Theta)\,\dot{\Theta} \tag{2.51}$$

dove

$${}^0G(\Theta) = \begin{pmatrix} c\vartheta_y c\vartheta_z & -s\vartheta_z & 0 \\ c\vartheta_y s\vartheta_z & c\vartheta_z & 0 \\ -s\vartheta_y & 0 & 1 \end{pmatrix} \tag{2.52}$$

Il determinante della matrice $G_0(\Theta)$è facilmente calcolabile ed è pari a $\cos\vartheta_y$. Pertanto la $G_0(\Theta)$ è singolare quando $\cos\vartheta_y = 0$, ovvero quando $\vartheta_y = \pm\frac{\pi}{2}$. In questi casi si ha

$${}^0G\left(\vartheta_x, \frac{\pi}{2}, \vartheta_z\right) = \begin{pmatrix} 0 & -s\vartheta_z & 0 \\ 0 & c\vartheta_z & 0 \\ -1 & 0 & 1 \end{pmatrix} \qquad {}^0G\left(\vartheta_x, -\frac{\pi}{2}, \vartheta_z\right) = \begin{pmatrix} 0 & -s\vartheta_z & 0 \\ 0 & c\vartheta_z & 0 \\ 1 & 0 & 1 \end{pmatrix} \tag{2.53}$$

Si può osservare che nelle due matrici la prima e l'ultima colonna sono proporzionali, e pertanto il contributo di ϑ_x e ϑ_z alla velocità angolare sono nella stessa direzione, infatti quando $\vartheta_y = \pm\frac{\pi}{2}$, l'asse x viene ruotato fino ad allinearsi con l'asse z.

La relazione inversa tra $\dot{\Theta}$ e $^0\omega$ si può quindi scrivere solo per $\cos\vartheta_y = 0$ $\left(\vartheta_y = \pm\frac{\pi}{2}\right)$, ed è data da

$$\dot{\Theta} = {}^0G^{-1}(\Theta)\ {}^0\omega \tag{2.54}$$

dove

$${}^0G^{-1}(\Theta) = \frac{1}{c\vartheta_y}\begin{pmatrix} c\vartheta_z & s\vartheta_z & 0 \\ -c\vartheta_y s\vartheta_z & c\vartheta_y c\vartheta_z & 0 \\ s\vartheta_y c\vartheta_z & s\vartheta_y s\vartheta_z & c\vartheta_y \end{pmatrix} \tag{2.55}$$

Negli orientamenti in cui l'angolo di pitch ϑ_y è pari a $\pm\frac{\pi}{2}$ la matrice ${}^0G(\Theta)$ non è invertibile, e pertanto non tutte le velocità angolari possono essere rappresentate mediante le derivate degli angoli RPY. Si ricordi inoltre che per tutti gli orientamenti caratterizzati dai valori $\pm\frac{\pi}{2}$ per l'angolo di pitch vi sono infinite rappresentazioni per mezzo degli angoli di roll e yaw.

Le relazioni tra la velocità angolare nelle coordinate di $^p\omega$ e la derivata degli

angoli di a Roll-Pitch-Yaw sono date da

$$ {}^{p}\omega = {}^{p}G(\Theta)\,\dot{\Theta} \quad con\ {}^{0}G(\Theta) = {}^{0}R_{p}^{T}(\Theta)^{p}G(\Theta) \tag{2.56} $$

$$ \dot{\Theta} = {}^{p}G^{-1}(\Theta)\,{}^{p}\omega\dot{\Theta} \quad con\ {}^{p}G^{-1}(\Theta) = {}^{0}G^{-1}(\Theta)\,{}^{0}R_{p} \tag{2.57} $$

$$ G_{p}^{-1}(\Theta) = \frac{1}{c\vartheta_y}\begin{pmatrix} c\vartheta_y & s\vartheta_x s\vartheta_y & c\vartheta_x s\vartheta_y \\ 0 & c\vartheta_x c\vartheta_y & -s\vartheta_x c\vartheta_y \\ 0 & s\vartheta_x & c\vartheta_x \end{pmatrix} \tag{2.58} $$

Gli orientamenti per i quali $\vartheta_y = \pm\frac{\pi}{2}$ sono detti *orientamenti singolari* per la rappresentazione con gli angoli di Roll-Pitch-Yaw. In altre parole, orientamenti del tipo $\Theta = \left(\vartheta_x, \pm\frac{\pi}{2}, \vartheta_z\right)$ costituiscono delle singolarità di rappresentazione. E bene sottolineare che tutte le rappresentazioni minime dell'orientamento di un corpo rigido hanno delle singolarità di rappresentazione.

2.2.3 Parte rotazionale

Dopo aver approfondito gli angoli di Roll Pitch e Yaw, e aver trovato la relzione che lega $\dot{\theta}$ e ω che richiamiamo rapidamente la relazione trovata

$$ \begin{pmatrix} \dot{\theta}_R \\ \dot{\theta_P} \\ \dot{\theta}_Y \end{pmatrix} = \frac{1}{\cos(\theta_P)}\begin{pmatrix} \cos(\theta_Y) & \sin(\theta_Y) & 0 \\ -\cos(\theta_P)\sin(\theta_Y) & \cos(\theta_P)\cos(\theta_Y) & 0 \\ \sin(\theta_P)\cos(\theta_Y) & \sin(\theta_P)\sin(\theta_Y) & \cos(\theta_P) \end{pmatrix}\begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} \tag{2.59} $$

Possiamo affrontare il problema del calcolo della dinamica della piattaforma, sottoposta a delle accelerazioni angolari α, avvalendoci dell' endomorfismo G per scrivere sempre le espressioni in funzione di ω, e non di $\dot{\theta}$.

Similmente alla parte traslazionale, le espressioni di velocità angolare e orientamento a partire dall' accelerazione angolare sono:

$$ \omega(t) = \int_0^t \alpha(t)dt \tag{2.60} $$

$$ \theta(t) = \int_0^t \dot{\theta}(t)dt = \int_0^t G^{-1}(\theta)\omega(t)dt \tag{2.61} $$

avendo supposto sia l' accelerazione angolare che la matrice $G^{-1}(\theta)$ costanti all' interno di un periodo di campionamento.

Con passaggi analoghi, otteniamo

$$\omega(t + kTs) = \alpha \int dt = \omega(kTs) + \alpha(kTs)t \tag{2.62}$$

$$\theta(t+kTs) = G^{-1} \int_{kTs}^{t+kTs} \omega(kTs)+\alpha t dt = \theta(kTs)+G^{-1}\omega(kTs)t+G^{-1}\alpha(kTs)\frac{t^2}{2} \tag{2.63}$$

avendo supposto sia l' accelerazione angolare α che la matrice $G^{-1}(\theta)$ costanti all' interno di un periodo di campionamento.

Valutando le espressioni per $t = Ts$ si ha

$$\theta((k+1)Ts) = \theta(kTs) + G^{-1}\omega(kTs)Ts + G^{-1}\alpha(kTs)\frac{Ts^2}{2} \tag{2.64}$$

$$\omega((k+1)Ts) = \omega(kTs) + G^{-1}\alpha(kTs)Ts \tag{2.65}$$

2.3 La dinamica della pallina

Una volta trovate le espressioni per ricostruire a partire dalle accelerazioni $\begin{pmatrix} a_k \\ \alpha_k \end{pmatrix}$ lo stato della piattaforma all' istante successivo $\begin{pmatrix} x_{k+1} \\ \dot{x}_{k+1} \\ \theta_{k+1} \\ \omega_{k+1} \end{pmatrix}$, possiamo passare ad affrontare il problema del moto della pallina, come già detto, dipendente dallo stato della piattaforma.

2.3.1 Le ipotesi di base

A tal scopo richiamiamo le già viste ipotesi semplificative fatte a monte del progetto, e vi estraiamo tutte le dovute implicazioni:

- **La pallina è *vincolata* al piatto: può muoversi solamente nel piano della piattaforma, non può "saltare"**

Grazie a questa ipotesi possiamo esprimere degli importanti vincoli sulle coordinate della pallina nel sistema di riferimento della piattaforma: la coordinata z, che corrisponde alla sua altezza rispetto al piatto, è costantemente uguale al raggio della pallina. Il vettore della sua posizione ha quindi forma ${}^{p}p = \begin{pmatrix} {}^{p}x(t) \\ {}^{p}y(t) \\ r_b \end{pmatrix}$, essendo ${}^{p}x$ e ${}^{p}y$ le generiche coordinate planari, variabili nel tempo, e r_b il raggio della pallina, ovviamente costante. Come diretta conseguenza il vettore velocità ${}^{p}\dot{p}$ avrà come terza componente, la derivata di una costante, pari a 0: avrà quindi la forma ${}^{p}\dot{p} = \begin{pmatrix} {}^{p}\dot{x}(t) \\ {}^{p}\dot{y}(t) \\ 0 \end{pmatrix}$.

- **La pallina è stata considerata un *punto materiale***

Su questo punto è importante fare chiarezza: ci sono di fatto 2 centri della pallina da prendere in considerazione. C'è il centro fisico, che come vedremo, nel programma di simulazione MATLAB è indicato con p, e serve per la rappresentazione grafica della pallina; fisicamente coincide con il suo baricentro. Oltre a questo, ci sono le coordinate della pallina rilevate dal touchscreen, che fisicamente corrispondono al suo punto di contatto con la piattaforma, indicate con $\bar{p}$: la relazione tra p *e* $\bar{p}$ nelle coordinate di terra è:

$$ {}^{0}p = {}^{0}\bar{p} + r_b {}^{0}R_p(\theta)\hat{z} \tag{2.66} $$

dove $\hat{z}$ è il versore $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$.

Nelle coordinate del piatto invece, ${}^{p}\bar{p}$ corrisponde semplicemente a ${}^{p}p$, a cui viene annullata la terza componente, cioè:

$$ {}^{p}\bar{p} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} {}^{p}p \tag{2.67} $$

La matrice $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$ è di fondamentale importanza in questa tesi e com-

pare moltissime volte anche nei capitoli successivi, quindi vale la pena dargli un nome, I_{03}.

$$I_{03} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{2.68}$$

Grazie all' ipotesi di punto materiale la pallina viene trattata, nell' algoritmo per il calcolo della traiettoria, come una sfera degenere di raggio nullo, in cui vale quindi la relazione

$$\bar{p} \equiv p$$

In conclusione, per svolgere i calcoli vongono usate le coordinate $\bar{p}$; p viene utilizzato solamente per visualizzare correttamente l' animazione

- **La massa della pallina e la sua influenza sul moto del robot è stata trascurata, poichè la sua forza peso è di un ordine di grandezza molto infieriore alle forze attuate dal polso robotico**

È l' ipotesi grazie alla quale le equazioni di moto della piattaforma sono indipendenti dallo stato della pallina.

2.3.2 Le leggi di moto della pallina

Tenendo a mente queste ipotesi possiamo iniziare a scrivere le leggi di moto della pallina; la prima cosa da fare, per questioni di praticità, è passare alle coordinate del sistema di riferimento solidale al piatto.

Siamo interessati, questa volta, a passare dalle coordinate di terra alle coordinate del piatto, perciò usiamo le formule inverse della (2.16) e della (2.25) esplicitando ${}^{p}p$ e ${}^{p}\dot{p}$: avremo

$${}^{p}p = {}^{p}R_0 \left({}^{0}p - {}^{0} o_{touch} \right) \tag{2.69}$$

$${}^{p}\dot{p} = {}^{p}R_0 \left({}^{0}\dot{p} - {}^{0}\dot{o}_{touch} - {}^{0}\omega \times {}^{0} r_{ball} \right) \tag{2.70}$$

${}^0r_{ball}$ rapresenta il braccio della rotazione; avendo introdotto la disinzione tra p e $\bar{p}$, possiamo affermare che ${}^0r_{ball}$ equivale a $\bar{p} - {}^0\dot{o}_{touch}$.

A questo punto bisogna fare una considerazione di cruciale importanza per la validità di tutto il discorso a seguire.

Nelle ipotesi semplificative enunciate fin dall' inizio, abbiamo ipotizzato che la pallina sia sempre vincolata al piatto: questo ci ha portato ad affermare che nelle coordinate ${}^pS$ la sua altezza rimanga costantemente uguale al raggio, e la sua velocità verticale sempre nulla. Nell' imporre tali vincoli sulle coordinate, tramite le equazioni che vedremo, avremo le componenti z di posizione e velocità della pallina fisse; ciò che si calcolerà attraverso le leggi dinamiche sono solamente le componenti x e y. Nelle coordinate della piattaforma si avrà perciò un moto *bidimensionale*, *planare*.

Alla luce di queste considerazioni possiamo affermare che, per la nostra analisi, ogni accelerazione perpendicolare alla piattaforma non dà alcun contributo alle leggi dinamiche, poichè la componente z della velocità è stata imposta costante e pari a zero: non danno perciò contributo l'accelerazione dovuta alla reazione vincolare del piatto, difficile da modellizzare soprattutto nel caso di piatto in movimento, e l' accelerazione impressa dalla rotazione del touchscreen, poichè entrambe sono vettori perpendicolari alla piattaforma.

Le uniche accelerazioni efficaci, cioè non perpendicolari alla piattaforma, sono quelle imposte dagli attuatori sul polso robotico e quella gravitazionale, prese nelle loro componenti parallele al piatto. Per annullare la componente z si usa un prodotto per la matrice I_{03} già definita nel paragrafo precedente in formula (2.68) ottenendo la proiezione dell' accelerazione risultante sul piano del touchscreen:

$$ {}^pa_{//} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} {}^pR_0 \left({}^0a_{trasl} + {}^0g \right) \tag{2.71} $$

Il vettore accelerazione ${}^pa_{//}$ avrà sempre la forma

$$ {}^pa_{//} = \begin{pmatrix} {}^pa_x \\ {}^pa_y \\ 0 \end{pmatrix} \tag{2.72} $$

A questo punto usiamo le formule (2.9) e (2.10) ben note di calcolo della traiettoria nel discreto a partire dall' accelerazione e dallo stato, proiettando

opportunamente le traiettorie sulla piattaforma si ottiene:

$$ {}^{p}p((k+1)Ts) = \begin{pmatrix} 0 \\ 0 \\ r \end{pmatrix} + I_{03}\,{}^{p}p(kTs) + I_{03}\,{}^{p}v(kTs)Ts + {}^{p}a_{//}(kTs)\frac{Ts^2}{2} \quad (2.73) $$

$$ {}^{p}v((k+1)Ts) = I_{03}\,{}^{p}v(kTs) + {}^{p}a_{//}(kTs)Ts \quad (2.74) $$

Conformemente alle ipotesi fatte, i vettori di posizione e velocità non subiranno mai una variazione sulla terza componente, mantenendole sempre pari a ${}^{p}p_z = r_b$ e ${}^{p}v_z = 0$. La loro forma sarà sempre del tipo

$$ {}^{p}p = \begin{pmatrix} {}^{p}p_x \\ {}^{p}p_y \\ r_b \end{pmatrix} \quad (2.75) $$

$$ {}^{p}v = \begin{pmatrix} {}^{p}v_x \\ {}^{p}v_y \\ 0 \end{pmatrix} \quad (2.76) $$

garantendo la validità dell' ipotesi di pallina vincolata alla piattaforma.

Ottenuti i vettori di posizione e velocità nelle coordinate della piattaforma, non resta che riportarli alle coordinate di terra attraverso le leggi (2.16) e (2.25) usate questa volta in formula diretta:

$$ {}^{0}p((k+1)Ts) = {}^{0}o_{touch} + {}^{0}R_{p}\,{}^{p}p((k+1)Ts) $$

$$ {}^{0}v((k+1)Ts) = {}^{0}\dot{o}_{touch} + {}^{0}R_{p}(t)\,{}^{p}v((k+1)Ts) + {}^{0}\omega \times {}^{0}r_{ball} $$

Chapter 3

La programmazione delle traiettorie

Nel linguaggio comune, con il termine traiettoria si intende il percorso da fare per muoversi da un punto ad un altro; in robotica il significato dell' espressione è molto più ampio. La prima distinzione da fare è tra traiettoria nello spazio dei giunti, e nello spazio operativo:

Lo spazio dei giunti è un sottoinsieme di $\mathbb{R}^n$, dove con n si intende il numero di gradi di libertà del robot, in cui ogni punto si trova in corrispondenza biunivoca con una possibile configurazione dei giunti. Una configurazione del robot è quindi identificata da un vettore delle variabili di giunto

$$q = \begin{pmatrix} q_1 \\ q_2 \\ . \\ . \\ . \\ q_{n-1} \\ q_n \end{pmatrix} \tag{3.1}$$

in cui l' elemento n-esimo rappresenta la posizione del giunto n-esimo. Un percorso nello spazio dei giunti è una curva γ in $\mathbb{R}^n$, detta *cammino geometrico*,

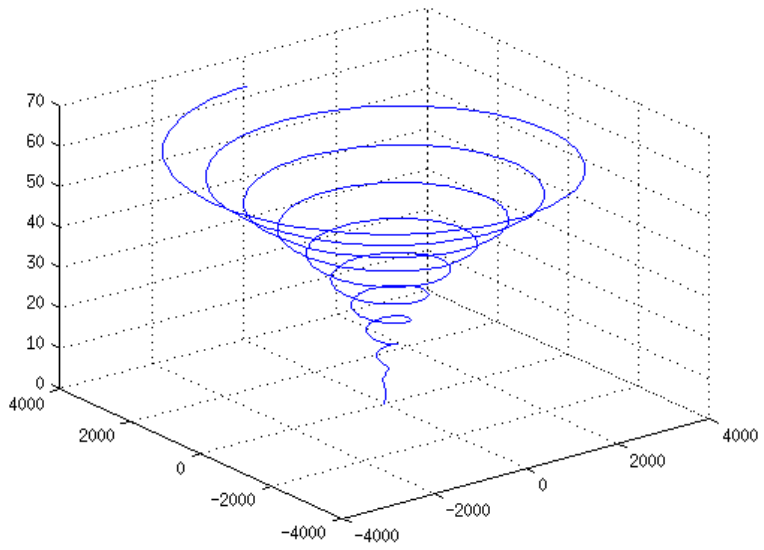

Figure 3.1: Un esempio di generica curva γ in uno spazio $\mathbb{R}^3$

i cui punti sono l' insieme delle configurazioni che assume il robot.

Se si vuole programmare un movimento, γ non è sufficiente ad identificarlo, ed è necessaria anche una funzione $s(t)$, detta *legge oraria,* che determini il profilo temporale di percorrenza. La traiettoria vera e propria corrisponde alla funzione $\gamma(s(t))$, in cui ad ogni istante di tempo $\bar{t}$ è associato un vettore

$$\gamma(s(\bar{t})) = \begin{pmatrix} \gamma_1(s(\bar{t})) \\ \gamma_2(s(\bar{t})) \\ . \\ . \\ . \\ \gamma_{n-1}(s(\bar{t})) \\ \gamma_n(s(\bar{t})) \end{pmatrix} \tag{3.2}$$

che identifica le coordinate del punto dello spazio dei giunti in cui ci si troverà all'istante $\bar{t}$.

Lo spazio operativo è un sottoinsieme di $\mathbb{R}^m$ in cui solitamente viene definito il compito da svolgere; con m si indica il numero di gradi di libertà necessari al suo compimento: ad esempio se il robot deve svolgere un compito planare,

come scrivere su una lavagna, i gradi di libertà di cui necessita sono due, e lo spazio operativo è il piano della lavagna; se invece deve effettuare un compito nello spazio tridimensionale in cui non è importante solo la posizione del polso, ma anche il suo orientamento, saranno necessari sei gradi di libertà, tre per la posizione e altri tre per l' orientamento. Un punto nello spazio operativo è rappresentato dal vettore

$$x = \begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_{m-1} \\ x_m \end{pmatrix} \tag{3.3}$$

È da rimarcare che affinchè il compito sia realizzabile dal robot considerato deve valere la condizione $m \leq n$. Esiste una funzione che fa corrispondere univocamente ad ogni punto nello spazio dei giunti, il corrispondente punto nello spazio operativo, detta *funzione cinematica*:

$$x = k(q) \tag{3.4}$$

generalmente non lineare e dipendente dalla struttura meccanica del robot.

Nelle applicazioni reali è certamente più comodo definire una traiettoria nello spazio operativo, poichè rappresenta direttamente il compito da svolgere: si definisce in modo analogo a quanto già visto una curva γ e la legge oraria $s(t)$. È da evidenziare il fatto che l' interpretazione di traiettoria è intuitiva fin quando si parla solamente compiti di posizione, in cui γ rappresenta la curva descrive il percorso che seguirà il **TCP** nello spazio cartesiano, ma diventa meno intuitivo quando si parla di orientamento; una traiettoria per l' orientamento, di fatto, non ammette interpretazione fisica, ma descrive solamente i valori che assumono le variabili angolari nel tempo. In questa tesi è decisamente più centrale il problema delle traiettorie di orientamento, rispetto a quelle di posizione, poichè per mantenere stabile la pallina è necessaria innanzitutto una rotazione della piattaforma touchscreen.

La programmazione di una traiettoria si articola in due fasi:

- scegliere un cammino geometrico γ opportuno per le esigenze

- scegliere una lagge oraria $s(t)$ con la quale percorrere la curva γ

Per definire in modo opportuno il cammino geometrico in $\mathbb{R}^m$ non si può usare una rappresentazione esplicita del tipo $x_m = f(x_1, x_2, ..., x_{m-1})$, ma è necessaria una forma parametrica del tipo

$$\begin{cases} x_1 = f_1(p) \\ x_2 = f_2(p) \\ . \\ . \qquad\qquad\qquad p \in \mathbb{R} \\ . \\ x_{m-1} = f_{m-1}(p) \\ x_m = f_m(p) \end{cases} \tag{3.5}$$

A questo punto si può inserire a posto di p la legge oraria $s(t)$ in modo da ottenere una parametrizzazione della traiettoria $\gamma(s(t))$

$$\begin{cases} x_1(t) = f_1(s(t)) \\ x_2(t) = f_2(s(t)) \\ . \\ . \\ . \\ x_{m-1}(t) = f_{m-1}(s(t)) \\ x_m(t) = f_m(s(t)) \end{cases} \tag{3.6}$$

Nel nostro caso, non avendo particolari esigenze, scegliamo la traiettoria più semplice possibile: il segmento. Dati due punti nello spazio operativo A e B

$$A = \begin{pmatrix} x_{A1} \\ x_{A2} \\ . \\ . \\ . \\ x_{Am-1} \\ x_{Am} \end{pmatrix}, \qquad B = \begin{pmatrix} x_{B1} \\ x_{B2} \\ . \\ . \\ . \\ x_{Bm-1} \\ x_{Bm} \end{pmatrix}$$

la parametrizzazione del segmento si ottiene come

$$\begin{cases} x_1 = x_{A1} + s(t)(x_{B1} - x_{A1}) \\ x_2 = x_{A2} + s(t)(x_{B2} - x_{A2}) \\ . \\ . \\ . \\ x_{m-1} = x_{Am-1} + s(t)(x_{Bm-1} - x_{Am-1}) \\ x_m = x_{Am} + s(t)(x_{Bm} - x_{Am}) \end{cases} \tag{3.7}$$

che si può scrivere in forma vettoriale più compatta come

$$x(t) = x_A + s(t)(x_B - x_A) \tag{3.8}$$

dove si deve rispettare la condizione sulla $s(t)$:

$$s(t) \in [0, 1] \qquad \forall t$$

Resta adesso il problema di come scegliere la legge oraria .

3.1 Profili di velocità

Per la scelta di una $s(t)$ buona per affrontare il nostro problema dobbiamo considerare vari aspetti:

- il moto dei bracci non deve sollecitare eccessivamente la struttura meccanica del robot
- non devono essere presenti discontinuità nella velocità, che oltre a sollecitare la struttura meccanica, farebbero con ampia probabilità staccare la pallina dalla piattaforma, cosa assolutamente da evitare, e nemmeno prevista nella simulazione su calcolatore
- per ipotesi semplificativa, non imponiamo alcun limite massimo all' accelerazione
- si cerca un profilo efficace, ma comunque semplice

Considerando queste specifiche, la scelta migliore è il profilo di velocità triangolare.

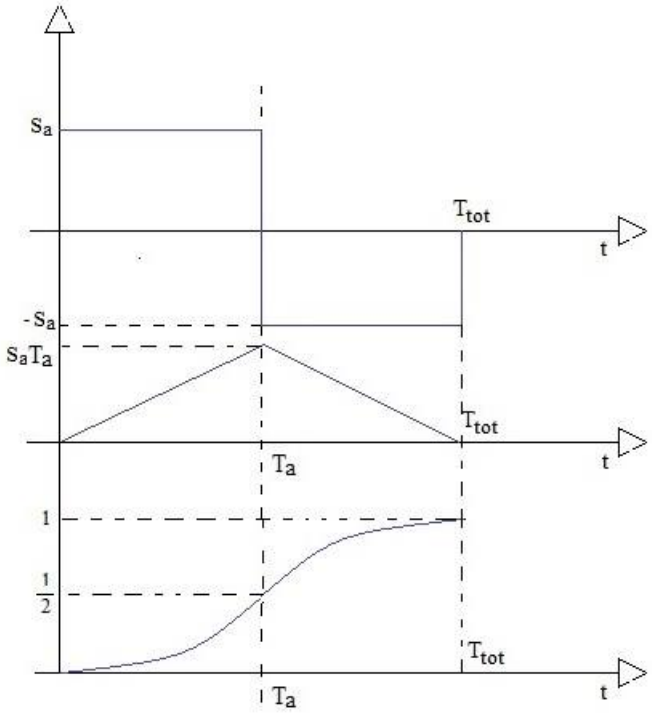

Figure 3.2: Profili ideali

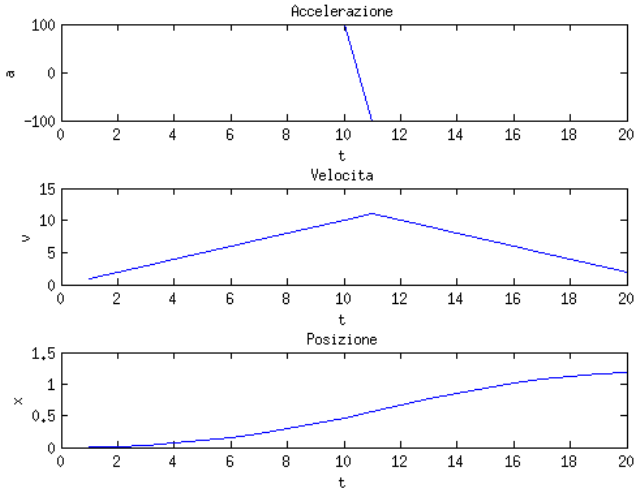

Figure 3.3: Profili campionati

Nel programmare questo tipo di traiettoria si parte sempre dall' accelerazione $\ddot{s}(t)$: si sceglie un tempo T_a e si traccia una funzione costituita da un primo tratto positivo e costante di durata T_a e un secondo opposto al primo, anch' esso di durata T_a: affinchè sia rispettata la condizione desiderata di velocità nulla al termine della traiettoria, deve valere

$$\int_0^{T_{tot}} a(t)dt = 0 \tag{3.9}$$

che è una prima condizione da rispettare per la corretta programmazione di una traiettoria. T_{tot} è il tempo totale in cui viene portata a termine la traiettoria, ed è pari a $2T_a$. Nella simulazione su MATLAB, come si vede dalla figura (3.3), è stato scelto un tempo di accelerazione T_a pari a 10 passi di campionamento, e perciò la durata complessiva è di T_{tot} di 20 passi di campionamento.

Un' altra condizione sulla $s(t)$ da rispettare è

$$\begin{cases} s(0) = 0 \\ s(T_{tot}) = 1 \end{cases} \tag{3.10}$$

che assicura che la traiettoria parta dal punto A e termini nel punto B:

$$\begin{cases} \gamma(s(0)) \equiv A \\ \gamma(s(T_{tot})) \equiv B \end{cases} \tag{3.11}$$

Adesso resta il problema di come scegliere il valore s_a da assegnare all' accelerazione nel tratto costante affinchè la traiettoria sia portata a termine rispettando tutte le condizioni sovraelencate: per fare questo si impone

$$s(T_a) = \frac{1}{2} \tag{3.12}$$

quindi si scrive l' equazione dell' accelerazione, si integra due volte per ottenere la posizione, e si trova il valore di $s_a$che soddisfi la (3.12).

Definiamo a tratti la funzione di accelerazione $\ddot{s}(t)$:

$$\begin{cases} s_a & t \in [0, T_a] \\ -s_a & t \in [T_a, T_{tot}] \end{cases} \tag{3.13}$$

la integriamo la prima volta ottenendo la velocità $\dot{s}(t)$, considerando sempre le condizioni iniziali

$$\begin{cases} s_a t & t \in [0, T_a] \\ -s_a(2T_a - t) & t \in [T_a, T_{tot}] \end{cases} \tag{3.14}$$

e integriamo la seconda volta per ottenere la posizione $s(t)$

$$\begin{cases} \frac{1}{2} s_a t^2 & t \in [0, T_a] \\ 1 - \frac{1}{2} s_a (2T_a - t)^2 & t \in [T_a, T_{tot}] \end{cases} \tag{3.15}$$

Imponendo ora la condizione (3.12) otteniamo le due equazioni equivalenti

$$\begin{cases} \frac{1}{2} s_a T_a^2 = \frac{1}{2} \\ 1 - \frac{1}{2} s_a (2T_a - T_a)^2 = \frac{1}{2} \end{cases} \tag{3.16}$$

che hanno per soluzione

$$s_a = \frac{1}{T_a^2} \tag{3.17}$$

Abbiamo adesso tutti gli strumenti per programmare una traiettoria a profilo di velocità triangolare. Ora studiamo come applicarla

3.2 Applicazione del profilo di velocità

Una volta trovati tutti i parametri opportuni per costruire la legge oraria, bisogna applicarla al cammino geometrico γ scelto, nel nostro caso il segmento, considerando sia le traslazioni che le rotazioni: l' interpretazione di traiettoria traslazionale è piuttosto intuitiva; lo è meno quella di traiettoria nello spazio degli orientamenti.

3.2.1 Traslazione

Considerando il punto iniziale ${}^0a$ e il punto finale ${}^0b$ l' espressione della traiettoria è esattamente la (3.8)

$$x(t) = x_A + s(t)(x_B - x_A)$$

considerando il caso campionato

$$x(k) = x_A + s(k)(x_B - x_A) \qquad k \in [0\,,\, T_{tot}]$$

3.2.2 Rotazione

Considerando piccole rotazioni, possiamo trattare la traiettoria nello spazio degli orientamenti come una traiettoria nello spazio degli angoli di Eulero. In tal caso avremo una formula simile a quella per la traslazione:

$$\theta(t) = \theta_A + s(t)(\theta_B - \theta_A)$$

nel discreto

$$\theta(k) = \theta_A + s(k)(\theta_B - \theta_A) \qquad k \in [0\,,\, T_{tot}]$$

Chapter 4

Il controllo PD

Il controllore PID è uno dei regolatori più diffusi nell' ambito dei controlli automatici: valutando in ingresso l' errore rispetto allo stato di equilibrio, imprime un'azione di controllo tale da riportarlo al valore nullo. La sua equazione nel dominio continuo del tempo è:

$$y(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

dove sono presenti tre termini:

- il termine *proporzionale* (P) rappresenta l' azione primaria del controllore: più l' errore è grande, più l' uscita sarà grande e di conseguenza più sarà forte l'azione degli attuatori per riportare il sistema allo stato di equilibrio. Guadagni grandi perciò, implicano grande prontezza nella risposta del sistema, ma di contro provocano oscillazioni molto pronunciate e tempi di assestamento piuttosto lunghi, oltre a un dispendio energetico eccessivo.
- il termine *integrale* (I) è l' unico dei tre termini che possiede memoria, e tende a minimizzare il discostamento dello stato dal suo valore di equilibrio. È molto importante perchè in situazioni in cui il termine proporzionale non è sufficientemente forte da portare l' errore a zero, causando un errore a regime permanente, tende ad annullarlo: in tal caso, infatti, il termine proporzionale rimane costante, poichè l' errore è costante, mentre quello

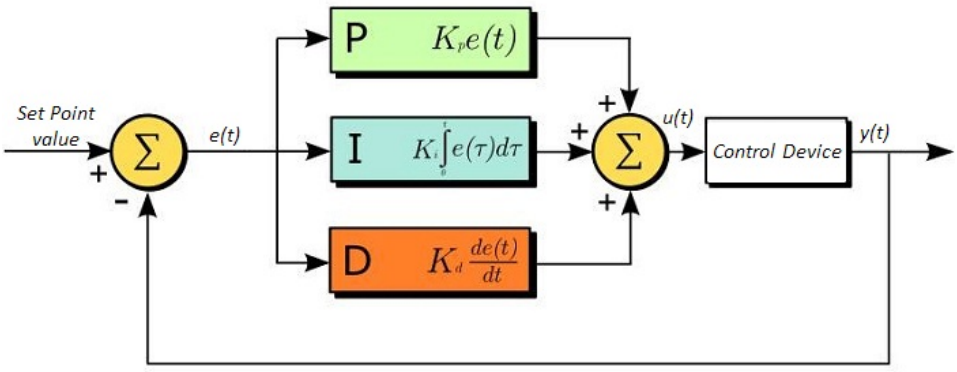

Figure 4.1: Lo schema di un controllore PID

integrale cresce linearmente nel tempo, poichè l' integrale di una costante è una retta crescente. Crescendo $\int_0^t e(t)dt$, l'uscita aumenterà e raggiungerà un valore sufficiente per annullare l' errore. Tuttavia in molte applicazioni si evita di inserire questo termine, perchè può portare problemi di instabilità, in particolare il fenomeno del wind-up.

- il termine *derivativo* (D) è un termine proporzionale alla derivata dell' errore, cioè alla sua variazione: intuitivamente si capisce che questo termine, in un certo senso, rallenta l' avvicinamento dell' errore allo 0, smorzando le sue oscillazioni: con valori del guadagno K_d troppo alti si rischia di avere una risposta con bassa prontezza, con tempi di salita molto lunghi, e conseguenti tempi di assestamento lunghi. Con un guadagni troppo bassi, al contrario, si rischia un sistema con risposta troppo sensibile e rapida, con ampie oscillazioni.

La diffusione di questo tipo di controllori è dovuta ai notevoli vantaggi che essi presentano:

- sono molto semplici da progettare
- non si ha necessariamente bisogno di conoscere il modello matematico del processo per implementarli

- i parametri regolatori sono solamente tre: il guadagno proporzionale, integrativo e derivativo
- nella maggior parte delle applicazioni hanno prestazioni accettabili
- possono essere implementati anche con circuiti analogici (tramite amplificatori operazionali in configurazioni come l' integratore di Miller), o addirittura con sistemi meccanici o idraulici
- lo schema a blocchi può essere progettato in diverse varianti

D'altra parte presentano anche degli svantaggi

- con tre soli gradi di libertà non si possono ottenere prestazioni paragonabili a controllori più articolati
- in alcuni applicazioni non sono adatti, e non stabilizzano il sistema

Nella nostra applicazione verrà controllato l' orientamento della piattaforma attraverso un controllore PD, composto cioè solamente dal termine proporzionale e quello derivativo; il motivo di ciò è il fatto che nel controllo LQR che descriveremo più tardi, abbiamo come variabili controllate solamente l' errore di posizione $p - x$ e la sua derivata, l' errore di velocità $\dot{p} - \dot{x}$, quindi si è cercato di avere controllori più simili possibile in modo da poter fare un paragone più equo. La derivata non viene eseguita direttamente, ma viene utilizzata la velocità già calcolata tramite le formule descritte nel capitolo della dinamica Newtoniana. Il controllo è stato fatto una variabile per volta: c'è un PD per controllare l' angolo di Roll, e un altro per il Pitch, ovviamente con guadagni uguali; l' angolo di Yaw, per la nostra applicazione, è un grado di libertà ridondante, e viene sempre posto a 0. L' orientamento finale è espresso come vettore con gli angoli di Roll Pitch e Yaw. Il controllo, come verrà spiegato in dettaglio nel capitolo sull' implementazione MATLAB, prevede anche la scelta di una *frequenza di azione*, dettata dal parametro `Ta`, che rappresenta gli istanti di campionamento che intercorrono tra un' azione e l' altra: in quel lasso di tempo gli attuatori del robot portano la piattaforma nell' orientamento stabilito dal controllore attraverso un profilo di velocità triangolare. La scelta del parametro `Ta`, quindi, è molto importante per la progettazione del sistema di controllo e

va abbinata con quella dei guadagni: un valore di `Ta` alto riserva un tempo lungo per ruotare la piattaforma, che quindi compierà un movimento armonioso e senza scatti; di contro si ha una frequenza di azione molto bassa, che potrebbe essere insufficiente per stabilizzare il sistema. Con un valore basso di `Ta`, invece, la frequenza di azione è alta, ma si hanno ingressi di controllo elevati e rotazioni molto brusche della piattaforma che, con alti guadagni, potrebbe raggiungere accelerazioni angolari tali da fal saltare la pallina. Nella simulazione si è considerato un tempo `Ta` pari a 16 campioni, ma nell' implementazione sul robot reale molto probabilmente si dovranno effettuare dei ritocchi su questa scelta, e in generale su quella di tutti i parametri.

4.1 Risultati

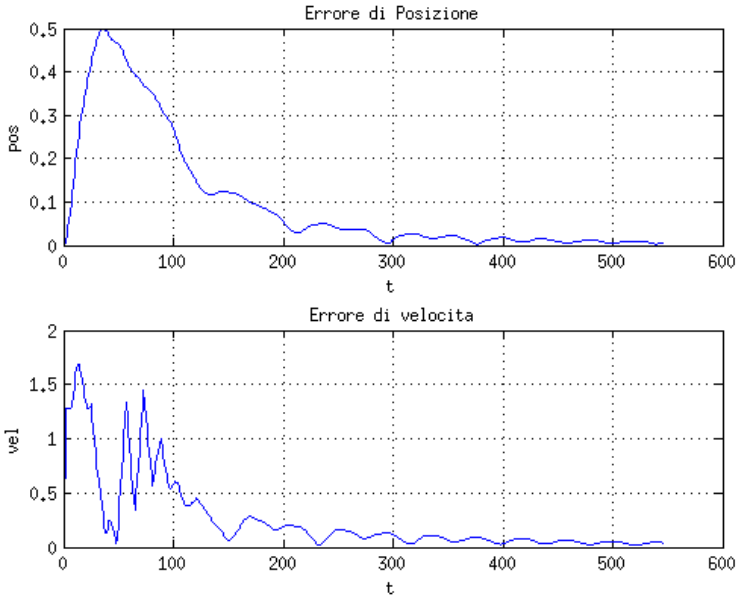

Figure 4.2: La norma degli errori di posizione e velocità

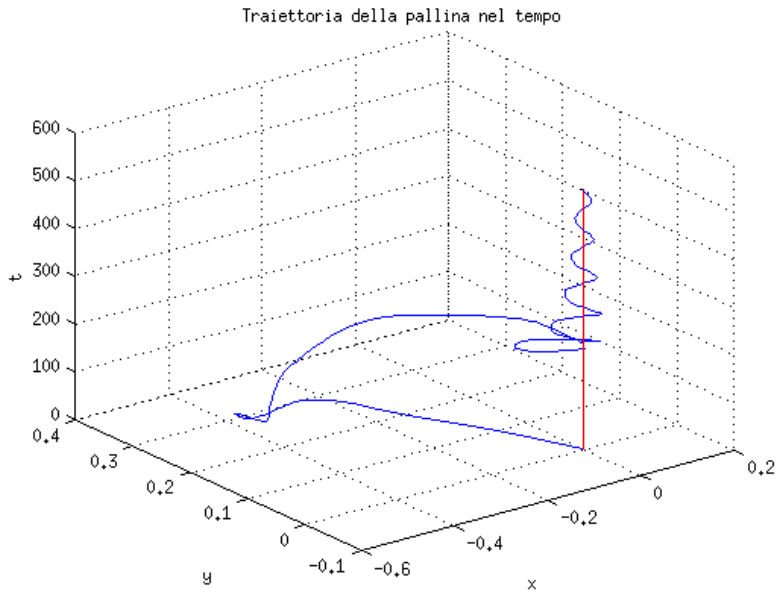

Figure 4.3: Grafico tridimensionale dell' errore di posizione

Le prestazioni del controllo PD possono essere dedotte dai grafici che verranno riportati in seguito; il primo grafico da analizzare è quello in figura 4.2, che riporta l' andamento gli errori di posizione e velocità, in norma. Notiamo nella posizione una grande sovraelongazione iniziale smorzata in tempi abbastanza lunghi e con la presenza di significative oscillazioni. Passando al grafico delle velocità si nota un profilo notevolmente frastagliato con ripide oscillazioni: questo è l' effetto di un tempo di azione `Ta` relativamente basso che si è scelto in simulazione, per evitare instabilità; anche qui la stabilizzazione avviene in maniera piuttosto lenta e con visibili oscillazioni attorno al punto di equilibrio.

Nel grafico in figura 4.3 è mostrata l' evoluzione della posizione in un grafico tridimensionale, in cui abbiamo il tempo sull' asse verticale, e la posizione sugli altri due assi. Si possono osservare le stesse cose viste nel precedente grafico, ma con una visione più completa. Si notano le oscillazioni intorno al punto di equilibrio che formano una spirale convergente.

Nella figura 4.4 sono graficate le accelerazioni lineari e angolari impresse alla piattaforma, sempre in norma. L' accelerazione lineare è costantemente nulla poichè tale controllore agisce solo sull' orientamento della piattaforma. L' accelerazione angolare invece è una funzione costante a tratti, poichè si usa un

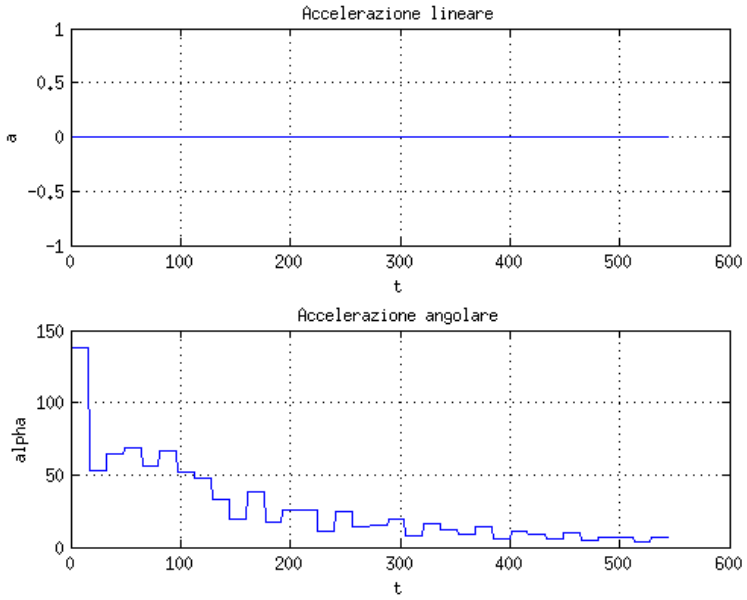

Figure 4.4: Accelerazioni lineari e angolari

profilo di velocità triangolare, nel quale l' accelerazione è un periodo di onda quadra, come mostrato nella figura 3.3 del capitolo sulla programmazione delle traiettorie.

Chapter 5

Controllo LQR del sistema linearizzato

Il controllore PID è molto diffuso per la sua semplicità e per il fatto che non richiede il modello matematico del sistema per effettuare i calcoli; di contro, le sue prestazioni non sono sempre eccellenti ed esistono molti regolatori decisamente migliori; tra questi troviamo quelli operanti nello spazio di stato. Per ottenere una rappresentazione su spazio di stato per un sistema non lineare come questo, è necessaria una linearizzazione attorno al punto di equilibrio, da cui si ottiene un sistema nella forma $\mathbf{q}(k+1) = A\mathbf{q}(k) + B\mathbf{u}(k)$; a quel punto possono essere utilizzate tecniche di controllo lineare. Una possibilità potrebbe essere l' assegnazione degli autovalori attraverso una controreazione dello stato, ma risulta piuttosto complicata per sistemi a più ingressi come questo, perciò si opta per un controllo ottimo lineare quadratico, abbreviato spesso con LQR. La parola "ottimo" va intesa nel suo significato latino di superlativo, ossia "il migliore": questo regolatore infatti è basato su una cotroreazione dello stato in cui la matrice dei guadagni K è calcolata in modo da minimizzare un dato funzionale di costo, e di conseguenza, da ottenere le prestazioni migliori possibili.

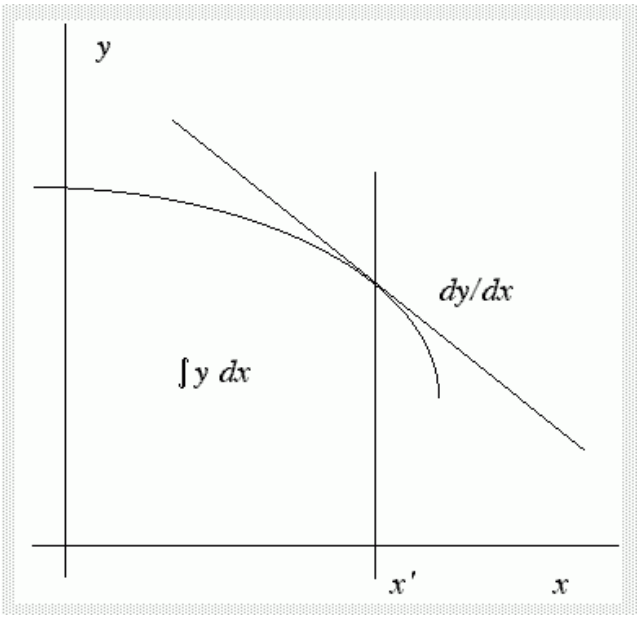

Figure 5.1: Approssimazione lineare in una dimensione

5.1 La linearizzazione intorno al punto di equilibrio

Per ottenere un sistema lineare del tipo $\mathbf{q}_{k+1} = A\mathbf{q}_k + B\mathbf{u}_k$, occorre innanzitutto esprimere la fisica del sistema tramite un' equazione vettoriale non lineare del tipo $\mathbf{q}_{k+1} = f(\mathbf{q}_k, \mathbf{u}_k)$. Una volta ottenuta una forma di questo tipo si può calcolare il sistema linearizzato attorno al punto di equilibrio, detto anche sistema tangente: infatti, facendo il parallelismo con una funzione monodimensionale, non si fa altro che approssimare una curva non lineare con la sua retta tangente nel punto prescelto; il comportamento della funzione non lineare, in un intorno del punto di tangenza, è molto simile a quello della retta tangente.

Tale approssimazione coincide con lo sviluppo di Taylor troncato al primo ordine:

$$\bar{f}(x) = f(x') + \frac{df}{dx}(x - x') \tag{5.1}$$

dove con $\bar{f}(x)$ si intende la funzione linearizzata intorno al punto x'.

Nel sistema piatto-pallina, trovandoci in uno spazio di stato multidimensionale composto da stato e ingresso, avremo una funzione non lineare del tipo

$\mathbf{q}_{k+1} = f(\mathbf{q}_k, \mathbf{u}_k)$, ricordando la definizione di stato $\mathbf{q}$ e ingresso $\mathbf{u}$ del sistema complessivo

$$\mathbf{q} = \begin{pmatrix} x \\ \dot{x} \\ \theta \\ \omega \\ p \\ \dot{p} \end{pmatrix} \qquad \mathbf{u} = \begin{pmatrix} \mathrm{a} \\ \alpha \end{pmatrix} \tag{5.2}$$

Nei paragrafi successivi verrà mostrato che per il effettuare il controllo è opportuno avere una forma diversa per lo stato $\mathbf{q}$ che sarà ottenuta tramite un cambio di coordinate ad opera di una trasformazione T

$$T = \begin{pmatrix} I & O & O & O & O & O \\ O & I & O & O & O & O \\ O & O & I & O & O & O \\ O & O & O & I & O & O \\ -I & O & O & O & I & O \\ O & -I & O & O & O & I \end{pmatrix} \tag{5.3}$$

Lo stato subirà una trasformazione per cui le sue ultime due coordinate rappresenteranno l' errore di posizione e velocità della pallina

$$\tilde{\mathbf{q}} = \begin{pmatrix} I & O & O & O & O & O \\ O & I & O & O & O & O \\ O & O & I & O & O & O \\ O & O & O & I & O & O \\ -I & O & O & O & I & O \\ O & -I & O & O & O & I \end{pmatrix} \begin{pmatrix} x \\ \dot{x} \\ \theta \\ \omega \\ p \\ \dot{p} \end{pmatrix} = \begin{pmatrix} x \\ \dot{x} \\ \theta \\ \omega \\ p - x \\ \dot{p} - \dot{x} \end{pmatrix}$$

Si considera adesso la funzione vettoriale non lineare che discrive il sistema

$$f(\mathbf{q}, \mathbf{u}) = \begin{pmatrix} f_x(\mathbf{q}, \mathbf{u}) \\ f_{\dot{x}}(\mathbf{q}, \mathbf{u}) \\ f_\theta(\mathbf{q}, \mathbf{u}) \\ f_\omega(\mathbf{q}, \mathbf{u}) \\ f_p(\mathbf{q}, \mathbf{u}) \\ f_p(\mathbf{q}, \mathbf{u}) \end{pmatrix} \tag{5.4}$$

si deve scegliere una fissata coppia stato-ingresso, attorno a cui linearizzare il sistema: come di convenzione, si sceglie il punto di equilibrio, così che il sistema linearizzato si comporti in maniera più fedele possibile in un intorno di questo punto. Si porrà $\mathbf{q}_\mathrm{e} = \begin{pmatrix} O & O & O & O & O & O \end{pmatrix}^T$ e $\mathbf{u}_e = \begin{pmatrix} O & O \end{pmatrix}^T$ corrispondente alla situazione in cui la piattaforma è nella sua posizione di inizializzazione, con la pallina posta sul centro, in quiete. È da precisare che per lo stato d' equilibrio $\mathbf{q}_\mathrm{e}$ è stata fatta una semplificazione che rende i calcoli molto meno laboriosi, e che ha un influenza assolutamente trascurabile sulla linearizzazione: il vero punto di equilibrio del sistema piatto-pallina è lo stato

$$\mathbf{q}_\mathrm{e} = \begin{pmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 0 \\ 0 \\ r \end{bmatrix} \\ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \end{pmatrix}$$

con la terza componente della posizione p della pallina non nulla, bensì pari al suo raggio, valore da noi trascurato.

Scelto il punto di equilibrio si procede con la linearizzazione usando lo sviluppo di Taylor troncato al primo ordine, equivalente alla (5.1) generalizzata al caso multi dimensionale:

$$\bar{f}(\mathbf{q},\mathbf{u}) = f(\mathbf{q}_e,\mathbf{u}_e) + \left.\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}}\right|_{\mathbf{q}=\mathbf{q}_e,\mathbf{u}=\mathbf{u}_e} (\mathbf{q}-\mathbf{q}_e) + \left.\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}}\right|_{\mathbf{q}=\mathbf{q}_e,\mathbf{u}=\mathbf{u}_e} (\mathbf{u}-\mathbf{u}_e) \tag{5.5}$$

ottenendo un sistema linearizzato in cui lo stato è rappresentato dall' errore $(\mathbf{q}-\mathbf{q}_e)$ che è usuale scrivere come $\delta\mathbf{q}$, e l' ingresso dall' errore $(\mathbf{u}-\mathbf{u}_e)$, quindi $\delta\mathbf{u}$. Il vantaggio di aver posto come punto di equilibrio lo stato e l' ingresso nullo è che valgono le relazioni $\mathbf{q} = \delta\mathbf{q}$ e $\mathbf{u} = \delta\mathbf{u}$ e, come si potrà verificare facilmente, $f(\mathbf{q}_e,\mathbf{u}_e) = 0$. La formula perciò si semplifica ulteriormente fino a diventare:

$$\bar{f}(\mathbf{q},\mathbf{u}) = \left.\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}}\right|_{\mathbf{q}=\mathbf{q}_e,\mathbf{u}=\mathbf{u}_e} \mathbf{q} + \left.\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}}\right|_{\mathbf{q}=\mathbf{q}_e,\mathbf{u}=\mathbf{u}_e} \mathbf{u} \tag{5.6}$$

La derivata parziale rispetto ad un vettore, di una funzione vettoriale è una matrice detta matrice Jacobiana, definita come:

$$\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}} = \begin{pmatrix} \frac{\partial f_x(\mathbf{q},\mathbf{u})}{\partial x} & \frac{\partial f_x(\mathbf{q},\mathbf{u})}{\partial \dot{x}} & \frac{\partial f_x(\mathbf{q},\mathbf{u})}{\partial \theta} & \frac{\partial f_x(\mathbf{q},\mathbf{u})}{\partial \omega} & \frac{\partial f_x(\mathbf{q},\mathbf{u})}{\partial p} & \frac{\partial f_x(\mathbf{q},\mathbf{u})}{\partial \dot{p}} \\ \frac{\partial f_{\dot{x}}(\mathbf{q},\mathbf{u})}{\partial x} & \frac{\partial f_{\dot{x}}(\mathbf{q},\mathbf{u})}{\partial \dot{x}} & \frac{\partial f_{\dot{x}}(\mathbf{q},\mathbf{u})}{\partial \theta} & \frac{\partial f_{\dot{x}}(\mathbf{q},\mathbf{u})}{\partial \omega} & \frac{\partial f_{\dot{x}}(\mathbf{q},\mathbf{u})}{\partial p} & \frac{\partial f_{\dot{x}}(\mathbf{q},\mathbf{u})}{\partial \dot{p}} \\ \frac{\partial f_\theta(\mathbf{q},\mathbf{u})}{\partial x} & \frac{\partial f_\theta(\mathbf{q},\mathbf{u})}{\partial \dot{x}} & \frac{\partial f_\theta(\mathbf{q},\mathbf{u})}{\partial \theta} & \frac{\partial f_\theta(\mathbf{q},\mathbf{u})}{\partial \omega} & \frac{\partial f_\theta(\mathbf{q},\mathbf{u})}{\partial p} & \frac{\partial f_\theta(\mathbf{q},\mathbf{u})}{\partial \dot{p}} \\ \frac{\partial f_\omega(\mathbf{q},\mathbf{u})}{\partial x} & \frac{\partial f_\omega(\mathbf{q},\mathbf{u})}{\partial \dot{x}} & \frac{\partial f_\omega(\mathbf{q},\mathbf{u})}{\partial \theta} & \frac{\partial f_\omega(\mathbf{q},\mathbf{u})}{\partial \omega} & \frac{\partial f_\omega(\mathbf{q},\mathbf{u})}{\partial p} & \frac{\partial f_\omega(\mathbf{q},\mathbf{u})}{\partial \dot{p}} \\ \frac{\partial f_p(\mathbf{q},\mathbf{u})}{\partial x} & \frac{\partial f_p(\mathbf{q},\mathbf{u})}{\partial \dot{x}} & \frac{\partial f_p(\mathbf{q},\mathbf{u})}{\partial \theta} & \frac{\partial f_p(\mathbf{q},\mathbf{u})}{\partial \omega} & \frac{\partial f_p(\mathbf{q},\mathbf{u})}{\partial p} & \frac{\partial f_p(\mathbf{q},\mathbf{u})}{\partial \dot{p}} \\ \frac{\partial f_{\dot{p}}(\mathbf{q},\mathbf{u})}{\partial x} & \frac{\partial f_{\dot{p}}(\mathbf{q},\mathbf{u})}{\partial \dot{x}} & \frac{\partial f_{\dot{p}}(\mathbf{q},\mathbf{u})}{\partial \theta} & \frac{\partial f_{\dot{p}}(\mathbf{q},\mathbf{u})}{\partial \omega} & \frac{\partial f_{\dot{p}}(\mathbf{q},\mathbf{u})}{\partial p} & \frac{\partial f_{\dot{p}}(\mathbf{q},\mathbf{u})}{\partial \dot{p}} \end{pmatrix} \tag{5.7}$$

$$\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}} = \begin{pmatrix} \frac{\partial f_x(\mathbf{q},\mathbf{u})}{\partial a} & \frac{\partial f_x(\mathbf{q},\mathbf{u})}{\partial \alpha} \\ \frac{\partial f_{\dot{x}}(\mathbf{q},\mathbf{u})}{\partial a} & \frac{\partial f_{\dot{x}}(\mathbf{q},\mathbf{u})}{\partial \alpha} \\ \frac{\partial f_\theta(\mathbf{q},\mathbf{u})}{\partial a} & \frac{\partial f_\theta(\mathbf{q},\mathbf{u})}{\partial \alpha} \\ \frac{\partial f_\omega(\mathbf{q},\mathbf{u})}{\partial a} & \frac{\partial f_\omega(\mathbf{q},\mathbf{u})}{\partial \alpha} \\ \frac{\partial f_p(\mathbf{q},\mathbf{u})}{\partial a} & \frac{\partial f_p(\mathbf{q},\mathbf{u})}{\partial \alpha} \\ \frac{\partial f_{\dot{p}}(\mathbf{q},\mathbf{u})}{\partial a} & \frac{\partial f_{\dot{p}}(\mathbf{q},\mathbf{u})}{\partial \alpha} \end{pmatrix} \tag{5.8}$$

Di fatto quindi le matrici $\left.\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}}\right|_{\mathbf{q}=\mathbf{q}_e,\mathbf{u}=\mathbf{u}_e}$ e $\left.\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}}\right|_{\mathbf{q}=\mathbf{q}_e,\mathbf{u}=\mathbf{u}_e}$ calcolate nel punto di equilibrio, possono essere viste rispettivamente come le matrici A_{lin} e B_{lin} del sistema equivalente linearizzato, che nel discreto corrisponde a

$$\mathbf{q}(k+1) = A_{lin}\mathbf{q}(k) + B_{lin}\mathbf{u}(k) \tag{5.9}$$

che in un intorno piccolo a piacere del punto di equilibrio si comporta come

il sistema non lineare.

5.2 Il calcolo delle Jacobiane $\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}}$ e $\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}}$

Per il calcolo delle Jacobiane per la linearizzazione si parte dal sistema non lineare $\mathbf{q}_{k+1} = f(\mathbf{q}_k, \mathbf{u}_k)$. Per quanto riguarda la parte del piatto, abbiamo le equazioni già nella forma desiderata, già definite nelle formule (2.9), (2.10), (2.64) e (2.65) del capitolo sulla dinamica Newtoniana:

$$x(k+1) = x(k) + \dot{x}(k)Ts + a(k)\frac{Ts^2}{2} \tag{5.10}$$

$$\dot{x}(k+1) = \dot{x}(k) + a(k)Ts \tag{5.11}$$

$$\theta(k+1) = \theta(k) + G^{-1}\omega(k)Ts + G^{-1}\alpha(k)\frac{Ts^2}{2} \tag{5.12}$$

$$\omega(k+1) = \omega(k) + G^{-1}\alpha(k)Ts \tag{5.13}$$

che rappresentano i primi quattro elementi della funzione vettoriale $f(\mathbf{q}_k, \mathbf{u}_k)$.

Seguendo il paragrafo 2.3.2, sulle leggi di moto della pallina verranno trovati gli ultimi due, quelli che descrivono le leggi di moto della pallina, che completeranno la $f(\mathbf{q}_k, \mathbf{u}_k)$: per il loro calcolo sono stati effettuati dei cambi di coordinate, perciò le formule sono più articolate rispetto a quelle relative alla piattaforma; tratteremo separatamente la parte relativa alla posizione e quella relativa alla velocità della pallina.

5.2.1 Il calcolo di $f_p(\mathbf{q}_k, \mathbf{u}_k)$, relativa alla posizione

La prima operazione da effettuare è il passaggio alle coordinate della piattaforma ${}^pS$ attraverso le note formule (2.69) e (2.70): per la posizione si ha

$${}^pp = {}^pR_0(\theta)({}^0p - {}^0x) \tag{5.14}$$

e per la velocità:

$${}^p\dot{p} = {}^pR_0(\theta)\left({}^0\dot{p} - {}^0\dot{x} - {}^0\omega \times {}^0R_p(\theta)I_{03}{}^pp\right) \tag{5.15}$$

essendo ${}^0R_p I_{03} {}^p p = {}^0 r_p$ il braccio della rotazione, già discusso nel capitolo della dinamica Newtoniana; ricordiamo che la matrice I_{03} è la matrice di proiezione definita come

$$I_{03} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Sostituendo la (5.14) nella (5.15) otteniamo

$${}^p\dot{p} = {}^pR_0(\theta)({}^0\dot{p} - {}^0\dot{x} - {}^0\omega \times {}^0R_p(\theta) I_{03} {}^pR_0(\theta)({}^0p - {}^0x)) \tag{5.16}$$

A questo punto, sempre trascurando il raggio della pallina, sostituiamo le espressioni trovate nella legge di moto proiettate sul piano

$${}^p p(k+1) = I_{03}\left(p(k) + {}^p\dot{p}(k)Ts + {}^pa(k)\frac{Ts^2}{2}\right) \tag{5.17}$$

ottenendo

$$\begin{aligned} {}^p p_{k+1} = I_{03}\Big[& {}^pR_0(\theta_k)\left({}^0p_k - {}^0x_k\right) + {}^pR_0(\theta_k)\left({}^0\dot{p_k} - {}^0\dot{x_k} + \right. \\ & \left. - {}^0\omega_{p,k} \times {}^0R_p(\theta_k) I_{03} {}^p p_k\right) Ts + {}^pR_0(\theta_k)\left({}^0a_k + {}^0g\right)\frac{Ts^2}{2}\Big] \end{aligned} \tag{5.18}$$

dove notiamo già da subito che si può raccogliere la matrice ${}^pR_0(\theta)$. Fatto questo non resta che tornare alle coordinate di terra con la formula

$${}^0p(k+1) = {}^0x + {}^0R_p {}^p p(k+1)$$

ottenendo, in conclusione, la formula cercata per la posizione con tutte le variabili nelle coordinate di terra

$$\begin{aligned} {}^0p_{k+1} = {}^0x_k + {}^0R_p(\theta_k) I_{03} {}^pR_0(\theta_k)\Big[& \left({}^0p_k - {}^0x_k\right) + \\ & + \left({}^0\dot{p_k} - {}^0\dot{x_k} - {}^0\omega_{p,k} \times {}^0R_p(\theta_k) I_{03} {}^p p_k\right) Ts + \left({}^0a_k + {}^0g\right)\frac{Ts^2}{2}\Big] \end{aligned} \tag{5.19}$$

che considerata in funzione delle variabili di stato, equivale a

$$p_{k+1} = x_k + R(\theta_k) I_{03} R^T(\theta_k) \Big[p_k - x_k + \dot{p}_k Ts - \dot{x_k} Ts - [\omega_k \times] R(\theta_k) I_{03} R^T(\theta_k) p_k Ts +$$

$$+ [\omega_k \times] R(\theta_k) I_{03} R^T(\theta_k) x_k Ts + a_k \frac{Ts^2}{2} + g \frac{Ts^2}{2} \Big] \tag{5.20}$$

che è la formula finale cercata.

5.2.2 Il calcolo di $f_{\dot{p}}(\mathbf{q}_k, \mathbf{u}_k)$, relativa alla velocità

Per $f_{\dot{p}}(\mathbf{q}_k, \mathbf{u}_k)$ si devono fare dei passaggi simili a quelli per la posizione: richiamiamo anche questa volta le equazioni (5.14) e (5.15)

$$^p p = {}^p R_0(\theta) \left({}^0 p - {}^0 x \right)$$

$$^p \dot{p} = {}^p R_0(\theta) \Big({}^0 \dot{p} - {}^0 \dot{x} - {}^0 \omega_1 \times {}^0 R_p(\theta) I_{03} {}^p p \Big)$$

che verranno inserite nell' equazione di moto

$$^p \dot{p}(k+1) = I_{03} \left[{}^p \dot{p}(k) + {}^p a(k) Ts \right] \tag{5.21}$$

Otterremo, raccogliendo ${}^p R_0(\theta)$:

$$^p \dot{p}_{k+1} = I_{03} {}^p R_0(\theta_k) \left[\Big({}^0 \dot{p}_k - {}^0 \dot{x_k} - {}^0 \omega_{p,k} \times {}^0 R_p(\theta_k) I_{03} {}^p p_k \Big) + \left({}^0 a_k + {}^0 g \right) Ts \right] \tag{5.22}$$

che per esteso è

$$^p \dot{p}_{k+1} = I_{03} {}^p R_0(\theta_k) \Big[{}^0 \dot{p_k} - {}^0 \dot{x_k} - {}^0 \omega_{p,k} \times {}^0 R_p(\theta_k) I_{03} {}^p R_0(\theta_k) {}^0 p_k +$$

$$+ {}^0 \omega_{p,k} \times {}^0 R_p(\theta_k) I_{03} {}^p R_0(\theta_k) {}^0 x_k) + {}^0 a_k Ts + {}^0 g Ts \Big] \tag{5.23}$$

Si riporta il tutto nelle coordinate di terra con la formula

$$^0 \dot{p}(k+1) = {}^0 \dot{x} + {}^0 R_p(\theta) {}^p \dot{p}(k+1) + {}^0 \omega_1 \times {}^0 r_p$$

ricalcolando ${}^0r_p$, che è diverso da quello calcolato in precedenza. Sviluppando otteniamo:

$$
\begin{aligned}
{}^0\dot{p}_{k+1} = {}^0\dot{x}_k + {}^0R_p(\theta_k) I_{03}{}^p R_0(\theta_k) \Big[{}^0\dot{p}_k - {}^0\dot{x}_k - \left[{}^0\omega_{p,k}\times\right] {}^0R_p(\theta_k) I_{03}{}^p R_0(\theta_k) {}^0p_k + \\
+ \left[{}^0\omega_{p,k}\times\right] {}^0R_p(\theta_k) I_{03}{}^p R_0(\theta_k) {}^0x_k + {}^0a_k Ts + {}^0gTs \Big] + \\
+ \left[{}^0\omega_{p,k}\times\right] {}^0R_p(\theta_k) I_{03}^2{}^p R_0(\theta_k) \Big[\left({}^0p_k - {}^0x_k\right) + \Big({}^0\dot{p}_k - {}^0\dot{x}_k - {}^0\omega_{p,k}\times {}^0R_p(\theta_k) I_{03}{}^p p_k + \\
+ {}^0\omega_{p,k}\times {}^0R_p(\theta_k) I_{03}{}^p x_k \Big) Ts + \left({}^0a_k + {}^0g\right) \frac{Ts^2}{2} \Big]
\end{aligned}
$$

che in funzione delle variabili di stato equivale a

$$
\begin{aligned}
\dot{p}_{k+1} = \dot{x}_k + R(\theta_k) I_{03} R^T(\theta_k) \Big[\dot{p}_k - \dot{x}_k - \left[\omega_k\times\right] R(\theta_k) I_{03} R^T(\theta_k) p_k + \\
\left[\omega_k\times\right] R(\theta_k) I_{03} R^T(\theta_k) x_k + a_k Ts + gTs \Big] + \\
+ \left[\omega_k\times\right] R(\theta_k) I_{03}^2 R^T(\theta_k) \Big[p_k - x_k + \dot{p}_k Ts - \dot{x}_k Ts - \left[\omega_k\times\right] R(\theta_k) I_{03} R^T(\theta_k) p_k Ts + \\
+ \left[\omega_k\times\right] R(\theta_k) I_{03} R^T(\theta_k) x_k Ts + a_k \frac{Ts^2}{2} + g\frac{Ts^2}{2} \Big]
\end{aligned}
\tag{5.24}
$$

5.2.3 Il proiettore $R(\theta) I_{03} R^T(\theta)$

Nell'effettuare i calcoli su p e $\dot{p}$ si nota che è ricorrente il termine premoltiplicativo $R(\theta) I_{03} R^T(\theta)$ nelle espressioni finali nelle coordinate terrestri: analizzando attentamente questo termine possiamo trovarne un'interpretazione molto interessante.

Partiamo dala matrice I_{03}, già vista nel capitolo della dinamica Newtoniana, che ricordiamo essere definita come

$$I_{03} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

La moltiplicazione di tale matrice per un qualsiasi vettore, produce l' annullamento della sua terza componente lasciando inalterate le prime due, effettuando quindi una proiezione sul piano orizzontale. Per questo motivo è usata nelle coordinate ${}^{p}S$ per annullare la terza componente di velocità e accelerazione per effettuare i calcoli in due dimensioni. La matrice I_{03} può essere interpretata, in queste coordinate, come un *proiettore* sul sottospazio della piattaforma.

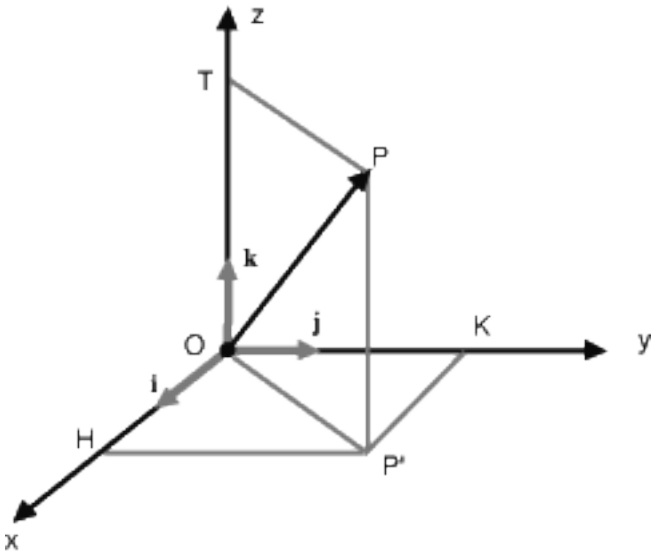

Figure 5.2: Proiezione di un vettore sul piano orizzontale

Calcolando il nucleo notiamo che:

$$\mathcal{N}(I_{03}) = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

che dimostra il fatto che un vettore verticale ha proiezione nulla sul piano orizzontale, come facile intuire.

Un'altra proprietà molto importante di questa matrice è la sua *idempotenza*:

Definizione Una matrice A si dice *idempotente* se $A^n = A$ con $\forall n \geq 1\,,\, n \in \mathbb{N}$

Intuitivamente, se ho già proiettato un vettore su un piano, proiettarlo un' altra volta non ha alcun effetto, perchè il vettore si trova già sul piano.

Nella nostra applicazione la matrice I_{03} si moltiplica sempre per un vettore nelle coordinate ${}^pS$, proiettandolo sulla piattaforma. Se vogliamo proiettare un vettore ${}^0p$ espresso nelle coordinate ${}^0S$, come è accaduto nel calcolo della funzione $f(\mathbf{q}, \mathbf{u})$ dobbiamo effettuare questa sequenza di operazioni:

- trasformare il vettore nelle coordinate ${}^pS$, ottenendo ${}^pR_0\,{}^0p$
- applicare al vettore appena trovato il proiettore I_{03}, ottenendo $I_{03}{}^pR_0\,{}^0p$
- riportare il tutto nelle coordinate ${}^0S$ di terra, ottenendo ${}^0R_pI_{03}{}^pR_0\,{}^0p$

Possiamo notare che compare nella formula il termine ${}^0R_pI_{03}{}^pR_0$ che è proprio quello che stiamo trattando: si può interpretare come il proiettore I_{03} espresso nelle coordinate di terra. Infatti per applicare un cambio di coordinate T ad una generica matrice A bisogna moltiplicarla a sinistra per T e a destra per T^{-1}, in formule TAT^{-1}, che corrisponde a ciò che è stato fatto alla matrice I_{03}.

Alla luce di questa osservazione possiamo introdurre una nuova notazione: chiameremo ${}^pI_{03}$ la matrice $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$, che è il proiettore nelle coordinate ${}^pS$, e ${}^0I_{03}$ la matrice ${}^0R_p\,{}^pI_{03}{}^pR_0$ che invece è il proiettore trasformato nelle coordinate ${}^0S$ di terra. Il nucleo di ${}^0I_{03}$, di conseguenza, risulta pari al versore ortogonale alla piattaforma:

$$\mathcal{N}\left({}^0R_p\,{}^pI_{03}{}^pR_0\right) = \hat{n} \tag{5.25}$$

Si può dimostrare con semplici passaggi che anche questa matrice gode della proprietà di idempotenza; considerando infatti il suo elevamento al quadrato abbiamo che:

$$\left({}^0R_p\,{}^pI_{03}{}^pR_0\right)^2 = {}^0R_p\,{}^pI_{03}{}^pR_0\,{}^0R_p\,{}^pI_{03}{}^pR_0 = {}^0R_p\,{}^pI_{03}^2{}^pR_0 = {}^0R_p\,{}^pI_{03}{}^pR_0 \tag{5.26}$$

motivo per cui abbiamo ottenuto notevoli semplificazioni nelle formule.

Consideriamo adesso le funzioni $f_p(\mathbf{q}_k, \mathbf{u}_k)$ e $f_{\dot{p}}(\mathbf{q}_k, \mathbf{u}_k)$ che dovranno essere derivate rispetto a $\mathbf{q}$ e $\mathbf{u}$ per calcolo dello Jacobiano. Come si intuisce ancora prima di effettuare i calcoli, quando si effettuerà la derivata $\frac{\partial}{\partial\theta}$, si deve considerare che le matrici di rotazione ${}^0R_p$ e la sua inversa ${}^pR_0$ sono funzione degli angoli di Roll Pitch e Yaw della piattaforma, e cioè di θ. Comparirà perciò un termine $\frac{\partial R(\theta) I_{03} R^T(\theta)}{\partial\theta}$ che è necessario calcolare. Poichè la derivata di una matrice rispetto ad un vettore non è così semplice da effettuare, si sfrutta il fatto che, essendo $R(\theta)I_{03}R^T(\theta)$ un endomorfismo, non si troverà mai da solo, ma sempre moltiplicato per un vettore, quello a cui va applicato. Definendo un generico vettore $b = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$, invece di derivare la matrice di proiezione, deriviamo rispetto a θ il vettore ottenuto dal prodotto $R(\theta)I_{03}R^T(\theta)b$, ottenendo una matrice funzione delle componenti del generico b, che battezziamo $M(b)$. Effettuiamo ora la differenziazione:

$$M(b) = \frac{\partial R(\theta) I_{03} R^T(\theta) b}{\partial\theta} =$$

$$= \frac{\partial}{\partial\theta}\left[\begin{pmatrix} c\vartheta_z c\vartheta_y & c\vartheta_z s\vartheta_y s\vartheta_x - s\vartheta_z c\vartheta_x & c\vartheta_z s\vartheta_y c\vartheta_x + s\vartheta_z s\vartheta_x \\ s\vartheta_z c\vartheta_y & s\vartheta_z s\vartheta_y s\vartheta_x + c\vartheta_z c\vartheta_x & s\vartheta_z s\vartheta_y c\vartheta_x - c\vartheta_z s\vartheta_x \\ -s\vartheta_y & c\vartheta_y s\vartheta_x & c\vartheta_y c\vartheta_x \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}\cdot\right.$$

$$\left.\cdot\begin{pmatrix} c\vartheta_z c\vartheta_y & c\vartheta_z s\vartheta_y s\vartheta_x - s\vartheta_z c\vartheta_x & c\vartheta_z s\vartheta_y c\vartheta_x + s\vartheta_z s\vartheta_x \\ s\vartheta_z c\vartheta_y & s\vartheta_z s\vartheta_y s\vartheta_x + c\vartheta_z c\vartheta_x & s\vartheta_z s\vartheta_y c\vartheta_x - c\vartheta_z s\vartheta_x \\ -s\vartheta_y & c\vartheta_y s\vartheta_x & c\vartheta_y c\vartheta_x \end{pmatrix}\begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}\right]$$

Come possiamo vedere il calcoli sono molto laboriosi, quasi impossibili da svolgere a mano. Per questo motivo è stata scritta una patch MATLAB apposita, chiamata derivtheta che utilizzando il calcolo simbolico effettua i prodotti matriciali e le derivare parziali. Il risultato restituito dalla funzione, sorprendentemente, è molto semplice, e pari a

$$M(b) = \begin{pmatrix} 0 & -b_3 & 0 \\ b_3 & 0 & 0 \\ b_2 & -b_1 & 0 \end{pmatrix}$$

A questo punto siamo pronti per calcolare riga per riga le matrici Jacobiane $\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}}$ e $\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}}$

5.2.4 La Jacobiana $\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}}$

I calcoli si effettuano per comodità a blocchi di tre righe, cioè variabile per variabile; la Jacobiana risultante si troverà concatenando i blocchi in questo modo:

$$\left.\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}}\right|_{q=0,u=0} = \left.\begin{pmatrix} \frac{\partial f_x(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}} \\ \frac{\partial f_{\dot{x}}(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}} \\ \frac{\partial f_\theta(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}} \\ \frac{\partial f_\omega(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}} \\ \frac{\partial f_p(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}} \\ \frac{\partial f_p(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}} \end{pmatrix}\right|_{q=0,u=0}$$

Dovendo derivare rispetto a sei variabili tridimensionali otterremo una matrice quadrata 18×18, coerentemente con la dimensione dello stato.

5.2.4.1 Il blocco $\frac{\partial f_x(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}}$

I calcoli sulle componenti di f che descrivono la piattaforma sono molto semplici: richiamiamo rapidamente la formula (5.10):

$$x(k+1) = x(k) + \dot{x}(k)Ts + a(k)\frac{Ts^2}{2}$$

senza troppe difficoltà otteniamo

$$\left.\frac{\partial f_x(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}}\right|_{q=0,u=0} = \begin{pmatrix} I & Ts & O & O & O & O \end{pmatrix} \tag{5.27}$$

Dove, d' ora in poi, per brevità useremo Ts per indicare $Ts{\cdot}I$.

5.2.4.2 Il blocco $\frac{\partial f_{\dot{x}}(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}}$

Anche qui non si incontrano grandi difficoltà. La derivata della formula (5.11)

$$\dot{x}(k+1) = \dot{x}(k) + a(k)Ts$$

è pari a

$$\left.\frac{\partial f_{\dot{x}}(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}}\right|_{q=0,u=0} = \begin{pmatrix} O & I & O & O & O & O \end{pmatrix}$$

5.2.4.3 Il blocco $\frac{\partial f_{\theta}(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}}$

Richiamiamo la formula (5.12)

$$\theta(k+1) = \theta(k) + G^{-1}(\theta)\omega(k)Ts + G^{-1}(\theta)\alpha(k)\frac{Ts^2}{2}$$

Qui abbiamo la matrice G^{-1}che compare nella derivata rispetto a ω. Considerando però che dobbiamo valutare il tutto per $\mathbf{q} = 0$ e che $G^{-1}(0) = I$ otteniamo come riga dello Jacobiano

$$\left.\frac{\partial f_{\theta}(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}}\right|_{q=0,u=0} = \begin{pmatrix} O & O & I & Ts & O & O \end{pmatrix}$$

5.2.4.4 Il blocco $\frac{\partial f_{\omega}(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}}$

Mostriamo per comodità la formula (5.13)

$$\omega(k+1) = \omega(k) + G^{-1}\alpha(k)Ts$$

ed effettuiamone la derivata rispetto a $\mathbf{q}$; la risposta è immediata:

$$\left.\frac{\partial f_{\omega}(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}}\right|_{q=0,u=0} = \begin{pmatrix} O & O & O & I & O & O \end{pmatrix}$$

5.2.4.5 Il blocco $\frac{\partial f_{p}(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}}$

Qui i calcoli si fanno decisamente più laboriosi: Richiamiamo la formula (5.20)

$$p_{k+1} = x_k + R(\theta_k) I_{03} R^T(\theta_k) \Big[p_k - x_k + \dot{p}_k Ts - \dot{x_k} Ts - [\omega_k \times] R(\theta_k) I_{03} R^T(\theta_k) p_k Ts +$$

$$+ [\omega_k \times] R(\theta_k) I_{03} R^T(\theta_k) x_k Ts + a_k \frac{Ts^2}{2} + g \frac{Ts^2}{2} \Big]$$

Deriviamo rispetto a ogni singola variabile di $\mathbf{q}$:

$$\left. \frac{\partial f_p(\mathbf{q}, \mathbf{u})}{\partial x} \right|_{\mathbf{q}=0, \mathbf{u}=0} = I - R(\theta) I_{03} R^T(\theta) + [\omega \times] R(\theta) I_{03} R^T(\theta) Ts \Big|_{\mathbf{q}=0, \mathbf{u}=0} = I - I_{03}$$

$$\left. \frac{\partial f_p(\mathbf{q}, \mathbf{u})}{\partial \dot{x}} \right|_{\mathbf{q}=0, \mathbf{u}=0} = -R(\theta) I_{03} R^T(\theta) Ts \Big|_{\mathbf{q}=0, \mathbf{u}=0} = -Ts I_{03}$$

Ancora prima di effettuare i calcoli osserviamo l' espressione: il proiettore $R(\theta_k) I_{03} R^T(\theta_k)$ premoltiplica molti vettore, che alla fine dovranno essere valutati in $\mathbf{q} = 0, \mathbf{u} = 0$. Essendo $M(0) = O$, abbiamo tutte le derivate nulle, tranne quella di $R(\theta_k) I_{03} R^T(\theta_k) g$

$$\left. \frac{\partial f_p(\mathbf{q}, \mathbf{u})}{\partial \theta} \right|_{\mathbf{q}=0, \mathbf{u}=0} = \frac{Ts^2}{2} M(g)$$

Anche qui evitiamo di fare i calcoli perchè troviamo il fattore $R(\theta_k) I_{03} R^T(\theta_k) [\omega_k \times] R(\theta_k) I_{03} R^T(\theta_k)$ moltiplicato per tutti vettori che andranno valutati in 0. La derivata parziale perciò è nulla

$$\left. \frac{\partial f_p(\mathbf{q}, \mathbf{u})}{\partial \omega} \right|_{\mathbf{q}=0, \mathbf{u}=0} = O$$

$$\left. \frac{\partial f_p(\mathbf{q}, \mathbf{u})}{\partial p} \right|_{\mathbf{q}=0, \mathbf{u}=0} = R(\theta) I_{03} R^T(\theta) - [\omega \times] R(\theta) I_{03} R^T(\theta) Ts \Big|_{\mathbf{q}=0, \mathbf{u}=0} = I_{03}$$

$$\frac{\partial f_p(\mathbf{q},\mathbf{u})}{\partial \dot{p}}\bigg|_{\mathbf{q}=0,\mathbf{u}=0} = R(\theta)I_{03}R^T(\theta)Ts\bigg|_{\mathbf{q}=0,\mathbf{u}=0} = TsI_{03}$$

5.2.4.6 Il blocco $\frac{\partial f_{\dot{p}}(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}}$

Anche per questo blocco i conti sono abbastanza laboriosi: richiamiamo la formula (5.24)

$$\dot{p}_{k+1} = \dot{x}_k + R(\theta_k)I_{03}R^T(\theta_k)\Big[\dot{p}_k - \dot{x}_k - [\omega_k\times]\,R(\theta_k)I_{03}R^T(\theta_k)p_k +$$

$$[\omega_k\times]\,R(\theta_k)I_{03}R^T(\theta_k)x_k + a_kTs + gTs\Big] +$$

$$+ [\omega_k\times]\,R(\theta_k)I_{03}^2R^T(\theta_k)\Big[p_k - x_k + \dot{p_k}Ts - x_k\dot{T}s - [\omega_k\times]\,R(\theta_k)I_{03}R^T(\theta_k)p_kTs +$$

$$+ [\omega_k\times]\,R(\theta_k)I_{03}R^T(\theta_k)x_kTs + a_k\frac{Ts^2}{2} + g\frac{Ts^2}{2}\Big]$$

e deriviamo rispetto a ogni singola variabile di $\mathbf{q}$:

$$\frac{\partial f_{\dot{p}}(\mathbf{q},\mathbf{u})}{\partial x}\bigg|_{\mathbf{q}=0,\mathbf{u}=0} = R(\theta)I_{03}R^T(\theta)\,[\omega\times]\,R(\theta)I_{03}R^T(\theta) - [\omega\times]\,R(\theta)I_{03}^2R^T(\theta)\bigg|_{\mathbf{q}=0,\mathbf{u}=0} = O$$

Eliminando direttamente i termini nulli si ottiene

$$\frac{\partial f_{\dot{p}}(\mathbf{q},\mathbf{u})}{\partial \dot{x}}\bigg|_{\mathbf{q}=0,\mathbf{u}=0} = I - R(\theta)I_{03}R^T(\theta)\bigg|_{\mathbf{q}=0,\mathbf{u}=0} = I - I_{03}$$

Anche qui come nella $\frac{\partial f_p(\mathbf{q},\mathbf{u})}{\partial \dot{x}}$ compare la matrice $M(g)$:

$$\frac{\partial f_{\dot{p}}(\mathbf{q},\mathbf{u})}{\partial \theta}\bigg|_{\mathbf{q}=0,\mathbf{u}=0} = TsM(g) + \frac{Ts^2}{2}\left[\omega\times\right]M(g)\bigg|_{\mathbf{q}=0,\mathbf{u}=0} = TsM(g)$$

In questo caso per arrivare al risulatato verrà sfruttata la proprietà associativa del prodotto matriciale

$$\frac{\partial f_{\dot{p}}(\mathbf{q},\mathbf{u})}{\partial \omega}\bigg|_{\mathbf{q}=0,\mathbf{u}=0} = \frac{Ts^2}{2}\left[\omega\times\right]R(\theta)I_{03}R^T(\theta)g\bigg|_{\mathbf{q}=0,\mathbf{u}=0} = \frac{Ts^2}{2}\left[\omega\times\right]I_{03}g =$$

$$= \frac{Ts^2}{2}\left[\omega\times\right](I_{03}g) = \frac{Ts^2}{2}\left[\omega\times\right]\begin{pmatrix}0\\0\\0\end{pmatrix} = O$$

e il fatto che $g \in \mathcal{N}(I_{03})$, cioè il vettore g di accelerazione gravitazionale, definito come $g = \begin{pmatrix}0\\0\\-9.8\end{pmatrix}$, appartiene al sottospazio annullante del proiettore ${}^pI_{03}$.

I termini, essendo tutti premoltiplicati per $[\omega_k\times]$ che dovrà essere valutata in 0, si annullano tutti:

$$\frac{\partial f_{\dot{p}}(\mathbf{q},\mathbf{u})}{\partial p}\bigg|_{\mathbf{q}=0,\mathbf{u}=0} = O$$

Evitando di considerare fin da subito i termini nulli si ha

$$\frac{\partial f_{\dot{p}}(\mathbf{q},\mathbf{u})}{\partial \dot{p}}\bigg|_{\mathbf{q}=0,\mathbf{u}=0} = R(\theta)I_{03}R^T(\theta)\bigg|_{\mathbf{q}=0,\mathbf{u}=0} = I_{03}$$

Terminati i calcoli abbiamo ottenuto la matrice Jacobiana $\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}}$ che scritta in forma compatta è pari a:

$$\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}} = \begin{pmatrix} I & Ts & O & O & O & O \\ O & I & O & O & O & O \\ O & O & I & Ts & O & O \\ O & O & O & I & O & O \\ I - I_{03} & -TsI_{03} & \frac{Ts^2}{2}M(g) & O & I_{03} & TsI_{03} \\ O & I - I_{03} & TsM(g) & O & O & I_{03} \end{pmatrix} \tag{5.28}$$

in forma estesa di matrice quadrata di dimensione 18:

$$\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}} = \begin{pmatrix} \begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} & \begin{bmatrix}Ts&0&0\\0&Ts&0\\0&0&Ts\end{bmatrix} & \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} & \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} & \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} & \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} \\ \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} & \begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} & \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} & \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} & \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} & \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} \\ \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} & \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} & \begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} & \begin{bmatrix}Ts&0&0\\0&Ts&0\\0&0&Ts\end{bmatrix} & \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} & \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} \\ \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} & \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} & \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} & \begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} & \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} & \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} \\ \begin{bmatrix}0&0&0\\0&0&0\\0&0&1\end{bmatrix} & \begin{bmatrix}-Ts&0&0\\0&-Ts&0\\0&0&0\end{bmatrix} & \begin{bmatrix}0&9.8\frac{Ts^2}{2}&0\\-9.8\frac{Ts^2}{2}&0&0\\0&0&0\end{bmatrix} & \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} & \begin{bmatrix}1&0&0\\0&1&0\\0&0&0\end{bmatrix} & \begin{bmatrix}Ts&0&0\\0&Ts&0\\0&0&0\end{bmatrix} \\ \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} & \begin{bmatrix}0&0&0\\0&0&0\\0&0&1\end{bmatrix} & \begin{bmatrix}0&9.8Ts&0\\-9.8Ts&0&0\\0&0&0\end{bmatrix} & \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} & \begin{bmatrix}0&0&0\\0&0&0\\0&0&0\end{bmatrix} & \begin{bmatrix}1&0&0\\0&1&0\\0&0&0\end{bmatrix} \end{pmatrix}$$

Passiamo adesso al calcolo dell' altra matrice Jacobiana $\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}}$, quella relativa all' ingresso

5.2.5 La Jacobiana $\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}}$

Procediamo in modo analogo: effettuiamo tutte le derivate parziali a blocchi di tre righe e poi concateniamo tutti i blocchi riga per ottenere la matrice richiesta. Dovendo derivare rispetto alle due variabili tridimensionali a e α otterremo una Jacobiana di dimensione 18×6: ciò è coerente con la dimensione dello stato, 18, e la dimensione dell' ingresso, 6.

5.2.5.1 Il blocco $\frac{\partial f_x(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}}$

I calcoli sulle componenti di f che descrivono la piattaforma sono molto semplici: richiamiamo rapidamente la formula (5.10):

$$x(k+1) = x(k) + \dot{x}(k)Ts + a(k)\frac{Ts^2}{2}$$

senza troppe difficoltà otteniamo

$$\left.\frac{\partial f_x(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}}\right|_{q=0,u=0} = \begin{pmatrix} \frac{Ts^2}{2} & O \end{pmatrix} \tag{5.29}$$

5.2.5.2 Il blocco $\frac{\partial f_{\dot{x}}(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}}$

Anche qui non si incontrano grandi difficoltà. La derivata della formula (5.11)

$$\dot{x}(k+1) = \dot{x}(k) + a(k)Ts$$

è pari a

$$\left.\frac{\partial f_{\dot{x}}(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}}\right|_{q=0,u=0} = \begin{pmatrix} Ts & O \end{pmatrix} \tag{5.30}$$

5.2.5.3 Il blocco $\frac{\partial f_\theta(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}}$

Richiamiamo la formula (5.12)

$$\theta(k+1) = \theta(k) + G^{-1}(\theta)\omega(k)Ts + G^{-1}(\theta)\alpha(k)\frac{Ts^2}{2}$$

Ricorando che $G^{-1}(0) = I$

$$\left.\frac{\partial f_\theta(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}}\right|_{q=0,u=0} = \begin{pmatrix} O & \frac{Ts^2}{2} \end{pmatrix}$$

5.2.5.4 Il blocco $\frac{\partial f_\omega(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}}$

Mostriamo per comodità la formula (5.13)

$$\omega(k+1) = \omega(k) + G^{-1}\alpha(k)Ts$$

valutando il tutto in zero otteniamo:

$$\left.\frac{\partial f_\omega(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}}\right|_{q=0,u=0} = \begin{pmatrix} O & Ts \end{pmatrix}$$

5.2.5.5 Il blocco $\frac{\partial f_p(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}}$

Richiamiamo la formula (5.20), dove compare solamente l' accelerazione lineare a

$$p_{k+1} = x_k + R(\theta_k)I_{03}R^T(\theta_k)\Big[p_k - x_k + \dot{p}_k Ts - \dot{x_k}Ts - [\omega_k\times]R(\theta_k)I_{03}R^T(\theta_k)p_k Ts +$$

$$+[\omega_k\times]R(\theta_k)I_{03}R^T(\theta_k)x_k Ts + a_k\frac{Ts^2}{2} + g\frac{Ts^2}{2}\Big]$$

La derivata parziale risulta

$$\left.\frac{\partial f_p(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}}\right|_{q=0,u=0} = R(\theta_k)I_{03}R^T(\theta_k)\begin{pmatrix} \frac{Ts^2}{2} & O \end{pmatrix}\Big|_{q=0,u=0} = \begin{pmatrix} \frac{Ts^2}{2}I_{03} & O \end{pmatrix}$$

5.2.5.6 Il blocco $\frac{\partial f_{\dot{p}}(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}}$

Anche qui richiamiamo la formula (5.24)

$$\dot{p}_{k+1} = \dot{x}_k + R(\theta_k)I_{03}R^T(\theta_k)\Big[\dot{p}_k - \dot{x}_k - [\omega_k\times]\,R(\theta_k)I_{03}R^T(\theta_k)p_k +$$

$$[\omega_k\times]\,R(\theta_k)I_{03}R^T(\theta_k)x_k + a_k Ts + gTs\Big] +$$

$$+\,[\omega_k\times]\,R(\theta_k)I_{03}^2R^T(\theta_k)\Big[p_k - x_k + \dot{p_k}Ts - x_k\dot{T}s - [\omega_k\times]\,R(\theta_k)I_{03}R^T(\theta_k)p_k Ts +$$

$$+ \left[\omega_k \times\right] R(\theta_k) I_{03} R^T(\theta_k) x_k Ts + a_k \frac{Ts^2}{2} + g \frac{Ts^2}{2}\Bigg]$$

ed effettuiamo le derivate sempre tralasciando da subito i termini che si annulleranno:

$$\frac{\partial f_{\dot{p}}(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}}\bigg|_{q=0,u=0} = R(\theta_k) I_{03} R^T(\theta_k) \begin{pmatrix} Ts & O \end{pmatrix}\Big|_{q=0,u=0} = \begin{pmatrix} Ts I_{03} & O \end{pmatrix}$$

La matrice Jacobiana ottenuta è pari a:

$$\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}} = \begin{pmatrix} \frac{Ts^2}{2} & O \\ Ts & O \\ O & \frac{Ts^2}{2} \\ O & Ts \\ I_{03}\frac{Ts^2}{2} & O \\ I_{03} Ts & O \end{pmatrix} \tag{5.31}$$

in forma estesa:

$$\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}} = \begin{pmatrix} \begin{bmatrix} \frac{Ts^2}{2} & O & O \\ O & \frac{Ts^2}{2} & O \\ O & O & \frac{Ts^2}{2} \end{bmatrix} & \begin{bmatrix} O & O & O \\ O & O & O \\ O & O & O \end{bmatrix} \\ \begin{bmatrix} Ts & O & O \\ O & Ts & O \\ O & O & Ts \end{bmatrix} & \begin{bmatrix} O & O & O \\ O & O & O \\ O & O & O \end{bmatrix} \\ \begin{bmatrix} O & O & O \\ O & O & O \\ O & O & O \end{bmatrix} & \begin{bmatrix} \frac{Ts^2}{2} & O & O \\ O & \frac{Ts^2}{2} & O \\ O & O & \frac{Ts^2}{2} \end{bmatrix} \\ \begin{bmatrix} O & O & O \\ O & O & O \\ O & O & O \end{bmatrix} & \begin{bmatrix} Ts & O & O \\ O & Ts & O \\ O & O & Ts \end{bmatrix} \\ \begin{bmatrix} \frac{Ts^2}{2} & O & O \\ O & \frac{Ts^2}{2} & O \\ O & O & O \end{bmatrix} & \begin{bmatrix} O & O & O \\ O & O & O \\ O & O & O \end{bmatrix} \\ \begin{bmatrix} Ts & O & O \\ O & Ts & O \\ O & O & O \end{bmatrix} & \begin{bmatrix} O & O & O \\ O & O & O \\ O & O & O \end{bmatrix} \end{pmatrix} \tag{5.32}$$

5.3 Due modelli: il modello non lineare, e il modello linearizzato nelle sue due realizzazioni di simulazione e di controllo

Ottenute le Jacobiane $\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}}$ e $\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}}$ siamo finalmente pronti per scrivere il sistema linearizzato: dedichiamo una sezione apposita per mostrare le due realizzazioni di tale modello.

Già in precedenza è stata accennata la necessità, per avere una forma più comoda per il controllo, di considerare come stato della pallina il suo errore di posizione rispetto al centro del piatto, invece che la sua posizione nelle coordinate terrestri: si fa quindi una distinzione tra il vettore di stato $\mathbf{q}$ le cui ultime due componenti sono la posizione e la velocità assoluta della pallina $\begin{pmatrix} p \\ \dot{p} \end{pmatrix}$, e $\tilde{\mathbf{q}}$ che ha per ultime componenti l' errore di posizione e di velocità rispetto alla piattaforma $\begin{pmatrix} p-x \\ \dot{p}-\dot{x} \end{pmatrix}$. Chiameremo la realizzazione con $\mathbf{q}$ realizzazione di simulazione, nelle coordinate $\mathbf{Q}$, e la realizzazione con $\tilde{\mathbf{q}}$ realizzazione di controllo, nelle coordinate $\tilde{\mathbf{Q}}$. La prima si usa per simulare il sistema fisico linearizzato, nel quale è importante conoscere la posizione e la velocità assoluta della pallina affinchè venga rappresentata correttamente nell' animazione, mentre la seconda si usa per il controllo LQR, nel quale l' importante è conoscere l' errore rispetto al punto di equilibrio. Ricordiamo inoltre che è stata definita nella (5.3) la matrice T che effettua la trasformazione $T: \ \mathbf{Q} \to \tilde{\mathbf{Q}}$ da una realizzazione all' altra.

Il punto di equilibrio che si vuole far raggiungere al sistema di controllo è definito nelle coordinate $\tilde{\mathbf{Q}}$ come vettore nullo. Nel calcolo dello Jacobiano, è stato posto come punto di equilibrio il vettore nullo, ma nelle coordinate $\mathbf{Q}$. È facile vericare che tale operazione non causa alcun errore nella linearizzazione, poichè trasformando lo stato d' equilibrio attraverso T

$$\mathbf{q}_e = T^{-1}\tilde{\mathbf{q}}_e = T^{-1}\begin{pmatrix}0\\0\\.\\.\\.\\0\\0\end{pmatrix} = \begin{pmatrix}0\\0\\.\\.\\.\\0\\0\end{pmatrix} = \mathbf{q}_e$$

ci si rende conto che il punto di equilibrio nelle coordinate $\mathbf{Q}$ è, allo stesso modo, il vettore nullo. Vale lo stesso ragionamento per l' ingresso $\mathbf{u}$.

Avendo calcolato $\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{q}}$ e $\frac{\partial f(\mathbf{q},\mathbf{u})}{\partial \mathbf{u}}$, che per brevità chiameremo rispettivamente A_{lin} e B_{lin}, nelle coordinate di simulazione, si può facilmente ottenere il sistema nelle coordinate di controllo applicando le note trasformazioni lineari

$$\begin{cases} \tilde{\mathbf{q}} = T\mathbf{q} \\ \tilde{A}_{lin} = TA_{lin}T^{-1} \\ \tilde{B}_{lin} = TB \end{cases} \tag{5.33}$$

In totale si hanno tre sistemi: il sistema reale governato dall' equazione non lineare $\mathbf{q}_{k+1} = f(\mathbf{q}_k, \mathbf{u}_k)$, il sistema linearizzato in realizzazione di simulazione definito da $\mathbf{q}_{k+1} = A_{lin}\mathbf{q}_k + B_{lin}\mathbf{u}_k$ e il sistema linearizzato in realizzazione di controllo descritto dall' equazione $\tilde{\mathbf{q}}_{k+1} = \tilde{A}_{lin}\tilde{\mathbf{q}}_k + \tilde{B}_{lin}\tilde{\mathbf{u}}_k$. Il sistema linearizzato, se simulato, presenta dei gravi paradossi fisici di cui vale la pena parlare:

- la pallina non risulta più vincolata al piatto, e addirittura in alcuni casi gli passa attraverso, soprattutto ad alte velocità: il motivo di ciò è il fatto che la linearizzazione è stata fatta a velocità nulle, quindi all' aumentare di esse il comportamento del sistema tangente si discosta sempre di più rispetto a quello del sistema reale, provocando comportamenti anomali
- la pallina non si alza nè si abbassa mai, ma si muove su un piano orizzontale; il motivo di cioè è il fatto che la linearizzazione è avvenuta su uno stato con orientamento orizzontale della piattaforma, e per tal motivo la traiettoria della pallina è planare

Verrà usato il sistema non lineare $\mathbf{q}_{k+1} = f(\mathbf{q}_k, \mathbf{u}_k)$ per la simulazione in coordinate $\mathbf{Q}$, e il sistema linearizzato in coordinate $\tilde{\mathbf{Q}}$ per il controllo. Il sistema

linearizzato in coordinate **Q** verrà scartato, perchè a causa dei paradossi appena visti, risulta inadeguato per una simulazione realistica.

Ricapitolando useremo due modelli:

- **Modello non lineare di simulazione**

$$\mathbf{q}_{k+1} = f(\mathbf{q}_k, \mathbf{u}_k), \tag{5.34}$$

- **Modello linearizzato in realizzazione di controllo**

$$\tilde{\mathbf{q}}_{k+1} = \tilde{A}_{lin}\tilde{\mathbf{q}}_k + \tilde{B}_{lin}\mathbf{u}_k \tag{5.35}$$

5.4 Il regolatore lineare quadratico

Il regolatore lineare quadratico, è un tipo di controllore applicabile in sistemi lineari, basato su una controreazione dello stato.

La caratteristica che lo contraddistingue è la scelta ottima della matrice K, tale da trovare l' ingresso a norma minima che faccia discostare lo stato il meno possibile dal punto di equilibrio: il tutto si traduce in un problema di minimo dove il funzionale di costo è definito, nel continuo, come:

$$J = \int_0^\infty e^T(t)Qe(t) + u^T(t)Ru(t) + e^T(t)Nu(t)\,dt \tag{5.36}$$

essendo $e(t)$ lo stato che si vuole portare a 0, nel nostro caso specifico $\tilde{\mathbf{q}}(t)$, e $u(t)$ l' ingresso dipendente dallo stato con legge di controllo $u(t) = -Ke(t) = -K\tilde{\mathbf{q}}(t)$; il funzionale nel nostro caso si può riscrivere come

$$J = \int_0^\infty \tilde{\mathbf{q}}^T(t)Q\tilde{\mathbf{q}}(t) + \tilde{\mathbf{q}}^T(t)KRK\tilde{\mathbf{q}}(t) - \tilde{\mathbf{q}}^T(t)NK\tilde{\mathbf{q}}(t)\,dt \tag{5.37}$$

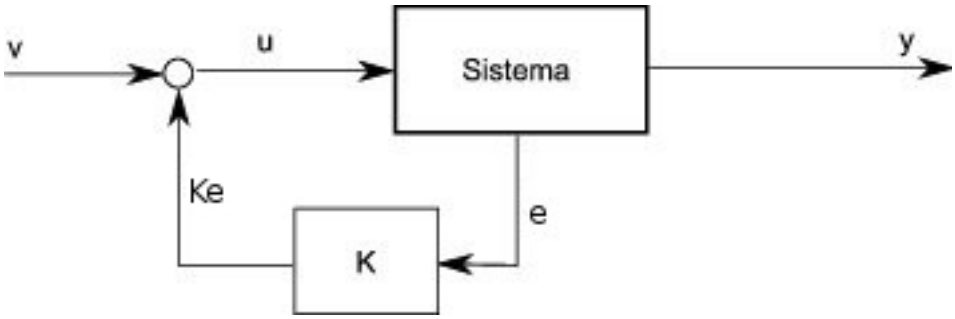

Figure 5.3: Sistema con retroazione dallo stato

dove Q e R sono le matrici dei pesi, definite positive. N è una matrice che effettua una norma mista tra stato e ingresso, ma verrà posta nulla. Riflettendo su questo funzionale, ci rendiamo conto che $e^T Qe$ e $u^T Ru$ sono forme quadratiche che esprimono le norme ponderate dello stato e dell' ingresso: facendone l' integrale otteniamo, pensando al caso monodimensionale e poi estendendolo a più dimensioni, l' area sottesa dalle funzioni, che è un indice di quanto esse si discostano dal punto d'equilibrio, e per quanto tempo. Minimizzando J, perciò, otteniamo una matrice di guadagni che garantisce massime prestazioni di controllo. Nel discreto, come nel nostro caso, il funzionale si definisce attraverso la sommatoria

$$J_d = \sum_{k=0}^{\infty} \tilde{\mathbf{q}}^T(k) Q \tilde{\mathbf{q}}(k) + \tilde{\mathbf{q}}^T(k) K^T R K \tilde{\mathbf{q}}(k) + \tilde{\mathbf{q}}^T(k) N K \tilde{\mathbf{q}}(k) \tag{5.38}$$

La soluzione del problema di minimo avviene grazie all' equazione di Riccati, che MATLAB implementa nella funzione `dlqr`(), nella quale vanno passate come parametri le matrici di sistema, ovviamente nelle coordinate di controllo $\tilde{\mathbf{Q}}$, $\tilde{A}_{lin}$e $\tilde{B}_{lin}$, e le matrici dei pesi Q , R, e N.

Un aspetto sicuramente fondamentale, che dimostra anche l' abilità del progettista, è la scelta delle matrici Q e R. Come già accennato, tali matrici devono necessariamente essere definite positive, ossia

$$\begin{cases} x^T Q x > 0 & \forall x \neq 0 \\ x^T R x > 0 & \forall x \neq 0 \end{cases}$$

condizione che garantisce che le norme pesate siano sempre positive.

Per verificare che una matrice sia definita positiva ci sono vari criteri; verranno indicati solo i più utlizzati

- Ogni matrice simmetrica con tutti gli autovalori strettamente positivi è definita positiva
- Criterio di Sylvester: una matrice simmetrica è definita positiva se e solo se i suoi minori principali sono tutti positivi

Per questa tesi, per non incontrare problemi, sono state usate matrici diagonali a elementi posivi, che risultano sicuramente definite positive per il primo criterio. Per la scelta degli elementi sulla diagonale della matrice Q è stato fatto un discorso di priorità delle variabili da stabilizzare: nel nostro caso la priorità è certamente stabilizzare la posizione della pallina nel piatto, e rendere la sua velocità nulla, quindi si daranno pesi elevati agli elementi della matrice riferiti a p e $\dot{p}$. L' importanza della posizione del piatto invece è secondaria, e quindi verranno dati pesi notevolmente minori alle variabili x e $\dot{x}$; alla luce di queste considerazioni si è scelta la matrice Q in questo modo:

$$Q = \text{diag}\left(\begin{bmatrix}500\\500\\500\end{bmatrix}\begin{bmatrix}2\\2\\2\end{bmatrix}\begin{bmatrix}3\\3\\3\end{bmatrix}\begin{bmatrix}5\\5\\5\end{bmatrix}\begin{bmatrix}1000\\1000\\1000\end{bmatrix}\begin{bmatrix}5\\5\\5\end{bmatrix}\right)$$

ottenendo prestazioni abbastanza buone.

Per quanto riguarda la matrice R, che invece dà il peso all' ingresso, si deve fare un discorso energetico: per prestazioni eccezionali è opportuno scegliere questa matrice con elementi il più possibile vicini allo zero, affinchè nel funzionale J il contributo del termine $u^T Ru$ sia trascurabile, e di fatto venga minimizzata solamente la norma ponderata dell' errore di stato, $e^T Qe$. In questo modo, però, si avrà uno sforzo di controllo elevatissimo, che può portare conseguenze come la saturazione degli attuatori, eccessivi dispendi energetici, un pericoloso surriscaldamento per effetto Joule delle componenti resistive dei motori, o conseguenze più gravi come danni alla struttura meccanica. Nella realtà quindi si sceglie, in base alle specifiche degli attuatori e del sistema meccanico, una matrice R che realizzi un compromesso tra prontezza di controllo e moderazione degli ingressi. Nel nostro caso è stata scelta piuttosto bassa, pari a

$$R = \text{diag}\left(\begin{bmatrix}0.1\\0.1\\0.1\end{bmatrix}\begin{bmatrix}0.1\\0.1\\0.1\end{bmatrix}\right)$$

La matrice N, come già detto, è stata posta nulla.

È da sottolineare il fatto che la scelta delle matrici di peso, soprattutto R, con grande probabilità dovrà essere rieffettuata da capo al momento dell' implementazione su robot reale, poichè dipende da parametri elettrici e strutturali presenti solo nel robot reale, non considerati in simulazione. Discorso simile vale per Q, che potrebbe generare un comportamento diverso da quello simulato a causa della linearizzazione del sistema e potrebbe aver bisogno di qualche ritocco in fase di test sul robot reale.

5.5 Risultati

Il controllo LQR dà ottimi risultati di stabilità: l' errore si annulla in un

tempo relativamente breve, con oscillazioni minime. Mostreremo in seguito alcuni grafici che descrivono l' andamento temporale della norma Euclidea di tutte le variabili di stato e dell' errore di posizione e velocità.

Il primo è quello in figura 5.4, che mostra l' evoluzione della posizione del centro x della piattaforma, e della pallina, p; si nota subito che i grafici sono pressappoco simili, poichè nella matrice dei pesi Q è stato dato un peso molto elevato all' errore di posizione, ragion per cui il controllore garantisce un discostamento molto piccolo della pallina dal centro del piatto; il valore a regime della posizione della pallina non è zero: bisogna ricordare che la pallina ha un raggio non nullo, e perciò in condizioni di equilibrio la sua posizione è $\begin{pmatrix} 0 \\ 0 \\ r \end{pmatrix}$ con r pari a 0.1.

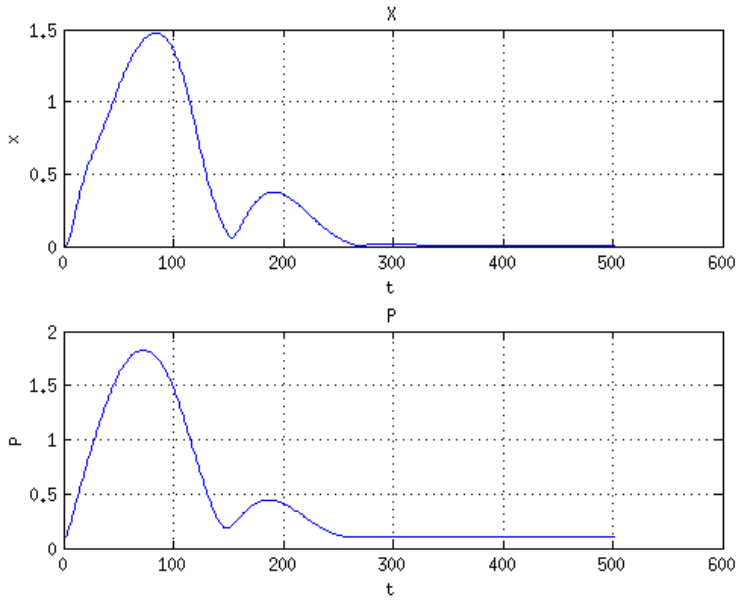

Figure 5.4: Grafico della posizione

Guardando il grafico della velocità in figura 5.5 notiamo che presenta delle oscillazioni più pronunciate e raggiunge valori relativamente alti, soprattutto per quanto riguarda la velocità della pallina. È un risultato che ci si poteva aspettare, considerando il fatto che abbiamo dato bassa priorità alla stabilizzazione delle velocità; una scelta progettuale di questo tipo è necessaria per l'

efficienza del controllo, perchè con velocità troppo basse della piattaforma si ha una minore destrezza nel recuperare la stabilità della pallina. Dal grafico della velocità della piattaforma $\dot{x}$ si intuisce anche il fatto che è stato dato un peso molto basso alla norma degli ingressi: si notano infatti pendenze piuttosto elevate dei grafici delle velocità e un profilo piuttosto frastagliato, sintomo di accelerazioni relativamente elevate come mostrato in figura 5.6.

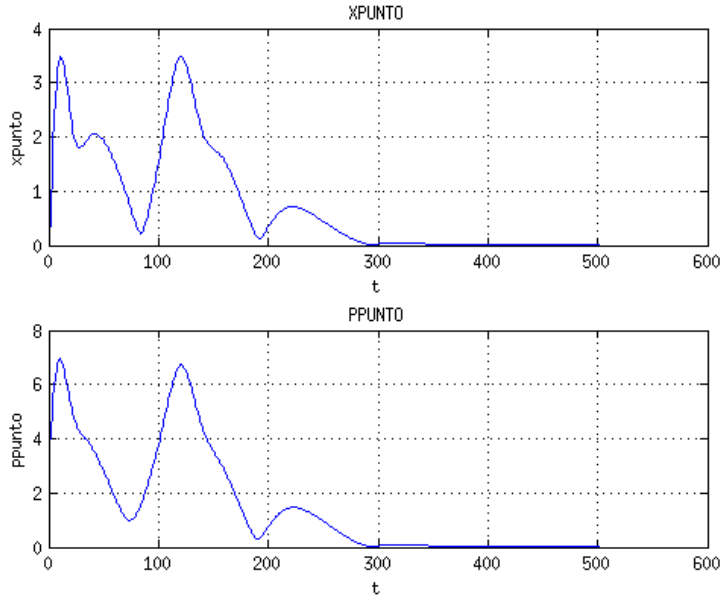

Figure 5.5: Grafico della velocità

In figura 5.7 abbiamo i grafici degli errori di posizione e velocità della pallina: notiamo che l' errore di posizione è piuttosto piccolo rispetto alla posizione assoluta, sintomo di un controllo efficace; l' errore di velocità è un po' più grande perchè la matrice dei pesi Q dà una priorità bassa alla minimizzazione di questa variabile. Il grafico della posizione può essere analizzato anche in una visualizzazione tridimensionale in figura 5.8, che mostra la traiettoria della pallina nel tempo: sugli assi orizzontali abbiamo le coordinate della pallina nel sistema di riferimento solidale alla piattaforma, e sull' asse z il tempo.

Mostriamo in figura 5.9 dei grafici di importanza secondaria rispetto a quelli finora illustrati, che rappresentano l' andamento delle norme dell' angolazione e della velocità angolare della piattaforma. Anche qui possiamo fare la stessa osservazione fatta sui grafici della velocità: l' evoluzione di ω presenta forti

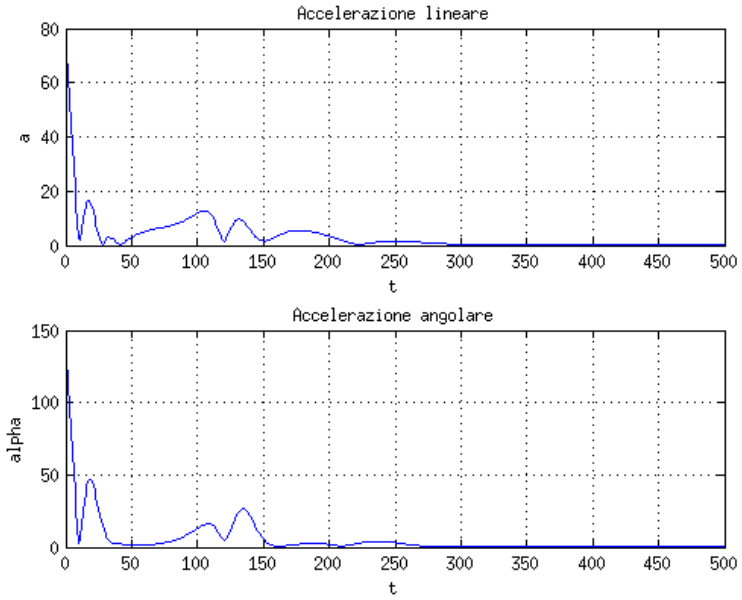

Figure 5.6: Accelerazioni della piattaforma

pendenze e brusche variazioni, dovute ad accelerazioni angolari elevate, non essendo state pesate molto nel processo di minimizzazione.

Per finire mostriamo in figura 80 un grafico che traccia la traiettoria percorsa dalla pallina durante l' intero tempo di simulazione

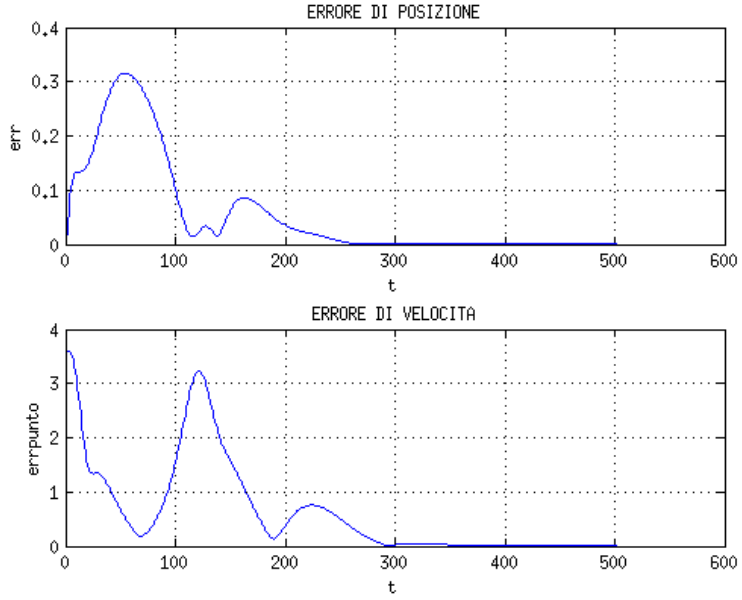

Figure 5.7: Grafico dell' errore di posizione e velocità

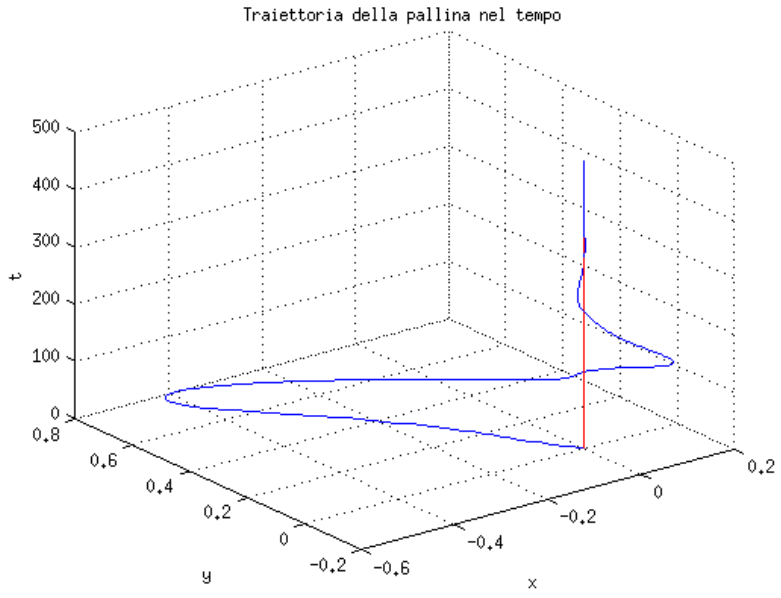

Figure 5.8: Grafico 3D dell' errore di posizione e velocità

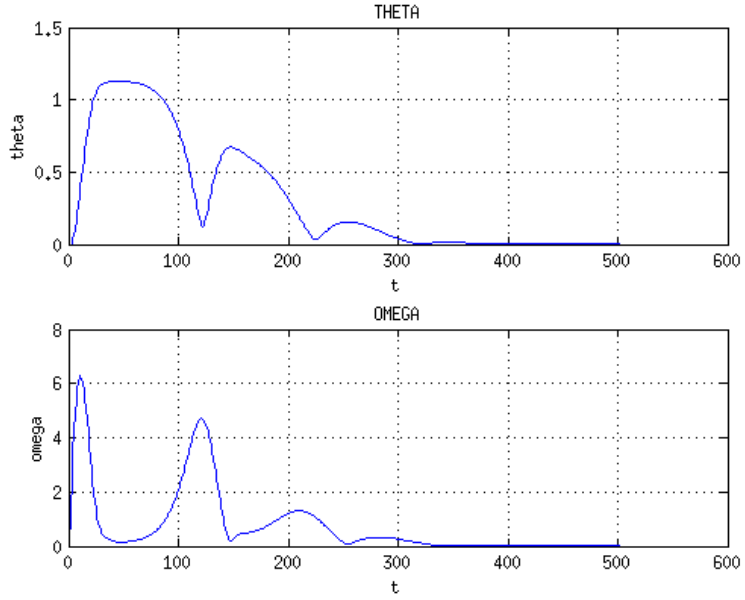

Figure 5.9: Grafico delle variabili angolari della piattaforma

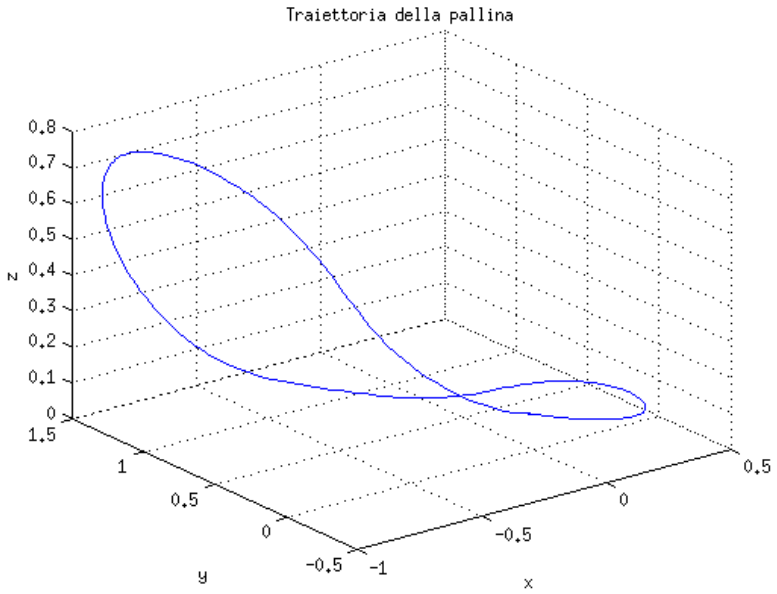

Figure 5.10: Traiettoria della pallina

Chapter 6

La programmazione ad oggetti in MATLAB

La definizione formale del termine oggetto è molto difficile da dare, e nel farlo è molto facile sfociare in ambiti filosofici. L' esigenza di utilizzare una programmazione orientata ad oggetti nasce dalla necessità di definire un modello, una struttura, sulla quale poter creare più enti, detti *oggetti.* In termini informatici, la struttura di un oggetto viene denominata *classe,* e un oggetto creato su tale struttura viene denominato *istanza della classe.*

Ogni classe è definita attraverso *attributi* e *metodi:* gli *attributi* dell' oggetto sono le sue caratteristiche, le sue proprietà intrinseche: nel caso della piattaforma, tra gli attributi troviamo la sua dimensione, la sua velocità, il suo orientamento. È da sottolineare il fatto che oggetti diversi appartenenti alla stessa classe sono accomunati dagli attributi che posseggono, ma possono differire per il valore che essi assumono: esistono infatti piattaforme di diverse dimensioni, tutte appartenenti alla stessa classe *piatto*; di un oggetto si può modificare il valore assunto dagli attributi, ma non se ne possono aggiungere di nuovi o eliminare di esistenti.

I *metodi* di una classe sono le operazioni che l' oggetto istanziato può eseguire: possono avere o meno variabili di input e variabili di output, e si programmano in modo simile alle funzioni; l' aspetto importante è che essi hanno

accesso agli attributi dell' oggetto a cui appartengono, e hanno la facoltà sia visualizzarli che di modificarli. Nella classe piatto tra i metodi troviamo `move`() che serve muovere la piattaforma, `PID`() che serve per effettuare il controllo PID. Per accedere ad un attributo oppure un metodo di un oggetto istanza di una classe si usa il punto. Ad esempio avendo un oggetto `v` della classe piatto, per riferirsi al valore del suo attributo `xpunto` si usa la notazione `v.xpunto`, e per richiamare il suo metodo `PID`() si usa analogamente `v.PID`() passando gli opportuni argomenti. Facciamo un esempio esplicativo della sintassi in MATLAB per dichiarare attributi e metodi: in questo ambiente è usuale chiamare gli attributi come proprietà:

- **Dichiarazione di proprietà, ossia attributi**

```
properties


    vass
    grafico
    ball
    x             % coordinate del centro
    xpunto        % velocità lineare del centro
    theta         % (angoli di eulero)
    omega         % velocità angolare del piatto

end
```

- **Dichiarazione di metodi**

```
methods


    function obj=piatto(v)
```

```
            ...
            ...
            ...

        end



        function disegna(obj)

            ...
            ...
            ...

        end
        ...
        ...
        ...

    end
```

Nel codice di esempio si notano in particolare due cose: la prima è la presenza di `obj`: si tratta di un puntatore all' oggetto stesso istanza della classe. Ad esempio se con un metodo voglio impostare nullo il valore l' orientamento, che tra gli attributi è denominato `theta` ed è un vettore colonna tridimensionale, uso la sintassi:

```
obj.theta = [0 0 0]';
```

Il puntatore `obj` serve per specificare che si vuole modificare l' *attributo* dell' oggetto stesso che sta usando il metodo; se si fosse inserito solamente `theta`, il programma avrebbe assegnato il valore ad una variabile locale `theta` che sarebbe andata persa al termine dell' esecuzione del metodo.

La seconda cosa importante da notare è la presenza di un metodo che ha lo stesso nome della classe su cui è dichiarato, nel nostro caso *piatto*: si tratta di

un metodo speciale, chiamato *costruttore,* che deve essere presente in ogni classe e si riconosce per la caratteristica di avere *sempre* lo stesso nome della classe a cui appartiene; viene richiamato ogni qual volta si crea un' istanza della classe. Il suo ruolo è quello di inizializzare tutti gli attributi dell' oggetto e svolgere tutte le operazioni necessarie affinchè la costruzione dello stesso vada a buon fine. Notiamo che come output di tale metodo abbiamo `obj`: ciò è coerente con il discorso fatto finora, perchè l' output del costruttore deve essere un oggetto della sua classe. Per creare un oggetto v di tipo piatto si usa la sintassi

```
v = piatto(vassoioiniziale);
```

passando la variabile `vassoioiniziale` che verrà gestita dal costruttore secondo le modalità con cui è stato programmato.

È inoltre importante notare che se un metodo di un oggetto deve modificare uno o più attributi dello stesso, come `disegna`(`obj`) nell' esempio di sopra, *solo nella sua dichiarazione all' interno della classe* è necessario passare `obj` come parametro; al momento della chiamata invece, non è necessario passare `obj`, e si passeranno normalmente tutti gli altri i parametri, se presenti. Ad esempio per richiamare `disegna` sul piatto v si scrive:

```
v.disegna;
```

Un' altra cosa importante è il fatto che un oggetto può avere tra gli attibuti anche altri oggetti: sulla base di questa caratteristica si apre una nuova sezione, che è quella delle relazioni tra le diverse classi di un programma.

6.1 Le relazioni tra le classi

I programmi non hanno mai una sola classe: se così fosse sarebbe inutile utilizzare un approccio orientato ad oggetti. In generale si hanno più classi che sono sempre legate tra di loro da determinate relazioni. Esistono vari tipi di relazioni, e qui verranno elencati solo alcuni, mostrando in figura 6.1 anche la notazione che usualmente si usa per schematizzarle.

Partiamo dall' associazione, la più generica delle relazioni tra classi: essa

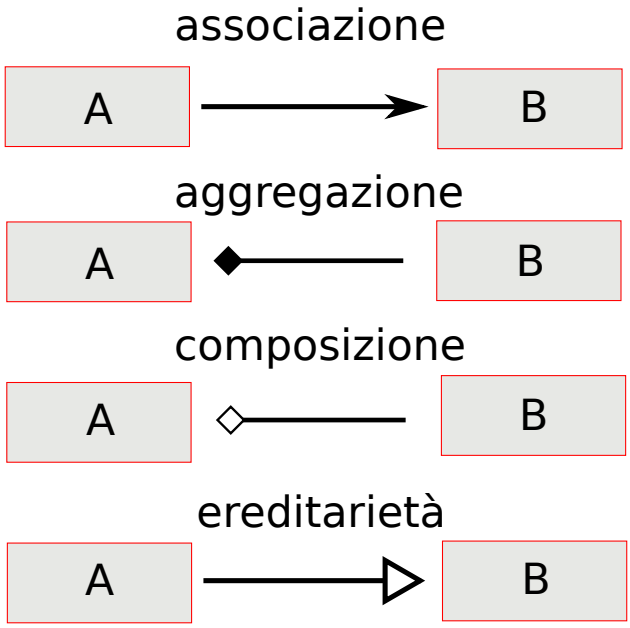

Figure 6.1: Relazioni tra classi

indica che c'è un collagamento tra gli oggetti istanza delle classi. Nel nostro caso c'è sicuramente un' associazione tra il piatto e la pallina, poichè lo stato del piatto influenza la traiettoria della pallina. Andando più in profondità possiamo raffinare la semantica delle associazioni distinguendo tra composizione e aggregazione: la composizione è la relazione per cui un oggetto è composto da altri oggetti e non può esistere senza questi: nel nostro caso possiamo fare l' esempio dell' oggetto *sistema* che si può dire composto da un *piatto* e una *pallina* e che non può esistere senza questi due. Si parla di aggregazione invece quando c'è un' associazione tra oggetti che possono esistere anche da soli ma che interagiscono tra loro, come nel caso della pallina e della piattaforma. È sempre bene mostrare un grafico come quello in figura 6.2 che mostra le relazioni tra le classi, per avere una buona visione complessiva del programma.

L' altra relazione molto importante tra classi è l' ereditarietà: il meccanismo tramite il quale una classe A ha implicitamente definito su se stessa ciascuno degli attributi e dei metodi di una classe B, come se tali attributi e operazioni fossero stati definiti per A stessa. Oltre a questi, che si dicono *ereditati* dalla classe B, detta *superclasse,* possono esserci anche altri attributi e metodi definiti solo sulla classe A. Nel nostro programma ogni classe eredita dalla classe *handle,*

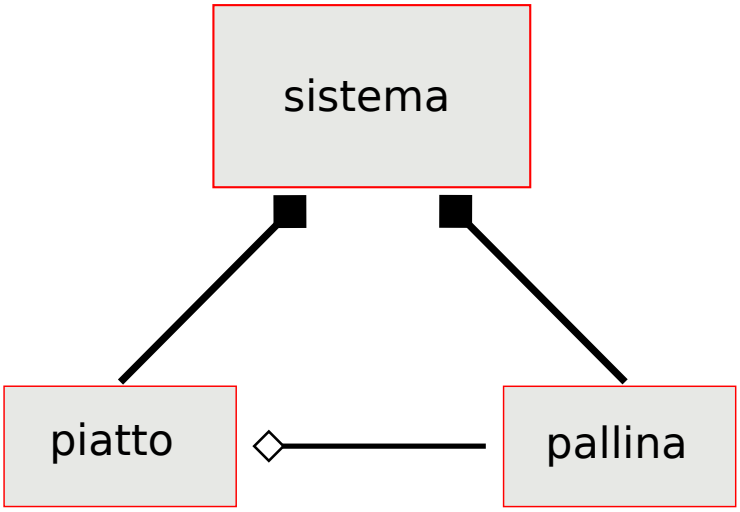

Figure 6.2: Schema delle relazioni tra le classi

definita dall' ambiente stesso, che contiene molti metodi che possono tornare utili in varie circostanze.

Chapter 7

Il modello di simulazione su MATLAB

Dopo aver trovato tutte le leggi fisico matematiche alla base del sistema in esame si può passare alla sua simulazione su calcolatore. A tal scopo, è stato scelto il programma MATLAB®, utilizzato con un approccio orientato ad oggetti: è stata quindi creata una classe per la piattaforma, una per la pallina, una per il sistema piattaforma-pallina, e un file main dove le tre classi vengono sfruttate.

7.1 Cenni sulla grafica in MATLAB

In ambiente MATLAB esistono alcune funzioni implementate nativamente, che consentono di creare animazioni grafiche: verrà di seguito illustrata la loro sintassi e il modo in cui sono state usate per creare l' animazione del piatto con la pallina:

7.1.1 La funzione `patch()`

Questa funzione è di fondamentale importanza per creare poligoni in 2D e in 3D.

La sintassi generica è del tipo

```
>> handle = patch(X,Y,Z,C)
```

dove

- X rappresenta il vettore di tutte le componenti x dei vertici
- Y rappresenta il vettore di tutte le componenti y dei vertici
- Z rappresenta il vettore di tutte le componenti z dei vertici
- C è una stringa che sta ad indicare il colore
- handle è un puntatore all' oggetto grafico

Per dichiarare in modo compatto l' insieme dei vertici del poligono, la strategia più conveniente è quella di scriverli sotto forma di matrice:

$$vass = \begin{bmatrix} x_1 & x_2 & & & & x_{n-1} & x_n \\ y_1 & y_2 & . & . & . & y_{n-1} & y_n \\ z_1 & z_2 & & & & z_{n-1} & z_n \end{bmatrix}$$

dove le n colonne sono le coordinate degli n vertici del poligono. Usando questa sintassi possiamo passare alla funzione `patch()`, come parametri X, Y, Z, le righe della matrice $vass$. Nel nostro caso il touchscreen ha forma rettangolare, quindi n è pari a quattro.

7.1.2 La funzione `set()`

Come già visto, nell' approccio object-oriented ogni oggetto ha degli attributi e dei metodi. La funzione `patch()`, di fatto, funge da costruttore per l' oggetto grafico poligonale che, come tutti gli oggetti, possiede attributi e metodi; `set()`

non è altro che un metodo che serve a cambiare gli attributi dell' oggetto: la sua sintassi è del tipo

```
set(handle,'PropertyName',PropertyValue)
```

La lista degli attributi di un oggetto poligonale creato con `patch`() è veramente lunga, quindi ne verranno elencati solo alcuni:

- FaceColor - regola il colore del poligono
- FaceAlpha - regola la trasparenza del poligono
- FaceLighting - regola in tipo di superficie de poligono, opaca o lucida
- EdgeColor - regola il colore dei bordi del poligono
- EdgeAlpha regola la trasparenza dei bordi
- EdgeLigthing - regola le modalità di riflessione della luce sui bordi
- XData - contiene le coordinate x di tutti i vertici
- YData - contiene le coordinate y di tutti i vertici
- ZData - contiene le coordinate z di tutti i vertici
- Visible - imposta la visibilità dell' oggetto
- EraseMode - imposta le modalità di refresh del grafico quando l' immagine viene aggiornata

Vediamo adesso come far muovere il poligono, cosa che verrà spiegata meglio in seguito: intuitivamente un movimento del poligono si ottiene cambiando le coordinate dei suoi vertici; la modifica viene fatta sugli attributi XData, YData e ZData utilizzando il comando `set`() con la sintassi

```
set(obj.grafico,'x',obj.vass(1,:));
set(obj.grafico,'y',obj.vass(2,:));
set(obj.grafico,'z',obj.vass(3,:));
```

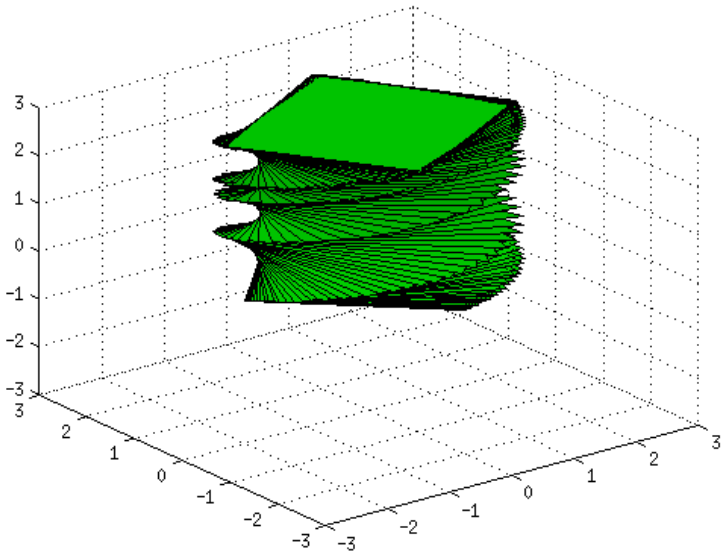

Figure 7.1: Sovrapposizione di fotogrammi

avendo ovviamente la matrice `vass` aggiornata alle nuove coordinate.

Utilizzando questo tipo di approccio grafico object oriented, si evitano molti problemi che si presentano invece utilizzando l' approccio tradizionale con il comando `plot3`(): uno dei più critici è la sovrapposizione dei fotogrami successivi come si può vedere in figura 7.1, che va risolto usando dei comandi di refresh della figura; l' approccio risulta comunque obsoleto, quindi è stato scartato.

7.1.3 La funzione `sphere`()

Per quanto riguarda la pallina, è stata usata la funzione `sphere`() per la rappresentazione; in ingresso viene impostato il numero di superfici con cui si approssima la sfera, in altre parole, i lati del poligono che si ottiene sezionandola. Ovviamente un numero elevato garantisce una grafica dettagliata, a discapito di un maggior carico computazionale.

Si sceglie come compromesso, considerando le dimensioni ridotte della pallina, un valore di 15.

La funzione ritorna tre matrici contenenti le coordinate (x, y, z) dei punti

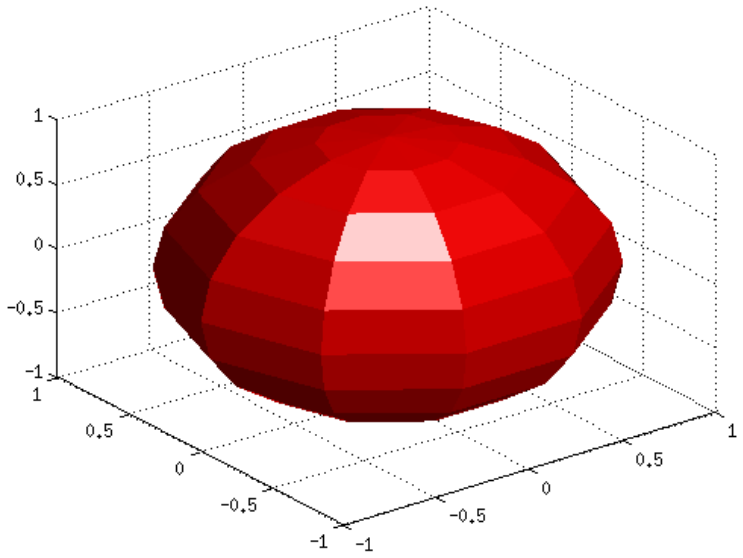

Figure 7.2: Sfera approssimata con 10 superfici

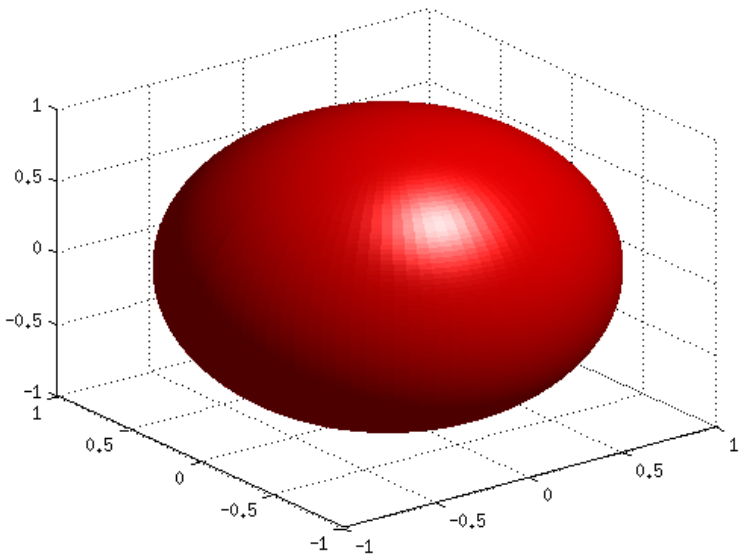

Figure 7.3: Sfera approssimata con 100 superfici

della sfera, da passare come argomento alla funzione `surf`() affichè questa disegni la sfera.

7.1.4 La funzione `surf`()

È la funzione utilizzata per rappresentare la sfera: prendendo in ingresso 3 variabili che esprimono le componenti X Y e Z del grafico, traccia la superficie corrispondente; le figure 7.2 e 7.3 sono state generate attraverso questa funzione. Analogamente a `patch`() può essere usata per creare oggetti grafici dotati di proprietà che possono essere modificate con il comando `set`(). I dettagli verranno spiegati nella classe pallina.

7.2 Le variabili cardine della simulazione

Il main è strutturato in modo da avere una prima parte in cui vengono dichiarate tutte le variabili *globali* fondamentali per il funzionamento dell' intero programma, che sono:

- Ts, il tempo di campionamento scelto per la simulazione grafica, pari a 0.01.
- g, l' accelerazione di gravità, che come sappiamo è un vettore rivolto verso il basso di modulo circa pari a 9.8. Viene quindi definito come $\begin{bmatrix} 0 \\ 0 \\ -9.80665 \end{bmatrix}$
- t, il vettore degli tempi; questo è un vettore fondamentale per la simulazione, perchè rappresenta di fatto la dimensione temporale del sistema; tutte le evoluzioni successive dello stato del sistema piatto-pallina sono calcolate in corrispondenza degli gli istanti contenuti in questo array. È stato preso un tempo totale di 5 secondi, quindi il vettore è stato definito come

```
t = linspace(0,5,5/Ts);
```

- *dimvass*, la dimensione del vassoio, posta pari a 2
- *vassoioiniziale*, la matrice contenente le coordinate dei vertici, al momento dell' inizializzazione. Equivale alle coordinate dei vertici quando si ha $x = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ e $\theta = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ L' utilità della globalizzazione di tale variabile sarà chiarita nel momento in cui verrà affrontato il metodo `vassoio`() della classe piatto; è stato scelto un piatto quadrato con una matrice di questo tipo:

$$vassoioiniziale = \begin{bmatrix} 0 & dimvass & 0 & -dimvass \\ -dimvass & 0 & dimvass & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

- $film$ o $filmPD$ è una matrice in cui viene memorizzata l' evoluzione dello stato del sistema: la colonna i-esima rappresenta lo stato all' istante i-esimo. La matrice ha un numero di righe pari a 18, dimensione dello stato, e un numero di colonne pari alla lunghezza del vettore t. È utile per raccogliere i dati di simulazione e poi successivamente graficarli.

7.2.1 La matrice $ACCEL$

Dedichiamo un paragrafo a parte alla variabile $ACCEL$, che sta alla base del movimento del piatto. Poichè la simulazione è stata fatta sulla base delle leggi di Newton, facendo quindi un' analisi di tutte le forze agenti sul sistema, si è preferito seguire un approccio basato sulle accelerazioni, per rendere più agevole il calcolo della traiettoria della pallina sul piatto;

Il movimento del piatto avviene grazie alla variabile $ACCEL$, definita come una matrice a sei righe, la cui k-esima colonna rappresenta l' accelerazione generalizzata $\begin{bmatrix} a_k \\ \alpha_k \end{bmatrix}$ impressa alla piattaforma al k-esimo istante:

$$ACCEL = \begin{bmatrix} a_{1,x} & a_{2,x} & & & a_{n-1,x} & a_{nx} \\ a_{1,y} & a_{2,y} & & & a_{n-1,y} & a_{ny} \\ a_{1,z} & a_{2,z} & . & . . & a_{n-1,z} & a_{nz} \\ \alpha_{1,x} & \alpha_{2,x} & . & . . & \alpha_{n-1,x} & \alpha_{nx} \\ \alpha_{1,y} & \alpha_{2,y} & & & \alpha_{n-1,y} & \alpha_{ny} \\ \alpha_{1,z} & \alpha_{2,z} & & & \alpha_{n-1,z} & \alpha_{nz} \end{bmatrix}$$

dove n è la lunghezza del vettore dei tempi t.

L' algoritmo di simulazione estrae ad ogni passo di campionamento la corrispondente colonna $\begin{bmatrix} a_k \\ \alpha_k \end{bmatrix}$ e la passa come parametro alla funzione `acceleration`(), dedicata al calcolo della posizione e della velocità all' istante di campionamento successivo.

Un concetto importante da sottolineare è che ogni funzione atta al movimento della piattaforma attraverso particolari traiettorie, **non la muove effettivamente,** ma aggiorna i valori futuri della matrice $ACCEL$ in modo da imprimere già dall' istante di campionamento successivo l' accelerazione desiderata.

Nel caso più generale si inizializza $ACCEL$ ad una matrice nulla, ma una situazione interessante potrebbe essere quella in cui la piattaforma touchscreen riceve un' accelerazione impulsiva all' inizio, e il controllo deve riportare il sistema perturbato all' equilibrio. Per questa situazione è stato predisposto il vettore ACC in cui verrà inserito il valore dell' accelerazione impulsiva, e la varibile intera $Tacc$ che indica la durata, in campioni, di tale accelerazione. In questo modo la matrice $ACCEL$ può essere inizializzata, ad esempio, in questo modo:

```
Tacc=3;
ACC=[0 3 1 , 0.5 2 0];
ACCEL = [ (diag(ACC) * ones (6,Tacc)) , zeros( 6 , (length(t)-Tacc) ) ];
```

avendo, in questo caso, una forma del tipo:

$$ACCEL = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & & & & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 & & & & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & . & . & . & 0 & 0 \\ 0.5 & 0.5 & 0.5 & 0 & 0 & . & . & . & 0 & 0 \\ 2 & 2 & 2 & 0 & 0 & & & & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & & & & 0 & 0 \end{bmatrix}$$

7.3 Le funzioni ausiliari

Nel programma sono state aggiunte anche alcune funzioni, che di frequente risultano utili per i calcoli.

7.3.1 `rot_elem()`

`Variabili in input:`

asse, è l' asse cardinale intorno al quale si vuole effettuare la rotazione, passato come stringa x, y, o z

angolo, è l' angolo di cui si vuole ruotare

`Variabili in output:`

R, la matrice che effettua la rotazione desiderata

La funzione valuta la stringa *asse* e sceglie di conseguenza la matrice da resistituire; chiamando l' angolo α le matrici di rotazione elementare sono:

- Con asse z :

$$R_z(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- con asse y:

$$R_y(\alpha) = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix}$$

- con asse x:

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

L' utilità di questa funzione si trova nel fatto che se si vuole trovare la matrice di rotazione che da un detereminato orientamento iniziale espresso con gli angoli di Eulero $\Theta_0 = \begin{bmatrix} \theta_x \\ \theta_y \\ \theta_z \end{bmatrix}$ che applicando una rotazione $\Psi = \begin{bmatrix} \psi_x \\ \psi_y \\ \psi_z \end{bmatrix}$ porti all' orientamento $\Theta_1 = \begin{bmatrix} \theta_x + \psi_x \\ \theta_y + \psi_y \\ \theta_z + \psi_z \end{bmatrix}$, basta fare il prodotto per le matrici elementari in questo modo:

$$R(\Psi) = R_z(\psi_z)R_y(\psi_y)R_x(\psi_x)$$

7.3.2 `asseangolo()`

`Variabili in input:`

s, è il vettore intorno al quale si vuole effettuare la rotazione

a, è l' angolo di cui si vuole ruotare

`Variabili in output:`

matrice, la matrice che effettua la rotazione desiderata

Questa funzione serve per trovare una matrice che effettui una rotazione di un angolo a attorno all' asse s. Per trovarla si effettuano le seguenti operazioni:

- si ruota il sistema attraverso rotazioni elementari intorno all' asse z e y fino a sovrapporre l' asse di rotazione s con l' asse z
- si effettua una rotazione pari a a intorno al nuovo asse z
- si ripristina l' orientamento originale

L' uscita della funzione, *matrice*, che effettua in sequenza queste tre operazione si ricava dai prodotti matriciali

$$matrice = R_z(\alpha)R_y(\beta)R_z(a)R_y(-\beta)R_z(-\alpha)$$

che sviluppando i calcoli danno come risultato

$$matrice = \begin{bmatrix} s_x^2(1-\cos a)+\cos a & s_x s_y(1-\cos a)-s_z\sin a & s_x s_z(1-\cos a)+s_y\sin a \\ s_x s_y(1-\cos a)+s_z\sin a & s_y^2(1-\cos a)+\cos a & s_y s_z(1-\cos a)-s_x\sin a \\ s_x s_z(1-\cos a)-s_y\sin a & s_y s_z(1-\cos a)+s_x\sin a & s_z^2(1-\cos a)+\cos a \end{bmatrix}$$

7.4 La classe *piatto*

7.4.1 Gli attributi

Tra gli attributi di questa classe vi sono attributi necessari per l' animazione grafica, e altri che rappresentano le variabili di stato del sistema piatto:

- *ball* non è nè un attributo grafico, nè una variabile di stato: poichè abbiamo una relazione di aggregazione tra la classe pallina e la classe piatto, tra gli attributi di quest' ultima deve esserci anche un' istanza della classe pallina

Attributi grafici

- *vass* è una matrice 3×4, le cui colonne rappresentano le coordinate dei vertici della piattaforma rettangolare. Durante la simulazione, in seguito al moto del piatto, le coordinate dei vertici cambieranno, e conseguentemente anche la matrice
- *grafico* è un attributo che rappresenta l'oggetto grafico del piatto in MATLAB. Al momento della creazione di un'istanza della classe piatto, viene inizializzato tramite la funzione `patch`(), a un poligono a 4 vertici, dalle coordinate date dalle colonne della matrice *vass* vengono poi aggiornate durante la simulazione tramite il comando `set`().

Variabili di stato

- x rappresenta la posizione del centro della piattaforma
- *xpunto* rappresenta la velocità del centro della piattaforma
- *theta* rappresenta l' orientamento della piattaforma: sono stati usati gli angoli di Roll Pitch e Yaw, con assi fissi.
- *omega* rappresenta la velocità angolare della piattaforma; si ricorda che questa non coincide con le derivate degli angoli RPY, ma rappresenta il vettore che ha come direzione l' asse della rotazione, e come modulo

il modulo della velocità angolare, scelto in base alla regola della mano destra.

7.4.2 I metodi

7.4.2.1 Il costruttore

`Variabili in input:`

v, la matrice delle coordinate dei vertici della piattaforma

Il costruttore è un metodo speciale presente in ogni classe, che viene richiamato al momento della creazione di un'istanza di quella classe: nel nostro caso specifico, passando in input la matrice dei vertici della piattaforma, il costruttore crea l' oggetto piatto impostando correttamente tutti i suoi attributi:

- x viene impostata alla posizione risultato della media dei vertici del piatto, passati in ingresso, che essendo un rettangolo, corrisponderà all' intersezione delle diagonali
- *xpunto* viene impostata nulla
- *theta* viene impostato nullo, che equivale ad un orientamento orizzontale
- *omega* viene impostato nullo
- viene creato l' oggetto grafico attraverso la funzione `patch()`*:* per una buona visualizzazione vengono impostati i limiti degli assi cartesiani ad un valore pari alla dimensione del vassoio +1 con il comando

```
d=dimvass + 1;
figure(1); grid on; axis([-d d -d d -d d]);
```

Vengono settati gli attributi grafici in modo da avere un piatto lucido e una corretta visualizzazione del movimento

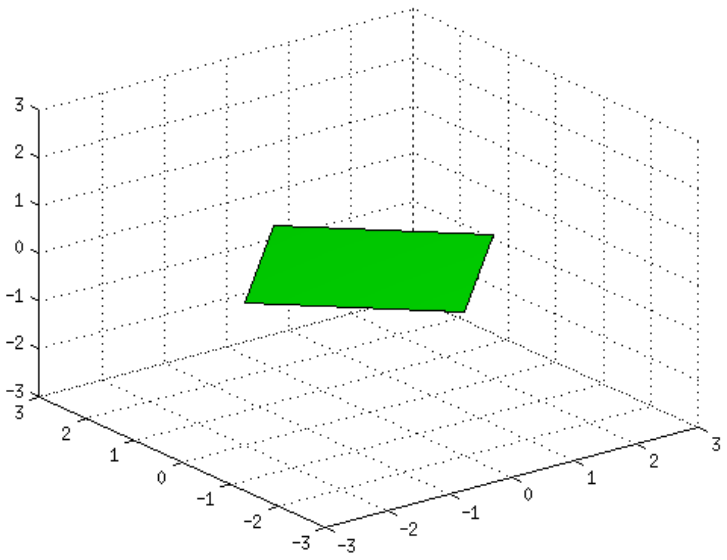

Figure 7.4: Il grafico della piattaforma al memento della creazione

```
set(obj.grafico,'FaceLighting','gouraud');
set(obj.grafico,'EdgeLighting','gouraud');
set(obj.grafico,'EraseMode','normal');
camlight;
```

7.4.2.2 `getnormal()`

`Variabili in output:`

n, il versore normale alla piattaforma

È di grande importanza, in molti punti del programma, conoscere il versore normale alla piattaforma; per calcolarlo ci sono diversi modi, ma quello scelto in questa tesi si basa su una riflessione sui sistemi di riferimento: abbiamo il sistema di terra ${}^0S$ e il sistema della piattaforma ${}^tS$, e per passare dal primo al secondo c'è la formula già vista nel capitolo dedicato, che richiamiamo velocemente:

$$^0p = ^0 o_{piatto} + ^0 R_p {}^p p$$

In questo caso non siamo interessati alla traslazione, ma solo all' orientamento, quindi assumiamo $^0o_{piatto} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$; di conseguenza si avrà:

$$^0p = {}^0R_p {}^p p$$

Interpretiamo adesso questa formula come trasformazione di generici vettori, non necessariamente di posizione: il versore normale alla piattaforma, nelle coordinate ^pS, è ovviamente uguale a $\hat{n} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$. Esprimendo $\hat{n}$ nelle coordinate ^0S, si ottiene il versore cercato

$$\hat{n} = {}^0R_p \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Riflettendo su questa espressione, si scopre può essere interpretata anche in un altro modo, che porta alla stessa conclusione: per definizione, la matrice di rotazione ha per colonne i 3 versori rappresentanti gli assi cartesiani del sistema ^pS espressi nelle coordinate di ^0S, ha quindi la forma:

$$^pR_0 = \begin{pmatrix} ^0x_x & ^0y_x & ^0z_x \\ ^0x_y & ^0y_y & ^0z_y \\ ^0x_z & ^0y_z & ^0z_z \end{pmatrix}$$

Moltiplicare una matrice per il vettore $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ equivale ad estrarre dalla matrice la terza colonna, che corrisponde proprio all' asse z del sistema di riferi-

mento ${}^{p}S$ espresso nelle coordinate ${}^{0}S$, che coincide per definizione con il versore normale alla piattaforma.

7.4.2.3 `getvass`()

Un semplicissimo metodo che stampa sulla command window l'attuale matrice dei vertici.

7.4.2.4 `status`()

`Variabili in output:`

v, un vettore dato dalla concatenazione delle variabili di stato x, $\dot{x}$, θ e ω.

Ritorna lo stato attuale del sistema come

$$v = \begin{bmatrix} x_x \\ x_y \\ x_z \\ \dot{x}_x \\ \dot{x}_y \\ \dot{x}_z \\ \theta_x \\ \theta_y \\ \theta_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

7.4.2.5 `acceleration`()

`Variabili in input:`

acc, il vettore accelerazione istantantanea $\begin{bmatrix} a \\ \alpha \end{bmatrix}$

`Variabili in output:`

cordball, la posizione della pallina nelle coordinate del piatto ${}^{p}S$

velball, la velocità della pallina nelle coordinate del piatto ${}^{p}S$

```
[cordball, velball ] = acceleration(obj,acc)
```

È probabilmente il metodo più importante di questa classe, poichè attua il movimento della piattaforma, modificando le sue variabili di stato. Utilizza le leggi già viste del capitolo della dinamica Newtoniana, che richiamiamo rapidamente:

$$x((k+1)Ts) = x(kTs) + v(kTs)Ts + a(kTs)\frac{Ts^2}{2}$$
$$v((k+1)Ts) = v(kTs) + a(kTs)Ts$$

$$\theta((k+1)Ts) = \theta(kTs) + G^{-1}\omega(kTs)Ts + G^{-1}\alpha(kTs)\frac{Ts^2}{2}$$

$$\omega((k+1)Ts) = \omega(kTs) + G^{-1}\alpha(kTs)Ts$$

Inoltre il metodo richiama la funzione `physics2`() della classe pallina e si preoccupa di aggiornare le coordinate dei vertici della piattaforma, richiamando l' apposita funzione `vassoio`() e aggiornando l' attributo *vass*.

7.4.2.6 `vassoio`()

`Variabili in output:`

v, la nuova matrice dei vertici, aggiornata secondo lo stato attuale della piattaforma

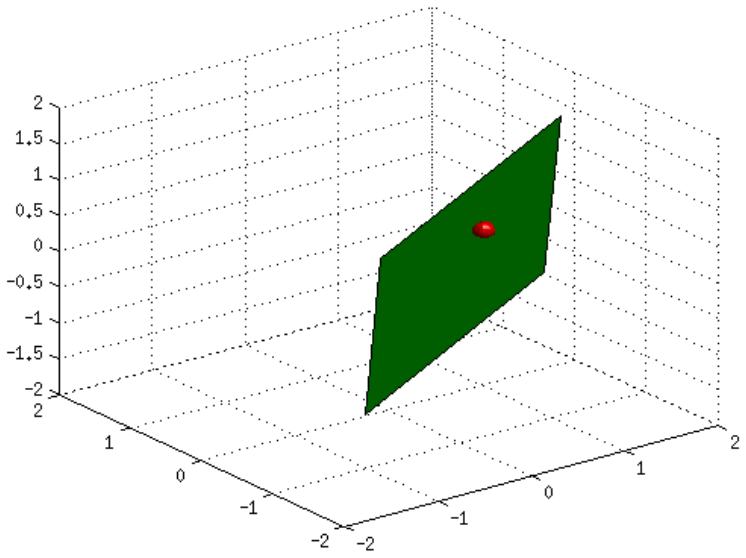

Figure 7.5: La piattaforma in movimento, con la pallina sopra

Questo metodo ha uno scopo puramente grafico: ritorna la matrice dei vertici, aggiornata secondo lo stato attuale.

Una caratteristica interessante di questa funzione è il fatto che non valuta lo spostamento del piatto rispetto alla posizione dell' istante precedente, bensì rispetto alla posizione iniziale: in altre parole, invece di considerare la rototraslazione infinitesimale che ha subito la piattaforma durante il periodo di campionamento, considera la sua configurazione rispetto alla configurazione iniziale, e la ridisegna "da capo" ad ogni istante, senza curarsi delle sue configurazioni passate. Questo metodo, anche se all' apparenza può sembrare contorto, risulta sicuramente migliore di quello in cui si considera la variazione di stato tra istanti successivi, poichè permette di fare i calcoli a partire dal solo stato attuale, senza dover salvare lo stato precedente ad ogni passo di campionamento, a patto che si conosca la configurazione iniziale. Da qui si comprende l' utilità della variabile globale *vassoioiniziale*, equivalente alle coordinate dei vertici del vassoio quando nella configurazione con $x = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ e $\theta = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$. L' algoritmo, in base all' orientamento θ attuale del piatto, calcola la matrice di rotazione

${}^0R_p$ e la premoltiplica per la matrice *vassoioiniziale*; è facile verificare che un prodotto di questo genere equivale ad applicare la rotazione ${}^0R_p$ ad ogni colonna della matrice *vassoioiniziale*, cioè ad ogni vertice della piattaforma. Successivamente si applica anche la traslazione sommando ad ogni colonna il valore della posizione x attuale; queste semplici operazione sono state implementate con il codice:

```
global vassoioiniziale;
R= rot_elem('z' , obj.theta(3)) * rot_elem('y' , obj.theta(2))...
...* rot_elem('x' ,obj.theta(1));
T= [ obj.x, obj.x, obj.x, obj.x ];
v= R*vassoioiniziale + T;
```

7.4.2.7 `disegna()`

È il metodo che realizza il grafico del piatto, tenendo conto della sua configurazione attuale. Innanzitutto richiama la funzione `vassoio()`, in modo da aggiornare le coordinate dei vertici, in caso non fossero già state aggiornate. Successivamente, attraverso il comando `set()`, imposta nel grafico la nuova posizione dei vertici, aggiornando in questo modo l' animazione. La sintassi è:

```
set(obj.grafico,'x',obj.vass(1,:));
set(obj.grafico,'y',obj.vass(2,:));
set(obj.grafico,'z',obj.vass(3,:));
```

Richiama infine anche il metodo `disegna()` della pallina, che analizzeramo nella sezione dedicata.

7.4.2.8 `move()`

`Variabili in input:`

trasl, la traslazione che si vuole far effettuare alla piattaforma

asse, l' asse della rotazione imposta

angolo, l' angolo di cui si vuole far ruotare la piattaforma

Ta, il tempo di accelerazione nel profilo di velocità triangolare

`Variabili in output:`

accelerazione, l' accelerazione da imporre per realizzare la manovra richiesta

Questo metodo è fondamentale per il movimento della piattaforma touchscreen: riceve in input le variabili necessarie a specificare la rototraslazione che si vuole effettuare e il tempo che deve impiegare a compierla, e restituisce l' accelerazione da applicare per attuare il movimento. Per un fatto di comodità e immediatezza nel dichiarare la rotazione desiderata, si è scelta la rappresentazione asse-angolo: il programma però, per fare i calcoli, utilizza la rappresentazione con gli angoli di Eulero, quindi dovrà effettuare la conversione in questo modo:

- viene calcolata la matrice di rotazione con la funzione `asseangolo`()
- vengono individuati gli elementi della matrice che combinati con semplici operazioni aritmetiche e trigonometriche resituiscono in modo univoco gli angoli di Eulero
- si ricava la rotazione equivalente espressa come traiettoria lineare nello spazio degli angoli RPY.

A questo punto si può utilizzare la traiettoria a profilo di velocità triangolare sia per la traslazione, sia per la rotazione: si definisce il profilo d' accelerazione costante a tratti e si moltiplica per il vettore della traslazione e della rotazione, seguendo semplicemente le modalità descritte nel capitolo sulle traiettorie; il tempo totale della manovra è quindi pari a $2Ta$. Il codice MATLAB appare in questo modo:

`Parte relativa alla legge oraria`

```
    sa = 1/((Ta/200).^2);
    x=[sa*t(1:Ta/2).^0, -sa*t(Ta/2+1:2*Ta/2).^0];


Parte relativa alla conversione da asse-angolo a RPY


    R = asseangolo(asse,angolo);
    rotaz3 = atan2 ( R(2,1), R(1,1) );
    rotaz2 = atan2 ( -R(3,1), sqrt( (R(3,2))^2 + (R(3,3))^2 ) );
    rotaz1 = atan2 ( R(3,2), R(3,3));
    rotaz=[rotaz1;rotaz2;rotaz3];


Parte per l' applicazione della legge oraria al cammino geometrico


    for j = 1 : Ta

        acct(:,j) = diag( x(j) ) * trasl;
        accr(:,j) = diag( x(j) ) * rotaz;


    end
    accelerazione=[acct;accr];
```

7.4.2.9 `tilt()`

```
Variabili in input:
```

φ, l' orientamento finale che si vuole raggiungere, espresso con gli angoli di Roll Pitch e Yaw

Ta, il tempo di accelerazione nel profilo di velocità triangolare

```
Variabili in output:
```

accelerazione, l' accelerazione da imporre per realizzare la manovra richiesta

Può essere definita in un certo senso, la funzione "sorella" di `move`(), perchè funziona allo stesso modo, ma invece di prendere in ingresso la rotazione da effettuare, prende l' orientamento finale da raggiungere, e restituisce l' accelerazione adatta allo scopo. È utile soprattutto al metodo `PID`(), che effettua il controllo di stabilità restituendo in uscita un orientamento, che subito dopo essere stato calcolato, viene dato in ingresso a `tilt`(), che calcola le accelerazioni tali da farlo raggiungere alla piattaforma in un tempo $2Ta$.

7.4.2.10 `PID`()

`Variabili in input:`

corddes, la posizione desiderata nelle coordinate del piatto

veldes, la velocità desiderata nelle coordinate del piatto

`Variabili in output:`

inclin, l' uscita del controllore, cioè l' angolo che la piattaforma deve assumere

È il metodo che effettua il controllo attraverso un controllore PD. Il termine integrativo non è presente. Il metodo effettua queste operazioni:

- trasforma le coordinate della pallina nel sistema della piattaforma: infatti, essendo *corddes* e *veldes* espresse in questo sistema di riferimento, per valutare l' errore dobbiamo trovarci necessariamente nelle coordinate del piatto
- calcola l' errore per differenza

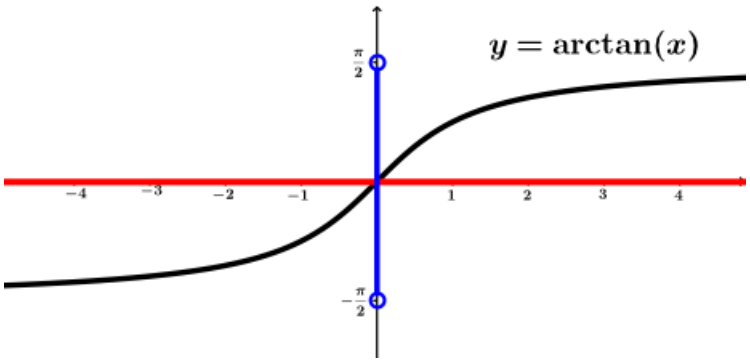

Figure 7.6: La funzione arcotangente

```
corderr = corddes - cordball;
velerr = veldes - velball;
```

- effettua un controllo PD con guadagni $Kp = 1$ e $Kd = 0.8$ separatamente per la componente x e per la y.

Un' importante osservazione è che è stata inserita la funzione arcotangente sul risultato del PD: avendo infatti in uscita al controllore un angolo espresso in radianti, sono indesiderati valori che superiori all' angolo retto, perchè implicano il ribaltamento della piattaforma. L' arcotangente è una funzione che ha per codominio l' intervallo $\left[-\frac{\pi}{2}, +\frac{\pi}{2}\right]$, e funge da limitatore anti-ribaltamento perchè non permette all' angolo in modulo di superare il valore di $\frac{\pi}{2}$.

```
inclinX = atan(Kp*corderr(1) + Kd* velerr(1));
            inclinY = atan(-Kp*corderr(2) + -Kd* velerr(2));
            inclin=[inclinY;inclinX;0];
```

È da notare inoltre che nel calcolo dell' ingresso di controllo per la Y sono stati inseriti dei segni meno ai guadagni: questo è dovuto semplicemente al modo in cui è stata programmata la grafica, per il quale, gli assi Y risultano invertiti.

7.5 La classe *pallina*

7.5.1 Gli attributi

- *raggio,* è il raggio della pallina, scelto pari a 0.1
- $grafico$, è l' handle per il grafico
- $\begin{cases} sferax \\ sferay \\ sferaz \end{cases}$, sono attributi necessari per memorizzare la sfera creata con la funzione `sphere`()

- *p,* è la posizione della pallina
- *ppunto,* è la velocità della pallina

7.5.2 I metodi

7.5.2.1 Il costruttore

`Variabili in input:`

piatt, l' oggetto *piatto* su cui giace la pallina

Come *piatto*, anche la classe *pallina* ha un suo costruttore che inizializza tutti gli attributi;

- *raggio* viene impostato a 0.1, come deciso in fase di progetto

- p viene impostato in modo che la pallina si trovi poggiata sul centro del piatto; considerando che la pallina ha un volume, il suo centro non coinciderà con il centro della piattaforma, ma sarà spostato di un valore pari al raggio, sulla normale al piatto. In formule:

$$^0p = {}^0x_{touch} + raggio \cdot \hat{n}$$

  dove con $\hat{n}$ si intende il versore normale alla piattaforma.

- *ppunto*, viene impostata solitamente nulla, ma molto spesso per testare il controllore si sceglie una velocità diversa da zero, così che la pallina venga riportata dal regolatore alla condizione di equilibrio

- $\begin{cases} sferax \\ sferay \\ sferaz \end{cases}$ vengono calcolati dalla funzione `sphere`(), e servono per rappresentare la sfera tramite la funzione surf

- $grafico$ viene inizializzato tramite la funzione `surf`()

7.5.2.2 `physics2`()

`Variabili in input:`

acc, il vettore dell' accelerazione applicata al piatto in quell' istante

vass, l' oggetto piatto su cui la pallina è situata

`Variabili in output:`

cordball, la posizione all' istante di campionamento successivo, nel sistema di riferimento della piattaforma

velball, la velocità all' istante di campionamento successivo, nel sistema di riferimento della piattaforma

È la funzione che calcola la traiettoria della pallina conoscendo lo stato del piatto: sfrutta le formule descritte nella sezione 2.3.2 dedicata alle leggi di moto della pallina, nel capitolo della dinamica Newtoniana: verrà esposto il codice e commentato:

- Viene effettuato il cambio di coordinate dal sistema di terra al sistema solidale alla piattaforma

```
I03       = [1 0 0 ; 0 1 0 ; 0 0 0];         % definizione di matrice di proiezione
vass      = obj.piatto;          % rinomino per comodità
R         = rot_elem('z' , vass.theta(3)) * rot_elem('y' , vass.theta(2))  * rot_elem('x' , vass.theta(1)); % matrice di rotazione
cordball  = R' * (obj.p-vass.x); % trasformazione di posizione nelle coordinate del piatto
r         =  R * I03 * cordball; % distanza della pallina dal centro senza considerare la sua altezza

velball   = R' * (obj.ppunto - vass.xpunto - cross(vass.omega, r ) );   % trasformazione della velocità
```

- Vengono effettuati i calcoli, imponendo nulla la terza componente della velocità, pari al raggio la terza componente della posizione

```
accparall = I03 * R' * ( acc(1:3) + g); %  vettore riga [ * * ] componenti dell' acc risultante rispetto ai 2 assi del piatto
velball   = I03*velball + accparall*Ts;        % calcolo della velocità
cordball  = [0;0;obj.raggio] + I03*cordball + I03*velball*Ts + 0.5*accparall*(Ts^2);  % calcolo della posizione
```

- Viene riportortato tutto nelle coordinate di terra, quelle in cui è definito lo stato della pallina

```
r =  R * I03 * cordball;   %    distanza della pallina dal centro senza considerare la sua altezza, aggiornata alla nuova posizione
obj.p = vass.x + R*cordball;             % trasformazione di posizione

obj.ppunto = vass.xpunto + R*velball + cross(vass.omega, r );   % trasformazione di velocità
```

7.5.2.3 `status`()

`Variabili in output:`

v, un vettore dato dalla concatenazione delle variabili di stato p, $\dot{p}$

È una funzione analoga a quella definita nella classe pallina: ritorna lo stato della pallina concatenando la sua posizione e la sua velocità: si avrà quindi

$$v = \begin{bmatrix} p_x \\ p_y \\ p_z \\ \dot{p}_x \\ \dot{p}_y \\ \dot{p}_z \end{bmatrix}$$

7.5.2.4 `disegna`()

Dalla descrizione di questa funzione verrà reso chiaro il motivo del salvataggio di $sferax$, $sferay$ e $sferaz$ tra gli attributi. Tali variabili, ottenute dalla funzione `sphere`(), contengono le coordinate (x, y, z) dei punti da unire, per mezzo della funzione `surf`(), per ottenere la sfera.

Di default la funzione `sphere`() restituisce le matrici per una sfera di raggio unitario centrata nell' origine. Per variare la posizione e la dimensione della sfera è stato usato il seguente metodo: dapprima vengono moltiplicate le matrici $sferax$, $sferay$ e $sferaz$ per il raggio. In questo modo, trovandosi la sfera centrata nell' origine, si otterrà una contrazione omogenea su tutte le direzioni della superficie, che manterrà quindi senza distorsione la sua forma sferica. Dopo aver effettuato questa operazione si somma il valore della coordinata posizionale della pallina a tutti gli elementi della matrice corrispondente, quindi p_x agli elementi di $sferax$ e così via. In questo modo si ottiene una traslazione uguale per tutti i punti della sfera, che sarà infine centrata nella sua posizione p. È importante non invertire l' ordine delle operazioni: se si effettua prima la traslazione e poi

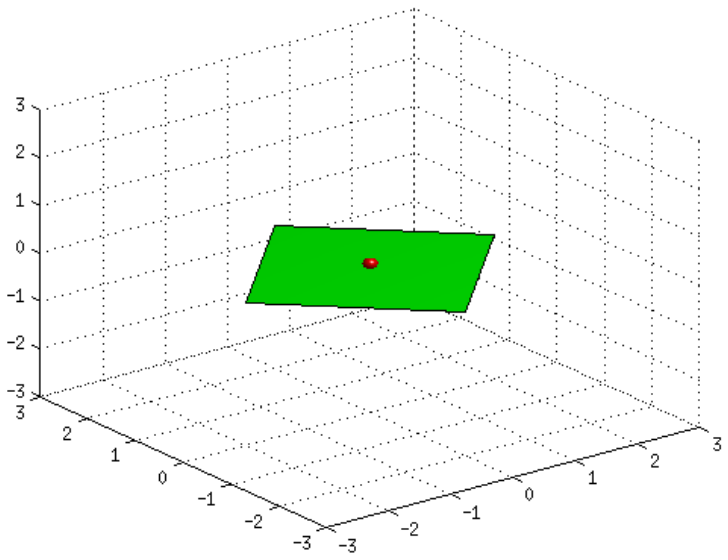

Figure 7.7: L' animazione della pallina sul piatto

il prodotto, non essendo più la sfera centrata nell' origine, la superficie subirà una deformazione.

Il codice usato è

```
set(obj.grafico,'x',obj.sferax*obj.raggio + obj.p(1));
set(obj.grafico,'y',obj.sferay*obj.raggio + obj.p(2));
set(obj.grafico,'z',obj.sferaz*obj.raggio + obj.p(3));
```

7.6 La classe *sistema*

L' ultima classe da descrivere è quella che rappresenta il sistema piatto-pallina. Analizziamo attributi e i metodi:

7.6.1 Gli attributi

- *touch*, è l' oggetto *piatto* del sistema piatto-pallina
- *ball*, è l' oggetto *pallina* del sistema piatto-pallina
- *jacA*, è lo Jacobiano $\frac{\partial f}{\partial \mathbf{q}}$ rispetto allo stato, equivalente alla matrice A_{lin} del sistema linearizzato
- *jacB*, è lo Jacobiano $\frac{\partial f}{\partial \mathbf{u}}$ rispetto all' ingresso, equivalente alla matrice B_{lin} del sistema linearizzato
- T, è la matrice del cambio di coordinate dalla realizzazione di simulazione alla realizzazione di controllo
- *jacerrA*, è lo Jacobiano $\frac{\partial \tilde{f}}{\partial \tilde{\mathbf{q}}}$ rispetto allo stato del sistema errore, equivalente alla matrice $\tilde{A}_{lin}$ del sistema linearizzato
- *jacerrB*, è lo Jacobiano $\frac{\partial \tilde{f}}{\partial \mathbf{u}}$ rispetto all ingresso del sistema errore, equivalente alla matrice $\tilde{B}_{lin}$ del sistema linearizzato
- *state*, è lo stato $\mathbf{q}$ del sistema
- *error*, è l' errore $\tilde{\mathbf{q}}$ del sistema

7.6.2 I metodi

7.6.2.1 Il costruttore

`Variabili in input:`

t, l' oggetto *piatto* del sistema

b, l' oggetto *pallina* del sistema

Come ogni classe, anche *sistema* ha il suo costruttore: vengono dapprima create tutte le variabili necessarie a inizializzare alcuni attributi come le Jacobiane: viene creato il proiettore I_{03}

$$I_{03} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

e la variabile Mg che corrisponde alla derivata parziale del proiettore, $M(\cdot)$ valutata per l' accelerazione gravitazionale g.

$$M(g) = \begin{bmatrix} 0 & 9.80665 & 0 \\ -9.80665 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

A questo punto si inizializzano gli attibuti in questo modo:

- *touch*, viene inizializzato all' istanza t passata in ingresso
- *ball*, viene inizializzata all' istanza b passata in ingresso
- *jacA*, viene inizializzato allo Jacobiano calcolato nel capitolo della linearizzazione, definito a blocchi in formula (5.28)

$$\frac{\partial f}{\partial \mathbf{q}} = \begin{bmatrix} I & Ts & O & O & O & O \\ O & I & O & O & O & O \\ O & O & I & Ts & O & O \\ O & O & O & I & O & O \\ I - I_{03} & -TsI_{03} & \frac{Ts^2}{2}Mg & O & I_{03} & TsI_{03} \\ O & I - I_{03} & TsMg & O & O & I_{03} \end{bmatrix}$$

- *jacB*, allo stesso modo, viene inizializzato come in formula (5.31)

$$\frac{\partial f}{\partial \mathbf{u}} = \begin{bmatrix} \frac{Ts^2}{2} & O \\ Ts & O \\ O & \frac{Ts^2}{2} \\ O & Ts \\ I_{03}\frac{Ts^2}{2} & O \\ I_{03}Ts & O \end{bmatrix}$$

- T, viene inizializzata come in formula (5.3)

$$T = \begin{bmatrix} I & O & O & O & O & O \\ O & I & O & O & O & O \\ O & O & I & O & O & O \\ O & O & O & I & O & O \\ -I & O & O & O & I & O \\ O & -I & O & O & O & I \end{bmatrix}$$

- $jacerrA$ viene inizializzato tramite trasformazione di coordinate di $jacA$ descritte nelle (5.33)

$$\frac{\partial \tilde{f}}{\partial \tilde{\mathbf{q}}} = T \frac{\partial f}{\partial \mathbf{q}} T^{-1}$$

- $jacerrB$ viene inizializzato secondo le stesse trasformazioni in (5.33)

$$\frac{\partial \tilde{f}}{\partial \mathbf{u}} = T \frac{\partial f}{\partial \mathbf{u}}$$

- *state*, come di norma viene definito come concatenazione dello stato della piattaforma e dello stato della pallina

$$\mathbf{q} = \begin{bmatrix} \mathbf{q}_{touch} \\ \mathbf{q}_{ball} \end{bmatrix}$$

- *error*, viene definito trasformando *state* attraverso la matrice T

$$\tilde{\mathbf{q}} = T\mathbf{q}$$

7.6.2.2 `status`()

`Variabili in output:`

s, lo stato del sistema

Anche qui abbiamo questa funzione. La funzione aggiorna l' attributo *state* prelevandolo da *touch* e *ball*, e lo restituisce in uscita nella forma concatenata già vista.

7.6.2.3 `disegna`()

È il metodo che disegna il sistema nel suo stato attuale. Non fa altro che richiamare il metodo `disegna`() del suo attributo *piatto*, che a sua volta richiama al suo interno il metodo analogo del suo attributo *pallina*.

7.6.2.4 `simulation`()

`Variabili in input:`

acc, l' accelerazione a cui è sottoposta la piattaforma

`Variabili in output:`

stato, lo stato del sistema calcolato

errore, l' errore di stato del sistema calcolato

Per una simulazione realistica è opportuno simulare il sistema originale non lineare, non quello linearizzato. I passaggi eseguiti sono i seguenti:

- Viene richiamato il metodo `acceleration`(`acc`) della classe piatto, passandogli *acc* in ingresso; `acceleration`(`acc`) a sua volta richiama `physics2`(`acc`) della pallina.

```
obj.touch.acceleration(acc); % faccio la simulazione non lineare
```

- Il nuovo stato è adesso memorizzato nello stato del piatto e della pallina, ma non in quello del sistema. Per questo motivo si richiama la funzione `status`() per aggiornare anche quello del sistema. Si aggiorna anche l' errore, ottenuto dallo stato tramite l' endomorfismo T.

```
obj.status;
stato = obj.state;
obj.error = obj.T * stato;
errore = obj.error;
```

7.6.2.5 `control`()

`Variabili in output:`

K, la matrice di retroazione dello stato K ottima

Il metodo definisce le 5 matrici da passare alla funzione `dlqr`(), la richiama, e restituisce in uscita la matrice ottima dei guadagni K calcolata dalla suddetta funzione tramite le equazioni di Riccati. Le matrici sono:

- A è lo Jacobiano di errore jacerrA
- B è lo Jacobiano di errore jacerrB
- Q è la matrice di peso per la norma dello stato
- R è la matrice di peso per la norma dell' ingresso
- N è la matrice di peso per la norma incrociata stato-ingresso

È importante notare che le matrici A e B che vengono passate sono quelle della realizzazione di controllo, non di simulazione, quindi sono riferite al sistema errore.

Le matrici di peso vengono inizializzate, come prestabilito, ai valori:

```
Q=diag([500 500 500, 2 2 2 , 3 3 3, 5 5 5, 5000 5000 5000, 5 5 5]);
R=0.1*eye(6);
N=zeros(18,6);
```

7.7 I main

Esitono due patch pricipali: il *main* e il *mainPD*; il primo impartisce i comandi necessari per effettuare la simulazione con controllo LQR, e il secondo con controllo PD. Le due patch hanno la prima parte in comune, nella quale avviene la dichiarazione e l' inizializzazione di tutte le variabili cardine descritte nella sezione 7.2, e la creazione di un oggetto piatto passando al costruttore la matrice *vassoioiniziale*.

```
v = piatto(vassoioiniziale);
```

Da questo punto in poi le due patch sono completamente diverse l' una dall' altra.

7.7.1 Parte del controllo LQR

La prima cosa da fare è creare un oggetto *sistema*: lo si farà passandogli il piatto già creato, e la pallina che il costruttore del piatto a sua volta crea. Poi si procede con il calcolo matrice di controreazione K, che rimarrà costante per tutta la durata della simulazione. Viene inizializzato l' ingresso all' istante iniziale, sempre pari a $,-Ke$, e la variabile $film$ allo stato iniziale.

```
sss=sistema(v,v.ball);  % creo il sistema
K=sss.control;          % calcolo la matrice K di controreazione
acc = -K * sss.error;   % inizializzo l' accelerazione
film = sss.status;      % inizializzo allo stato iniziale
```

A questo punto abbiamo una serie di istruzioni da ripetere ad ogni passo di campionamento, insierite per questo motivo in un ciclo `for`. Le istruzioni sono già opportunamente commentate e intuitive da comprendere quindi si eviterà si spiegarle una seconda volta. Il risultato finale che si avrà da questo ciclo `for` è la visualizzazione dell' animazione del controllo.

```
for i = 1 : length(t)


    ACCEL(:,i) = ACCEL(:,i) + acc;   % scrivo l' accelerazione su ACCEL
    [stato , errore] = sss.simulation(ACCEL(:,i));  %  effettuo la simulazione
    acc = -K * errore;   %  aggiorno l' ingresso per il ciclo successivo
    film = [ film , stato ]; % memorizzo lo stato attuale
    figure(1);
    sss.disegna; % aggiorno l' animazione
    pause(Ts);


end
```

Successivamente c'è la parte della visualizzazione dei grafici, possibile grazie all' estrazione dei dati cercati dalle matrici $film$ e $ACCEL$. Poichè lo spazio di stato è formato da sei variabili *tridimensionali*, rappresentare l' andamento nel tempo di una di queste richiederebbe un grafico a quattro dimensioni, impossibile da realizzare e da comprendere. Si è scelto quindi di rappresentare l' andamento nel tempo della norma Euclidea, o norma 2, di ogni variabile, definita per un generico vettore tridimensionale v come

$$\left\| v \right\|_2 = \sqrt{v_x^2 + v_y^2 + v_z^2}$$

Da qui si tracciano attraverso il comando `subplot`() tutti i grafici che sono stati mostrati alla fine del capitolo in gruppi di due per ogni finestra.

7.7.2 Parte del controllo PID

La prima cosa da fare è inizializzare la posizione e la velocità desiderata attraverso

```
corddes = [0, 0, v.ball.raggio]'; % NELLE COORD DEL PIATTO
veldes = [0 0 0]'; % NELLE COORD DEL PIATTO
```

e successivamente avviare un ciclo `for` per effettuare i calcoli ad ogni istante di campionamento, di cui alleghiamo il codice commentato:

```
for i = 1 : length(t)

        if mod(i,Ta) == 1  % effettua queste operazioni ogni Ta tempi di campionamento

              angolo=v.PID(corddes,veldes); % calcola l' angolo che deve assumere il piatto

            acc=v.tilt(angolo,Ta); % calcola l' accelerazione necessaria per portarli in quell' angolo
            ACCEL (: , i:i+Ta-1) = ACCEL (: , i:i+Ta-1) + acc; % la aggiunge alla matrice delle accelerazioni
        end
        [cordball, velball ] = v.acceleration( ACCEL(:,i) ); % simula l' accelerazione trovata
        figure(1);
        v.disegna; % aggiorna il grafico
        pause(Ts)


end
```

Una cosa molto importante da notare in questo algoritmo è il fatto che le operazioni di simulazione sono effettuate in ogni singolo istante di campionamento, mentre le operazioni di controllo ogni `Ta` tempi di campionamento: il controllore PD infatti restituisce l' orientamento che deve assumere la piattaforma, che lo raggiungerà con un profilo di velocità triangolare, in un tempo `Ta`. Nel lasso di tempo in cui la piattaforma è in movimento verso la nuova angolazione, il controllo non agirà, attendendo che questa sia arrivata a destinazione: per questo motivo, la funzione PID() va richiamata ogni `Ta` istanti di campionamento.

Al termine della simulazione vengono visualizzati i grafici, raccolti nella variabile $filmPD$ con le stesse modalità viste per la parte relativa al controllo LQR.

Bibliography

[1] *L. Sciavicco B.Siciliano:* ***Robotica Industriale*** *- Modellistica e Controllo di manipolatori*

[2] *C. Manes: Appunti di Robotica Industriale*

[3] *G. Di Stefano: Slides del corso di Programmazione ad Oggetti*

[4] *Mathworks®: Matlab® Official Documentation*

[5] *Wikipedia the free Encyclopedia*