Università degli Studi di Parma

Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Triennale in Ingegneria Informatica

Tesi di Laurea

Progettazione e realizzazione di un carrello automatizzato "camera Dolly"

Design and implementation of an automated camera Dolly cart

Candidato:
Francesco Meli
Matricola 231818

Relatore:
Chiar.mo prof. Aurelio Piazzi

Correlatore:
Chiar.mo prof. Luca Consoli

Anno Accademico 2012–2013

al Magico garage in Via Alfieri

Indice

Capitolo 1

Introduzione

L'informatica, oggi, è ormai divenuta parte integrante della vita di ognuno di noi. Essendo una scienza in continua espansione ancora non se ne conoscono i confini.

Per Informatica si intende la scienza applicata che studia le modalità di raccolta, di trattamento e di trasmissione delle informazioni mediante elaboratori elettronici. Gli elaboratori elettronici, come una CPU di un computer o componenti meno complessi, come un microcontrollore di una semplice apparecchiatura domestica, sono dispositivi per mezzo dei quali è possibile eseguire calcoli matematici complessi, ed una svariata gamma di elaborazione di dati. Essendo stimolante l'approfondimento per altre discipline dell'Ingegneria come l'Elettronica e la Meccanica si è pensato potesse essere, questa, l'occasione per provare a sviluppare in un unico progetto unitamente allo script software, anche l'integrato elettronico hardware, oltreché progettare la struttura portante con poche, ma solide basi meccaniche.

Questo è ciò che ha portato alla progettazione e alla realizzazione di un carrello automatizzato Dolly per la creazione di video timelapse utilizzabile con macchine fotografiche Nikon.

Dolly, (dall'Inglese bambola), è una sorta di carrello sul quale viene installata una macchina fotografica digitale od una videocamera. Il movimento di questo carrello può essere controllato manualmente, oppure può essere automatizzato tramite l'utilizzo di un motore.

Timelapse invece, è una tecnica cinematografica per mezzo della quale la frequenza di cattura di ogni fotogramma scattato con una macchina fotografica digitale è molto inferiore a quella di riproduzione. A causa di questa discrepanza, la proiezione con un frame-rate standard di 24 fps, (dall'Inglese *frame per second*: fotogramma per secondo), fa sì che il tempo, nel filmato, appaia scorrere più velocemente del normale.

La combinazione di queste due funzioni appena descritte rappresenta ciò che questo progetto intende realizzare, ovvero, un sistema finalizzato alla creazione di video timelapse in movimento, che diversamente dal semplice timelapse statico, dove l'inquadratura della fotocamera rimane costante nel tempo, sfrutta il movimento lineare di un carrello, dando così luce ad un inquadratura in movimento. Si può osservare una foto di dolly in figura 1.1.

Figura 1.1: Dolly

Il complesso è automatizzato da un PIC16F887 della Microchip programmato in linguaggio C ed, in un paio di circostanze in linguaggio Assembly (per la generazione di precisi tempi di attesa). Il movimento del carrello viene generato previo il controllo del PIC16F887 su un motore passo-passo Nema23 7BYGH56 602A, l'albero del quale, congiunto ad una vite senza fine avvitata ad una flangia connessa al carrello, ne determina un movimento lineare. Il carrello provvisto di cuscinetti a ricircolo di sfere, scorre su un binario realizzato da due tubi in alluminio. L'interfaccia utente è provvista di un set di pulsanti, un encoder ed un display retroilluminato JHD162A. Il tutto verrà alimentato da una batteria al piombo 12 V 12 Ah.

Capitolo 2

Meccanica

L'obiettivo di questo progetto è stato costruire il carrello Dolly, con la prerogativa di essere facilmente trasportabile, oltreché molto robusto, in considerazione del fatto che la ripresa di video timelapse è preferibilmente svolta all'aperto per ottenere risultati migliori dal punto di vista artistico.

Inoltre, la costruzione del carrello Dolly implica vari aspetti progettuali che devono tener conto di varie esigenze di natura meccanica come la necessità di inibire, al massimo, spostamenti imprevisti onde evitare fotografie mosse, o fuori asse, rispetto all'asse verticale del terreno, relativamente alla posizione di partenza.

2.1 Il binario

La struttura fisica del binario è stata pensata ripetutamente prima di arrivare alla progettazione della versione finale.

Le problematiche principali erano sempre le medesime: ottenere un carrello solido e funzionale, utilizzando il minor quantitativo di materiale possibile; il tutto ovviamente a costi ridotti. Il prototipo più interessante prima della versione finale, consisteva nell'utilizzare come guida, due alberi in acciaio temprati di diametro 12 mm, passanti all'interno del carrello, una vite trapezoidale centrale di diametro 12 mm per determinarne il movimento e due tubi vuoti a

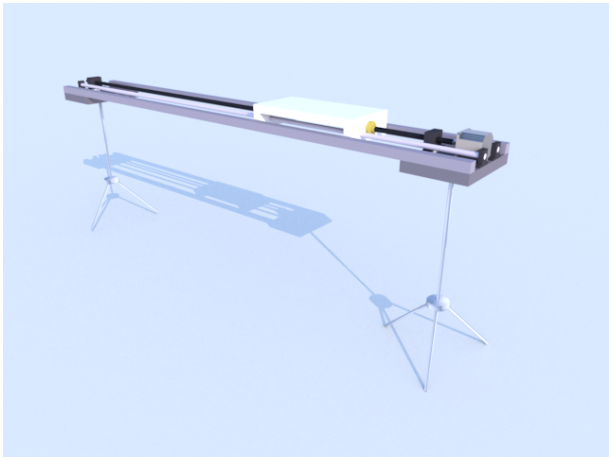

Figura 2.1: Rendering Dolly versione 1.0

sezione quadrata di lato 20 mm per tenere le due estremità portanti legate insieme. I due alberi in acciaio erano sostenuti alle estremità portanti grazie ad un supporto a T, mentre la vite era guidata da una coppia di cuscinetti SB201 sempre installati agli estremi del carrello. Per diminuire gli attriti statici si ricorreva ad una coppia di bronzine da incastrarsi all'interno del carrello, dove poter far scorrere i due alberi in acciaio. Infine, il carrello era avvitato alla vite trapezoidale, grazie all'utilizzo di una chiocciola flangiata.

Si può osservare in figura 2.1 un *rendering* del disegno progettato in SolidWorks di questa prima versione.

Tale soluzione oltre che risultare poco convincente a causa degli attriti statici abbastanza elevati, risultava inoltre essere abbastanza ingombrante, costosa e poco gradevole sotto il profilo estetico.

La versione attuale del binario è stata realizzata grazie all'utilizzo di due tubi vuoti in alluminio con diametro esterno di 25 mm e diametro interno di 23 mm. La tipologia della vite trapezoidale utilizzata rimane invariata rispetto alla versione descritta precedentemente. La differenza principale tra le due versioni, risulta essere la tecnica di fissaggio del binario agli estremi portanti. In questa versione è stato deciso di incastrare i tubi in alluminio direttamente alle estremità. Per quanto riguarda la guida della vite trapezoidale, si è ricorso ad una coppia

Figura 2.2: Rendering Dolly versione finale

di cuscinetti SKF61901 (riferimento datasheet [10]) fissati sempre ad incastro sulle estremità.

Si noti il rendering in SolidWorks della versione definitiva del binario in figura 2.2.

La versione finale del carrello permetterà al binario di sostenere pesi molto elevati sebbene il carico massimo previsto da un'applicazione come questa, non supererà mai la dozzina di chilogrammi. Infine il quantitativo di attrito statico è stato eliminato quasi completamente grazie all'uso di due cuscinetti a ricircolo di sfera incastrati all'interno del carrello.

2.2 Il carrello

Essendo in generale richiesto a questo tipo di applicazioni una precisione di scorrimento del carrello sul binario molto elevata, è stato deciso di montare ad incastro, due cuscinetti a ricircolo di sfera KH2540-PP (riferimento datasheet [7]) di diametro 35 mm.

In figura 2.3 si può capire meglio la fisica interna del carrello. Si può notare la differenza di sezione di entrata e di uscita dei fori laterali. Questi sono infatti più larghi da un lato (per permettere l'alloggiamento del cuscinetto) e più stretti dall'altro adeguandosi di fatto alla sezione dei tubi del binario. Il foro centrale invece è stato dimensionato in modo tale da permetter il passaggio della vite senza fine, che ne permette quindi il movimento. Il carrello, come il binario, è stato costruito in alluminio per ottenere un componente robusto ma allo stesso

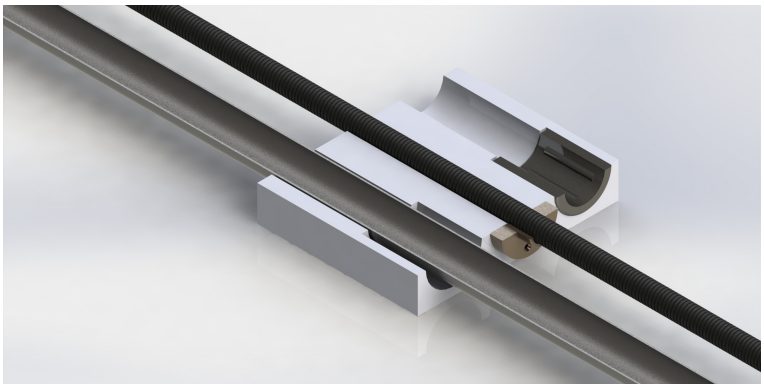

Figura 2.3: Rendering del carrello sezionato

Figura 2.4: Il carrello

tempo abbastanza leggero per facilitarne il trasporto. Per finire, al carrello è stata aggiunta una testa da treppiede per un fissaggio stabile della fotocamera sul carrello con una tenuta fino a 12 Kg.

In figura 2.4 si può osservare il componente tornito del carrello ed i vari componenti che lo compongono, assemblati.

Capitolo 3

Elettronica

3.1 Microcontrollore PIC16F887

Per questo progetto è stato utilizzato un microcontrollore PIC16F887 a 40-Pin *packaging* PDIP della MicroChip (riferimento datasheet [8]). Il packaging PDIP dall'Inglese parallel dual in-line package è una tecnica di imballaggio per dispositivi elettronici che offre un alloggiamento rettangolare con 2 file di piedini paralleli, variabili in numero dato il dispositivo.

Si può osservare uno schema rappresentante il *pinout* dell'integrato in figura 3.1.

È stato scelto questo particolare modello data la popolarità ed il quantitativo di informazioni reperibili sulla rete web. Inoltre, è stato necessario l'impiego di un componente a 40 piedini dati i numerosi componenti da interconnettere in input ed output, che saranno descritti nelle sezioni successive di questo capitolo.

Come si può osservare in figura 3.1, il PIC16F887 mette a disposizione 36 pin da utilizzarsi come I/O logici di 40 totali. I rimanenti 4 pin sono per l'alimentazione, la quale può variare da un minimo di 2.0 V ad un massimo di 5.5 V. Anche se per questo progetto non sono stati utilizzati, questo chip offre 14 registri interni per la conversione A/D che potrebbero essere sfruttati per implementare funzionalità aggiuntive in future versione di Dolly. La frequenza del microcontrollore può essere impostata tramite un oscillatore interno a 2, 4, od 8 MHz,

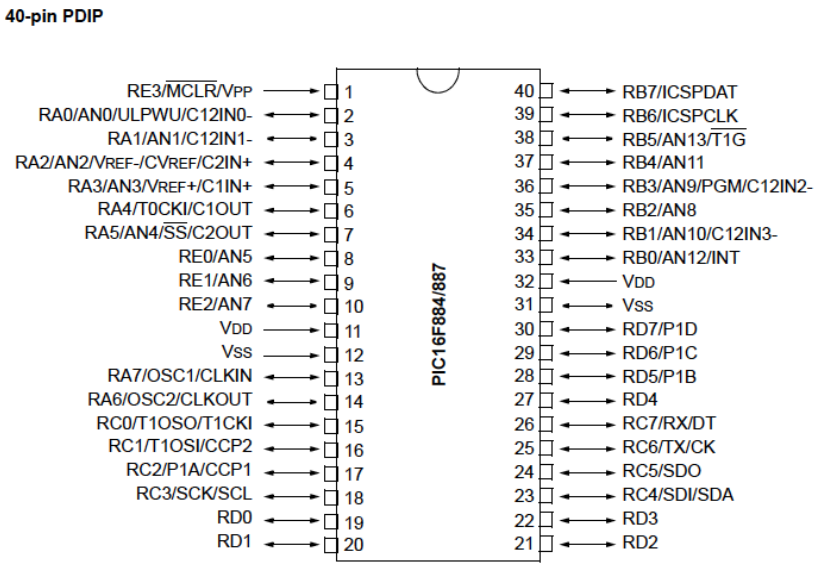

Figura 3.1: Pin out PIC16F887

oppure, se risulta necessario, fino a 20 MHz grazie all'utilizzo di appositi cristalli esterni. Per questo progetto è stato deciso di usare una frequenza a 8 Mhz invece che usare la frequenza impostata di default a 4 MHz per migliorare la modulazione del segnale IR, argomento discusso rispettivamente nella sezione 3.5.1 e nel capitolo 5.2.

Per quanto riguarda le memorie, questo PIC mette a disposizione 368 byte di memoria RAM, 256 byte di memoria EEPROM e 14,3 Kb di memoria Flash. La memoria volatile RAM è utilizzata per la memorizzazione delle variabili del programma. La EEPROM è una memoria non volatile riscrivibile un massimo di un milione di volte ed è una facoltà del programmatore se utilizzarla o meno per la memorizzazione di constanti; nello specifico in questo progetto non è stato necessario utilizzarla anche se da questa ne deriverebbe una soluzione elegante per una futura versione di Dolly. In fine la memoria Flash, è la memoria utilizzata per il salvataggio del codice sorgente.

3.2 Alimentazione

In questa applicazione la scelta del tipo di alimentazione risulta un'operazione delicata e non da sottovalutare. L'ideale per una batteria è di accumulare un'alta quantità di energia, mantenere una tensione costante, avere una piccola resistenza interna ed eventualmente avere brevi tempi di ricarica. Batterie al piombo ricaricabili come quella utilizzata, cominciano a perdere di tensione quando la carica interna risulta minore approssimativamente del 70%.

È difficile soddisfare contemporaneamente tutti questi requisiti ed è necessario, come sempre, arrivare ad un utile compromesso. Pertanto ci si è indirizzati per una scelta di batteria sovradimensionata con una tensione di 12 V ed una carica interna di 12 Ah, per non escludere periodi di lavoro molto lunghi (fino a 24 ore di ripresa video) sebbene la fotocamera adottata per la ripresa del video potrebbe comportarsi da strozzatura, potendosi scaricare prima della batteria stessa.

In fine, considerando che il PIC, insieme al ChipL298N ed al display dovranno essere alimentati con una tensione pari a 5 V, verrà utilizzato un regolatore di tensione LM7805 per ottenere una tensione adeguata.

3.3 ChipL298N - H Bridge

In riferimento al datasheet [3] ed alla guida [9] è stato possibile gestire la parte di potenza del motore in modo soddisfacente grazie al chipL298N, un circuito integrato a doppio ponte H sviluppato tramite la tecnologia TTL (dall'Inglese *Transistor Transistor Logic*). La piedinatura di questo circuito è rappresentata in figura 3.2.

Questo chip necessita di un'alimentazione Vss a 5 V per la parte logica, mentre per quanto riguarda la parte di potenza è accettata una tensione in ingresso Vs fino a 50 V con ovviamente una massa GND comune. In questo caso, si è deciso di applicare a Vs una tensione pari a 12 V. I pin di input ed output saranno opportunamente collegati rispettivamente al PIC16F887 ed al motore. Tra i pin CURRENT SENSING (sensori di corrente) e la massa saranno connesse

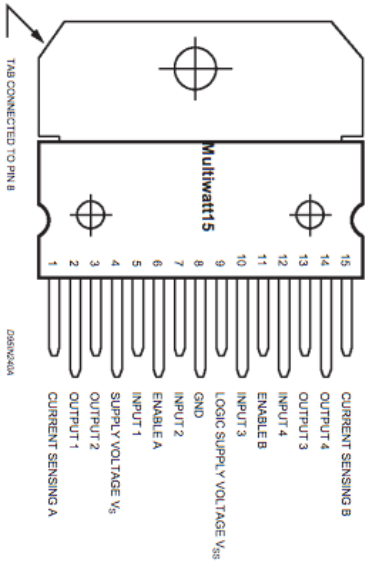

Figura 3.2: Pin out L298N

2 resistenze sensorie per controllare la corrente sul carico. I pin di ENABLE verranno tenuti sempre ad ON considerando che si vorranno i 2 ponti H sempre chiusi. Questo integrato infatti, oltre a poter controllare un motore passo-passo, potrebbe essere utilizzato anche per 2 motori in corrente continua controllati indipendentemente, ma questa tecnica non verrà discussa in questo capito.

3.4 Interfaccia utente

Nella seguente sezione verrà descritta la componentistica necessaria usata per la costruzione di un interfaccia utente adeguata.

Questa interfaccia sarà composta da un display standard 16x2 caratteri JHD162A con la quale l'utilizzatore si potrà interfacciare ad un menu utente, utilizzabile tramite un *encoder* usato per la selezione veloce di valori numerici, due bottoni rispettivamente per la selezione e deselezione di voci di menu, ed in fine un interruttore on-off per accendere e spegnere la Dolly.

3.4.1 Display JHD162A

Il display 16x2 caratteri JHD162A (riferimento datasheet [2]) è uno dei display più utilizzati con un'interfaccia utente semplice ma pur sempre efficace. Questo display è provvisto di 16 piedini come mostrato in figura 3.3.

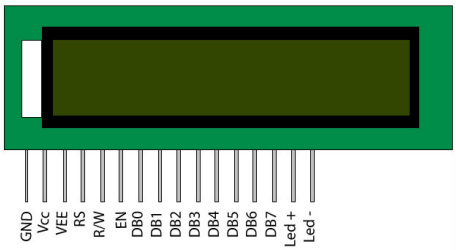

Figura 3.3: Pin out JHD162A

Quindi, le funzionalità, seguendo l'ordine di disposizione dei pin da sinistra verso destra, sono le seguenti:

2 pin GND e Vcc per l'alimentazione, 1 pin Vee per il contrasto, 1 pin RS per la selezione dei registri, 1 pin R/W per impostare lo stato di lettura/scrittura, un pin EN di enable, 8 pin (DB0 - DB7) per il trasferimento dati (istruzioni o caratteri) e 2 pin Led+ Led- per la retroilluminazione del dispositivo. Il pic RS sarà controllato in base a cosa si stà per inviare da PIC a display: istruzioni o caratteri da visualizzare. Il pin R/W sarà regolato secondo si voglia fare una lettura da display, od una scrittura: in questo caso si utilizzerà solo la modalità scrittura su display. Per quanto riguarda il trasferimento dati, questo display permette di dimezzare il numero di PIN utilizzati sul PIC (4 degli 8 disponibili) anche se questo comporta un tempo di trasferimento più lungo che però non inficia negativamente sull'applicazione in essere. Per questa applicazione si controllerà il display utilizzando solo 4 bit, dato che non ci sono esigenze particolare per quanto riguarda l'aggiornamento del display.

3.4.2 Encoder e Pulsanti

Per l'interfaccia di controllo utente sono stati utilizzati 2 pulsanti normalmente aperti per la selezione e deselezione delle voci di menu ed un encoder a 24 posizioni per la selezione dei relativi valori numerici.

I pulsanti sono connessi come mostrato in figura 3.4 nella configurazione a pull down; quindi, ad una pressione del pulsante corrisponderà un 1 logico, mentre in stato di riposo il pulsante genererà uno 0 logico (riferimento [11]).

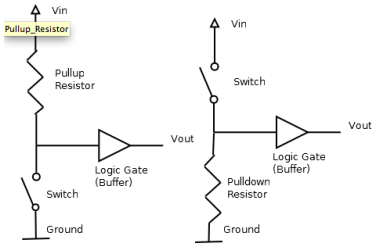

Figura 3.4: Esempio di resistenza in Pull up e resistenza in Pull down

Un encoder rotativo, è un dispositivo elettromeccanico che converte la posizione angolare o il movimento di un albero o di asse in un codice digitale. In questo caso si utilizzerà un encoder per incrementare o decrementare un valore numerico in modo rapido ma allo stesso tempo capace di selezionare valori in modo preciso. Il modello dell'encoder utilizzato è uno STEC16B03 a 24 posizioni (riferimento datasheet [6]). Questo dispositivo possiede due contatti che si aprono e chiudono secondo la posizione dell'encoder, generando così

2^n con $n = 2$

possibili stati, dove n è il numero di contatti all'interno del encoder.
In binario:

$00 \rightarrow 01 \rightarrow 10 \rightarrow 11$

Questo pattern viene quindi ripetuto 6 volte per angolo giro, dando luogo a 24 possibili posizioni con una rotazione di 360°. In figura 3.5 il pattern dell'encoder generato grazie ad un generatore online (riferimento [5]).

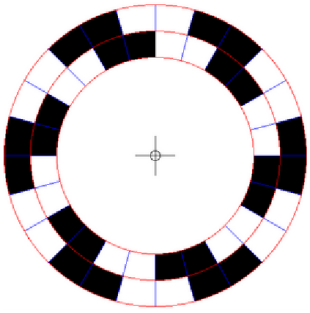

Figura 3.5: Disco di un encoder a 2bit

Si utilizzerà una tecnica di *grey coding* per interpretare il movimento rotativo.

3.5 Interfaccia macchina fotografica

3.5.1 Shutter per macchine fotografiche Nikon

In questa sezione viene affrontato lo studio della generazione e modulazione del segnale infrarosso, per far scattare una macchina reflex Nikon. La macchina fotografica utilizzata per il testing è una Nikon D40.

Grazie all'argomento trattato nel riferimento bibliografico [4] è stato possibile inquadrare il problema e risolverlo in modo opportuno anche se è stato necessario apportare leggere modifiche al codice C proposto, al fine di migliorare la modulazione del segnale. Questa fase del progetto è stata piuttosto semplice dal punto di vista elettronico dato che la modulazione del segnale è stata fatta a livello software tramite il PIC e non con un dispositivo hardware esterno.

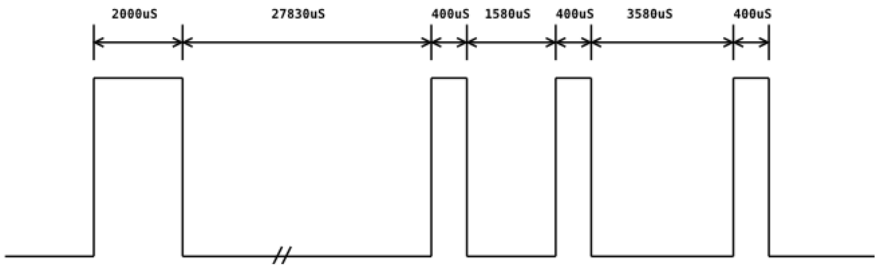

Figura 3.6: Treno di impulsi IR per scatto macchina Nikon

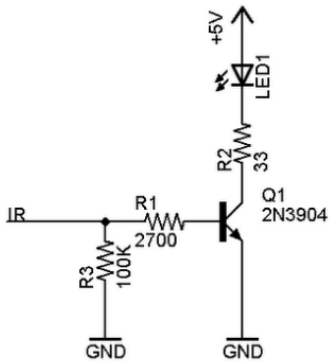

Figura 3.7: Connessione Led IR

Per quanto riguarda l'elettronica si è riscontrata una sola difficoltà, dove l'IR led, data l'alta velocità degli impulsi, non arriva mai alla completa accensione non aderendo così all'onda rappresentata in figura 3.6. Grazie all'utilizzo di un transistore NPN BD137G (riferimento datasheet [1]) utilizzato come amplificatore di corrente, è stato aumentato il flusso di corrente che attraversa il diodo led, aumentando di conseguenza la luce erogata. Grazie quindi all'applicazione di un transistore è stato possibile aumentare il raggio d'azione da approssimativamente 1 metro ad un massimo di 15 metri.

Si può vedere in figura 3.7 la connessione finale del led al PIC dove grazie ad un dimensionamento di R1 si può controllare l'amplificazione di corrente.

3.6 Circuito stampato

Il corredo necessario per la creazione di un circuito stampato comprende l'installazione di un software CAD per la progettazione di PCB (dall'Inglese *printed circuit board*), un foglio trasparente sulla quale stampare il circuito progettato, una basetta fotosensibile ai raggi UV, un bromografo dotato di lampada UV, un trapano a colonna con punte da 0.8 mm, una soluzione alcalina rilevatrice (soda caustica), una soluzione satura per incisione (cloruro ferrico) ed in aggiunta una vaschetta in plastica per il lavaggio.

3.6.1 Il progetto Eagle

Per realizzare il circuito stampato ci si avvarrà, come detto, di un software CAD per la progettazione di PCB come EAGLE, riconosciuto per essere molto pratico e ricco di librerie di componenti. Con Eagle è stata creata la schematica del circuito ed in seguito ne è stata derivata un impronta stampabile per la creazione del PCB. In figura 3.8 e 3.9 è possibile osservare rispettivamente la schematica e l'impronta stampabile derivata dalla schematica.

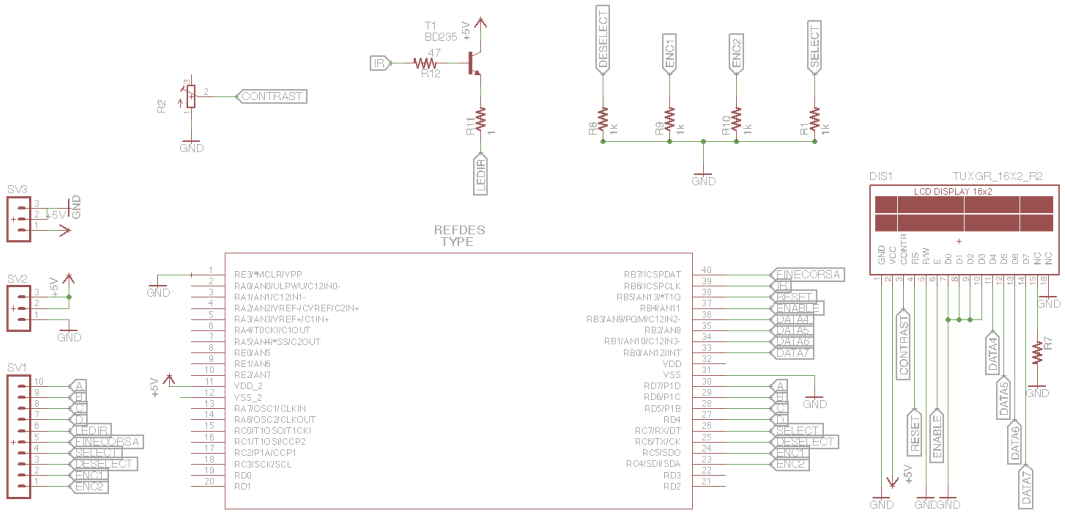

Figura 3.8: Schematica EAGLE

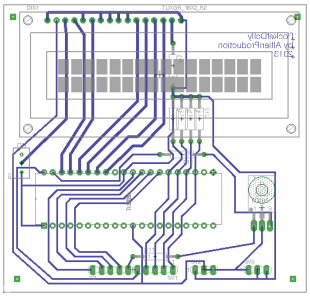

Figura 3.9: Impronta stampabile EAGLE

3.6.2 Il bromografo e gli acidi

Dopo la realizzazione dell'impronta stampabile, si passa alla fase di sbrogliatura che porterà il disegno su lucido, regolando al massimo la quantità di inchiostro. La migliore soluzione è di utilizzare una stampante laser. Una volta stampato il disegno sul lucido, lo si posiziona sulla bassetta fotosensibile dalla parte dell'inchiostro, dopo aver tolto la pellicola protettiva. Infine si posiziona la basetta all'interno del bromografo esponendola all'azione della lampada UV per approssimativamente un minuto. Sulla basetta impressionata correttamente si noterà il disegno delle piste in leggero colore verde su sfondo marrone chiaro. Dopodiché immettere la basetta nella soluzione di sviluppo per piastre fotosensibili positive, costituita da una soluzione di soda caustica con una concentrazione di 7 g/l di acqua distillata. Attendere alcuni minuti che lo strato fotosensibile illuminato dai raggi si distacchi dal rame. Quando non si stacca più nulla si può risciacquare la basetta sotto acqua di rubinetto ed in fine asciugarla. Il passo successivo è di immergere la basetta nell'acido di incisione. Come acido si può usare del comune cloruro ferrico in soluzione satura, che porterà via tutto il rame ad eccezione di quello sotto le piste coperte dal fotosensibile. Si risciacqua nuovamente la basetta ed infine la si pulisce dai residui dello strato fotosensibile utilizzando un panno con dell'alcool o dell'acetone. A questo punto si passa alla fase di foratura utilizzando le classiche punte da 0.8 mm. In figura 3.10 il risultato finale.

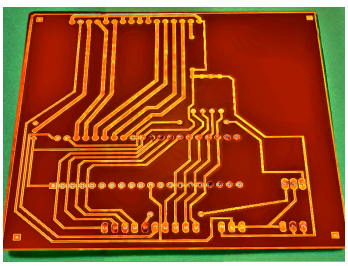

Figura 3.10: Il circuito stampato

Capitolo 4

Sensori e Attuatori

4.1 Motore stepper Nema 23 57BYGH56-602A

Il motore passo-passo spesso anche chiamato dall'Inglese, *stepper* o *stepper motor*, è un motore elettrico sincrono in corrente continua senza spazzole che può suddividere la propria rotazione in un grande numero di passi, dall'inglese *steps*.

La posizione del motore può essere controllata accuratamente senza dover ricorrere ad un controllo retroazionato (o ad anello chiuso, dall'Inglese *closed loop feedback*) purché la taglia, il tipo di motore ed il modo in cui viene controllato vengano scelti in modo adeguato. Questa appena descritta è una delle caratteristiche più note dei motori passo-passo rendendoli tra i candidati migliori per applicazioni dove la precisione nello spostamento angolare è fondamentale.

Questi sono alcuni dei dati trovati nel datasheet del Nema 23 57BYGH56-602A, motore adottato per questa applicazione:

Passo	Corrente nominale	Resistenza di fase	Induttanza di fase	Coppia	N. fili	Inerzia	Peso
$1,8°$	$2,0A$	$1,8\Omega$	$2,5mH$	$9,0Kg \times cm$	6	$300g \times cm^2$	$0,7Kg$

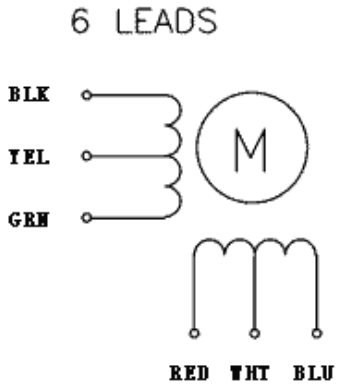

Figura 4.1: vista semplificata motore stepper a 6 fili

La figura 4.1 è una vista semplificata del motore passo-passo a 6 fili come quello utilizzato:

Quindi tramite la polarizzazione di una bobina (controllo a singola fase dall'inglese *one phase on*), o di due (controllo a doppia fase dall'inglese *two phases on*), è possibile posizionare l'ingranaggio connesso al rotore. Se poi la polarizzazione/depolarizzazione delle bobine viene eseguita coordinatamente, secondo certe sequenze, è possibile far ruotare il rotore ad una velocità angolare.

Esistono tre principali tecniche per pilotare motori passo-passo: *fullstepping*, *halfstepping* e *microstepping*, mostrate in figura 4.2.

In questo progetto, considerando che si tratteranno solo segnali digitali (onde quadre) e non analogici (sinusoidi) si utilizzerà una tecnica *halfstepping* dato un considerevole aumento in precisione, ed una buona diminuzione delle vibrazioni paragonata alla tecnica *fullstepping*. La tecnica *microstepping* rimane comunque il modo migliore per pilotare i motori passo-passo, nel caso l'utilizzo di segnali analogici fosse una possibilità.

Pertanto avendo un angolo per step equivalente a 1,8 gradi, ne consegue che il numero di step per giro può essere calcolato come segue:

$\frac{360°}{1{,}8°} = 200\frac{passi}{giro}$

Quindi ora si ha una velocità angolare calcolabile pari a

$\omega = 2\pi f$ con $f = 1/T$ dove $T = 200t$

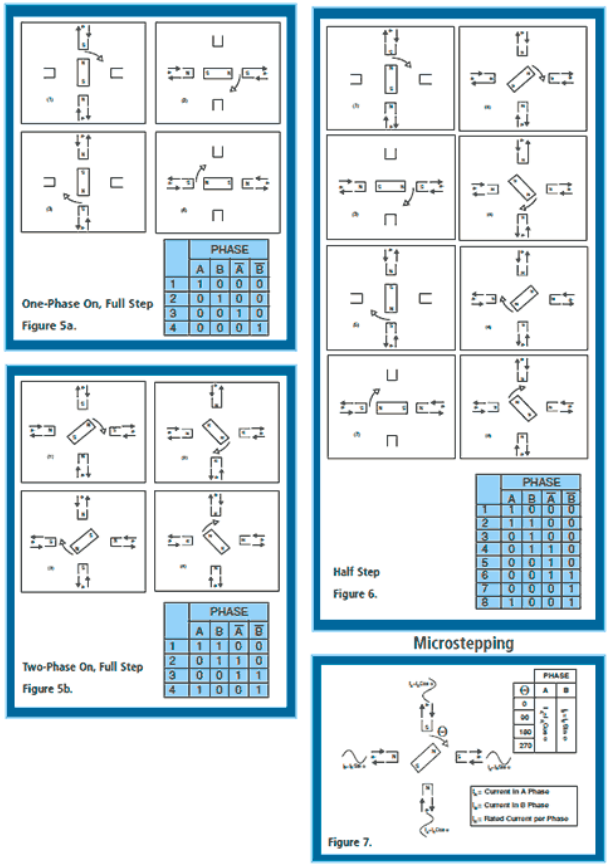

Figura 4.2: *fullstepping*, *halfstepping* e *microstepping*

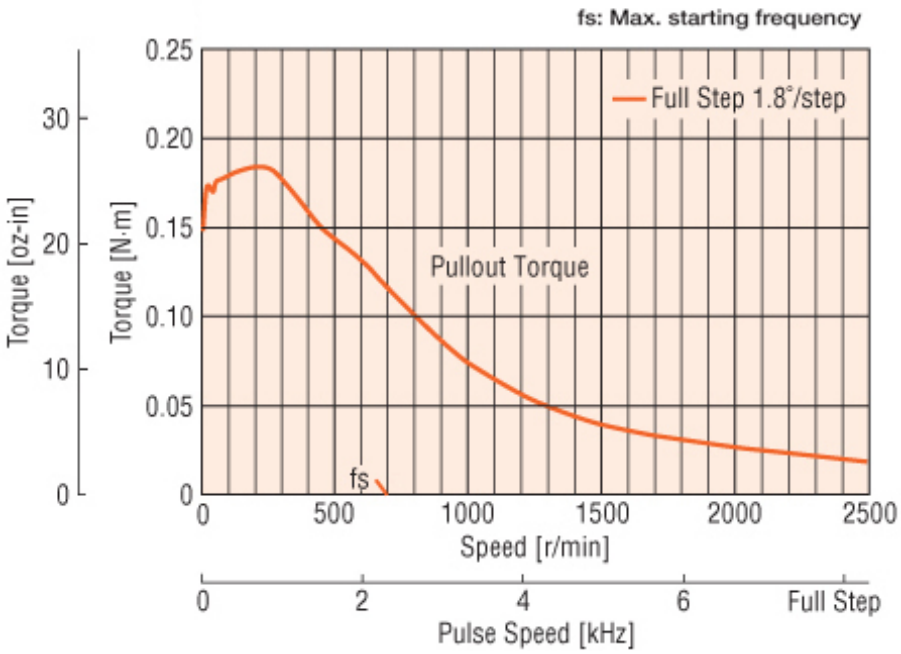

Figura 4.3: grafico della coppia in funzione della frequenza

I motori passo-passo hanno un legame tra tenuta di coppia (in inglese *toque*) e velocità di impulso (*pulse speed*), che ne aiuta il dimensionamento in fase di progettazione dell'applicazione.

Il grafico riportato in figura 4.3, rappresenta la coppia del motore 57BYGH56-602A in funzione della frequenza dei passi. Il grafico coppia velocità non è stato trovato sul datasheet del motore utilizzato. Si possono comunque considerare i dati di questo grafico sufficientemente attendibili data l'estrema similarità di questo motore con l'originale.

Sul grafico è chiaramente illustrata la velocità di impulsi del motore per la quale abbiamo la coppia massima.

Si chiarisce il concetto definendo la seguente uguaglianza:

$1Hertz = 1Passo/secondo$

Questo é ovviamente il punto di massimo della funzione e si aggira intorno ai 200 Hz ossia ai 200 passi per secondo. Avendo appurato che 200 passi sono equivalenti ad un giro completo,

questo implica che il motore scelto avrà coppia massima facendolo girare ad un giro per secondo. Un altro dettaglio importante da non trascurare è quello che nel grafico viene indicato come fs, ossia la massima frequenza di partenza; infatti per poter pilotare il motore a frequenze maggiori di fs sarà necessario accelerare il motore in modo graduale.

Sempre dal grafico si nota che la coppia del motore tende a diminuire in un modo lineare tra i 200 Hz e i 500 Hz con un coefficiente angolare leggermente minore di -1, dopodiché la coppia tende a calare più lentamente tendendo sempre più verso lo zero.

4.2 Finecorsa

L'inizializzazione della posizione del carrello rispetto al binario è stata ottenuta grazie all'utilizzo di un finecorsa fissato ad un estremo della Dolly. Lo scostamento del carrello rispetto al finecorsa sarà poi controllato a livello software grazie alle considerazioni fatte nel capitolo 4.1, dove un giro di motore equivale a $200 step$ e dato che, la lunghezza di passo della vite equivale a $3mm$, volendo convertire la lunghezza L del binario da mm a numero di step totali S si avrà

$S = \frac{L}{3} * 200 = 66.\overline{6}L$

da cui, nel caso in cui si avesse una lunghezza $L = 800mm$ si potranno contare un numero totale di passi pari a

$S = 800 * 66.\overline{6} \simeq 53\,333$

In conclusione, una volta inizializzata la posizione del carrello, la condizione necessaria da rispettare sarà di non eccedere 53 333 passi. Perché tutto ciò abbia senso è necessaria una precisa inizializzazione della posizione del carrello. Per questo motivo si è deciso di controllare l'autenticità dell'inizializzazione in modo tale da impedire all'utente di modificarla, cosa che ovviamente manometterebbe la funzionalità del sistema. Questo è stato reso possibile tramite una doppia pressione intervallata del finecorsa difficilmente ripetibile da utente.

Capitolo 5

Informatica

In questo capitolo verranno brevemente descritti gli strumenti e l'ambiente di sviluppo utilizzati per lo creazione del programma in C, ed in fine verrà proposto il sorgente del programma.

5.1 Pickit 3, MPLAB X e Hi-Tech C Compiler

L'ambiente di sviluppo utilizzato per la scrittura del programma è la versione X di MPLAB. Questo ambiente dà la possibilità di utilizzare diversi compilatori C di terze parti a pagamento o gratuiti: per questo progetto è strato scelto un compilatore che nella sua versione lite è gratuito, chiamato High Tech C. Questo compilatore però non offre librerie, come per esempio la *delay library*, che altri compilatori a pagamento offrono, come per esempio il compilatore MikroC.

5.2 Il codice sorgente

Di seguito è possibile leggere il codice sorgente in linguaggio C del progetto.

```
/*****************************************************************************
*  File:            main.c                                                   *
*  Author:          Francesco Meli                                           *
*  Created on:      May 1, 2013, 3:43PM                                      *
*  Description:     This program has been written to interface all needed    *
*                   devices (which are parts of dolly cart) and makes them   *
*                   operate to generate partially customized timelapse.      *
*****************************************************************************/
#include <stdio.h>                  // standard io library
#include <stdlib.h>                 // standard library
#include <htc.h>                    // hi tech library
#include <math.h>                   // math library

#define PIC_CLK             8000000 // set pic clock to 8Mhz for delay library
#define _XTAL_FREQ          8000000 // set pic clock to 8Mhz
#define LDOLLY              817.0   // effective length of dolly
#define LSPIN               3.0     // cart movement for every motor in [mm]
#define TSPIN               350000.0// time taken by the motor to spin once[us]
#define MOTOR1              RA0     // ports for motor
#define MOTOR2              RA1     //
#define MOTOR3              RA2     //
#define MOTOR4              RA3     //
#define EN                  RC6     // display enable pin
#define RS                  RC7     // display register select pin
#define IR                  RC5     // led IR
#define DATABITS            PORTC   // display data (used just rc0,rc1,rc2,rc3)
#define BTNENCODER1         RB4     // per contatto 1 decoder
#define BTNENCODER2         RB7     // per contatto 2 decoder
#define SELECT              RB6     // button for selecting
#define CANCEL              RB5     // button for deselecting
#define ENDLINE             RC4     // button for stopping the motor
#define TRUE                1               // define true value as 1
#define FALSE               0               // define false value as 0
#define delay_us(x)         __delay_us(x)   // define cleaner delay functions
#define delay_ms(x)         __delay_ms(x)   //
#define delay_s(x)          __delay_s(x)    //
#define fix                 0.9             // fix for delayBigMs()
#define MI                  menus.pos.mIndex// shortcut used to make
#define IN                  menus.pos.index // code cleaner
#define NOP1                asm("nop");     // equivalent to a delay of 1us
#define NOP2                NOP1 NOP1       // equivalent to a delay of 2us
#define NOP4                NOP2 NOP2       // equivalent to a delay of 4us
#define NOP8                NOP4 NOP4       // equivalent to a delay of 8us

// Configuration settings for the project: they are static options written
// directly into the HEX file that is programmed into PIC
__CONFIG(CONFIG1 & DEBUG_OFF & LVP_OFF & FCMEN_OFF & IESO_OFF & BOREN_OFF & CPD_OFF &
    CP_OFF & MCLRE_OFF & PWRTE_OFF & WDTE_OFF & FOSC_INTRC_NOCLKOUT);
```

```
__CONFIG(CONFIG2 & WRT_OFF & BOR4V_BOR21V);

/* Array of menu lengths ************************************************/
const unsigned char menuLengths[4] = {    3,      // length of Main Menu
                                          2,      // length of Setting Menu
                                          3       // length of Try Features Menu
};
/* Structure for saving the current Menu Position ***********************/
typedef struct menuPosition {
    unsigned char mIndex;               // current menu
    unsigned char index;                // current index of menu mIndex
    unsigned char set;                  // for menu voices
                                        //   with values != null this values is
                                        //   false - Im not in "set value mode"
                                        //   true - Im in "set value mode"
};
/* Structure for a menu Item ********************************************/
typedef struct menuItem {
    const char              * voice;    // menu voice
    const char              * mask;     // menu mask: it is the current scalar
                                        //   unit value [mm,us] or nothing if
                                        //   the value is not scalar
    const unsigned char     up;         // points to the above menu
    const unsigned char     indexUp;    // points to the above menu index
    const unsigned char     down;       // points to the below menu
                                        //   or to a short function to be
                                        //   executed right away
                                        //   (see function Select)
    unsigned int            value;      // if the menu voice has a value,
                                        //  it is contained here
};
/* Structure for a menu *************************************************/
typedef struct menu {
    int     start;                      // start is set as
                                        //   true - dolly has started
                                        //   false - dolly hasn't started
    struct menuPosition pos;            // defines the current position in menu
    struct menuItem * m[3];             // array of menuItems pointers
                                        //   pointing to arrays of menuItems
};
/* Structure for data collection ****************************************/
typedef struct data {
    unsigned int      steps;            // number of steps made between shots
    unsigned int      delay;            // time to sleep between shots [ms]
    unsigned int      tShots;           // number of pictures of the timelapse
    unsigned int      taken;            // number of pictures taken since the
                                        //   start of the timelpase
    unsigned char     wise;             // direction of the motor rotation
                                        //   true brings the car toward stepper
                                        //   false brings it far away from it
};
```

```
/* Function Prototyping ************************************************/

// Initialization functions
void            pic_init(void);                 // pic initialization
void            lcd_init(void);                 // lcd initialization

// Functions to manage Menu
char            getKey(char old, char new);     // get pressed key
char            read_encoder();                 // read encoder state
char            compare(char old, char new);    // check encoder direction
void            printMenu(struct menu menus);   // print menu menu
char            getMarker(char insert);         // print marker next to m. voice
struct menu     left(struct menu menus);        // action after left pressed
struct menu     right(struct menu menus);       // action after right pressed
struct menu     select(struct menu menus);      // action after select pressed
struct menu     deselect(struct menu menus);    // action after deselect press.

// Function to manage Data
struct data     getData(struct menu menus);     // get data set by user
void            start(struct data data);        // makes dolly timelapse
void            printError(void);               // prints error in values are
                                                //   not valid
// Function to manage Stepper
void            position_init(struct data data);    // cart init position
void            motorRelax(void);                   // turns motor off
void            doSteps(unsigned int n, char wise); // do n steps
void            doHSteps(unsigned int n, char wise);// do n half steps
void            spin(struct data data);             // spin motor as data says

// Function to manage IR control shutter for Nikons
unsigned int    takePic(unsigned int actShots); // send IR impulses to shoot
void            sendIRSequence();               // send IR train of impulses
void            sendIRPulseAt4Khz(int cycles);  // modulated high to camera
void            waitExactUsHex(char H, char L); // wait exactly n us

// Function to manage display lcd
void            lcd_strobe(void);               // it clears and sends data/cmd
void            cmd(unsigned char c);           // send 4bit cmd to display
void            data(unsigned char c);          // send 4bit data to display
void            string(const char *q);          // send chunked str to display
void            clear(void);                    // clears display

// try single feature functions
void            doOneSpinHalfStep(int wise);    // do one spin halfstepping
void            doOneSpinFullStep(int wise);    // do one spin fullstepping

// delay functions
void            DelayBigMs(unsigned int cnt);   // delay ms
void            DelayBigUs(unsigned int cnt);   // delay us

// debug function
```

```
void            test(void);                     // function to enclosure tests

/* Function Definition *********************************************************/

/*******************************************************************************
*  Description: Main function                                                  *
*  Algorithm:   This main function will never end.                             *
*               The function prompts the user for certain data used to         *
*               control Dolly, it starts, and once it ends it prompts the      *
*               the user again for another timelpase.                          *
*******************************************************************************/
void main(void) {

    char key,                   // key pressed by user
         old=read_encoder(),    // last state of the encoder
         new=read_encoder();    // new state of the encoder

    //note: in the following menuItems definition 255 is meant to be null

    static struct menuItem mainMenu[3] = {  // Main Menu definition
        {"Settings","",255,255,1,255},
        {"Try Features","",255,255,2,255},
        {"Start","",255,255,254,255}
    };

    static struct menuItem settings[2] = {  // Settings Submenu definition
        {"Total Time","",0,0,255,10800},    //
        {"Total Shots","",0,0,255,600}      //
    };

    static struct menuItem try[3] = {       // Try Features Submenu definition
        {"Take a Picture","",0,1,253,255},  //
        {"Spin HalfStep","",0,1,252,255},
        {"Spin FullStep","",0,1,251,255}
    };

    struct menu menus = {                   // MyMenu definition
        0,                                  // start
        {0,0,0},                            // menu positions
        {mainMenu,settings,try}             // bidimentional array of menuItems
    };

    struct data data = {0,0,0,0,0};         // it will collect data set by user

    pic_init();                             // pic initialization
    lcd_init();                             // lcd initialization
    motorRelax();                           // say relax to motor
    //test();                               // used for testing purposes


    while (1) {
```

```
printMenu(menus);                    // print menu on display lcd:
                                     //   this function will be called
                                     //   later on, only if the user
                                     //   presses a button otherwise
                                     //   no lcd update is needed

while(menus.start == 0) {            // while user hasn't started

    new=read_encoder();              // read new state of the encoder

    key=getKey(old,new);             // get button or encoder change of
                                     //   state

    old=new;                         // last state becomes the new read

    switch (key) {                   // statment to address the action
        case 0: {
            break;
        }
        case 1: {
            menus=left(menus);       // user pressed left
            printMenu(menus);
            break;
        }
        case 2: {
            menus=right(menus);      // user pressed right
            printMenu(menus);
            break;
        }
        case 3: {
            menus=select(menus);     // user pressed select
            printMenu(menus);
            break;
        }
        case 4: {
            menus=deselect(menus);   // user pressed deselect
            printMenu(menus);
            break;
        }
    }
}

data=getData(menus);          // get data from menu and put in data

if (data.delay>0)
    start(data);              // start timelapse using data set by user
else
    printError();             // print error if data is not valid

menus.start=0;                // reset start variable
```

```
    }
}
/*****************************************************************************
*  Description: This function intialize the pic. Sets:                       *
*               - Frequency of the pic throughout OSCCON register            *
*               - Analogic Selector throughout ANSELH and ANSEL registers    *
*               - Ports in TRISA, TRISB, TRISC, TRISD, TRISE to Input/Output *
*  Input:       void                                                         *
*  Output:      void                                                         *
*****************************************************************************/
void pic_init(void) {

    OSCCON = 0b11110001;    // sets pic frequency @ 8Mhz
    ANSELH = 0b00000000;    // set analogic selector to none:
    ANSEL  = 0b00000000;    //   I do all digital
    TRISA  = 0b00000000;    // Set ports to Input or Output on TRISA
    TRISB  = 0b11110000;    // Set ports to Input or Output on TRISB
    TRISC  = 0b00010000;    // Set ports to Input or Output on TRISC
    TRISD  = 0b00000000;    // Set ports to Input or Output on TRISD
    TRISE  = 0b00000000;    // Set ports to Input or Output on TRISE

    PORTA = 0;              // Reset all ports of the pic
    PORTB = 0;              //
    PORTC = 0;              //
    PORTD = 0;              //
    PORTE = 0;              //
}
/*****************************************************************************
*  Description: This function intialize the lcd.                             *
*  Input:       void                                                         *
*  Output:      void                                                         *
*****************************************************************************/
void lcd_init(void) {

    cmd(0x38);          // 8bit, multiple line, 5x7 display
    cmd(0x38);          // 8bit, multiple line, 5x7 display
    cmd(0x38);          // 8bit, multiple line, 5x7 display
    cmd(0x28);          // Function set (4-bit interface, 2 lines, 5*7Pixels)
    cmd(0x28);          // Function set (4-bit interface, 2 lines, 5*7Pixels)
    cmd(0x28);          // Function set (4-bit interface, 2 lines, 5*7Pixels)
    cmd(0x0c);          // Make cursor invisible
    cmd(0x01);          // Clear screen
    cmd(0x6);           // Set entry Mode(auto increment of cursor)
}
/*****************************************************************************
*  Description: This function "turns off" the stepper motor                  *
*  Input:       void                                                         *
*  Output:      void                                                         *
*  Algorithm:   It sets all 4 data motor inputs to 0                         *
*****************************************************************************/
void motorRelax(void) {
```

```
    MOTOR1=0;                           // Depolarize first coil
    MOTOR2=0;                           // Depolarize second coil
    MOTOR3=0;                           // Depolarize third coil
    MOTOR4=0;                           // Depolrize fourth coil
}
/*****************************************************************************
*  Description: This function prints the menu on the lcd display             *
*  Input:       void                                                         *
*  Output:      void                                                         *
*****************************************************************************/
void printMenu(struct menu menus) {

    char buffer[16];                    // will save the string to print on
                                        // first or second 16 characters row
    clear();                            // clear the screen

    //print the first line//

    sprintf(buffer,                     // set buffer to cointain a
            "%c %s",                    // set pointer ">" to the current voice
            getMarker(!menus.pos.set),
            menus.m[MI][IN].voice       // current voice
    );

    cmd(0x80);                          // set address of line one
    string(buffer);                     // print first line

    //print the second line if menu value voice exists //

    if (menus.m[MI][IN].value!=255) {
        if ((MI==1)&&(IN==0)) {
            sprintf(buffer,
                "%c H:M:S %02d:%02d:%02d",
                getMarker(menus.pos.set),
                menus.m[MI][IN].value/3600,
                (menus.m[MI][IN].value%3600)/60,
                ((menus.m[MI][IN].value%3600)%60)
            );
        }
        else
            sprintf(buffer,
                "%c %u %s",
                getMarker(menus.pos.set),
                menus.m[MI][IN].value,
                menus.m[MI][IN].mask    // mask ex: cm,s...
            );

        cmd(0xc0);                      // set address of line second line
        string(buffer);                 // print second line
    }
```

```
}
/*****************************************************************************
*  Description: This function get the key pressed by the user                *
*  Input:       void                                                         *
*  Output:      void                                                         *
*  Algorithm:   getKey, polls changes of button or encoder state             *
*****************************************************************************/
char getKey(char old, char new) {

    char btn=0;
    char encoder_wise=compare(old,new);                // compare last and new

    if (encoder_wise==(char)-1)   {btn=1;}              // polls left
    else if (encoder_wise==(char)1)    {btn=2;}         // polls right
    else if (SELECT==1)   {btn=3; while (SELECT==1);}   // polls select
    else if (CANCEL==1)   {btn=4; while (CANCEL==1);}   // polls cancel

    return btn;
}
/*****************************************************************************
*  Description: This function read the encoder using gray code               *
*  Input:       void                                                         *
*  Output:      the encoder state                                            *
*  Algorithm:   The 4 states are 00,10,11,01. One of the is going to be      *
*               returned and then will be compared with the previous state   *
*               in the main menu                                             *
*****************************************************************************/
char read_encoder() {

    char encoder=0;

    if (BTNENCODER2) encoder += 1;
    if (BTNENCODER1) encoder += 10;
    return encoder;
}
/*****************************************************************************
*  Description: Compare the last state of the encoder with the new one       *
*  Input:       last and new state of the encoder                            *
*  Output:      the encoder direction: clockwise or counter clockwise        *
*****************************************************************************/
char compare(char old, char new) {

    char encoder_wise=0;

    if(new!=old) {
       if ((new/10)==(old%10)) encoder_wise=1;
       else                    encoder_wise=-1;
    }
    return encoder_wise;
}
/*****************************************************************************
```

```
*  Description: This function manages the pression of the left button        *
*  Input:       menu                                                         *
*  Output:      menu                                                         *
*  Algorithm:   Left, checks if I'm in a "set value" stage:                  *
*               If true, decreases the value of 1 unit                       *
*               otherwise it decrease of 1, the position index of            *
*               the menu voice.                                              *
*               Error checking on low limits of set value and                *
*                 menu index.                                                *
*****************************************************************************/
struct menu left(struct menu menus) {

    if (menus.pos.set==1) {
        if (menus.m[MI][IN].value>0)            // checks limit
            menus.m[MI][IN].value--;            // decreases value
    }
    else
        if (IN>0)                               // checks limit
            IN--;                               // decrease index

    return menus;
}
/*****************************************************************************
*  Description: This function manages the pression of the right button       *
*  Input:       menu                                                         *
*  Output:      menu                                                         *
*  Algorithm:   Right, checks if I'm in a "set value" stage:                 *
*               If true, increases the value of 1 unit                       *
*               otherwise it increase of 1, the position index of            *
*               the menu voice.                                              *
*               Error checking on high limits of set value and               *
*                 menu index.                                                *
*****************************************************************************/
struct menu right(struct menu menus) {          // not commented as
                                                //  complementary function left
    if (menus.pos.set==1) {
        if (menus.m[MI][IN].value<=65534)
            menus.m[MI][IN].value++;
    }
    else {
        if (IN<menuLengths[MI]-1)
            IN++;
    }
    return menus;
}
/*****************************************************************************
*  Description: This function manages the pression of the select button      *
*  Input:       menus                                                        *
*  Output:      menus                                                        *
*  Algorithm:   Select, primarly checks the down value of the menu voice:    *
*               if down is set to a reserved value above 250, it executes a  *
```

```
*               function, otherwise goes to the submenu of index equal down.  *
*               when value is equal to 255 it means the the menu voice has    *
*               a setable value.                                               *
*******************************************************************************/
struct menu select(struct menu menus) {

    // if down is one of the reserved values
    if (menus.m[MI][IN].down>=250) {
        switch (menus.m[MI][IN].down) {
            case 254: menus.start=TRUE;          break;
            case 253: takePic(0);                break;
            case 252: doOneSpinHalfStep(TRUE);   break;
            case 251: doOneSpinFullStep(TRUE);   break;
            case 255: if (menus.pos.set==1)                // if "set value"
                          menus.pos.set=0;                 // unset
                      else menus.pos.set=1;      break;      // else set
        }
    }
    // else set position to submenu
    else {
        MI=menus.m[MI][IN].down;
        IN=0;
    }
    return menus;
}
/*******************************************************************************
*  Description: This function manages the pression of the deselect button      *
*  Input:       menus                                                          *
*  Output:      menus                                                          *
*  Algorithm:   Deselect, checks if "set value" stage in set:                  *
*               If set, unset.                                                 *
*               If not set, go to parent menu.                                 *
*******************************************************************************/
struct menu deselect(struct menu menus) {
    if (menus.pos.set==1) menus.pos.set=0;
    else {
        if (menus.m[MI][IN].up!=255) {
            IN=menus.m[MI][IN].indexUp;
            MI=menus.m[MI][IN].up;
        }
    }
    return menus;
}
/*******************************************************************************
*  Description: This function sets up the struct data, elaborating / saving    *
*                  values obtained by values in menus                          *
*  Input:       menus                                                          *
*  Output:      data                                                           *
*  Algorithm:   getData gets data from menus such as tShots                    *
*               of the timelapse. It also calculates for example,              *
*               the time to wait between one pictures and the very next one.   *
```

```
*****************************************************************************/
struct data getData(struct menu menus) {
    struct data data;
    double fatDelay, spinDelay;

    data.tShots=menus.m[1][1].value;                    // total shots to take

    data.steps=round((200/LSPIN)*(LDOLLY/(data.tShots-1)));
                                                        // between steps
    fatDelay=1000.0*menus.m[1][0].value/(data.tShots);  // in [ms]

    spinDelay=(TSPIN/1000)*((float)data.steps/200);     // in [ms]

    if (fatDelay>(spinDelay))
        data.delay=round(fatDelay-spinDelay);
    else
        data.delay=0;

    data.wise=TRUE;                                     // brings cart
                                                        //   to initial position
                                                        //   toward stepper mot.
    data.taken=0;                                       // sets taken pictures
                                                        //   to 0.
    return data;
}
/*****************************************************************************
*  Description: This function simply prints data on the screen for the user  *
*               to be updated on the state of the current timelapse.         *
*  Input:       data                                                         *
*  Output:      void                                                         *
*  Algorithm:   This function print some formatted data on the lcd display   *
*****************************************************************************/
void printData (struct data data) {

    char buffer[16];
    clear();
    sprintf(buffer,"Shots %u/%u",data.taken, data.tShots);
    cmd(0x80);
    string(buffer);
    sprintf(buffer,"St:%u Ms:%u",data.steps, data.delay);
    cmd(0xc0);
    string(buffer);
}
/*****************************************************************************
*  Description: This function prints error if the timelapse is impossible    *
*               to take with current settings.                               *
*  Input:       void                                                         *
*  Output:      void                                                         *
*****************************************************************************/
void printError(void) {
```

```
    clear();
    cmd(0x80);
    string("Error: set");
    cmd(0xc0);
    string("higher time!");

    DelayBigMs(2000);
}
/*****************************************************************************
*  Description: This function starts the timelapse with data set by user.    *
*  Input:       data                                                          *
*  Output:      void                                                          *
*  Algorithm:   This function initialize the posizion of the cart,            *
*               and then it starts a while cycle where dolly takes a picture  *
*               and afterwards it makes the motor spin and then it waits      *
*               until the time to takes the next picure comes.                *
*****************************************************************************/
void start(struct data data) {

    printData(data);                // print data on lcd display
    position_init(data);            // initialize position of the cart

    DelayBigMs(2000);

    while (data.taken<data.tShots) {    // loop as many times as pictures to take
        data.taken=takePic(data.taken); // finally take a picture... Cheers!
        printData(data);                // print updated data on lcd
        if (data.taken<data.tShots) {   // if it's not the last taken picture
            spin(data);                 // spins moving the car to some [mm]
            DelayBigMs(fix*data.delay); //   wait before looping back
        }
    }

    DelayBigMs(2000);               // time for the user to check last infos
    motorRelax();
}
/*****************************************************************************
*  Description: This function sets the initial position of the cart to d_0    *
*  Input:       data                                                          *
*  Output:      void                                                          *
*  Algorithm:   It makes the motor spin until the cart presses the ENDLINE.   *
*               This function also tries to identify if the initial poistion  *
*               is authetic or if it has been corrupted by user.              *
*               In that case ignores the pression of endline and ends the     *
*               initialization.                                               *
*****************************************************************************/
void position_init(struct data data) {
    int wise;

    if (data.wise) wise=FALSE; else wise=TRUE; // it initialize in the opposite
                                               //   direction of the one set
```

```
                                        //   by user in order to get set
                                        //   at d_0

    while (ENDLINE!=1)                  // do until endline reached
        doHSteps(8,wise);               // do half spin at a time clockwise

    if (wise==FALSE) wise=TRUE; else wise=FALSE;

    doHSteps(400,wise);                 // do one spin

    if (wise==FALSE) wise=TRUE; else wise=FALSE;

    doHSteps(400,wise);                 // comes back spinning once

    if (!ENDLINE) position_init(data);  //   and double checks if that was
                                        //   an actual authetic initilization
                                        //   if it's not keep initializing.

}
/*************************************************************************
*  Description: This function makes the stepper do full steps            *
*  Input:       n in the number of steps to do and wise can be set to true*
*                 or false and is the wise of the motor spin.             *
*  Output:      void                                                      *
*  Algorithm:   This function polarizes and depolarizes the coils inside the *
*                   stepper motor to make it step.                        *
*               Pulse is meant to be read as follows:                     *
*                   pulse[in][k] = this is the signal to polarize/depolarize *
*                   the k coil to make the step i out of 4 possible which *
*                   have to be made in a recycling matter to make the motor *
*                   spin.                                                 *
*************************************************************************/
void doSteps(unsigned int n, char w) {

    unsigned int i;

    char pulse[4][4] = {{0,1,1,0},      // fullstep 1
                        {1,0,1,0},      // fullstep 2
                        {1,0,0,1},      // fullstep 3
                        {0,1,0,1}       // fullstep 4
    };

    if (w) w=3; else w=0;               // if w (as wise):
                                        //   true: scan the array from 0 to 3
                                        //   false: scan the array from 3 to 0

    for (i=0; i<n; i++) {               // do n steps
        delay_us(TSPIN/200);            // time between steps
        MOTOR1=pulse[i%4][abs(3-w)];    // polarize first coil  MOTOR1
        MOTOR2=pulse[i%4][abs(2-w)];    // polarize second coil MOTOR2
        MOTOR3=pulse[i%4][abs(1-w)];    // polarize third coil  MOTOR3
```

```
        MOTOR4=pulse[i%4][abs(0-w)];     // polarize fourth coil MOTOR4
    }
}
/*******************************************************************************
*  Description: This function makes the stepper do half steps                  *
*  Input:       n in the number of steps to do and wise can be set to true     *
*                 or false and is the wise of the motor spin.                  *
*  Output:      void                                                           *
*******************************************************************************/
void doHSteps(unsigned int n, char w) {

    unsigned int i;

    char pulse[8][4] = {{0,1,0,0},       // halfstep 1
                        {0,1,1,0},       // halfstep 2
                        {0,0,1,0},       // halfstep 3
                        {1,0,1,0},       // halfstep 4
                        {1,0,0,0},       // halfstep 5
                        {1,0,0,1},       // halfstep 6
                        {0,0,0,1},       // halfstep 7
                        {0,1,0,1}        // halfstep 8
    };

    if (w) w=3; else w=0;                // if w (as wise):
                                         //   true: scan the array from 0 to 3
                                         //   false: scan the array from 3 to 0

    for (i=0; i<n; i++) {                // do n steps
        delay_us(TSPIN/400);             // time between halfsteps
        MOTOR1=pulse[i%8][abs(3-w)];     // polarize first coil  MOTOR1
        MOTOR2=pulse[i%8][abs(2-w)];     // polarize second coil MOTOR2
        MOTOR3=pulse[i%8][abs(1-w)];     // polarize third coil  MOTOR3
        MOTOR4=pulse[i%8][abs(0-w)];     // polarize fourth coil MOTOR4
    }
}
/*******************************************************************************
*  Description: This function makes stepper motor spin between one picture     *
*               and another                                                     *
*  Input:       data                                                           *
*  Output:      void                                                           *
*  Algorithm:   just passes the data calcualed previously to doSteps           *
*******************************************************************************/
void spin(struct data data) {

    doHSteps(data.steps*2,data.wise);        // make it step!
    motorRelax();                            // turns stepper motor off
}
/*******************************************************************************
*  Description: This function send IR signal to Nikon D40 to take a picture    *
*  Input:       number of taken pictures                                       *
*  Output:      updates number of taken pictures                               *
```

```
*  Algorithm:   Sends two trains of IR impulses to the camera and then      *
*                   increases the number of taken pictures.                 *
*  Problems:    Don't get a check feedback if the picture was actually taken *
*****************************************************************************/
unsigned int takePic(unsigned int actShots) {

    sendIRSequence();   // first sequence
    delay_ms(63);       // delay between trains
    sendIRSequence();   // second sequence
    actShots++;
    return actShots;
}
/*****************************************************************************
*  Description: This function sends a modulated IR signal to the camera      *
*  Input:       void                                                         *
*  Output:      void                                                         *
*  Algorithm:   Sends high modulated signal and waits when the signal is     *
*               is suppose to be low.                                        *
*****************************************************************************/
void sendIRSequence(void) {
    sendIRPulseAt4Khz(2250);    //  2250[us] modulated HIGH
    delay_us(27600);            // 27600[us] modulated LOW
    sendIRPulseAt4Khz(650);     //   650[us] modulated HIGH
    delay_us(1375);             //  1375[us] modulated LOW
    sendIRPulseAt4Khz(575);     //   575[us] modulated HIGH
    delay_us(3350);             //  3350[us] modulated LOW
    sendIRPulseAt4Khz(650);     //   650[us] modulated HIGH
}
/*****************************************************************************
*  Description: This function gets the length of a high pulse and modulates  *
*               it at 4Khz.                                                  *
*  Input:       Length of an high value to be modulated                      *
*  Output:      void                                                         *
*  Algorithm:   This function receives as input the length of an high signal *
*               and modulates it in the following way:                       *
*               divides the length of input signal into the length of the    *
*               period of the modulating wave: in this particular case, 25us *
*               equivalent to a 4Khz frequency.                              *
*               It loops as many times as the result of the division,        *
*               try to reach a wave form with 50% duty cycle.                *
*               This function has been calibrated with an oscilloscope.      *
*****************************************************************************/
void sendIRPulseAt4Khz(int length) {

    char cycles = length/25;        // the freqeuncy of modulation
                                    // is (1/25)[us] is equal 4Khz

    while (cycles--) {              // this loop is exactly 25us, 50% dutycycle
        IR = 1;                     // turns ir on
        NOP8; NOP8; NOP4;           // wait ideally 12.5 us
        IR = 0;                     // turn ir off
```

```
        NOP8;                       // wait ideally 12.5 us including the time
    }                               // to execute the while statment above.
}
/*****************************************************************************
*  Description: waitExactUsHex waits exactly a certain number of us define    *
*               as HEX word (2 separates bytes).                              *
*               Right above it is defined a helper to input the               *
                                                *
*               number of [us] to wait in decimal.                            *
*                                                                             *
*  Input:       number of [us] to wait                                        *
*  Output:      void                                                          *
*  Algorithm:                                                                 *
*****************************************************************************/
#define waitExactUs(x) waitExactUsHex(x/256, (x%256)/5);
void waitExactUsHex(char hByte, char lByte)
{
    char i;
    while (lByte--) { continue; }   //delay for the extra bit
    while (hByte--) {               //delay loop for the bulk of the time
        i = 61;
        while (i--) { continue; }
    }
}
/*****************************************************************************
*  Description: This function makes the stepper motor do one complete spin    *
*  Input:       Wise of the stepper motor                                     *
*  Output:      void                                                          *
*  Algorithm:   Does 200 half steps, each one equivalent to 1.8 degree to     *
*               complete a 360 degrees spin                                   *
*****************************************************************************/
void doOneSpinFullStep(int wise) {
    doSteps(200,wise);      // 200 steps clockwise or counterclockwise
    motorRelax();           // turns off motor
}
/*****************************************************************************
*  Description: This function makes the stepper motor do one complete spin    *
*  Input:       Wise of the stepper motor                                     *
*  Output:      void                                                          *
*  Algorithm:   Does 400 half steps, each one equivalent to 0.9 degree to     *
*               complete a 360 degrees spin                                   *
*****************************************************************************/
void doOneSpinHalfStep(int wise) {
    doHSteps(400,wise);     // 400 halfsteps clockwise or counterclockwise
    motorRelax();           // turns off motor
}
/*****************************************************************************
*  Description: This function triggers the enable pin of the lcd whenever     *
*               data (commands or string to be printed) is ready              *
*  Input:       void                                                          *
*  Output:      void                                                          *
```

```
*************************************************************************/
void lcd_strobe(void) {
    EN = 1;
    delay_us(1);
    EN = 0;
}
/*************************************************************************
*  Description: This function sends an 8 bit command to the lcd display *
*               broken up in two nibbles                                 *
*  Input:       command to be sent to the lcd display                    *
*  Output:      void                                                     *
*  Algorithm:   This function After setting the Register Selector to 0   *
*               sends the first 4 MSB of the command to DATABITS         *
*               and then it send the 4LSM.                               *
*************************************************************************/
void cmd(unsigned char c) {
    RS = 0;
    delay_us(50);
    DATABITS = (0b00001111&(c>>4)); // use the & statment to mask the MSB
                                    //   of DATABITS since they may not belong
                                    //   to pins dedicated for the display

    lcd_strobe();                   // enable

    DATABITS = (0b00001111&c);      // LSB
    lcd_strobe();
    delay_ms(1);
}
/*************************************************************************
*  Description: This function sends an 8 bit data to the lcd display    *
*               broken up in two nibbles                                 *
*  Input:       command to be sent to the lcd display                    *
*  Output:      void                                                     *
*  Algorithm:   This function Before setting the RS (Register Selector) to 1 *
*               sends the first 4 MSB of data to DATABITS                *
*               and then it send the 4LSM.                               *
*  Problems:    DATABITS is set to be taking an entire set o ports, say  *
*               all PORTC, meaning RC0, RC1, .. , RC7 when DATABITS      *
*               should be just the 4 LSB RC0-RC3.                        *
*               So if RS has to be set at 1 in order to send data, make  *
*               sure to set it after the definition of DATABITS not      *
*               be overwritter by DATABITS setup.                        *
*************************************************************************/
void data(unsigned char c) {
    //RS = 1;                           // This would have been overwritten
                                        // 2 lines below
    delay_us(50);
    DATABITS = (0b00001111&(c>>4));     // MSB of data
    RS = 1;                             // This is right!
    lcd_strobe();
    DATABITS = (0b00001111&c);          // LSB of data
```

```
    RS = 1;
    lcd_strobe();
}
/*****************************************************************************
*  Description: This function gets a string and sends it as data to the lcd   *
*  Input:       String to output to lcd data                                  *
*  Output:      void                                                          *
*****************************************************************************/
void string(const char *q) {
    while (*q)      //repeat till pointer of the screen is 0 aka endstring
        data(*q++); //sends out pointer char to display
}
/*****************************************************************************
*  Description: This function clear the display lcd                          *
*  Input:       void                                                          *
*  Output:      void                                                          *
*****************************************************************************/
void clear(void) {
    cmd(0x01);      //clear
}
/*****************************************************************************
*  Description: This function clear the display lcd                          *
*  Input:       void                                                          *
*  Output:      void                                                          *
*****************************************************************************/
unsigned char getMarker(char insert) {

    if (insert) return 0x3E;    //returns pointer ">"
    else return 0x20;           //returns " "
}
/*****************************************************************************
*  Description: This function delays us                                      *
*  Input:       number of us to wait                                          *
*  Output:      void                                                          *
*  Problems:    Here there is an maximum error equal to 255us so it is        *
*               important to choose a large cnt in order to obtain a          *
*               good delay.                                                   *
*****************************************************************************/
void DelayBigUs(unsigned int cnt){
    unsigned char i;

    i = (unsigned char)(cnt>>8);
    while(i>=1)
    {
            i--;
            delay_us(253);
            CLRWDT();
    }
}
/*****************************************************************************
*  Description: This function delays ms                                      *
```

```
*  Input:        number of ms to wait                                    *
*  Output:       void                                                    *
*****************************************************************************/
void DelayBigMs(unsigned int cnt){
    unsigned char    i;
    do{
        i = 4;
            do{
                delay_us(250);
                CLRWDT();
            } while(--i);
    } while(--cnt);
}
/*****************************************************************************
*  Description: This is a function to enclosure debug tests                 *
*  Input:        void                                                       *
*  Output:       void                                                       *
*****************************************************************************/
void test() {

}
/* End Function Definition ***************************************************/
```

Capitolo 6

Conclusioni

Il carrello automatizzato Dolly è regolato da un microcontrollore della famiglia Mid-range 8-bitMCUs. Il motore essendo stato selezionato per movimenti estremamente precisi rende possibile la creazione di timelapse di altissima qualità. Il controllo della macchina fotografica ad infrarosso, anziché a filo, facilita la connessione o disconnessione della fotocamera all'apparato con estrema praticità.

Essendo il carrello finalizzato alla ripresa di video timelapse, le componenti meccaniche ed elettroniche sono state costruite su misura ad eccezione di quelle con tecnologia avanzata che sono state acquistate. Per dar vita alla programmazione software e renderla operativa nel concreto s'è dovuto dar corpo ad uno pseudo robot con la necessità, di affrontare e risolvere problematiche di natura meccanica ed elettronica, oltreché informatiche.

Questo è stato certamente un percorso assai istruttivo e formativo in quanto si è progettato e sviluppato un oggetto di media complessità nella sua completezza. Il carrello automatizzato Dolly è il primo stadio di un progetto che nelle intenzioni di chi scrive avrà, in futuro, altri interessanti e più svariati sviluppi finalizzati ad applicazioni più complesse.

Bibliografia

[1] Alldatasheet.com. Bd137g. [Online; Ultimo accesso 18 Giugno 2013].

[2] Alldatasheet.com. Jhd162a series. [Online; Ultimo accesso 18 Giugno 2013].

[3] Alldatasheet.com. L298n dual full bridge driver. [Online; Ultimo accesso 18 Giugno 2013].

[4] Bigmike. The open hardware project to build an inexpensive and customizable infrared remote control for nikon cameras. [Online; Ultimo accesso 18 Giugno 2013].

[5] Bushytails. Optical encoder wheel generator. [Online; Ultimo accesso 18 Giugno 2013].

[6] Distrelec.it. Codificatori 24pos, stec16b03. [Online; Ultimo accesso 18 Giugno 2013].

[7] Emporiodelcuscinetto.it. Manicotti / cuscinetti lineari a ricircolo di sfere. [Online; Ultimo accesso 18 Giugno 2013].

[8] Microchip.com. Pic16f882/883/884/886/887. [Online; Ultimo accesso 18 Giugno 2013].

[9] Pennybuying.com. L298 dual h-bridge motor driver user's guide. [Online; Ultimo accesso 18 Giugno 2013].

[10] Skf.com. Product catalogue. [Online; Ultimo accesso 18 Giugno 2013].

[11] Tonynwk88. Digital input: Push button. [Online; Ultimo accesso 18 Giugno 2013].

Ringraziamenti

Un sincerissimo grazie ad

Andrea Torlai,

Fabio Coccomeri e

Roberto Rinaldi