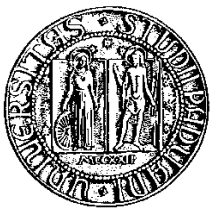


Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

tesi di laurea

Una guida per l'utilizzazione educativa del robot Mindstorms NXT con programmazione Java

la componente hardware

Relatore: Michele Moro

Laureando: Massimo Perotto

26 aprile 2010

Indice

Elenco delle tabelle

Elenco delle figure

Introduzione

Questa tesi nasce con l'obbiettivo di creare un corso per studenti delle classi medie superiori che dia ad essi le basi per la programmazione del LEGO ® MINDSTORMS ® NXT mediante il linguaggio Java. Tale necessità deriva dal fatto che il linguaggio NXT-G, con cui viene fornito il robot, ha dei limiti tra cui la gestione dei numeri nel solo formato intero e l'assenza di possibilità di "esecuzione parallela" di parti di codice. Oltre a ciò Java è un linguaggio molto flessibile, orientato agli oggetti e di larga diffusione su cui gli studenti già lavorano essendo il linguaggio trattato nelle lezioni di informatica.
Anche se gli studenti hanno già lavorato sugli NXT, con il linguaggio NXT-G, verranno comunque ripresentate le caratteristiche di queste macchine. È stata effettuata tale scelta perché l'elaborato sia di utilità, oltre che per gli studenti, anche per qualsiasi persona, come hobbisti o docenti, che, con un minimo di conoscenza nella programmazione, decida di avvicinarsi a questo ambito. Si cercherà quindi di creare un insieme di informazioni coerenti tra loro al fine di velocizzare e facilitare l'apprendimento.
Il lavoro è stato diviso in due parti. In questa si tratteranno argomenti come le caratteristiche hardware dell'NXT, dei motori e dei sensori disponibili. Per quanto riguarda gli ultimi due aspetti si cercherà di illustrare il loro funzionamento da dati ricavati sperimentalmente, in modo che l'utente di questo elaborato sia facilitato a desumerne il comportamento. Si cercherà di spiegare le argomentazioni in maniera semplificata, adatta ad un pubblico che si affaccia per la prima volta in quest'ambito. Essendo comunque un elaborato destinato, in primo luogo, agli studenti, si realizzeranno attività di laboratorio atte a legare la robotica con nozioni di matematica e fisica come integrazione, derivazione, spazio, velocità, accelerazione, forze, momenti, lavoro e potenza.

Capitolo 1

Hardware del brick

Come già affermato nell'introduzione questo corso è ideato per chi ha già avuto modo di lavorare su un NXT, ma anche per le persone che si affacciano per la prima volta nell'ambito della robotica. In questo capitolo intendiamo spiegare in dettaglio "cosa c'è all'interno" del brick in modo da gettare le basi per meglio comprendere le sue potenzialità, la sua programmazione e come vengono gestiti i sensori ed i motori.

1.1 Caratteristiche principali

Con la sigle NXT si intende un kit di montaggio sostanzialmente compatibile con la linea LEGO TECHNIC ma finalizzato alla costruzione di robot. Il suo elemento principale, detto brick in riferimento alla sua forma a mattone, è un piccolo elaboratore. All'esterno di esso balzano subito all'occhio i fori adatti all'assemblaggio con i classici componenti LEGO, il display, i pulsanti e i connettori per sensori e motori. Ma all'interno cosa vi è contenuto? Elenchiamo in breve le principali componenti del brick:

- **Processore**[1]**:** Atmel ® a 32 bit processore ARM7 ®, AT91SAM7S256;
- **Co-processore**[2]**:** Atmel ® 8-bit AVR modello ATMEGA48;
- **Bluetooth**[3]**:** CSR BlueCoreTM 4 versione 2.0;
- **Porta di comunicazione USB 2.0:** con velocità di trasmissione di 12 Mbit/s. Questa porta è utilizzata per l'upload dei nostri programmi all'interno del brick, scaricare file (ad esempio i risultati di un'elaborazione), oppure aggiornare il firmware;

[1] Per le caratteristiche dettagliate si veda la sezione 1.2
[2] Per le caratteristiche dettagliate si veda la sezione 1.2
[3] Per le caratteristiche dettagliate si veda la sezione 1.5

Figura 1.1: Scheda di un NXT

- **4 porte di ingresso**[4]**:** con connettore RJ12 a 6 fili;
- **3 porte di uscita**[5]**:** con connettore RJ12 a 6 fili;
- **Display** 100 x 64 pixel, LCD in bianco e nero, con area visualizzabile pari a 26 X 40,6 millimetri. La comunicazione tra l'ARM7 e il dispositivo è ottenuta tramite una connessione SPI[6] con frequenza pari a 2 MHz. Per il totale update del display sono necessari 17ms;
- **Altoparlante**: diametro pari a 21 mm e resistenza di 16 Ohm. Il canale di uscita audio ha una risoluzione di 8-bit e una frequenza di campionamento compresa tra 2 e 16 KHz. Il segnale audio è riprodotto dall'ARM7, è amplificato da un amplificatore operazionale con guadagno pari a 20 e inviato all'altoparlante;
- **4 bottoni** per la navigazione del menù da parte dell'utente e il cui stato è gestibile in programmazione;
- **Alimentazione:** tramite 6 batterie AA di tipo alcalino, oppure è disponibile una batteria ricaricabile Li-Ion con una tensione di 7.2V e 1400 mA/h;

[4]Per informazione sulla loro gestione si veda la sezione 1.4
[5]Per informazione sulla loro gestione si veda la sezione 1.3
[6]Per le caratteristiche dettagliate si veda la sezione 1.2

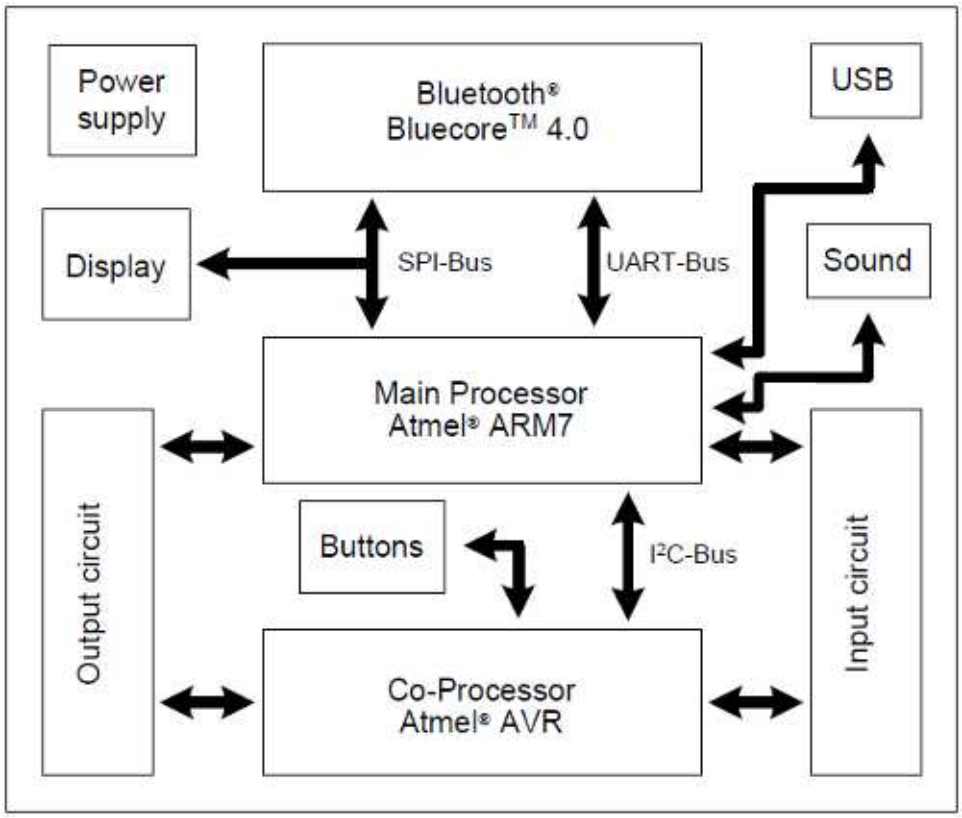


Figura 1.2: Diagramma a blocchi delle componenti di un NXT

1.2 Processore e Co-Processore

Il microcontrollore è un chip a basso consumo basato sul ARM7TDMI, processore a 32 bit di tipologia RISC (Reduced Instruction Set Computer, cioè basato su un set di istruzioni semplici il cui tempo di esecuzione è simile). È dotato di 256 Kbyte di Flash interna su cui è possibile salvare i file eseguibili, mentre 64 Kbyte di SRAM sono utilizzati come memoria volatile per i programmi in esecuzione.
Nell'insieme dei dispositivi hardware e delle funzioni implementate si annovera:

- **SAM-BA** (SAM Boot Assistant) standard che permette la programmazione diretta della Flash interna per mezzo di USB, RS232 o Bluetooth.

- **USB 2.0** per la connessione col pc;

- **UART** (Universal Asynchronous Receiver-Transmitter) per la conversione di flussi bit da parallelo a seriale asincrono e viceversa;

- **SPI** (Serial Peripheral Interface) per la comunicazione seriale di tipo sincrona, cioè con clock, master/slave e full duplex, ossia contemporaneamente in invio e ricezione;
- **SSC** (Synchronous Serial Controller) per la gestione della comunicazione seriale su più canali;
- **TWI** (Two Wire Interface) bus per la comunicazione I^2C dei sensori digitali [7];
- **Quantizzatore ADC a 10-bit**: periferica che permette di digitalizzare un segnale analogico trasformandone l'informazione in un flusso di bit. Si possono identificare 1024 livelli diversi grazie ai 10 bit del dispositivo;
- **Controller DMA** (Direct Memory Access, accesso diretto alla memoria), dispositivo utilizzato per evitare di far gestire al processore il trasferimento di dati tra le periferiche, o con la memoria, evitando così colli di bottiglia e rallentamenti nell'elaborazione;
- **Gestione degli interrupt**, cioè delle richieste asincrone pervenute dalle periferiche che indicano il presentarsi di un evento, come ad esempio la ricezione di dati da un dispositivo;
- **Timer**;
- **Debug** tramite l'interfaccia JTAG (Joint Test Action Group) che prevede, su alcuni pin predefiniti, la possibilità di passare in modalità debug per l'invio di comandi standard al chip al fine di testare eventuali anomalie;

Nello sviluppo di questo chip è stato previsto, al suo interno, l'inserimento di un gran numero di periferiche e funzionalità, con lo scopo di ridurre significativamente il numero di chip esterni eventualmente necessari per implementare questi controlli e, quindi, riducendo al minimo il consumo di energia.
A temperatura standard la massima frequenza di clock è di 55MHz, ma nell'NXT essa è impostata a 48MHz. La tensione di base tipica del core è pari a 1.8V mentre ai pin I/O, grazie al regolatore di tensione integrato, sono forniti 1.8, 3.3 o 5 Volt.
L'AT91SAM7S25 ha 64 pin ed è fornito in due tipi di package diversi:

- **LQFP** (Low-profile Quad Flat Package) formato simile alla maggioranza dei chip che si possono vedere su una scheda madre (chip audio, north-bridge, ecc);
- **QFN** (Quad Flat No Leads) senza piedini esterni (simile ai processori di ultima generazione).

Nell'NXT il formato di questo chip è LQFP visibile nella figura 1.1 in basso a sinistra e più in dettaglio nella figura 1.3.

[7] Per le caratteristiche dettagliate si veda la sottosezione 1.4.2

Figura 1.3: Una foto esemplificativa del package dell'ARM7

Il Coprocessore, di tipo RISC, è utilizzato per la gestione dell'input e output. Esso ha una frequenza di clock pari a 8 MHz ed è dotato di 4 KB di FLASH per il salvataggio delle istruzioni e dei dati (attenzione, l'utente non agisce su queste istruzioni) e 512 Byte SRAM. I compiti principali di questo microcontrollore all'interno dell'NXT sono:

- la gestione dell'alimentazione delle periferiche;
- la creazione dei segnali PWM per i tre motori [8];
- eseguire la conversione dei segnali analogici ricevuti dalla porte in ingresso e inviare i dati in formato digitale al processore principale.

La comunicazione tra i due processori avviene per mezzo dell'interfaccia I^2C [9]. Per garantire l'indipendenza nell'esecuzione dei due chip, il trasferimento dei dati tra di essi, è realizzato mediante due aree di memoria, utilizzate come buffer di invio e ricezione, che vengono aggiornate scambiando le informazioni ogni 2ms mediante un bus I^2C.

[8] Per chiarimenti si veda la sezione 1.3
[9] Per le caratteristiche dettagliate si veda la sottosezione 1.4.2

1.3 Porte di Uscita

Il LEGO ® MINDSTORMS ® NXT è dotato di tre porte utilizzate come output per controllare altrettanti motori. Esse hanno un'interfaccia digitale creata in modo che i dispositivi di output possano inviare informazioni all'NXT, senza dover ricorrere a una porta di ingresso. La figura 1.4 esemplifica lo schema della porta di uscita A. Le porte B e C differiscono solo per il nome delle linee, non per la loro funzione (in es MA0→MB0, ecc)

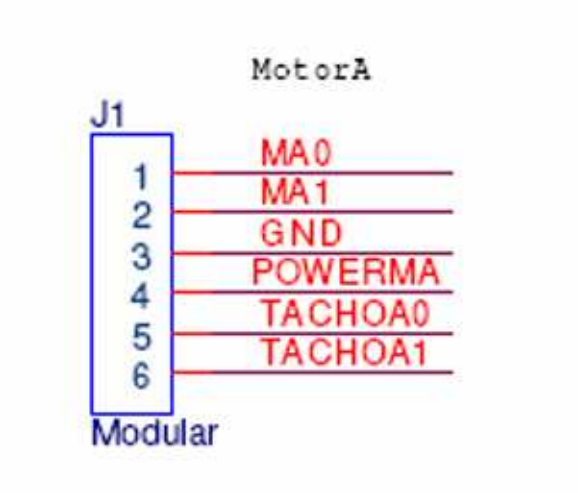


Figura 1.4: Il connettore di una porta di uscita e il significato dei suoi pin

I pin **MA0** e **MA1** controllano il motore. Il segnale inviato è modulato in PWM (pulse-width modulation) cioè modulazione a larghezza di impulso. Questa modulazione si basa sulla durata dell'impulso tra un clock e l'altro, cioè al variare del duty cycle. Se si invia un impulso pari allo 0% del periodo il motore è fermo. Se l'impulso è pari a 100% del periodo il motore è al massimo. Si utilizza questo sistema per limitare l'assorbimento di potenza, inviando informazioni impulsive invece di un segnale costante di ampiezza proporzionale alla potenza che si vuole impostare al motore. Il segnale in questione può avere un amperaggio approssimato di 700mA, con duty cycle pari al 100%, e picchi di 1A. Il trasduttore, nel motore, converte quest'informazione in potenza da fornire al motore. Se dal brick, per un determinato lasso di tempo, perviene un amperaggio superiore al massimo consentito, subentra la protezione termica del motore.

Il pin **POWERMA**, internamente al brick, è collegato in comune a tutte le porte di input e output. È utilizzato, nelle porte di output, per alimentare il circuito di controllo nel motore. La corrente massima erogata da questa linea è pari a 180mA ma, solitamente, i valori si aggirano intorno ai 20mA. Se la corrente in questo circuito au-

menta, l'NXT la limita automaticamente. Se questa linea viene cortocircuitata con il pin ground (GND) il brick si resetta.

I pin **TACHOA0** e **TACHOA1** sono utilizzati in input e connessi all'ARM7 tramite un trigger di Schmith. Questo circuito permette di trasformare un segnale analogico in un'uscita che vari soltanto tra due valori di tensione, a seconda che l'ingresso superi o no una certa soglia preimpostata. Il trigger di Schmith è quindi qui utilizzato per squadrare gli impulsi provenienti dai due pin, TACHOA0 e TACHOA1, tra i valori di tensione riconosciuti dall'ARM7 come 0 e 1. Ogni impulso rappresenta la variazione di un grado del motore[10]. Siccome il segnale proveniente da questa linea non è sincronizzato col clock di sistema, cioè è asincrono, un meccanismo di sincronizzazione previene la perdita dei dati.

1.4 Porte di Ingresso

Il LEGO ® MINDSTORMS ® NXT dispone di 4 porte in ingresso per acquisire i valori misurati dai sensori. Queste porte sono fornite di interfaccia analogica e digitale per poter supportare entrambi i tipi di sensore. La figura 1.5 esemplifica lo schema della porta di ingresso 1. Le porte 2, 3 e 4 differiscono solo per il nome delle linee, non per la loro funzione (in es ANA→ANB, ecc)

Il pin **ANA** è la linea per l'ingresso analogico ed è connessa al quantizzatore a 10 bit ADC del co-processore AVR. Oltre che ad esso, questo pin è connesso ad un generatore identificato col pin VCC, che fornisce una tensione pari a 5V. Esso è necessario per creare l'impulso per l'attivazione dei "sensori analogici[11]".

Il pin **IPOWERA** è collegato con tutte le altre uscite di potenza delle porte di ingresso e uscita. Per le porte di uscita, come visto nel paragrafo precedente, questo pin prende il nome di POWERMA e, quindi, l'IPOWERA ne presenta le medesime caratteristiche che non ripeteremo.

I pin **DIGIAI0** e **DIGIAI1** sono utilizzati per la comunicazione digitale con interfaccia I^2C [12].

[10] Per le caratteristiche dettagliate si veda il capitolo 2.1.2
[11] Per le caratteristiche dettagliate si veda la sottosezione 1.4.1
[12] Per le caratteristiche dettagliate si veda la sottosezione 1.4.2

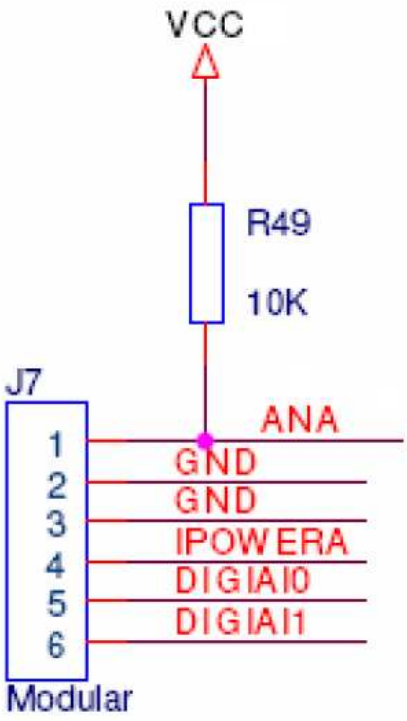


Figura 1.5: Il connettore di una porta di ingresso e il significato dei suoi pin

1.4.1 Connessione coi sensori

I sensori verranno trattati in maniera approfondita nel capitolo 4 a loro dedicato. In questa sezione si intende evidenziare in che modo le diverse tipologie di sensori vengono trattate dall'NXT. Bisogna infatti specificare che i sensori si dividono in:

- **Analogici Attivi:** Di questa categoria fanno parte il Light Sensor ed il Gyro Sensor. Questi sensori sono detti attivi in quanto per effettuare la misura di un fenomeno fisico hanno la necessità di emettere un'interrogazione misurandone la risposta. Essi hanno perciò bisogno di tempi di misurazione ben precisi e devono per questo essere sincronizzati con il brick. Essendo sensori analogici il valore del dato, che essi forniscono all'NXT, deve essere digitalizzato e questo avviene tramite il quantizzatore interno al co-processore AVR. Il problema è quindi la sincronizzazione dei dati. Per far ciò il pin ANA, come precedentemente detto, è fornito di una differenza di potenziale pari a 5V e circa 18mA. Questa potenza è erogata per 2.9ms e, trascorso questo periodo, il pin VCC viene posto a massa per i successivi 0.1ms, tempo in cui il brick acquisisce il valore da digitalizzare. Si ha così un tempo di campionamento del segnale analogico pari a 3ms e, perciò,

una freqenza di campionamento di circa 333 Hz. I dati dei sensori attivi vengono quindi acquisiti in maniera sincrona.

- **Analogici Passivi:** Sono i sensori analogici che non hanno bisogno di interrogare l'ambiente circostante per l'acquisizione dei dati, ma traducono direttamente il fenomeno fisico in un segnale elettrico. Fanno parte di questa categoria il Touch Sensor, il Light Sensor, il Sound Sensor e il Temperature Sensor. Questi sensori non necessitano di sincronizzazione ma, essendo unico il quantizzatore che gestisce i sensori attivi e passivi, è necessario porre anche a questi ultimi un tempo di campionamento pari a 3ms. Questa imposizione è necessaria per evitare interferenze tra le acquisizioni dei dati. Le acquisizioni sono infatti sincronizzate tra le 4 porte che utilizzano un unico quantizzatore in comune.

- **Digitali:** Tutti i sensori che utilizzano l'interfaccia di comunicazione I^2C[13] vengono denominati digitali in quanto, al loro interno, vi è un controller e un quantizzatore che si occupa direttamente della digitalizzazione dell'informazione, senza quindi dover utilizzare l'ADC interno al brick. Fanno parte di questa categoria l'Ultrasonic Sensor e l'Accelerometer Sensor.

1.4.2 La connessione I^2C

Nell'NXT l'interfaccia digitale è implementata mediante il protocollo I^2C acronimo di Inter Integrated Circuit. Questo protocollo seriale è stato creato nel 1982 dalla Philips come standard di comunicazione industriale e, da allora, largamente diffuso e utilizzato in molti componenti in cui è richiesta una semplice comunicazione digitale. Esso permette una velocità di trasmissione pari a 9600 bit/s.
Nell'NXT questo protocollo è utilizzato per la comunicazione con i dispositivi digitali, consentendo ad essi di svolgere un funzionamento indipendente e garantendo la connessione da e verso l'NXT. Il brick è fornito di 4 canali I^2C, uno per ogni porta input, che funzionano in modalità master, è cioè l'NXT che controlla il flusso dei dati dei canali di comunicazione ed emette il segnale di clock su cui i dispositivi si sincronizzano. Gli scenari che potranno quindi verificarsi sono: l'NXT controlla il clock e trasmette a uno slave, l'NXT controlla il clock e riceve da uno slave, uno slave non gestisce il clock ma si sincronizza su di esso per trasmettere, uno slave non gestisce il clock ma si sincronizza su di esso per ricevere.
Nei paragrafi precedenti abbiamo identificato con DIGIXI0 e DIGIXI1 i pin 5 e 6 della porta `X` (co `X`=A,B,C,D), senza però specificare altro tranne il loro utilizzo nella comunicazione digitale. Puntualizziamo ora che la linea DIGIXI0 fornisce il clock alla connessione e DIGIXI1 trasferisce i dati. La comunicazione ha inizio con una transizione da livello alto a basso del bus dati mentre il bus di clock è a livello alto. Essa

[13] Per le caratteristiche dettagliate si veda la sottosezione 1.4.2

termina con una transizione da basso ad alto livello del bus dati mentre il bus di clock a livello alto. Un acknowledgement, che possiamo definire come segnale di conferma, avviene forzando a livello basso la linea dati. C'è da precisare che il protocollo I^2C è stato realizzato per la comunicazione di più dispositivi su un unico bus. Per questo i dispositivi vengono identificati mediante un indirizzo di 7bit che viene inviato subito dopo il bit start e subito prima del bit che indica se la connessione deve essere in ricezione o in trasmissione. Si vuole far notare questo per precisare che è teoricamente possibile connettere più sensori digitali ad una sola porta, con l'accortezza che i loro indirizzi siano diversi. LEGO ha per questo creato una lista di indirizzi prenotabili per le aziende che sviluppano sensori, in modo da non avere duplicazioni di indirizzo. Ad esempio i sensor ad ultrasuoni LEGO, in dotazione col kit, hanno indirizzo 2. Per la comunicazione ogni canale è dotato di un buffer di input e uno di output, entrambi dalla dimensione di 16 byte, quindi, per ogni scambio di dati, si creano flussi di dati per un massimo di 16 byte. Come detto in precedenza un sensore digitale è un'apparecchiatura che ha un funzionamento autonomo indipendente dal brick. L'NXT infatti si occupa solo di trasmettere le richieste al sensore e acquisire le informazioni inviate da quest'ultimo. Per effettuare ciò i dati vengono salvati in apposite aree di memoria, appartenenti al dispositivo periferico, in cui l'NXT può leggere o scrivere. Nella tabella 1.1 si possono osservare alcuni comandi standard utilizzati sui sensori digitali. I byte che seguono identificano la trasmissione da brick a sensore in esadecimale. Il byte 0 è il primo inviato e rappresenta l'indirizzo della periferica, il secondo l'area di memoria utilizzata ed il terzo il comando. Quest'ultimo in caso di richiesta in lettura è pari a 0x03.

Comandi	**Byte 0**	**Byte 1**	**Byte 2**
Leggi la versione	Indirizzo	0x00	0x03
Leggi l'ID di produzione	Indirizzo	0x08	0x03
Leggi il tipo di Sensore	Indirizzo	0x10	0x03
Leggi il valore di zero (di fabbrica)	Indirizzo	0x18	0x03
Leggi il fattore di scala (di fabbrica)	Indirizzo	0x19	0x03
Leggi il divisore di scala (di fabbrica)	Indirizzo	0x1A	0x03
Leggi l'unità di misura	Indirizzo	0x1B	0x03
Leggi la variabile X	Indirizzo	0x40	0x03
Invia il comando X	Indirizzo	0x80	Comando

Tabella 1.1: Esempio di comandi inviati ad un sensore digitale

1.4.3 La porta RS485

La porta di ingresso 4 differisce dalle altre porte: infatti, pur mantenendo le caratteristiche già citate, essa può funzionare come porta ad alta velocità con standard RS485. Questa soluzione permette l'utilizzo dei pin DIGIDI0 e DIGIDI1, oltre che con la normale comunicazione I^2C, come bus bi-direzionale per la comunicazione a lunga distanza con una velocità teorica di 921.6 Kbit/s. Momentaneamente non esistono sensori che utilizzino questa soluzione, che è stata inserita in progettazione hardware per un eventuale sviluppo futuro. Questa porta implementa il protocollo di rete P-Net, rivolto alla creazione di reti a bus condiviso, che gestiscono un insieme di sensori slave da parte di un'unità master, che in questo caso è rappresentata dall'NXT.

1.5 Bluetooth

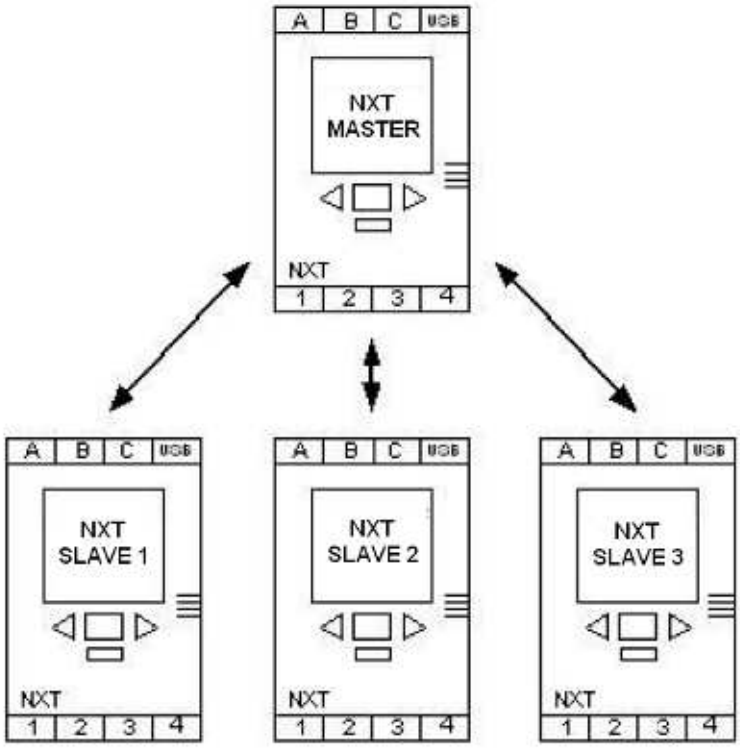


Figura 1.6: Trasmissione tra unità master e slave

L'NXT supporta la connessione Bluetooth grazie al chip della CSR modello BlueCore 4 versione 2.0 con il quale può effettuare fino a tre connessioni contemporaneamente, non consentendo però l'invio e la ricezione contemporanea tra esse. Il chip sostiene la funzione Serial Port Profile (SPP) per la sincronizzazione dei dati potendo, con essa, gestire la porta wireless come una normale porta seriale. I profili supportati da questo standard sono: SPP-A, utilizzato quando il dispositivo è in modalità master, e SPP-B, quando è in slave. La modalità di trasmissione quindi è del tipo master/slave. Ciò significa che, nella rete di connessione creata, un brick avrà la funzione di master e tutti gli altri comunicheranno con lui. Non è possibile creare un'unità che sia allo stesso tempo master e slave, questo per evitare eventuali perdite di dati. Infatti, quando un'unità master comunica con una slave, se una terza unità cerca di inviare dati al master viene ignorata finché la prima richiesta non viene esaudita. Nella rete quindi, tutte le unità slave potranno comunicare solamente con l'unità master e non tra di loro. L'unità master fa quindi da ponte per la comunicazioni tra due unità slave.
Più in dettaglio l'hardware Bluethooth è un chip a 16 bit che lavora ad una frequenza di clock pari a 26 MHz. Ha al suo interno installata una memoria RAM pari a 47 KByte e 8 Mbit di FLASH esterna. Il firmware intallato è il Bluelab versione 3.2. La connessione con l'ARM7 è realizzata mediante l'interfaccia UART (a 8 bit, senza controllo parità e con bit di stop) e ha la possibilità di essere utilizzata in due modalità, cioè per trasferire comandi o flussi di informazione. Entrambe le modalità permettono una velocità massima di 460 Kbit/s. La prima è utilizzata per istruzioni atte a creare la connessione, mentre la seconda viene utilizzata, a connessione avviata, per lo scambio dei dati. La tecnologia bluetooth implementata è la Class II con la quale si può comunicare con il brick ad una distanza di circa 10 metri. La velocità teorica massima per la trasmissione dati tra unità bluetooth è pari a 0.46 Mbit/s. Esiste poi una seconda interfaccia di connessione tra il chip BlueCore e l'ARM. Questa interfaccia, di tipo SPI, è utilizzata per l'upload del chip BlueCore.

Capitolo 2

Il Motore

Nel capitolo che segue presenteremo il motore in tutti i suoi aspetti. Vogliamo dare agli utenti di questo elaborato un approccio completo a tale strumento partendo dalla realizzazione hardware e software al fine di fornire i comandi basilari per l'utilizzo. Successivamente, servendosi di alcuni concetti fisici, determineremo la potenza erogata da questo attuatore. Come ultima sezione dedicheremo la nostra attenzione alla comprensione della gestione del motore mediante il sistema PID, incorporato nella classe che gestisce il motore, spiegandone l'utilità e il funzionamento.

2.1 Hardware

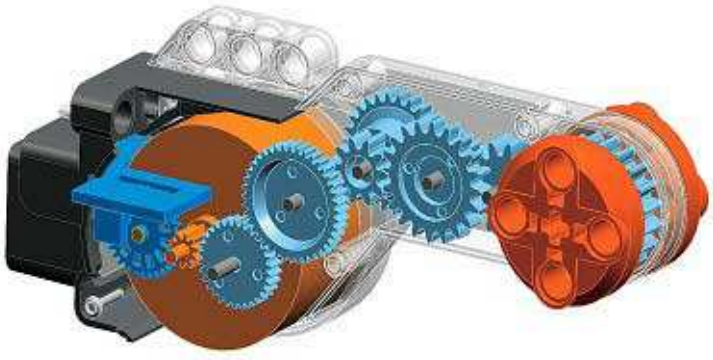


Figura 2.1: L'esploso del motore di un NXT

Il motore dell'NXT è un motore elettrico in corrente continua il cui albero non è direttamente collegato a disposizione dell'utente, bensì tramite una scatola di ingranaggi di riduzione. Infatti, se noi osserviamo esternamente questo componente potremo nota-

re una parte cilindrica, che corrisponde al motore vero e proprio, e una parte allungata in cui risiede la scatola degli ingranaggi.

2.1.1 Riduttore

La scatola degli ingranaggi è una struttura che contiene coppie di ingranaggi il cui numero di denti, partendo dal primo fissato sull'albero motore, è: 10:30, 30:40, 9:27, 10:20, 10:13, 13:20.
Il rapporto di riduzione totale dall'albero motore all'albero a cui l'utente può connet-

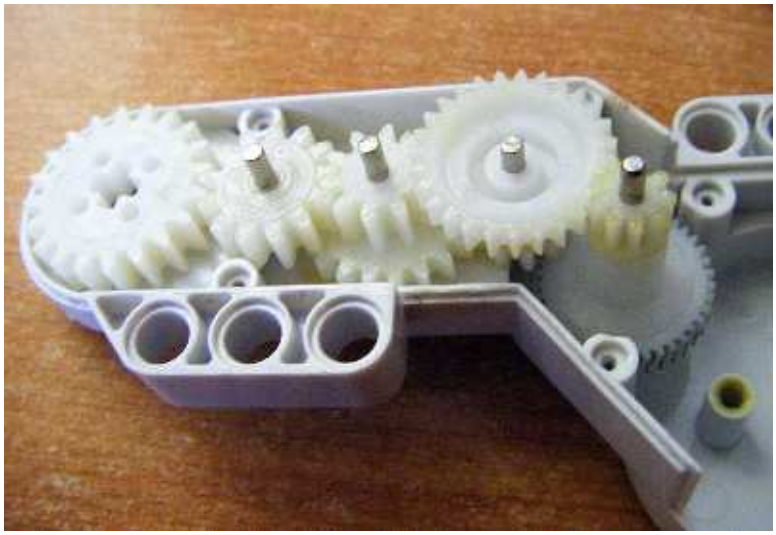


Figura 2.2: La scatola degli ingranaggi di riduzione all'interno del motore

tere i componenti Lego è quindi di 1:48, cioè per un giro del motore, l'albero che noi vediamo esternamente compirà 1/48esimo di giro o meglio, per far compiere un giro all'albero a cui l'utente ha accesso, il motore ne fa 48. Questa riduzione di velocità di rotazione fornisce al motore una notevole "forza". Infatti, se la velocità di rotazione viene ridotta di 48 volte il momento della forza[1], agente sull'albero finale, aumenta dello stesso valore. Per dare l'idea di momento di una forza pensiamo ad esso come la più comunemente nota coppia, commercialmente indicata per qualsiasi motore o come valore di serraggio per viti e bulloni. Nella sezione successiva, mediante un esperimento, spiegheremo e calcoleremo questo valore. Per ora si tenga conto che la realizzazione del motore è stata pensata, principalmente, per fornire il moto al robot, compito quindi gravoso, e questo giustifica la realizzazione così complessa e ingombrante di un riduttore che, oltre a ciò, comporta una dissipazione di energia dovuta agli attriti tra

[1] Per una delucidazione sul concetto si faccia riferimento alla sezione 2.3

gli ingranaggi e con gli alberi. Questa perdita è comunque minima e, in assenza di un riduttore, all'albero motore non ci sarebbe la coppia necessaria per far muovere il robot e, di contro, un valore di velocità angolare troppo elevato.

2.1.2 Encoder

Oltre a quanto precedentemente detto il motore è dotato di un encoder.
Esso è un dispositivo elettro-meccanico che permette di trasformare una posizione an-

Figura 2.3: Foto dell'encoder montato sull'NXT

golare, o una variazione di essa, in un segnale elettrico. Ne esistono di due tipologie: Assoluti o Relativi. I primi forniscono digitalmente la posizione esatta dell'albero cifrata con un'opportuna sequenza di valori binari. I secondi invece indicano la variazione della posizione, si può sapere cioè quando varia di un'unità di misura prescelta (per esempio quando si ha uno spostamento di 1°), ma non si può sapere la posizione attuale dell'albero a meno che, a priori, non si tenga un conteggio delle variazioni registrate e sia nota la posizione di partenza. Con questi dati si può rilevare in qualsiasi istante in che posizione sia il nostro motore e questo è il sistema utilizzato per l'encoder del motore dell'NXT. Generalmente il meccanismo si basa, per effettuare la misura, su un rotore che presenta delle feritoie a distanza costante ed è collegato all'albero motore. Dei fotodiodi identificano quando il rotore è su una feritoia oppure no, cioè se vedono o no la luce emessa da un dispositivo posto dall'altra faccia del rotore. Questa variazione di stato, da luce a buio o da buio a luce, di un fotodiodo identifica la percorrenza di una

determinata porzione d'angolo nota a priori. Quindi un solo fotodiodo è necessario per comprendere l'entità della rotazione effettuata dal rotore. La Lego ha implementato questo sistema realizzando il motore. Si nota, dalla figura 2.3, che il rotore dell'NXT presenta 12 feritoie, quindi si avranno 24 cambiamenti di stato per ogni sua rotazione. Come detto in precedenza, il motore deve compiere 48 giri affinché l'albero finale ne compia 1. Il rotore è collegato mediante una coppia di ingranaggi con un numero di denti pari a 10:32. Effettuando gli opportuni calcoli, ad ogni giro dell'albero finale il rotore compie 15 giri che, moltiplicato per 24 cambiamenti di stato per ogni suo giro, identifica 360 cambiamenti di stato per ogni rotazione dell'albero a cui colleghiamo i nostri componenti LEGO. Abbiamo così determinato che l'encoder ha una risoluzione pari ad 1°. Le informazioni vengono inviate al NXT tramite le linee TACHOX0 e TACHOX1 (X=A,B,C che identifica le porte). Ad ognuna delle due linee è collegato un fotodiodo. Ma perchè utilizzare due fotodiodi quando, come precedentemente detto, ne è sufficiente uno? Perchè il motore può ruotare in due direzioni. I fotodiodi sono posti ad una distanza tra loro inferiore a quella della dimensione di una feritoia e precisamente pari alla metà di essa. Grazie a questa semplice ed economica realizzazione si è in grado, oltre che a determinare la variazione della posizione del motore, di dedurne il verso di rotazione. Infatti, essendo i sensori posizionati come specificato sopra, il cambiamento di uno stato, cioè il passaggio da una feritoia all'altra, avverrà prima su uno e poi sull'altro sensore se il motore va in un verso. Il contrario se va nell'altro. Queste informazioni vengono lette dall'NXT che è così in grado di interpretare la direzione del motore.
L'immagine 2.4 rappresenta l'andamento dei segnali provenienti dai fotodioti in caso di

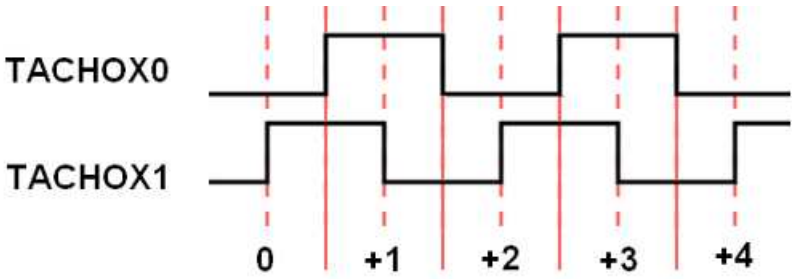
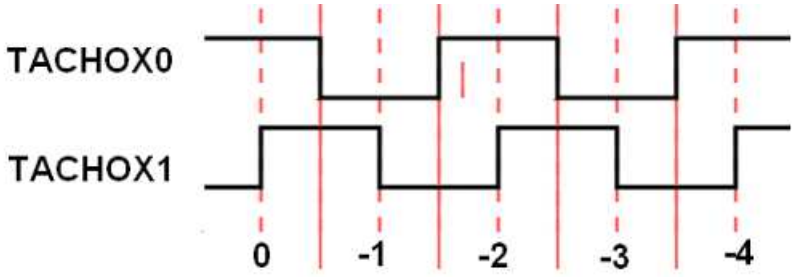


Figura 2.4: Sopra rotazione oraria e sotto antioraria



rotazione oraria e antioraria. Come si vede un periodo di un'onda quadra corrisponde alla variazione di due gradi in quanto si hanno due cambiamenti di stato per ciascun fotodiodo. Si può notare che, le due onde sono in quadratura tra loro, cioè sfasate di 90°, questo dovuto al posizionamento dei sensori. Nelle tabelle 2.1 e 2.2 si può osservare l'andamento dei segnali provenienti dai due fotodiodi e trasmessi attraverso le linee TACHOX0 e TACHOX1. Identifichiamo con 1 la presenza, di fronte al diodo, delle feritoia.

Per concludere questa sezione vogliamo solo invitare i lettori a non cercare di aprire

TACHOX0	TACHOX1
0	0
0	1
1	1
1	0

Tabella 2.1: Andamento dei segnali nel caso di rotazione oraria

TACHOX0	TACHOX1
1	0
1	1
0	1
0	0

Tabella 2.2: Andamento dei segnali nel caso di rotazione antioraria

il motore di un NXT in quanto l'albero finale non è divisibile, cioè le due parti delle pulegge arancioni sono incollate tra loro. Le foto sono state ricavate dal web da un utente che ha aperto un motore bruciato.

2.2 Metodi della classe Motor

Il Lejos mette a disposizione la classe `Motor` per la gestione dei motori. Questa classe contiene tre oggetti statici, `Motor.A`, `Motor.B` e `Motor.C`, che permettono la gestione dei relativi motori. Per utilizzare un motore non è quindi necessario dichiarare un oggetto di tipo `Motor`, ma è sufficiente utilizzare quelli già messi a disposizione dalla

suddetta classe. Oltre ad essi la classe `Motor` dispone di vari metodi per l'utilizzo dei motori. Ad esempio per invocare un metodo sul motore A sarà sufficiente digitare `Motor.A.NomeMetodo()`.
Prima di elencare e spiegare i metodi disponibili specifichiamo che per controllare il motore è necessario inviare alla relativa porta due soli dati: la potenza, da 0 a 100, e la modalità del motore (stop, forward, backward, float). Oltre a questi dati in output la porta fornisce, in input, la posizione del motore. Quindi tutti i metodi della classe Motor agiscono su due semplici comandi, `controlMotor(int power, int mode)` e `getTachoCount`.
Oltre a ciò occupiamoci di una particolarità della classe `Motor`. In essa è contenuta una classe interna, `Regulator`. Possiamo pensare ad essa come una porzione di codice (un thread[2]) che viene eseguita in parallelo al codice che noi scriviamo ed eseguiamo. Questa classe permette di:

- utilizzare la funzione `lock` che mantiene il motore nella posizione richiesta anche se esso viene sottoposto a forze esterne, cioè se, per esempio, cerchiamo di farlo ruotare con una mano;
- utilizzare le funzioni `rotate` e `rotateTo` che permettono uno spostamento pari, o verso, un angolo richiesto;
- regolare la velocità[3] del motore alla velocità richiesta, per mezzo del PID, correggendo, ove possibile, eventuali variazioni di essa. Per esempio se costruiamo un robot che si muove per mezzo dei motori e, facendolo correre su un piano, incontra una salita che farà diminuire la velocità, il PID aumenterà la potenza impostata al motore cercando di riportare la velocità a quella richiesta.

Chiariti questi aspetti elenchiamo e indichiamo la funzione dei metodi pubblici:

- **`forward()`**: avvia il moto del motore in senso orario;
- **`backward()`**: avvia il moto del motore in senso antiorario;
- **`reverseDirection()`**: inverte la direzione del moto;
- **`flt()`**: ferma il motore togliendo la potenza ad esso erogata. Chiamando questo metodo il motore si arresta per inerzia cioè continuerà il suo moto senza fermarsi bruscamente ed imprimendo quindi una bassa resistenza;
- **`stop()`**: ferma in maniera istantanea, o quasi, la rotazione del motore che tenderà ad arrestarsi bruscamente fornendo resistenza al moto. Per capire in maniera chiara le differenze con il comando flt() invitiamo a provare la chiamata dei due metodi su un motore e testare la resistenza che esso oppone quando cerchiamo di farlo ruotare;

[2] Per la spiegazione di questo termine si invita alla visione della seconda parte della tesi relativa al software

[3] Per chiarimenti si veda la sezione 2.4

- **`lock(int power)`**: questo metodo è utilizzabile solo quando il thread `Regulate` è in esecuzione. Permette il bloccaggio del motore all'angolo in cui esso si trova al momento in cui il metodo `lock` viene chiamato. Questo angolo è chiamato `limitAngle`. Il `Regulate` riporterà il motore alla posizione prefissata nel caso esso venga spostato. Se l'angolo a cui si trova il motore è maggiore o minore di `limitAngle`, il motore verrà fatto ruotare in direzione del `limitAngle` con una potenza pari a `power`, valore passato alla chiamata del metodo;
- **`isMoving()`**: ritorna un valore booleano pari a `true` se il motore è in movimento, `false` nel caso opposto;
- **`rotateTo(int limitAngle, boolean immediateReturn)`**: questo metodo è utilizzabile solo quando il thread `Regulate` è in esecuzione. Permette di far ruotare il motore ad un angolo indicato con la variabile `limitAngle`. Ad esempio se, quando il metodo viene chiamato, il motore si trova all'angolo 130 e la variabile `limitAngle` ha un valore pari a 30, esso ruoterà fino a raggiungere la posizione 30. Attenzione: quando il motore arriva nella posizione indicata non viene richiamato il metodo `lock` ma il metodo `stop`. Questo vale anche per i metodi sotto. Se la variabile `immediateReturn` è `true` nel tempo necessario alla rotazione l'esecuzione del codice continuerà. Se è impostata a `false` il nostro programma continuerà la sua esecuzione solo al termine della rotazione. In esempio a ciò, se si deve far muovere un robot di un determinato spazio e, successivamente, controllare con il sensore ad ultrasuoni se ci sono ostacoli, imposteremo `immediateReturn` a `false`. Se invece dobbiamo far muovere un robot di un determinato spazio e, nel frattempo, controllare se ci sono ostacoli su cui esso può impattare, imposteremo immediateReturn a true;
- **`rotateTo(int limitAngle)`**: overloading del metodo `rotateTo(int limitAngle, boolean immediateReturn)`. Questo metodo ha lo stesso funzionamento del precedente per quanto riguarda l'entità della rotazione. Differisce da esso perché la variabile `immediateReturn` è `false` per definizione;
- **`rotate(int angle)`**: overloading del metodo `rotate(int limitAngle, boolean immediateReturn)`. Questo metodo ha lo stesso funzionamento del successivo per quanto riguarda l'entità della rotazione. Differisce da esso perché la variabile `immediateReturn` è `false` per definizione;
- **`rotate(int angle, boolean immediateReturn)`**: Questo metodo permette una rotazione del motore, relativamente alla posizione in cui esso si trova al momento della chiamata del metodo, pari all'angolo, `angle`, passato come parametro. Ad esempio se, quando il metodo viene chiamato, il motore si trova all'angolo 130 e la variabile `angle` ha un valore pari a 30, esso ruoterà fino a raggiungere la posizione 160. Rotate effettua una chiamata al metodo `rotateTo(int limitAngle, boolean immediateReturn)` passando come parametri la posizione attuale del motore sommata al valore `angle` della rotazione e la variabile `immediateReturn`.

- **`shutdown()`**: disattiva il thread `Regulate`. Attenzione: non sarà più possibile utilizzare i metodi `rotate`, `rotateTo` e `lock`. Oltre a questo non si gioverà più del controllo di velocità fornito dal PID[4];

- **`regulateSpeed(boolean yes)`**: assegnando alla variabile `yes` i valori `true` o `false` attiva o disattiva il controllo di velocità fornito dal PID[5];

- **`smoothAcceleration(boolean yes)`**; assegnando alla variabile yes i valori true o false, attiva o disattiva la funzione denominata *Smooth Acceleration*[6] fornita dal PID;

- **`setSpeed(int speed)`**: imposta la velocità del motore. Il valore da assegnare alla variabile `speed` è pari alla velocità, in gradi al secondo, a cui si desidera che il motore ruoti. Attenzione: la velocità reale varia. Se il PID è attivato essa coincide con il valore impostato nella variabile `speed`; se il PID è disattivo non si assicura che essa sia pari al valore desiderato[7]. Il metodo accetta qualsiasi valore ma non si consiglia di impostare questa velocità a più di 900 con la batteria al litio in quanto la tensione di 7,2V fornita non permette valori più elevati;

- **`setPower(int power)`**: imposta la potenza del motore, un valore di `power` negativo inverte il moto. Attenzione: è bene non utilizzare questo comando se il PID è attivato in quanto darebbe reazioni imprevedibili del motore. Per impostare la velocità di rotazione si consiglia di utilizzare sempre `setSpeed` sia che il PID sia attivo o disattivo;

- **`getSpeed()`**: ritorna un intero pari al valore della velocità impostata al motore;

- **`getMode()`**: ritorna un intero che indica la modalità in cui si trova il motore: 1 = forward, 2= backward, 3 = stop, 4 = float;

- **`getPower()`**: ritorna un intero, da 0 a 100, pari al valore della potenza impostata al motore;

- **`getLimitAngle()`**: ritorna un valore intero pari all'angolo impostato dalla funzione `lock` o `rotateTo`;

- **`isRegulating()`**: ritorna un valore booleano pari a `true` se il PID è attivo, `false` altrimenti;

- **`getRotationSpeed()`**: ritorna un intero pari al valore della velocità reale del motore. Questo valore è calcolato come media della velocità sostenuta negli ultimi 100ms ed è fornito in gradi al secondo;

[4] Per chiarimenti si veda la sezione 2.4
[5] Per chiarimenti si veda la sezione 2.4
[6] Per chiarimenti si veda la sezione 2.4
[7] Per chiarimenti si veda la sezione 2.4

- **`getTachoCount()`**: ritorna un intero pari al valore della posizione istantanea del motore;
- **`resetTachoCount()`**: resetta la posizione del motore impostando come 0 la posizione a cui il motore si trova al momento in cui questo metodo viene chiamato;
- **`getError()`**: ritorna un floating point pari al valore dell'errore di inseguimento[8];
- **`getBasePower()`**: ritorna un floating point pari al valore della componente integrativa del PID[9];
- **`setBrakePower(int pwr)`**: imposta il valore di potenza che il motore utilizza per effettuare la correzione finale della sua posizione verso l'angolo impostato con il metodo rotate o rotateTo[10].

2.3 Calcolo coppia e potenza meccanica del motore

Per avere un primo approccio all'uso del motore affrontiamo ora un'esercitazione pratica. Questo esperimento è stato pensato al fine di applicare alcune nozioni di fisica alla programmazione e, con esse, ricavare in maniera sperimentale la coppia meccanica fornita da un motore dell'NXT. Ma cos'è quella che comunemente è chiamata coppia in un motore?

2.3.1 Il momento di una forza

Il termine coppia deriva dal fatto che questa forza può essere vista, appunto, come una coppia di forze di ugual modulo. Esse sono parallele, di verso opposto, giacenti sullo stesso piano e agenti sugli estremi di un asse, perpendicolarmente ad esso. Quest'ultimo è imperniato a metà, cioè il suo fulcro è equidistante dalle forze. Parliamo quindi di forza agente su un braccio e pertanto siamo in un caso particolare del momento di una forza.
Consideriamo una forza **F** agente su una particella la cui posizione *P*, rispetto all'origine *O* di un sistema di riferimento inerziale, è data dal vettore posizione **r**. Poiché i

[8] Per chiarimenti si veda la sezione 2.4
[9] Per chiarimenti si veda la sezione 2.4
[10] Per chiarimenti si veda la sezione 2.5

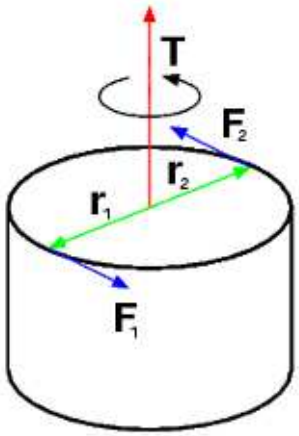


Figura 2.5: Una coppia di forze

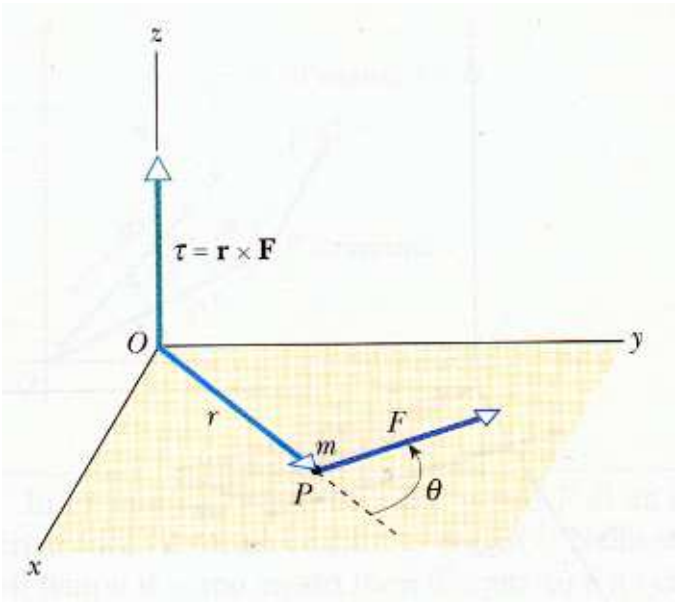


Figura 2.6: Rappresentazione vettoriale del momento di una forza

due vettori individuano un piano, il piano *xy* è stato scelto in modo da contenere **r** e **F**. Il momento τ della forza **F** rispetto all'origine *O* è definito come il prodotto vettoriale tra **r** e **F** cioè $\tau = \mathbf{r} \times \mathbf{F}$. Il momento di una forza è quindi una grandezza vettoriale il cui modulo è dato da $\tau = \mathbf{rF}\sin\theta$ dove θ è l'angolo tra **r** e **F**. La direzione di questo momento è perpendicolare al piano formato da **r** e **F** cioè, in questo caso, parallela all'asse *z.* Il verso di questo vettore è dato dalla regola della mano destra: guardando il piano

xy va verso l'osservatore se la rotazione generata dalla forza **F** è antioraria, opposta se la rotazione è oraria. Il momento quindi è espresso come Nm. L'unità di misura può sembrare coincidente a quella utilizzata per il lavoro, ma dobbiamo ricordare che il momento è un vettore, a differenza del lavoro che è uno scalare, quindi il momento non si esprime in J = Nm ma in Nm.
Come si nota dalla figura 2.7 possiamo scrivere l'equazione $\tau = \mathbf{r}\mathbf{F}\sin\theta$ come $\tau = (\mathbf{r}$-

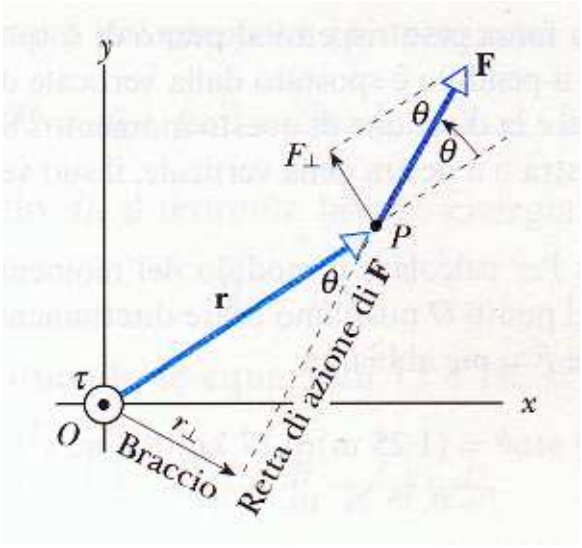


Figura 2.7: Rappresentazione vettoriale del momento di una forza

$\sin\theta)\mathbf{F} = \mathbf{r_p}\mathbf{F}$ o come $\tau = \mathbf{r}(\mathbf{F}\sin\theta) = \mathbf{F_p}\mathbf{r}$ dove $\mathbf{r_p}$ e $\mathbf{F_p}$ sono rispettivamente la componente di **r** perpendicolare alla retta di azione di **F**, mentre la seconda è la componente di **F** perpendicolare al braccio **r**. Nella prima equazione r_p assume il significato di braccio al posto di **r**. Quindi con queste equazioni si vuol mostrare che solo la componente di **F** perpendicolare al relativo braccio contribuisce al momento della forza, mentre la componente di **F** parallela al braccio agisce sull'origine su cui il braccio è imperniato, ma assumiamo essa come inamovibile.
Introdotti questi concetti, per calcolare il momento creato dal motore dell'NXT abbiamo bisogno di valutare quale sia l'intensità del vettore $\mathbf{F_p}$ generato da esso, avendo noto il braccio di applicazione. Per valutare ciò potremmo applicare, allo stesso braccio, una forza nota che crei un momento opposto a quello del motore. Il motore creerà un momento superiore e contrario a quello creato dalla forza sottoponendo il punto *P*, estremo del braccio di applicazione, ad un'accelerazione. Non avendo potenza infinita, raggiunta una determinata velocità angolare, il momento creato dal motore sarà uguale e opposto a quello creato dalla forza. Il moto continua a velocità costante e, di conseguenza, accelerazione nulla. Applicando diverse forze, via via crescenti, si può delineare un'insieme di punti le cui coordinate sono la velocità angolare e il momento esercitato. Esso è il τ massimo che il motore può generare a tale velocità. Si può in

questo modo tracciare un grafico rappresentante la curva commercialmente nota come curva di coppia del motore.

2.3.2 Realizzazione strumentazione per l'esperimento

Ora pensiamo a come generare una forza di intensità nota e direzione costante perpendicolare al braccio. La forza che, a nostro avviso, è più facile da creare e sfruttare al nostro scopo, è la forza peso. Essa è espressa, attraverso la seconda legge della dinamica, come **F = mg** dove **g** è l'accelerazione gravitazionale esercitata dalla terra e **m** il peso noto della massa. Esiste ora un altro problema: essa deve essere esercitata, attraverso un braccio, al motore e deve essere perpendicolare ad esso in ogni momento della rotazione. Possiamo realizzare ciò tramite una puleggia che, opportunamente fissata al motore, avvolga un filo a cui è attaccato un corpo di cui è nota la massa.
Questa soluzione ci permette di avere un braccio di lunghezza fissata, e una forza **F**

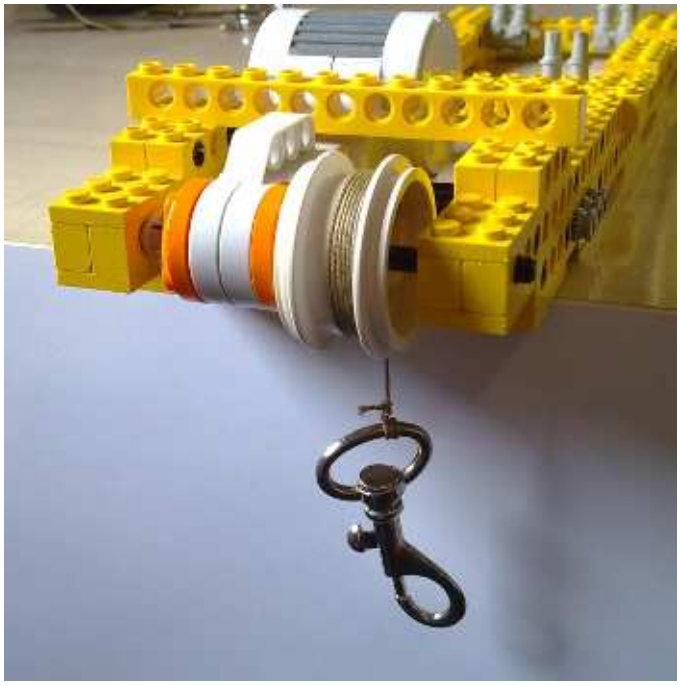


Figura 2.8: Struttura realizzata per effettuare l'esperimento

costantemente perpendicolare ad esso, il cui modulo è scelto a priori. Già da una prima analisi possiamo però intuire quali possono essere i problemi. La puleggia e il filo devono avere una massa trascurabile rispetto a quella del corpo utilizzato per generare la forza. La massa della puleggia applicata all'asse del motore, e sottoposta a rotazione, crea inerzia mentre la massa del filo varia la sua azione nel corso dell'avvolgimento.

All'inizio infatti, quando il filo è completamente srotolato, esso contribuisce aumentando il peso della massa **m** e quindi anche il momento esercitato sul motore. Man mano che il filo si avvolge alla puleggia il peso di esso si trasferisce a quest'ultima aumentandone l'inerzia e diminuendo la forza **F** applicata. Questi problemi possono essere facilmente resi trascurabili dimensionando opportunamente le parti in movimento e scegliendo un filo ed una puleggia con un ridotto peso. Un ulteriore problema consiste nel fatto che il filo, avvolgendosi, può creare spire sovrapposte e, quindi, variare la lunghezza del braccio a cui la forza **F** viene applicata. Si cercherà di risolvere questo problema utilizzando un filo il cui diametro si possa considerare trascurabile rispetto al braccio e cercando di far avvolgere il filo a spire costanti. Non preoccupiamoci molto di questo aspetto in quanto, da osservazioni ripetute durante l'esperimento, il filo si avvolge con spire elicoidali una affianco all'altra, tutte con lo stesso raggio. Un ultimo accorgimento è quello di lubrificare gli eventuali perni in modo che l'attrito, esercitato da essi sulle superfici, non sfalsi le misurazioni creando una forza contraria al moto.

2.3.3 La Potenza

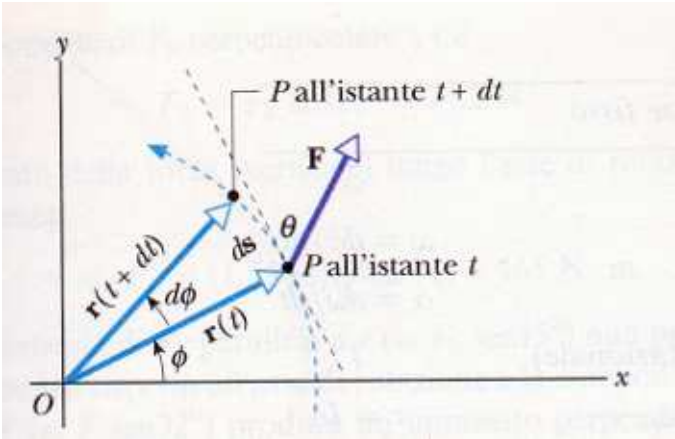


Figura 2.9: Rappresentazione del Lavoro dovuto al momento di una forza

Risolti questi problemi di realizzazione saremo in grado di calcolare il grafico "*Momento della Forza / Velocità Angolare*". Ma questo calcolo è fine a se stesso? Calcolato in questo modo il modulo del momento τ avremmo nota qual'è la massima forza che il motore può esercitare su un braccio di lunghezza definita ad una determinata velocità angolare. Un'asta, imperniata nel punto *O*, soggetta ad una forza, applicata nel punto *P*, è soggetta ad una rotazione di un angolo $d\phi$ in un tempo *d*t e descrive, al punto *P*, un arco di lunghezza $d\mathrm{s} = \mathbf{r}d\phi$. Il lavoro compiuto *d*L è pari a $\mathbf{F}d\mathrm{s}$. Come per quanto

già accennato, per il momento τ, dobbiamo considerare solo la componente della forza perpendicolare al braccio, quindi, come da figura 2.9, il lavoro dL è pari a $\mathbf{F}\cos\theta\, ds$. Scomponendo ds come $\mathbf{r}\, d\phi$, otteniamo che il lavoro $dL = (\mathbf{F}\cos\theta)\, \mathbf{r}\, d\phi$. Ma $\mathbf{F}\cos\theta$ è $\mathbf{F}_p$ quindi, per la definizione di momento di una forza, il lavoro $dL = \tau\, d\phi$. Anche questa grandezza ha come unità di misura il Nm ma, in questo caso, le componenti di spostamento e forza sono parallele quindi esso è uno scalare. Il Nm per il lavoro assume il valore di Joule (J). Calcolato il lavoro, nel caso di moto rotazionale, si definisce la potenza $P = dL / dt$, cioè come la rapidità con cui il lavoro viene compiuto. Preso $d\phi = \omega\, dt$, dove ω è la velocità angolare (espressa in radianti al secondo), si può esprimere dL come $\tau\, \omega\, dt$ quindi $P = \tau\, \omega$. La potenza può essere definita come il momento della forza applicata moltiplicato per la velocità di rotazione che ha il corpo e ha come unità di misura il J/s cioè il Watt. Attenzione, la potenza calcolata è meccanica. La potenza elettrica assorbita dal motore, anch'essa in Watt, è diversa da quella meccanica non avendo il motore rendimento pari a 1 ed essendoci comunque dissipazione di energia dovuta ad attriti.

2.3.4 L'acquisizione e i risultati

Affrontati gli aspetti fisici dell'esperimento passiamo alla programmazione. Elenchiamo di seguito i passi che il nostro programma dovrà effettuare al fine di raccogliere i dati per l'esperimento.

1. Disattiviamo il PID con il comando `Motor.A.regulateSpeed(false);`
2. Impostiamo la velocità oltre al massimo consentito con la batteria al litio con `Motor.A.setSpeed(1000)` in modo che al motore sia impostato un fattore di potenza pari al 100%;
3. Avviamo il movimento del motore con `Motor.A.forward();`
4. Finché il motore non ha compiuto un giro evitiamo di raccogliere i dati in quanto sarebbero falsati. Inizialmente infatti, il motore riesce a mantenere un'accelerazione angolare maggiore a 0, quindi esercita un momento τ maggiore a quello contrario effettuato dalla massa. Questo avviene finché l'attuatore non raggiunge la velocità massima sostenibile con il momento contrario a cui è sottoposto;
5. Finché il motore non riavvolge tutto il filo (bisogna calcolare quanti giri sono necessari) calcoliamo, ad ogni variazione di 3 gradi, la velocità istantanea come: `(posizione attuale - posizione precedente) / (tempo attuale - tempo precedente);`
6. Effettuiamo la media dei dati rilevati;
7. Stampiamo a monitor la media ottenuta.

Eseguendo più volte questo programma e variando la massa applicata, e con essa il momento della forza, avremo un risultato simile a quello illustrato nella figura 2.10
In questo grafico possiamo osservare i dati ricavati per via sperimentale. In blu è rap-

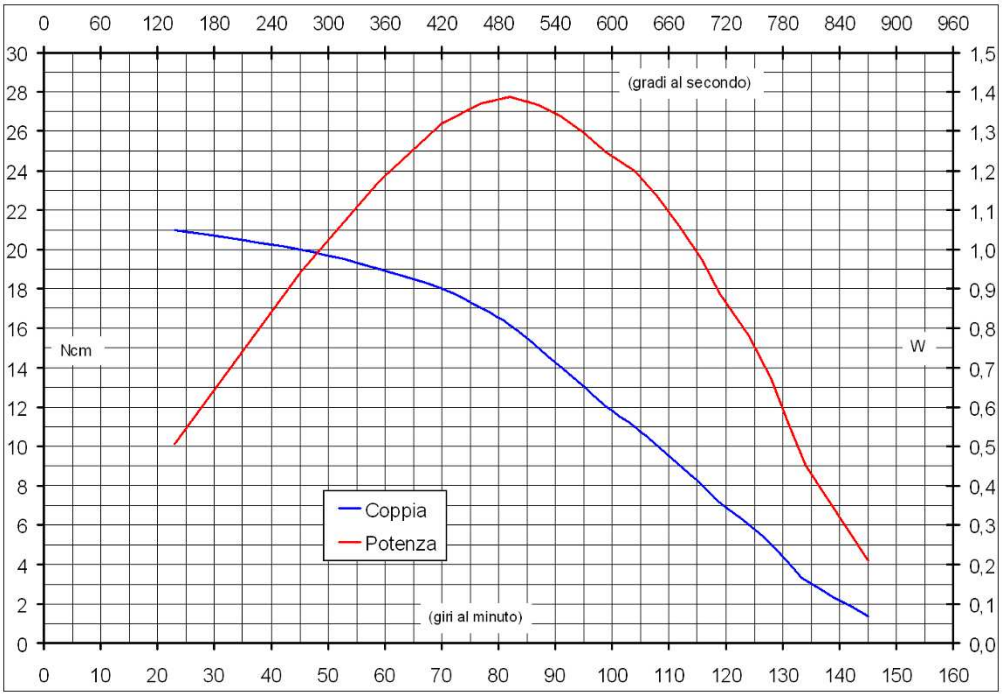


Figura 2.10: Grafico dei dati rilevati

presentato il valore del massimo momento τ sostenibile alla rispettiva velocità angolare. L'unità di questa misurazione è Ncm e fa riferimento all'asse verticale sinistro. In rosso è rappresentata la potenza meccanica del motore. I valori sono espressi in Watt e fanno riferimento all'asse verticale destro del grafico. Le unità di misura per la velocità sono i gradi/secondi e giri/minuti.
Questo esperimento, come già affermato, è pensato per applicare delle nozioni fisiche alla programmazione robotica in modo da approfondire entrambi gli aspetti. Si invitano gli studenti ad effettuare a loro volta questa esperienza in modo da confermare, o confutare, i dati rilevati. In questo modo essi potranno cimentarsi per la prima volta nella stesura di un programma.
Oltre a ciò, è comunque utile osservare il grafico risultante in modo da rendersi conto dei limiti meccanici del motore quando vi si applicano forze che provocano momenti opposti al moto. È bene quindi tenerne conto ed eventualmente stimare la velocità di questi movimenti per verificare se il motore è in grado di fornire il momento τ necessario a compierli alla velocità voluta. Per esempio, se costruiamo una macchina con il nostro NXT e la facciamo salire lungo un piano inclinato, la componente della forza di gravità, parallela ad esso, ostacolerà il moto. Poniamo che il piano sia inclinato di 20

gradi e la macchina pesi 1kg. Questi presupposti impongono una forza opposta al moto pari a 3.36N che, applicata al motore mediante le ruote di raggio pari a 3 cm, creano un momento pari a circa 10Ncm. Osservando sul grafico possiamo determinare che in queste condizioni il motore può, al massimo, mantenere una velocità di circa 105 giri al minuto, cioè 630 gradi al secondo, con una velocità massima della macchina pari a 19.8 metri al minuto.

2.4 Il PID

In questa sezione tratteremo degli argomenti che potrebbero essere di difficile comprensione per uno studente delle scuole medie superiori in quanto faremo riferimento ai sistemi dinamici e al loro controllo. Abbiamo comunque cercato di esprimere questi concetti con l'abbondante uso di supporti grafici al fine di facilitare la comprensione delle considerazioni da noi portate.

2.4.1 Cos'è un controllore PID

PID è l'acronimo di controllo Proporzionale Integrativo Derivativo e realizza un sistema di controllo negativamente retroazionato. Ma questo cosa significa?
Il termine controllo definisce l'azione svolta per portare e mantenere, ad un valore prefissato, un parametro fisico di un impianto o di un processo come, ad esempio, la temperatura di un forno, il livello di un fluido in un serbatoio, la posizione del braccio di un robot, la velocità di rotazione di un motore, ecc.
La retroazione (in inglese feedback) definisce la capacità dei sistemi dinamici di tenere conto dei risultati del sistema per modificare le caratteristiche dello stesso. In un controllo in retroazione il valore della variabile in uscita viene letto dal controllore che agisce modificando l'ingresso del sistema. In questi controlli, detti ad anello chiuso, il valore viene determinato e corretto in base alla misura della variabile controllata e alla verifica della sua rispondenza, per questo motivo i sistemi retroazionati vengono anche chiamati esplorativi. La retroazione poi è detta negativa quando i risultati del sistema vanno a smorzare il funzionamento dello stesso stabilizzandolo. I sistemi con retroazione negativa sono in genere stabili e tipicamente tendono a convergere, cioè a stabilizzarsi attorno ad un certo valore.
Indicando con `r(t)` il valore che si vuole far assumere alla variabile controllata e con `y(t)` il valore effettivamente assunto da tale grandezza, possiamo introdurre una funzione d'errore definita come: `e(t) = r(t) - y(t)`. Lo scopo del PID è quello di

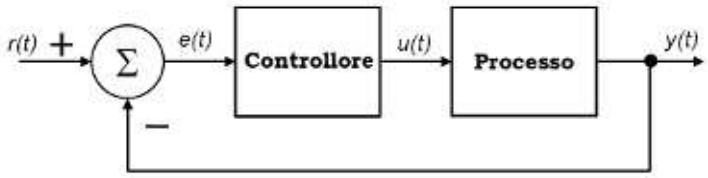


Figura 2.11: Rappresentazione mediante schema a blocchi di un sistema retroazionato

applicare la migliore scelta possibile dei valori della funzione $u(t)$, detta variabile di controllo, che:

- renda il sistema asintoticamente stabile;
- minimizzi il valore medio di $e(t)$;
- riduca al livello minore possibile il tempo di risposta;
- riduca al livello minore possibile le fluttuazioni intorno al valore $r(t)$ in concomitanza delle repentine variazioni di questa variabile.

L'azione di controllo è eseguita dal blocco controllore, rappresentato in figura 2.12, che ha il compito di trasformare il segnale d'errore $e(t)$ in un segnale $u(t)$ che agisce sul processo sottoposto a controllo. Per effettuare tale traduzione si tiene conto delle componenti proporzionale, integrativa e derivativa dell'errore.
La prima componente è quella proporzionale. Essa rende il segnale di controllo proporzionale al segnale di errore. L'uscita $u(t)$ prende quindi la forma $k_p e(t)$ e il contributo di essa diminuisce quindi man mano che l'errore si avvicina a zero. Un sistema sottoposto ad un controllo di questo tipo presenta, in risposta ad un segnale d'errore con limitate fluttuazioni, valori molto vicini a quelli attesi ma, nonostante ciò, presenta un errore a regime cioè uno scostamento prolungato dei valori dell'uscita $y(t)$ rispetto a quelli di riferimento $r(t)$. Oltre a ciò, per andamenti incostanti della variabile di errore, il controllo induce il sistema a instabilità dovuta ad improvvise variazioni della variabile $u(t)$.
L'inserimento della componente integrale dell'errore risolve il problema dell'errore a regime. La funzione che rappresenta il segnale $u(t)$ assume la forma $k_p e(t) + k_i \int_{t_0}^{t} e(T) dT$. Grazie alla componente integrativa il segnale di controllo è costante e di valore diverso da zero anche quando il segnale d'errore in ingresso è zero, questo perché l'integrale rappresenta la somma di tutti i valori passati di $e(t)$, cioè ha un'azione di memoria storica. Il sistema sottoposto ad un controllo di questo tipo non presenta più l'errore a regime, ma risponde ancora con instabilità in caso di variazioni improvvise di $e(t)$ dovute al cambiamento dell'uscita $y(t)$ o della variabile di riferimento $r(t)$.
L'ultima componente che viene aggiunta è quella derivativa che dipende dalla velocità con cui varia l'errore. La funzione che rappresenta il segnale $u(t)$ assume ora la

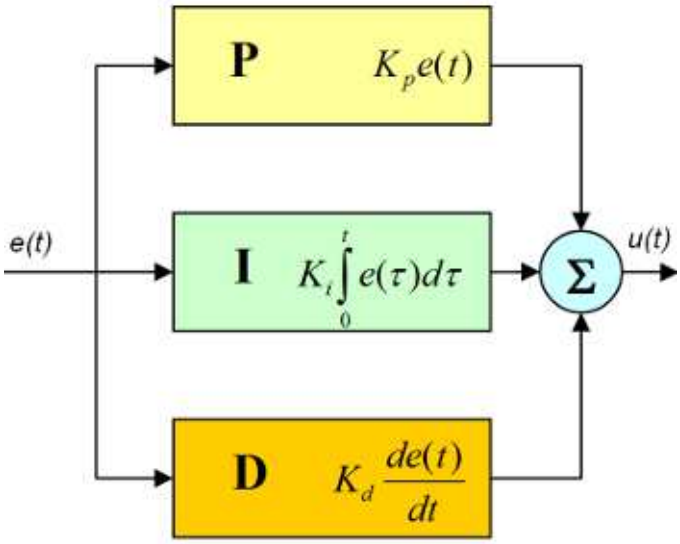


Figura 2.12: Rappresentazione mediante lo schema a blocchi di un controllore PID

forma canonica del PID cioè $u(t) = k_p e(t) + k_i \int_{t_0}^{t} e(T)dT + k_d de(t)/dt$. L'introduzione di questa componente cerca di anticipare il verificarsi dell'errore essendo proporzionale alla velocità con cui esso varia. Si limitano quindi i ritardi alla risposta nel caso di repentine variazioni dell'errore ma, di contro, si rende "nervosa" la risposta del sistema.

2.4.2 L'algoritmo del PID

Il controllore precedentemente descritto è messo a disposizione dalla classe `Motor` ed è implementato nella classe interna `Regulate`. Il codice che rappresenta il PID è eseguito ciclicamente ogni 4ms ed è mostrato qui sotto.

```
int elapsed = (int)System.currentTimeMillis()-time0;
int angle = getTachoCount()-angle0;
int absA = angle;
if(angle<0)absA = -angle;
if(_rampUp)
{
  if(elapsed<ts)
  {
    error = elapsed*elapsed*_speed*7/(ts*2000);
    error = error+  elapsed*_speed*3/2000;
    error = error - 10*absA;
  }
  else
```

```
  {
    error = ((elapsed - ts/2)* _speed)/100 - 10*absA;
  }
}
else     error = (elapsed*_speed/100)- 10*absA;
int gain = 5;
int extrap = 4;
power = basePower/10 + gain*(error + extrap*(error - e0))/10;
e0 = error;
if(power < 0) power = 0;
if(power > 100) power = 100;
int smooth = 12;
basePower = basePower +smooth*(10*power-basePower)/1000;
setPower(power);
```

Di seguito indichiamo il significato delle variabili utilizzate:

- **`elapsed`**: rappresenta il tempo trascorso da quando è stato lanciato il comando che sta provocando il movimento del motore. Questo dato esprime la differenza tra il tempo assoluto e la variabile `time0` aggiornata ogni qualvolta si varia lo stato del motore;
- **`angle`**: rappresenta l'angolo compiuto da quando è stato lanciato il comando che sta provocando il movimento del motore. Questo dato esprime la differenza tra l'angolo attuale e la variabile `angle0` aggiornata ogni qualvolta si varia lo stato del motore;
- **`absA`**: non è altro che il valore della variabile precedente posto in valore assoluto;
- **`_speed`**: è la variabile che rappresenta il valore della velocità angolare a cui vogliamo che il nostro motore ruoti. Viene impostata a priori con il metodo `setSpeed`;
- **`gain`**, **`extrap`** e **`smooth`** sono costanti calcolate sperimentalmente che effettuano le correzioni attuate dalle variabili k_p, k_i, k_d;
- **`power`** è il valore in percentuale della potenza passata al motore. Questo dato è limitato tra 0 e 100;
- **`basePower`** ha funzione di memoria per la componente integrativa;
- **`e0`** contiene il valore dell'errore rilevato alla misurazione precedente;
- **`error`** rappresenta il valore dell'errore.

Chiarito cosa rappresentano le suddette variabili tralasciamo, per ora, la parte del codice interna alla condizione `if(_rumpUp)` che implementa la funzione *Smooth Acceleration*. Concentriamoci quindi, in primo luogo, sul funzionamento del PID il quale permette di correggere l'uscita `y(t)` grazie ad una variabile di controllo `u(t)`. Il valore

di quest'ultima è ricavato, per mezzo del controllore, a partire dall'errore `e(t)` dovuto alla differenza tra l'uscita `y(t)` e una variabile di riferimento `r(t)` che rappresenta l'ingresso del sistema. Ma quali sono, nel nostro caso, le grandezze che rappresentano le suddette variabili? `r(t)` è identificata dalla posizione, in gradi, che il motore dovrebbe avere all'istante `t` ed è calcolata come `elapsed*_speed`, cioè come tempo trascorso moltiplicato per la velocità impostata attraverso il metodo `setSpeed`. Questo significa che la velocità, derivata dalla variabile di riferimento `r(t)` al momento dell'avvio passa dal valore precedente, per esempio zero se il motore era fermo, al valore impostato dalla variabile `_speed` con un andamento a gradino. L'uscita `y(t)` del sistema è data dalla posizione reale del motore in gradi, cioè con la variabile `absA` che è ricavata dal metodo `getTachoCount()`. L'errore `e(t)` è dato quindi dalla differenza tra la posizione reale e quella di riferimento. Infine la variabile di controllo `u(t)` è la potenza del motore. Essa viene calcolata come `power = basePower/10 + gain*(error + extrap*(error - e0))/10`. Tralasciando la divisione per 10, dovuta ad un semplice fatto di precisione, le componendi sono:

- **`gain*error`** identifica la componente proporzionale;
- **`gain*extrap*(error-e0)`** rappresenta la componente derivativa dell'errore essendo proporzionale alla velocità con cui l'errore varia. Esso è infatti calcolato come dato attuale meno dato precedente;
- **`basePower`** rappresenta la componente integrativa, che non si riferisce però all'errore, bensì alla variabile `u(t)`. Essa viene infatti calcolata come `basePower + smooth*(10*power-basePower)/1000` acquisendo quindi il significato di memoria storica delle variazioni subite dalla variabile `power`.

Come abbiamo già detto, questo calcolo dell'errore cerca di approssimare l'andamento della velocità ad un gradino. Esso provoca quindi una forte accelerazione iniziale. Per esempio, nel caso si usi il motore per spostare un robot, esso potrebbe rispondere con un movimento "nervoso" o, ancor peggio, con problemi di instabilità. Per ovviare a questi inconvenienti viene inserito nel codice un calcolo alternativo della variabile `e(t)` indicato come *Smooth Acceleration.* Riprendiamo quindi ora la parte di codice precedentemente evitata e racchiusa dall'`if(_rumpUp)`. Ricordiamo che questa funzione, attiva di default, viene disattivata o riattivata con il metodo `smoothAcceleration(boolean yes)` che agisce sullo stato della variabile `_rumpUp`. Con questa modalità nei primi istanti di rotazione, esattamente per i primi 120ms pari al valore della variabile `ts`, l'errore viene calcolato non solo tenendo conto della velocità, ma anche cercando di imprimere un'accelerazione costante. La variabile `r(t)` viene calcolata come `elapsed * elapsed * _speed * 7/(ts*2000) + elapsed * _speed * 3/2000`. Si possono quindi osservare due componenti per il calcolo di `r(t)`:

- **`0,3 * (elapsed * _speed / 100) * 1/2`** rappresenta la componente lineare e ha peso pari a 3/10 nel calcolo della variabile;

- `0,7 * (elapsed/ts) * (elapsed * _speed / 100) * 1/2` rappresenta la componente esponenziale atta a mantenere un'accelerazione costante. Questa seconda parte ha, nel calcolo, peso pari a 7/10.

Per comprendere meglio l'andamento delle due variabili di riferimento `r(t)`, cioè con la funzione *smooth acceleration* attiva o no, invitiamo alla visione della figura 2.13. Nel grafico la velocità impostata è pari a 700 gradi/secondo. In ascissa si presenta il

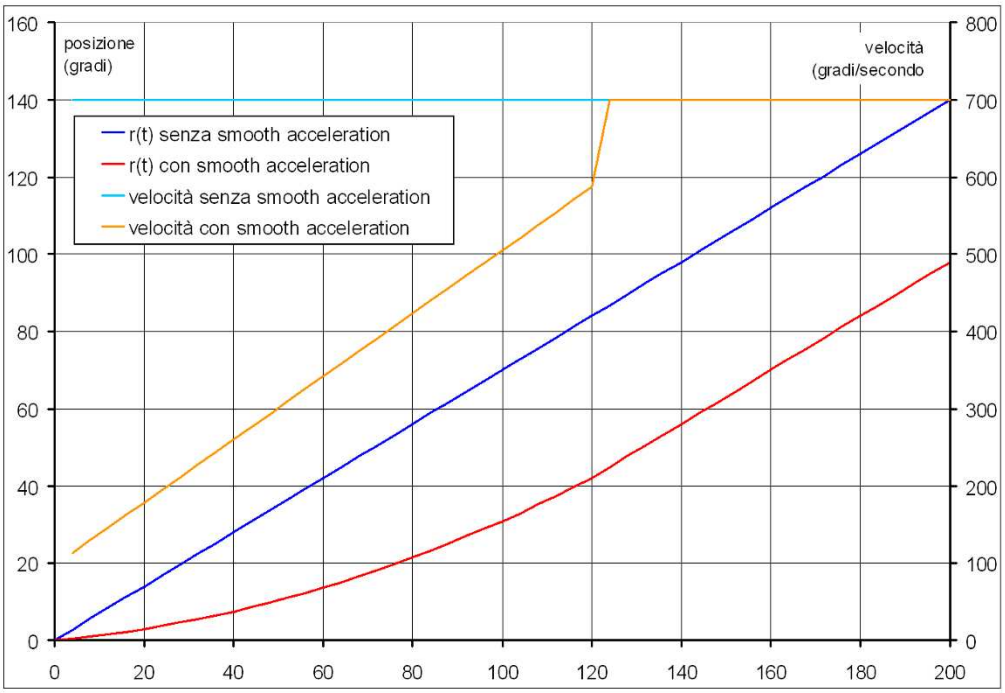


Figura 2.13: I valori assunti dalle variabili `r(t)` e le relative velocità derivate

tempo, in ms, trascorso dalla partenza del movimento del motore. Con i dati relativi a `r(t)` si identifica, come detto, il valore della variabile di riferimento. Essa rappresenta il valore di ingresso al sistema e quest'ultimo ha lo scopo di generare un'uscita di valore pari ad `r(t)`. Con questo si vuol dire che la posizione angolare del motore fornita dal metodo `getTachoCount` ad un istante t_1 dovrebbe essere pari al valore della variabile `r(t)` con $t=t_1$ ovviamente sommata all'angolo di partenza.
La linea in blu rappresenta la variabile `r(t)` nel caso in cui `Smooth Acceleration` sia disattivo. La linea rossa invece rappresenta `r(t)` in base al calcolo effettuato con la funzione `Smooth Acceleration` attiva. Si nota subito che la progressione iniziale della linea in rosso richiede un'accelerazione minore. Questa conclusione può essere confermata visionando la differenza tra le linee in azzurro e arancione. Esse rappresentano infatti la velocità che dovrebbe avere il motore rispettivamente con la funzione *Smooth Acceleration* disattivata ed attivata. Si confermano le ipotesi secondo le quali, nel caso questa funzione sia disattiva, la velocità ha un andamento a gradino. Nel

secondo caso invece l'andamento della velocità richiesta è, per i primi 120ms, paragonabile ad una retta con un aumento di velocità costante e pari a circa 16,3 gradi al secondo ogni 4ms.
Concludiamo questa analisi con un ultimo sguardo al codice. Dobbiamo osservare infatti che, trascorso un tempo pari a `ts` dalla partenza cioè 120ms, anche nel caso con *Smooth Acceleration* attiva viene utilizzata la formula `elapsed*speed` per il calcolo della variabile r(t). Si può infatti notare nel grafico che, dopo tale tempo, la differenza tra le due linee è costante. Dopo 120ms infatti esse proseguono parallele, traslate l'una dall'altra di 60ms. Definendo con `A(t)` la funzione contrassegnata in blu e `B(t)` quella in rosso, ne segue che `A(t-60ms) = B(t)` per `t > 120ms`. Vengono a conforto di questa osservazione anche i dati riguardanti le velocità derivate dalle due funzioni `r(t)`. I valori per `t>ts` infatti coincidono.

2.4.3 Raccolta dei dati

Per una più profonda comprensione del funzionamento del PID abbiamo realizzato un esperimento, al fine di acquisire periodicamente l'evoluzione delle variabili del sistema sottoponendo il motore a diversi momenti τ. Queste rilevazioni sono state effettuate utilizzando la struttura vista nella sezione 2.3.
Le prove sono state compiute con l'intervento di 3 diversi disturbi esterni:

- senza applicare al motore alcuna forza;
- applicando al motore un momento opposto al moto pari a 3.3Ncm;
- applicando al motore un momento nello stesso verso del moto e pari a 3.3Ncm.

Per ognuno di questi è stata rilevata la posizione e la velocità:

- con la funzione *Smooth Acceleration* attivata;
- con la funzione *Smooth Acceleration* disattivata;
- con il controllo del PID disattivato.

Le rilevazioni sono state effettuate, per ogni casistica, ad una velocità pari a:

- 700 gradi al secondo;
- 500 gradi al secondo;
- 300 gradi al secondo.

Si hanno quindi in tutto 27 test diversi.
L'acquisizione dei dati è stata effettuata salvando, su un vettore, il tempo assoluto in nanosecondi ad ogni variazione pari a 3 gradi del motore. Ad esso è stato impartito il comando `rotateTo(540)` che lo fa ruotare fino all'angolo assoluto passato come parametro. Arrivato al 540° grado e conclusa quindi la misurazione, il motore è stato riportato all'angolo 0 mediante il comando `rotateTo(0)`. Questa operazione è stata ripetuta per 5 volte effettuando la media delle misurazioni. Da essa, mediante la formula `(posizione[i] - posizione[i-1]) / (tempo[i] - tempo[i-1])`, è stata calcolata la velocità. Per puntualizzare quando osserviamo i valori di una velocità, per esempio al 6° grado, il dato rappresenta la velocità media che il motore ha sostenuto tra il 3° e il 6° grado. Per questo al grado 0 la velocità è pari a 0 mentre al 540°, angolo di stop del motore, la velocità è diversa da 0 in quanto media sostenuta tra il 537° e 540° grado. I dati del vettore sono stati salvati su file e, successivamente, scaricati dal brick. Ad una prima valutazione i valori presentavano oscillazioni dovute a imprecisioni di misurazione. I risultati sono stati rielaborati mediante la formula `datoFiltrato[i] = 0.25*dato[i-1] + 0.5*dato[i] + 0.25*dato[i+1]`, filtro che, visionando i grafici, ha dato una risposta che segue più fedelmente la media delle oscillazioni dei dati originali.

2.4.4 Risultati in assenza del PID

Prima di osservare i dati relativi all'azione del PID proponiamo la visione dei dati ottenuti avendo disattivato questo controllo con il metodo `regulateSpeed(false)`. Suggeriamo questa osservazione per capire, prima di tutto, cosa accade in assenza del PID.
In figura 2.14 sono rappresentate le velocità rilevate nei casi in cui al motore siano applicati tre momenti diversi e rispettivamente pari a: 3.3Ncm, -3.3Ncm e 0. Si può notare come, in questo caso, non vi sia alcun controllo correttivo essendo il sistema del tipo ad anello aperto e quindi predittivo. Infatti tramite il metodo `calcPower`, una volta impostata la velocità con `setSpeed`, viene calcolato a priori il relativo valore di potenza (da 0 a 100%) che dovrebbe azionare il motore alla velocità voluta. Questo valore viene impostato al motore attraverso la relativa porta e rappresenta la variabile `u(t)`. Quando è attivo il PID agisce periodicamente come correzione su questo valore, aumentandolo o diminuendolo in base all'uscita rilevata. Essendo invece il controllore disattivato, la stima iniziale viene mantenuta tale qualsiasi sia la variazione dell'uscita del sistema. Si nota come, nei tre casi, il sistema non risenta di azioni correttive fornendo uscite molto diverse, dove nessuna delle quali rispecchia la velocità richiesta di 700 gradi al secondo. Per questo si invita ad evitare, salvo casi particolari, di disattivare il PID non avendo risultati apprezzabili in termini di rispetto della velocità impostata.

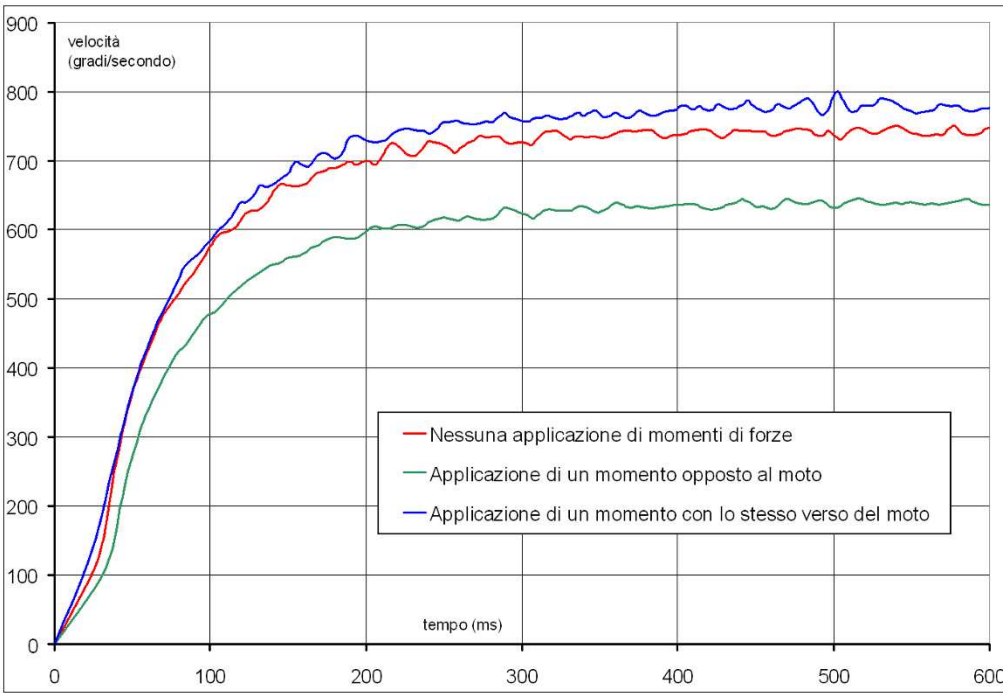


Figura 2.14: Valori di uscita del sistema senza l'azione del PID

2.4.5 Comportamento in caso di interventi nell'ingresso

Per ingresso del sistema intendiamo, come già mostrato nelle sezioni precedenti, la funzione `r(t)`. Una variazione dell'andamento di essa si può verificare con un cambio della velocità impostata al motore. Nell'esempio presentato osserviamo il comportamento in caso di avvio del moto dell'attuatore dove la velocità, derivata dalla variabile di riferimento `r(t)`, passa da 0 a 700 gradi al secondo.
Nella figura 2.15 possiamo osservare il comportamento delle funzioni `r(t)`, `y(t)`, `e(t)` e `u(t)` nel caso in cui la funzione *Smooth Acceleration* sia stata disattivata. La figura 2.16 rispecchia invece i dati rilevati con la suddetta funzione attiva.
Si nota subito come nel primo caso si verifica un'ampia sovraelongazione della velocità al di sopra di quella desiderata. Questo fatto è dovuto all'incremento iniziale dell'errore. Nel secondo caso con lo *Smooth Acceleration* attivo l'errore viene notevolmente limitato evitando il suddetto fenomeno. Qui la funzione `r(t)` è stata scelta con un andamento esponenziale che, come si vede, rispecchia i valori di accelerazione che il motore può assumere nei primi 120ms. Con *Smooth Acceleration* disattivo invece il calcolo lineare di `r(t)` introduce un notevole errore iniziale. Questo porta il controllore ad aumentare il valore della variabile `u(t)` ponendola, come si nota dal grafico, al valore massimo pari a 100% per un discreto lasso di tempo. Tutto ciò si ripercuote in un aumento della velocità oltre a quella richiesta fintantoché l'errore non si annulla.

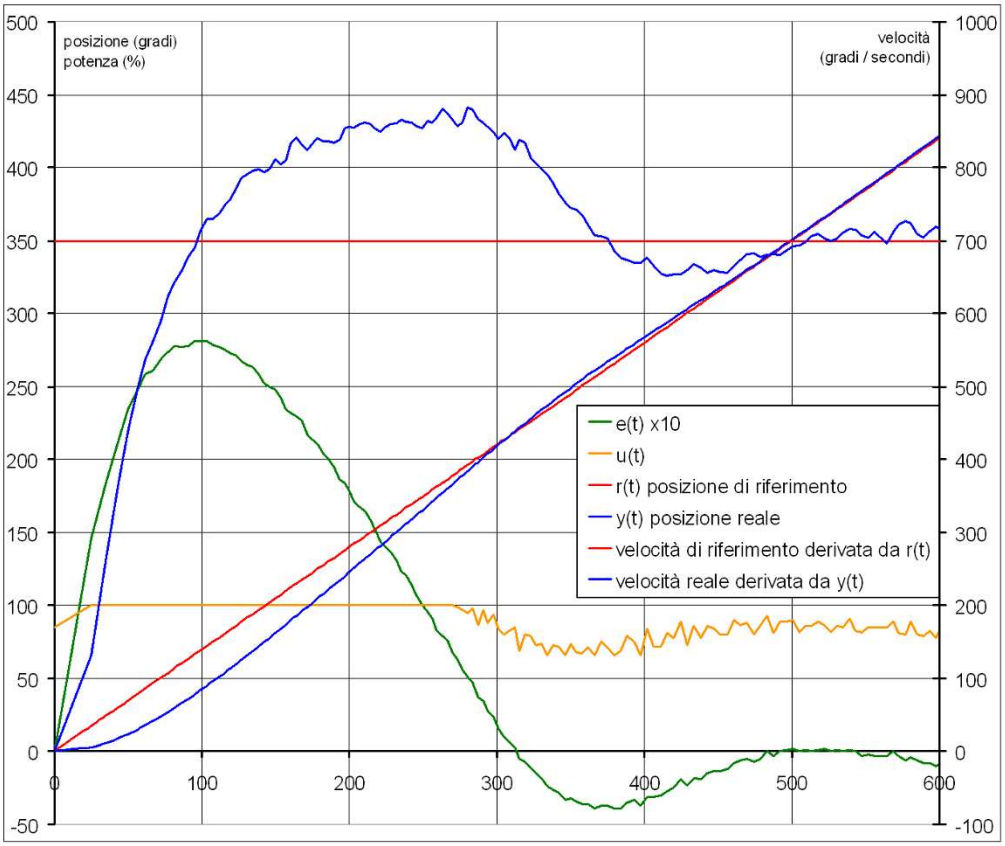


Figura 2.15: Risultati senza la funzione *Smooth Acceleration*

Successivamente esso assume valore negativo, fatto che si ripercuote sulla potenza e quindi sulla velocità ponendola minore al valore di 700 gradi al secondo. Questo andamento altalenante dei dati rilevati continua nel tempo tendendo comunque a convergere al valore di velocità desiderato, assumendo un carattere simile ad una funzione sinusoidale smorzata. In entrambi i grafici si può notare un'alternanza dei risultati ma, come è visibile, la funzione *Smooth Acceleration* ne limita di molto gli scostamenti.
I dati hanno evidenziato molti benefici della funzione *Smooth Acceleration*. Perché allora disporre di due modalità per il calcolo dei valori della variabile di riferimento `r(t)`? Se si ha la necessità di avere la massima prontezza nei transitori e il minimo tempo di reazione alle variazioni dell'ingresso, la modalità *Smooth Acceleration* non fa al caso nostro. Si può notare come, nel primo grafico, si raggiunga il valore di velocità prefissato dopo meno di 100ms mentre nel secondo caso è necessario un tempo doppio. È comunque curioso notare come un errore di anche solo qualche grado si ripercuota sulla velocità. Ad esempio, nel secondo grafico che rappresenta i valori rilevati con *Smooth Acceleration* attivo, prima dei 100ms la velocità reale è superiore a quella impostata. L'errore quindi assume un valore negativo che subito, intorno a 100ms, si

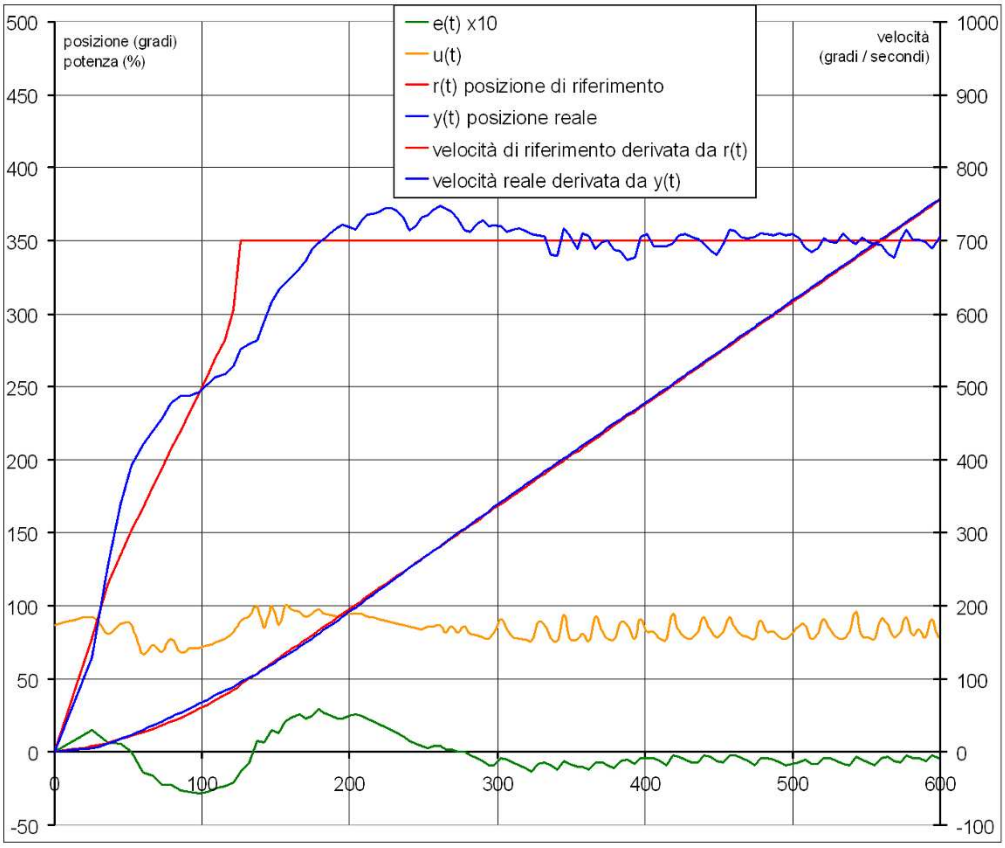


Figura 2.16: Risultati con la funzione *Smooth Acceleration* attiva

ripercuote sulla velocità spostando la curva reale sotto quella impostata. Attorno ai 150ms questa tendenza si inverte. Si noti che queste correzioni sono dovute ad un errore pari ad un massimo di 3 gradi, valore quindi molto ridotto.

2.4.6 Comportamento in caso di interventi nell'uscita

Con l'uscita del sistema intendiamo `y(t)`. Una variazione di questa funzione può denotare l'intervento di forze esterne che vanno a interferire con il moto creando momenti opposti o favorevoli alla rotazione del motore. La figura 2.17 rispecchia il comportamento dell'attuatore nel caso in cui ad un tempo t_0 intervenga una forza che crea un momento opposto al moto e pari a 3.3Ncm.
Tali dati sono stati raccolti sfruttando lo stesso sistema utilizzato nelle altre sezioni di questo capitolo. Qui però il motore è stato portato alla velocità voluta e durante la sua rotazione è stata applicata una massa al filo che viene avvolto dal motore, creando così

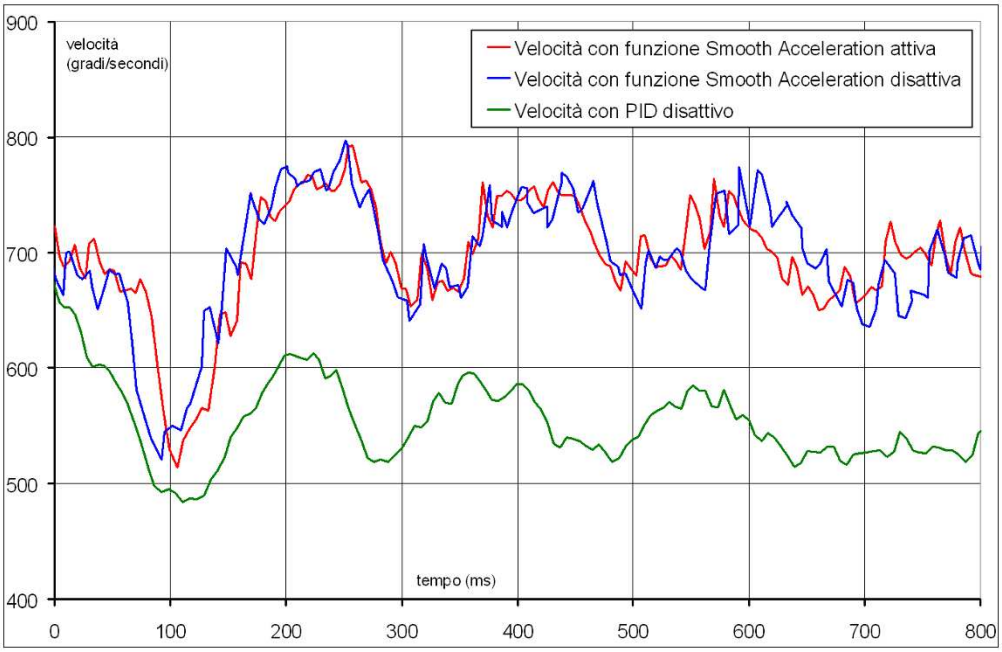


Figura 2.17: Valori di uscita dal sistema con e senza PID o Smooth Acceleration

un momento opposto al moto. Assumiamo che nel grafico prima di t = 0 il motore viaggiasse costantemente a 700 gradi al secondo. Dopo t = 0 la massa viene rilasciata. Notiamo subito che nel caso il PID sia disattivato la velocità decresce stabilizzandosi ad un valore inferiore a quello impostato. Infatti senza il PID l'evolversi dell'uscita non è monitorato al fine di apportare correzioni al sistema. Oltre a ciò notiamo che i dati rappresentanti la velocità con e senza *Smooth Acceleration* hanno pressoché lo stesso andamento. Giustifichiamo questo ricordando che la funzione *Smooth Acceleration* agisce in caso di variazione della variabile di riferimento `r(t)`. Quindi, come si evidenzia nel grafico, questa funzione non ha influsso in un tal contesto. *Smooth Acceleration* agisce sull'evoluzione del sistema per i primi 120ms misurati dal momento in cui si ha una variazione dell'andamento della funzione di ingresso `r(t)` e non di `y(t)`.

2.4.7 Comportamento in caso di interventi nell'ingresso e nell'uscita

Osserviamo ora come si comporta il motore nel caso in cui debba effettuare una partenza sostenendo un momento τ diverso da 0. Per avere un metro di misura compariamo i risultati ottenuti con quelli presentati nella sezione 2.4.5.
Come si nota nella figura 2.18 la modalità *Smooth Acceleration* risponde egregiamente alle condizioni impostate. Le tre linee, che rappresentano i valori di uscita nelle tre

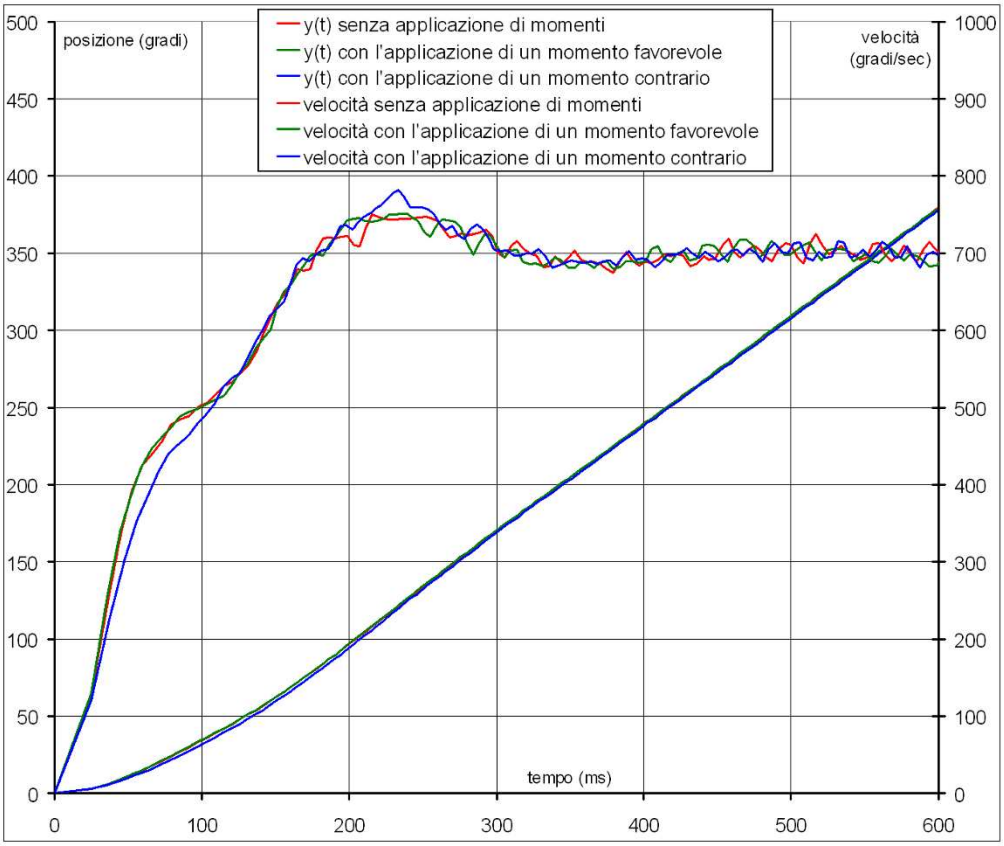


Figura 2.18: Applicazione di diversi momenti con Smooth Acceleration

diverse situazioni di carico, hanno andamento pressoché sovrapponibile. Ma è sempre così? Come abbiamo verificato nella sezione 2.3 il motore ha ovviamente dei limiti. Nel caso vi si applichi un momento favorevole al moto il PID reagisce abbassando i valori di uscita della variabile `u(t)` cercando di mantenere la velocità in prossimità di quella impostata. In questo caso si ha più margine di intervento che nell'eventualità in cui venga applicato un momento opposto al moto. Osservando il grafico possiamo con certezza affermare che risultati simili sono ottenibili finché la velocità sostenibile dal motore, con il momento applicato, è maggiore almeno del 10% rispetto quella impostata. Affermiamo ciò in base alla limitata sovraelongazione che si verifica con la funzione *Smooth Acceleration* pari a circa il 10% del valore della velocità impostata. Ci si può aiutare per questi calcoli con i risultati ottenuti nella sezione 2.3.
Notiamo invece che, nella figura 2.19, con la funzione *Smooth Acceleration* disattiva in caso vi sia applicato un momento opposto, la curva è nettamente diversa da quella degli altri due risultati. Osservando la precedente figura 2.10 si deduce che la velocità massima sostenibile dal motore, con un momento τ opposto al moto e pari a 3.3Ncm, è di 800 gradi al secondo. I dati combaciano. Il PID in questo caso è soggetto a saturazione.

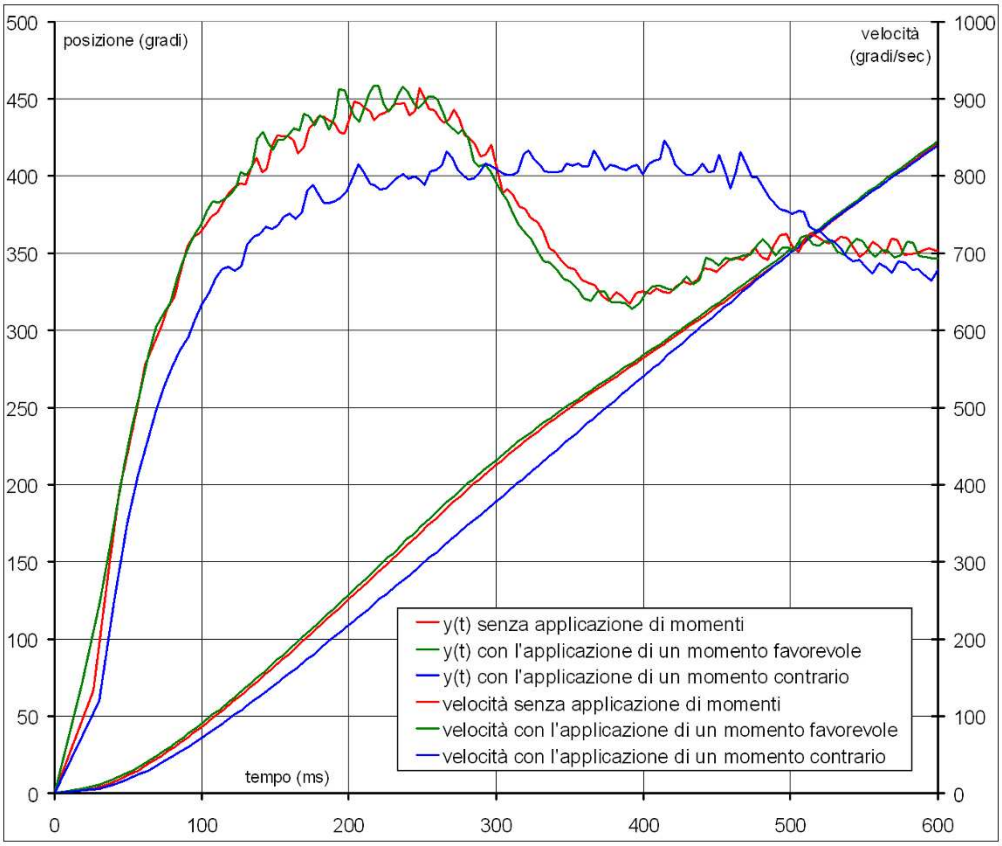


Figura 2.19: Applicazione di diversi momenti senza Smooth Acceleration

2.4.8 Saturazione

Questo fenomeno si verifica quando il PID impone una velocità più elevata di quella sostenibile dal motore. I casi di questo genere possono essere due:

- è stata impostata una velocità superiore a quella che il motore può sostenere, quindi l'errore cresce di continuo e l'uscita `y(t)` non sarà mai pari all'ingresso `r(t)`. Si ricade quindi in uno stato di saturazione continua che si può osservare nella figura 2.20;

- è stata impostata una velocità inferiore a quella che il motore può sostenere ma il PID ne richiede un picco al fine di annullare l'errore nel minor tempo possibile. Il controllore impone la variabile `u(t)` pari al 100% portando il motore a raggiungere la massima velocità sostenibile. Essendo quest'ultima superiore a quella richiesta, in un lasso di tempo più o meno lungo si otterrà l'annullamento dell'errore. Questo caso evidenzia quindi una saturazione momentanea che si può osservare nella figura 2.21.

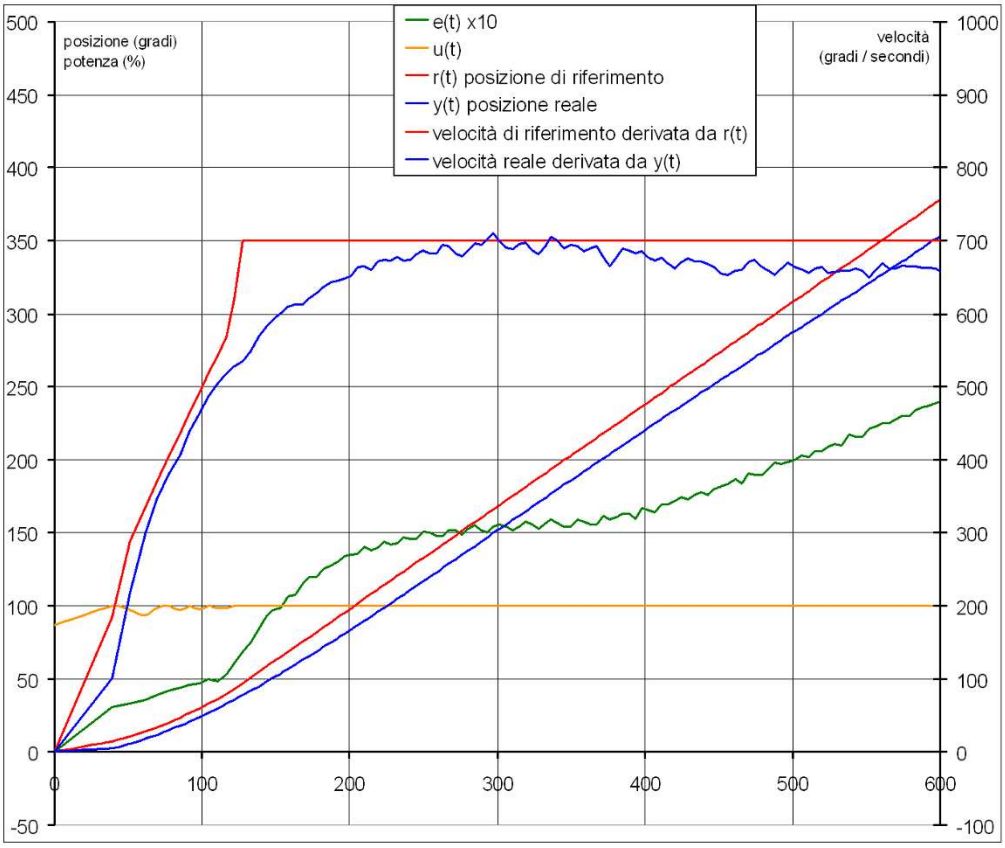


Figura 2.20: Saturazione continua, l'errore aumenta

Vogliamo specificare che tali riflessioni sono rivolte ad indurre gli utenti della guida ad effettuare sempre un preventivo studio del sistema costruito in modo da evitare di ricadere nelle problematiche presentate.

2.5 Lo Stop

Come già detto i comandi `rotate` e `rotateTo` fanno muovere il motore ad un angolo richiesto. Per rendere possibile ciò è necessaria una funzione che effettui lo stop del motore all'angolo preciso. Non è sufficiente il metodo `stop()` perché il motore, dopo aver ricevuto il comando, non si ferma istantaneamente. È necessario quindi capire quanto prima bisogna lanciare il comando `stop` per fermare il motore al preciso angolo

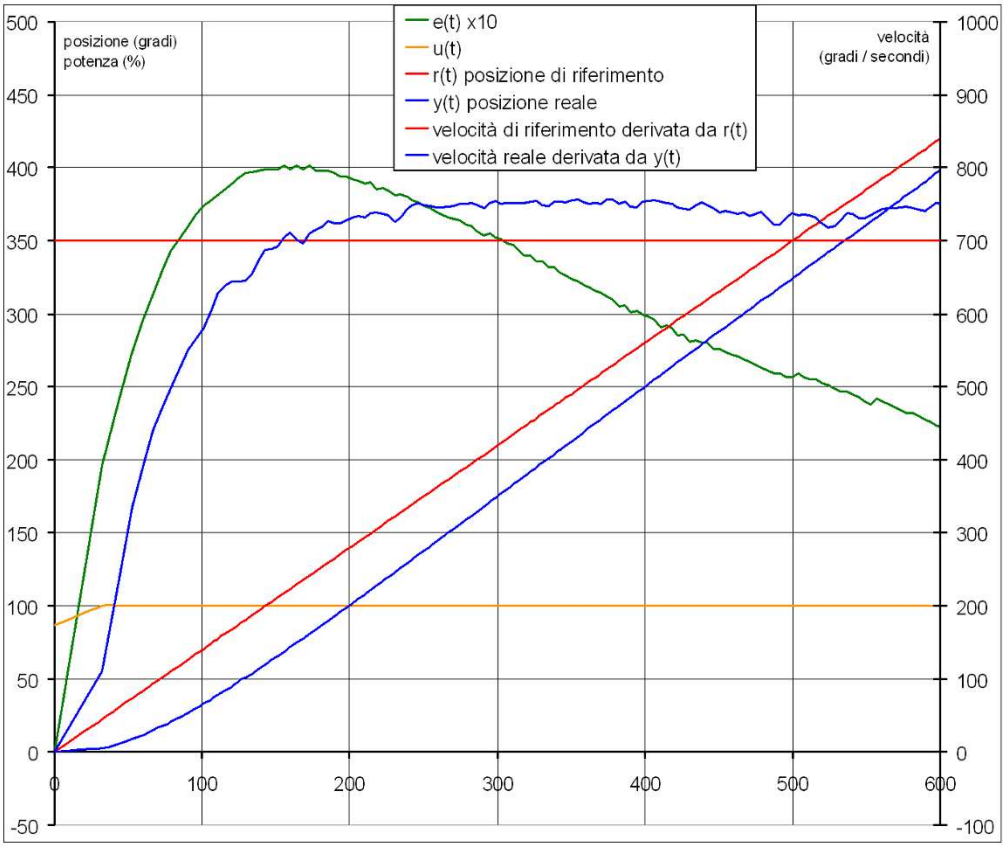


Figura 2.21: Saturazione momentanea, l'errore tende ad azzerarsi

richiesto. Ci viene per questo in aiuto il metodo `overshoot(int angle)`. Esso viene chiamato quando si effettua un `rotate` o un `rotateTo` e determina lo `stopAngle`, cioè il valore di anticipo rispetto l'angolo finale a cui si deve invocare il metodo `stop()`. Il calcolo dei valori della funzione `overshoot` è empirico, si basa cioè su prove effettuate dai programmatori della classe `Motor`. I risultati sono ottenuti partendo dalla velocità impostata e dall'entità della rotazione da effettuare. Essendo, come si può intendere, un calcolo predittivo, non tiene conto di eventuali influssi esterni sul moto dell'attuatore perciò, se a quest'ultimo viene applicato un momento τ di qualsiasi valore diverso da zero, il motore non si fermerà alla posizione voluta. Per ovviare a questo inconveniente i progettisti hanno inserito il metodo `stopAtLimit()` nella classe `Regulate`. Esso si avvale del metodo `stopAtAngle()` che lancia ripetutamente il comando `stop()` fintantoché il motore non raggiunge una velocità inferiore a 100 gradi al secondo. Successivamente il metodo `stopAtLimit()` esegue la seguente porzione di codice:

```
while ( k < 40)
{
```

```
  error = _limitAngle - getTachoCount();
  if (error == e0)
  {
    t1++;
    if( t1 > 20)
    {
      pwr += 10;
      t1 = 0;
    }
  }
  else
  {
    t1 = 0;
    if( error == 0) pwr = _brakePower;
    e0 = error;
  }
  if(error < -1)
  {
    _mode = BACKWARD;
    setPower(pwr);
    k = 0;
  }
  else if (error > 1 )
  {
    _mode = FORWARD ;
    setPower(pwr);
    k = 0;
  }
  else
  {
    _mode = STOP;
    port.controlMotor (0, STOP);
    k++;
  }
  Delay.msDelay(1);
}
```

L'uscita dal ciclo avviene quando il motore resta in un intorno di un grado dall'angolo desiderato per almeno 40ms. L'attuatore è quindi considerato fermo se si trova a ± 1 dal punto desiderato e con una velocità inferiore ai 50 gradi al secondo. Analizziamo quindi qual'è il comportamento dell'algoritmo sopra elencato. I passi fondamentali sono:

1. se il motore si trova nella stessa posizione precedente incrementa la variabile `t1`;

2. se la variabile `t1=20` indica che il motore non si muove da 20ms quindi aumenta di 10% la sua potenza. Questo controllo cerca di evitare che il motore si sposti ad una velocità inferiore a 50 gradi al secondo;

3. se il motore si trova ad un angolo superiore di 1° rispetto a quello prestabilito ritorna indietro col metodo `backward()` e azzera la variabile `k`;

4. se il motore si trova ad un angolo inferiore di 1° rispetto a quello prestabilito vai avanti col metodo `forward()` e azzera la variabile `k`;

5. se il motore si trova ad un angolo compreso tra $\pm 1°$ rispetto all'angolo prestabilito ferma il moto col metodo `stop()` e incrementa la variabile `k`;

6. attendi 1ms;

7. se la variabile `k=40` esci, altrimenti riparti dal punto 1.

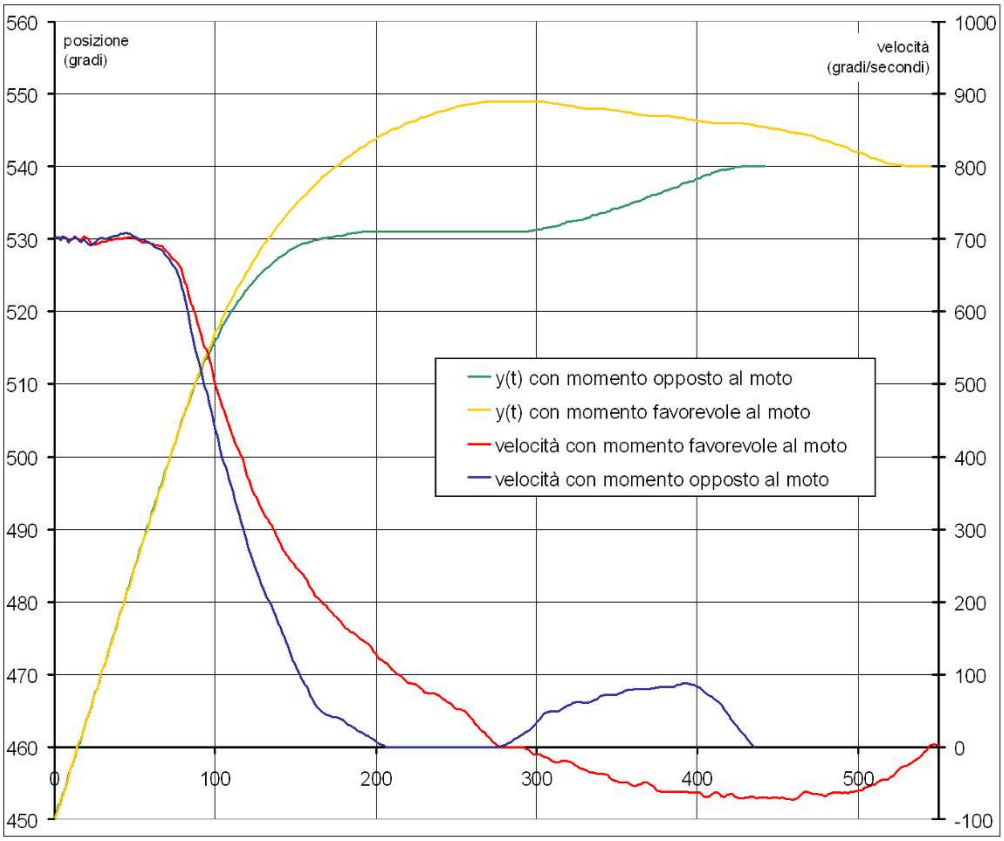


Figura 2.22: Stop con applicazione di due momenti opposti

Analizzati questi passi possiamo affermare che questo algoritmo cerca di portare nella posizione prefissata il motore nel caso esso si fermi in un punto errato. Esso corregge quindi gli eventuali errori di posizionamento che sorgono in presenza di disturbi esterni. Più precisamente, se il motore è soggetto ad un momento opposto al moto tenderà a fermarsi prima, viceversa tenderà ad andare oltre il punto prefissato. L'algoritmo corregge questi errori riavviando il movimento del motore nel verso opportuno e, in più, nel caso il momento applicato abbia un modulo elevato, aumenta la potenza del motore

al fine di vincere la resistenza applicata alla rotazione.
Nella figura 2.22 possiamo osservare il comportamento del motore che deve fermarsi all'angolo 540. I dati sono relativi a due test effettuati applicando una forza che crea un momento positivo e una che ne crea uno negativo. Come si osserva, se il momento è opposto al moto l'attuatore tende ad arrestare la sua rotazione intorno al grado 530. Successivamente subentra l'algoritmo precedentemente presentato che porta il motore all'angolo 540.
Nel caso in cui il momento applicato sia concorde al verso di rotazione, il motore supera l'angolo 540 fermandosi al 550° grado da cui il metodo `stopAtLimit` lo porta nella posizione finale. Le velocità rispecchiano la variazione di posizione registrata. Imprecisioni nel loro andamento sono dovute al fatto che riguardano valori molto ridotti e calcolati a partire dallo spazio percorso registrato dall'encoder del motore che, ricordiamo, ha una risoluzione molto ridotta pari ad 1°.

Capitolo 3

I sensori

In questo capitolo intendiamo presentare i sensori compresi nel kit LEGO ® MINDSTORMS ® NXT e, oltre ad essi, il sensore giroscopico e di accelerazione prodotti dalla ditta HiTechnic. Non ci limiteremo solo ad elencare i metodi messi a disposizione, vogliamo anche dare una nutrita spiegazione del funzionamento dei sensori. Intendiamo soprattutto spiegare, nella maniera più semplice e chiara possibile, i fenomeni fisici che permettono il funzionamento di questi strumenti, continuando così il lavoro volto a dimostrare come nella robotica convergano le conoscenze acquisite nei diversi ambiti di studio. Proseguiremo poi presentando tre esperimenti che sono mirati a chiarire delle problematiche, evidenziate dagli studenti a lezione, sui possibili usi del sensore giroscopico e di accelerazione.

Innanzitutto vogliamo portare a conoscenza dei lettori che i programmatori delle classi hanno messo a disposizione quattro oggetti statici: `SensorPort.S1`, `SensorPort.S2`, `SensorPort.S3`, `SensorPort.S4` che rappresentano le quattro porte di ingresso. Ad un costruttore di un oggetto rappresentante un sensore viene fornito come parametro uno di essi al fine di identificare la porta a cui il suddetto sensore è collegato.

3.1 Touch

3.1.1 Principi di funzionamento

Il Touch Sensor è il sensore più semplice a disposizione. È un sensore analogico passivo e presenta su un suo lato un bottone. Esso consiste in un semplice contatto elettrico che ha due posizioni: aperto e chiuso. L'NXT, utilizzando il convertitore analogico-digitale, trasforma il segnale in ingresso dal sensore in un valore numerico che rispecchia il livello di tensione. Un valore fornito dall'ADC inferiore a 600 è interpretato come se la linea analogica fosse posta a massa, ed equivale quindi al bottone premuto. Per valori maggiori a 600 indicano il contrario. Questo semplice sensore ha molteplici

Figura 3.1: Touch Sensor

applicazioni. Può essere usato per controllare che il robot non stia impattando contro una parete, oppure come fine corsa per limitare i movimenti di un braccio o anche per azionare il robot sotto il comando dell'utente.

3.1.2 Metodi

- **`TouchSensor NomeOggetto=new TouchSensor(SensorPort.SY)`** dove Y = 1, 2, 3 o 4 in base alla porta a cui è collegato il sensore. In questo modo si dichiara un oggetto di tipo `TouchSensor`;
- **`isPressed()`**: ritorna un valore booleano pari a `true` se il bottone è premuto, `false` altrimenti.

3.2 Sound

3.2.1 Principi di funzionamento

Il sensore di suono non permette di registrare un segnale audio, ma di monitorare la pressione sonora dell'ambiente circostante, cioè il valore di SPL (sound pressure level). Con questo termine intendiamo la variazione di pressione rispetto alla condizione

Figura 3.2: Sound Sensor

di quiete causata da una perturbazione, ovvero da un'onda sonora. Il range di valori acquisibili, dai dati dichiarati della LEGO, arriva a 90dB. Il decibel (dB) è un valore che deriva da una funzione logaritmica che esprime il rapporto tra due dati confrontabili e perciò è adimensionale. Nel nostro caso il livello di pressione sonora è definito con la formula $\mathtt{SPL} = 20\ \log_{10}(p/p_0)$, dove p è la pressione attuale e p_0 quella di riferimento definita dallo standard ANSI pari a 20μPa. Ad ogni incremento di 6dB il valore di pressione sonora p raddoppia. Il livello sonoro fornito è espresso, per semplicità di comprensione, in percentuale. La LEGO rende nota anche una scala comparativa che indica la corrispondenza dei dati rilevati con quelli di comuni fonti sonore:

- **4-5%** pari ad una stanza silenziosa;
- **5-10%** valore registrabile se qualcuno parla nella stanza a distanza dal sensore
- **10-30%** rumore prodotto da una conversazione nei pressi del sensore
- **30-100%** livello di pressione sonora prodotta da un grido o da musica ascoltata a volume alto
- **100%** pari al rumore prodotto dal motore di un tosaerba.

Questo sensore fornisce due modalità di acquisizione dati denominate come *dB* e *dBA*. Esse sono selezionate modificando lo stato dei pin DIGIYI0 e DIGIYI1, dove con `Y` indichiamo la porta a cui è collegato il sensore. In questo caso queste due linee non hanno la funzione di acquisizione dati in formato digitale, bensì sono utilizzate, a livello logico alto o basso, per cambiare l'andamento del guadagno negli amplificatori interni al sensore. Per selezionare le due modalità vengono utilizzati i livelli logici rappresentati dalla tabella 3.1.
Veniamo quindi a conoscenza del fatto che le linee digitali, nel caso in cui la porta

Modalità attivata	DIGIYI0	DIGIYI1
dB	alto	basso
dBA	basso	alto

Tabella 3.1: Livelli logici utilizzati per identificare le due curve di guadagno

sia utilizzata in modalità analogica, possono servire per impostare diverse modalità di funzionamento nei sensori.
Identificato in che modo l'NXT comunica al sensore la selezione di una o dell'altra

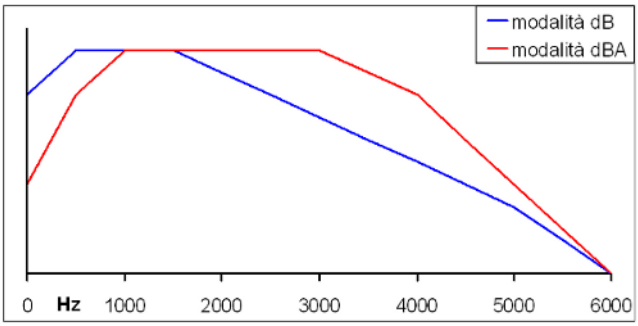


Figura 3.3: Sensibilità al variare della frequenza

modalità, non ci resta che spiegare il loro scopo. Come accennato, esse variano il guadagno degli amplificatori interni. Questa modifica permette di operare su due range di frequenza diversi. La modalità *dB* ha un range di acquisizione che si estende fino alle basse frequenze. Osservando dei dati pubblicati online e visibili in maniera comparativa nel grafico di figura 3.3, si nota che essa ha un guadagno molto elevato per frequenze basse fino al raggiungimento dei 2000Hz, soglia oltre la quale la curva di sensibilità cala costantemente. In modalità *dBA*, invece, vengono tagliate le frequenze al di sotto dei 1000Hz, fornendo un guadagno maggiore alle frequenze comprese tra i 3 e 6000 Hz. Questa modalità ha lo scopo di modulare il guadagno del microfono al fine di renderlo simile alla sensibilità dell'orecchio umano. Se proviamo le due modalità parlando di fronte al sensore con potenza diversa e via via crescente, ci accorgiamo che i dati ottenuti in *dBA* rispecchiano in miglior maniera i diversi livelli sonori a cui sottoponiamo il microfono. Non ci sentiamo comunque di trarre conclusioni senza aver effettuato nostre prove. Invitiamo quindi l'utente a testare le due modalità e selezionare quella che meglio risponde al range di suoni che esso vuole monitorare.

3.2.2 Metodi

- **`SoundSensor NomeOggetto=new SoundSensor(SensorPort.SY)`** dove Y = 1, 2, 3 o 4 in base alla porta a cui è collegato il sensore. In questo modo si dichiara un oggetto di tipo `SoundSensor` impostato di default in modalità *dB*;

- **`SoundSensor NomeOggetto=new SoundSensor(SensorPort.SY, boolean modalità)`** dove Y = 1, 2, 3 o 4 in base alla porta a cui è collegato il sensore. In questo modo si dichiara un oggetto di tipo `SoundSensor` con la possibilità di indicare a priori la modalità di utilizzo tra `dB` e *dBA*. Se la variabile `modalità` assume valore `true` viene selezionato l'utilizzo in *dBA*, mentre con `false` il sensore viene impostato in modalità *dB* come nel caso precedente;

- **`setDBA(boolean dba)`**: passando come parametro `true` attiva la modalità *dbA*, mentre se è `false` si passa in *dB*;

- **`readValue())`**: ritorna un intero rappresentante la potenza sonora registrata dal sensore. Il dato è espresso in percentuale.

3.3 Light

Figura 3.4: Light Sensor

3.3.1 Principi di funzionamento

Il sensore di luce misura il livello di luminosità dell'ambiente. Non è un sensore di colore. Esso fornisce un dato proporzionale alla lunghezza d'onda della luce che colpisce il fototransistor. Per immaginare il risultato ottenuto da questo sensore pensiamo a quando prendiamo una foto a colori e la convertiamo in "bianco e nero". Si ottiene una scala di grigi.
Abbiamo accennato che il componente che permette al nostro robot di "vedere" è detto fototransistor. Esso, in sostanza, è un transistor a giunzione bipolare che viene inscatolato in un contenitore trasparente in modo che la luce possa raggiungere la giunzione del collettore di base, variando così il guadagno del transistor in relazione alla lunghezza d'onda della luce che lo colpisce.
Il fototransistor utilizzato in questo sensore è il SFH 309 che ha lo stesso package di

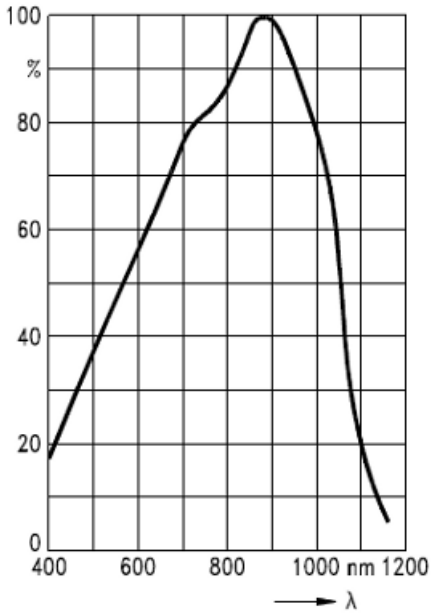


Figura 3.5: Risposta al variare della lunghezza d'onda che colpisce il sensore

un LED con un rivestimento trasparente. La casa madre dichiara che le variazioni della lunghezza d'onda della luce misurabili da questo strumento abbracciano i valori dai 380 nm ai 1180 nm. Lo spettro visibile, cioè quella parte dello spettro elettromagnetico che cade tra il rosso ed il violetto includendo tutti i colori percepibili dall'occhio umano, va indicativamente dai 380 ai 750 nm. Osserviamo dal grafico in figura 3.5 che il transistor risponde linearmente in questo intervallo, fatto molto utile per poter risalire, a partire dai valori di grigio misurati, al colore che il sensore sta "vedendo". Un colore più scuro risulta meno luminoso di uno chiaro alla vista del sensore. La variazione di luminosità dipende dalla frequenza dell'onda che colpisce il sensore.

Altra caratteristica utile di questo strumento è la sua direzionalità mostrata in figura

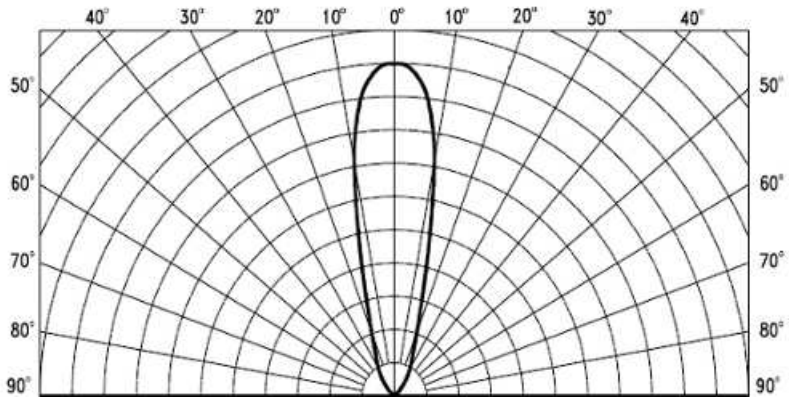


Figura 3.6: Risposta al variare della direzione della luce che colpisce il sensore

3.6. Ciò è indispensabile per avere la certezza che il dato rilevato riguarda il punto in cui dirigiamo il sensore e non è frutto della luce che arriva dall'ambiente circostante.
Nonostante queste premesse dobbiamo renderci conto che il sensore ha ovvi limiti dovuti a diverse variabili che entrano in gioco, come l'illuminazione esterna e la rifrazione della superficie. Andando per esempio a testare il sensore ponendolo a diverse distanze dallo stesso obbiettivo si otterranno valori diversi. Anche se abbiamo l'accortezza di porre alla stessa distanza di misura il Light Sensor e con lo stesso livello di illuminazione ambientale, esso fornirà misure diverse nel caso in cui osserviamo due superfici apparentemente con lo stesso colore, ma una liscia e una satinata.
Accantonando le difficoltà di acquisizione dei dati, continuiamo citando che il sensore dà la possibilità all'utente di selezionare se attivare o no il LED di illuminazione. Quindi possiamo vedere questo sensore come attivo o passivo a seconda della modalità in cui lo utilizziamo. Per attivare o disattivare il LED viene utilizzata la linea DIGIYI0 dove Y identifica la porta a cui è connesso il sensore.
I valori ottenuti sono posti in percentuale, ma se a noi interessa la variazione di una determinata porzione dello spettro, e quindi dei colori, possiamo impostare il valore minimo e massimo di rilevamento in modo che i risultati ottenuti abbiano una maggiore precisione all'interno del range voluto. Questo significa che il valore minimo impostato corrisponderà allo 0% e il massimo al 100%. Valori al di fuori del range sono identificati da risultati maggiori di 100 o minori di 0.

3.3.2 Metodi

- **`LightSensor NomeOggetto=new LightSensor(SensorPort.SY)`**
  dove Y = 1, 2, 3 o 4 in base alla porta a cui è collegato il sensore. In questo modo

si dichiara un oggetto di tipo `LightSensor` che utilizza l'illuminazione riflessa del LED rosso;

- **`LightSensor NomeOggetto=new LightSensor(SensorPort.SY, boolean floodlight)`** dove Y = 1, 2, 3 o 4 in base alla porta a cui è collegato il sensore. In questo modo si dichiara un oggetto di tipo `LightSensor`, con la possibilità di indicare a priori se attivare o no il LED. Se la variabile `floodlight` assume valore `false`, il LED viene lasciato spento utilizzando dunque la luce ambientale, mentre con `true` il LED viene attivato come nel caso precedente;

- **`setFloodlight(boolean floodlight)`** è un metodo implementato dell'interfaccia `LampLightDetector` e da la possibilità di attivare o disattivare il LED passando come valore rispettivamente `true` o `false`;

- **`setFloodlight(Colors.Color color)`** è un metodo implementato dell'interfaccia `LampLightDetector` e permette di attivare il LED del colore specifico passato come paramentro. Se il colore esiste, e quindi il LED si attiva, questo metodo ritorna il valore `true`, altrimenti `false`. Ovviamente `true` si otterrà solamente se il parametro passato al metodo corrisponde al valore rosso, cioè `Colors.Color.RED`;

- **`getFloodlight()`** è un'altro metodo implementato dell'interfaccia `LampLightDetector` che ritorna un oggetto di tipo `Colors.Color` rappresentante il colore del LED attivo. Ovviamente questo metodo torna solo il valore rappresentante la luce rossa, cioè `Colors.Color.RED` quando il LED è attivo. In alternativa il metodo ritorna `Colors.Color.NONE` che indica l'assenza di una emissione luminosa;

- **`getLightValue()`** metodo implementato dell'interfaccia `LightDetector` che fornisce un intero rappresentante il valore, in percentuale, rilevato dal sensore. Esso è calibrato e normalizzato. Con calibrato intendiamo che il dato è rilevato in un range definito da due parametri, rappresentati dal valore minimo e massimo. Per normalizzato intendiamo che il valore 0 corrisponde al valore dell'estremo inferiore, mentre 100 a quello dell'estremo superiore. Se l'estremo superiore e quello inferiore coincidono il metodo ritornerà in qualsiasi caso 0;

- **`readValue()`** effettua una chiamata al metodo `getLightValue()` ritornandone il valore. Presenta quindi il medesimo funzionamento;

- **`getNormalizedLightValue()`** metodo implementato dell'interfaccia `LightDetector` che ritorna un intero con valori da 0 a 1024. Esso rappresenta il dato vero e proprio fornito dal sensore e quantizzato dall'ADC dell'NXT senza normalizzazioni o riduzioni di scala. I progettisti della classe `LightSensor` dichiarano che all'incirca un valore pari a 145 rappresenta il nero mentre 890 il bianco;

- **`readNormalizedValue()`** effettua una chiamata al metodo `getNormalizedLightValue()` ritornandone il valore. Presenta quindi il medesimo funzionamento;
- **`calibrateLow()`** assume come estremo inferiore il valore rilevato dal sensore nel momento in cui il metodo viene lanciato. Questo dato corrisponderà quindi allo 0% nei metodi `readValue()` e `getLightValue()`;
- **`calibrateHigh()`** assume come estremo superiore il valore rilevato dal sensore nel momento in cui il metodo viene lanciato. Questo dato corrisponderà quindi al 100% nei metodi `readValue()` e `getLightValue()`;
- **`setLow(int low)`** assume come estremo inferiore il valore passato come parametro. Questo dato corrisponderà quindi allo 0% nei metodi `readValue()` e `getLightValue()`;
- **`setHigh(int high)`** assume come estremo superiore il valore passato come parametro. Questo dato corrisponderà quindi al 100% nei metodi `readValue()` e `getLightValue()`;
- **`getLow()`** metodo implementato dell'interfaccia `LightDetector` che ritorna un intero rappresentante il valore dell'estremo inferiore impostato coi metodi sopra citati;
- **`getHigh()`** metodo implementato dell'interfaccia `LightDetector` che ritorna un intero rappresentante il valore dell'estremo superiore impostato coi metodi sopra citati;
- **`isFloodlightOn()`** è un metodo implementato dell'interfaccia `LampLightDetector` e ritorna `true` o `false` nel caso in cui il LED sia rispettivamente attivo o spento.

3.4 Ultrasonic

3.4.1 Principi di funzionamento

I sensori ad ultrasuoni utilizzano onde meccaniche sonore le cui frequenze sono superiori a quelle mediamente udibili da un orecchio umano. La frequenza convenzionalmente utilizzata per discriminare onde soniche da onde ultrasoniche è fissata in 20 kHz, ma i sensori utilizzano un range da 40 a 250KHz. Lo stesso termine ultrasuono

Figura 3.7: Ultrasonic Sensor

indica chiaramente ciò che è al di là (ultra) del suono, indicando con suono le frequenze udibili dall'orecchio umano. Questi sensori sono utilizzati in un'ampia varietà di applicazioni come, ad esempio, per rilevare la velocità di un oggetto, la prossimità di un ostacolo, ottenere misure di distanza ed altre svariate applicazioni. Per effettuare questi rilevamenti i dispositivi emettono una raffica di ultrasuoni, che se colpiscono un oggetto nelle vicinanze, si riflettono e ritornano al sensore. Il sistema di controllo misura il tempo che impiega l'eco di ritorno e calcola la distanza dell'oggetto conoscendo la velocità degli ultrasuoni nel mezzo in cui essi si propagano. I sensori ad ultrasuoni perciò non sono intrusivi, nel senso che non necessitano di toccare il loro obiettivo e sono in grado di rilevare alcuni oggetti che possono essere di difficile individuazione da sensori visivi, come per esempio oggetti trasparenti. D'altra parte però, le loro misurazioni sono molto sensibili alla temperatura e all'angolo con cui il bersaglio viene colpito.
I sensori possono essere composti da uno o due elementi. Nel primo caso lo stesso componente emette l'onda e poi si mette in ascolto dell'eco. Nel secondo caso un elemento ha la funzione di emettitore del segnale e l'altro rimane in ascolto dell'eco. Ovviamente in quest'ultimo caso è possibile emettere più impulsi nel tempo di andata e ritorno del segnale.
Nel NXT il sensore ad ultrasuoni è utilizzato per misurare la distanza di un oggetto. Se esso è in movimento, la frequenza del segnale ricevuto differisce da quella del segnale trasmesso a causa dell'effetto Doppler ma, se l'oggetto è statico, la distanza può essere calcolata come $L = (v\Delta t(\cos(\alpha)))/2$. Si può osservare qunto detto nella formula precedente nella figura 3.8, specificando che con Δt identifichiamo il tempo che si interpone tra la trasmissione e la ricezione, v rappresenta la velocità degli ultrasuoni, e 2α è l'angolo tra l'onda in andata e quella di eco. Se l'oggetto è ad una distanza relativamente maggiore a quella che separa il ricevitore dall'emettitore, l'angolo α assume un valore prossimo allo 0. Questo implica che il coseno di quest'angolo è 1 calcolando così la distanza dell'oggetto come $L = (v\Delta t)/2$.
Bisogna considerare che i sensori ad ultrasuoni, come accennato prima, sono molto

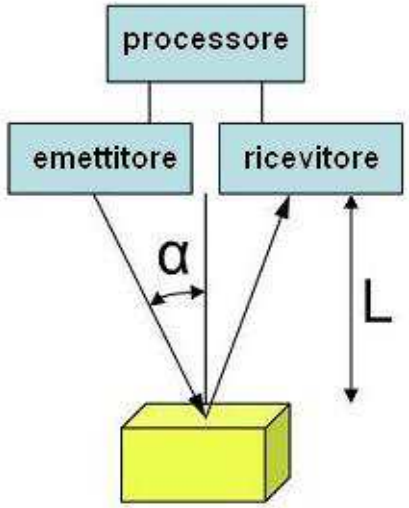


Figura 3.8: Calcolo della misura

sensibili alle variazioni ambientali. Elenchiamo alcune problematiche che influiscono sulla misurazione:

- La **temperatura dell'ambiente** influisce sul calcolo della distanza in quanto la velocità del suono dipende dalla temperatura del mezzo trasmissivo. La velocità del suono in aria si può determinare dalla formula $v(T)=0,3261\sqrt{(1+(T/273))}$, dove $v(T)$ è misurata in m/s e T è la temperatura dell'aria in °C. Bisognerebbe quindi tarare lo strumento in base alla temperatura ambientale rilevata all'istante di misurazione;
- L'**umidità dell'aria** influisce sul segnale attenuandolo, riducendo così la distanza che l'ultrasuono può raggiungere;
- La **superficie dell'obbiettivo** deve essere perpendicolare al trasmettitore perchè il ricevitore riceva l'eco riflessa. Oggetti rotondi sono quindi più facilmente identificabili in quanto riflettono sempre perpendicolarmente il segnale. Quando il target è un oggetto piatto, per garantire che il segnale venga ricevuto, la sua angolazione rispetto al sensore non deve superare un determinato intervallo calcolabile sperimentalmente;
- Il **rumore di fondo** può interferire modificando le onde sonore inviate;
- I sensori ad ultrasuoni hanno in genere una "**zona morta**" immediatamente davanti a loro e in cui gli oggetti non possono essere rilevati in quanto l'onda viene riflessa e ritorna al ricevitore prima che questo sia attivato. Esso viene infatti spento per un determinato tempo al momento della trasmissione onde evitare che capti il segnale emesso scambiandolo per un'eco;

- Alcuni **materiali** sono più assorbenti rispetto agli altri e riflettono meno gli ultrasuoni. Questo aumenta quindi l'attenuazione del segnale riducendo la distanza massima a cui gli oggetti vengono rilevati;

- Se **più sensori coesistono** all'interno dello stesso ambiente uno di essi può ricevere il segnale di un altro interpretandolo come un'eco fornendo così misurazioni errate.

Il trasmettitore e il ricevitore montati sul sensore dell'NXT sono dei trasduttori piezoelettrici. Ma come funzionano? In breve sulla superficie di un disco di materiale piezoelettrico vengono metallizzate due armature. In trasmissione ad esse viene applicata una tensione che sottopone il materiale tra di esse ad un campo elettrico, facendolo così deformare per effetto piezoelettrico inverso. Queste deformazioni producono delle vibrazioni che si trasmettono all'aria. In ricezione, l'onda ultrasonica deforma il materiale che produce cariche per effetto piezoelettrico diretto le quali possono essere rilevate come differenza di tensione tra le due armature. Si può osservare in figura 3.9 lo schema di un trasduttore piezoelettrico.
I due trasduttori montati nel sensore Lego sono il TCT40-12S2 e il TCT40-12F2, ri-

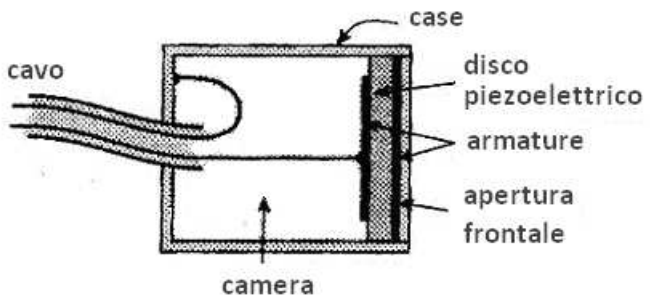


Figura 3.9: Disegno di un trasduttore piezoelettrico

spettivamente il trasmettitore e il ricevitore. Dai dati dichiarati, il primo emette una potenza sonora pari a 110dB e il secondo ha una sensibilità minima di -70dB.
Come si può notare dal grafico in figura 3.10 il microfono presenta un ampio cono di ascolto. Questo è utile nel caso in cui l'eco sia stata riflessa da una superficie inclinata, consentendo al ricevitore di captare comunque il segnale. Risulta però svantaggioso nel caso in cui siano montati più sensori vicini, potendo l'uno captare le trasmissione dell'altro.
Ovviamente i sensori sopra descritti hanno il solo ruolo di trasduttori nell'emettere e ricevere il segnale. Il coordinamento dei tempi di invio, di attesa e il calcolo della distanza sono effettuati da un μcontrollore denominato *eSC015*. Esso presenta una frequenza di clock pari a 4Mhz e, oltre all'elaborazione dei dati, si occupa anche della trasmissione mediante l'interfaccia I^2C. Come citato infatti nel capitolo 1, il sensore

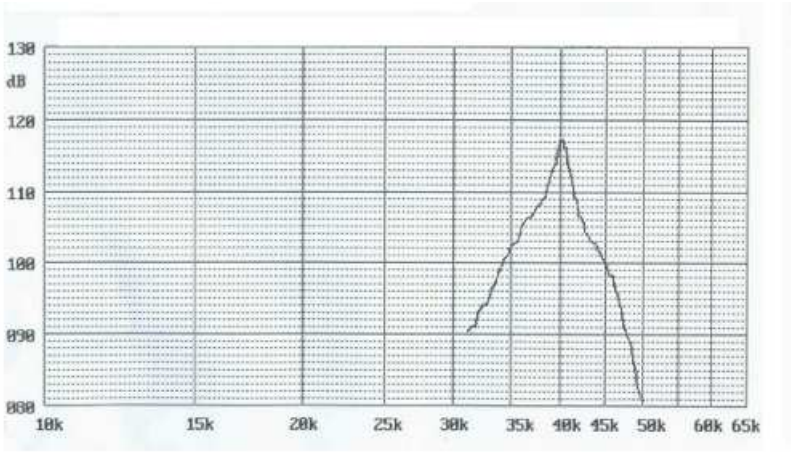
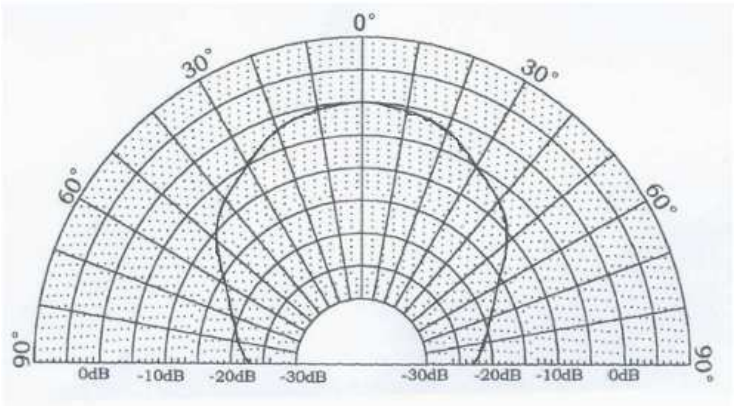


Figura 3.10: Sopra SPL in trasmissione e sotto sensibilità in ricezione



ad ultrasuoni non è analogico ma del tipo digitale e trasmette i dati all'NXT tramite la suddetta interfaccia. Osserviamo quindi nella tabella 3.2 i comandi di lettura inviata dal brick al sensore ad ultrasuoni e il valore ottenuto come risposta da quest'ultimo. I byte 0, 1, 2 identificano i dati trasmessi al sensore in esadecimale. Il primo rappresenta l'indirizzo della periferica e il secondo l'area di memoria interessata. Il terzo byte inviato indicherebbe il comando da scrivere ma, essendo queste richieste effettuate in lettura, è impostato ad un valore standard. La colonna denominata *Risposta* identifica il valore fornito dal sensore. I valori tra apici rappresentano un'interpretazione del significato del dato. Quelli in esadecimale corrispondono al dato vero e proprio scritto nell'area di memoria

La tabella 3.3 illustra invece i comandi di scrittura inviata dal brick al sensore ad ultrasuoni. Il primo byte rappresenta l'indirizzo della periferica, il secondo l'area di memoria interessata e il terzo il comando inviato da scrivere nella locazione indicata.

Comandi	Byte 0	Byte 1	Byte 2	Risposta
Leggi la versione	0x02	0x00	0x03	“V1.0”
Leggi l’ID di produzione	0x02	0x08	0x03	“LEGO”
Leggi il tipo di Sensore	0x02	0x10	0x03	“Sonar”
Leggi il valore di zero (di fabbrica)	0x02	0x11	0x03	0x00
Leggi l’unità minima (di fabbrica)	0x02	0x12	0x03	0x01
Leggi il divisore di scala (di fabbrica)	0x02	0x13	0x03	0x0E
Leggi l’unità di misura	0x02	0x14	0x03	“10E-2m”
Leggi l’intervallo di misura	0x02	0x40	0x03	“Intervallo in ms”
Leggi la modalità di funzionamento	0x02	0x41	0x03	“Modalità (0,1,2,3,4)”
Leggi il dato 0	0x02	0x42	0x03	“Dato 0”
Leggi il dato 1	0x02	0x43	0x03	“Dato 1”
Leggi il dato 2	0x02	0x44	0x03	“Dato 2”
Leggi il dato 3	0x02	0x45	0x03	“Dato 3”
Leggi il dato 4	0x02	0x46	0x03	“Dato 4”
Leggi il dato 5	0x02	0x47	0x03	“Dato 5”
Leggi il dato 6	0x02	0x48	0x03	“Dato 6”
Leggi il dato 7	0x02	0x49	0x03	“Dato 7”
Leggi il valore di zero (attuale)	0x02	0x4A	0x03	“Zero attuale”
Leggi l’unità minima (attuale)	0x02	0x4B	0x03	“Unità minima attuale”
Leggi il divisore di scala (attuale)	0x02	0x4C	0x03	“Divisore attuale”

Tabella 3.2: Comandi di lettura inviati dal brick e relativa risposta del sensore

3.4.2 Metodi

- **`UltrasonicSensor NomeOggetto=new UltrasonicSensor(SensorPort.SY)`** dove Y = 1, 2, 3 o 4 in base alla porta in cui è collegato il sensore. In questo modo si dichiara un oggetto di tipo `UltrasonicSensor`;

- **`getData(int register, byte [] buf, int len)`** questo metodo permette di leggere un’area di memoria, a partire dall’indirizzo indicato col parametro `register`, e di inserire il risultato in un vettore di byte identificato col nome `buf`. La variabile `len` indica la lunghezza del dato da acquisire. Il metodo

Comandi	Byte 0	Byte 1	Byte 2
Modalità off	0x02	0x41	0x00
Modalità ping	0x02	0x41	0x01
Modalità continuous	0x02	0x41	0x02
Modalità event capture	0x02	0x41	0x03
Modalità reset	0x02	0x41	0x04
Imposta l'intervallo di misura	0x02	0x40	"Intervallo (ms)"
Imposta il valore di zero (attuale)	0x02	0x4A	"Zero attuale"
Imposta l'unità minima (attuale)	0x02	0x4B	"Unità minima"
Imposta il divisore di scala (attuale)	0x4C	0x51	"Divisore attuale"

Tabella 3.3: Comandi di scrittura inviati dal brick

ritorna un intero pari a 0 se la trasmissione è andata a buon fine, un valore diverso altrimenti;

- **`sendData(int register, byte [] buf, int len)`** questo metodo permette di scrivere in un'area di memoria, a partire dall'indirizzo indicato col parametro `register`, i dati contenuti in un vettore di byte identificato col nome `buf`. La variabile `len` indica la lunghezza del dato inviato. Il metodo ritorna un intero pari a 0 se la trasmissione è andata a buon fine, un valore diverso altrimenti;

- **`getDistance()`** ritorna un valore intero da 0 a 254 che identifica la distanza misurata dal sensore. Un valore pari a 255 indica che il dato rilevato è fuori dal range di misurazione. Ci sono due modalità per l'acquisizione dei dati, *continuous* e *ping*. La misura non può essere effettuata in tempo reale in quanto è necessario inviare un impulso e aspettare il tempo necessario perché esso compia almeno due volte la distanza massima. La modalità *continuous* sopperisce a questo problema inviando periodicamente impulsi e aggiornando i dati nei registri del sensore. Una variabile memorizza l'ultimo dato scaricato dal sensore e, se la chiamata successiva al metodo `getDistance()` avviene in un tempo inferiore a quello necessario ad effettuare un'altra misurazione, il metodo non attende che il prossimo dato sia pronto ma ritorna il valore precedentemente misurato. Nella modalità `ping` viene invece inviato un treno di 8 impulsi consecutivi e, dopo di ciò, il sensore si ferma. Il valore che il metodo ritorna è relativo al primo impuso;

- **`getRange()`** metodo implementato dell'interfaccia `RangeFinder`. Esso effettua una chiamata al metodo `getDistance()` ritornando il valore ottenuto in

formato floating point;

- **`getDistances(int dist[])`** overloading del metodo `getDistance`. Il vettore di interi passato come parametro può essere della dimensione decisa dall'utente, però maggiore o uguale a 8. Il metodo inserisce nel vettore gli 8 elementi ottenuti dall'esecuzione di una misurazione in modalità `ping`. Essa infatti, come precedentemente detto, effettua un treno di 8 rilevazioni. Se il vettore ha una dimensione inferiore a 8 il metodo ritorna il valore -1, così come se il sensore non è impostato in modalità *ping*, necessaria per l'utilizzo di questa procedura. Il metodo ritorna 0 se la registrazione è andata a buon fine;

- **`ping()`** attiva la modalità *ping*. Questo metodo ritorna un intero pari a 0 se l'operazione è andata a buon fine, diverso altrimenti;

- **`continuous()`** attiva la modalità *continuous*. Questo metodo ritorna un intero pari a 0 se l'operazione è andata a buon fine, diverso altrimenti;

- **`off()`** imposta il sensore in *off*. In questo modo nessun segnale viene inviato o ascoltato dal sensore. Questo metodo ritorna un intero pari a 0 se l'operazione è andata a buon fine, diverso altrimenti;

- **`capture()`** attiva la modalità *event capture* in cui il sensore non emette ultrasuoni ma rimane in ascolto al fine di percepire se ci sono altri sensori nelle vicinanze;

- **`reset()`** resetta il sensore. Esso viene reimpostato ai valori di default e riprende la sua funzionalità in modalità *continuous*. Questo metodo ritorna un intero pari a 0 se l'operazione è andata a buon fine, diverso altrimenti;

- **`getFactoryData(byte data[])`** inserisce nel vettore le informazioni "di fabbrica". Il vettore `data` non può avere dimensione inferiore a 3 in quanto i dati inseriti sono lo zero, l'unità minima e il fattore di scala. Se il vettore non è della dimensione specificata il metodo ritorna il valore `-1`. In alternativa viene fornito un intero pari a 0 se l'operazione è andata a buon fine, diverso altrimenti;

- **`getUnits()`** ritorna un oggetto `string` contenente l'unità di misura. Il valore della stringa è `10E-2m` se l'operazione va a buon fine. Altrimenti viene fornita una stringa contenente 7 spazi.

- **`getCalibrationData(byte data[])`** inserisce nel vettore le informazioni contenute nelle aree di memoria 0x4A, 0x4B e 0x4C rappresentanti i dati di calibrazione. Essi corrispondono allo zero, all'unità minima e al fattore di scala attualmente impostati. Se il vettore non ha una dimensione maggiore o uguale a 3, il metodo ritorna il valore `-1`. In alternativa fornisce un intero pari a 0 se l'operazione è andata a buon fine, diverso altrimenti;

- **`setCalibrationData(byte data[])`** effettua l'operazione inversa del metodo precedente, inserendo nelle aree di memoria 0x4A, 0x4B e 0x4C i valori contenuti nel vettore e andando così ad aggiornare i dati di calibrazione attualmente impostati. Se il vettore non ha una dimensione maggiore o uguale a 3, il metodo ritorna il valore `-1`. In alternativa fornisce un intero pari a 0 se l'operazione è andata a buon fine, diverso altrimenti;
- **`getContinuousInterval()`** ritorna un `byte` contenente il valore impostato come intervallo di attesa tra due misurazioni effettuate con la modalità *continuous*. Se l'operazione non è andata a buon fine il valore fornito è -1;
- **`setContinuousInterval(byte interval)`** effettua l'operazione opposta del metodo precedente, rendendo possibile impostare l'intervallo di attesa tra due misurazioni effettuate con la modalità *continuous*. I valori accettati vanno da 1 a 15. Il metodo fornisce un intero pari a 0 se l'operazione è andata a buon fine, diverso altrimenti;
- **`getMode()`** questo metodo ritorna un `byte` che indica la modalità in cui il sensore è impostato. Più precisamente il valore 0 indica che il sensore è *off*, 1 che è impostato in modalità *ping*, 2 in *continuous*, 3 in *event capture*. Se l'operazione non è andata a buon fine il valore fornito è -1;

3.5 Acceleration

Figura 3.11: Accelerometer Sensor Hitechnic

3.5.1 Principi di funzionamento

L'accelerometro è un dispositivo elettromeccanico che misura l'accelerazione. Essa può essere dovuta a una forza statica, come la forza di gravità che ogni giorno con la stessa intensità "tira" ai nostri piedi, o potrebbe essere dovuta a forze dinamiche che variano nel tempo in maniera anche brusca. Ma per quali scopi sono utilizzati questi strumenti? Essi sono principalmente tre:

- ricavare la velocità e lo spostamento che un corpo, per esempio un aereo, ha sostenuto nel tempo;
- identificare l'inclinazione;
- per misurare vibrazioni o urti a cui un corpo è soggetto;

Nella maggior parte degli accelerometri il principio di funzionamento è il medesimo: si basa sulla rilevazione dell'inerzia che esercita una massa quando viene sottoposta ad una accelerazione. La massa viene sospesa ad un elemento elastico, mentre un qualche tipo di sensore ne rileva lo spostamento rispetto alla struttura fissa del dispositivo. In presenza di un'accelerazione, la massa, che è dotata di una propria inerzia, si sposta dalla propria posizione di riposo in modo proporzionale all'accelerazione rilevata, coerentemente con la seconda legge del moto di Newton (`F = ma`). Ovviamente si ha la necessità che l'elemento elastico abbia una costante di allungamento `K` lineare rispetto alla forza `F`. La relazione sarà del tipo `F = Ks` dove `s` è lo spostamento. Il sensore trasforma questa variazione di posizione in un segnale elettrico acquisibile dai sistemi di misura.
Quello che distingue le varie tipologie di accelerometro è il modo con cui viene effettuata la rilevazione della posizione. Le tipologie più note sono:

- **Piezoelettrico** in cui la rilevazione dell'accelerazione viene effettuata grazie alla compressione di un cristallo piezoelettrico. La massa viene sospesa sul cristallo piezoelettrico, che costituisce sia il sensore, che l'elemento elastico. In presenza di un'accelerazione la massa comprime il cristallo, il quale, genera un segnale elettrico proporzionale alla compressione e rilevabile come differenza di potenziale. Essendo un cristallo, il sensore è relativamente immune alle variazioni di temperatura, di contro presenta però una sensibilità bassa dovuta alla costante elastica `K`. Un'ulteriore problematica che questi sensori presentano è l'impossibilità di misurare un'accelerazione statica. Come detto, il cristallo genera un segnale elettrico proporzionale alla compressione, ma se essa permani sul cristallo, il segnale generato tende a dissiparsi dopo un breve periodo. Questi accelerometri trovano impiego in applicazioni dove si richiede di rilevare accelerazioni dinamiche come quelle che si generano nelle vibrazioni e negli shock meccanici anche di elevato valore. Questa è infatti una peculiarità di sensori piezoelettrici che resistono a valori di anche 1000g;
- **Capacitivo** in cui la rilevazione è effettuata misurando la variazione della capacità di un condensatore. La massa in questo caso costituisce un'armatura del

condensatore, mentre la seconda armatura è fissata alla struttura del sensore. La massa è sospesa su un materiale semirigido che funge da dialettrico tra le armature. La compressione o l'allungamento di quest'ultimo incidono variando la capacità del condensatore così costruito e quindi variando la tensione ai capi delle due armature. Sono strumenti abbastanza precisi e non risentono particolarmente delle variazioni delle condizioni ambientali;

- **LVDT** (Linear Variable Differential Transformer): esso sfrutta, per la rilevazione dell'accelerazione, una massa che costituisce un nucleo ferromagnetico. Essa è sospesa su elementi elastici e scorre in un canale avvolto da spire che costituiscono un solenoide. La massa, per induzione, crea una variazione del flusso di carica nel solenoide, creando così una differenza di potenziale ai capi di esso;
- **MEMS** (Micro-Electro-Mechanical Sensors): il principio di base del funzionamento che sta dietro l'accelerometro MEMS è lo spostamento di una piccola massa incisa nella superficie di silicio del circuito integrato. Essa è sospesa da piccole "travi" che agiscono come una molla mentre l'aria racchiusa all'interno del circuito integrato funge da ammortizzatore. Lo spostamento della massa viene solitamente calcolato sfruttando il principio dell'accelerometro capacitivo grazie a dei sensori detti a piatti paralleli. Questa tecnologia permette di creare dispositivi con ridotti costi e dimensioni che hanno ottenuto, negli ultimi anni, una diffusione su larga scala di questi oggetti.

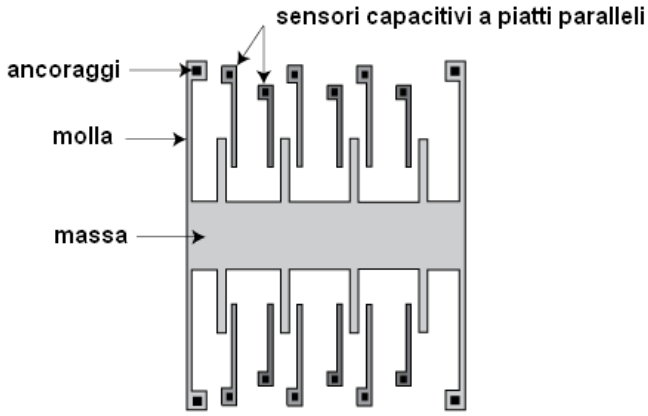


Figura 3.12: Rappresentazione dell'incisione di un accelerometro MEMS

L'accelerometro Hitechnic appartiene a quest'ultima categoria citata e da la possibilità di rilevare l'accelerazione presente sui 3 assi. Come affermato nel capitolo 1 questo è un sensore digitale che comunica col brick per mezzo dell'interfaccia I^2C. I valori ricavati dai sensori sono digitalizzati per mezzo di un convertitore A/D a 10bit, ma i registri di memoria e l'unità di trasferimento della connessione I^2C sono in byte. Per

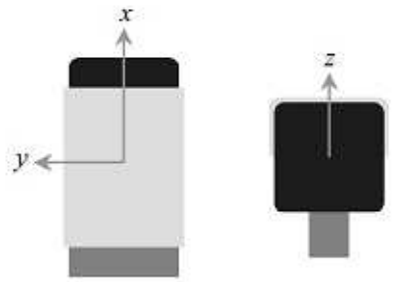


Figura 3.13: Orientamento degli assi dell'Accelerometer Sensor Hitechnic

questo i valori rappresentanti un singolo asse sono divisi in due registri, uno dei quali rappresenta gli 8bit più significativi e l'altro i rimanenti 2. Per ottenere il dato risultante è sufficiente effettuare uno shift di 2 bit e aggiungere gli altri 2. Preso `upper` per rappresentare gli 8bit più significativi e `lower` i rimanenti due, possiamo esprimere il dato come `accelerazione = upper*4 + lower`. I valori sono contenuti agli indirizzi di memoria indicati nella tabella 3.4.
Da preventive misurazioni effettuate con due sensori, abbiamo constatato che essi non

Contenuto	Indirizzo
Asse X upper	0x42
Asse Y upper	0x43
Asse Z upper	0x44
Asse X lower	0x45
Asse Y lower	0x46
Asse Z lower	0x47

Tabella 3.4: Indirizzo dei dati contenuti nell'Acceleration Sensor

forniscono gli stessi dati. Come esempio prendiamo la grandezza nota e costante dell'accelerazione dovuta alla forza di gravità. Da diverse misurazioni si nota che essa può corrispondere a valori diversi in sensori diversi, e addirittura tra assi diversi dello stesso sensore. Precisiamo che abbiamo registrato imprecisioni del 3% che possono essere attribuibili alla scarsa qualità dell'integrato. In un caso abbiamo registrato uno scostamento maggiore al 10% e riteniamo che esso possa essere dovuto ad accelerazioni troppo elevate che il sensore può aver subito e che possono aver alterato la costante elastica delle "molle" che sorreggono la massa. Bisogna tener conto che questi sensori hanno ovvi limiti è non è bene scuoterli in maniera troppo brusca. La casa costruttrice

indica una scala misurabile pari a $\pm$2G. Questo è il massimo valore misurabile ma non la massima accelerazione sostenibile dal sensore. Sicuramente il valore di sollecitazione massima è superiore, ma non è noto quale sia. Consigliamo quindi di trattare con cura questi strumenti onde evitare di arrecare danni alla struttura interna.

3.5.2 Metodi

Precisiamo che abbiamo trovato la classe `TiltSensor` proposta dal Lejos alquanto scarna, prevedendo solo i metodi di acquisizione dei 3 valori relativi ai 3 assi. Abbiamo perciò proposto una nostra classe, nominata `AccelSensor`[1], che funziona solamente col sensore della Hitecnich.

- **`AccelSensor NomeOggetto=new AccelSensor(SensorPort.SY)`** dove Y = 1, 2, 3 o 4 in base alla porta in cui è collegato il sensore. In questo modo si dichiara un oggetto di tipo `AccelSensor`;
- **`getXData()`** fornisce un intero rapresentante il valore letto dal sensore. Non ha un'unità di misura precisata, è solo proporzionale all'accelerazione misurata. I metodi `getYData` e `getZData` sono volti a fornire le letture degli altri assi;
- **`calibrateX(int max, int min)`** questo metodo permette di impostare il valore corrispondente alla forza di gravità registrata quando il relativo asse è puntato a terra o verso l'alto. Per ottenere i dati di taratura bisogna utilizzare la funzione `getData` che non apporta correzioni ai dati. Questa calibrazione è utili per normalizzare i valori ricevuti dai diversi assi del sensore, avendo come metro di paragone l'accelerazione[2] prodotta dalla forza di gravità. Di default i valori sono impostati a 200, ma abbiamo riscontrato come quasi nessun sensore rispetti questo valore. Per ottenere dati precisi dai metodi `getTilt`, `getAccel` e `getNormalizeData` è indispensabile impostare questi estremi, essendo soggettivi da sensore a sensore. Sono disponibili anche i metodi `calibrateY` e `calibrateZ` per gli altri assi;
- **`getXTilt()`** ritorna una variabile a doppia precisione che indica l'inclinazione[3] dell'asse in gradi. I metodi `getYTilt` e `getZTilt` hanno lo stesso scopo inerente agli altri assi;
- **`getXAccel()`** fornisce un intero rappresentante l'accelerazione in mm/sec^2. In esempio una lettura pari a 9810 rappresenta un'accelerazione pari a quella gravitazionale. Sono disponibili anche i metodi `getYAccel` e `getZAccel` per la lettura degli altri assi.

[1] Il codice della classe `AccelSensor` è disponibile in Appendice

[2] Si invita alla lettura della sezione 3.7 per un'approfondimento sul metodo di taratura

[3] Per l'approfondimento del calcolo e delle problematiche riscontrabili con l'uso di questo medoto di invita alla lettura della sezione 3.7

- **`getXNormalizeData()`** ritorna una variabile a doppia precisione che indica l'accelerazione in G volte la forza di gravità. Come per gli altri metodi anche per questo esistono le varianti `getYNormalizeData` e `getZNormalizeData`;

3.6 Gyro

Figura 3.14: Gyro Sensor Hitechnic

3.6.1 Principi di funzionamento

Un giroscopio è un dispositivo utilizzato principalmente per la misurazione della velocità angolare di 1, 2 o 3 assi. Molto spesso uno di questi strumenti a 3 assi è affiancato ad un un accelerometro a 3 assi per fornire ben 6 gradi di libertà, realizzando così un sistema di navigazione inerziale[4]. I giroscopi sono realizzati sfruttando diversi principi fisici. Ne presentiamo i principali due.

Giroscopio rotante: rappresenta la realizzazione classica di questo strumento che si basa sul principio della conservazione del momento angolare. Esso afferma che il momento angolare totale di un sistema è costante in grandezza e in direzione, se il momento risultante delle forze esterne agenti sul sistema è nullo. Questi giroscopi sono

[4]Per un semplice sistema di questo tipo di invita alla lettura della sezione 3.9

costituiti da un disco rotante su un asse, detto *spin*, e montato con una serie di sospensioni cardaniche. Ogni sospensione cardanica offre al disco rotante un elemento aggiuntivo in libertà di rotazione attorno ad un asse. Le sospensioni cardaniche consentono al rotore di girare, senza applicare alcuna coppia netta esterna sul giroscopio. Quindi, grazie a questa realizzazione e al principio della conservazione del momento angolare, fino a quando il giroscopio è in rotazione, l'asse di *spin* manterrà un orientamento costante anche se la struttura esterna viene fatta ruotare.
Ma in che modo si misura la velocità di rotazione? Essa viene calcolata grazie al mo-

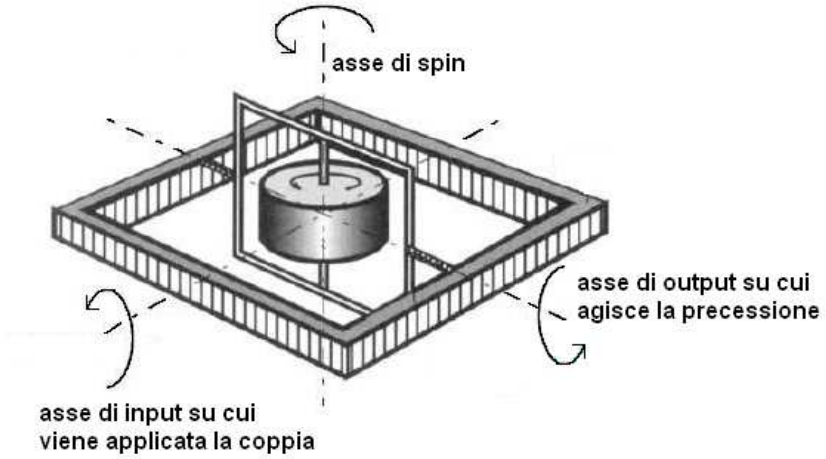


Figura 3.15: Raffigurazione di un giroscopio rotante con i suoi assi

to di precessione. In questo caso, rappresentato in figura 3.15, la massa ruota attorno all'asse di *spin*. Se una coppia esterna viene applicata in una direzione perpendicolare all'asse di rotazione, identificato come asse di *input*, il sistema non è più sottoposto ad un momento esterno nullo. Si crea quindi un momento di precessione perpendicolare sia al momento di ingesso, che all'asse di rotazione. Esso farà ruotare l'asse di *spin* attorno all'asse indicato come di `output` finché i momenti esterni non si annullano. Per cui un momento applicato attorno all'asse di ingresso produce precessione attorno all'asse di uscita, facendolo ruotare di un valore proporzionale a quello di ingresso. Se il momento di ingresso è dovuto all'inerzia che oppone il giroscopio ad una rotazione, misurando periodicamente la variazione angolare dell'asse di uscita si rileva un fattore proporzionale alla velocità angolare a cui il sistema è sottoposto.
I giroscopi rotativi sono usati principalmente in applicazioni di stabilizzazione. La presenza di parti in movimento come cardani e rotori implica che questi strumenti sono sottoposti ad usura. Un'ulteriore complicazione di questa realizzazione sta nel fatto che, essendo uno strumento realizzato da organi meccanici con movimenti complessi, non è possibile una implementazione miniaturizzata. I giroscopi rotanti, essendo però resistenti ad urti e vibrazioni intense, sono per lo più utilizzati in ambienti militari e navali dove le dimensioni fisiche ridotte non sono la prima caratteristica cercata.

Giroscopio Vibrante: sfrutta la forza di Coriolis per misurare l'accelerazione. Essa è una forza apparente, cioè che non è rilevabile da un sistema inerziale. Osservando dall'alto un disco che ruota ad una velocità ω, lanciamo una sfera che parta dal centro e diretta radialmente verso l'esterno. In assenza o quasi di attrito l'oggetto ci sembrerà muoversi di moto rettilineo uniforme. Di contro, se ci troviamo sul disco, vedremo che la traiettoria della sfera è curva.
Questo effetto è dovuto al fatto che, anche se la sfera si muove di moto rettilineo unifor-

Figura 3.16: Traiettorie della massa nei due sistemi di riferimento

me, man mano che essa si avvicina all'esterno la velocità tangenziale del disco aumenta. A parità di velocità angolare, espressa in gradi al secondo, la velocità tangenziale di due cerchi di raggio diverso sarà maggiore in quello col raggio maggiore. Questo perchè lo spazio compiuto nella rotazione di un grado è pari a `2rΠ/360`. Analizzando punto per punto il movimento della sfera, ci rendiamo conto che essa compie un disallineamento dal moto rettilineo crescente man mano che ci si avvicina al bordo del disco. Tale forza che, vista da un osservatore solidale al disco agisce sulla sfera, è detta di Coriolis ed agisce ortogonalmente al moto rettilineo della sfera. Il giroscopio vibrante sfrutta proprio la rilevazione di questa grandezza per determinare la velocità angolare del sistema. La forza di Coriolis è infatti espressa come `Fc = -2mω×v` dove `v` e `m` sono rispettivamente il vettore velocità e la massa della sfera, mentre ω è la velocità angolare del sistema. Nel nostro caso possiamo sostituire il prodotto interno tra i vettori `v` e ω con una moltiplicazione tra i loro moduli ed un fattore pari a ± 1, dipendente dal verso concorde o meno. Essendo il valore della massa e la sua velocità fissati al momento della realizzazione del sistema, rilevando la forza di Coriolis possiamo ricavare la velocità angolare come `ω = -Fc/(2mv)`.
Ma come è realizzato questo strumento di misura integrato? Esso è realizzato con la già citata tecnologia MEMS (Micro-Electro-Mechanical-Sensor) grazie alla quale viene incisa una massa di peso `m` su un circuito di silicio. Grazie all'applicazione di una differenza di potenziale variabile nel tempo essa è fatta vibrare ad una nota velocità `v`. Questo corrisponde al sistema inerziale. Esso è montato su sospensioni e, mediante un sensore capacitivo realizzato con piatti paralleli, è possibile misurare la variazione

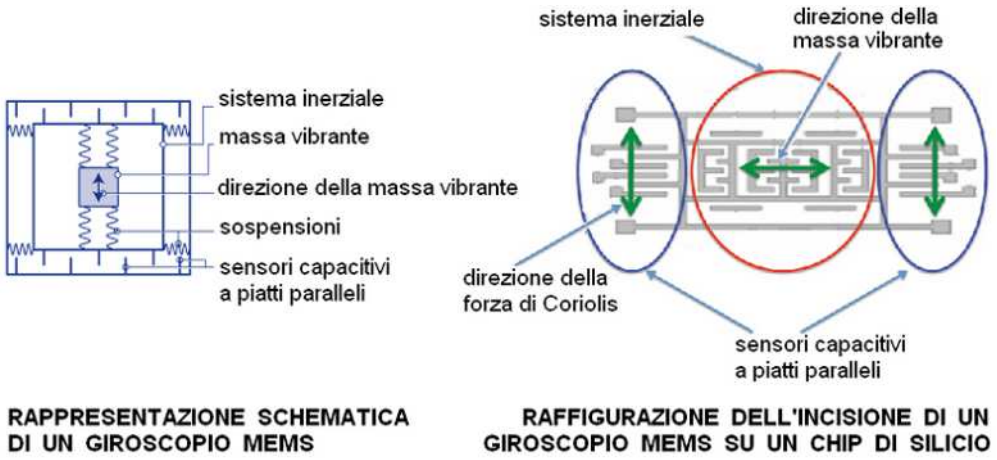


Figura 3.17: Raffigurazione di un giroscopio MEMS e schema della sua incisione

della capacità dovuta allo spostamento delle armature. Si può osservare questo sistema in figura 3.17. Misurando questo spostamento indotto dalla forza di Coriolis e applicando la formula citata prima, è possibile determinare la velocità angolare a cui questo strumento di misura è sottoposto.
Si osservi, come rappresentato nella figura 3.18, che a parità di direzione e velocità

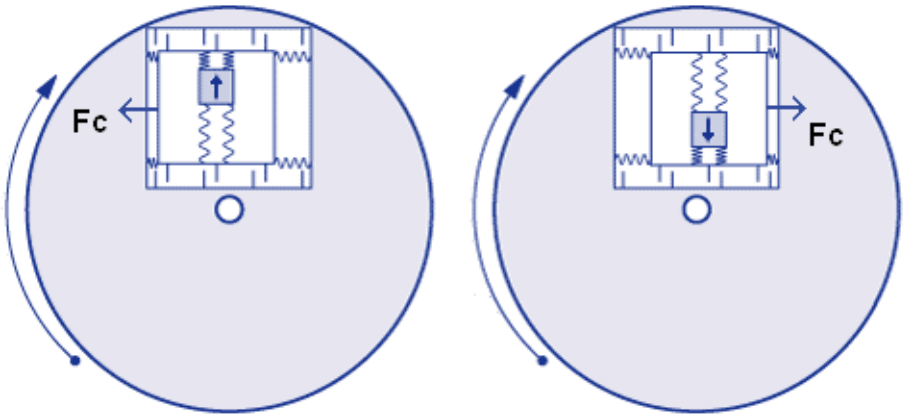


Figura 3.18: Direzione della forza di Coriolis

di rotazione, quando la massa si muove verso l'alto la forza di Coriolis genera uno spostamento del sistema inerziale verso sinistra. Viceversa, quando la massa si muove verso il basso, la forza generata crea uno spostamento verso destra. La forza di Coriolis dipende quindi non solo dall'intensità della velocità angolare applicata al sistema, ma anche dalla direzione del vettore velocità applicato alla massa vibrante.

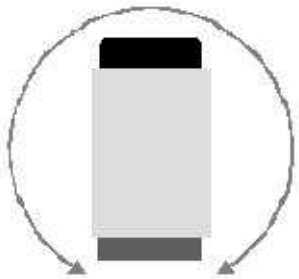


Figura 3.19: Orientamento dell'asse del GyroSensor HiTechnic

Nel sensore prodotto dalla ditta Hitechnic il giroscopio è del tipo MEMS. Come detto, esso è un sensore analogico e, dai dati dichiarati dalla casa produttrice, la velocità angolare massima registrabile è pari a 360°/secondo. Esso fornisce la possibilità di rilevare la velocità angolare lungo un solo asse il cui orientamento è identificato in figura 3.19.

3.6.2 Metodi

- **`GyroSensor NomeOggetto = new GyroSensor(SensorPort.SY)`** dove Y = 1, 2, 3 o 4 in base alla porta in cui è collegato il sensore. In questo modo si dichiara un oggetto di tipo `GyroSensor`;
- **`readValue()`** ritorna un valore intero rappresentante la velocità angolare misurata dal giroscopio;
- **`setOffset(int offset)`** imposta l'offset rappresentante il valore di velocità 0. Di default è impostato un valore di offset pari a 600. Questo significa che, al dato letto dalla porta, viene sottratto il suddetto valore, per cui tutte le letture minori di 600 assumono valore negativo. La possibilità di modificare l'offset è utile nel caso in cui il metodo `readValue` fornisca un valore diverso da zero, anche quando il sensore non è sottoposto a rotazione. In questo caso è necessario usare questo metodo per impostare un offset pari a 600 più il valore rilevato a sensore fermo.

3.7 Calcolo dell'inclinazione mediante il sensore di accelerazione

Tra i metodi che abbiamo messo a disposizione con la classe `AccelSensor` vi è `getTilt` che fornisce un valore in gradi rappresentante l'inclinazione dell'asse richiesto. Ma come viene calcolato questo dato e che problematiche si presentano?
L'inclinazione viene ricavata misurando l'intensità della componente della forza di gra-

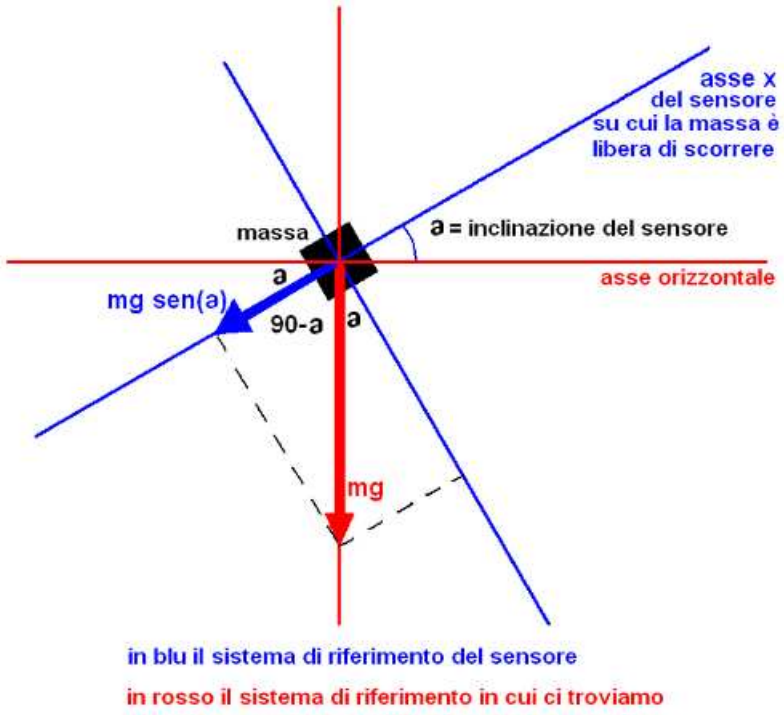


Figura 3.20: Diagramma vettoriale delle forze agenti sulla massa dell'accelerometro

vità che agisce sulla massa dell'asse del sensore. Nella figura 3.20 si può osservare come viene scomposta la forza di gravità portando come esempio il diagramma vettoriale delle forze agenti sulla massa appartenente all'asse x del sensore. Essa, per costruzione, è vincolata a muoversi parallelamente a quest'asse. Il modulo della componente della forza di gravità che agisce con una direzione parallela a `x` è pari a `mg sen(`∂`)`, dove ∂ è l'angolo che l'asse del sensore forma con il piano orizzontale del sistema di riferimento in cui ci troviamo. Quindi indicando con `x` il valore letto dal sensore possiamo determinare il suddetto angolo con la formula ∂ `= arcsen(x/mg)`, dove `arcsen` è la funzione inversa del seno definita per valori compresi tra ± 1. Per applicare questa formula dobbiamo quindi conoscere il valore della forza di gravità. Sappiamo che esso è universalmente riconosciuto, salvo leggere fluttuazioni, pari a 9,81m/s^2. Ma nel nostro accelerometro questo valore a cosa corrisponde? Riprendiamo il metodo `calibrate`

che abbiamo definito per la classe `AccelSensor`. Richiedavamo di passare come parametro due variabili, `max` e `min`, i cui valori dovevano essere ottenuti mediante il metodo `getData` e ponendo il sensore con l'asse prima rivolto verso l'alto e successivamente verso il basso. Se diamo uno sguardo alla figura 3.20 ed immaginiamo di elevare l'asse orizzontale del sensore fino a sovrapporlo a quello verticale del sistema, notiamo che l'angolo ∂ tende a 90°. sin(∂) corrisponde quindi a sin(Π/2) cioè 1. Ne segue che il valore misurato dal sensore in questo caso è `mg`. Se rivolgiamo l'asse `x` verso il basso applicando lo stesso ragionamento otteniamo `-mg`.
Per questo è importante calibrare l'accelerometro prima di utilizzare i metodi `getTilt`

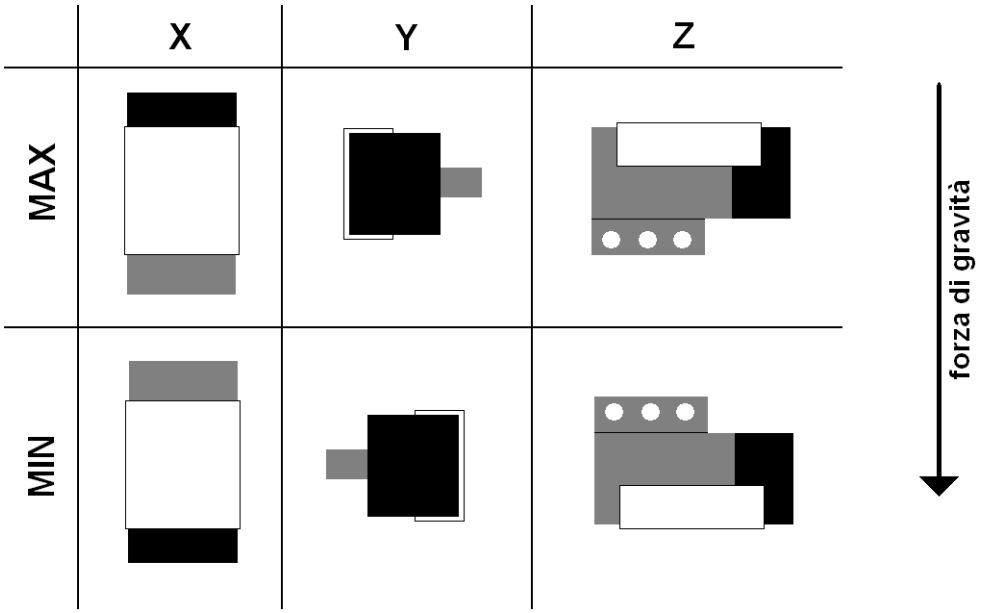


Figura 3.21: Posizioni da far assumere al sensore di accelerazione in taratura

in quanto si acquisisce il valore della forza di gravità indispensabile per il corretto funzionamento di queste procedure e che molto spesso varia da sensore a sensore. Ovviamente l'operazione di calibrazione deve essere effettuata per tutti gli assi. Come aiuto per svolgere questa procedura invitiamo ad osservare la figura 3.21 che rappresenta in che posizione porre il sensore per acquisire i valori `max` e `min` dei relativi assi.
Concentriamo ora la nostra attenzione sulla precisione di questa misura. Assumiamo che il sensore sia calibrato per un valore di `max` e `min` pari a $\pm$200 quindi `mg=200`. In figura 3.22 è rappresentato l'andamento della funzione `arcsin`. Abbiamo posto in ascissa i valori rilevati dal sensore e in ordinata il valore fornito dalla funzione `arcsin(x/mg)`. Notiamo come fino ad un'inclinazione pari a $\pm$45° si possa assumere la funzione come lineare, mentre per valori superiori essa tende ad assumere un andamento esponenziale. Questo fatto si ripercuote sulla precisione del nostro strumento. Per capire ciò abbiamo inserito nel grafico un insieme di dati indicato come *errore*. Es-

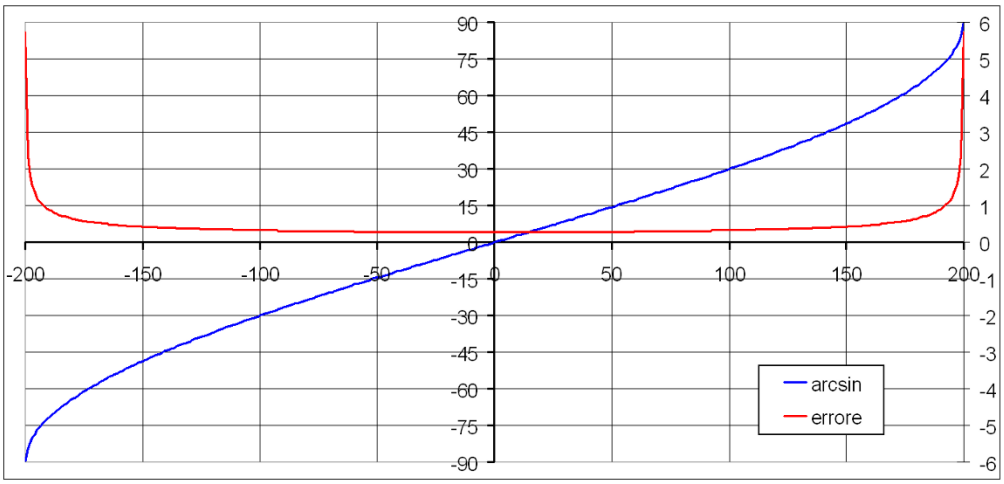


Figura 3.22: Funzione arcsin

so rappresenta la massima variazione del valore fornito della funzione `arcsin(x/mg)` se `x` varia di ± 1. Per esempio un valore fornito dal sensore pari a 100 corrisponde, nel nostro caso, ad un angolo di 30°. Se `x` varia di + o - 1 l'inclinazione corrisponderà a 30,33° nel primo caso, 29,66° nel secondo. Effettuiamo una seconda prova. Se il valore misurato dal sensore è 192 l'inclinazione corrisponde a 73,74°. Variando di + o - 1 il valore di `x` l'inclinazione corrisponderà a 74,8° nel primo caso, 72,75° nel secondo. Si nota che la precisione del sensore varia e diminuisce man mano che ci avviciniamo ai $\pm 90°$. Infatti negli esempi portati sopra, a 30° si ha uno scarto di $\pm 0{,}3°$ mentre a 73,74° lo scarto sale a $\pm 1°$. Ora seguendo l'andamento della linea rossa possiamo osservare che a 90° si ha uno scarto di quasi 6°. Si consiglia perciò di utilizzare il sensore su valori compresi ad almeno $\pm 60°$ altrimenti cambiare asse. In esempio a ciò si può utilizzare l'asse `z` per aumentare la precisione della misurazione dell'asse `x` con il seguente codice:

```
x=getXTilt();
if(x<=45 && x>=-45)
        xTilt=x;
else
{
  if(x>45)
    xTilt=90-getZTilt();
  else
    xTilt=-90-getZTilt();
}
```

Ovviamente questo accorgimento funziona solo se il sensore si muove su un unico piano identificato dall'asse `x` e `z`, cioè se `getYTilt()`$\cong$`0` per ogni misurazione effettuata. È doveroso indicare che tutti i calcoli e le considerazioni precedentemente presenta-

ti valgono solo se il sensore è sottoposto ad un moto quasi stazionario. Vibrazioni, variazioni improvvise della velocità, ma anche spostamenti con moto uniforme, introducono accelerazioni che andrebbero a sommarsi a quella gravitazionale variando i risultati. Ricordiamo sempre che il sensore di accelerazione misura la somma delle accelerazioni a cui è sottoposto.
Concludiamo indicando, come si può notare nel codice presentato sopra, che è conveniente evitare molte chiamate ai metodi di lettura, in quanto essi impiegano circa 10ms per la risposta. Il valore dipende comunque dalla modalità impostata nel sensore. Un frammento di codice come il seguente può necessitare di almeno il 50% di tempo in più, pur avendo il medesimo risultato del precedente:

```
if(getXTilt()<=45 && getXTilt()>=-45)
        xTilt=getXTilt();
else
{
  if(getXTilt()>45)
    xTilt=90-getZTilt();
  else if(getXTilt()<-45)
    xTilt=-90-getZTilt();
}
```

3.8 Spazio, velocità, accelerazione

Con questo esperimento vogliamo raggiungere due obbiettivi:

- esprimere il legame che intercorre tra spazio, velocità e accelerazione;
- valutare la precisione del sensore giroscopico e di accelerazione.

Per effettuare ciò dobbiamo:

- costruire una struttura che ci permetta di acquisire dati in tempo reale dai due sensori;
- confrontare i dati tra loro per osservare il legame che intercorre tra le funzioni rappresentanti lo spazio, la velocità e l'accelerazione;
- confrontare i dati rilevati dai sensori con i dati calcolati in maniera teorica al fine di constatare la precisione degli strumenti.

3.8.1 Realizzazione della struttura e del programma

Abbiamo scelto di sottoporre per questo esperimento i sensori ad un moto armonico in quanto esso determina una variazione continua nel tempo sia dello spazio, che della velocità e dell'accelerazione. Funzioni simili sono il coseno e il seno. Abbiamo bisogno di una struttura che possa creare questo moto con costanza. Dato che il Gyro Sensor effettua misure riguardanti la velocità angolare, abbiamo la necessità che il moto sia rotatorio. Ovviamente, essendo i sensori collegati con dei cavi, non possiamo far compiere una rotazione completa ma la struttura realizzata dovrà oscillare fra due posizioni limite. Abbiamo per prima cosa creato un braccio, imperniato ad una estremità, ed alloggiato il sensore giroscopico in modo che il suo asse coincida col perno della struttura. Dalla parte opposta del braccio abbiamo montato il sensore di accelerazione. Il problema che ora ci si pone d'innanzi è: come far oscillare in maniera sinusoidale questa asta? L'unico attuatore che abbiamo a disposizione è il motore dell'NXT che ci fornisce però un moto rotatorio. Questo moto è garantito a velocità costante dal PID. Ma come possiamo trasformarlo in un moto sinusoidale?
Pensiamo ad un motore a 2 o 4 tempi in cui il moto rettilineo fornito dai pistoni vie-

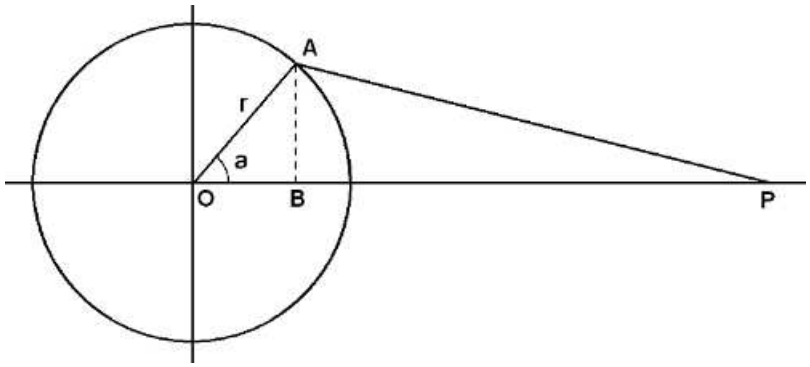


Figura 3.23: Calcolo spostamento del punto P

ne trasformato in un moto rotatorio grazie al meccanismo biella manovella. Possiamo sfruttare questo principio in maniera opposta. La nostra biella sarà imperniata ad una distanza `r'` dal fulcro dell'asta dei sensori, facendola così oscillare come un pendolo. Ma il moto risultante avrà un andamento sinusoidale? Applichiamo alcune nozioni di trigonometria per determinare l'andamento temporale del punto `P`. Osservando la figura 3.23 calcoliamo la lunghezza del segmento `OP`. Possiamo esprimere `OB = r cos(∂)` e `AB = r sin(∂)`. Essendo `AP` la lunghezza della nostra biella, `BP =` $\sqrt{\texttt{AP}^2\texttt{-AB}^2}$ e `OP = OB+BP`.
Nel grafico di figura 3.24 è visibile l'andamento del punto `P` al variare dell'angolo ∂. I valori sono normalizzati tra ± 1. Questa operazione è stata effettuata sottraendo la lunghezza della biella e dividendo per `r`. Definiamo la corsa come la distanza massima che il punto `P` percorre, pari quindi a `2r` ed osserviamo i dati calcolati al variare del rapporto `biella/corsa`. Se esso assume un valore basso il moto non rispecchia l'andamento

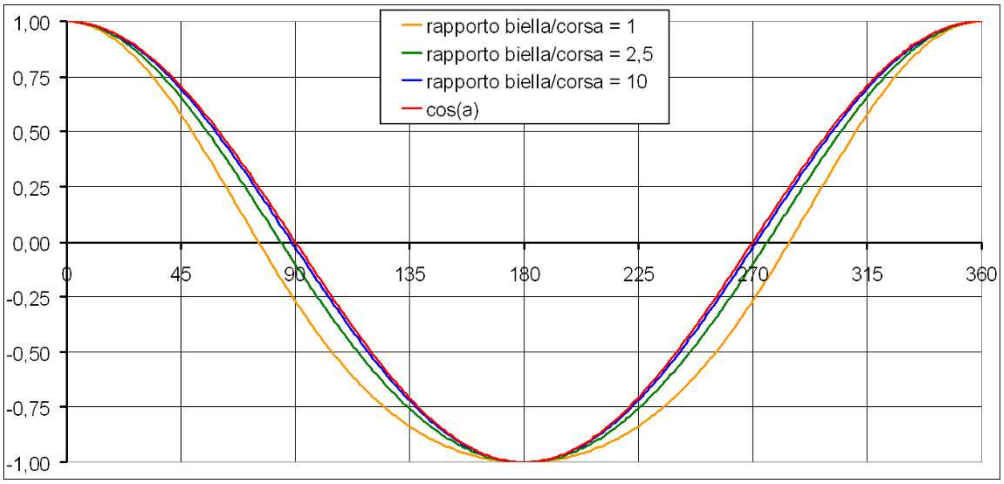


Figura 3.24: Valori assunti dal punto P al variare dell'angolo ∂

sinusoidale fornito dalla funzione `cos(`∂`)` ma, all'aumentare di questo rapporto, i dati si avvicinano sempre più all'andamento armonico. Bisogna tenere conto di questo aspetto creando la nostra struttura, cercando di massimizzare il valore del rapporto *biella/corsa*.
Determinata l'entità dello spostamento del punto `P` dobbiamo calcolare come si ripercuote questo movimento sull'inclinazione dell'asta dei sensori. `P` è imperniato, sul braccio a cui sono collegati gli strumenti di misura, ad una distanza `r'` dall'asse di rotazione e può quindi percorrere una porzione di circonferenza del medesimo raggio. Perciò indicando con `x` lo spostamento del punto `P`, l'angolo α compiuto dall'asta sarà pari `x(2r'`Π`)/360`. Esso è quindi proporzionale allo spostamento effettuato dal punto `P` e per questo assumerà anch'esso un andamento armonico. Abbiamo ora tutte le conoscenze necessarie per creare la nostra struttura. Essa presenta le seguenti misure: `r`=16mm, `r'`=24mm, biella=324mm e un rapporto `biella/corsa` pari a 10.
Realizzata la parte hardware dobbiamo occuparci della parte software. Come nelle altre sezioni riportiamo lo pseudocodice del programma realizzato, essendo dal punto di vista didattico più utile avere un elenco dei passi da effettuare piuttosto che il listato vero e proprio.

1. creare 6 vettori per salvare i valori di posizione, velocità, accelerazione e l'istante a cui essi vengono rilevati. Salviamo il tempo in cui ogni singolo dato viene rilevato perché vogliamo evitare errori che si possono verificare a causa dei ritardi nell'acquisizione;

2. dichiarare gli oggetti di tipo `AccelSensor` e `GyroSensor`;

3. calibrare l'accelerometro;

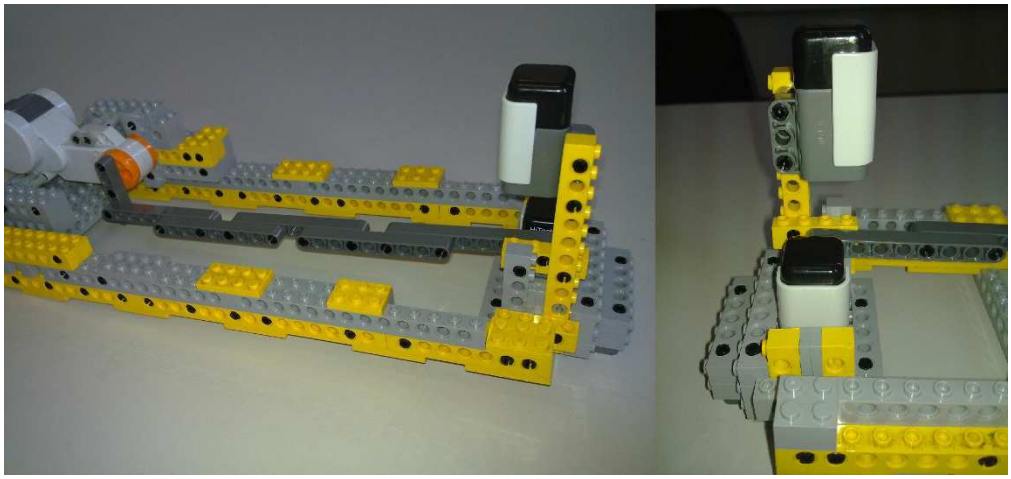
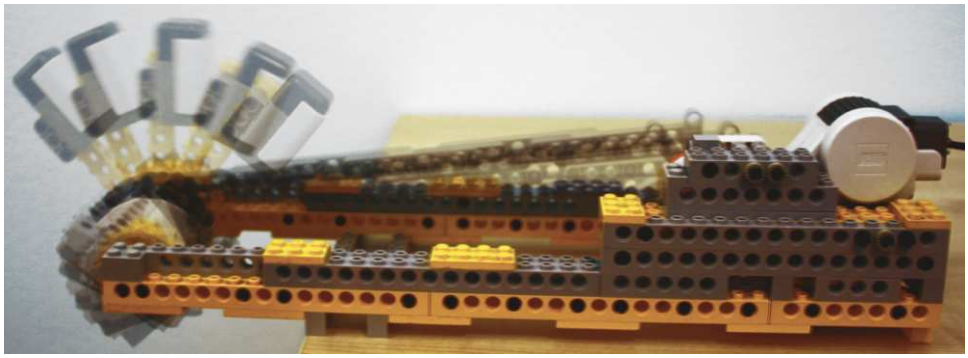


Figura 3.25: La struttura realizzata per l'esperimento e il suo movimento



4. fintanto che non si esauriscono le posizioni disponibili nei vettori acquisiamo i dati;

5. salvare i dati su file. Salviamo prima su un vettore e, successivamente, trasferiamo le informazioni raccolte in un file in quanto il salvataggio in tempo reale su quest'ultimo rallenta la frequenza di acquisizione dei dati.

Prima di visualizzare i dati dobbiamo però affrontare due punti critici riguardanti il sensore di accelerazione. In primis esso misura la somma delle accelerazioni a cui è sottoposto. Il nostro problema sta nel fatto che, oltre all'accelerazione indotta dal movimento dell'asta a cui il sensore è collegato, su di esso agirà una componente dovuta alla forza di gravità. Essa è variabile in base all'angolo a cui l'asta dei sensori si trova. Questo non costituisce per noi un problema perché, come spiegato nella sottosezione precedente, α = arcsin(x/mg) preso α pari all'angolo dell'asta e x il valore misurato dell'accelerometro. Per cui, conoscendo la posizione angolare dell'asta al momento dell'acquisizione del dato dall'accelerometro, possiamo risalire al valore dell'accelerazione impressa dalla forza di gravità pari a mg sin(α). Tolta questa componente il

dato ricavato dovrebbe rispecchiare l'accelerazione tangenziale del sistema.
Un secondo problema consiste proprio nel fatto che l'accelerometro, così posizionato, fornisce il valore dell'accelerazione tangenziale. Essa è espressa in m/sec^2 mentre i valori di posizione e velocità sono noti in gradi e gradi/sec. Per ora diciamo solo che il valore dell'accelerazione fornito dall'accelerometro deve essere diviso per un fattore pari a 1,57 per trasformare l'unità di misura da m/sec^2 a gradi/sec^2. Approfondiremo in seguito questo aspetto. Prima di ciò vogliamo richiamare i concetti di velocità e accelerazione.

3.8.2 La velocità e l'accelerazione

La velocità media di un intervallo è espressa come $\bar{v} = (x_2\text{-}x_1)/(t_2\text{-}t_1)$ dove x_1 e x_2 rappresentano la posizione rispettivamente al tempo t_1 e t_2. Perciò possiamo esprimere la velocità media come la variazione dello spazio nell'intervallo di tempo, cioè $\bar{v} = \Delta x/\Delta t$ con unità fondamentale il m/s. La velocità media è quindi la pendenza della retta che passa per i punti di coordinate $\{t_1;x_1\}$ e $\{t_2;x_2\}$. Essa però rappresenta il comportamento medio nell'intervallo di tempo. Perciò si introduce il concetto di velocità istantanea, non più relativa ad un intervallo di tempo ma come limite riferito ad un istante preciso. Con il termine velocità solitamente si intende il valore istantaneo che è indicato dalla funzione $v(t)$. Per determinalo riprendiamo la definizione di velocità media. Se facciamo tendere $\Delta t \to 0$ otteniamo $v = \lim_{\Delta t \to 0} \Delta x/\Delta t$. La retta che congiunge x_1 e x_2 si avvicina alla tangente nel punto $x(t)$. Essa, per definizione, è la derivata della funzione $x(t)$ che rappresenta lo spazio. Per cui $v(t) = dx(t)/dt$ e coincide con la derivata dello spazio, rappresentando la rapidità di variazione della posizione nel tempo.
In linea con la spiegazione precedente possiamo definire l'accelerazione media come $\bar{a} = (v_2\text{-}v_1)/(t_2\text{-}t_1)$ dove v_1 e v_2 rappresentano la velocità rispettivamente al tempo t_1 e t_2. Essa è definita quindi come la variazione di velocità nell'unità di tempo, per cui $\bar{a} = \Delta v/\Delta t$ espressa in m/s^2. Anche l'accelerazione media non fornisce molte indicazioni su come varia la velocità nell'intervallo di tempo Δt. Definiamo quindi l'accelerazione istantanea, o semplicemente accelerazione, $a = \lim_{\Delta t \to 0} \Delta v/\Delta t$ che corrisponde alla derivata della funzione $v(t)$. Esprimiamo per cui l'accelerazione come $a(t) = dv(t)/dt$ e rappresenta la rapidità con cui la velocità varia nel tempo.
Riportiamo in figura 3.26 l'andamento dello spazio, della velocità e dell'accelerazione in un moto armonico. La funzione $x(t)$ pari a $\cos(2\Pi t)$ rappresenta lo spazio all'istante t. Vediamo come la velocità assuma un andamento pari a $-2\Pi\sin(2\Pi t)$ e l'accelerazione a $-4\Pi^2\cos(2\Pi t)$. Queste due funzioni sono rispettivamente la derivata prima e la derivata seconda di $x(t)$. Le tre grandezze hanno lo stesso periodo ma sono tra loro sfasate. Quando lo spostamento è massimo, la velocità, dovendo cambiare di segno ovvero invertire il verso, è nulla. L'accelerazione in questo istante è massima in modulo e di segno opposto allo spostamento. Invece, quando lo spostamento è nullo,

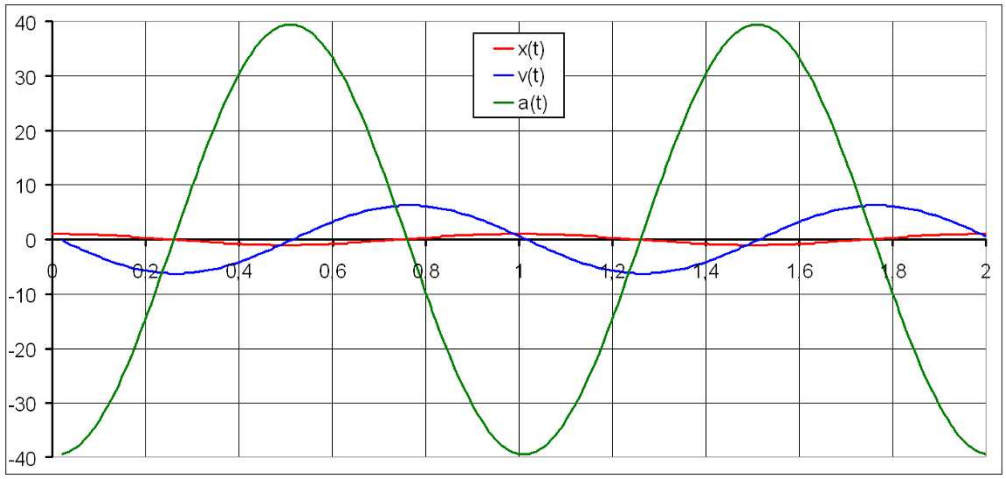


Figura 3.26: Andamento delle funzioni x(t), v(t) e a(t)

la velocità è massima e l'accelerazione è nulla. La velocità cresce quando ci si dirige verso la posizione 0 e decresce quando se ne allontana. Questo comportamento è pari a quello di un sistema massa-molla portato come esempio raffigurante il moto armonico in tutti i libri di fisica.
Un'ultima considerazione su queste due grandezze. Come abbiamo detto, per velocità e accelerazione consideriamo i valori istantanei e, essendo definiti tali, l'intervallo tra due valori tende a 0. Tuttavia nessuno strumento di misura può garantire questo. Le letture hanno sempre un intervallo minimo molto lontano dal valore 0. Ricordiamoci che il tempo minimo tra due letture di un sensore analogico è pari a circa 3ms, mentre l'analogo periodo relativo ai sensori digitali si attesta attorno ai 10ms. I dati rilevati esprimono perciò l'andamento medio tra due misurazioni. Perdiamo in questo caso informazioni importanti relative ai valori della funzione? Dobbiamo dire che, nonostante le misurazioni presentino un intervallo considerevole, possiamo definire la velocità e l'accelerazione rilevate quasi come dei valori istantanei. Affermiamo ciò in quanto la frequenza di acquisizione è di gran lunga superiore alla frequenza con cui variano le grandezze monitorate. Ovviamente non è sempre così. Se acquisiamo ogni secondo la velocità di un moto armonico che ha periodo pari a 10 millisecondi queste considerazioni non sono più valide.
Riprendiamo ora la discussione sulla corrispondenza tra accelerazione tangenziale e angolare. L'accelerometro è posizionato a 90mm dal perno del braccio. La circonferenza risultante ha una lunghezza pari a 565,49mm. Quindi, dividendo per 360, otterremo che lo spostamento di un grado corrisponde alla percorrenza di 1,57mm sulla circonferenza. Posto quindi y_1 pari allo spostamento del sensore di accelerazione e α_1 la variazione angolare corrispondente si instaurerà la relazione $\alpha_1 = y_1/1,57$. La velocità angolare è espressa come *spazio percorso/tempo di percorrenza*. Perciò preso quest'ultimo pari a t_1 e con la relazione sopra, possiamo affermare che $\alpha 1/t1 = (y1/1,57)/t_1$

cioè *velocità angolare = velocità tangenziale/1,57*. Per l'accelerazione valgono le stesse considerazioni. Possiamo perciò affermare che l'accelerazione angolare del sistema è pari all'accelerazione tangenziale misurata dal sensore divisa per un fattore di scala pari a 1,57.

3.8.3 I risultati

Pubblicheremo e analizzeremo ora i dati raccolti impostando al motore una velocità pari a 300 e 540 gradi/secondo. Vogliamo solo precisare che la posizione dell'asta è ricavata dal valore angolare letto grazie al tachimetro del motore ed utilizzato come variabile ∂ nella formula della sezione 3.8.1. Siccome gli snodi presentano dei giochi, l'andamento reale non si rivelerà così regolare ma, in ogni caso, molto vicino a quello da noi rappresentato. I dati acquisiti dall'accelerometro sono stati filtrati con un filtro a media mobile[5] con frequenza di taglio pari a circa 14Hz. Questa operazione è stata necessaria in quanto l'andamento presentava irregolarità dovute a vibrazioni e sbalzi nel moto di origine meccanica.
Nel grafico di figura 3.27 abbiamo inserito i dati ricavati, normalizzandoli tra 1 e -1 al

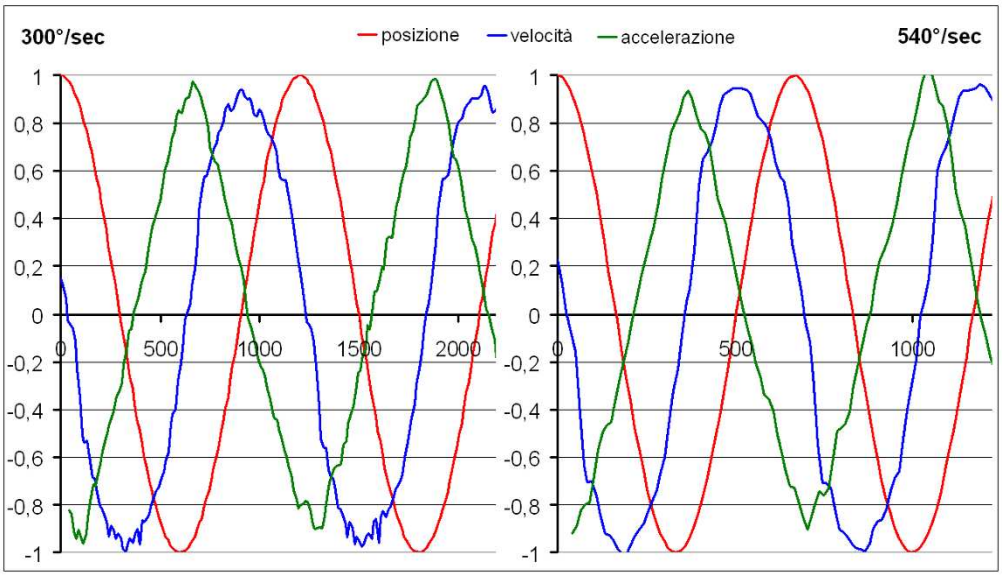


Figura 3.27: Posizione, velocità e accelerazione al variare del tempo

fine di focalizzare l'attenzione sullo sfasamento delle grandezze rappresentate. Si nota che i dati rispecchiano i valori attesi. Come descritto nella sezione precedente, anche

[5] Per approfondire l'argomento si invita il lettore a leggere la sezione pubblicata in Appendice

nei dati sperimentali raccolti la velocità ha modulo massimo quando accelerazione e spazio sono nulli. Viceversa essa si annulla quando le altre due grandezze hanno modulo massimo. L'accelerazione e lo spazio hanno invece un andamento tra loro opposto. Si nota comunque un lieve ritardo nell'andamento dell'accelerazione dovuto alla scarsa reattività dei movimenti della massa dell'accelerometro.
Anche la velocità sembra presentare un ritardo che però risulta di difficile valutazio-

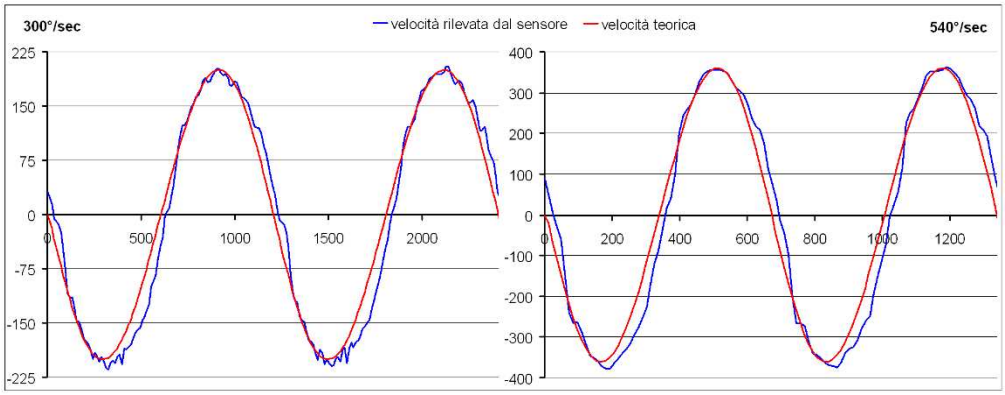


Figura 3.28: Confronto fra i valori di velocità sperimentali e teorici

ne. In figura 3.28 confrontiamo perciò i dati ottenuti dal sensore con quelli calcolati teoricamente. Notiamo un leggero ritardo nella risposta che si verifica non su tutto l'andamento dei dati, ma solamente quando il sistema, avendo raggiunto il picco di velocità, inverte la tendenza passando da accelerazione a decelerazione. In questo caso si nota come la curva dei dati sperimentali si scosti da quella dei dati teorici. Nonostante ciò, siamo soddisfatti delle performance dimostrate da questo sensore che fornisce un valore esatto in riguardo al modulo massimo della velocità angolare.
In figura 3.29 abbiamo aumentando la frequenza dell'oscillazione del sistema impostando la velocità del motore pari a 800° al secondo. La scheda tecnica del giroscopio garantiva una velocità massima misurabile pari a 360 gradi al secondo. Dalle misure da noi effettuate riscontriamo che il valore è di poco maggiore e pari a circa 387°/sec. Notare che il fondo scala del sensore non è simmetrico. Per valori negativi esso sembra avere ancora margine per il rilevamento. Non siamo però riusciti a misurare questo valore in quanto la struttura non ci permetteva di sostenere una frequenza di oscillazione più elevata.
Effettuando la stessa analisi per quanto riguarda i dati relativi all'accelerazione notiamo che, oltre ad essere sottoposta ad un notevole ritardo, la curva dei dati rilevati presenta una notevole sovraelongazione rispetto a quella rappresentante i valori teorici. Tutto ciò è rappresentato in figura 3.30. Quindi, anche se abbiamo eliminato la componente dovuta alla forza di gravità, nel sensore agiscono forze che inducono un'accelerazione che va a sommarsi a quella derivata dal moto. Possiamo ipotizzare che questo errore sia dovuto all'inerzia che presenta la massa incisa nel chip, che tende a conservare la

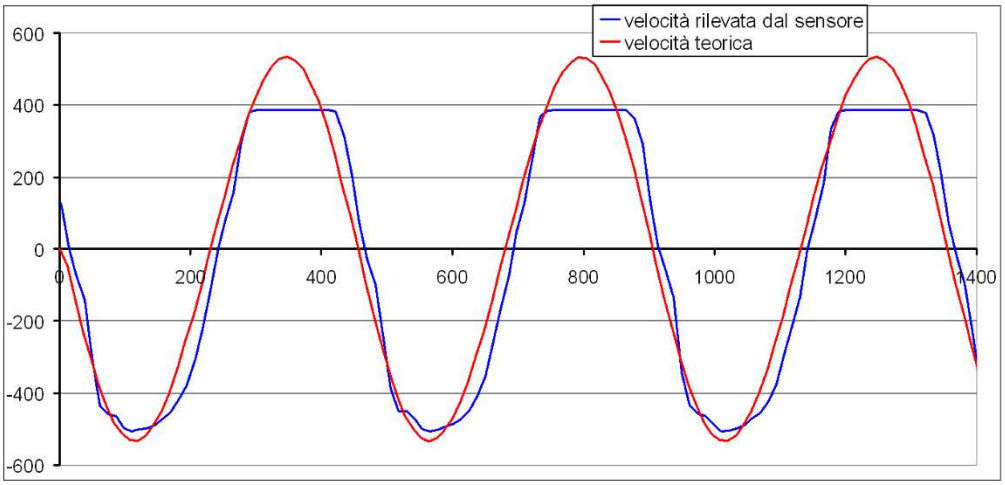


Figura 3.29: Valori limite del giroscopio

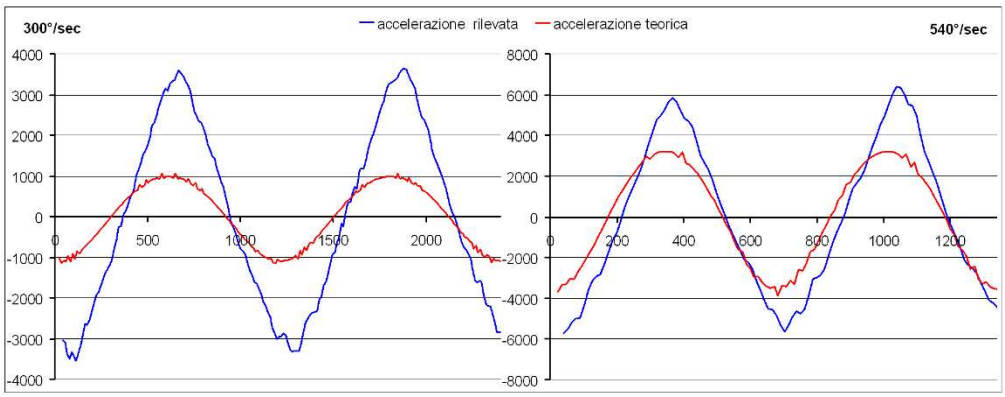


Figura 3.30: Confronto fra i valori di accelerazione sperimentali e teorici

quantità di moto a cui è sottoposta.
Osservando i due dati a confronto si nota che il rapporto *accelerazione massima/errore massimo* tende ad aumentare al crescere della frequenza di oscillazione. Proviamo, per avvalorare la nostra tesi, ad impostare la massima velocità sostenibile dal sistema. I dati rilevati sono pubblicati nel grafico di figura 3.31 che conferma le nostre ipotesi. Si vede come per oscillazioni di elevata intensità l'accelerometro risponda meglio, seguendo più fedelmente la curva che rappresenta i dati teorici. Possiamo perciò concludere che il sensore è sottoposto ad un errore costante. Man mano che il valore massimo dell'accelerazione aumenta, il rapporto *segnale/errore* cresce facendo tendere, come abbiamo verificato, i valori misurati ai dati reali. Si potevano quindi raccogliere risultati migliori anche nel caso in cui la velocità impostata fosse stata minore. Ciò è possibile aumen-

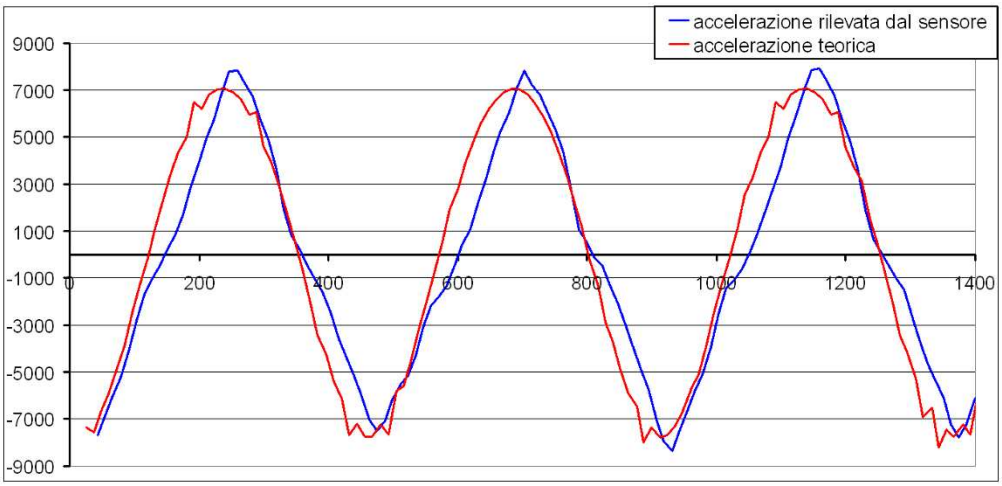


Figura 3.31: Valori di accelerazione raccolti alla massima velocità del sistema

tando la distanza del sensore dal perno aumentando così il modulo dell'accelerazione tangenziale cui esso è sottoposto e "coprendo" così le anomalie di misurazione.

3.9 Rilevamento della navigazione

L'obbiettivo principale di questa esperienza consiste nel determinare la navigazione effettuata da un robot. Per navigazione intendiamo il percorso o traiettoria che un oggetto compie nello spazio. Abbiamo a disposizione due sistemi per effettuare questo rilevamento:

- mediante il tachimetro dei motori;
- grazie al sensore giroscopico e di accelerazione, sistema che viene definito di *"navigazione inerziale"* e utilizzato, per esempio, per tracciare le rotte degli aerei.

Per determinare ciò, dobbiamo calcolare il modulo e l'orientamento del vettore spostamento per ogni istante di tempo e porre queste informazioni nel piano xy. Possiamo identificare un punto che sta in un piano tramite il modulo di un vettore e l'orientamento che esso ha con il sistema di riferimento. Col vettore spostamento indichiamo quindi la distanza compiuta e la direzione che essa ha rispetto ad un asse definito. Se calcoliamo periodicamente i valori di questo vettore e identifichiamo ogni volta come

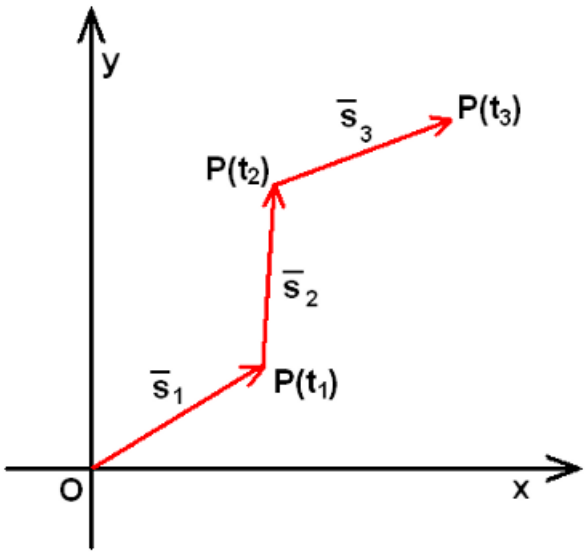


Figura 3.32: Vettori spostamento che identificano la traiettoria di P all'istante t_1,t_2,t_3

origine del vettore successivo la destinazione di quello precedente, otterremo la traiettoria dello spostamento. In figura 3.32 si può osservare lo scenario precedentemente descritto che permette il raggiungimento dell'obbiettivo dell'esperimento.

3.9.1 Struttura realizzata per l'esperimento

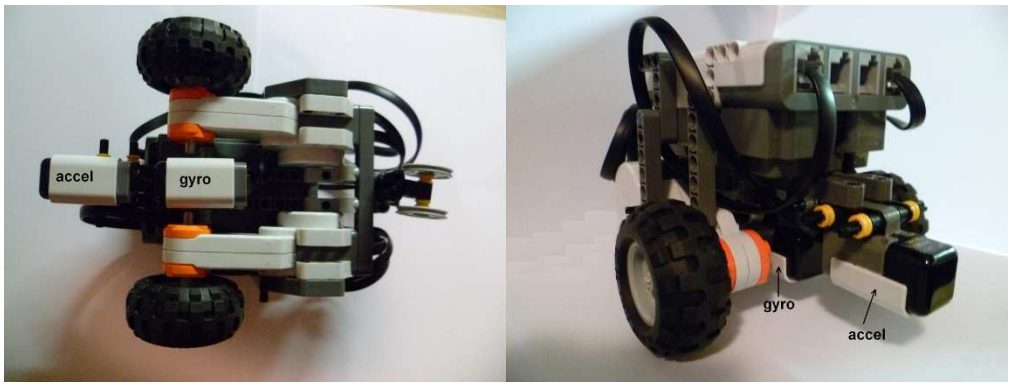


Figura 3.33: Il tribot

In figura 3.33 è ripreso il robot che abbiamo utilizzato. Esso è assemblato in configurazione tribot con due motori posizionati sul medesimo asse e che possono ruotare a diversa velocità e direzione. Il giroscopio ha l'asse perpendicolare al piano per permettere il rilevamento della velocità con cui il robot ruota su se stesso. Per rendere più precisa la misura si è cercato di far coincidere l'asse del giroscopio con l'asse di rotazione del robot. L'accelerometro è invece posizionato anteriormente con l'asse x concorde al movimento del tribot.

3.9.2 Rilevamento tachimetrico

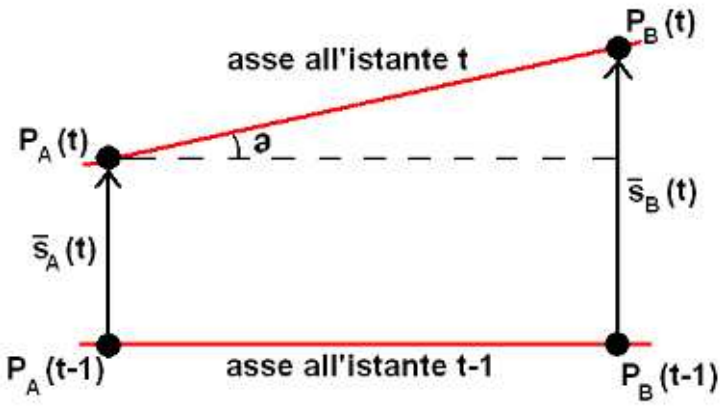


Figura 3.34: Vettori

Basandoci sul tachimetro possiamo determinare in un istante generico t l'entità del vettore spostamento effettuato da t a t-1. Per semplicità assumiamo che l'area di contatto delle ruote con il piano si riduca a due punti, P_A e P_B rispettivamente per il motore A e B come rappresentato in figura 3.34. In lasso di tempo compreso tra due acquisizioni del valore $tacho_A$ e $tacho_B$ essi compiranno una rotazione di n_A=$tacho_A$(t)-$tacho_A$(t-1) e n_B=$tacho_B$(t)-$tacho_B$(t-1). La rotazione della ruota corrisponderà a due vettori spostamento $\bar{s}_A$ e $\bar{s}_B$. Molti utenti potrebbero giustamente affermare che in sterzata le ruote del robot effettuerebbero un percorso curvilineo, ma ricordiamo che i due vettori identificano lo spostamento medio tra l'istante t-1 e t. Non ci è quindi dato a sapere il percorso intermedio anche se utilizzare un lasso di tempo molto ridotto tra due acquisizioni renderà la traiettoria delle ruote sovrapponibile ai vettori $\bar{s}_A$ e $\bar{s}_B$. Preso il diametro esterno del pneumatico pari 56mm, la rotazione di un grado si tradurrà in uno spostamento di 56*Π/360 = 0,4887mm/gradi. Quindi s_A= n_A*0,4887 e s_B= n_B*0,4887. Essendo il vettore spostamento posto al centro del robot, possiamo impostare il suo modulo come

$s(t)=(s_A(t)+s_B(t))/2$. L'angolo $\partial(t)$, ovvero la rotazione dell'asse del robot e del vettore, sarà pari ad $\arcsin((s_B(t)-s_A(t)/D)$, dove D è la distanza tra le ruote. Attenzione che la direzione è intesa rispetto al sistema di riferimento precedente, cioè al vettore precedente. La direzione del vettore rispetto al sistema di riferimento ad un istante t sarà pari ad $\alpha(t)=\sum_{k=0}^{k=n}\partial(k)$. Precisiamo che, trovandoci in campo discreto, con $n=t/\Delta t$ rappresentiamo il numero del campione acquisito al tempo t, dove quest'ultimo indica il lasso di tempo trascorso dall'inizio dell'acquisizione. Quindi, grazie a questi dati, possiamo tracciare la posizione del robot nel piano xy ad un istante t le cui le coordinate saranno:

- $x(t) = \sum_{k=0}^{k=n} s(k)\cos(\alpha(k))$
- $y(t) = \sum_{k=0}^{k=n} s(k)\sin(\alpha(k))$

Il programma perciò deve eseguire ciclicamente le seguenti istruzioni:

1. acquisire i valori attuali di `tachoA` e `tachoB` e sottrarvi i valori misurati nella lettura precedente ottenendo `nA` e `nB`;
2. calcolare `sA = nA*0,4887` e `sB = nB*0,4887`;
3. `s = (sA+sB)/2`;
4. `delta = arcsin((sB-sA)/D)`;
5. `alfa = alfa+delta`.
6. `x = x+s*cos(alfa)` e `y = y+s*sin(alfa)`;

3.9.3 Rilevamento inerziale

Un secondo metodo è realizzabile tramite il sensore giroscopico e di accelerazione. Come sappiamo dalla definizione di derivata e di integrale, $dF(x)/dx = f(x)$ e $\int f(x)dx= F(x)$. Abbiamo visto nella sezione 3.8 che la velocità è la derivata dello spazio e che l'accelerazione è la derivata della velocità. Per cui, grazie alle precedenti definizioni, preso come $s(t)$ come lo spazio, $v(t)$ la velocità e $a(t)$ l'accelerazione, possiamo dire che $v(t)=\int a(t)dt$ e $s(t)=\int v(t)dt$. Per cui possiamo ricavare il modulo del vettore spostamento integrando per due volte l'accelerazione misurata lungo la traiettoria del robot, cioè l'accelerazione tangenziale. Necessitiamo però, per ricavare la posizione del nostro robot nel piano, anche della direzione dello spostamento. Possiamo ricavare questo dato tramite la velocità angolare misurata dal giroscopio. Montato infatti questo strumento coincidente all'asse di rotazione del robot e integrando i dati ottenuti, possiamo ricavare l'entità della rotazione sostenuta tra questo istante e il precedente. Similarmente al caso della sottosezione precedente, sommando tutti i dati

ottenuti fino all'istante t, otterremmo la direzione del vettore spostamento al suddetto tempo e relativa al riferimento cartesiano del piano xy.
Come calcolare però un integrale? Un integrale definito riflette l'area di una funzione

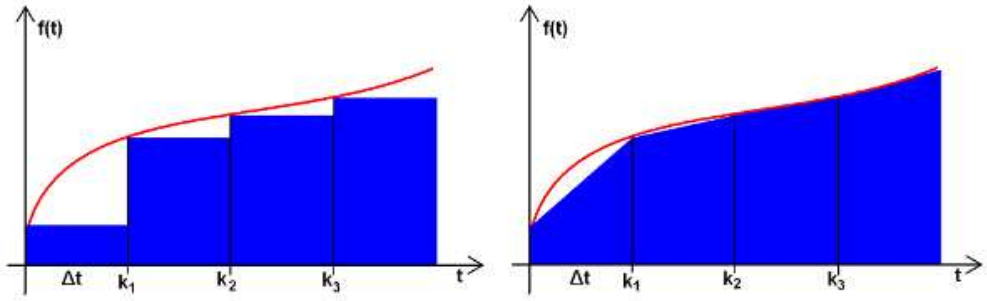


Figura 3.35: Calcolo dell'area sottesa da una funzione

racchiusa tra due estremi. La nostra difficoltà sta nel fatto che otteniamo i dati ogni Δt pari al tempo di campionamento. La funzione non è continua ma discreta per cui tra due rilevamenti essa assumerà infiniti valori. Come ottenere quindi l'area della funzione? Estrapolando dalla definizione di integrale di Riemann essa può essere vista come la somma dell'area di un pluri-rettangolo, pari quindi a $\sum_{k=0}^{k=n}$f(k)*Δt dove k indica il k-esimo campione e f(k) il valore della funzione al tempo in cui viene rilevato il campione. La figura 3.35 di sinistra descrive questa soluzione applicata ad una funzione continua come può essere l'accelerazione o la velocità angolare. Come osserviamo, l'area ottenuta è un'approssimazione di quella sottesa dalla funzione. Possiamo portare questo valore ancor più prossimo all'area reale, mantenendo lo stesso tempo di campionamento Δt? Si, sommando, invece che i rettangoli, i trapezi che compongono l'area della funzione. Ogni trapezio ha una base pari al k-esimo dato e l'altra pari al (k-1)-esimo dato, mentre l'altezza è pari all'intervallo di tempo che intercorre tra l'uno e l'altro dato. L'area della funzione ad un istante t è quindi pari a $\sum_{k=1}^{k=n}$[f(k)+f(k-1)]*Δt/2. Un'approssimazione migliore si otterrebbe diminuendo il tempo di campionamento, ma non si può andare al di sotto del lasso di tempo necessario per l'acquisizione dei dati dai sensori.
Indichiamo quindi con accel(k) e gyro(k) le funzioni rappresentanti i dati ottenuti rispettivamente dall'accelerometro e dal giroscopio. In seguito alle considerazioni sopra citate possiamo definire la direzione del vettore spostamento come α(t) =$\sum_{k=1}^{k=n}$-[gyro(k)+gyro(k-1)]*Δt/2. Il modulo del suddetto vettore s(t)=[accel(n)+-accel(n-1)]*Δt/2. In linea coi calcoli precedenti identifichiamo la posizione all'istante t nel piano xy come:

- x(t) = $\sum_{k=0}^{k=n}$s(k)cos(α(k))
- y(t) = $\sum_{k=0}^{k=n}$s(k)sin(α(k))

Abbiamo indentificato con `accelPrec`, `valPrec` e `gyroPrec` i valori rilevati nelle precedenti misurazioni. Il programma deve eseguire ciclicamente le seguenti istruzioni:

1. acquisire i valori attuali dai sensori e inserirli in `accel` e `gyro`;
2. `vel = (accel+accelPrec)*Tc/2` e `s = (vel+velPrec)*Tc/2`;
3. `alfa = alfa + (gyro+gyroPrec)*Tc/2`;
4. `x = x+s*cos(alfa)` e `y = y+s*sin(alfa)`;
5. `gyroPrec = gyro`, `accelPrec=Accel` e `velPrec=vel`.

3.9.4 Risultati

Per effettuare la prova è stato fatto compiere al tribot un percorso circolare che permettesse di misurare l'errore per via grafica in quanto, in questa maniera, la posizione iniziale e quella finale dovrebbero coincidere. La velocità di partenza è stata impostata pari a 100 gradi al secondo. Successivamente, il suo andamento è crescente con incrementi di 10 gradi al secondo ogni 100ms fino al raggiungimento di un valore pari a 500. Essendo il moto circolare, la velocità del motore esterno è pari a quella del motore interno moltiplicata per una costante determinata sperimentalmente, che permetta la realizzazione di un percorso chiuso.
I risultati relativi ai sensori, per essere utilizzabili, sono stati filtrati con un filtro passa basso con frequenza di taglio pari a 5Hz[6]. Questa operazione è stata eseguita per eliminare le componenti dovute alle vibrazioni del tribot.
In figura 3.36 abbiamo confrontato i moduli dei vettori spostamento ottenuti nelle due modalità. Possiamo notare come la rilevazione tachimetrica dia un risultato migliore. Infatti, come si è già evidenziato nell'esperimento di sezione 3.8, l'accelerometro ha una scarsa precisione nella misurazione di accelerazioni di bassa intensità.
La figura 3.37 rappresenta invece il confronto tra la direzione ottenuta dal rilevamento integrale e tachimetrico. Come si nota, la direzione integrata dai valori del giroscopio raggiunge un valore massimo di molto inferiori ai 360 gradi. Questo errore è dovuto probabilmente alla scarsa precisione del giroscopio nella misurazione di velocità angolari di bassa intensità. Abbiamo detto nella sezione precedente che eravamo soddisfatti della precisione di questo strumento invece, in questo caso, esso ha dimostrato i suoi limiti. Entrambi i sensori, quindi, sono scarsamente precisi nel caso in cui le grandezze misurate abbiano un modulo ridotto.
I risultati posti nel piano `xy` sono quelli rappresentati in figura 3.38. Come ci aspettavamo, il sistema inerziale si è dimostrato impreciso. Buoni risultati sono invece stati raccolti dal sistema tachimetrico, il quale ha compiuto una rotazione quasi completa

[6] Per chiarimenti sull'utilizzo si veda la sezione in appendice

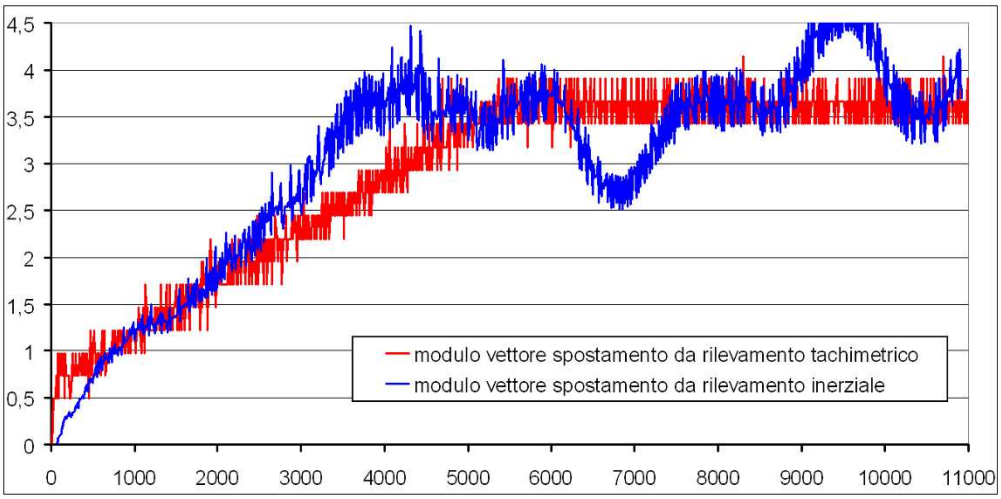


Figura 3.36: Modulo del vettore spostamento all'istante t

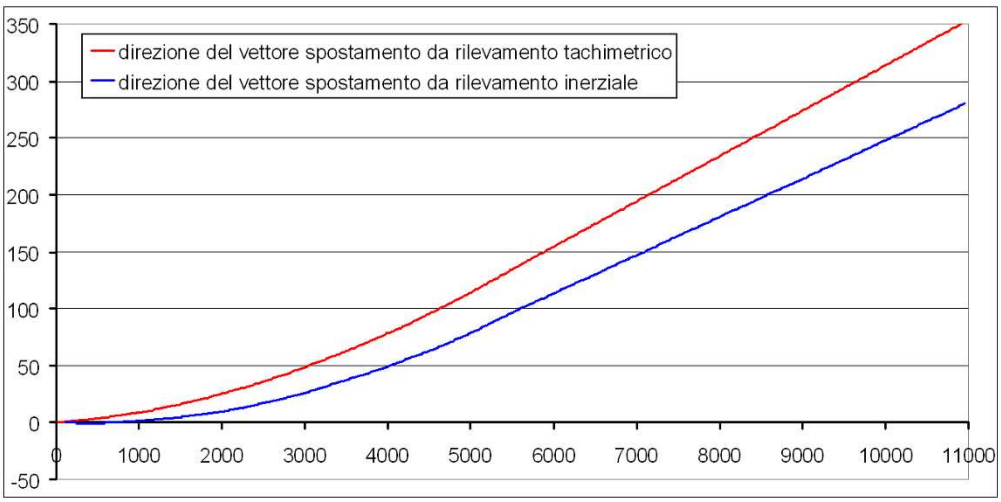


Figura 3.37: Direzione del vettore spostamento all'istante t

bloccando il suo percorso ad una distanza dall'origine di 50mm. Questo errore è ancor più ridotto dal fatto che il robot, nello svolgimento dell'esperimento, non si fermava esattamente nell'origine, ma leggermente prima ed esternamente, in maniera similare a quanto raccolto. Valutiamo quindi lo scostamento tra la posizione finale effettiva e quella rilevata dal robot, pari a meno del 2% dello spostamento totale. Ovviamente i dati dipendono molto da come si presenta il percorso e da che attrito presentano le ruote con il pavimento. Possiamo comunque identificare questo metodo di rilevamento come un valido sistema per tracciare la posizione del robot nel piano.

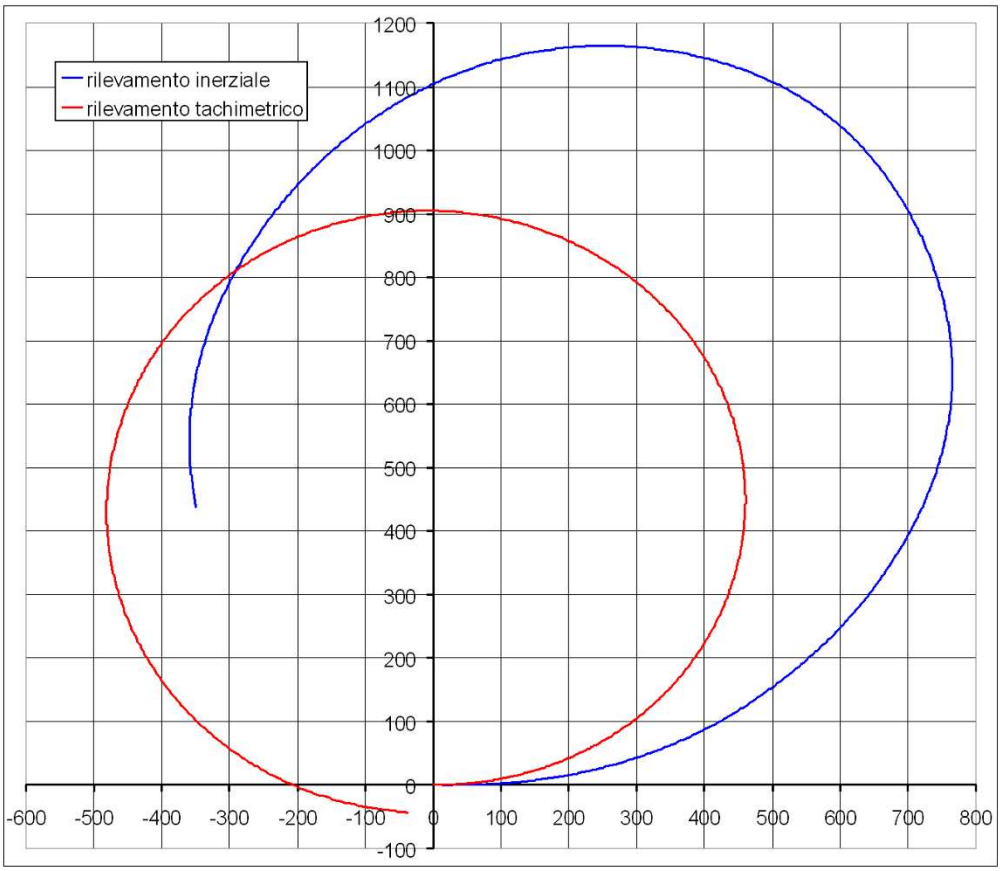


Figura 3.38: Traiettoria del robot. L'unità di misura utilizzata è il millimetro

I risultati raccolti col sistema inerziale hanno invece fornito un'errore tale da non permettere l'utilizzo di questa tecnica. Non vogliamo dire che essa sia sbagliata teoricamente, bensì che gli strumenti di misura a disposizione non ci permettono di ottenere risultati con un buon grado di precisione. Vogliamo ricordare che questo metodo di rilevamento è utilizzato comunemente più del sistema tachimetrico. I nostri dati sono così imprecisi in quanto gli strumenti non hanno un'alta sensibilità atta a rilevare i minimi cambiamenti di accelerazione e velocità angolare a cui e sottoposto il robot. In più, nella realtà, i valori ricavati dall'accelerometro sono filtrati con particolari filtri, detti di Kalman, che cercano di prevedere l'andamento futuro grazie ai valori precedenti.
Questo esperimento torna utile nelle attività svolte dagli studenti come nella simulazione di un robot di soccorso che, trovata in una stanza una persona in difficoltà, la porti fuori da dove è entrato. In questo caso è necessario sapere come raggiungere nel minor tempo possibile l'uscita una volta che si ha soccorso il ferito. Questo metodo permette al robot di dirigersi con buona approssimazione nelle vicinanze della breccia da cui è entrato. Una volta arrivato nei paraggi si potrebbe prevedere di cercare col sensore ad

ultrasuoni l’uscita.

Capitolo 4

Appendice

4.1 La classe AccelSensor

```
import lejos.nxt.I2CPort;
import lejos.nxt.I2CSensor;
import java.lang.Math;
import lejos.nxt.SensorPort;

//-------------------------------------------------------------------------
//  Questa classe è stata scritta da Massimo Perotto per il sensore
//  di accelerazione HiTechnic. Una documentazione dettagliata
//  sull'uso è disponibile nella sezione 4.5 e 4.7 della tesi: "Una
//  guida per l'utilizzazione educativa del robot Mindstorms NXT con
//  programmazione Java: la componente hardware"
//-------------------------------------------------------------------------

public class AccelSensor extends I2CSensor
{
  //8bit piu significativi
  byte [] bufUpper = new byte[1];

  //2bit meno significativi
  byte [] bufLower = new byte[1];

  //Indica l'errore o il buon termine della procedura getdata
  int ret;

  //valore rilevato
  int val;

  //valore in doppia precisione
  double valD;

  //valori di taratura
  int minX,maxX,minY,maxY,minZ,maxZ;
```

```
  //indirizzo dei registri di memoria del sensore contenenti i dati
  private static byte X_UPPER_BITS = 0x42;
  private static byte Y_UPPER_BITS = 0x43;
  private static byte Z_UPPER_BITS = 0x44;
  private static byte X_LOWER_BITS = 0x45;
  private static byte Y_LOWER_BITS = 0x46;
  private static byte Z_LOWER_BITS = 0x47;

//---------------------------------------------------------------------

  public AccelSensor(I2CPort port)
  {
    super(port);
    port.setType(TYPE_LOWSPEED_9V);
    if(this.getProductID().equals("mndsnsrs"))
      throw new IllegalArgumentException();
    minX=maxX=minY=maxY=minZ=maxZ=200;
  }

//---------------------------------------------------------------------
//  Questi metodi sono necessari per la calibrazione dello strumento.
//  Vedere la sezione 4.7 per l'utilizzo
//---------------------------------------------------------------------

  public void calibrateX(int max, int min)
  {
    minX=min;
    maxX=max;
  }

  public void calibrateY(int max, int min)
  {
    minY=min;
    maxY=max;
  }

  public void calibrateZ(int max, int min)
  {
    minZ=min;
    maxZ=max;
  }

//---------------------------------------------------------------------
//  Questi metodi forniscono il valore dell'inclinazione in gradi del
//  sensore. Il valore viene ricavato dalla componente dell'accelera-
//  zione impressa dalla forza gravitazionale. ATTENZIONE: il valore
//  è valido solo se il sensore è fermo nella posizione di cui si
//  vuole misurare l'inclinazione o si muove di moto quasi staziona-
//  rio. Se il sensore è sottoposto a movimento, al valore dell'acce-
//  lerazione dovuto alla forza gravitazionale si sommano componenti
//  derivate dal moto che in maniera più o meno grave possono intro-
//  durre errori.
//---------------------------------------------------------------------
```

```
  public double getXTilt()
  {
    return tilt(X_UPPER_BITS,X_LOWER_BITS,maxX,minX);
  }

  public double getYTilt()
  {
    return tilt(Y_UPPER_BITS,Y_LOWER_BITS,maxY,minY);
  }

  public double getZTilt()
  {
    return tilt(Z_UPPER_BITS,Z_LOWER_BITS,maxZ,minZ);
  }

//-----------------------------------------------------------------
//  Questi metodi forniscono il valore di accelerazione in mm/sec^2
//-----------------------------------------------------------------

  public int getXAccel()
  {
    return accel(X_UPPER_BITS,X_LOWER_BITS,maxX,minX);
  }

  public int getYAccel()
  {
    return accel(Y_UPPER_BITS,Y_LOWER_BITS,maxY,minY);
  }

  public int getZAccel()
  {
    return accel(Z_UPPER_BITS,Z_LOWER_BITS,maxZ,minZ);
  }

//-----------------------------------------------------------------
//  Questi metodi forniscono il valore così come viene letto dal
//  sensore.
//-----------------------------------------------------------------

  public double getXData()
  {
    return getData(X_UPPER_BITS,X_LOWER_BITS);
  }

  public double getYData()
  {
    return getData(Y_UPPER_BITS,Y_LOWER_BITS);
  }

  public double getZData()
  {
    return getData(Z_UPPER_BITS,Z_LOWER_BITS);
```

```
  }

//-------------------------------------------------------------------------
//  Questi metodi forniscono il valore di accelerazione misurato in G
//  volte la forza di gravità
//-------------------------------------------------------------------------

  public double getXNormalizeData()
  {
    return getNormalizeData(X_UPPER_BITS,X_LOWER_BITS,maxX,minX);
  }

  public double getYNormalizeData()
  {
    return getNormalizeData(Y_UPPER_BITS,Y_LOWER_BITS,maxY,minY);
  }

  public double getZNormalizeData()
  {
    return getNormalizeData(Z_UPPER_BITS,Z_LOWER_BITS,maxZ,minZ);
  }

//-------------------------------------------------------------------------

  private double tilt(byte UPPER_BITS, byte LOWER_BITS, int max,
                                                              int min)
  {
    valD=getNormalizeData(UPPER_BITS, LOWER_BITS, max, min);
    if(valD>1.0) valD=1.0;
    if(valD<-1.0) valD=-1.0;
    valD= Math.asin(valD);
    return Math.toDegrees(valD);
  }

  private int accel(byte UPPER_BITS, byte LOWER_BITS, int max,
                                                              int min)
  {
    valD=getNormalizeData(UPPER_BITS, LOWER_BITS, max, min);
    return (int)Math.round(valD*9810);
  }

  private double getNormalizeData(byte UPPER_BITS, byte LOWER_BITS,
                                                    int max, int min)
  {
    val=getData(UPPER_BITS, LOWER_BITS);
    if(min<0) min=-min;
    if(max<0) max=-max;
    valD=val-((max-min)/2.0);
    valD=valD/((max+min)/2.0);
    return valD;
  }

  private int getData(byte UPPER_BITS, byte LOWER_BITS)
```

```
  {
    ret = getData(UPPER_BITS, bufUpper, 1);
    if (ret != 0) throw new IllegalArgumentException();
    ret = getData(LOWER_BITS, bufLower, 1);
    if (ret != 0) throw new IllegalArgumentException();
    int val = bufUpper[0];
    val = val * 4;
    val += bufLower[0];
    return val;
  }

//-------------------------------------------------------------------------
}
```

4.2 I filtri digitali

Nei nostri esperimenti siamo ricorsi molte volte all'uso di filtri digitali. Non abbiamo mai voluto approfondire l'argomento in quanto essi richiedono una profonda conoscenza della convoluzione e della trasformata *zeta* per i segnali a tempo discreto. In questa sessione vogliamo cercare di metter a disposizione degli utenti della guida un semplice sistema per il filtraggio dei dati. Non spiegheremo la teoria dei segnali che sta dietro a questi filtri, ma cercheremo di desumere il loro comportamento attraverso i dati sperimentali. Questo strumento può tornare utile nel caso in cui i valori rilevati, per esempio dall'accelerometro, siano disturbati da vibrazioni a cui lo strumento è soggetto.
Ipotizziamo che il nostro segnale sia rilevato dall'accelerometro che, come nella sezio-

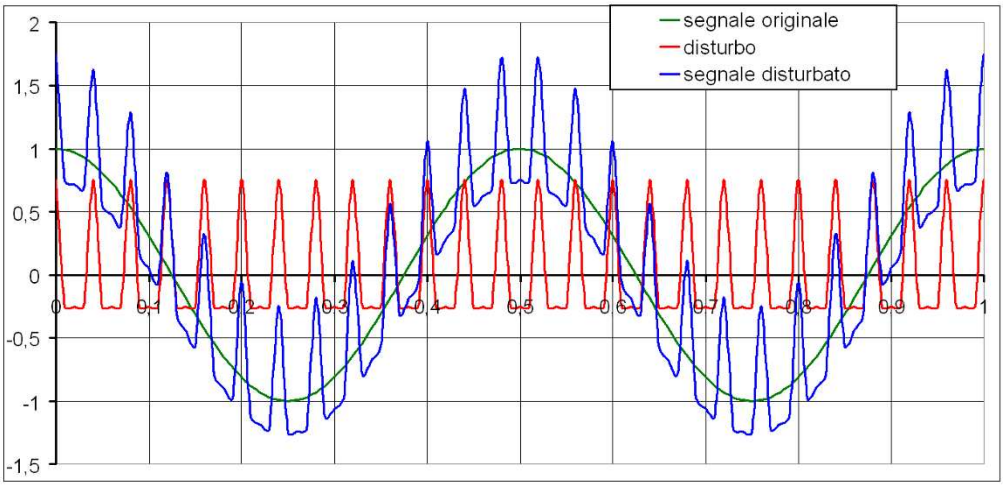


Figura 4.1: Segnale originale, disturbo e segnale disturbato

ne 3.8, acquisisce i valori di accelerazione forniti da un moto armonico di periodo pari a 0,5s. La frequenza del segnale ricavato sarà quindi $f_s = 1/0,5s = 2Hz$. I dati così rilevati sono espressi dalla funzione $\cos(2f_s\Pi t)$. Inseriamo ora un disturbo di forte intensità che potremmo esprimere come $(1/2)\cos(2f_d\Pi t)+(1/4)\cos(4f_d\Pi t)$ con f_d = 25. I segnali descritti sono rappresentati in figura 4.1. Per il filtraggio utilizziamo un filtro a media mobile. Come abbiamo già spiegato nei precedenti capitoli, noi otteniamo un insieme di campioni che rappresentano l'andamento della funzione che monitoriamo. Il tempo che intercorre tra due campioni acquisiti con l'accelerometro è pari a 10ms per cui abbiamo una frequenza di campionamento $f_c=1/0,01=100Hz$. Applicando il suddetto filtro il dato i-esimo assume valore pari alla media dei dati precedentemente ottenuti ed esattamente dal dato i al dato $i-D$. Con D rappresentiamo la dimensione della finestra espressa come il numero di campioni coinvolti nel calcolo. Quindi il primo dato sarà ottenibile dal filtro solo quando avremo effettuato un numero di letture pari a D. La figura 4.2 mostra i risultati ottenuti con una finestra di dimensione pari a 7.
Si riesce così ad eliminare il disturbo riottenendo all'incirca il segnale originale. Ab-

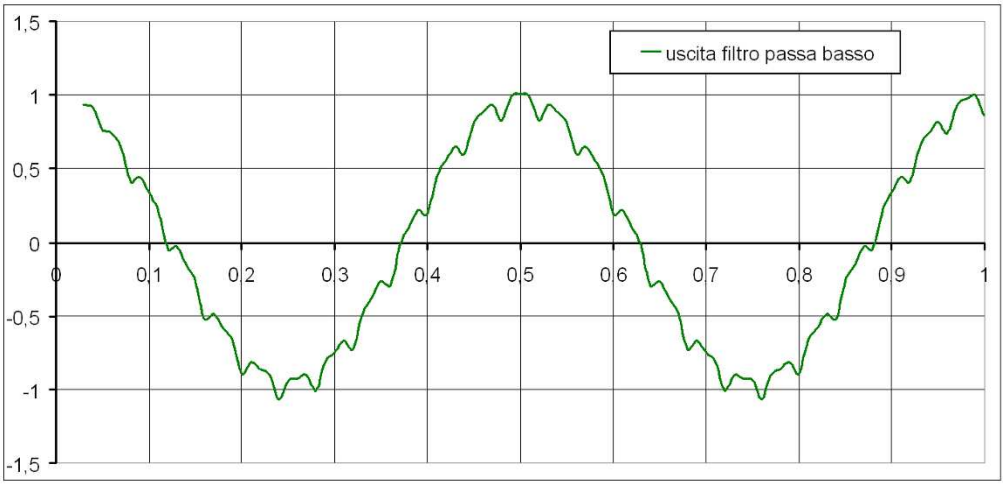


Figura 4.2: Segnale disturbato filtrato

biamo così un filtro passa basso. Ma come trovare la dimensione della finestra?
Utilizziamo la funzione $\cos(2f\Pi t)$ con un valore di f variabile tra 1 e 20Hz. Selezioniamo valori per D pari a 5, 7, 11, 15 e proviamo a calcolare l'attenuazione che i filtri così creati inducono sul segnale precedentemente citato. Il grafico in figura 4.3 mostra i dati raccolti. L'ordinata rappresenta il fattore di moltiplicazione a cui è soggetto il modulo del segnale mentre in ascissa indichiamo la frequenza. I dati così ottenuti rappresenteranno quindi la funzione di trasferimento del filtro nel dominio della frequenza. Notiamo subito come il guadagno dei filtri si azzeri per un valore multiplo di $1/(D*T_c)$ dove T_c è il tempo di campionamento che intercorre tra un rilevamento e il successivo. Quindi la funzione di trasferimento si azzera per valori multipli dell'inverso del tempo

sotteso dalla finestra.
Come utilizzare questi strumenti? Non c'è una dimensione precisa della finestra. Essa

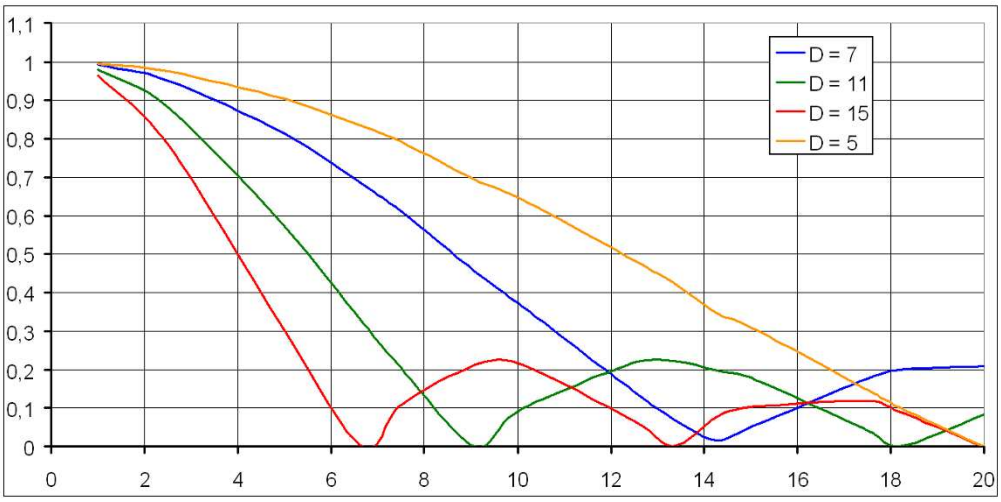


Figura 4.3: Funzione di trasferimento nel dominio della frequenza

dipende dalla frequenza del segnale che vogliamo rilevare. Ad esempio, all'inizio di questa sezione, abbiamo filtrato un segnale avente frequenza pari a 2Hz con un filtro dove D = 7. Filtrare il suddetto segnale con una finestra di dimensione pari a 15 non sarebbe corretto in quanto, come si evince dal grafico, l'ampiezza del segnale ottenuto sarebbe pari a 0,85 volte quella del segnale originale. In esempio, se il segnale ha una frequenza pari a 5Hz nessuno dei filtri presentati a nostro avviso andrebbe bene, in quanto tutti attenuerebbero almeno del 10% il segnale che vogliamo estrapolare. Oltre a ciò, la scelta della dimensione della finestra dipende anche dalla frequenza dei disturbi che colpiscono il segnale che vogliamo monitorare.
I passi dell'algoritmo che implementa questo filtro potrebbero essere i seguenti:

1. dichiarare un vettore `finestra` di dimensione `D` ed acquisire dati finché non si riempiono tutte le posizioni;
2. identificare con `i` l'indice del dato attuale;
3. acquisire ed inserire il nuovo dato nella posizione `i` del vettore e incrementare la variabile `i` come `(i+1)%D`. In `finestra` saranno così presenti sempre gli ultimi `D` dati letti;
4. calcolare il valore in uscita dal filtro pari alla somma di tutti i valori contenuti nel vettore divisa per `D`;
5. visualizzare o salvare il dato;
6. ritornare al punto 3. Ripetere finché non si decide di far terminare l'acquisizione.

Attenzione che per finestre di dimensioni elevate questo meccanismo potrebbe risultare gravoso.

Conclusioni

Abbiamo cercato di affrontare con questa tesi ogni aspetto rilevante inerente all'hardware che avevamo a disposizione. In ogni sezione ed esperimento svolto sono state inserite le relative considerazioni e conclusioni, perciò riassumeremo brevemente qui solo gli aspetti più salienti:

- sono state elencate le caratteristiche più importanti della componentistica hardware racchiusa nel brick e le interfacce implementate. Questi aspetti sono stati elencati con il ben preciso intento di gettare le basi per comprendere meglio come vengono gestiti i sensori e i motori;

- abbiamo determinato il valore del massimo momento τ erogabile dal motore al variare della velocità angolare. Questo per permettere agli utenti di prevedere se l'attuatore sarà in grado di sostenere una forza ad esso applicata oppure no;

- è stato introdotto il controllore PID fornendo una descrizione dell'implementazione software adottata e spiegando, con l'ausilio dei dati rilevati, il suo comportamento. Questo argomento è stato affrontato con l'intento di far capire agli utenti come potrebbe comportarsi il motore ed il suo sistema di controllo nelle varie situazione che si potrebbero presentare, al fine di evitare il fenomeno della saturazione;

- un'ultimo sguardo al motore è stato dato nel caso in cui sia richiesto che esso si fermi ad un angolo ben preciso. Sono stati evidenziati i problemi che si potrebbero generare se ad esso sono applicate delle forze in modo da far comprendere all'utente che questa richiesta non può sempre essere esaudita nel migliore dei modi;

- è stato presentato in maniera approfondita il funzionamento dei sensori indicando le caratteristiche dei componenti ed i principi fisici grazie ai quali essi funzionano. Tutto ciò permette all'utente di comprendere meglio le problematiche e gli errori di misura che potrebbero verificarsi durante l'utilizzo;

- da una richiesta sorta a lezione abbiamo presentato come ricavare l'inclinazione degli assi del sensore di accelerazione esprimendo delle considerazioni su quali accortezze si debbono osservare al fine di rendere preciso questo dato;

- avendo compreso che gli studenti non hanno ben chiare quali siano le grandezze che possono essere monitorate dal sensore giroscopico e di accelerazione, abbiamo presentato un esperimento che dimostrasse come utilizzare questi strumenti. Abbiamo così colto l'occasione per focalizzare l'attenzione degli studenti sulla relazione che intercorre tra lo spazio, la velocità e l'accelerazione. Oltre a questi aspetti abbiamo espresso delle conclusioni sulla sensibilità di questi strumenti;

- sono state presentate due modalità di navigazione, tachimetrica ed inerziale, con l'obbiettivo di rilevare la posizione del robot nel piano. Le due tecniche sono state paragonate concludendo lo studio sulle applicazioni del Gyro e dell'Accel Sensor. In più è stato affrontato l'argomento inerente agli integrali di funzioni discrete;

- sono state approntante delle considerazioni sui filtri digitali al fine di permetterne una semplice comprensione agli utenti della guida, essendo questi strumenti un aiuto indispensabile per l'utilizzo di alcuni dati rilevati.

Abbiamo quindi cercato di creare, con questa tesi, un percorso di approfondimento e di arricchimento culturale in merito alle tecnologie ed ai principi che permettono l'utilizzo del robot NXT, esplicando le problematiche affrontate nel modo più chiaro e semplice possibile. Ci auguriamo che la lettura abbia facilitato l'apprendimento dell'uso e della programmazione in Java di questo robot, sperando di aver risolto i dubbi delle persone che cercavano, in questo elaborato, la soluzione a problemi ed aspetti che altre guide non avevano chiarito.

Ringraziamenti

Voglio innanzitutto ringraziare la professoressa Burlin che ci ha proposto l'obbiettivo di questa tesi ed inoltre si è mossa al fine di mettere a nostra disposizione il laboratorio di robotica dell'istituto Pio X e i robot su cui lavorare. In merito a ciò ringrazio anche il Preside che ha acconsentito alla richiesta della professoressa.
Ringrazio il professor Moro per essersi dimostrato disponibile, seguendo e correggendo il mio lavoro al fine di orientarlo verso la giusta direzione, volta al conseguimento degli obbiettivi che ci eravamo inizialmente imposti.
Ringrazio il mio collega Marco Rivello che ha curato la successiva parte di questa guida e con cui ho avuto modo di confrontarmi sulle problematiche affrontate.
Ringrazio tutti i colleghi con cui ho affrontato la carriera universitaria. Essi sono stati importanti fonti di discussione e di lavoro nella preparazione degli esami.
Infine, ma sicuramente non meno importante, ringrazio la mia ragazza, Sara, per essersi dimostrata paziente in questo periodo di lavoro che ha visto sottrarre molte ore alla nostra relazione.
Saluto i lettori di questa tesi e li invito, per eventuali chiarimenti o suggerimenti, a scrivere all'indirizzo *peretz85@hotmail.it*