



UNIVERSITÀ DI PISA



MASTER DEGREE in EMBEDDED COMPUTING SYSTEMS
A.Y. 2016 - 2017

Relazione Finale Design of Embedded Systems

Full Professor
Marco Di Natale

Students
Silvio Bacci
Andrea Baldecchi

Indice

1 User Requirements	4
1.1 Descrizione generale	4
1.2 Data Dictionary	4
1.3 Descrizione componenti	5
2 Functional Specifications	8
2.1 Segnali di I/O e parametri del sistema	8
2.2 Modalità funzionali della macchina	9
3 Implementazione Hardware	12
3.1 Hardware utilizzato	12
3.2 Cablaggio	17
3.2.1 Cablaggio Motori	17
3.2.2 Cablaggio Infrarossi	18
3.2.3 Cablaggio Ultrasuoni	21
4 Modello SysML	22
4.1 Profile Diagram	22
4.2 Type Block Definition Diagram	22
4.3 System Block Definition Diagram	23
4.4 System Internal Block Diagram	23
5 Obstacle Avoidance	25
5.1 Introduzione	25
5.2 Descrizione controllore in anello aperto	25
6 Implementazione Software	27
6.1 Introduzione	27
6.2 Classi/Librerie	27
6.3 Setup	28
6.4 Main Loop	28
6.5 Motori e PWM	28
6.6 Sensore ad ultrasuoni e lettura dei dati	30
7 Cunit Testing	31
7.1 Introduzione	31
7.2 Test: Turning Direction	31
7.3 Test: Set Avoid Param	33
7.4 Test: Search Line	36
7.5 Conclusioni	37
7.6 Gcov Testing	38

1 User Requirements

1.1 Descrizione generale

Il sistema implementato è composto da una macchina, da una mappa predefinita nella quale questa può muoversi e da un insieme di ostacoli che la macchina deve evitare.

1.2 Data Dictionary

Nella seguente tabella vengono riportate le grandezze fisiche utili alla descrizione delle componenti:

Nome	Descrizione	Unità	Default	Min	Max
L	Semi larghezza del veicolo	cm	6	5	7
d	Distanza ruota-anteriore	cm	6	5	7
s	Spessore della linea	cm	1,5	1	2
l	Semi larghezza ostacolo	cm	6	1	6
h	Altezza ostacolo	cm	15	15	-
D	Distanza tra curva e ostacolo	cm	25	20	-
m	Margine esterno al circuito	cm	35	35	-
c	Raggio di curvatura linea	cm	-	20	-
w	Distanza fra i tratti della linea	cm	-	10	-
e	Distanza fra centro-macchina e centro-linea	cm	-	0	5

1.3 Descrizione componenti

1. Mappa predefinita

- 1.1. La superficie deve essere piana
- 1.2. La superficie deve essere bianca
- 1.3. Dentro la mappa deve essere presente una linea
 - 1.3.1. La linea deve essere nera
 - 1.3.2. La linea deve avere larghezza pari a s
 - 1.3.3. La linea deve creare un circuito chiuso
 - 1.3.4. La linea non deve attraversare se stessa
 - 1.3.5. La distanza fra tratti diversi della linea deve essere w
 - 1.3.6. Il raggio di curvatura che la linea può formare deve essere pari a d
- 1.4. Lo spazio libero sul lato esterno della linea nera deve essere pari a m

2. Ostacoli

- 2.1. Devono essere dei parallelepipedi
 - 2.1.1. Altezza: h
 - 2.1.2. Larghezza: l
 - 2.1.3. Profondità: l
- 2.2. Devono essere posizionati al centro della linea
- 2.3. Devono essere posizionati su rettilinei a distanza D dalla curva

3. Macchina

- 3.1. La macchina deve avere le seguenti caratteristiche:
 - 3.1.1. Altezza inferiore al valore min di h
 - 3.1.2. Larghezza: $2 \cdot L$
 - 3.1.3. Distanza ruota-anteriore: d
- 3.2. La macchina deve rilevare la linea nera quando sotto di essa.
- 3.3. La macchina deve seguire la linea nera.
 - 3.3.1. Ogni volta che la linea crea una curva, la macchina deve rilevare la curva.
 - 3.3.2. Una volta rilevata la curva, la macchina deve modificare la sua direzione in accordo con il senso della curva in modo da riportarsi centrata sulla linea (ad esempio, una curva a destra deve far girare la macchina a destra).
 - 3.3.3. La macchina deve mantenersi centrata al di sopra della linea nera.
 - 3.3.3.1. *Tratto di circuito rettilineo*
 - 3.3.3.1.1. Sia e la distanza tra centro della macchina e il centro della linea nera, questo errore non deve mai superare 5 cm.
 - 3.3.3.2. Tratto di circuito in curva
 - 3.3.3.2.1. L'errore e preferibilmente non deve superare 5 cm.
 - 3.3.3.2.2. Nel caso in cui l'errore e superi il valore di soglia (5 cm), la macchina deve continuare a curvare riportando l'errore sotto il valore di soglia.
- 3.4. La macchina deve rilevare un ostacolo lungo il circuito.
- 3.5. La macchina deve evitare l'ostacolo rilevato uscendo dalla linea e ritornando su di essa una volta evitato.
 - 3.5.1. Ogni volta che la macchina rileva un ostacolo deve modificare la sua direzione portandosi preferibilmente verso il centro del circuito.
 - 3.5.2. La macchina deve oltrepassare l'ostacolo senza urtarlo.
 - 3.5.3. La macchina deve tornare verso la linea finché non la rileva.

- 3.6. In fase iniziale deve essere posizionata all'interno del circuito
- 3.7. In fase iniziale deve essere capace di trovare la linea autonomamente
 - 3.7.1. La macchina deve muoversi a dritto fino a rilevare la linea.

2 Functional Specifications

2.1 Segnali di I/O e parametri del sistema

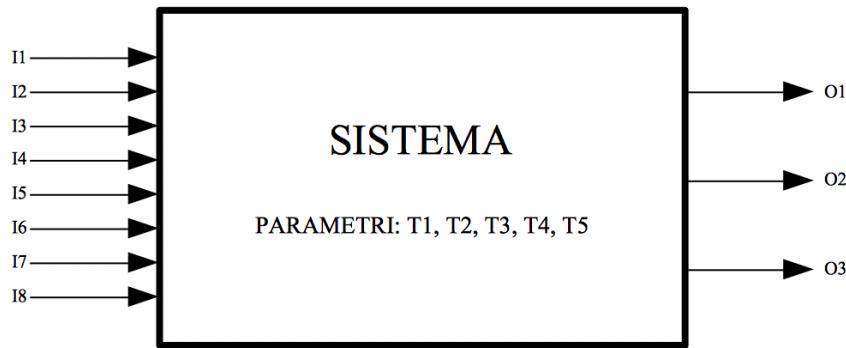
Facendo riferimento all'implementazione hardware presentata nel Capitolo 3, i segnali di I/O del sistema da noi considerato sono:

ID	Nome	Descrizione	Triggered	Direzione	Tipo	Min	Max	Risoluzione	PIN
I1	C_S1_INFRA	Colore rilevato dal sensore ad infrarossi più a sinistra	SI	INPUT	BOOLEAN	0	1	1	A0
I2	C_S2_INFRA	Colore rilevato dal secondo sensore ad infrarossi	SI	INPUT	BOOLEAN	0	1	1	A1
I3	C_S3_INFRA	Colore rilevato dal sensore ad infrarossi a centro-sinistra	SI	INPUT	BOOLEAN	0	1	1	A2
I4	C_S4_INFRA	Colore rilevato dal sensore ad infrarossi a centro-destra	SI	INPUT	BOOLEAN	0	1	1	A3
I5	C_S5_INFRA	Colore rilevato dal quinto sensore ad infrarossi	SI	INPUT	BOOLEAN	0	1	1	A4
I6	C_S6_INFRA	Colore rilevato dal sensore ad infrarossi più a destra	SI	INPUT	BOOLEAN	0	1	1	A5
I7	D_ULTRA	Distanza dall'ostacolo più vicino rilevata dal sensore ad ultrasuoni	NO	INPUT	INTEGER	0	50	1	D7
I8	B_RST	Bottone di reset della scheda Arduino M0 Pro	SI	INPUT	BOOLEAN	0	1	1	-
O1	PWM_RIGHT_SERVO	Larghezza dell'impulso della PWM applicata al motore destro	NO	OUTPUT	INTEGER	0	200	1	D10
O2	PWM_LEFT_SERVO	Larghezza dell'impulso della PWM applicata al motore sinistro	NO	OUTPUT	INTEGER	0	200	1	D11
O3	LED	Segnale di controllo del LED	SI	OUTPUT	BOOLEAN	0	1	1	D13

I parametri del sistema da noi considerato sono i seguenti:

ID	Nome	Descrizione	Unità	Default	Min	Max	Risoluzione
T1	CAR_SPEED	Velocità della macchina	cm/s	10	0	17	Step variabile
T2	NUM_INFRA_SENSORS	Numero di sensori ad infrarossi	-	6	0	8	1
T3	SAFETY_ANGLER	Angolo di sicurezza utilizzato per evitare l'ostacolo come specificato nel paragrafo 3.3	Gradi (°)	20	20	20	0
T4	ULTRA_RANGE	Massima distanza rilevabile dagli ultrasuoni	cm	50	0	5000	1
T5	PWM_WIDTH	Durata impulso della PWM	μs	50	0	200	Step variabile

Il sistema considerato è quindi il seguente:



2.2 Modalità funzionali della macchina

Ci sono 4 differenti modalità funzionali:

1. Setup

1.1. *Entering in mode*

1.1.1. La macchina viene accesa collegando l'alimentazione.

1.2. *While in mode*

1.2.1. I segnali O1 e O2 vengono impostati allo stesso valore pari a T5.

1.2.1.1. La macchina inizia a muoversi a dritto con una velocità pari al valore default del parametro T1.

1.3. *Exiting from mode*

1.3.1. La macchina entra in modalità “Line Searching”

1.3.2. Il segnale O3 viene settato a 1

1.3.2.1. Il LED si accende

2. Line Searching

2.1. Entering in mode

2.1.1. I segnali O1 e O2 non vengono modificati

2.1.1.1. La macchina continua a muoversi nella direzione corrente con la stessa velocità

2.2. While in mode

2.2.1. La macchina acquisisce periodicamente i valori dei segnali I1 I7

2.2.2. La macchina interpreta i valori rilevati in modo tale da conoscere il colore della superficie sotto ciascuno di essi.

2.3. Exiting from mode

2.3.1. Se almeno uno dei segnali I1 I6 ha rilevato il colore nero

2.3.1.1. La macchina ha trovato la linea nera ed entra in modalità “Line Following”

2.3.2. Se il segnale I7 diventa minore o uguale al parametro T4

2.3.2.1. La macchina ha rilevato un ostacolo ed entra in modalità “Line Following”

3. Line Following

3.1. Entering in mode

3.1.1. Il segnale O3 viene resettato a 0

3.1.1.1. Il LED si spegne

3.2. While in mode

3.2.1. La macchina acquisisce periodicamente i valori dei segnali I1 I7

3.2.2. I segnali O1 e O2 vengono modificati in funzione di I1 I6 come riportato nella seguente tabella della verità:

I1	I2	I3	I4	I5	I6	Direzione	O1	O2
0	0	0	0	1	-	Destra	0	T5
0	0	0	0	-	1			
1	-	0	0	0	0	Sinistra	T5	0
-	1	0	0	0	0			
0	0	1	-	0	0	Centro	T5	T5
0	0	-	1	0	0			
1	-	0	0	1	-	Linea piena	0	T5
1	-	0	0	-	1			
-	1	0	0	1	-			
-	1	0	0	-	1			
0	0	0	0	0	0	No linea	T5	T5

3.2.2.1. La macchina segue la linea sterzando in modo tale da rimanere sempre centrata su di essa.

3.3. *Exiting from the mode*

3.3.1. Il segnale I7 diventa minore o uguale al parametro T4

3.3.2. La macchina entra in modalità “Obstacle Avoiding”

4. Obstacle Avoiding

4.1. *Entering in mode*

4.1.1. Il segnale O3 viene settato a 1

4.1.1.1. Il LED si accende

4.2. *While in mode*

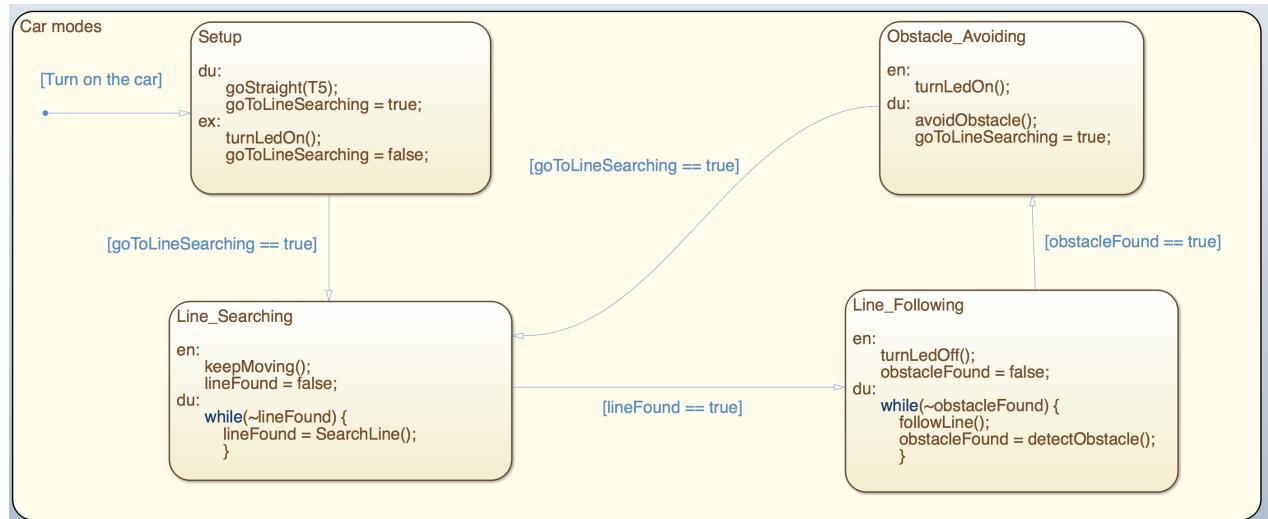
4.2.1. La macchina acquisisce periodicamente i valori dei segnali I1 I7

4.2.2. In funzione del valore corrente del segnale I7 viene pianificata la traiettoria (come specificato nel paragrafo 5.2) per evitare l'ostacolo.

4.3. *Exiting from the mode*

4.3.1. La macchina entra in modalità “Line Searching”

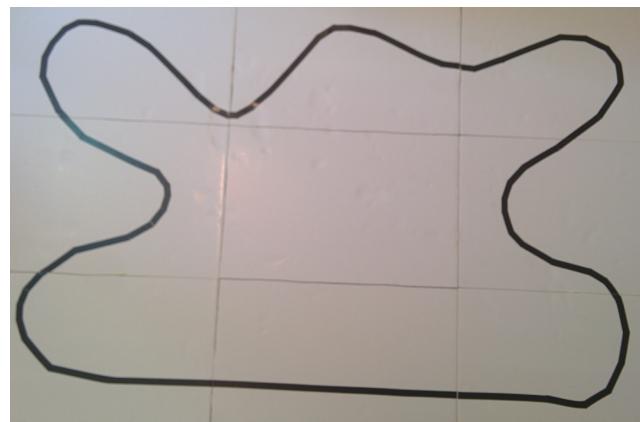
Di seguito riportiamo inoltre un diagramma a stati finiti che riassume quanto indicato sopra:



3 Implementazione Hardware

3.1 Hardware utilizzato

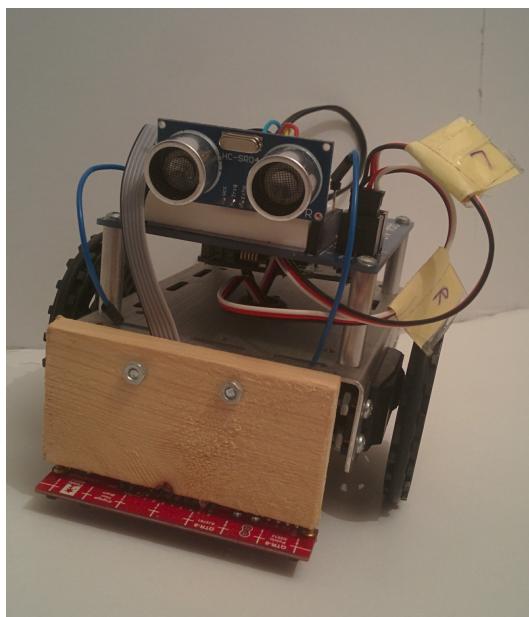
1. La mappa predefinita è stata realizzata con un foglio di cartone bianco di dimensioni 1m x 1.5m
 - 1.1. La linea nera è stata realizzata usando del nastro isolante di larghezza 1.5 cm



2. Gli ostacoli, con dimensione 6 cm x 6 cm x 15 cm, sono stati realizzati con fogli di cartone bianco



3. La macchina scelta è la “*Parallax Robot Shield*” con scheda “*Arduino M0 Pro*”.

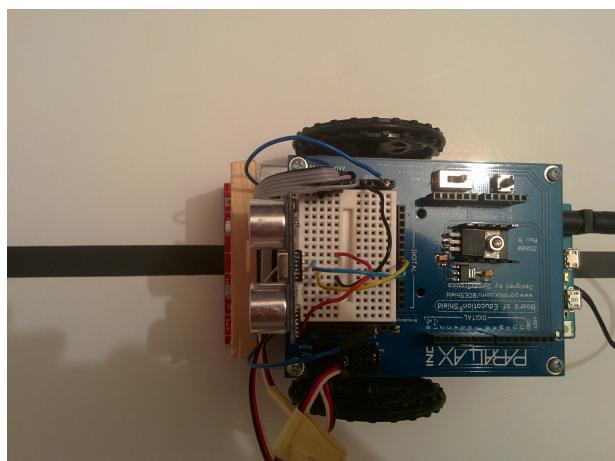


La macchina è stata inoltre equipaggiata di:

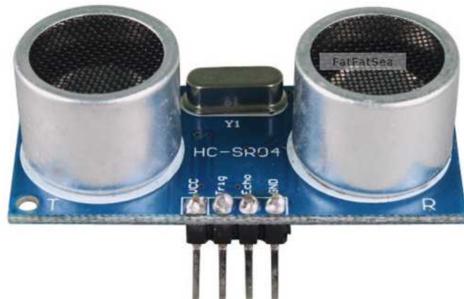
- 3.1. 6 sensori analogici di riflessione ad infrarossi “*QTR-8A Pololu*” per la rilevazione della linea nera



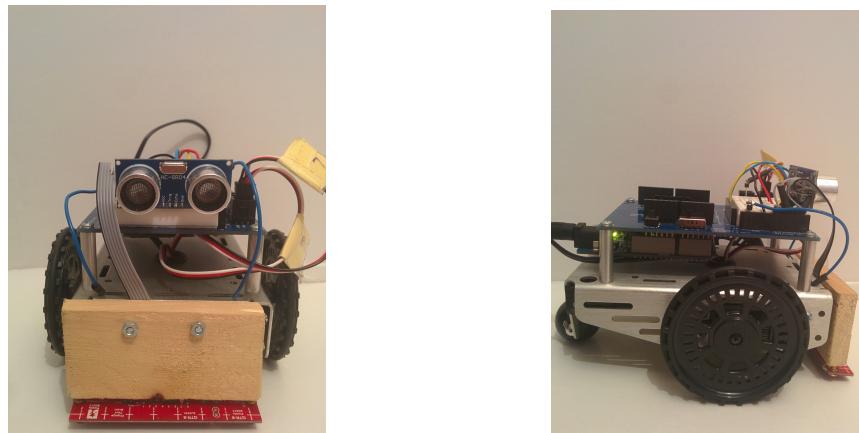
- 3.1.1. Posizionati parallelamente al semiasse della macchina



- 3.2. Un sensore digitale ad ultrasuoni “*HC-SR04 Itead Studio*” per la rilevazione degli ostacoli



- 3.2.1. Posizionato nella parte anteriore della macchina
3.2.2. Ha un angolo di misurazione di 15°



- 3.3. Un LED arancione (contrassegnato in figura con la lettera “L”) che indica la modalità funzionale della macchina



Da notare che le dimensioni utilizzate sono quelle indicate come valore predefinito nella tabella riportata nel paragrafo 1.2.

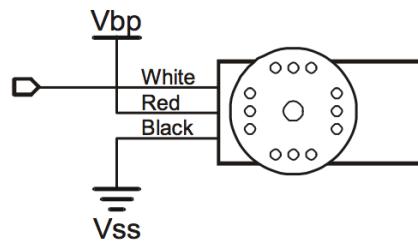
3.2 Cablaggio

3.2.1 Cablaggio Motori

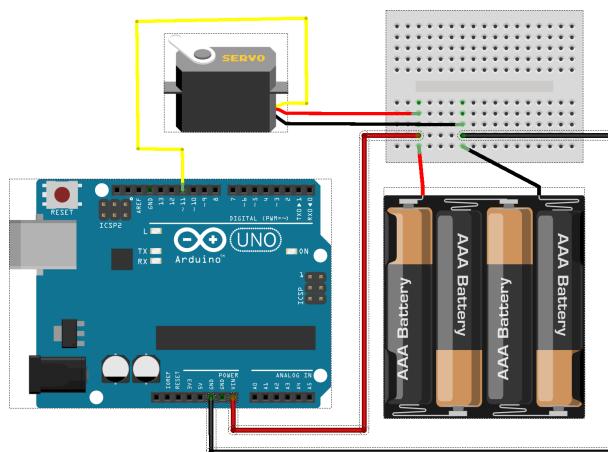
I Servo da noi utilizzati sono i “*Parallax Continuous Rotation Servo*”. Questi motori sono dotati di 3 differenti PIN da utilizzare come riportato nella seguente tabella:

Pin	Nome	Descrizione	Min	Tipico	Max	Unità
Bianco	Signal	Input: TTL o CMOS	3,3	5	Vservo + 0,2	V
Rosso	Vservo	Alimentazione	4	5	6	V
Nero	Vss	Ground	-	0	-	V

I PIN devono essere collegati come segue:



Il PIN bianco sarà collegato ad uno dei PIN digitali della scheda Arduino M0 Pro in modo da fornire una PWM al motore. Il collegamento fra la scheda e il motore è stato realizzato come segue:

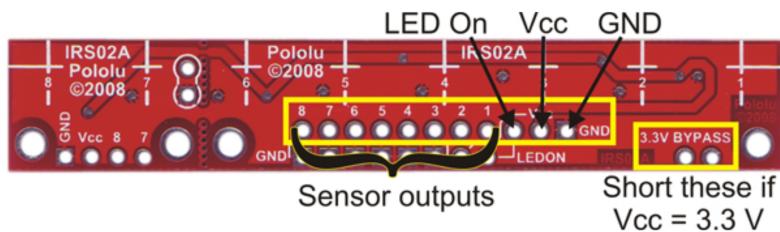


3.2.2 Cablaggio Infrarossi

I sensori infrarossi “*QTR-8A Pololu*” sono caratterizzati dalla presenza di:

Pin	Descrizione	Min	Tipico	Max	Unità
8 sensor output	Analog voltages	0	-	VCC	V
VCC	Alimentazione	4	5	6	V
GND	Ground	-	0	-	V

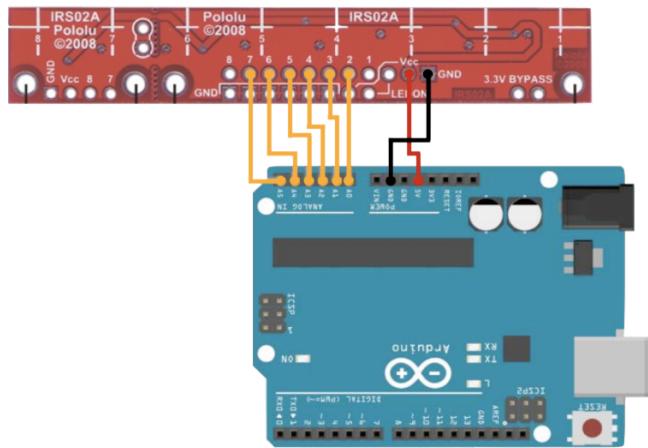
Riportando di seguito quanto indicato sul Datasheet del produttore, i collegamenti da effettuare sono i seguenti:



Rispettando quanto indicato, abbiamo corto-circuitato il “*3.3V BYPASS*” come segue:



Non abbiamo collegato nessun LED ed abbiamo utilizzato soltanto 6 delle 8 uscite disponibili. Pertanto il cablaggio effettuato è stato il seguente:

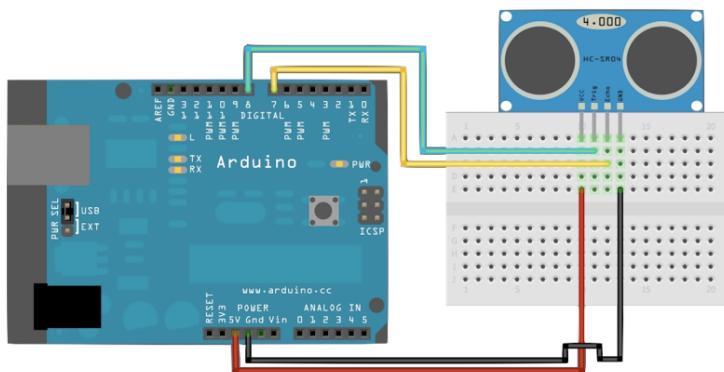


3.2.3 Cablaggio Ultrasuoni

Il sensore ad ultrasuoni “*HC-SR04 Itead Studio*” è caratterizzato dalla presenza di:

Pin	Descrizione	Mi n	Tipico	Max	Unità
Trigger Input Signal	10uS TTL pulse	0	-	VCC	V
Echo Output Signal	Input TTL level signal and the range in proportion	0	-	VCC	V
VCC	Alimentazione	4	5	6	V
GND	Ground	-	0	-	V

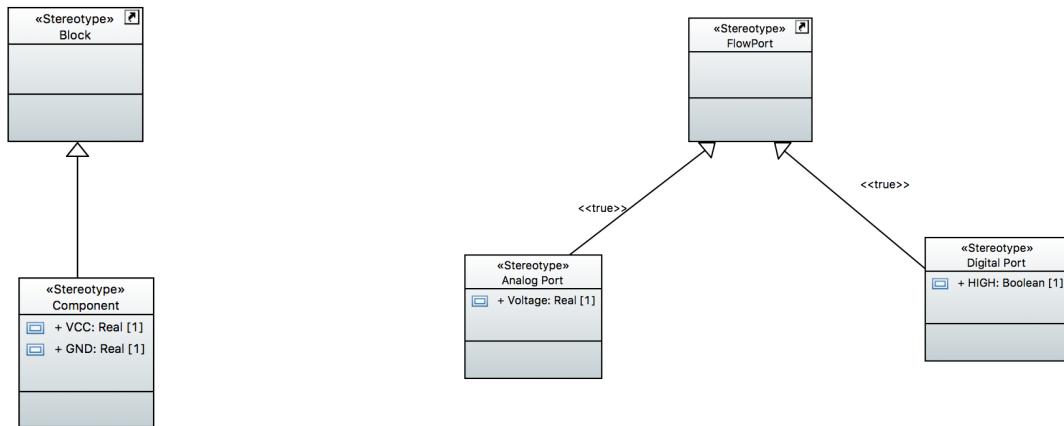
I PIN sono stati collegati come segue:



4 Modello SysML

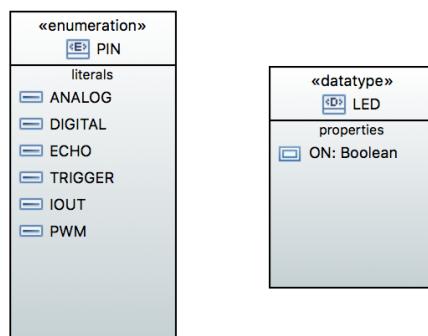
4.1 Profile Diagram

Di seguito viene riportato il “*Profile Diagram*” applicato al diagramma SysML:



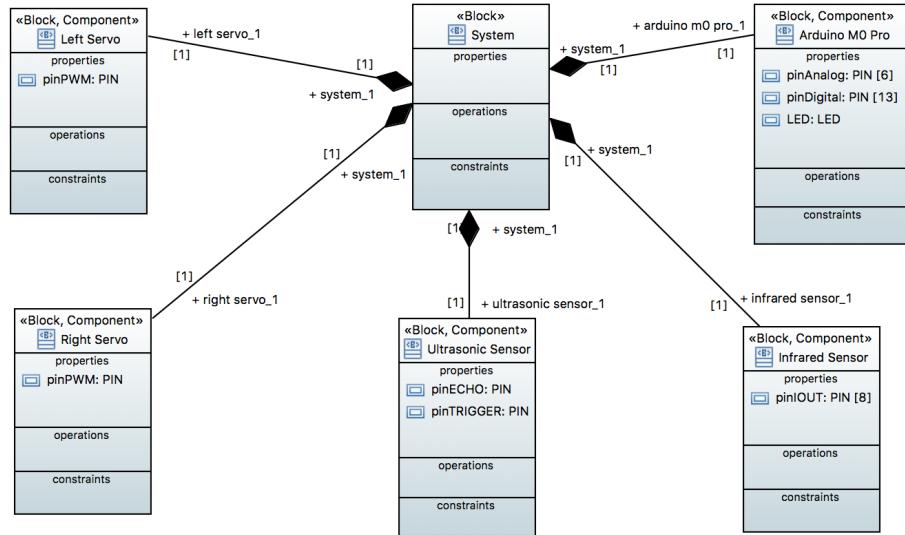
4.2 Type Block Definition Diagram

Di seguito viene riportato il “*Type Block Definition Diagram*” in cui sono definiti i tipi utilizzati nel modello:



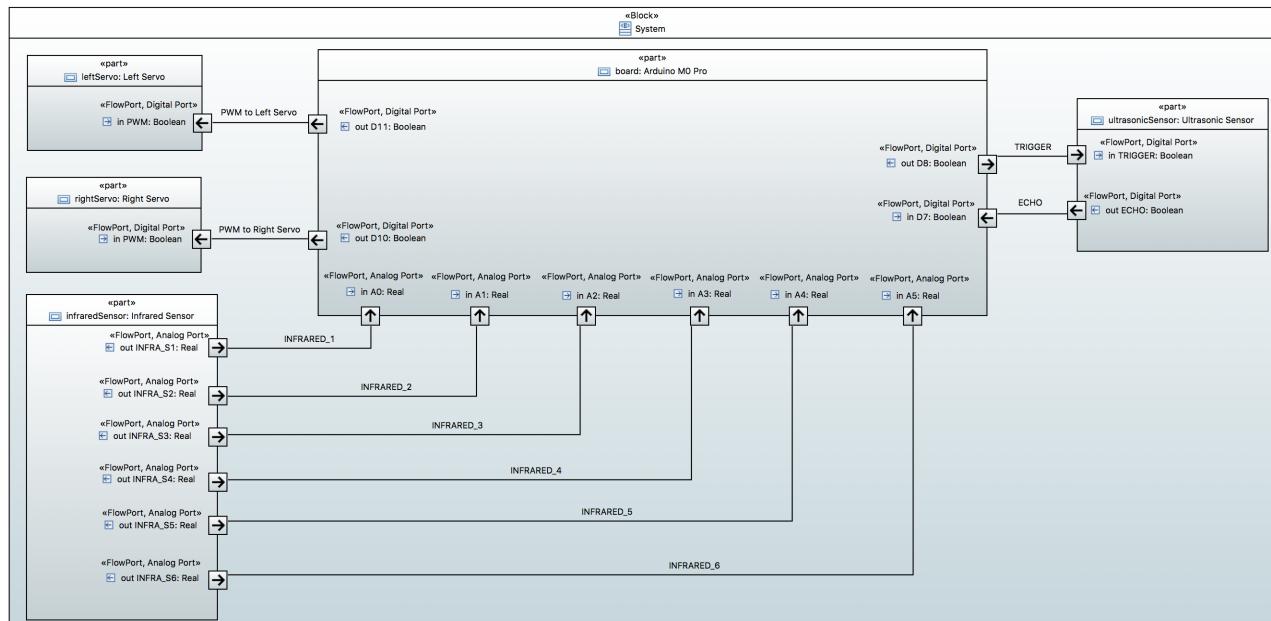
4.3 System Block Definition Diagram

Di seguito viene riportato il “System Block Definition Diagram” che descrive le componenti del sistema:



4.4 System Internal Block Diagram

Di seguito viene riportato il “System Internal Block Diagram” che descrive le relazioni tra le componenti del sistema:



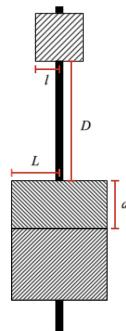
5 Obstacle Avoidance

5.1 Introduzione

La macchina quando rileva la presenza di un ostacolo (ovvero quando il segnale I7 diventa minore o uguale al parametro T4), inizia una sequenza di operazioni atte ad evitare l'ostacolo stesso. L'idea è quella di creare un controllore in anello aperto sapendo le condizioni iniziali del sistema.

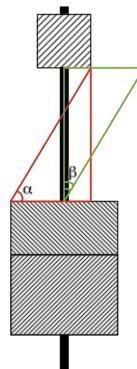
5.2 Descrizione controllore in anello aperto

Il sistema da noi considerato in questo caso è costituito dalla macchina e dall'ostacolo posizionati come segue:



La traiettoria da eseguire per evitare l'ostacolo è divisa in 3 fasi:

1. La macchina modifica la sua direzione verso l'interno del circuito con un angolo β riportato nella seguente figura:



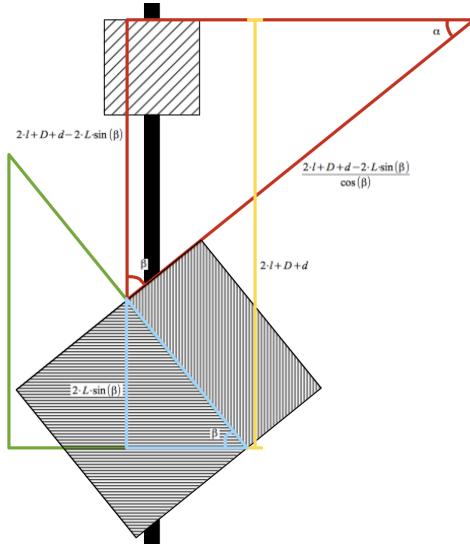
Facendo riferimento al triangolo in rosso, si calcola l'angolo α come:

$$\alpha = \arctg\left(\frac{D}{L+l}\right)$$

L'angolo β è calcolabile come il complementare di α (ovvero $\beta=90^\circ-\alpha$). Per essere sicuri di evitare l'ostacolo aggiungiamo un margine all'angolo β pari al parametro T3 (ovvero $\beta=\beta+T3$).

2. La macchina inizia a muoversi a dritto per una distanza tale da permettere alla macchina di portare le ruote in linea con lo spigolo posteriore dell'ostacolo. Facendo riferimento al segmento in giallo nella seguente figura, la distanza tra ruota e spigolo posteriore dell'ostacolo è $2 \cdot l + D + d$. Considerando il triangolo celeste, possiamo calcolare il suo cateto verticale come $2 \cdot L \cdot \sin(\beta)$. In questo modo il cateto verticale del triangolo in rosso diventerà $2 \cdot l + D + d - 2 \cdot L \cdot \sin(\beta)$. La distanza da percorrere sarà uguale all'ipotenusa del triangolo colorato in rosso, che possiamo calcolare dividendo la precedente espressione per $\cos(\beta)$, ottenendo così

$$\frac{2 \cdot l + D + d - 2 \cdot L \cdot \sin(\beta)}{\cos(\beta)}$$



3. Per tornare sulla linea, viene eseguita una rotazione nel verso opposto al precedente, pari a $2 \cdot \beta$.
4. La macchina inizia a muoversi a dritto, in modalità “Line Searching”.

6 Implementazione Software

6.1 Introduzione

Per l'implementazione software, avendo utilizzato una scheda *Arduino M0 Pro*, abbiamo scelto di implementare la componente software tramite l'ambiente di sviluppo integrato rilasciato da Arduino per la programmazione del microcontrollore (*Arduino IDE*).

6.2 Classi/Librerie

Per la gestione della varie periferiche sono state create 4 classi C++ utilizzate in seguito come librerie:

- Car
 - Libreria che permette di gestire il movimento della macchina (velocità, direzione, ecc..)
- UltrasonicSensor
 - Libreria che permette di leggere ed elaborare i dati provenienti dal sensore ad ultrasuoni
- Infrared
 - Libreria che permette di leggere ed elaborare i dati provenienti dai sensori ad infrarossi
- Timer
 - Libreria che permette di impostare timeouts per l'esecuzione di specifiche operazioni

6.3 Setup

Nella funzione di inizializzazione “*setup()*” vengono create le istanze delle classi/librerie e vengono inizializzate tutte le strutture, le variabili ed i parametri necessari per la gestione delle varie componenti.

6.4 Main Loop

Il codice principale è caratterizzato da un loop infinito che ciclicamente, senza alcun meccanismo di sincronizzazione, ripete le stesse operazioni. Questo significa che non appena il codice di una iterazione viene completato, viene eseguita subito una nuova iterazione. Ad ogni iterazione vengono controllati i dati rilevati dai sensori e, in base ai dati stessi, vengono prese le relative decisioni.

6.5 Motori e PWM

Per l'utilizzo dei motori è stata utilizzata l'apposita libreria “*Servo*” già installata nell'ambiente di sviluppo. Per permettere alle ruote di muoversi, ogni motore deve ricevere in ingresso una PWM (pulse width modulation). Ricordiamo che il “*Duty Cycle*” viene calcolato come segue

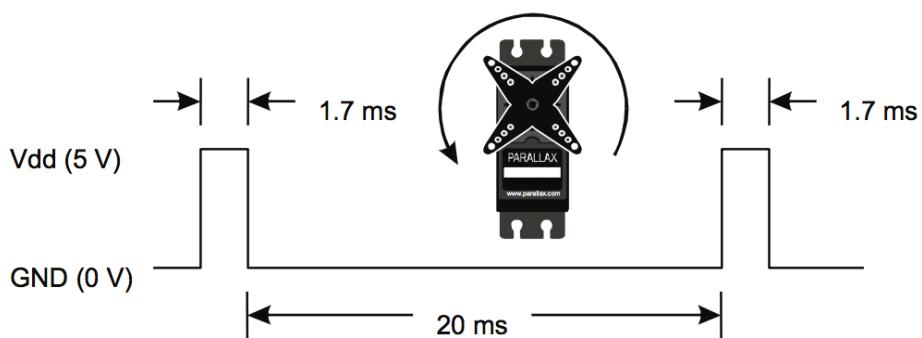
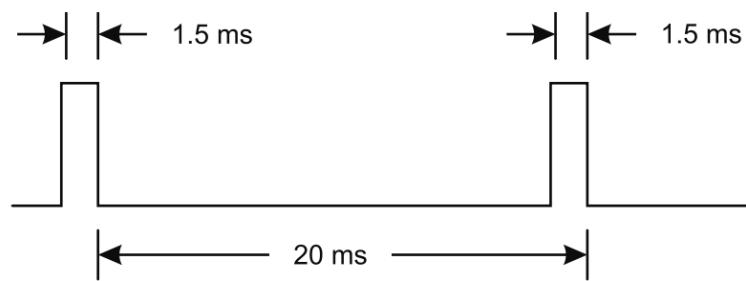
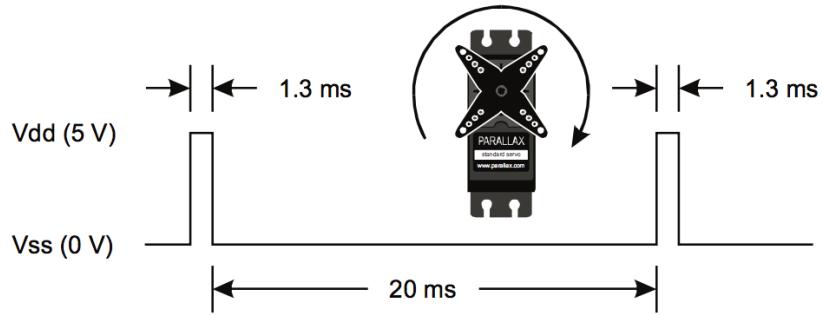
$$\text{Duty Cycle} = \frac{\text{Periodo a livello alto}}{\text{Periodo Totale}}$$

Le librerie da noi utilizzate, prevedono in ingresso l'ampiezza dell'intervallo in cui il segnale rimane a livello alto. Infatti la PWM che verrà fornita, è caratterizzata da un intervallo a livello basso fisso, pari a 20 ms. Cambiando quindi l'ampiezza dell'intervallo a livello alto otteniamo i seguenti casi:

Periodo a livello alto	Periodo a livello basso	Periodo Totale	Duty Cycle	Effetto motore destro	Effetto motore sinistro
1,3 ms	20 ms	21,3 ms	6 %	Massima velocità senso orario	Massima velocità senso antiorario
1,5 ms	20 ms	21,5 ms	7 %	Ruota ferma	Ruota ferma
1,7 ms	20 ms	21,7 ms	8 %	Massima velocità senso antiorario	Massima velocità senso orario

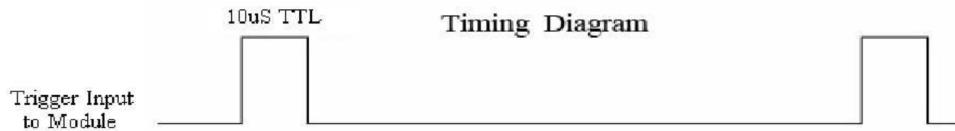
Le ruote hanno un comportamento opposto in quanto sono state installate in maniera speculare. Inoltre per ottenere velocità di rotazioni intermedie basterà variare l'ampiezza dell'intervallo tra [1.3 – 1.7].

La rappresentazione grafica della PWM inviata ai servo nei vari casi è la seguente:

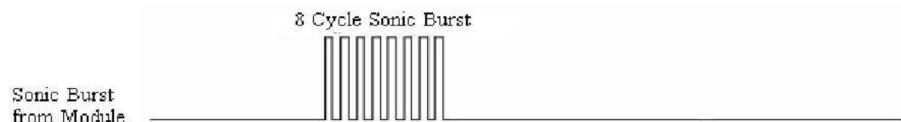


6.6 Sensore ad ultrasuoni e lettura dei dati

Per poter leggere i dati dal sensore ad ultrasuoni, il primo passo è quello di inviare attraverso il pin TRIGGER un segnale alto per $10\mu s$.



In questo modo il sensore ad ultrasuoni si attiva, inviando “impulsi ad ultrasuoni” ad una frequenza pari a 40 KHz.



Infine, grazie al pin ECHO è possibile intuire la distanza dall'ostacolo. Infatti l'intervallo di tempo in cui il segnale rimane alto è proporzionale alla distanza rilevata:



Per poter convertire l'intervallo di tempo nella distanza rilevata è necessario moltiplicare metà del numero di μs individuati (poiché il tempo indicato è quello necessario affinché il suono percorra la distanza due volte, andata e ritorno) per la velocità del suono, ovvero 340 m/s. Quindi otteniamo:

$$Distanza = Periodo(s) \cdot \frac{340}{2} = Periodo(s) \cdot 170 = Periodo(\mu s) \cdot 0,017 = \frac{Periodo(s)}{58}$$

7 Cunit Testing

7.1 Introduzione

La fase di testing è stata svolta attraverso l'ausilio delle librerie *Cunit*. Le funzioni da noi scelte per essere testate sono:

1. *dir turningDirection(int m)*: data la modalità *m* e una sestupla binaria rappresentante i colori rilevati dai sensori la funzione restituisce la direzione nella quale curvare sotto forma di un enumerato *dir*.
2. *void setAvoidParam()*: data la velocità angolare e lineare della macchina e la distanza tra ostacolo e macchina la funzione calcola i parametri relativi alla traiettoria da seguire per evitare un ostacolo.
3. *void searchLine()*: date le variabili *newDirection* e *directionToAvoidObstacle* la funzione aggiorna le variabili *mode* e *directionToAvoidObstacle*.

Sono state utilizzate 3 suites differenti, una per ciascuna funzione da testare, il cui codice può essere trovato all'interno del file “*Cunit_Test.c*”. Abbiamo inoltre scelto di utilizzare l'interfaccia *Basic*.

7.2 Test: Turning Direction

La prima funzione che abbiamo scelto di testare è la “*dir turningDirection(int m)*” appartenente alla classe Infrared che si trova nel file “*infrared.cpp*”. Gli input di questa funzione sono due:

- Un intero *m* passato come parametro, che rappresenta la modalità corrente dell'intero sistema. Questo può assumere 3 valori distinti quali *LINE_FOLLOWING*, *OBSTACLE_AVOIDING* e *LINE_SEARCHING*.
- Una sestupla binaria rappresentante i colori rilevati da ciascuno dei sei sensori infrarossi utilizzati.

L'output della funzione è:

- Un intero che rappresenta la direzione nella quale curvare. Questo parametro può assumere 5 valori distinti quali *LEFT*, *CENTER*, *RIGHT*, *NO_LINE* e *FULL_LINE*.

Come si evince dal codice della funzione, riportato di seguito, i valori dell'intero m che influiscono sulla logica interna della funzione diventano sostanzialmente due. Possiamo quindi trattare anche questo parametro come binario:

```
dir turningDirection(int m) {
    if((sensorColour[0] == BLACK || sensorColour[1] == BLACK) ...
        && (sensorColour[4] == BLACK || sensorColour[5] == BLACK))
        return FULL_LINE;
    if(sensorColour[4] == BLACK || sensorColour[5] == BLACK)
        return RIGHT;
    if(sensorColour[0] == BLACK || sensorColour[1] == BLACK)
        return LEFT;

    switch(m) {
        case LINE_FOLLOWING:
            if(sensorColour[2] == BLACK || sensorColour[3] == BLACK)
                return CENTER;
        default:
            if(sensorColour[2] == BLACK && sensorColour[3] == BLACK)
                return CENTER;
    }

    return NO_LINE;
}
```

Abbiamo quindi sette valori binari in input, per un totale di $2^7=128$ combinazioni possibili. Di seguito riportiamo, una porzione della tabella di verità di questa funzione:

m	S0	S1	S2	S3	S4	S5	dir
1	0	0	1	1	0	0	CENTER
1	0	0	1	1	0	1	RIGHT
1	0	0	1	1	1	0	RIGHT
1	0	0	1	1	1	1	RIGHT
1	0	1	0	0	0	0	LEFT
1	0	1	0	0	0	1	FULL_LINE
1	0	1	0	0	1	0	FULL_LINE

Per il testing di questa funzione abbiamo creato due file di testo contenenti le varie combinazioni in input e i relativi valori output che la funzione dovrebbe restituire. Una volta acquisiti questi parametri andiamo ad eseguire la funzione da testare per ogni caso definito, confrontando il valore di ritorno della funzione con il relativo valore letto dal file di testo. L’istruzione della libreria *Cunit* utilizzata in questo caso è quindi solo:

```
CU_ASSERT(results[i] == turningDirection(m[i]))
```

Attraverso questa suite sono stati eseguiti quindi 128 test che hanno avuto tutti esito positivo.

7.3 Test: Set Avoid Param

La funzione “*void setAvoidParam()*” calcola i parametri relativi alla traiettoria da seguire per evitare un ostacolo una volta che quest’ultimo viene rilevato dal sensore ad ultrasuoni. Gli input di questa funzione sono 3:

- *carSpeed(speed)*: velocità lineare della macchina in cm/s
- *carAngularSpeed(speed)*: velocità angolare della macchina in rad/s
- *D*: distanza tra ostacolo e macchina

Gli output di questa funzione sono 4:

- *T1*: tempo necessario affinché la macchina ruoti di un angolo β
- *T2*: tempo necessario affinché la macchina percorra una distanza pari a *safety_distance*
- *beta*: angolo di rotazione necessario per evitare l’ostacolo
- *safety_distance*: distanza da percorrere per superare l’ostacolo

I primi due parametri in ingresso, rappresentando la velocità della macchina espressa in modalità differenti, sono vincolati tra loro. Possiamo quindi trattarli come un unico parametro che può assumere quattro differenti valori relativi alle velocità settabili: *LOW*, *MEDIUM*, *HIGH* e *MAX*. Come già detto, questa funzione viene eseguita quando il sensore ad ultrasuoni rileva l’ostacolo. In particolar modo la funzione viene eseguita quando la distanza fra l’ostacolo e la macchina è inferiore o uguale a 15 cm.

Possiamo quindi limitarci a testare la funzione per distanze che vanno da 0 a 15. Pertanto utilizzeremo tutti i valori interi compresi fra 0 e 15, in quanto la distanza viene memorizzata tramite una variabile intera. Possiamo quindi affermare che il numero totale di combinazioni su cui testare questa funzione è $4 \cdot 16 = 64$.

```
void setAvoidingParam() {
    beta = M_PI/2 - atan2(D,L+1) + SAFETY_ANGLE;
    T1 = (int) (beta / ((double) carAngularSpeed(speed)) * 1000);
    safety_distance = (d + D + 2*L - 2*L*sin(beta))/cos(beta);
    T2 = (int) (safety_distance / carSpeed(speed) * 1000);
}
```

Com'è possibile notare analizzando il codice riportato sopra, questa funzione esegue esclusivamente calcoli aritmetici. Abbiamo quindi scelto di testare i calcoli relativi ai valori temporali $T1$ e $T2$ ed i calcoli relativi all'angolo di sterzata β . Per quanto riguarda $T2$, calcoliamo la distanza percorsa facendo procedere la macchina a velocità costante per l'intervallo di tempo calcolato. Tale valore dovrà ovviamente essere uguale al valore contenuto nella variabile *safety_distance*. Il controllo che andiamo ad effettuare sarà quindi il seguente:

```
CU_ASSERT_DOUBLE_EQUAL(safety_distance,carSpeed(speed)*((double)T2)/1000,safety_distance/100);
```

Questa istruzione di assert ha esito positivo se e solo se il valore assoluto della differenza dei valori dei primi due parametri è minore uguale al valore del terzo parametro (*granularity*). Com'è possibile notare dall'istruzione abbiamo scelto come valore di soglia l' 1% del valore iniziale.

Per la variabile $T1$ abbiamo utilizzato una logica analoga, andando a calcolare l'angolo di sterzata effettuato ruotando per un intervallo temporale pari a $T1$ con la velocità angolare utilizzata dalla funzione. Anche in questo caso, come si evince dall'istruzione riportata di seguito, il valore di soglia utilizzato è pari all' 1% del valore iniziale:

```
CU_ASSERT_DOUBLE_EQUAL(beta, carAngularSpeed(speed)*T1/1000, beta/100);
```

L'ultima condizione che andiamo a testare riguarda l'angolo β . Facendo riferimento al controllore in anello aperto presentato nel capitolo 5, l'angolo di sterzata β viene calcolato utilizzando la seguente formula:

$$\beta = 90^\circ - \arctg\left(\frac{D}{L+l}\right)$$

Dove:

- L è la semi larghezza del veicolo
- l è la semi larghezza del ostacolo
- D è la distanza tra macchina e ostacolo

In teoria quindi:

$$\beta = \begin{cases} 90 & D=0 \\ (30,90) & 0 < D < 15 \\ 30 & D=15 \end{cases}$$

Nella nostra implementazione utilizziamo un angolo di margine pari a 20° per assicurarci che la macchina non urti l'ostacolo, quindi otteniamo:

$$50^\circ \leq \beta \leq 110^\circ$$

L'istruzione utilizzata per testare la validità di questo angolo sarà quindi la seguente:

```
CU_ASSERT(50 <= beta * RAD_TO_DEG && beta * RAD_TO_DEG <= 110);
```

7.4 Test: Search Line

La funzione “`void searchLine()`” aggiorna le variabili `mode` e `directionToAvoidObstacle` (output della funzione) in funzione dei valori assegnati alle variabili `newDirection` e `directionToAvoidObstacle` (input della funzione). Nella seguente tabella di verità riportiamo le combinazioni dei valori dei parametri di input associate ai relativi effetti della funzione:

<code>newDirection</code>	<code>directionToAvoidObstacle</code>	Effetto
CENTER	LEFT	Mode = LINE_FOLLOWING
CENTER	CENTER	Mode = LINE_FOLLOWING
CENTER	RIGHT	Mode = LINE_FOLLOWING
CENTER	NO_LINE	Mode = LINE_FOLLOWING
CENTER	FULL_LINE	Mode = LINE_FOLLOWING
FULL_LINE	NO_LINE	DirectionToAvoidObstacle = RIGHT

Come si evince analizzando il codice della funzione riportato di seguito, queste sono le uniche combinazioni dei parametri in input che permettono alla funzione di aggiornare i valori relativi ai parametri in output.

```
void searchLine() {
    switch(newDirection[i]){
        case CENTER:
            mode = LINE_FOLLOWING;
            //digitalWrite(LED_PIN, LOW);
            break;
        case FULL_LINE:
            //myCar.turnRight();
            if(directionToAvoidObstacle[i] == NO_LINE)
                directionToAvoidObstacle[i] = RIGHT;
            break;
        default:
            break;
    }
}
```

I test effettuati sono quindi solamente 6. Attraverso i primi 5 test testiamo il valore della variabile di stato *mode* utilizzando la seguente istruzione:

```
CU_ASSERT(mode == LINE_FOLLOWING);
```

L'ultima combinazione di valori deve attivare il branch d'esecuzione relativo al secondo *case* dello *switch*. Ci si aspetta quindi l'aggiornamento di *directionToAvoidObstacle*. L'istruzione di assert utilizzata in questo caso è la seguente:

```
CU_ASSERT(directionToAvoidObstacle[i] == RIGHT);
```

7.5 Conclusioni

Riassumendo attraverso le 3 suites, abbiamo effettuato 326 assert, di cui:

- 1 assert · 128 casi possibili nella Suite_1
- 3 assert · 64 casi possibili nella Suite_2
- 1 assert · 6 casi possibili nella Suite_3

I risultati del testing, riportati attraverso l'interfaccia *Basic*, sono i seguenti:

```
CUnit - A unit testing framework for C - Version 2.1-3
http://cunit.sourceforge.net/

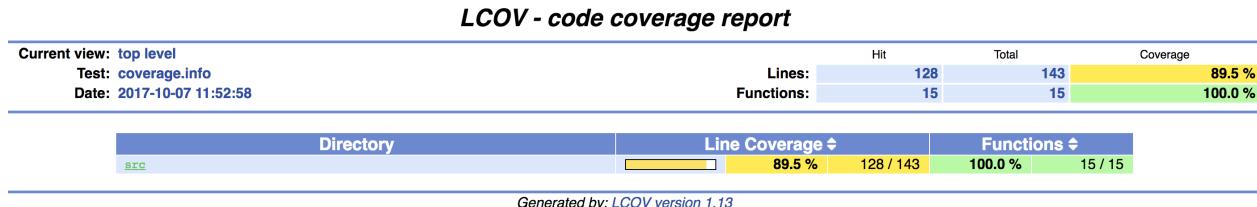
Suite: Suite_1
  Test: Test of turningDirection() ...passed
Suite: Suite_2
  Test: Test of setAvoidingParam() ...passed
Suite: Suite_3
  Test: Test of SearchLine() ...passed

Run Summary:    Type   Total     Ran  Passed Failed Inactive
               suites      3       3    n/a      0       0
                  tests      3       3       3      0       0
                 asserts    326     326     326      0     n/a

Elapsed time =    0.000 seconds
```

7.6 Gcov Testing

Per essere sicuri che i test da noi eseguiti fossero esaustivi, abbiamo testato con gcov il file “Cunit_Test.c”. I risultati ottenuti dal test sono i seguenti:



Andando ad analizzare le linee di codice mai raggiunte, abbiamo appurato che le uniche istruzioni non eseguite sono quelle relative a casi di errore, ovvero istruzioni presenti per gestire casi di errore che normalmente non si verificano. Esempi sono:

- Default case mai eseguiti negli switch
- Istruzioni per l'interruzione del programma in caso di errori quali impossibilità di aggiungere una suite, un test o altro.

Alla luce di queste osservazioni possiamo ritenere il testing effettuato con Cunit esaustivo.