Università di Pisa

Laurea Magistrale
in Ingegneria Robotica e dell'Automazione

Progetto di Sistemi di Guida e Navigazione

Charlie esplora l'universo

Autori:
Alessia Biondi
Francesco Petracci

Professore:
Lorenzo Pollini

Indice

Introduzione

L'obiettivo di questo progetto è stato quello di migliorare lo stato del veicolo, partendo dal risolvere le molte problematiche accumulatesi nel passaggio di testimone tra i vari gruppi. Lo scopo principale dell'intero sistema, composto dal veicolo affiancato da una serie di sensori, è quello di riuscire a localizzarsi all'interno di una mappa preacquisita e di navigare al suo interno. La posizione è ottenuta seguendo due metodologie tra loro complementari: da una parte si sfrutta il lidar montato sul corpo del veicolo, che permette di avere buoni risultati in ambienti chiusi dove siano presenti pareti e confini ben precisi, dall'altra si appoggia ad un sistema Ultra Wide Band (UWB), che ha invece performance migliori in ambienti esterni privi di ostacoli sui quali il segnale possa avere interferenze dovute a scattering. È importante focalizzare fin da subito che, attraverso il lidar, non viene effettuata una SLAM vera e propria bensì uno Scan Matching. Infatti, l'algoritmo di localizzazione in condizioni nominali prende come posa del veicolo quella ottenuta dallo scan matcher. Quest'ultima viene periodicamente confrontata con quella misurata dal sistema UWB: solo nel momento in cui i due valori restituiti differiscono di molto, viene riposizionato il veicolo all'ultima posa ottenuta dalle antenne. In questo modo si ottiene un sistema robusto alla perdita del lidar, che può verificarsi a seguito di una rottura o nel momento in cui sono esplorati ambienti dove le condizioni non permettono di avere misure affidabili.

Funzionamento in due parole

Molto sinteticamente andiamo a descrivere il funzionamento di Charlie. In primis viene utilizzato il pacchetto hector-slam per realizzare una mappa. In un secondo momento, utilizzando questa mappa si riesce a localizzarsi confrontando le misure acquisite dal lidar con le feature della mappa stessa attraverso l'algoritmo Adaptive Monte Carlo Localization.

Dato che questo algoritmo può convergere su una posa del veicolo non corretta a causa, per esempio, di ambiguità tra simmetrie della mappa il sistema UWB fa sì che venga reinizializzato l'algoritmo AMCL. In questo utilizzo quindi il sistema UWB serve solamente come check.

Come ottenere codice

Il codice sviluppato è disponibile nella repository github e un backup del catkin workspace sul Google Drive associato a Charlie. e mini organizzazione **FARE ALLA FINE QUANDO PRESENTIAMO IL CODICE**

1 Descrizione Hardware

Il veicolo, per gli amici e i lettori Charlie, è basato su un Crawler RC, una piattaforma meccanica radio–comandata, su cui sono stati installati dei sensori e delle schede elettroniche.
A bordo si trovano quindi due unità centrali:

- un Raspberry Pi 4 (8Gb Ram), con sistema operativo Linux 18.04 su cui viene eseguito Robot Operating System (ROS)
- una scheda STM32F407 connessa ad una pcb Icaro su cui è implementato il sistema di guida e alcuni filtraggi

Come sensori sono presenti:

- Lidar Slamtec RPLIDAR-A3
- due tag del sistema UWB creato da Pozyx che dialogano con 4 anchors disposte nell'ambiente

Alimentazione e Connessioni

Il robot è dotato di due batterie:

- LiPo 14.80 V 4200 mA h, dedicata ad un'alimentazione generica, che viene sfruttata da tutti i componenti tranne i motori; da questa partono 3 linee di alimentazione:
  - a 14.80 V per la STM
  - a 5 V per la Raspberry con connettore usb-c
  - a 5 V per fornire alimentazione ausiliaria all'HUB-USB
- LiPo 7.40 V 6000 mA h o NiMH 7.20 V 3000 mA h, dedicate ai motori

I vari convertitori di tensione sono tutti installati su una PCB, che è posizionata all'interno di un box metallico.

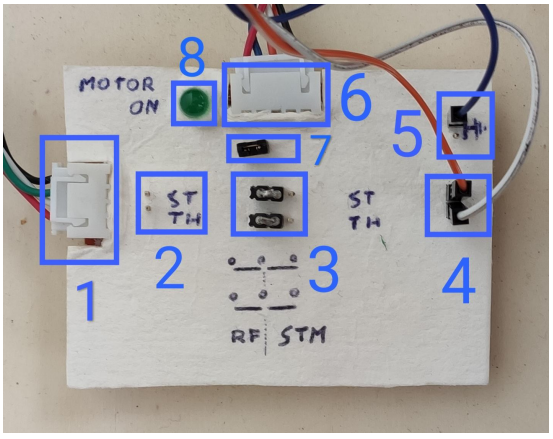

Figura 1: PCB controlli

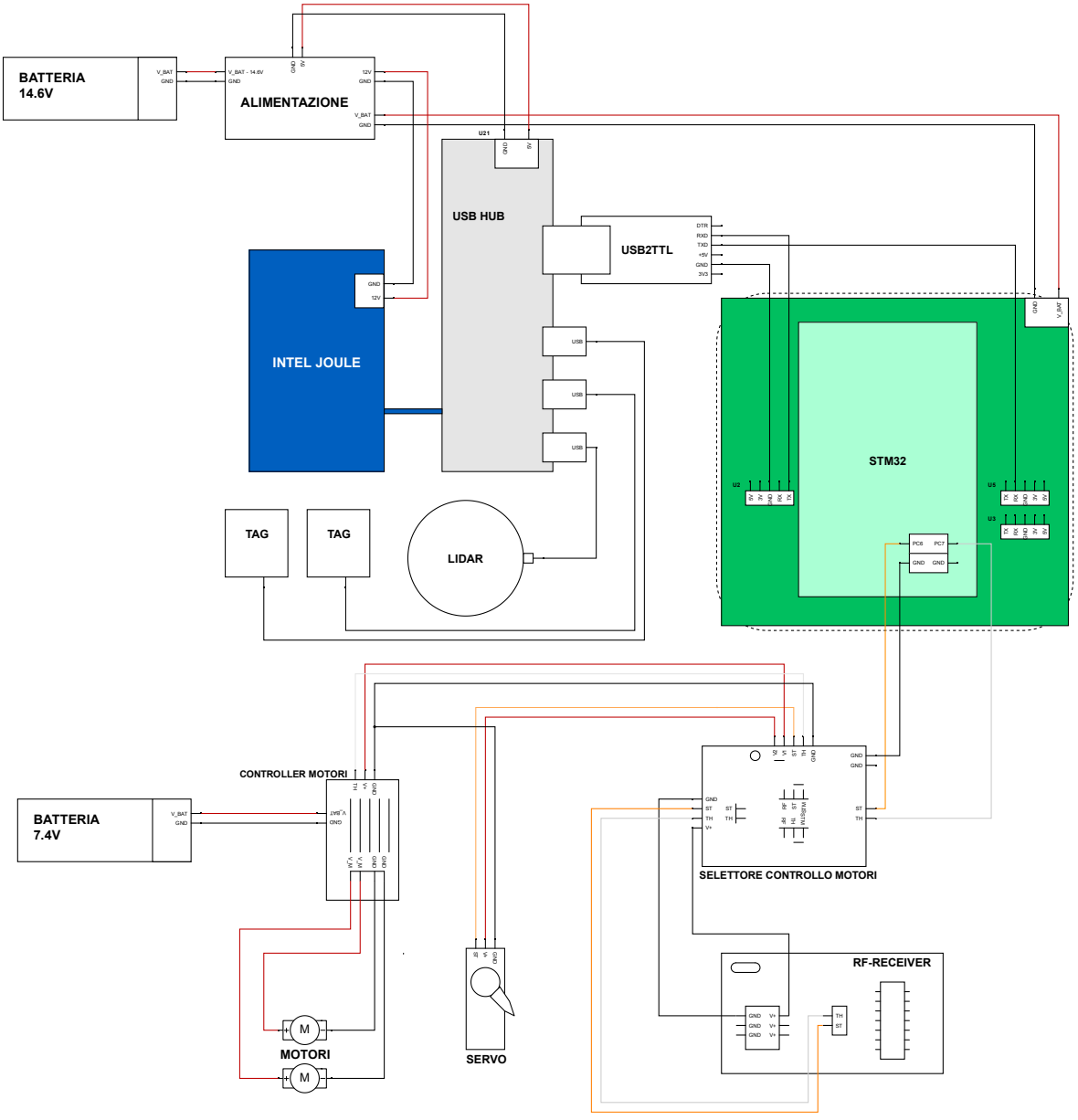

Figura 2: Schema connessioni completo. **DA EDITARE CON RASPBERRY**

Altro elemento da evidenziare è la piccola PCB che consente di utilizzare, alternativamente all'STM, un radiocomando per il controllo di sterzo e acceleratore. Questa PCB, visibile in fig. 1, ha un led di stato verde che notifica quando l'alimentazione dei motori è attiva; inoltre sono presenti due jumper che permettono di scegliere tra STM e radiocomando come sorgente del controllo per sterzo e acceleratore. Sono sempre disponibili all'utente i contatti per potersi connettere e leggere quanto prodotto dal radiocomando.

1. Connettore per il ricevitore del radiocomando;
2. Pin per prelievo segnali PWM (steering/throttle) provenienti dal radiocomando;
3. Jumper di selezione per la fonte di controllo: (jumper a sinistra) controllo da radiocomando e (jumper a destra) controllo da STM;
4. Pin per connessione dei canali PWM provenienti da STM (steering/throttle);
5. Pin per connessione GND comune;
6. Connettore per i motori;
7. Jumper di abilitazione alimentazione motori;
8. LED di stato, indica quando l'alimentazione dei motori è attiva.

Per ulteriori dettagli e motivazioni dietro alle connessioni fatte riferirsi a [DWP21]. Lo schema completo è mostrato in fig. 2.

1.1 Primo collegamento a Raspberry

Per iniziare a lavorare sulla raspberry in ssh, modalità richiesta per gli esperimenti, è necessario impostare le connessioni WiFi a cui la raspberry si connette automaticamente all'avvio. Per comodità suggeriamo di collegare la raspberry ad un monitor tramite un cavo microHDMI-HDMI e una tastiera/mouse usb.

Abbiamo scritto tutto il necessario nel file: `Info/README_wifi_settings.txt` e adesso lo riportiamo brevemente anche qui. Il file con le varie impostazioni WiFi chiamato `interfaces` si trova in `/etc/network/`. Per editarlo:

```
sudo nano /etc/network/interfaces
```

Il file legge le configurazioni salvate nei file `.conf` di cui sono indicati i path.

Per aggiungere una configurazione aggiungere una riga per file `.conf` e commentare le altre (con #). Ad esempio se volessimo aggiungere una configurazione di rete con nome "NOME_RETE" è consigliato inserire la seguente riga all'interno del file (attenzione: l'ordine conta! Vengono tentate prima le reti riportate):

```
1 wpa-conf /etc/wpa_supplicant/wpa_supplicant_NOME_RETE.conf
```

Il file `wpa_supplicant_NOME_RETE.conf` deve contenere:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=IT

network={
    ssid="NOME_RETE"
    psk="password"
}
```

e deve essere posizionato nel path specificato. Nel caso si usi quello di default (`/etc/wpa_supplicant/`), si riporta un comando utile per creare un nuovo file:

```
sudo nano /etc/wpa_supplicant/wpa_supplicant_NOME_RETE.conf
```

Al prossimo riavvio la raspberry si connetterà automaticamente alla prima rete (in ordine di righe) configurata disponibile. Dato che si è già connessi tramite lo schermo suggeriamo di fare un backup, tramite l'apposito tool all'interno di raspbian, della scheda originale così da avere un punto di ripristino.

1.2 Ros master/slave

Ai fine di non sovraccaricare la raspberry e di avere un'interazione fluida con rviz e altri tool grafici di ros è fortemente consigliato sfruttare ros su più computer. Riportiamo i passaggi utilizzati da noi così da facilitare i nuovi utenti; per saperne di più consultare la guida ufficiale.

Quello che vogliamo fare è avere il RosMaster su raspberry e utilizzare il pc per tool grafici, esempio rviz o rqt. Otteniamo l'ip dei nostri dispositivi attraverso vari comandi (come `ifconfig` o `ip address`) e generalmente sarà qulcosa del tipo: `192.168.43.247`. Ipotizziamo quindi che l'ip della raspberry sia `IPrasp` e quello del pc `IPpc`. Adesso occorre modificare il file `.bashrc` sia su raspberry che su pc. Iniziamo da pc:

```
nano ~/.bashrc
```

e inseriamo a fine file:

```
# ROS MASTER SU RASPBERRY, LATO PC
export ROS_MASTER_URI=http://IPrasp:11311/
export ROS_HOSTNAME=IPpc
export ROS_IP=IPpc
```

Ripetiamo l'operazione su raspberry:

```
nano ~/.bashrc
```

e quindi:

```
% # ROS MASTER SU RASPBERRY, LATO RASPBERRY
% export ROS_MASTER_URI=http://localhost:11311/
% export ROS_HOSTNAME=IPrasp
```

```
4    % export ROS_IP=IPrasp
```

A questo punto ravviare le shell dei terminali che si vogliono utilizzare, e sarà possibile utilizzare lanciare i nodi, installati su pc, da pc ma sfruttando il master su raspberry.

2 RPlidar

Il lidar utilizzato su Charlie è un RPlidar A3 prodotto da Slamtec, documentazione e caratteristiche sono disponibili sulla pagina del produttore. Per quanto riguarda il codice, abbiamo scelto di utilizzare il pacchetto ros sviluppato proprio da Slamtec. Alcune accortezze hardware sono state:

- utilizzare un cavo micro usb di buona qualità. Infatti questo cavo è responsabile sia dell'alimentazione del motore brushless sia della seriale di comunicazione. Ci siamo accorti che se il voltaggio in ingresso al lidar cala sotto i 4.70 V la comunicazione seriale si interrompe e non è più possibile sfruttare i nodi ros di RPlidar.
- orientare correttamente il sistema di riferimento del lidar, in quanto le rappresentazioni nel datasheet sono invertite di π. L'asse x ($\vartheta = 0$) infatti coincide con l'uscita del cavo dalla struttura principale come si può vedere in fig. 3.
- cercare di far vibrare il meno possibile la struttura di sostegno e rialzo del lidar.

Per evitare che l'ordine di inizializzazione delle porte usb della raspberry non influenzasse il nome del collegamento (per esempio: `ttyUSB0` oppure `ttyUSB1`) abbiamo realizzato un symlink in modo tale che la porta seriale del lidar sia sempre chiamata `ttyUSBlidar`. Su come si realizzi tale symlink abbiamo riportato la procedura dentro il file `/Info/renaming_ttyUSB.txt`.
In ambiente ros viene eseguito il lidar andando a lanciare il nodo `rplidarNode` che pubblica sul topic `/scan`. È stato sviluppato il file di lancio `rplidar_a3.launch` con i parametri opportuni. Per quanto riguarda l'utilizzo del lidar non è stato necessario alcun sviluppo software in quanto il pacchetto ros risulta già completo.

Figura 3: Riferimenti **DA EDITARE**

3 Sistema Pozyx

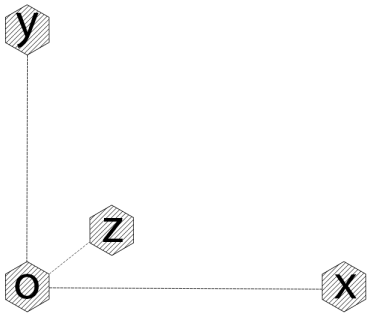

Figura 4: Disposizione ancore

Il Pozyx è un sistema di localizzazione alternativo al GPS, basato sulla tecnologia ultra-wideband (UWB). In breve, si tratta di una tecnica di trasmissione sviluppata per la trasmissione e la ricezione di segnali sfruttando degli impulsi di energia a radiofrequenza di durata temporale molto ridotta (nanosecondi) con conseguente banda spettrale ampia. Nel sistema Pozyx si possono distinguere due entità: le ancore e le tag. La configurazione base prevede l'utilizzo 4 ancore, che fungono da dispositivi fissi, e un tag, che rappresenta il dispositivo mobile. Grazie all'utilizzo di un algoritmo di triangolazione, il tag si può localizzare rispetto alle ancore. Teoricamente, la localizzazione effettuata sfruttando la tecnologia ultra-wideband permette di ottenere un posizionamento con una accuratezza al centimetro anche in ambienti indoor e in presenza di ostacoli di natura non metallica, condizioni sfavorevoli per la natura fisica del sistema (che presenta migliori performance in ambienti aperti ed ampi, privi di oggetti metallici e ostacoli sui quali possano verificarsi fenomeni di scatterning del segnale inviato). Sia all'interno delle tag che delle ancore è presente un microcontrollore, nel quale vengono eseguiti la maggior parte dei task richiesti, come gli algoritmi di triangolazione e autocalibrazione. A lui è demandata anche la gestione dell'interfaccia con dispositivi esterni (come Arduino). Solo nel caso delle tag, che per questo sono utilizzate come dispositivi mobili, sono presenti i seguenti sensori: un magnetometro, un giroscopio, un accelerometro e un sensore di pressione (altimetro). I dati dei sensori vengono utilizzati dal microcontrollore per la triangolazione e sono disponibili all'utente. Per quanto riguarda l'utilizzo di tale sistema di posizionamento, è necessario prima di tutto predisporre l'ambiente di lavoro. La prima cosa da fare è posizionare le ancore in modo tale che delimitino uno spazio entro il quale il sistema di localizzazione funzioni correttamente. Il miglior modo di disporle è quello di metterle in alto e sulla line-of-sight dell'utente: questo tipo di scelta aumenta infatti la possibilità di ricevere un buon segnale in quanto si limita la presenza di possibili ostacoli. Per un buon funzionamento e per ridurre il più possibile l'errore di posizionamento, è consigliato inoltre distribuire le ancore affinché tutte le direzioni siano coperte. Nel caso, infatti, in cui gli ancoraggi si trovino su una linea retta, l'errore di posizionamento risultante sarà molto grande e potranno verificarsi ambiguità di posizione legate alla simmetria della configurazione. Importante sottolineare che le ancore devono essere disposte verticalmente con l'antenna UWB rivolta verso l'alto.

Per tutte le operazioni che vengono richieste tra le ancore e le tag, è necessario specificare tra quali dispositivi lavorare. Con l'identificativo assegnato al campo `remote_id`, si dichiara la tag o l'ancora sulla quale vogliamo usare le funzioni di registro. Nel caso (di default) in cui questo sia settato al valore `None`, verrà usata la tag locale, ovvero quella connessa al pc (su cui viene utilizzata la seriale). Presa ad esempio la funzione doRanging(destination, `device_range`, `remote_id=None`), essa effettua una misurazione della distanza tra i dispositivi i cui identificativi sono `remote_id` e destination. Lasciando il valore di default per `remote_id`, cioè `None`, la distanza viene automaticamente valutata tra il dispositivo connesso alla seriale del PC e il dispositivo il cui identificativo è destination. Se si desidera invece che il ranging venga effettuato tra due diversi dispositivi della rete basterà inserire l'opportuno valore di `remote_id`.

Nel nostro caso, questo aspetto è importante nel momento in cui vengono prelevate le posizioni relative delle ancore nella configurazione attuale. Con `remote_id=None` chiediamo di fatto alla tag connessa in seriale ad Arduino le informazioni delle ancore, ovvero leggiamo sul suo chip i valori relativi all'ancora richiesta (posizione, id...). Per questo motivo, ad ogni nuovo avvio del sistema è necessario salvare all'interno delle tag i valori di posizione più recenti delle ancore, ovvero gli ultimi salvati all'interno delle ancore stesse al termine del processo di autocalibrazione. Affinché il sistema funzioni è infatti necessario che ogni ancora abbia coscienza della sua posizione rispetto alle altre ancore: la procedura di autocalibrazione è quella utilizzata a tale scopo. Per informazioni inerenti alla procedura adottata, vedere app. A. Per come funziona l'algoritmo, l'ancora 0 è quella scelta come origine del frame UWB che verrà costruito. L'ancora 1 è quella che fornisce la direzione dell'asse y, ovvero le cui coordinate finali saranno

```
(0, y1, 0)
```

, mentre l'ancora 2 quella dell'asse x. La quarta ancora, posizionata a una altezza diversa dalle altre 3 (che devono invece essece complanari), evidenzia la direzione dell'asse z.

infine, è importante verificare che siano assenti eventuali interferenze elettromagnetiche tra i dispositivi, le quali, specialmente se interposte tra le ancore, possono rendere difficile la comunicazione tra i dispositivi della rete. ********

* tag * anchors, come disporre le ancore
* autocalibrazione * come gestire flash memory dei device * warning: remote_id problematiche relative al doPositioning veloce (servono le pause) pacchetto ros custom descrivere in breve cosa fanno i file dentro charlie_pozyx

4 Sistema Vicon

come si installa
come si crea un oggetto
come si calibra

5 L'esperimento

procedura con i comandi, ogni cosa che facciamo è commentata e descritta
topic `/scan` letto da hector_slam produce mappa e da scan_tools produce la tf da laser

odom a base_link
albero nodi
albero tf

6 Prove?

6.1 Confronto Vicon

6.2 Esperimento nel cortile

6.3 Esperimento all'aperto

7 Guida breve all'esperimento

In questa sezione è scritta, in modo molto sintetico, la procedura per lanciare un esperimento. Per una guida dettagliata rifarsi a sez. 5.

Prima di tutto è necessario disporre le ancore come descritto in sez. 3 e come si può vedere in fig. 4. Assicurarsi di aver impostato correttamente le impostazioni WiFi (sez. 1.1) e di avere il PC e la Raspberry connessi alla stessa rete WiFi.

Connettersi **ssh** alla Raspberry dal PC digitando da terminale del PC, con password `robot`:

```
ssh -X -C pi@raspberrypi.local   # avvia ssh
```

Per l'**autocalibrazione** (necessaria ogni volta che viene riposizionato il sistema di ancore) lanciare dalla cartella `home` (da terminale della Raspberry):

```
python3 /charlie_autocalibration/autocalibration_ransac.py
```

infine rispondere "y" per salvare i risultati nella memoria flash delle ancore.

Adesso è necessario avviare il **posizionamento** delle tag pozyx e la comunicazione **seriale** con Icaro (suggeriamo di utilizzare più terminali con terminator):

```
roslaunch charlie_launch start_uwb.launch  # avvia uwb
roslaunch charlie_launch start_serial.launch # avvia seriale con stm
```

Nuova Mappa

Una nuova mappa è necessaria quando si cambia posizionamento alle ancore o banalmente si cambia luogo. Consigliamo di muovere Charlie attraverso il radiocomando in questa fase.

```
roslaunch charlie_launch save_map_origin.launch  # con Charlie fermo!
roslaunch charlie_launch new_map.launch  # muovere Charlie lentamente
```

Una volta soddisfatti del risultato salvare la mappa attraverso:

```
cd charlie_ws/maps
rosrun map_server map_saver -f NOME_MAPPA
```

Localizzazione

Sostituire il nome della mappa che si vuole utilizzare nel file `/home/pi/charlie_ws/src/charlie_launch/launch/localization.launch`:

```
14    <arg name="file_NUM" default="NOME_MAPPA" />
15    <node name="map_server" pkg="map_server" type="map_server" args="$(arg
          ␣path)$(arg␣file_NUM).yaml"/>
```

Quindi lanciare:

```
roslaunch charlie_launch localization.launch
```

Nello stesso file è possibile impostare di visualizzare rviz attraverso la connessione ssh con il "Compressed X11 Forwarding " scommentando la riga corrispondente di rviz. Questa opzione è molto sconsigliata da noi in quanto rende la visione poco reattiva. Per ovviare a questo problema, dopo aver configurato pc e raspberry come descritto in sez. 1.2, è possibile lanciare da pc:

```
roslaunch charlie_remote rviz_remote.launch
```

In questo momento l'esperimento è iniziato!

Waypoints

È possibile indicare una posa-goal tramite il comando `2DNavgoal` direttamente da rviz (tenere premuto per assegnare l'orientazione). Per attivare i motori e permettere al robot di spostarsi, pubblicare il seguente messaggio sul topic `start\_and\_stop`:

```
rostopic pub /start_and_stop std_msgs/Float64 "data: 1.0"
```

e per fermarli:

```
rostopic pub /start_and_stop std_msgs/Float64 "data: 0.0"
```

Sistema Vicon

Ovviamente per utilizzare il sistema Vicon è necessario essere nella stanza del volo. Per l'installazione rifarsi a sez. 4. Una volta che il sistema è in funzione, avviare l'applicazione "Vicon Tracker 3.7.0 x64" e selezionare nella lista oggetti: `Charlie` e `Active Wand v2 (Origin Tracking)`.

Per avviare i dialoghi tra ros e il sistema Vicon avviare da pc:

```
roslaunch charlie_remote vicon_charlie.launch
```

poi salvare la trasformazione tra vicon e uwb (all'utente è richiesto di posizionare la wand in successione sulle ancore):

```
rosrun charlie_remote vicon2uwb_tf.py
```

e infine lanciare il nodo che pubblica la posizione di Charlie in frame map:

```
rosrun charlie_remote charlie_vicon2map.py
```

Rosbag

Per registrare i dati attraverso una rosbag suggeriamo di non sottoscriversi a tutti i topic ma di lanciare il seguente comando da pc (dopo essersi spostati nella cartella desiderata):

```
rosbag record /clock /initialpose /map /orientation /particlecloud /
   robot_pose /rosout /rosout_agg /scan /tag_center /tf /amcl_pose /
   charlie_vicon_map -O NOME_BAG.bag
```

Per eseguire i topic necessari a AMCL, prima riconfigurare il file:
`../charlie_remote/launch/exec_bag.launch` con i file e il path che si vogliono utilizzare e quindi lanciare da pc:

```
roslaunch charlie_remote exec_bag.launch
```

e da raspberry (modificando il nome della mappa da utilizzare all'interno del file di lancio `localization_bag.launch`):

```
roslaunch charlie_launch localization_bag.launch
```

A Autocalibrazione

L'autocalibrazione si basa sullo script `python3` "autocalibration_ransac.py". Per farlo funzionare occorre per prima cosa avere quattro ancore correttamente alimentate ed un dispositivo Pozyx connesso, il quale servirà da comunicazione seriale tra la rete Pozyx e l'utente. Il dispositivo seriale può, a discrezione dell'utente, essere un'ancora o un tag. É possibile, se si desidera, effettuare una calibrazione manuale delle ancore, andando a settare la variabile `autoCal` a `True`. In tal caso si dovrà utilizzare un metro per misurare la distanza relativa tra le coppie di ancore della rete ed inserire manualmente i valori misurati nelle opportune variabili `r01`, `r02`, `r03`, `r12`, `r13` ed `r23`, che rappresentano le distanze tra le rispettive antenne. Per quanto riguarda invece la calibrazione automatica, lo script prevede una fase di acquisizione dei dati necessari, successivamente viene eseguito l'algoritmo ransac ed infine viene utilizzato l'algoritmo algebrico per determinare le coordinate effettive delle ancore. L'algoritmo ransac rimuove eventuali outliers dalle misurazioni delle distanze relative tra le antenne e fornisce quindi una stima della distanza tra ciascuna coppia di ancore basata sui dati senza outliers. Nel corso della procedura di autocalibrazione vengono stampati sul terminale vari dati, tra cui i fondamentali sono:

- Coordinate dei dispositivi all'accensione del sistema, prima che sia effettuata la nuova calibrazione;
- Risultato del settaggio dei parametri UWB della rete;
- Il risultato dell'algoritmo ransac per la distanza tra le ancore;
- Risultato dell'algoritmo algebrico per determinare le coordinate delle ancore: in caso di fallimento dell'algoritmo, è mostrato su terminale l'errore;

Una volta terminata la calibrazione, sia questa stata manuale o automatica, ciascuna ancora avrà salvato nella propria lista dei dispositivi le proprie coordinate, ossia conoscerà le proprie coordinate. Questi dati sono salvati permanentemente nella memoria flash. In una successiva fase di positioning è quindi buona norma che il dispositivo che si deve localiz- zare interroghi le ancore per conoscere la loro posizione e crei quindi una propria lista interna dei dispositivi della rete.
Per maggiori dettagli, fare riferimento al dettagliato lavoro [CT18].

Riferimenti bibliografici

[CT18] Luca Crosato Cristian Tesconi, *Studio e caratterizzazione del sistema pozyx*, Progetto di Sistemi di Guida e Navigazione (2018).

[DWP21] Federico Viviani Do Won Park Luca Tedeschi, *Localizzazione mista di un veicolo attraverso sensori lidar e ultra-wideband*, Progetto di Sistemi di Guida e Navigazione (2021).