Real-Time Systems Course
Master Degree in Embedded Computing Systems

Autonomous Rover: obstacle avoidance

Studente

Davide Rasla

davide.rasla@gmail.com

505828

Contenuti

1. Modello fisico

1.1 Modello fisico del Rover

1.1.1 Struttura

Il Rover è fisicamente formato da 2 servo collegati allo shield che fa da interfaccia con arduino due.
Sulla parte frontale ha un sensore ad ultrasuoni posto sopra un servo che lo fa girare da 0 a 180 gradi. Inoltre possiede una bussola CMPS03, che ritorna un valore compreso tra 0 e 3600 usato per determinare la posizione in cui si sta muovendo il Rover.

1.1.2 Modello fisico del sensore ad ultrasuoni

Il modello usato è l'HC-SR04 che dispone di 4 pin: Vcc (+5V), Trigger, Echo, GND.

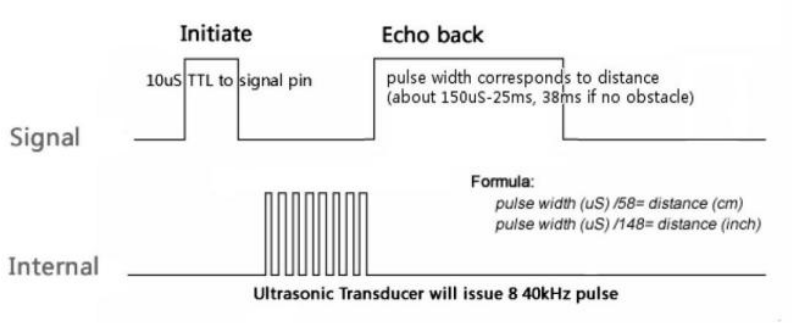


Figura 1.1: Temporizzazione sensore

Per ottenere una risposta s'invia un impulso alto sul pin Trigger per almeno 10 microsecondi, a questo punto il sensore invierà una serie di ultrasuoni per i quali aspetterà gli echi di ritorno, per poi rispondere sul pin Echo con un impulso alto della durata corrispondente a quella di viaggio delle onde sonore. Si nota che questa durata è il tempo di andata e ritorno delle onde sonore, quindi nel calcolo successivo va divisa per 2.

Per sicurezza si aspettano in genere 60 microsecondi ma dopo varie prove ho sperimentalmente notato che si assicura l'assenza di echi se si attende per 80 microsecondi. Cosi assicuriamo che le misure successive non subiscano interferenze.

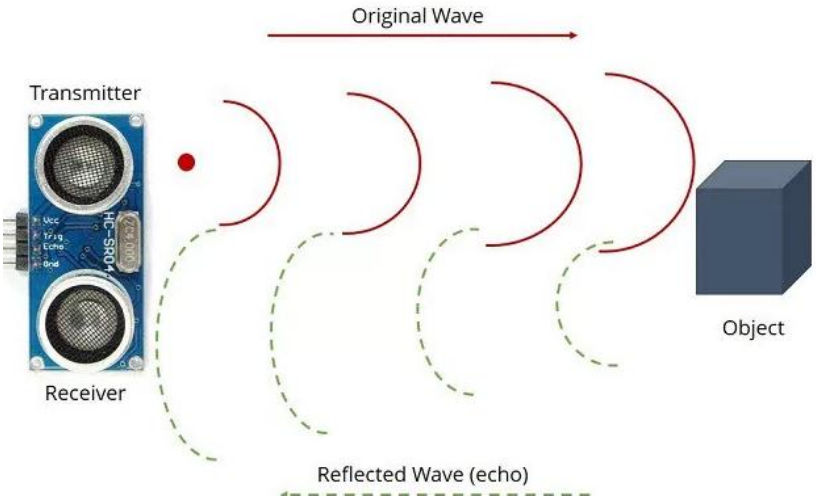


Figura 1.2: Fisica del sensore

Dopo aver atteso 80 microsecondi si legge il pin echo che rimane alto per la durata del viaggio delle onde sonore. Le onde sonore, in un materiale fluido come l'aria a temperatura ambiente, si muovono a 343 m/s a 20°.

Quindi dalla semplice equazione

$$v = \frac{s}{t}$$

Si ha che possiamo conoscere s tramite:

$$s = v * t$$

Notando però, che il tempo è quello impiegato per l'andata ed il ritorno degli echi e va quindi diviso per 2, ottenendo:

$$s = v * \left(\frac{t}{2}\right)$$

Il tutto si traduce in:

$$\text{distance} = 0.0343 * \text{duration} / 2$$

dove:

- 0,0343 è in cm/microsec;
- duration è in microsec

1.1.3 Modulo Bluetooth

Il modulo è l'RN-42 che ha la seguente struttura:

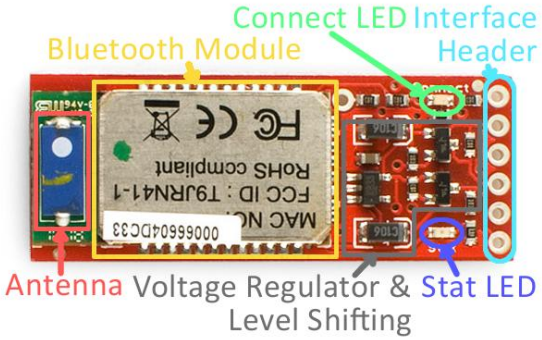


Figura 1.3: Struttura del modulo

Pin Label	Pin Function	Input, Output, Power?	Descrizione
RTS-O	Request to send	Output	Non usato
RX-I	Serial receive	Input	Qui si ricevono i dati da inviare, deve essere connesso quindi alla seriale TX sul pin 17 di arduino.
TX-O	Serial transmit	Output	Qui si ricevono i dati da ricevere, deve essere connesso quindi alla seriale RX sul pin 16 di arduino.
VCC	Voltage supply	Power In	Alimentato a 3,3V
CTS-I	Clear to send	Input	Non usato
GND	Ground	Power In	0V

Figura 1.4: Piedinatura

Al reset bisogna portare il modulo sia in fase di inquiry, e a tale scopo, leggendo la documentazione e i datasheet ho implementato la possibilità di farlo ad ogni reset di arduino, durante la funzione setup() tramite Serial2.begin(115200), per settare il baudrate e scrivendo 3 volte il carattere "$", interpretato dal modulo come ENTER-COMMAND-MODE.

In questa fase il modulo lampeggia velocemente per un minuto, tempo entro il quale va connesso al ricevitore del computer.

A questo punto può essere usato come una normale seriale tramite il metodo print().

1.1.4 Bussola

Il modello usato è il CMPS03 che ha la seguente struttura:

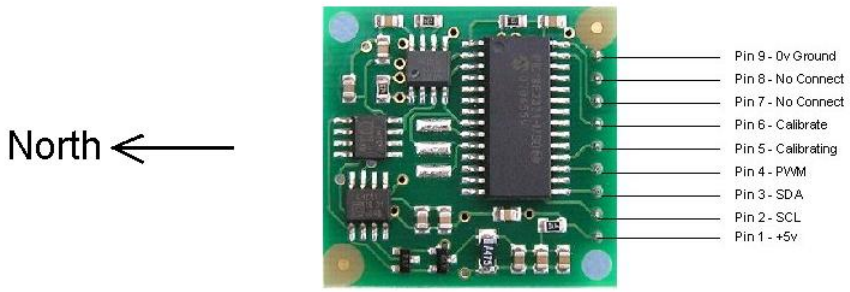


Figura 1.5: Struttura bussola

In particolare:

- i pin 2 e 3, rispettivamente SCL e SDA sono usati per leggere il valore di ritorno del modulo;
- Il pin 4 è pilotato in PWM, con l'ampiezza dell'impulso che varia da 1ms a 36.99ms;
- I pin 5 e 6 li ho usati per ricalibrare correttamente il modulo.

Arduino tramite la libreria "CMPS03.h" mette a disposizione la cmps03.read() che ha come ritorno un valore da 1 a 3600.

Range di ritorno	Direzione	Valore inviato sulla seriale
0<= ReturnValue <=600	Nord	1
601<= ReturnValue <=900	Nord-Est	2
901<= ReturnValue <=1400	Est	3
1401<= ReturnValue <=1900	Sud-Est	4
1901<= ReturnValue <=2200	Sud	5
2201<= ReturnValue <=2600	Sud-Ovest	6
2601<= ReturnValue <=3100	Ovest	7
3101<= ReturnValue <=3600	Nord-Ovest	8

2 Design choice

2.1 Visione e comportamento del Rover

Il Rover, durante la fase di scansione dell'ambiente circostante, esegue 5 misurazioni ad angoli prestabiliti; 0, 45, 90, 135 e 180 gradi a partire dalla sua destra. Queste misurazioni non superano i 100 cm. In particolare il sensore può arrivare ad una distanza teorica di 400 cm ma ho deciso che, indipendentemente dalla distanza effettiva, se misura per 3 volte una distanza superiore a 100 cm, assume l'ostacolo ad una distanza di 100cm.

Una volta ottenute le 5 misurazioni tramite il ping, sceglie la più distante e, gira le ruote in quella direzione.

In seguito prosegue in tale direzione, a velocità costante, continuando a pingare tramite il sensore ad ultrasuoni e, appena rileva una distanza inferiore a 30 cm, ferma i servo e riparte a scansionare in 5 direzioni.

2.2 Comunicazione wireless

Tramite il modulo RN-42 bluetooth si setta una comunicazione a 152000 come valore di baudrate, con il ricevitore del computer (si veda 1.1.3 per i dettagli di avvio del modulo in modalità inquiry). Il simulatore apre la seriale creata dal sistema che viene chiamata rfcomm X, dove X è un intero positivo. Non essendo noto a priori, viene passato come argomento al momento del lancio della simulazione tramite il comando

./avoidance NumeroPorta

2.2.1 Frequenza di aggiornamento

Una nuova serie di 10 byte viene inviata non appena un task effettua un ping, cosi facendo si massimizza la frequenza rendendola il più fedele possibile a quanto realmente visualizzato dal Rover. In particolare l'istogramma risulta aggiornato quasi in tempo reale con la visione dell'ultima distanza pingata dal Rover.

3 Simulazione su allegro

3.1 Introduzione

Ho creato su allegro una simulazione grafica dell'ambiente circostante intorno al Rover.
La simulazione permette di vedere a che distanza si trovano gli ostacoli dal Rover, la scelta effettuata dal Rover sulla direzione in cui spostarsi, la visualizzazione dei dati sensibili del Rover inviati via bluetooth tramite seriale e l'ultimo ping effettuato dal sensore ad ultrasuoni in tempo pressoché reale.

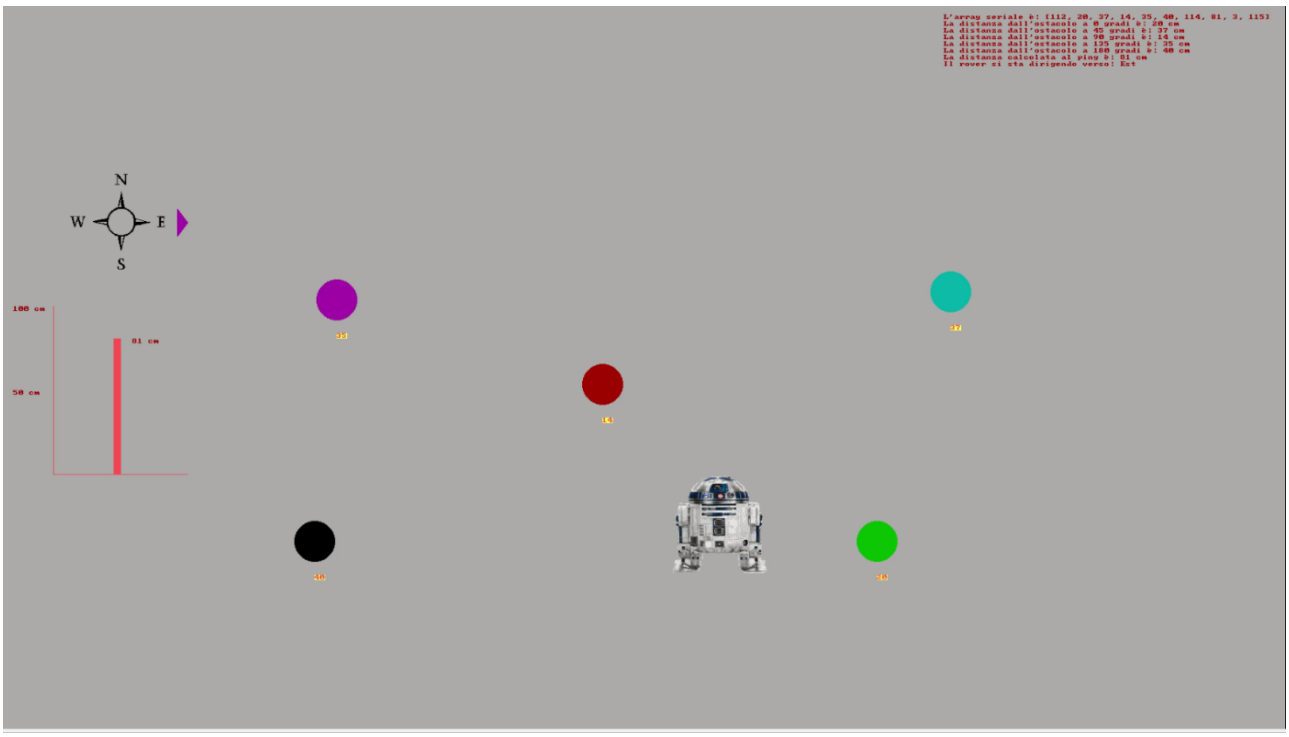


Figura 3.1: Schermata simulatore

In figura 3.1 vediamo sulla sinistra l'istogramma aggiornato in tempo reale dell'ultimo ping effettuato dal Rover ed inviato via bluetooth, e la bussola con evidenziata una freccia. Questa indica la direzione in cui si sta spostando il Rover, in questo caso est.

Sulla destra in alto vi è uno box riassuntivo con tutto quello ricevuto.

Al centro vediamo il Rover e i 5 ostacoli intorno a lui posizionati in proporzione alla loro reale distanza dal Rover.

3.2 Interfacce

All'avvio vediamo una situazione generata sulla base di quanto letto dalla seriale e riprodotta tramite una bussola, un istogramma ed una box in alto a destra con i dati ricevuti

3.2.1 Bussola

La bussola rappresenta la direzione in cui si sta muovendo il Rover, in quanto il modulo CMPS03 è posto sul fronte del Rover e, essendo sensibile al campo magnetico terrestre, permette di capire verso quale direzione ci stiamo dirigendo tra Nord, Nord-est, Est, Sud-Est, Sud, Sud-Ovest, Ovest, Nord-Ovest.

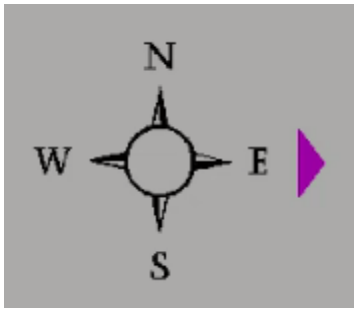


Figura 3.2: direzione Est

Per creare la bussola si legge dalla seriale alla posizione 8, in cui si ha un unsigned int da 1 a 8 che sta ad indicare la direzione presa dal Rover in tempo reale.

Il task che si occupa della grafica aggiorna quindi la posizione della freccia relativa alla direzione in base al valore letto sulla seriale

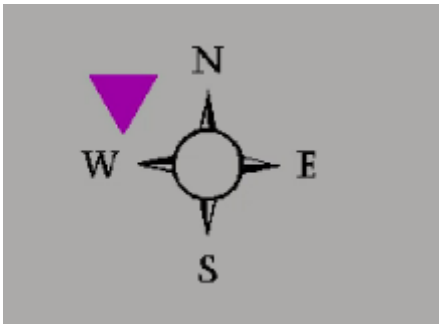


Figura 3.3: Direzione Nord-Ovest

3.2.2 Istogramma

Per massimizzare la frequenza di invio dei dati via seriale, ad ogni ping il Rover trasmette l'array di 10 byte contenente anche, alla posizione 9, l'ultimo valore di distanza pingata dal sensore ad ultrasuoni. Tale valore è nel range [0, 100] cm.

In particolare l'istogramma viene alzato o abbassato incrementando, tramite una proporzione, le 2 coordinate YP2 e YP3 dell'istogramma stesso tramite la funzione GetY(DistanzaPingata)

```
int YP2 = GetY(BufFromSerialBT[PosPing]);
int YP3 = GetY(BufFromSerialBT[PosPing]);

int pointsHistogram[SizePoints]={XP1, YP1, XP2, YP2, XP3, YP3, XP4, YP4};
```

Cosi facendo, in base a quanto letto dalla seriale, si incrementa o decrementa l'altezza tramite la proporzione:

250:100cm = x: Ping

Dove 250 è l'altezza massima dell'istogramma

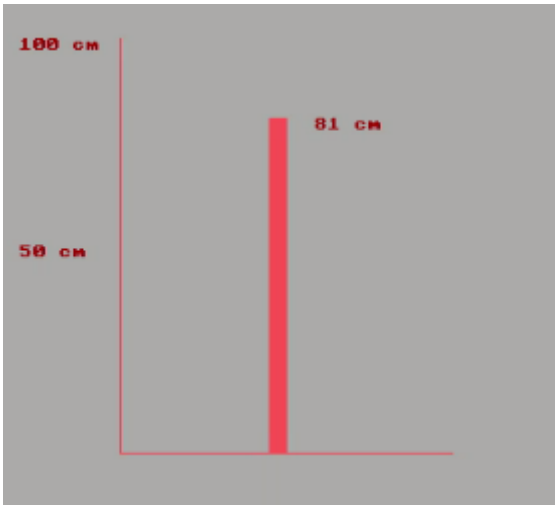


Figura 3.4: Esempio di ping

In figura 3,4 vediamo un esempio in cui il Rover ha pingato 81cm come distanza e l'istogramma è stato creato in proporzione a questo valore.

3.2.3 Box della seriale

Il box in alto a destra contiene ad ogni istante la visualizzazione dell'array di 10 byte inviati dal Rover nella seguente forma:

BufferFromSerial [0]	StartByte	112
BufferFromSerial [1]	Distanza dall'ostacolo a 0 gradi	[0-100]cm
BufferFromSerial [2]	Distanza dall'ostacolo a 45 gradi	[0-100]cm
BufferFromSerial [3]	Distanza dall'ostacolo a 90 gradi	[0-100]cm
BufferFromSerial [4]	Distanza dall'ostacolo a 135 gradi	[0-100]cm
BufferFromSerial [5]	Distanza dall'ostacolo a 180 gradi	[0-100]cm
BufferFromSerial [6]	Carattere 'r', delimitatore	114
BufferFromSerial [7]	Distanza pingata	[0-100]cm
BufferFromSerial [8]	Direzione del Rover	[0-100]cm
BufferFromSerial [9]	Carattere 's', delimitatore	115

Figura 3.5: Struttura del buffer

e viene aggiornata dal task che esegue UpdateGraph leggendo continuamente dalla seriale.

Graficamente si presenta in questa forma:

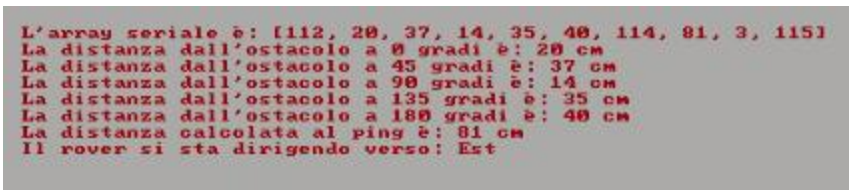


Figura 3.6: Visualizzazione grafica del buffer

4. Task e strutture dati usate

4.1 Numero e tipologia di task

La simulazione usa nove task in totale:

- Cinque task per gestire ogni ostacolo, essendo oggetti concettualmente separati tra loro;
- Un task per il gestire la posizione del Rover;
- Un task per gestire la comunicazione wireless via seriale, che avviene attraverso il trasmettitore ed il ricevitore bluetooth, leggendola come un file a gruppi di 10 byte;
- Un task per gestire la grafica che proietta sulla bmap i vari oggetti;
- Un task per ricevere i comandi dall'utente. In particolare il comando di uscita dalla simulazione.

4.1.1 Gestore della GUI

Questo task esegue continuamente la distruzione e creazione della bmap con le posizioni aggiornate degli oggetti, gestite però dagli altri task. In particolare gli altri task si occupano, leggendo dalla seriale, di modificare la posizione degli oggetti nella simulazione salvando i dati nelle relative strutture. Il task che gestisce la grafica si limita a leggere da queste strutture le posizioni aggiornate ed a proiettarle.

Si occupa quindi della proiezione della box dello status, della bussola, dell'istogramma, degli ostacoli e del Rover.

4.1.2 Gestore della serial BT

Questo task si occupa di leggere continuamente dalla seriale e di aggiornare il buffer BufFromSerialBT. Oltre a leggere sincronizza anche la comunicazione tramite un'operazione di correzione in tempo reale di quanto ricevuto.

Per farlo, ho implementato la comunicazione secondo uno standard preciso della struttura dei 10 byte inviati ogni volta dal Rover come si vede nel paragrafo 3.2.3

Di conseguenza, si rileva un errore se il primo byte differisce da '112' e si esegue uno shift, continuando a leggere, di quanto ricevuto. Cosi si ottiene sempre un dato corretto in ricezione.
Riporto qui un esempio di correzione

```
BufferFromSerialBT [0]: 36                      // byte non valido
BufferFromSerialBT [1]: 36                      // byte non valido
BufferFromSerialBT [2]: 36                      // byte non valido
BufferFromSerialBT [3]: 112  <====== Posizione in cui leggo 112
BufferFromSerialBT [4]: 89                      // Distanza 0 gradi
BufferFromSerialBT [5]: 39                      // Distanza 45 gradi
BufferFromSerialBT [6]: 40                      // Distanza 90 gradi
BufferFromSerialBT [7]: 42                      // Distanza 135 gradi
BufferFromSerialBT [8]: 79                      // Distanza 180 gradi
BufferFromSerialBT [9]: 114                     // carattere 'q'

Correzione

BufferFromSerialBT [0]: 112  <====== Posizione in cui leggo 112
BufferFromSerialBT [1]: 89                      // Distanza 0 gradi
BufferFromSerialBT [2]: 39                      // Distanza 45 gradi
BufferFromSerialBT [3]: 40                      // Distanza 90 gradi
BufferFromSerialBT [4]: 42                      // Distanza 135 gradi
BufferFromSerialBT [5]: 79                      // Distanza 180 gradi
BufferFromSerialBT [6]: 114                     // carattere 'q'
BufferFromSerialBT [7]: 78                       // Ultima distanza pingata
BufferFromSerialBT [8]: 2                       //  Direzione del Rover
BufferFromSerialBT [9]: 115                     //  carattere 's'
```

4.1.3 Task per ogni Ostacolo

Vi è un task per ogni ostacolo. Ogni task ha come argomento un valore da 1 a 5, corrispondente con la posizione all'interno del buffer seriale in cui si trova la sua nuova distanza dal Rover.
Ogni task quindi, legge all'interno del buffer seriale in una posizione specifica e calcola, tramite le funzioni GetDistanceX() e GetDistanceY(), a che distanza posizionarsi sullo schermo dal Rover.

In particolare per calcolare a che distanza porsi usa una proporzione con valori calcolati sperimentalmente

MaxRangeX: 100 = DistanzaAggiornataX : DistanzaLettaDallaSeriale

Dove MaxRangeX indica la distanza massima sull'asse X, a destra del Rover, dove può posizionarsi l'ostacolo.

Al valore ottenuto da questa proporzione si aggiunge un RangeFromXP per simulare la distanza di 30 cm dal Rover oltre il quale non vi devono essere ostacoli, in quanto il Rover si ferma prima.

Per l'asse Y vale la stessa metodologia, avremo quindi:

MaxRangeY: 100 = DistanzaAggiornataY : DistanzaLettaDallaSeriale

Il task salverà poi il valore aggiornato nella sua struttura dedicata. Tale valore verrà poi letto dal task che gestisce la grafica per aggiornare graficamente la posizione.

4.1.4 Task che simula il Rover

Questo task usa uno sprite per simulare il Rover che si muove nello spazio circostante.
Per farlo usa una costante grafica DXrover che moltiplica per un valore da 1 a 5 e somma alla posizione attuale del Rover, in maniera ciclica.

Nel dettaglio:

- Calcola tramite getmaxPosition() la direzione in cui vi è l'ostacolo più lontano. Questa funzione torna un valore da 1 a 5, esattamente come il numero di direzioni possibili;
- In base al valore di ritorno letto, aggiorna le due costanti RoverXPos e RoverYPos, contenenti la posizione base del Rover aggiungendo o sottraendo un valore;
- Questo valore è ottenuto in maniera incrementale moltiplicando la costante DXrover per tutti i valori interi da 1 a 5, compresi;

In questo modo si simula l'avanzamento del Rover nello spazio nella direzione più lontana in cui si sta muovendo il Rover.

4.1.4 Task di ricezione comandi

Questo task si occupa ti leggere i comandi da tastiera dell'utente ed in particolare, alla pressione del tasto esc, setta ad uno la variabile globale end.

Ogni task ha come condizione nel relativo while che end valga 0, quindi settando ad 1 questa variabile globale tutti i task, al successivo controllo di tale variabile usciranno dal loro ciclo while per poi terminare.

4.2 Strutture dati

Le strutture dati usate sono:

- struct Obstacle che contiene:
  - il colore dell'ostacolo;
  - il suo raggio;
  - la sua posizione sull'asse x;
  - la sua posizione sull'asse y.

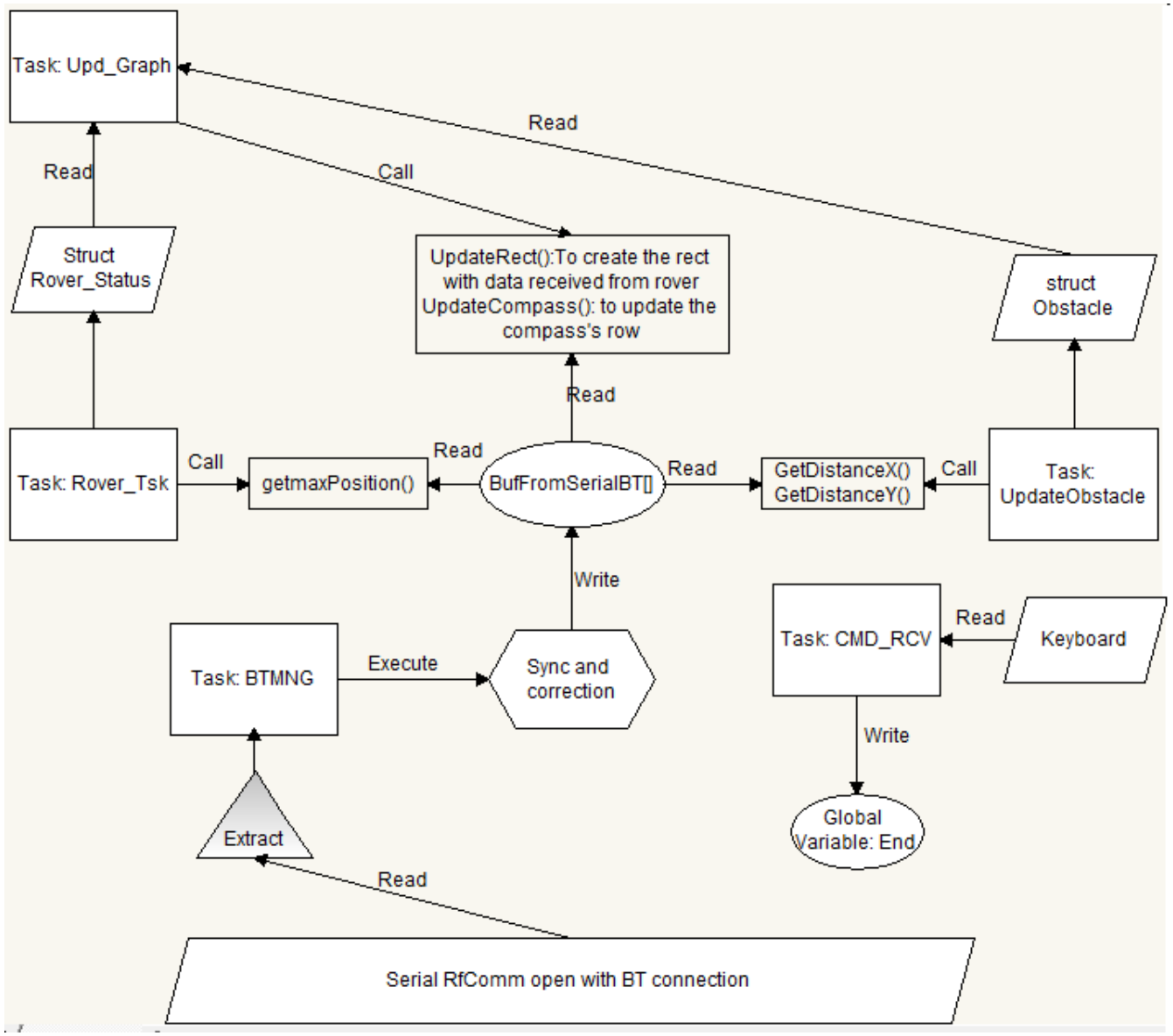
- struct Rover_Status che contiene:
  - La sua posizione sull'asse x;
  - La sua posizione sull'asse y;
  - La sua distanza da ogni ostacolo su entrambi gli assi.

- BufFromSerialBT[DimBTBuffer] dove:
    - DimBTBuffer vale 10
    - È il buffer dove continuamente si scrive quanto ricevuto dal Rover. I vari task accedono al buffer in posizioni diverse a seconda del loro scopo.

4.2 Interazione tra task

L'interazione tra i vari task può essere riassunta come segue:

- Il gestore BTMNG estrae continuamente dati dalla seriale, leggendola a gruppi di 10 byte e correggendo eventuali sincronizzazioni errate.
- I task degli ostacoli leggono continuamente, in posizioni diverse, dal buffer la loro nuova distanza ed aggiornano la loro struttura
- Il task Rover_Tsk che gestisce il Rover scorre tramite la getmaxPosition() il buffer dalla posizione 1 alla 5 (in cui vi sono le distanze), calcola quella più lontana e aggiorna la posizione del Rover nella struttura StatusRover muovendolo in tale direzione
- Il task Upd_Graph che gestisce la grafica legge le varie strutture e proietta gli oggetti sulla bitmap screen. Inoltre tramite UpdateRect() e UpdateCompass() aggiorna, leggendo dalla seriale, rispettivamente la box con le informazioni sullo stato ricevute e la direzione del Rover.



5 Task e strutture dati su arduino

5.1 Numero e tipologia di task

Su arduino vi sono 4 task, di cui uno è il task dummy.

- Il primo si occupa di scansionare in 5 direzioni ed ottenere quindi 5 distanze ad una distanza massima di un metro, per poi decidere la direzione più lontana da ogni ostacolo;
- Il secondo si occupa di girare nella direzione più lontana, scelta dal primo task, l'intero Rover;
- Il terzo si occupa di proseguire in tale direzione continuando ad effettuare dei ping tramite il sensore ad ultrasuoni ed, appena si rileva una distanza inferiore a 30cm, disattivare i motori permettendo una nuova scansione delle distanze.

5.1.1 Scansione

In questa fase arduino si pone, tramite un servo che gira il sensore ad ultra suoni, a gradi specifici. In particolare, partendo da destra, si pone a 0, 45, 90, 135, 180 gradi ed in ogni direzione effettua un ping. Se rileva per 3 volte una

distanza superiore a 100cm, pone come distanza esattamente 100cm cosi da assumerla come potenziale direzione in cui muoversi, altrimenti si sposta nella direzione ottenuta trovando la massima distanza tra quelle calcolate.

5.1.2 Cambio di direzione

In questa fase, una volta che si ha una scansione valida e una direzione in cui andare, questo task legge la variabile globale PosToGo, settata dal primo task, ed in base a tale valore gira le ruote per posizionare il Rover in tale direzione

5.1.3 Movimento e scansione

In questa fase il Rover si trova in una situazione chiara con tutte le distanze note e la direzione in cui proseguire definita. Può quindi proseguire dritto a velocità costante continuando ad effettuare dei ping per poi fermarsi quando, per due volte, si misura una distanza inferiore a 30cm.
La scelta di effettuare due misurazioni è dovuta al voler evitare falsi positivi dovuti a qualche interferenza o eco di ultrasuoni.

5.1.4 Comunicazione via bluetooth

Per massimizzare la frequenza di invio dati dal Rover al bluetooth, soprattutto per creare un istogramma che sia il più fedele possibile a quanto sta avvenendo in tempo reale, la comunicazione e l'invio dei 10 byte via bluetooth avviene ogni volta che il Rover effettua un ping, tramite la funzione SendViaBt() che tramite una serial2.print() comunica con il modulo bluetooth che provvede ad inviare i dati al ricevitore sul computer.

5.2 Strutture dati e funzioni

Le principali strutture dati che permettono il corretto funzionamento dei task sono:

- MaxDistance: settata dal primo task, memorizza la distanza massima tra le 5 calcolate;
- PosToGo: settata dal primo task e usata dal secondo, memorizza la direzione in cui andare;
- ScanValid: Settata ad uno quando il Rover si trova in una situazione in cui ha una scansione valida;
- TurnedValid; Settata ad uno quando il Rover si è posto nella direzione in cui muoversi successivamente;
- isMoving: Settata ad uno quando il Rover è in movimento;
- CompassPosition: contente l'intero da 1 a 3600 ottenuto dal modulo CMPS03;
- Distance: Variabile globale contenente l'ultima distanza pingata dal Rover, usata per creare l'istogramma;
- DistObstacle []: Array contentente le 5 distanze calcolate dalla funzione Scan().

Tra le funzioni principali si ha:

- ConvertCompass(unsigned int) che converte il range di valori della bussola in un intero da 1 a 8, inviato poi via bluetooth al computer;
- cmDistance (): Usata per effettuare ping con il sensore ad ultrasuoni fino ad ottenere una distanza valida che viene memorizzata nella variabile globale Distance. Inoltre chiama la funzione SendViaBt() per inviare dati al ricevitore BT;
- Scan(int * MaxDist, int DistOb[]): usata per scansionare in 5 direzioni. Tali direzioni sono scritte nell'array DistObstacle[]. Inoltre la funzione setta la variabile globale MaxDist con la massima distanza calcolata. Ritorna la direzione in cui andare, settando la variabile PosToGo;

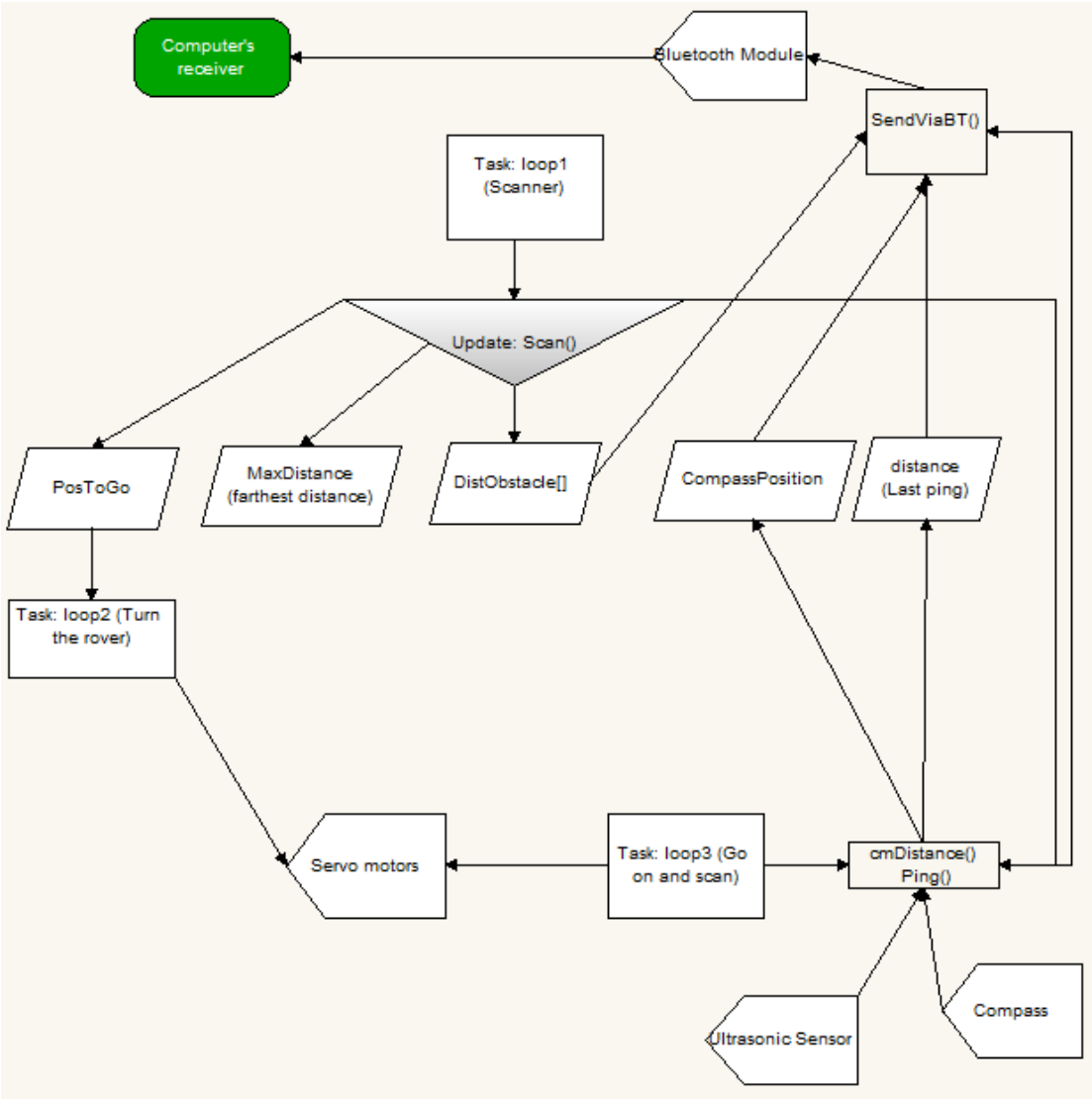
- ChangeDirection(PosToGo): legge la variabile globale PosToGo settata dal task 1 ed in base al valore letto gira il Rover nella direzione opportuna;

- SendViaBT(): Usata per inviare dati via bluetooth, è chiamata da cmDistance(), quindi ogni volta che il Rover effettua un ping.

5.3 Interazione tra i task

I task vengono continuamente schedulati e leggono variabili globali in base al quale è noto lo stato in cui si trova il Rover. In particolare il task che si occupa di girare, verifica che ci sia una direzione valida in cui andare ed una scansione completa, mentre il task che si occupa di proseguire in una direzione verifica che il Rover si trovi esattamente in quella direzione prima di proseguire.

Il seguente diagramma riassume il tutto:



Dove In particolare si nota:

- Il task loop1 che tramite la scan(), che chiama la cmDistance(), aggiorna le variabili globali;
- Il task loop2 che legge la variabile PosToGo e attiva i motori servo per girare il Rover;
- Il task loop3 che attiva i servo per muoversi in avanti e contemporaneamente tramite la cmDistance() effettua dei ping;
- La funzione cmDistance chiama la SendViaBt() che prende le variabili globali aggiornate e le invia via bluetooth.

5.4 Schema elettrico

Il seguente è lo schema elettrico dei collegamenti di arduino. Per questioni di qualità, all'interno della cartella lascio anche i .PNG