Progetto di Meccatronica
Robotic sorter

Tosi Ubaldo
Passerella Filippo

December 31, 2024

Corso di studi di Ingegneria elettronica

Contents

Chapter 1

Introduzione

Abbiamo realizzato il controllo di un manipolatore per smistamento di cubi colorati.

Il sistema riconosce il colore del cubo posto sull'apposita pedana e lo riposiziona nella zona appropriata. Poi torna alla posizione iniziale, in attesa del cubo seguente.

Abbiamo provato il sistema in più condizioni e non ci sono noti casi di malfunzionamento in ambiente adatto.

1.1 Divisione delle funzioni

Abbiamo deciso di dividere il programma finale in due parti, un modulo di percezione e un modulo di attuazione.

Il primo si occupa di interfacciarsi con il sensore di colore e temporizzare il secondo. Il secondo controlla il manipolatore e gestisce i percorsi.

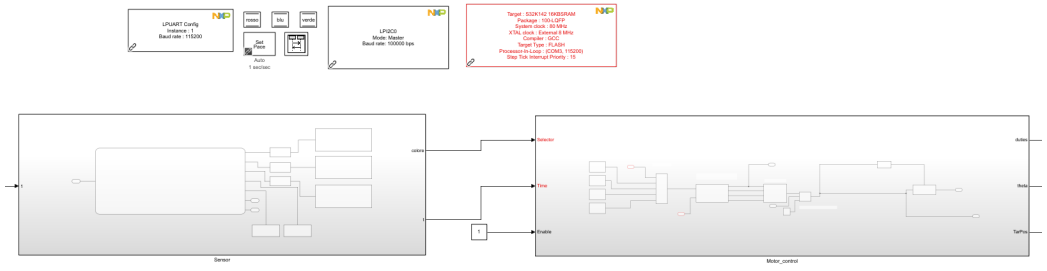

Figure 1.1: Moduli

Le funzioni dei due moduli erano sufficientemente indipendenti da poter essere sviluppati e verificati parallelamente.

Ottenuti dei prototipi funzionanti dei due moduli, abbiamo definito l'interfaccia che ha poi determinato la loro versione finale.

1.2 Modalità d'uso

Innanzitutto, occorre collegare l'hardware e accendere il dispositivo.

Si vedrà il braccio posizionarsi nella configurazione di attesa.

Posizionare un cubo colorato sulla pedana e premere *sw2*.

Il manipolatore sposterà il cubo nella zona appropriata e tornerà nella configurazione di attesa.

Chapter 2

Percezione

2.1 Sensore di colore

Il PCB del sensore di colore è costituito da

- il sensore di colore vero e proprio (TAOS TCS3472)
- un led bianco (che permette di illuminare gli oggetti per misurare il colore della luce riflessa), spegnibile nel caso in cui si voglia misurare il colore di uno schermo o comunque qualcosa che emette luce propria
- un regolatore $5V \rightarrow 3.3V$, che ci permette di alimentare il sensore a 5V e di usare la I2C a 5V per comunicare con il sensore (grazie a due transistori messi come in figura 2.2)
- resistenze di pull-up

2.2 TAOS TCS3472

Il sensore è costituito da una matrice 3x4 di fotodiodi con un filtro rosso, blu, verde e non filtrati (luce bianca). Tutti i fotodiodi hanno un filtro ad infrarossi per aumentare l'accuratezza.

La corrente generata da questi fotodiodi viene amplificata da un amplificatore trans-resistivo a guadagno variabile e campionata da un ADC integrativo a 16 bit per ogni colore. Il tempo di integrazione è programmabile.

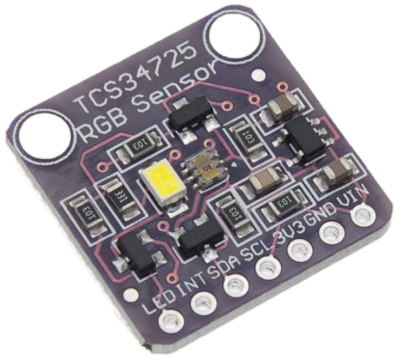

Figure 2.1: PCB Sensore

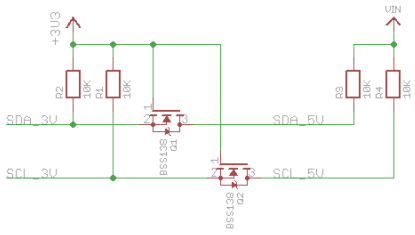

Figure 2.2: I2C

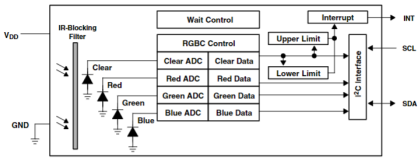

Figure 2.3: Blocco funzionale

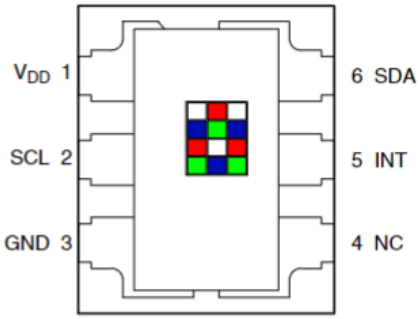

Figure 2.4: Matrice fotodiodi sensore

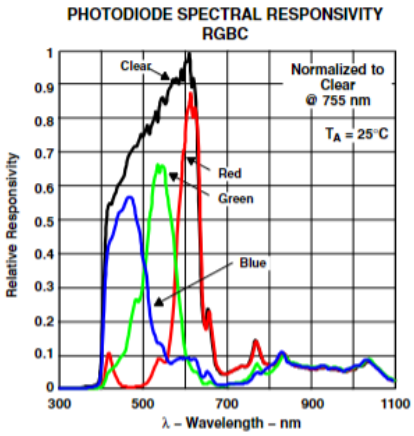

Figure 2.5: Responsività del sensore

I dati poi vengono comunicati all'esterno attraverso l'interfaccia I2C (fino a 400 KHz).

Il sensore può mandare un interrupt una volta che il valore del clear è sopra o sotto una certa threshold.

Come si può vedere dall'immagine, il sensore risponde molto bene al rosso e la responsività diminuisce andando verso il verde ed il blu. Questo, come potremo vedere più avanti, unito al fatto che la luce del led tende al giallo, porterà a problemi nella misura.

2.3 I2C

La scrittura dei registri mediante I2C viene fatta mandando per prima cosa lo start bit, poi l'indirizzo della periferica e il write. Una volta ricevuto l'acknowledge si manda il Command code. Questo configura il registro di Command.

Nel registro di Command possiamo stabilire se far leggere sempre lo stesso registro o scorrerli e da che indirizzo iniziare. Inoltre possiamo pulire gli interrupt.

Per la lettura, una volta impostato correttamente il registro command, basta mandare lo start, l'indirizzo e il read e lui continuerà dopo ogni ack a mandare il dato successivo fino a che non riceve lo stop bit.

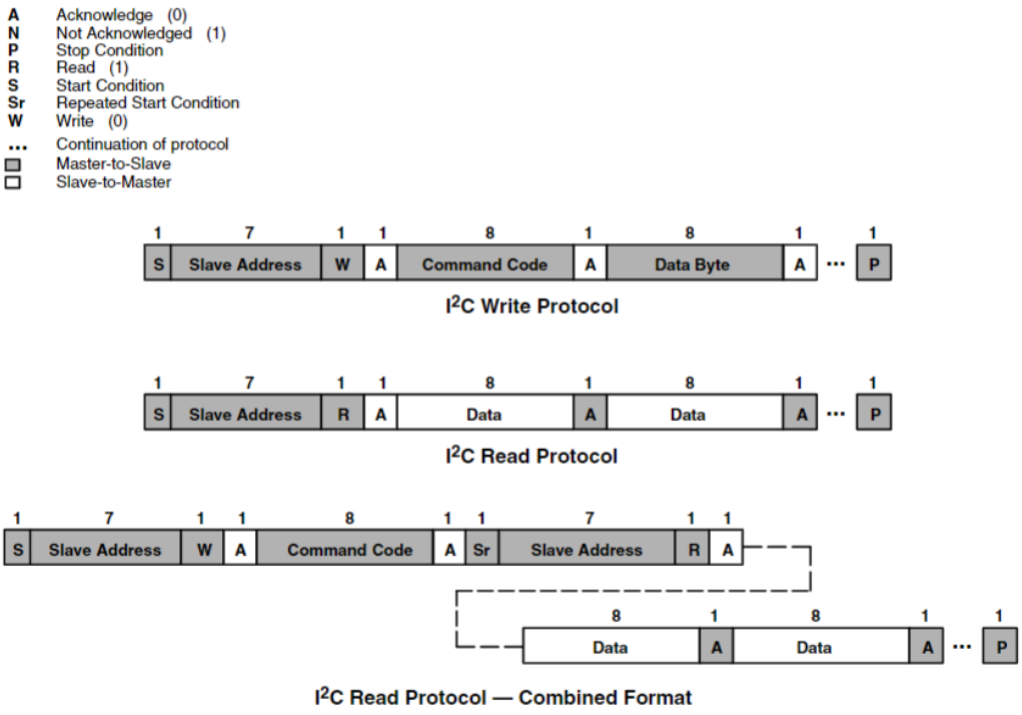

Figure 2.6: I2C

Table 4. Command Register

	7	6	5	4	3	2	1	0	
COMMAND	CMD	TYPE			ADDR/SF				--

FIELD	BITS	DESCRIPTION
CMD	7	Select Command Register. Must write as 1 when addressing COMMAND register.
TYPE	6:5	Selects type of transaction to follow in subsequent data transfers:
		FIELD VALUE / **INTEGRATION TIME**
		00 — Repeated byte protocol transaction
		01 — Auto-increment protocol transaction
		10 — Reserved — Do not use
		11 — Special function — See description below
		Byte protocol will repeatedly read the same register with each data access. Block protocol will provide auto-increment function to read successive bytes.
ADDR/SF	4:0	Address field/special function field. Depending on the transaction type, see above, this field either specifies a special function command or selects the specific control-status-data register for subsequent read and write transactions. The field values listed below only apply to special function commands:
		FIELD VALUE / **READ VALUE**
		00110 — Clear channel interrupt clear
		other — Reserved — Do not write
		The Clear channel interrupt clear special function clears any pending interrupt and is self-clearing.

Figure 2.7: Registro Command

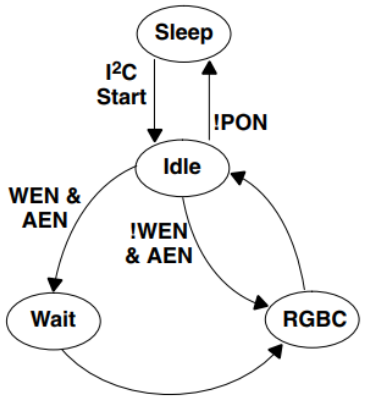

Figure 6. Simplified State Diagram

Figure 2.8: Macchina a stati

Indirizzo	Nome	Valore
—-	Command	10100000
0x00	Enable	0x03
0x01	Timing	0xC0
0x0F	Control	0x03

Table 2.1: Indirizzi settati

2.4 Macchina a Stati sensore

Il sensore per risparmiare energia rimane nello stato di sleep. Una volta che riceve lo start dall'I2C va in idle ed in base a se è impostato il bit di wait o meno fa subito la misura oppure aspetta un certo tempo.

Una volta fatta la misura torna in idle e in caso di power on disabilitato torna in sleep. La macchina a stati dipende principalmente dai parametri del registro Enable (0x00).

2.5 Parametri scelti

Solo questi registri sono stati settati in quanto abbiamo scelto di non utilizzare l'interrupt per decidere quando leggere il sensore, ma piuttosto di fare un polling alla pressione di un pulsante ed eseguire a quel punto 3 misure

Table 3. Register Address

ADDRESS	RESISTER NAME	R/W	REGISTER FUNCTION	RESET VALUE
--	COMMAND	W	Specifies register address	0x00
0x00	ENABLE	R/W	Enables states and interrupts	0x00
0x01	ATIME	R/W	RGBC time	0xFF
0x03	WTIME	R/W	Wait time	0xFF
0x04	AILTL	R/W	Clear interrupt low threshold low byte	0x00
0x05	AILTH	R/W	Clear interrupt low threshold high byte	0x00
0x06	AIHTL	R/W	Clear interrupt high threshold low byte	0x00
0x07	AIHTH	R/W	Clear interrupt high threshold high byte	0x00
0x0C	PERS	R/W	Interrupt persistence filter	0x00
0x0D	CONFIG	R/W	Configuration	0x00
0x0F	CONTROL	R/W	Control	0x00
0x12	ID	R	Device ID	ID
0x13	STATUS	R	Device status	0x00
0x14	CDATAL	R	Clear data low byte	0x00
0x15	CDATAH	R	Clear data high byte	0x00
0x16	RDATAL	R	Red data low byte	0x00
0x17	RDATAH	R	Red data high byte	0x00
0x18	GDATAL	R	Green data low byte	0x00
0x19	GDATAH	R	Green data high byte	0x00
0x1A	BDATAL	R	Blue data low byte	0x00
0x1B	BDATAH	R	Blue data high byte	0x00

Figure 2.9: Registri sensore

così da implementare un voter.

Questo perché i livelli di luminosità registrati dal sensore erano simili tra con il cubetto sopra e senza e variavano più in base al colore del cubetto che dal fatto che fosse sopra o no.

2.6 Command

Come visibile nella figura 2.7 per fare la lettura e scrittura abbiamo deciso di usare registro di Enable come primo registro e di fare le letture sequenziali degli indirizzi.

2.7 Enable

Come scritto nella sezione 2.5 abbiamo ovviamente attivato il power on e gli ADC mentre non abbiamo attivato gli interrupt e il wait.

Enable Register (0x00)

The Enable register is used primarily to power the TCS3472 device on and off, and enable functions and interrupts as shown in Table 5.

Table 5. Enable Register

	7	6	5	4	3	2	1	0	
ENABLE	Reserved			AIEN	WEN	Reserved	AEN	PON	Address 0x00

FIELD	BITS	DESCRIPTION
Reserved	7:5	Reserved. Write as 0.
AIEN	4	RGBC interrupt enable. When asserted, permits RGBC interrupts to be generated.
WEN	3	Wait enable. This bit activates the wait feature. Writing a 1 activates the wait timer. Writing a 0 disables the wait timer.
Reserved	2	Reserved. Write as 0.
AEN	1	RGBC enable. This bit actives the two-channel ADC. Writing a 1 activates the RGBC. Writing a 0 disables the RGBC.
PON [1, 2]	0	Power ON. This bit activates the internal oscillator to permit the timers and ADC channels to operate. Writing a 1 activates the oscillator. Writing a 0 disables the oscillator.

Figure 2.10: Enable Register

Non è stato attivato il wait in quanto per la nostra applicazione abbiamo preferito dare priorità alla velocità di lettura che al power management tenendo anche conto che si fanno 3 letture del sensore ogni volta.

2.8 Timing e Control

I valori dei registri di timing e control sono stati scelti in tandem in quanto sono i valori che influenzano la lettura e la sua accuratezza. Il registro di timing modifica il tempo di integrazione degli ADC e quindi influisce sulla sensibilità e sulla risoluzione della misura mentre quello di control determina il guadagno dell'amplificatore.

Inizialmente per il timing ci siamo imposti un tempo massimo per le 3 misure di 1.5 secondi. Sapendo che $Timing = (0xFF - 0xATIME) * 2.4ms$ Possiamo calcolare il $ATIME_{min} = 0x2A$.

Abbiamo provato con il massimo guadagno (x60) e abbiamo aumentato il tempo di integrazione fino a raggiungere dei valori con un buon valore massimo e minimo con tutti i colori evitando la saturazione ovvero $ATIME_1 = 0xC0$ ovvero Timing $= 154ms$ così da avere un buon compromesso tra il tempo di integrazione e la risoluzione e sensibilità dei valori misurati.

Successivamente abbiamo fatto una prova usando il massimo tempo di

RGBC Timing Register (0x01)

The RGBC timing register controls the internal integration time of the RGBC clear and IR channel ADCs in 2.4-ms increments. Max RGBC Count = (256 – ATIME) × 1024 up to a maximum of 65535.

Table 6. RGBC Timing Register

FIELD	BITS	DESCRIPTION			
ATIME	7:0	VALUE	INTEG_CYCLES	TIME	MAX COUNT
		0xFF	1	2.4 ms	1024
		0xF6	10	24 ms	10240
		0xD5	42	101 ms	43008
		0xC0	64	154 ms	65535
		0x00	256	700 ms	65535

Figure 2.11: Registro timing

Table 11. Control Register

	7	6	5	4	3	2	1	0	
CONTROL	Reserved						AGAIN		Address 0x0F

FIELD	BITS	DESCRIPTION	
Reserved	7:2	Reserved. Write bits as 0	
AGAIN	1:0	RGBC Gain Control.	
		FIELD VALUE	RGBC GAIN VALUE
		00	1× gain
		01	4× gain
		10	16× gain
		11	60× gain

Figure 2.12: Registro control

Registro	Configurazione 1	Configurazione 2
Timing	0xC0	0x2A
Control	0x03	0x02

Table 2.2: Configurazioni

Registro	Configurazione 1	Configurazione 2
Red_L	01	6A
Red_H	FF	D6
$Green_L$	31	E4
$Green_H$	2F	26
$Blue_L$	62	97
$Blue_H$	25	1F

Table 2.3: Cubo rosso

integrazione considerato accettabile ($ATIME_2 = 0x2A$ ovvero Timing = $510ms$) ed aumentato il guadagno fino ad arrivare ad un valore né troppo basso né in saturazione per ogni colore, ovvero (x16).

Vedendo che la differenza tra le due configurazioni era trascurabile, si è optato per l'uso della prima, così da risparmiare un secondo nella misura.

Un problema riscontrato è stato quello della misura sul cubo blu in quanto il colore dal valore più alto è risultato il verde ed il blu solo il secondo più alto. Provando a cambiare colore con un cartoncino più scuro il colore con il valore più alto risultava invece il rosso a causa del grande assorbimento di luce del cartoncino blu scuro. Questo è dovuto in parte al fatto che la luce del led in dotazione tende un po' verso il giallo (componente rossa più accentuata) e alla maggior responsività alla luce rossa e verde rispetto alla blu.

Registro	Configurazione 1	Configurazione 2
Red_L	2D	E4
Red_H	49	43
$Green_L$	03	2B
$Green_H$	80	79
$Blue_L$	77	F4
$Blue_H$	6C	66

Table 2.4: Cubo blu

Registro	Configurazione 1	Configurazione 2
Red_L	79	48
Red_H	5A	55
$Green_L$	CD	89
$Green_H$	80	7C
$Blue_L$	9A	6C
$Blue_H$	3D	38

Table 2.5: Cubo verde

2.9 Lettura

Si è deciso di leggere tutti i blocchi ogni volta che fosse stata necessaria la lettura, per evitare dopo la scrittura di dover riscrivere il registro command con il nuovo indirizzo di partenza, visto che il guadagno in termini di tempo era trascurabile.

Una volta eseguita la lettura di esegue la somma pesata dei valori H e L dei registri di ogni colore per avere il valore effettivo a 16 bit.

2.10 Macchina a stati

Come possiamo vedere nella figura 2.16, all'accensione viene configurato il sensore e messo colore a 4 per mettere il braccio in una posizione di default.

Successivamente si entra in uno stato di wait in cui si resettano tutti i contatori.

Premendo il pulsante sw2 si passa allo stato di reset del voter.

Si eseguono 3 letture con conseguente decisione del colore per ogni misura. A quel punto il voter decide il colore del cubo se vi sono 2 colori uguali oppure rifà le misure se sono tutti e 3 diversi.

Per finire incrementa un contatore ogni 200ms per temporizzare l'attuazione.

2.11 Voter

Per stabilire con accuratezza il colore del cubo è stato necessario implementare un voter in quanto durante i test la misura a volte risultava diversa.

Per cui abbiamo deciso di applicare una ridondanza nelle misure a scapito della velocità di misura.

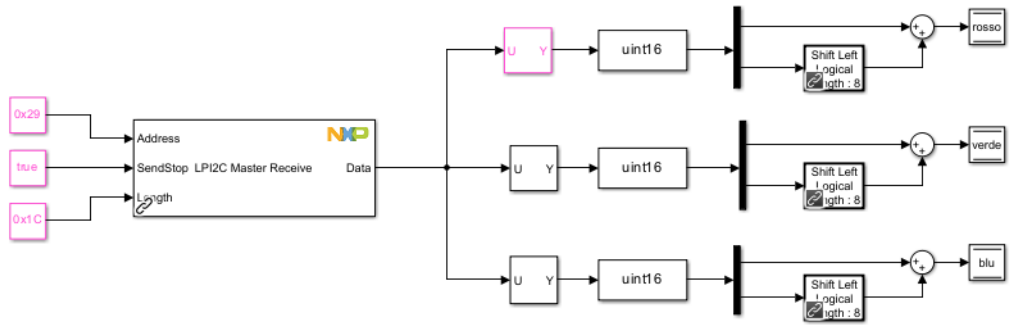

Figure 2.13: Lettura sensore

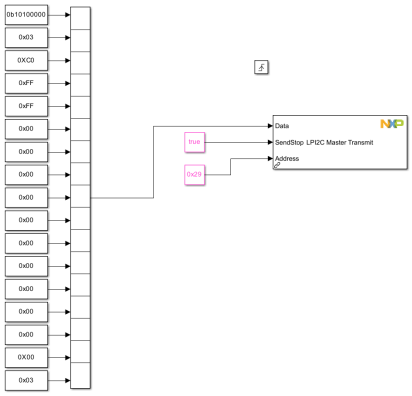

Figure 2.14: Scrittura sensore

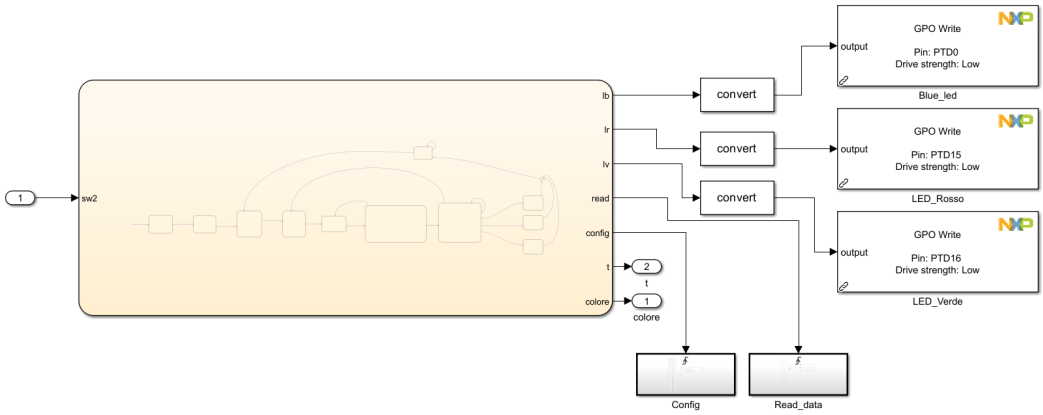

Figure 2.15: Matlab sensing

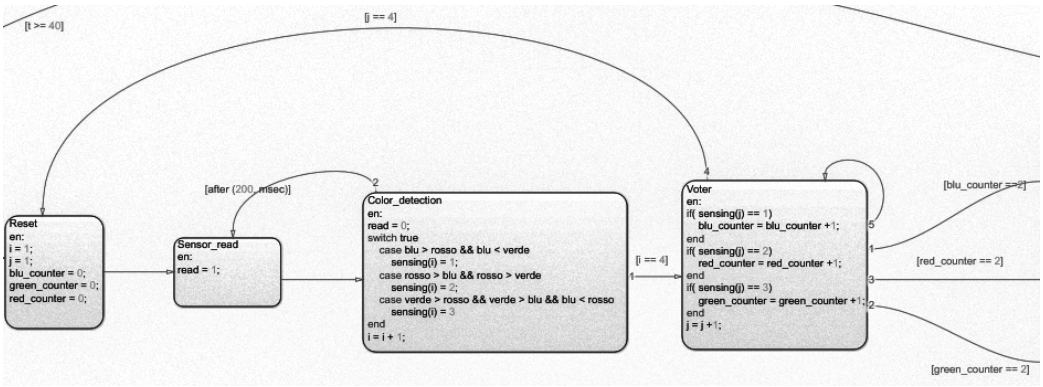

Figure 2.16: Macchina a stati

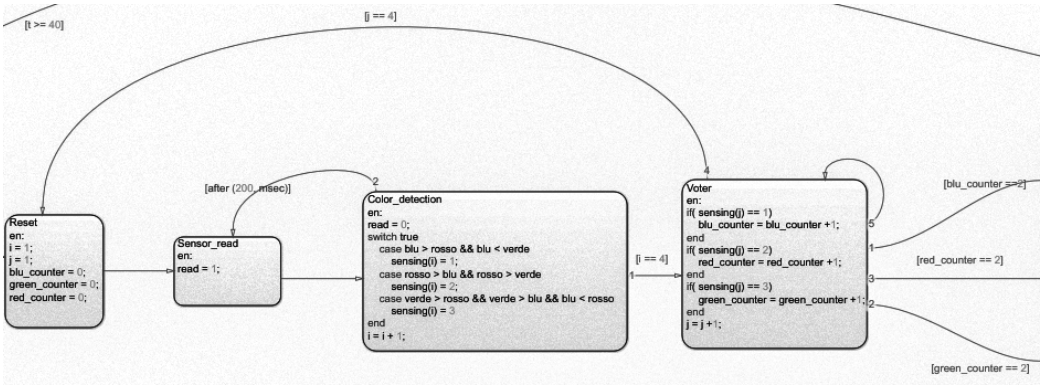

Figure 2.17: Voter

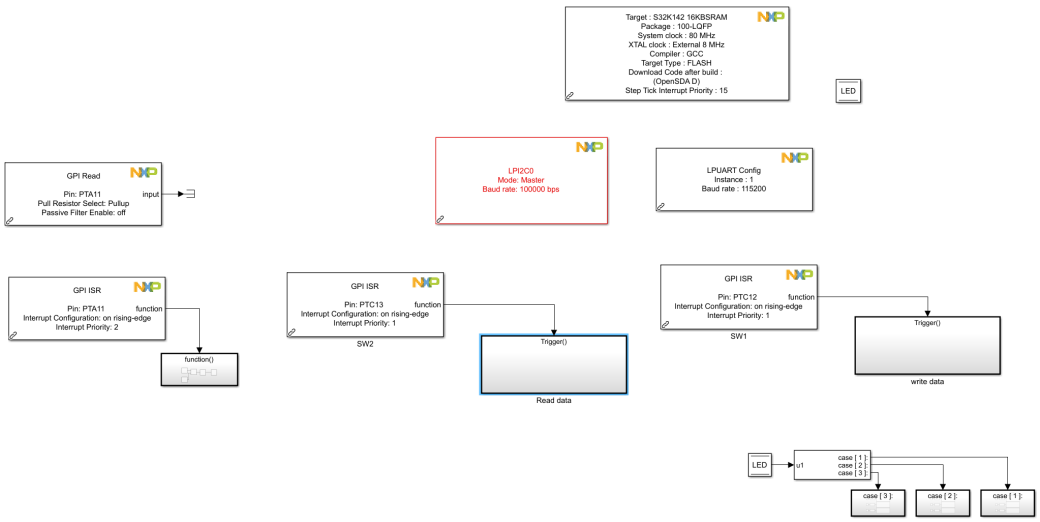

Figure 2.18: Test I2C

Per le ragioni esposte nella sezione 2.8, nel blocco **color detection** è stato necessario, per il riconoscimento del blu, mettere i valori del blu compresi tra la soglia del verde e la soglia del rosso.

2.12 Test

Purtroppo non è stato possibile fare il PIL dei blocchi contenenti la comunicazione via I2C, in quanto incompatibili.

Per eseguire i test della parte di sensore, inizialmente è stato fatto un file Matlab che si occupava solo di comunicare con il sensore e mandare i dati ricevuti via uart al pc così da poter valutare i parametri migliori per il sensore.

Una volta individuati è stata fatta la macchina a stati e testata prima in MIL e poi usando la uart e il led rgb per il debug.

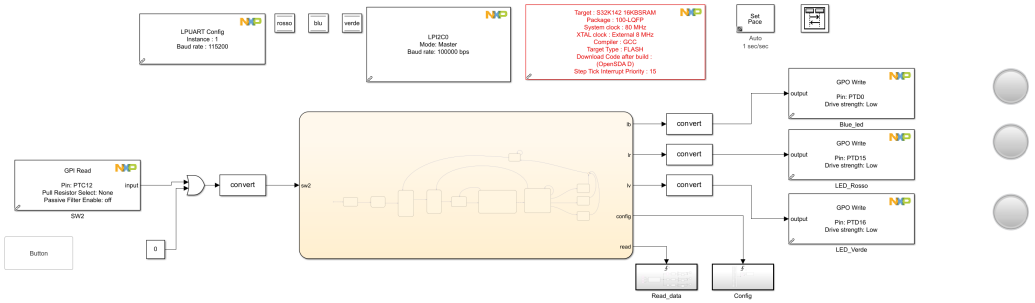

Figure 2.19: Test macchina a stati

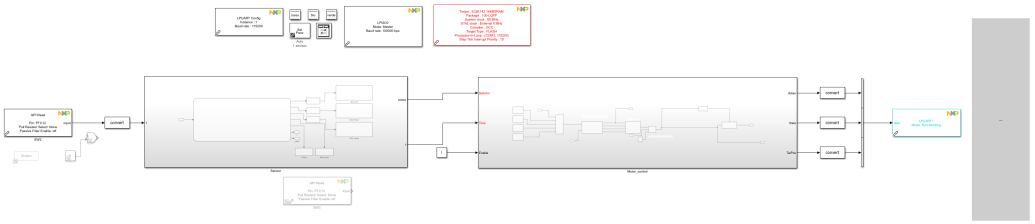

Figure 2.20: MIL completo

Chapter 3

Attuazione

Il modulo di attuazione controlla l'uscita PWM per i servo-motori per guidare il braccio robotico lungo percorsi predefiniti.

Due input determinano il percorso da seguire e il punto corrente del percorso.

Altri input e output sono usati solo in fase di debug.

Il modulo è stato ideato per permettere un controllo semplice della velocità di esecuzione e flessibilità nella scelta del percorso.

3.1 Principio di funzionamento

I blocchi **Path1...Path4** restituiscono ognuno una matrice costante che rappresenta un percorso per il braccio robotico. Ogni riga rappresenta un punto chiave del percorso. A partire dai punti chiave vengono successivamete calcolati punti intermedi.

Ogni riga contiene 7 elementi:

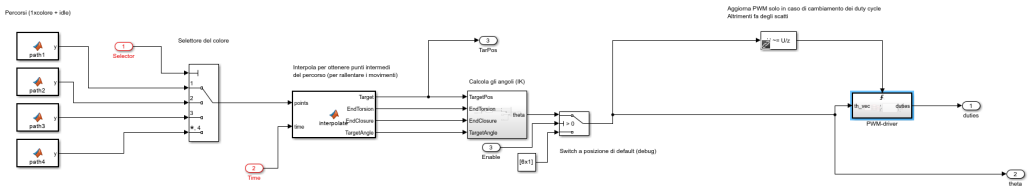

Figure 3.1: Diagramma completo

```
function y = path1()

 % [Distanza angolo(asse) verticale orientazione(pinza) polso chiusura
 % tempo
    y = [
          30  9.6 10   180  0  0    0;
          30  14  4.5   180 90  0    4;
          30  14  4.5   180 90 73    8;
          30  9.6 10   180 90 73    12;
         120  9.6 10   180 90 73    16;
         120   12  9   180 90 73   20;
         120   16  4.5   180 90 73   24;
         120   16  4.5   180 90  0   28;
         120   12  9   180 90  0   32;
          30  9.6 10   180  0  0   36;
        ];
```

Figure 3.2: Esempio di percorso

- Distanza radiale in coordinate cilindriche dell'obiettivo dal centro della base del braccio robotico.
- Angolo in cordinate cilindriche dell'obiettivo.
- Distanza verticale in coordinate cilindriche dell'obiettivo dalla base del braccio robotico.
- Angolo tra il vettore parallelo all'ultimo segmento del manipolatore (positivo verso la pinza) e la verticale (positiva verso l'alto).
- Angolo del polso.
- Angolo di chiusura della pinza.
- Istante temporale del punto nel percorso.

L'input **Selector** determina quale percorso verrà usato.

Il blocco **Interpolate** genera un punto intermedio interpolando linearmente due punti successivi nel percorso in base all'input **Time**.

Parte del blocco serve a garantire le corrette dimensioni della matrice in ingresso.

Vengono usati i punti estremi se **Time** eccede gli istanti temporali estremi.

Il blocco successivo calcola gli angoli dei servo-motori a partire dal punto interpolato.

Lo switch che segue permette di sostituire gli angoli calcolati con angoli predefiniti, per portare il manipolatore in una configurazione sicura durante il debug.

Il blocco **PWM-driver** imposta le uscite PWM a partire dagli angoli desiderati.

Questo blocco viene eseguito solo se gli angoli desiderati cambiano.

3.2 Reimpostazione periodica del PWM

Se il blocco **PWM-driver** venisse attivato liberamente (senza trigger al cambiamento) i servo-motori subirebbero uno scatto dovuto alla reimpostazione del PWM, che avviene anche quando i duty-cycle non variano.

Per questo è importante usare il blocco **Detect Change**.

3.3 Cinematica inversa

Ci sono 6 motori in totale, denominati da M1 a M6.

M6 controlla la chiusura della pinza, M5 la rotazione del polso. I motori che determinano posizione e orientazione dell'end-effector sono 4.

M2, M3 e M4 possono spostare l'end-effector solo sul piano determinato da M1. Fissata la posizione dell'end-effector, è determinata la rotazione di M1 per avere l'allineamento. Fissata anche l'orientazione finale desiderata, anche la posizione di M4 è identificata. Data questa, M2 e M3 formano uno Scara con M4 come end-effector.

3.4 Test

Il sistema di attuazione è stato prima verificato in SIL e in PIL in più iterazioni.

Sono stati progressivamente aggiunti blocchi ad ogni iterazione che ha avuto successo, in questo ordine: PWM-driver, IK, Interpolate, Selector.

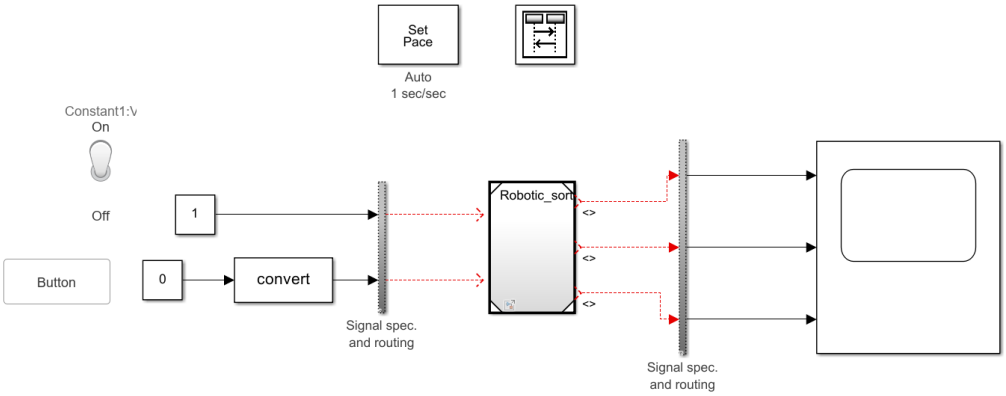

Figure 3.3: Harness del modulo di attuazione (screenshot scattato a fine progetto, differisce parzialmente da quello usato.

Le uscite di debug mostrano la posizione desiderata, gli angoli previsti e i duty cycle impostati.

Un input di debug permette di riportare il braccio in una posizione sicura ed è stato usato in PIL.

Chapter 4

Verifica finale

Sviluppati e verificati indipendentemente i due moduli, abbiamo cominciato la fase finale.

I due moduli sono stati combinati velocemente, grazie alla semplice interfaccia.

Ci siamo accorti di non poter fare simulazioni *in the loop* perché incompatibili con la periferica I2C.

Abbiamo usato la UART per verificare che tutto proseguisse come previsto durante la verifica.

Non abbiamo riscontrato problemi oltre all'impossibilità di usare SIL/PIL o i timer (per mandare messaggi temporizzati tramite UART). Così abbiamo subito potuto definire e verificare i percorsi.

Avevamo preparato delle bozze dei percorsi usando un modello del braccio realizzato e animato su Blender3D. Modificando queste in base alle verifiche abbiamo ottenuto dei percorsi che ci soddisfacessero.

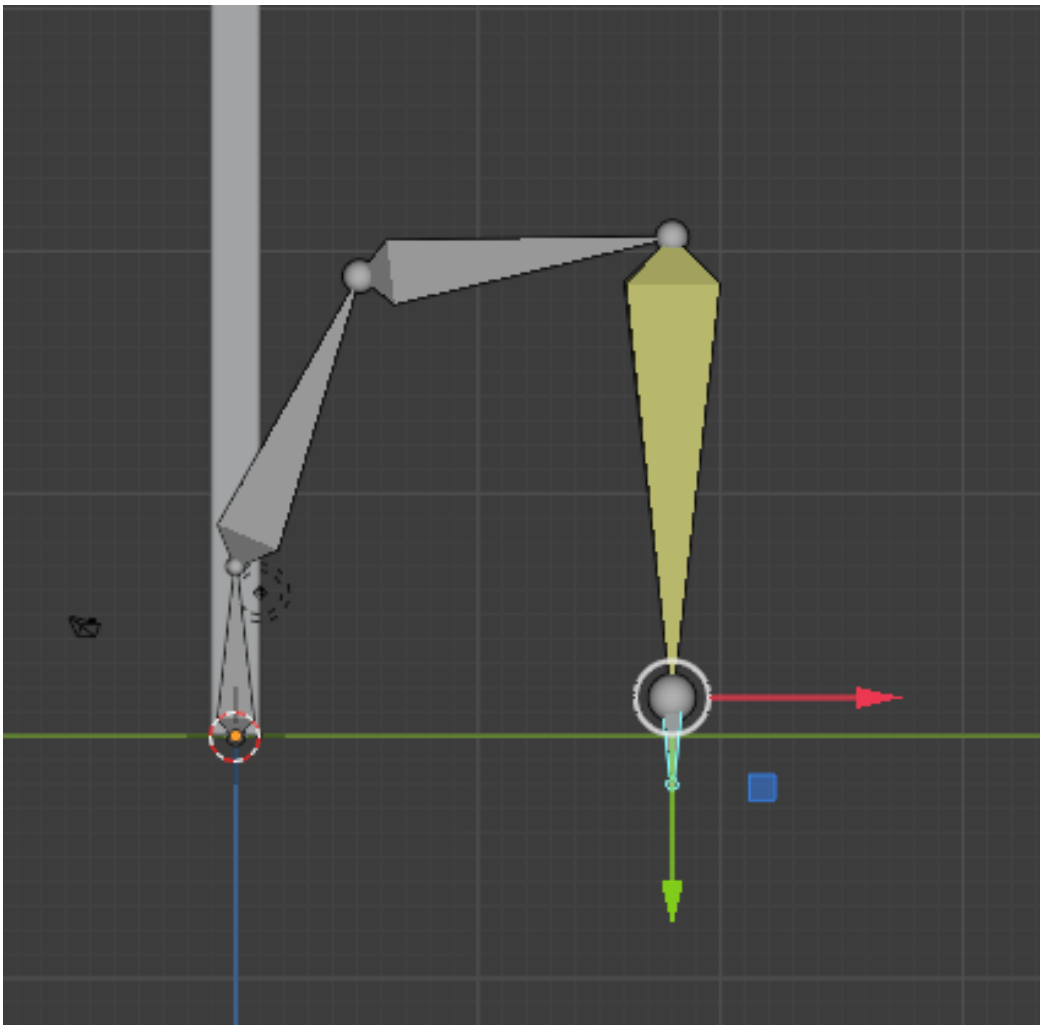

Figure 4.1: Armatura del braccio su Blender3D

Chapter 5

Problemi durante lo sviluppo

Durante i test PIL, due pezzi stampati 3D per la trasmissione della coppia da servo-motori a struttura si sono rotti. In particolare si sono spanati gli incastri per i servo-motori.

Il primo guasto è stato dovuto a un blocca di ritardo nell'harness: la condizione iniziale ha fatto piegare il braccio su se stesso all'avvio.

Il secondo guasto è stato dovuto alla base fissata male (svitando le viti si sono rovinati i fori) in seguito alle riparazioni del guasto precedente.

Chapter 6

Conclusione

Realizzata la parte funzionale, abbiamo preparato una base pieghevole su cui abbiamo montato la pedana di riconoscimento e dei cartoncini colorati per indicare le zone di smistamento.

Riguardo la parte di attuazione, i due passaggi chiave sono stati la riduzione dell'algoritmo IK a quello di uno scara e l'uso estensivo di SIL e PIL.

——————Bla bla bla

Noi possiamo ritenere con soddisfazione questo progetto un successo, dal punto di vista sia del risultato finale, sia della flessibilità e futura modificabilità della funzione, sia del processo in sé.