SAPIENZA
UNIVERSITÀ DI ROMA

Progetto del veicolo a guida indoor autonoma Dorothy

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica

Corso di Laurea Magistrale in Ingegneria Elettronica

Marco Balsi,
Laboratorio II - Multidisciplinare

Alessandro Dei Giudici, 1398361
Pierpaolo Granello 1571747

Anno Accademico 2017/2018

Progetto del veicolo a guida indoor autonoma Dorothy
Laboratotio Multidisciplinare II - Marco Balsi. Sapienza – Università di Roma

Questa relazione è stata composta con LaTeX e la classe Sapthesis.

Versione: 27 maggio 2018

Email dell'autore: deigiudici.1398361@studenti.uniroma1.it, granello.1571747@studenti.uniroma1.it

Indice

Capitolo 1

Introduzione

Lo scopo del presente progetto è quello di simulare lo sviluppo di un veicolo a guida indoor autonoma in base alle specifiche fornite da un ipotetico committente. A partire dalle richieste espresse da quest'ultimo, in linguaggio poco tecnico e a volte parziale (*User Needs*), sono state formulate specifiche più tecniche e dettagliate (*User Requirements*). In questa fase, vengono elaborate le richieste dell'utente al fine di ottenere caratteristiche tecniche del dispositivo che l'utente può aver omesso o non considerato affatto; devono, di conseguenza, essere complete, chiare, tracciabili (deve esserci un collegamento chiaro tra esigenze e requisiti) e non ridondanti (non ce ne devono essere di più di quelli richieste).

Superata questa fase si passa allo sviluppo tecnico dove il committente non partecipa e per questo si redige una relazione sullo sviluppo delle specifiche tecniche e delle specifiche dei componenti. Le prime (*System Specification*) descrivono dettagliamente il dispositivo che verrà progettato, le seconde (*Component Specifications*) descrivono le caratteristiche dei singoli componenti necessari per la realizzazione effettiva del prototipo. Entrambe le specifiche devono anche qui presentare gli stessi attributi di completezza, tracciabilità e non ridondanza come descritto in precedenza per gli *User Requirements*.

Al fine di verificare il rispetto delle specifiche stilate sono stati sviluppati dei codici di test (*Verifica e validazione*). L'ultima fase è stata quindi quella della realizzazione fisica del dispositivo.

Capitolo 2

Creazione delle specifiche

2.1 User Needs

Di seguito verranno riportate le specifiche richieste inizialmente dall'utente:

- UN1 - veicolo autonomo capace di navigare in un ambiente indoor sconosciuto
- UN2 - piccolo veicolo che si muove in modo autonomo scegliendo il percorso più libero in un raggio di 10m tenendo conto di oggetti anche piccoli a diverse altezze
- UN3 - se urta un ostacolo deve fermarsi
- UN4 - velocità media almeno 0.2m/s, autonomia almeno 10'
- UN5 - visualizzare su un terminale remoto le misure acquisite
- UN6 - il prezzo del sistema non deve eccedere 300€

2.2 User Requirements

A partire dalle specifiche comunicate dall'utente sono stati elaborati i seguenti requisiti:

- UR01 – veicolo di dimensioni massime non oltre i 25cm, capace di portare un carico (payload) di 100g, muovendosi su una superficie liscia in ambiente occupato da oggetti artificiali.
- UR02 – il veicolo deve esplorare l'ambiente percorrendo una direzione libera nei suoi dintorni, mantenendo la stessa direzione finché è possibile senza collidere con ostacoli, e cambiando direzione quando non può procedere oltre.
- UR03 – I sensori devono poter riconoscere la presenza di un ostacolo fino a 10m di distanza con una accuratezza trasversale di 5cm a 1m di distanza, valutandone la posizione verticale con accuratezza di 10cm
- UR04 – Il carrello deve potersi muovere in modo da mantenere una velocità media di almeno 0.2m/s

- UR05 – L'autonomia di funzionamento deve essere di almeno 10'
- UR06 – Trasferire le misure acquisite attraverso connessione senza fili verso un computer remoto per visualizzarle
- UR07 – In caso di urto frontale il veicolo deve fermarsi immediatamente e cercando un nuovo percorso libero come per la UR02
- UR08 – Il prezzo del sistema (con riferimento all'ordine di 1000 pezzi) non deve eccedere 300€

Di seguito si riporta la matrice di tracciabilità tra esigenze di utente e requisiti di sistema:

Tabella 2.1. Matrice di tracciabilità tra esigenze di utente e requisiti di utente

	UN01	UN02	UN03	UN04	UN05	UN06
UR01	✓	✓				
UR02	✓	✓	✓			
UR03		✓				
UR04				✓		
UR05				✓		
UR06					✓	
UR07			✓			
UR08						✓

2.3 System Specification

A partire dalle specifiche concordate con il cliente, sono state formulate le seguenti specifiche di sistema:

- SS01 - veicolo di diametro non oltre i 25cm, capace di portare un carico (payload) di 100g, muovendosi su una superficie liscia
- SS02 – il sistema di controllo utilizza osservazioni provenienti da opportuni sensori per determinare lo spazio libero per il movimento
- SS03 - il sistema di controllo muove il veicolo in direzione rettilinea per almeno 1m purché le misure ottenute dai sensori indichino che c'è spazio sufficiente coerentemente con il movimento programmato fino alla successiva lettura dei sensori stessi
- SS04 – se le misure ottenute dai sensori indicano che non c'è spazio sufficiente per proseguire nella stessa direzione, il sistema di controllo programma una rotazione di 45° e effettua un nuovo tentativo di procedere in avanti
- SS05 - In caso di ostacolo frontalmente, il veicolo compie una rotazione sul posto a passo di 45° con direzione decisa in base ai dati acquisiti precedentemente dal LiDAR ed esegue nuovamente la scansione. Se la ricerca fallisce per 8 volte consecutive (un giro completo), si arresta a meno di un segnale di reset.

- SS06 – I sensori devono poter osservare oggetti artificiali fino a 10 m di distanza
- SS07 – I sensori devono acquisire misure di distanza eseguendo un campionamento in azimuth con risoluzione angolare non eccedente 2.5° [atn(0.05/1)] nell'ambito di un angolo complessivo di 45°
- SS08 – I sensori devono acquisire misure di distanza eseguendo un campionamento in elevazione con risoluzione angolare non eccedente 5° [atn(0.1/1)] nell'ambito di un angolo complessivo di 25°
- SS09 – Il veicolo deve poter mantenere una velocità media (su 30") di almeno 0.2m/s, tenendo conto di eventuali fermate non superiori a 2", necessarie per l'osservazione dell'ambiente
- SS10 – Il veicolo deve essere dotato di un sensore di contatto frontale e il sistema di controllo deve reagire immediatamente in caso di contatto bloccando i motori fino a individuare un percorso libero coerentemente a SS05
- SS11 - Il sistema deve essere dotato di una connessione radio per dati capace di trasferire una quantità di misure pari a quelle acquisite in base a SS07 e SS08. Il computer remoto deve essere dotato di software per acquisizione e visualizzazione delle misure
- SS12 – Il sistema deve avere una riserva di energia sufficiente per 10' di funzionamento
- SS13 – Il costo totale dei componenti non deve eccedere 150€

Tabella 2.2. Matrice di tracciabilità tra requisiti dell'utente e specifiche di sistema

	UR01	UR02	UR03	UR04	UR05	UR06	UR07	UR08
SS01	✓			✓	✓			✓
SS02		✓					✓	
SS03		✓					✓	
SS04		✓					✓	
SS05		✓						
SS06			✓					✓
SS07			✓		✓			✓
SS08			✓		✓			✓
SS09	✓			✓	✓			✓
SS10		✓					✓	
SS11					✓	✓		✓
SS12	✓			✓	✓	✓		✓
SS13				✓	✓	✓		✓

2.4 Component Specificications

A partire dalle specifiche di sistema sono state stilate le seguenti specifiche sui componenti acquistati:

- SC01 - veicolo di diametro non oltre i 25cm, capace di portare un carico (payload) di 100g, dotato di due ruote motrici più una libera oppure di due cingoli indipendenti, azionati da due motori elettrici in grado di garantire una velocità media di 0.2m/s
- SC02 - I sensori di contatto sono realizzati con 2 semplici switch in pulldown montati frontalmente al veicolo
- SC03 - Il sensore per ostacoli è costituito da un LiDAR, montato su una coppia di servo opportunamente articolati, e da due switch per l'arresto immediato del veicolo in caso di urto con un oggetto
- SC04 - La movimentazione del servo sul piano orizzontale prevede almeno 19 punti, spaziati di 2.5° da -22.5° a +22.5° rispetto alla direzione frontale
- SC05 - La movimentazione del servo sul piano verticale prevede almeno 6 punti, spaziati di 5° da 0° a +25° rispetto alla direzione frontale
- SC06 - Ad ogni sosta il LiDAR esegue una scansione completa (19x6 punti) e prende una decisione sul successivo spostamento del veicolo in meno di 2"
- SC07[1] - La connessione wireless è implementata utilizzando un trasmettitore ed un ricevitore radio funzionanti a 433 MHz.

Tabella 2.3. Matrice di tracciabilità tra specifiche dei componenti e specifiche di sistema

	SC01	SC02	SC03	SC04	SC05	SC06	SC07
SS01	✓						✓
SS02							
SS03			✓				
SS04							
SS05							
SS06			✓				
SS07			✓	✓		✓	
SS08			✓		✓	✓	
SS09	✓		✓	✓	✓	✓	
SS10		✓					
SS11							✓
SS12	✓		✓	✓	✓	✓	✓
SS13	✓	✓	✓			✓	✓

[1] In seguito ad una succeccissiva discussione con l'utente in merito alla SC07, si è concordato di elidere la richiesta UN5, date le difficoltà della sua implementazione. Conseguentemente si trascura la specifica di sistema SS11 nella realizzazione del veicolo.

Capitolo 3

List of Components

Per la costruzione del veicolo progettato, il seguente è un elenco dei componenti necessari per la sua realizzazione:

- Arduino Uno R3,
- Kit Chassis 2 ruote,
- Driver PWM per motori Pololu - TB6612FNG,
- Benewake TFMINI Micro LiDAR Module (12 m),
- DFRobot DF05BB Tilt/Pan Kit (5kg),
- 433 MHz RF Transmitter and Receiver Kit,
- 2x Condensatori ELNA 2200 μF, 35V,
- 1x Condensatore Nichicon 33 μF, 35V,
- 4x Batterie AA NiMh,
- 1x Batteria 9V,
- 2x Switch Canal M140-T01.

3.1 Arduino Uno R3

Alla base del veicolo vi è Arduino/Genuino Uno R3, una scheda di sviluppo basata su microcontrollore ATMega368P prodotto da Atmel con architettura RISC funzionante a 16 Mhz. Questa offre 32KB di memoria flash, 2KB di SRAM, 1KB di EEPROM e 14 porte di I/O: 6 di queste porte I/O possiedono funzionalità PWM, altre 6 sono porte analogiche. La piattaforma Arduino è alimentata tramite una batteria da 9V utilizzato l'ingresso DCin che presenta un regolatore di tensione a 5V idoneo all'alimentazione del microcontrollore.
Si è scelto tale microcontrollore per il numero di porte disponibili, sufficienti a soddisfare le connessioni richieste, senza eccedere in dimensioni, peso, costo e consumo e senza allo stesso tempo andare in difetto in termini di potenza di calcolo.

Inoltre, è stato scelto per la semplicità di programmazione vista la disponibilità di librerie già scritte da terzi per l'interfacciamento dello stesso con le periferiche utilizzate, riducendo quindi i costi di ingegnerizzazione iniziale.

Figura 3.1. Arduino Mega 2560

3.2 Kit chassis

La struttura del veicolo è realizzata a partire dal seguente kit in plexiglass dedicato. Nel kit sono presenti due motori con relative ruote gommate pilotati indipendente dal microcontrollore e una terza ruota in plastica pivotante. Il vano batterie raffigurato fornisce i 6 Volt necessari per alimentare LiDAR, servomotori e motori.

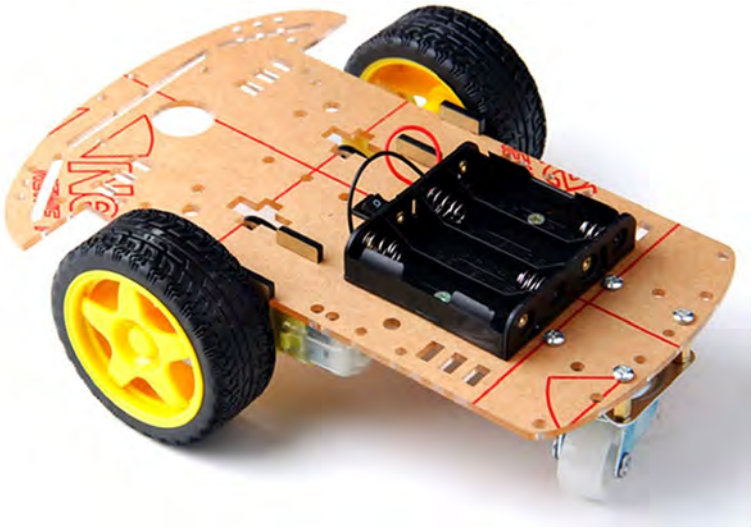

Figura 3.2. Kit chassis con 2 ruote

Si è scelto tale kit per il costo ridotto e la predisposizione parziale al montaggio dei vari componenti. Al fine di migliorare la distribuzione dei pesi, migliorare l'altezza di osservazione dei dispositivi di acquisizione e la loro mobilità, la struttura è stata comunque opportunatamente modificata.

3.3 Driver PWM per motori Pololu - TB6612FNG

A causa dell'elevato assorbimento di corrente da parte dei motori, non è possibile pilotarli direttamente dai pin PWM di Arduino in quanto la corrente richiesta è superiore a quella massima dichiarata nell'electrical datasheet del microcontrollore. Si è perciò interposto l'integrato TB6612FNG di Pololu.

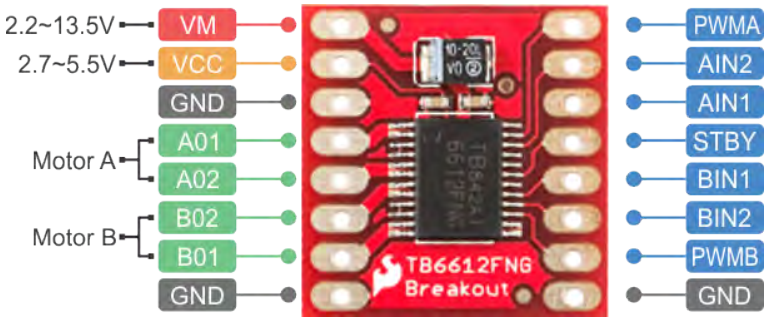

Figura 3.3. Driver ponte H TB6612FNG di Pololu

Si è scelto tale driver per la sua facile reperebilità e capacità di pilotare due motori indipendentemente con quattro modalità di funzionamento: clockwise, counter clockwise, short brake e stop mode. Il driver è capace erogare 1.2 Ampere con picchi di 3.2 Ampere, pienamente in grado quindi di pilotare le correnti richieste dai motori utilizzati (assorbimento massimo pari a 0.8 Ampere). Presenta inoltre un circuito di protezione rispetto a temperatura e cortocircuito.

3.4 Benewake TFMINI Micro LiDAR Module (12 m)

Per la ricostruzione dell'ambiente circostante al veicolo si è scelto il LiDAR TFMini Micro, prodotto da Benewake. Questo dispositivo è in grado di acquisire i dati fino ad una distanza massima di 12 metri e si collega con il microcontrollore tramite interfaccia UART. Per stabilizzare l'alimentazione durante l'acquisizione e la trasmissione dei dati è stata posta una capacità in parallelo all'alimentazione da 33 μF, 35 V prodotti da Nichicon.

Figura 3.4. Modulo LiDAR Benewake TFMINI

Si è scelto questo modello per costo, peso e dimensioni ridotti, oltre che per il basso consumo e la possibilità di condividere l'alimentazione con il resto del sistema.

3.5 DFRobot DF05BB Tilt/Pan Kit (5kg)

Per la scansione dell'amnbiente circostante il LiDAR è stato montato sul presente kit composto da due servomotori DF05BB di DFRobot: uno dedicato al movimento verticale (Tilt) e l'altro al movimento orizzonatale (Pan). Per disaccoppiare i picchi d'assorbimento di corrente durante il movimento degli stessi è stata posta per ciascuno una capacità in parallelo da 2200 μF, 35 V prodotti da ELNA.

Figura 3.5. Modulo LiDAR Benewake TFMINI

I motori sono controllati tramite PWM da Arduino. Si è scelto questo kit per il basso costo e la rapidità di montaggio.

3.6 Switch Canal M140-T01

Per le gestione degli urti frontali si è utilizzato una coppia di switch Canal M140-T01. Alla base di questa scelta, la semplice reperibilità. Gli switch sono stati collegati al microcontrollore in configurazione pulldown come in figura 3.7 per non avere consumo di corrente staticamente. In caso di urto gli switch chiudono il circuito verso massa generando una variazione dell'ingresso che verrà gestita dal microcontrollore come un interrupt.

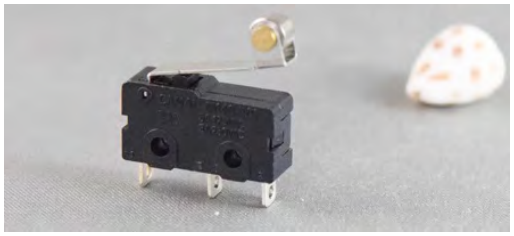

Figura 3.6. Switch Canal M140-T01

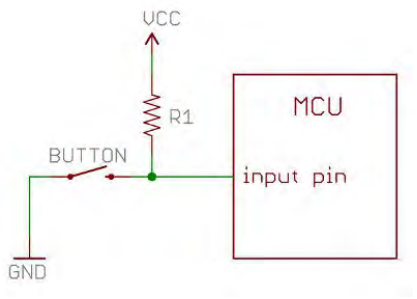

Figura 3.7. Switch PullDown

La resistenza R1 è dimensionata al fine di consumare la minore corrente possibile durante il periodo di chiusura. Volendo fissare la corrente a 1 milliAmpere , ed essendo l'alimentazione a 5 Volt si ottiene una resistenza pari a 5 kΩ.

3.7 Assemblaggio finale

Il veicolo è costituito da uno chassis in plexiglass dove sono montati direttamente i motori e i servo del LiDAR. E' presente un supporto rialzato in plastica nel lato posteriore dove sono stati fissati la scheda Arduino Uno e il driver dei motori. Anteriormente sono stati posti i due switch che cortocircuitano a massa in caso di contatto frontale fisico con un ostacolo generando un evento di interrupt sul microcontrollore. Il vano da 4 batterie AA alimenta il driver dei motori e i servomotori ed è posto sul lato inferiore della chassis. La batteria da 9V alimenta Arduino tramite il regalatore di tensione integrato che alimenta anche il LiDAR montato in posizione frontale. Le due capacità poste frontalmente, come precedentemente descritto, stabilizzano l'alimentazione a fronte dei picchi di corrente richiesti dai servomotori.

In figura 3.8 è riportato lo schema dei collegamenti elettrici del veicolo, mentre successivamente sono riportate alcune foto del prototipo realizzato:

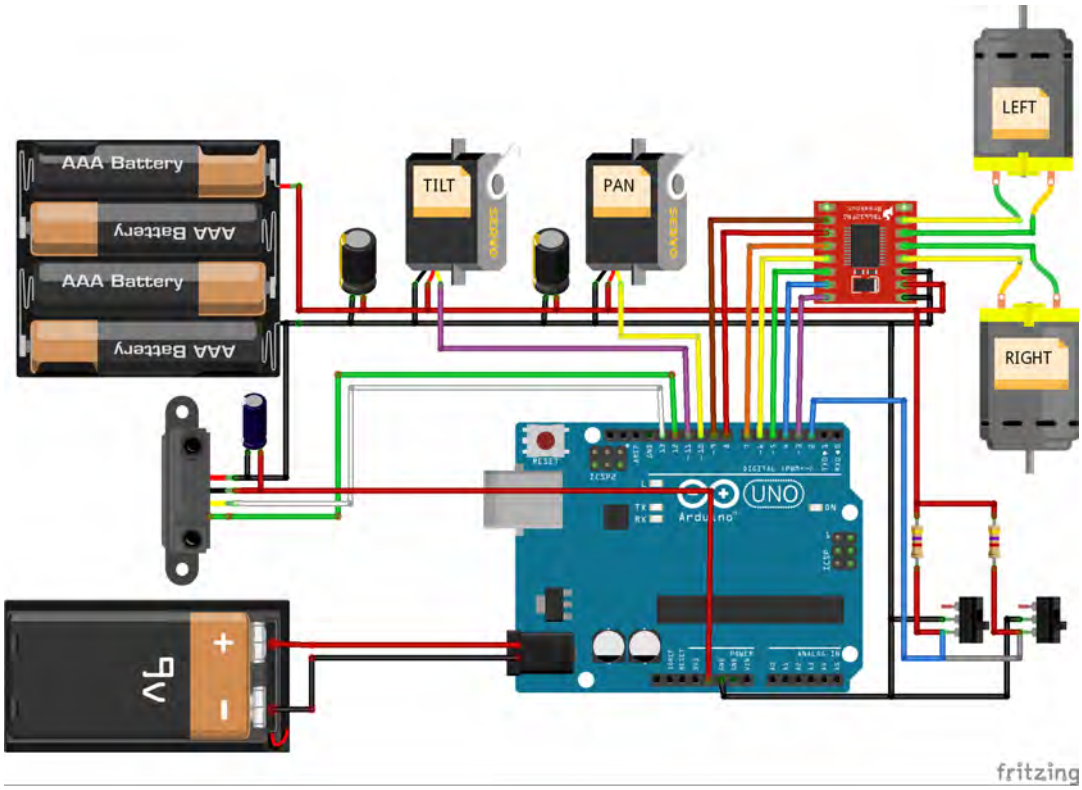

Figura 3.8. Schematico dei collegamenti del veicolo

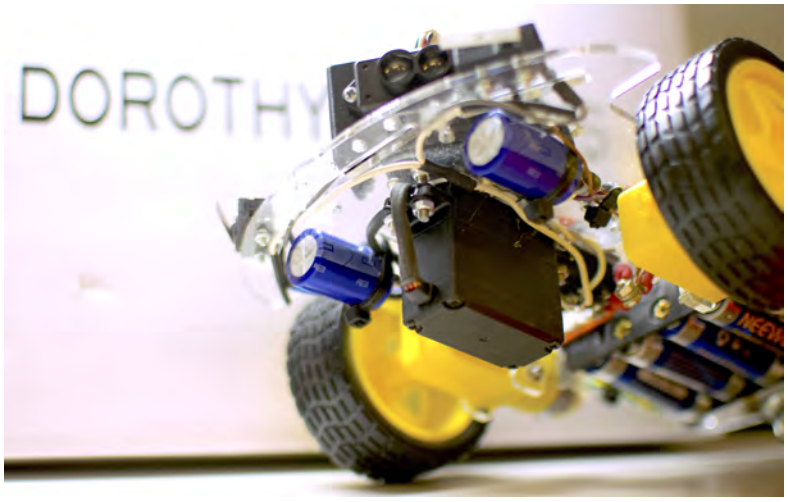

Figura 3.9. Veicolo realizzato

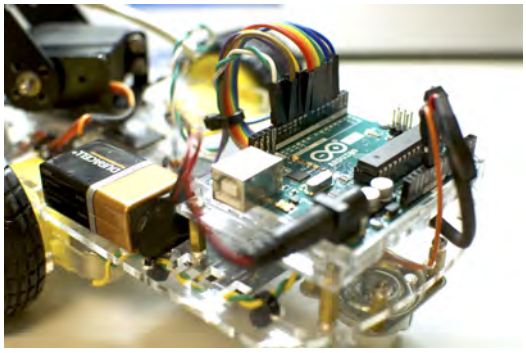

Figura 3.10. Dettaglio posteriore

Figura 3.11. Dettaglio anteriore

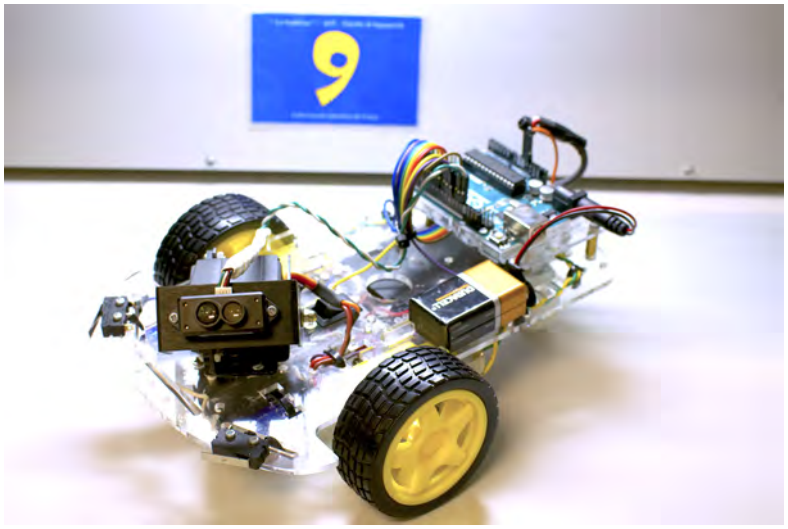

Figura 3.12. Veicolo dall'alto

3.8 Bill of Materials

Il costo dei componenti utilizzati viene riportato nella seguente tabella:

Tabella 3.1. Matrice di tracciabilità tra esigenze di utente e requisiti di utente

#	**Componente**	**Prezzo**[2]	**Venditore**
1	Arduino Uno R3	20.00€	Arduino.cc
1	Kit Chassis 2 ruote	10,90€	ebay
1	Driver Pololu - TB6612FNG	4.82€	robotshop
1	Benewake TFMINI Micro LiDAR	38.40€	robotshop
1	DFRobot DF05BB Tilt/Pan Kit (5kg)	28.34€	robotshop
1	433 MHz RF TX & RX Kit	2,40€	ittgroup
2	Condensatori ELNA 2200 μF, 35V	0.28€	Mouser
1	Condensatore Nichicon 33 μF, 35V	0.04€	Mouser
4	Batterie AA NiMh	0.42€	Mouser
1	Batteria 9V	1.02€	Mouser
2	Switch	0.10€	Mouser
30	Jumper	0.63€	Farnell
	TOTALE	107.37€	

In seguito alla valutazione del costo totale, la specifica di sistema SS13 risulta verificata.

[2]I prezzi considerati sono riferiti ad un ordine di almeno 1000 pezzi.

Capitolo 4

Verifica e validazione

I seguenti test sono stati eseguiti sui singoli componenti per verificarne il loro corretto funzionamento dopo averne controllato il corretto collegamento tramite la modalità *"Continuity Test mode"* del multimetro.

4.1 TS01: verifica della specifica di sistema SS01

- **Setup di verifica:** Veicolo posizionato su una superficie liscia con sufficiente spazio per consentire il movimento libero. Payload di 200 grammi posizionato centralmente sul veicolo.
- **Esecuzione del test:** Le dimensioni del veicolo sono state misurate con un metro. Successivamente il veicolo è stato fatto muovere con un firmware appositamente realizzato.
- **Criterio di successo:** Se il diametro misurato risulta essere minore o uguale a 25 centimetri e le ruote si muovono in modo coerente con quanto programmato, nonostante l'aggiunta del payload, il test ha esito positivo.
- **Osservazioni:** Il diametro misurato risulta essere pari a 24 centimetri e il movimento risulta fluido nonostante la presenza del payload. La specifica SS01 è verificata.

4.2 TS02: verifica delle specifiche di sistema SS02, SS06

- **Setup di verifica:** Veicolo con firmware LiDAR ed oggetto con distanza frontale dal LiDAR variata durante il test.
- **Esecuzione del test:** La distanza dell'oggetto mosso durante il test viene rilevata tramite porta seriale.
- **Criterio di successo:** Se la distanza misurata è coerente con quella reale, il test ha esito positivo.
- **Osservazioni:** Le misure rilevate risultano coerenti nel range di distanze compreso tra 0,2 e 12 metri. Le specifiche SS02 e SS06 risultano verificate.

4.3 TS03: verifica delle specifiche di sistema SS03, SS04, SS05

- **Setup di verifica:** Veicolo posto in un ambiente di test con firmware Servo (in appendice).
- **Esecuzione del test:** Ambiente inizialmente libero e successivamente occupato da ostacoli posti frontalmente al veicolo.
- **Criterio di successo:** Nel caso in cui il veicolo non rilevi ostacoli lungo il percorso e percorra un metro di distanza in rettilineo, oppure, in caso contrario, ruoti a passi di 45° fino a trovarne uno libero (per un numero massimo di 8 tentativi, altrimenti si arresta), il test ha esito positivo.
- **Osservazioni:** Il comportamento del veicolo risulta coerente con quanto richiesto. Le specifiche SS03, SS04 e SS05 risultano verificate.

4.4 TS04: verifica delle specifiche di sistema SS07, SS08

- **Setup di verifica:** Veicolo alimentato posto in un ambiente di test con firmware specifico.
- **Esecuzione del test:** Avvio di una routine in ambiente Processing. La routine plotta su un grafico i punti acquisiti dal LiDAR e ricevuti dal PC tramite collegamento con porta seriale.
- **Criterio di successo:** Se il dispositivo acquisisce correttamente i valori delle distanze degli oggetti posti frontalmente al veicolo, effettuando 19 scansioni orizzontali in combinazione a 6 scansioni orizzontali, il test ha esito positivo.
- **Osservazioni:** Il comportamento dei servomotori accoppiato a quello del LiDAR risulta coerente con quanto richiesto. Le specifiche SS07 e SS08 risultano verificate.

4.5 TS05: verifica della specifica di sistema SS09

- **Setup di verifica:** Veicolo alimentato posto in un ambiente di test con firmware finale.
- **Esecuzione del test:** Tramite misura della distanza percorsa in un determinato intervallo temporale, viene determinata la velocità media del dispositivo durante il regolare funzionamento.
- **Criterio di successo:** Se la velocità media del dispositivo risulta essere pari a 0.2 m/s, tenendo conto di eventuali fermate non superiori a 2", il test ha esito positivo.
- **Osservazioni:** Il comportamento del veicolo risulta coerente con quanto richiesto. La specifica SS09 risulta verificata.

4.6 TS06: verifica della specifica di sistema SS10

- **Setup di verifica:** Veicolo alimentato posto in un ambiente di test con firmware Interrupt (in appendice).
- **Esecuzione del test:** Il veicolo è posto su una superficie piana con ruote rialzate. L'esecuzione del firmware comanda un movimento continuo di entrambe le ruote a velocità costante.
- **Criterio di successo:** Se all'azionamento individuale o contemporaneo degli switch posti frontalmente al veicolo la rotazione delle ruote si arresta, il test ha esito positivo.
- **Osservazioni:** Il comportamento dei sensori di contatto frontali risulta coerente con quanto richiesto. La specifica SS10 risulta verificata.

4.7 TS07: verifica della specifica di sistema SS12

- **Setup di verifica:** Veicolo posto in un ambiente di test con firmware finale.
- **Esecuzione del test:** Il veicolo viene fatto funzionare senza interruzioni per 10'. Sono simulate più condizioni con batterie ogni volta alla massima carica:
  - il veicolo trova sempre almeno un percorso libero,
  - il veicolo urta un ostacolo e rimane in situazione di arresto,
  - il veicolo non riesce a trovare alcuna direzione libera e va in idle,

  Per ogni test viene misurata l'autonomia massima del dispositivo.
- **Criterio di successo:** Se il veicolo, per ognuna delle condizioni descritte, riesce ad avere autonomia maggiore di 10' il test ha esito positivo.
- **Osservazioni:** La specifica SS12 sull'autonomia risulta verificata.

Tabella 4.1. Matrice di tracciabilità tra requisiti dell'utente e specifiche di sistema

	TS01	TS02	TS03	TS04	TS05	TS06	TS07	BoM
SS01	✓							
SS02		✓						
SS03			✓					
SS04			✓					
SS05			✓					
SS06		✓						
SS07				✓				
SS08				✓				
SS09					✓			
SS10						✓		
SS11								
SS12							✓	
SS13								✓

Appendice

Codice per test dei componenti

Test Lidar

```
#include <SoftwareSerial.h>
#include "TFMini.h"

//Uno RX (TFMINI TX), Uno TX (TFMINI RX)
SoftwareSerial mySerial(12,13);
TFMini tfmini;

void setup() {
  Serial.begin(115200);
  while (!Serial);
  Serial.println ("Initializing...");
  mySerial.begin(TFMINI_BAUDRATE);
  tfmini.begin(&mySerial);
}

void loop() {
  uint16_t dist = tfmini.getDistance();
  Serial.print(dist);
  Serial.print("␣cm␣␣␣␣␣␣");
  //Wait some time before next measurement
  delay(25);
}
```

Test Servo

```
#include <Servo.h>

#define Min_Tilt   46     //Tilt -12.5 degree
#define Mid_Tilt   50   //Servo Mid Position
#define Max_Tilt   71     //Tilt +12.5 degree
#define Min_Pan    47.5   //Pan -22.5 degree
#define Mid_Pan    75     //Servo Mid Position
#define Max_Pan    92.5   //Pan +22.5 degree

Servo TILT;         //Create servo object to control a servo
```

```
11 Servo PAN;          //Create servo object to control a servo
12 float xpos = Min_Pan;      //Store the servo PAN position
13 float ypos = Min_Tilt;     //Store the servo TILT position
14 char  xscanDirection = 0;
15 char  yscanDirection = 0;
16
17 void setup() {
18   PAN.attach(10);     //attaches the Horizontal servo on pin 11
19   TILT.attach(11);    //attaches the Vertical servo on pin 10
20   PAN.write(Min_Pan);
21   TILT.write(Min_Tilt);
22 }
23
24 void Pan_Flow(){
25   if(xscanDirection == 0){
26     for (xpos = Min_Pan; xpos <= Max_Pan; xpos += 2.5) {
27       PAN.write(xpos);    //go to position in variable 'pos'
28       delay(30);
29       Tilt_Flow();
30       yscanDirection = ~yscanDirection;
31     }
32   }
33   else{
34     for (xpos = Max_Pan; xpos >= Min_Pan; xpos -= 2.5) {
35       PAN.write(xpos);    //go to position in variable 'pos'
36       delay(30);
37       Tilt_Flow();
38       yscanDirection = ~yscanDirection;
39     }
40   }
41   xscanDirection = ~xscanDirection;
42 }
43
44 void Tilt_Flow(){
45   if(yscanDirection == 0){
46     for (ypos = Min_Tilt; ypos <= Max_Tilt; ypos += 5) {
47       TILT.write(ypos);    //go to position in variable 'pos'
48       delay(30);
49     }
50   }
51   else{
52     for (ypos = Max_Tilt; ypos >= Min_Tilt; ypos -= 5) {
53       TILT.write(ypos);    //go to position in variable 'pos'
54       delay(30);
55     }
56   }
57 }
58
59 void loop() {
60   Pan_Flow();
61   delay(150);
62 }
```

Test Motori

```
#include <SparkFun_TB6612.h>

#define AIN1 2
#define BIN1 7
#define AIN2 4
#define BIN2 8
#define PWMA 5
#define PWMB 6
#define STBY 9

const int offsetA = 1;
const int offsetB = 1;
Motor motor1 = Motor(AIN1, AIN2, PWMA, offsetA, STBY);
Motor motor2 = Motor(BIN1, BIN2, PWMB, offsetB, STBY);

void setup() {
}


void loop() {
    motor1.drive(255,1000);
    motor1.drive(-255,1000);
    motor1.brake();
    delay(1000);

    motor2.drive(255,1000);
    motor2.drive(-255,1000);
    motor2.brake();
    delay(1000);

    forward(motor1, motor2, 150);
    delay(1000);

    back(motor1, motor2, -150);
    delay(1000);

    brake(motor1, motor2);
    delay(1000);

    left(motor1, motor2, 100);
    delay(1000);
    right(motor1, motor2, 100);
    delay(1000);

    brake(motor1, motor2);
    delay(1000);    l
}
```

Test Interrupt

```
#include <SparkFun_TB6612.h>  //Contains the class Motor

// Pins for all inputs, PWM defines must be on PWM pins
#define AIN1 5  //2 Verde
#define BIN1 7  //7 Arancione
#define AIN2 4  //4 Blu
#define BIN2 8  //8 Rosso
#define PWMA 3  //5 Viola
#define PWMB 11 //6 Marrone
#define STBY 6  //  Giallo

//motors configuration line up with function like forward: 1,-1
const int offsetA = 1, offsetB = 1;
Motor LEFT = Motor(BIN1, BIN2, PWMB, offsetB, STBY);
Motor RIGHT = Motor(AIN1, AIN2, PWMA, offsetA, STBY);

void STOP() {
    brake(LEFT, RIGHT);
}

void setup() {
  attachInterrupt(digitalPinToInterrupt(2), STOP, LOW);
}

void loop() {
  forward(LEFT, RIGHT, 50);
  delay(5000);
}
```

Firmware

Dorothy.ino

```
/* Dorothy il fregno che si muove da solo
 ---------------------------------------------------
 by Pierpaolo Granello e Alessandro Dei Giudici

  Laboratorio Multidisciplinare di Elettronica
  Magistrale Ingegneria Elettronica - Sapienza
 ---------------------------------------------------
*/

#include <SoftwareSerial.h>
#include "PWMServo.h"
#include "TFMini.h"
#include <SparkFun_TB6612.h>  //Contains the class Motor
#include <avr/power.h>

#define Min_Tilt   50     //Tilt -12.5 degree
#define Mid_Tilt   62.5   //Servo Mid Position
#define Max_Tilt   75     //Tilt +12.5 degree

#define Min_Pan    52.5   //Pan -22.5 degree
#define Mid_Pan    75     //Servo Mid Position
#define Max_Pan    97.5   //Pan +22.5 degree

//(Writes distance and Strenght for each location)
#define Iter_Tilt    6      //Points per Row
#define Iter_Pan     19     //Points per Column

// Pins for all inputs, PWM defines must be on PWM pins
#define AIN1 5  //2 Verde
#define BIN1 7  //7 Arancione
#define AIN2 4  //4 Blu
#define BIN2 8  //8 Rosso
#define PWMA 3  //5 Viola
#define PWMB 11 //6 Marrone
#define STBY 6  //  Giallo

//Servo Setup
PWMServo TILT;            //Create servo object to control a servo
PWMServo PAN;             //Create servo object to control a servo
//Serial port Setup
//Uno RX (TFMINI TX verde),Uno TX (TFMINI RX bianco)
SoftwareSerial mySerial(12 , 13);
TFMini tfmini;

//Motors configuration line up with function like forward: 1,-1
const int offsetA = 1;
const int offsetB = 1;
Motor LEFT = Motor(BIN1, BIN2, PWMB, offsetB , STBY);
Motor RIGHT = Motor(AIN1, AIN2, PWMA, offsetA , STBY);
```

```
50 //Allocating Memory for LIDAR results
51 int *LiDAR_Radius = malloc(Iter_Pan*sizeof(uint16_t*));
52
53 boolean dir = 0;
54 int search = 0;    // index for ambient scan
55 int xPan = 0;      // for storing radius distance
56 int yTilt = 0;     // for storing radius distance
57 float xpos = Min_Pan;    // Index for servo PAN position
58 float ypos = Min_Tilt;   // Index for servo TILT position
59 char  xscanDirection = 0;
60 char  yscanDirection = 0;
61 float pi = 3.14159265;
62 float deg2rad = pi / 180.0;
63
64 //Functions
65 void Pan_Flow();
66 void Tilt_Flow();
67 void lidar_acquisition();
68 //void printMatrix();
69 int ChooseDirection();
70 void ChangeDirection();
71 void STOP();
72
73
74 void setup() {
75   attachInterrupt(digitalPinToInterrupt(2), STOP, LOW);
76   PAN.attach(SERVO_PIN_A);         // Giallo
77   TILT.attach(SERVO_PIN_B);        // Viola
78   PAN.write(Min_Pan);
79   TILT.write(Min_Tilt);
80
81   Serial.begin(115200); //Initialize HW serial port (debug port)
82   while (!Serial);      //Wait for serial port.
83   Serial.println ("Initializing...");
84   mySerial.begin(TFMINI_BAUDRATE);  //data rate for the SWSerial
85   tfmini.begin(&mySerial);  //Initialize the TF Mini sensor
86 }
87
88 void loop() {
89   while (search < 8) {
90     for (int i=0; i<Iter_Pan; ++i) {
91       LiDAR_Radius[i] = 100;
92     }
93     Pan_Flow();         //Servo and Lidar actions
94
95     if(ChooseDirection()==1){    // Go Straight
96       search = 0;
97       forward(LEFT,RIGHT,100);
98       delay(1500);
99       brake(LEFT, RIGHT);
100    }
101    else {
102      ChangeDirection();
```

```
103       }
104     }
105     power_adc_disable();
106     power_spi_disable();
107     power_timer0_disable();
108     power_timer1_disable();
109     power_timer2_disable();
110     power_twi_disable();
111   }
```

Functions.ino

```
1   void Pan_Flow() {
2     if (xscanDirection == 0) {
3       for (xpos=Min_Pan,xPan=0; xpos<=Max_Pan; xpos+=2.5,++xPan) {
4         PAN.write(xpos);    //go to position in variable 'pos'
5         delay(5);
6         Tilt_Flow();
7         yscanDirection = ~yscanDirection;
8       }
9     }
10    else {
11      for (xpos=Max_Pan,xPan=Iter_Pan-1;xpos>=Min_Pan;xpos-=2.5,--
          xPan) {
12        PAN.write(xpos);    //go to position in variable 'pos'
13        delay(5);
14        Tilt_Flow();
15        yscanDirection = ~yscanDirection;
16      }
17    }
18    xscanDirection = ~xscanDirection;
19  }
20
21
22  void Tilt_Flow() {
23    if (yscanDirection == 0) {
24      for (ypos = Min_Tilt; ypos <= Max_Tilt; ypos += 5) {
25        TILT.write(ypos);    //go to position in variable 'pos'
26        delay(5);
27        lidar_acquisition();
28      }
29    }
30    else {
31      for (ypos = Max_Tilt; ypos >= Min_Tilt; ypos -= 5) {
32        TILT.write(ypos);    //go to position in variable 'pos'
33        delay(5);
34        lidar_acquisition();
35      }
36    }
37  }
```

```
38
39
40 void lidar_acquisition() {
41   uint16_t radius = tfmini.getDistance();
42   if (radius < LiDAR_Radius[xPan]) LiDAR_Radius[xPan] = radius;
43
44   float azimuth = (-90 + Max_Pan + xpos) * deg2rad;
45   float elevation = (45 + ypos) * deg2rad;
46   double x = radius * sin(elevation) * cos(azimuth);
47   double y = radius * sin(elevation) * sin(azimuth);
48   double z = radius * cos(elevation);
49   Serial.println(String(x,5)+" "+String(y,5)+" "+String(-z,5));
50
51   delay(10);  //Lidar Acquisition Delay. No Movement can be done
52 }
53
54
55 int ChooseDirection() {
56   for (int i=0; i<Iter_Pan; ++i) {
57     if(LiDAR_Radius[i] < 100) {
58        ++search;
59        return 0;
60     }
61     else return 1;
62   }
63 }
64
65
66 void ChangeDirection() {
67       LEFT.drive(120);
68       RIGHT.drive(-120);
69       delay(280);
70       brake(LEFT, RIGHT);
71 }
72
73
74 void STOP() {
75   brake(LEFT,RIGHT);
76 }
```