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 gruppi precedenti. 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 viene sfruttato un lidar montato sul corpo del veicolo, che permette di avere buoni risultati in ambienti chiusi in cui 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 e privi di ostacoli, sui quali il segnale potrebbe avere interferenze dovute a scattering. È importante sottolineare fin da subito che, tramite 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 quindi periodicamente confrontata con quella misurata dal sistema UWB, la quale non rappresenta, in condizioni standard, un indice della posizione del veicolo. Solo nel momento in cui i due valori, restituiti da Lidar e UWB, differiscono di molto, allora l'ultima posa ottenuta dalle antenne viene assegnata al veicolo stesso come sua posa attuale. In questo modo si ottiene un sistema robusto alla perdita del lidar, che può verificarsi a seguito di una rottura del sensore 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 allo scopo di pre-acquisire e realizzare una mappa dell'ambiente nel quale il veicolo dovrà poi muoversi. Solo in un secondo momento, utilizzando questa mappa, avrà luogo la fase di moto, durante la quale il sistema riesce a localizzarsi nell'ambiente attraverso l'algoritmo Adaptive Monte Carlo Localization, nel quale ha luogo il confronto tra le misure acquisite dal lidar e le feature della mappa stessa.
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, posizionata all'interno di un box metallico.

Altro componente fondamentale è il convertitore usb-seriale "TTL-232R-PCB" prodotto da "Future Technology Devices International Ltd". Questo consente di avere una linea di comunicazione seriale con la STM, attraverso la quale trasmettere informazioni necessarie all'algoritmo di navigazione. Per conoscere quali informazioni vengono trasmesse, rifarsi alla sez. 5, mentre per sapere la funzione dei vari pin, si consiglia di consultare il datasheet o, più comodamente, di rifarsi alla fig. 3. Per migliorare la semplicità di utilizzo ed evitare di incorrere in errori legati alle comunicazioni seriali, è stato associato al dispositivo "TTL-232R-PCB" il symlink `ttyUSBserial`, in modo tale che l'ordine di inizializzazione delle porte usb della raspberry non influenzi in alcun modo il nome del collegamento (per esempio: `ttyUSB0` oppure `ttyUSB1`) Per conoscere i passaggi necessari alla realizzazione di tale symlink, si riporta la procedura dentro il file `/Info/renaming_ttyUSB.txt`, fornito insieme al codice.

L'algoritmo implementato all'interno dell'STM non è l'unico mezzo attraverso il quale far muovere il veicolo: grazie alla presenza di una piccola PCB, sulla quale è presente il

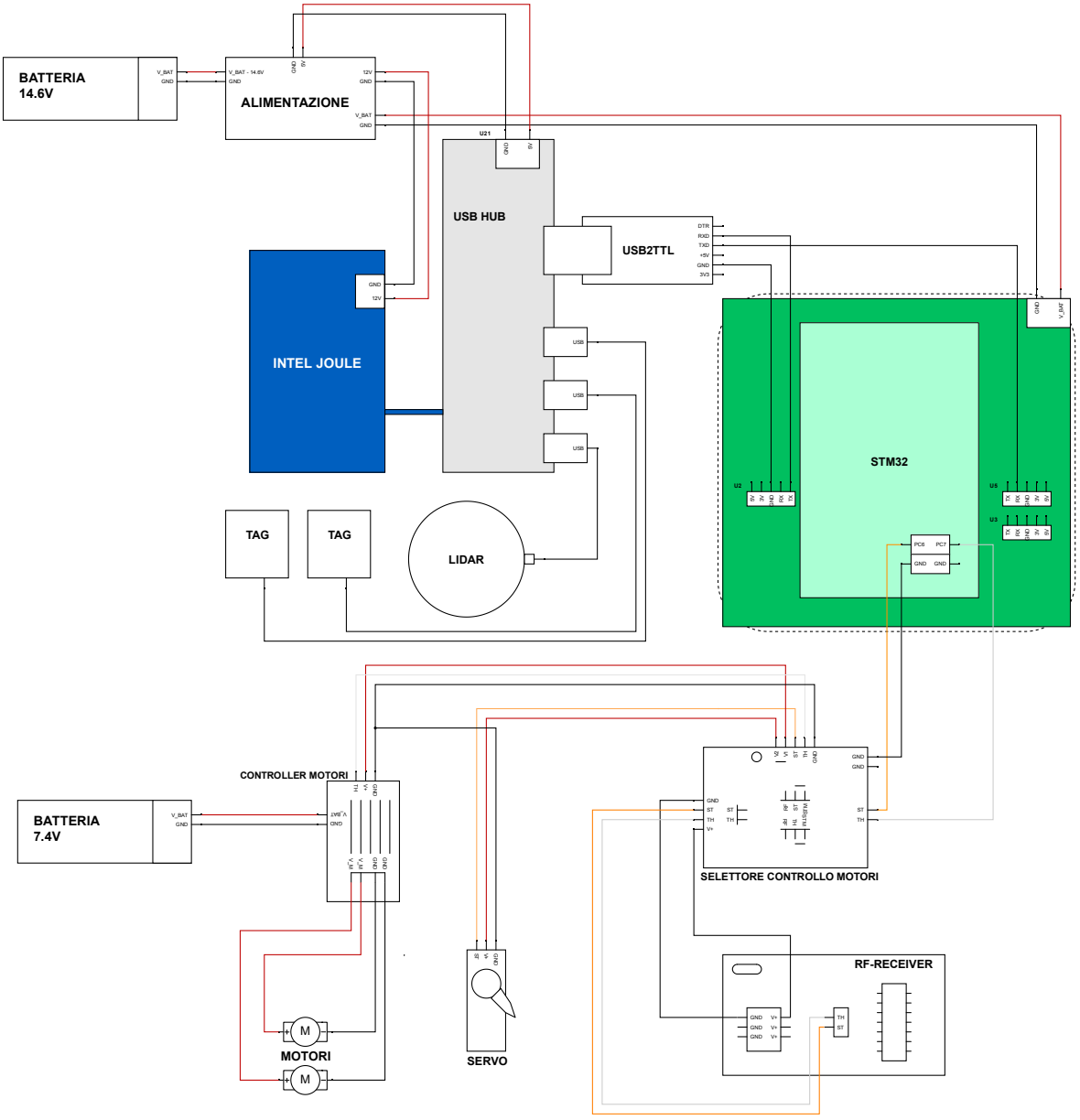

Figura 1: Schema connessioni completo. **DA EDITARE CON RASPBERRY**

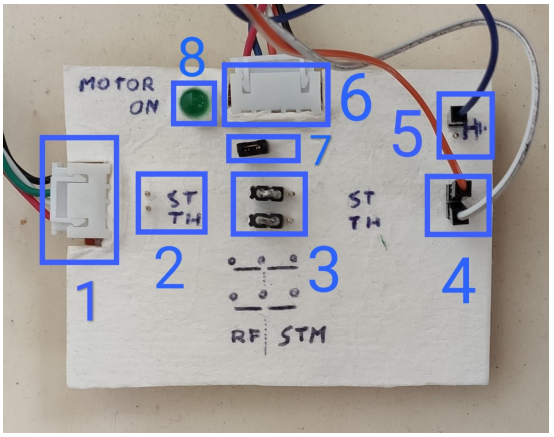

Figura 2: PCB controlli

Figura 3: Pinout del convertitore USB-seriale. **DA FARE!!!**

circuito adibito allo switch di sistema di controllo e al check dello stato dei motori, si ha la possibilità di utilizzare, alternativamente alla STM, un radiocomando per il controllo di sterzo e acceleratore. Questa PCB, visibile in fig. 2, è dotata infatti di un led di stato verde che notifica quando l'alimentazione dei motori è attiva; sono inoltre 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 approfondimenti sulle scelte che stanno dietro alle connessioni fatte, riferirsi a [DWP21]. Lo schema completo è mostrato in fig. 1.

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 di utilizzo, suggeriamo di collegare la raspberry ad un monitor tramite un cavo microHDMI-HDMI e una tastiera/mouse usb.

Tutto il necessario da sapere riguardo i passaggi per il primo collegamento a raspberry si trovano nel file: `Info/README_wifi_settings.txt`, dei quali riportiamo qui un breve riassunto. Come prima cosa è va impostato il WiFi: il file con le varie impostazioni WiFi, chiamato `interfaces`, si trova in `/etc/network/`. Per editarlo:

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

Questo file si occupa di leggere le configurazioni salvate nei file `.conf` di cui sono indicati i path al suo interno.

Per aggiungere una nuova configurazione, inserire 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 connessioni alle reti riportate più in alto nel file):

```
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, ovvero (`/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 configurata (in ordine di righe) che sia disponibile. Dato che è già in atto la connessione tramite lo schermo, che permette di visualizzare il contenuto di raspberry, suggeriamo come prima cosa di procedere con un backup della scheda originale: esiste un apposito tool all'interno di raspbian che consentirà di farlo agilmente. È buona norma creare il backup a monte dell'apporto di qualsiasi modifica o aggiunta, in quanto ciò consentirà di avere, in caso di necessità, un punto di ripristino funzionante.

1.2 Ros master/slave

Al fine di non sovraccaricare la raspberry e di avere un'interazione fluida con rviz e altri tool grafici di ros, è fortemente consigliato l'utilizzo di ros su altri computer in parallelo, delegando loro l'esecuzione dei task più costosi a livello computazionale (come appunto quelli grafici). Sono riportati di seguito i passaggi utilizzati da noi per impostare l'intero sistema, al fine di facilitare i nuovi utenti; per saperne di più consultare la guida ufficiale.

Dopo svariate prove, siamo giunti alla conclusione che, per arrivare ad avere una interfaccia per utente sufficientemente fluida ed efficace, quello che dovevamo fare era avere il RosMaster su raspberry e utilizzare invece il pc per i tool grafici, esempio rviz o rqt. Per farlo, dobbiamo ottenere l'ip dei nostri dispositivi: è possibile farlo attraverso vari comandi (come `ifconfig` o `ip address`) e generalmente risulterà essere 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
% export ROS_IP=IPrasp
```

A questo punto, è sufficiente riavviare le shell dei terminali che si vogliono utilizzare: sarà possibile in essi lanciare i nodi, installati su pc, direttamente dal pc, ma sfruttando il master su raspberry.

2 RPlidar

Il lidar utilizzato su Charlie è un RPlidar A3 prodotto da Slamtec, di cui 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. 4.
- cercare di far vibrare il meno possibile la struttura di sostegno e rialzo del lidar.

Per evitare, di nuovo, che l'ordine di inizializzazione delle porte usb della raspberry possa influenzare il nome del collegamento (per esempio: `ttyUSB0` oppure `ttyUSB1`) abbiamo realizzato anche per questo dispositivo il symlink `ttyUSBlidar`:ciò fa sì che la porta seriale

Figura 4: Riferimenti **DA EDITARE**

del lidar sia sempre chiamata `ttyUSBlidar`. Per realizzare tale symlink, consultare la procedura riportata nel file `/Info/renaming_ttyUSB.txt`.

In ambiente ros viene eseguito il lidar andando a lanciare il nodo `rplidarNode`, il quale pubblica sul topic `/scan`. È stato sviluppato il file di lancio `rplidar_a3.launch` con i parametri opportuni per il nostro sistema. Per quanto riguarda l'utilizzo del lidar, non è stato necessario alcun sviluppo software in quanto il pacchetto ros risulta già completo.

3 Sistema Pozyx

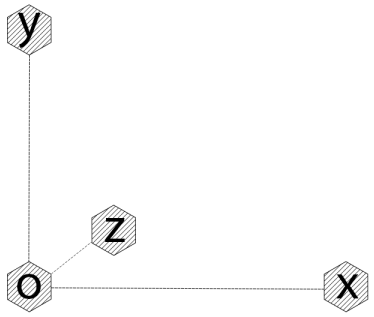

Figura 5: Disposizione ancore

Il Pozyx è un sistema di localizzazione alternativo al GPS, basato sulla tecnologia ultra-wideband (UWB). In breve, si tratta di una tecnica per la trasmissione e la ricezione di segnali sviluppata in modo tale che vengano sfruttati impulsi di energia a radiofrequenza aventi durata temporale molto ridotta (nanosecondi), con conseguente banda spettrale ampia. Nel sistema Pozyx si possono distinguere due entità, ciascuna con le proprie peculiarità: le ancore e le tag. La configurazione base prevede l'utilizzo 4 ancore, che fungono da dispositivi fissi, e una tag, che rappresenta il dispositivo mobile. Grazie all'utilizzo di un algoritmo di triangolazione, la tag si può localizzare rispetto alle ancore. Teoricamente, la localizzazione tramite 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 il 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 la possibilità di ricevere un buon segnale, in quanto permette di limitare 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 sarebbe molto grande e potrebbero verificarsi ambiguità di posizione legate alla simmetria della configurazione. Per un buon funzionamento del sistema, è infine importante disporre le ancore verticalmente e con l'antenna UWB rivolta verso l'alto.

Una volta predisposto l'ambiente di lavoro, si passa alla parte di comunicazione tra i vari agenti in gioco. 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 risulta essere fondamentale nel momento in cui vengono prelevate le posizioni relative delle ancore nella configurazione corrente. Con `remote_id=None`, chiediamo di fatto alla tag connessa in seriale ad Arduino di fornirci 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 propria 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 è stati scritto l'algoritmo in questione, l'ancora 0 è quella la cui posizione fornisce l'origine del frame UWB che verrà costruito. L'ancora 1 indica invece la direzione dell'asse y, ovvero le sue coordinate finali saranno `(0, y1, 0)`, così come l'ancora 2 quella dell'asse x. La quarta ancora, posizionata a una altezza diversa dalle altre 3 (che devono invece essece complanari), evidenzia infine 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.

Per quanto riguarda la componente software, sono stati utilizzati nodi custom basati sul modulo Python di PyPozyx, rispetto ai quali abbiamo apportato modifiche per adattarli al nostro sistema. Per maggiori dettagli, vedere nella cartella `...\Workspace\src\charlie_pozyx\src\sc` dove all'interno di ciascun nodo è presente una breve descrizione del suo funzionamento e della sua utilità.

Focalizzandoci su uno di questi nodi, ovvero il `pos_pub_2_tag.py`, descriviamo un errore che abbiamo riscontrato e che, plausibilmente, è stato una causa di errori nelle esecuzioni precedenti. Prima di tutto, questo nodo si occupa di pubblicare la posa delle due tag (posizione + quaternione) e sfrutta al suo interno la funzione, sempre custom, `doPos_pubtf`, all'interno della quale è fatto girare l'algoritmo `doPositioning`. Questo, a sua volta, ricava la posizione delle tag pozyx connesse in seriale. Come spiegato in [CT18], l'algoritmo `doPositioning` viene eseguito utilizzando la modalità `POZYX_RANGE_PROTOCOL_PRECISION` e questa richiede l'utilizzo di molti sample. Per questo motivo, dato che il `doPositioning` deve essere eseguito in modo sequenziale prima sulla tag0, poi sulla tag1, poi nuovamente sulla tag0 e così via, si rende necessario l'inserimento di una pausa tra una esecuzione e la successiva, per permettere all'algoritmo di terminare prima di essere lanciato nuovamente.

In assenza di tali pause temporali, abbiamo infatti notato che i risultati venivano corrotti: dopo l'aggiunta di tali intervalli, invece, le esecuzioni successive non si influenzano più a vicenda, come voluto.

4 Sistema Vicon

Il sistema Vicon è un sistema molto accurato di motion capture. In questo progetto è stato utilizzato per fornire un *ground-truth* e un confronto all'algoritmo di navigazione, del quale possiamo così valutare la bontà. In tal senso, quindi, abbiamo ritenuto opportuno sviluppare i nodi e i topic di dialogo con il sistema Vicon all'esterno del workspace della raspberry: infatti, questi si trovano su pc nel pacchetto in `charlie_remote`.

Per poter dialogare con il sistema Vicon, è sufficiente includere nel proprio `catkin_ws` il pacchetto `vrpn_client_ros`, disponibile sulla wiki.ros e scaricabile dalla repo GitHub. Questo pubblicherà la posa, il twist e l'accelerazione di ogni oggetto selezionato nel software di tracking (maggiori dettagli su quali oggetti selezionare nelle sezioni 5 e 7).

Creazione oggetto

Nonostante siano già presenti gli oggetti necessari per il tracking di Charlie, riportiamo la procedura di creazione di un nuovo oggetto, la quale potrebbe essere utile in caso di modifiche ai marker o in altre circostanze. Sinteticamente è necessario procedere come segue:

1. Disporre 4 o più marker sull'oggetto desiderato in modo asimmetrico. Per facilitare i passi successivi, è consigliato disporre alcuni marker sugli assi del sistema di riferimento body.

2. Selezionare i marker nell'applicazione "Vicon Tracker 3.7.0 x64" semplicemente clickando e tenendo premuto il tasto sinistro del mouse, poi premere pausa (pulsante collocato a sinistra).

3. spostarsi nel tab Objects, assegnare un nome in basso e premere Create Object. L'applicazione assegna un sistema locale di default.

4. Modificare il sistema locale premendo sugli assi o accedendo nelle proprietà dell'oggetto appena creato. Una volta soddisfatti, premere `Ctrl+S` per salvare l'oggetto.

5. Premere di nuovo il pulsante di pausa e verificare che il tracciamento stia funzionando.

Calibrazione

Per effettuare una calibrazione del sistema, necessaria di tanto in tanto affiché il motion capture non accumuli errori dovuti a cambiamenti delle condizioni dell'ambiente o a piccoli spostamenti delle camere, è necessario utilizzare la wand. La procedura richiesta si compone dei seguenti passi, riportati sempre in via sintetica. Per prima cosa, è necessario accedere all'applicazione "Vicon Tracker 3.7.0 x64" e collocarsi nella tab "Calibrate".

Premere poi su “Start” nella sezione “CALIBRATE CAMERAS”, avviando così la procedura di autocalibrazione: spostare la wand con pattern circolari all’interno della stanza assicurandosi di avere sempre più camere “a vista”. Il software interromperà da solo l’acquisizione dei dati una volta che questi risultino essere sufficienti.

Adesso è necessario definire il sistema di riferimento. Per fare questo, andiamo a posizionare l’incrocio degli assi della wand nel punto in cui vogliamo l’origine e gli assi stessi dell’oggetto in modo che siano allineati con gli assi-Vicon desiderati. Selezioniamo quindi il marker all’incrocio come origine e gli altri marker come asse x e asse y, completando così la calibrazione del sistema. Nella stanza del volo del DII, il sistema di riferimento più usato è rappresentato in fig.REF. **FARE IMMAGINEEE**

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

5.1 Sistema di riferimento body

FARE FIGURINAAAAAAAAAAAAAAAA

6 Prove?

Per quanto riguarda l'impostazione di prove pensate per una verifica del buon funzionamento delle procedure di fix di posa e dell'algoritmo di navigazione, anche su più stanze, dato che non sono state apportate modifiche alla loro struttura durante il nostro lavoro, rimandiamo ai risultati riportati nella relazione precedente, ovvero [DWP21]. Nel nostro caso, abbiamo deciso di orientarci più verso due obiettivi specifici: il primo è stato quello di effettuare una analisi più dettagliata degli errori compiuti dal sistema durante la navigazione con UWB, Lidar e AMCL. Per farlo, abbiamo utilizzato come ground-truth il sistema Vicon, disponibile però solo nella stanza del volo e, per questo motivo, di limitata ripetibilità pratica. Abbiamo deciso comunque di riportarne almeno i risultati in modo da fornire un riferimento affidabile di quelli che sono gli errori compiuti, per i quali ci aspettiamo una somiglianza in tutte le configurazioni simili a quella ricreabile nella stanza del volo stessa (ambiente chiuso, presenza di finestre, estensione non troppo elevata, assenza di ostacoli all'interno dell'area di moto). Come secondo obiettivo, viste le modifiche effettuate sul sistema UWB per quanto concerne la procedura di salvataggio delle pose delle ancore a fine autocalibrazione e il loro successivo utilizzo da parte della tag connessa in seriale durante la navigazione di Charlie, abbiamo predisposto due prove in esterno: qua il sistema UWB si trova in un ambiente favorevole rispetto a quanto succede quando si lavora indoor. In particolar modo, come prima prova in esterno andremo a testarne il funzionamento in un cortile interno al polo A di Ingegneria, dove ci aspettiamo diversi disturbi per le UWB, legati alla presenza di molti oggetti e strutture metalliche e di finestre poste in line of sight con le ancore stesse. Il secondo scenario scelto, sempre nei pressi del polo A di Ingegneria, è un grande prato esterno, dove il sistema UWB dovrebbe essere meno disturbato, nel quale poter valutare il raggio di azione di questo sistema e, di controparte, la difficoltà che può riscontrare il lidar quando si trova in assenza di riferimenti fissi, come pareti, per l'acquisizione dei suoi dati.

6.1 Confronto Vicon

La prima prova, come accennato, è stata eseguita all'interno della stanza del volo del polo A, in quanto questa è munita di un sistema Vicon funzionante che permette di effettuare un motion capture estremamente preciso e affidabile degli oggetti contrassegnati con gli

appositi marker. Come prima cosa, è necessario posizionare i marker sul veicolo in modo asimmetrico. Nel nostro caso sono stati posizionati come in fig. (FAI RIF), dove vediamo 4 marker: uno nella parte frontale e centrato, due laterali a circa metà della struttura e un quarto nella parte posteriore, spostato verso destra per ottenere l'asimmetria necessaria ad evitare ambiguità. Il sistema Vicon riesce con essi a costruire un oggetto virtuale corrispondente a Charlie senza ambiguità di orientazione. Una volta costruito tale oggetto, per vedere come riferirsi al 4, si può procedere con l'esperimento vero e proprio. In questo caso, per i comandi di spostamento è stato utilizzato il radiocomando, così da avere maggiore manovrabilità. L'esperimento di per sé è molto semplice: una volta avviato l'esperimento standard, come descritto in 7, e il sistema di acquisizione dati del Vicon, vedere 4, e aver aggiunto i topic relativi al Vicon nella lista dei topic da salvare nella bag, si procede semplicemente spostando Charlie nella stanza, raccogliendo così le informazioni in due modi distinti, uno (il Vicon) considerato corretto e, quindi, privo di errori e l'altro, quello implementato nel sistema per la navigazione, affetto da vari errori, che verranno appunto evidenziati attraverso un successivo confronto. Si procede, infatti, all'analisi dei dati raccolti attraverso Matlab, dove in particolar modo si vanno ad evidenziare le posizioni e le orientazioni associate a Charlie dal Vicon, le medesime ricavate dal sistema integrato sul veicolo e, di particolare interesse nel nostro caso, gli errori che quest'ultimo metodo di localizzazione presenta rispetto al Vicon. Interessante andare a distinguere le varie fasi di moto di Charlie, ovvero i punti in cui sta fermo, quelli in cui procede in linea retta e quelli dove, invece, compie traiettorie curvilinee, ed associare ad essi i pattern seguiti dai vari errori: come ci aspettiamo, la fase in cui l'errore si riduce maggiormente coincide con quella in cui Charlie sta fermo (da `50 s` a `60 s`).

valutazione dei risultati di questa prima prova con i vari grafici

Dai risultati ottenuti, abbiamo riscontrato un'errore sbilanciato verso la parte negativa, che aumenta soprattutto nelle fasi di moto. Per valutare se la colpa sia dell'autocalibrazione e del modo in cui sono state posizionate le ancore nell'ambiente, abbiamo deciso di ripetere un secondo esperimento uguale al precedente, ma posizionando le ancore con una differenza di altezza maggiore, che ci consentisse di distanziare maggiormente la quarta ancora dall'origine senza che ciò generi errori nella creazione del frame UWB.

FINIREEEEEEE E ANALIZZARE I GRAFICI E RIPORTARLI

6.2 Esperimento nel cortile

6.3 Esperimento all'aperto

7 Guida breve all'esperimento

In questa sezione è riportata, sommariamente, la procedura da eseguire per lanciare un esperimento completo. Per una guida dettagliata, riferirsi a sez. 5.

Prima di tutto, è necessario disporre le ancore come descritto in sez. 3, come si può vedere in fig. 5. 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 poi in **ssh** alla Raspberry dal PC digitando, da terminale del PC e con password `robot`, il seguente comando:

```
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 (necessario per avere informazioni corrette sulle loro posizioni, vedere sez. 3).

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, per averne un maggiore controllo, soprattutto in termini di limitarne la velocità di avanzamento, che in questa fase di acquisizione della mappa è bene che sia molto bassa.

```
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 all'interno di una cartella dedicata, detta qua `maps` attraverso i comandi:

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

Localizzazione

Prima di procedere, è necessario sostituire il nome della mappa che si vuole utilizzare all'interno del 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, si può adesso lanciare il file appena modificato con il comando standard per eseguire file `.launch`:

```
roslaunch charlie_launch localization.launch
```

All'interno dello stesso file è predisposta anche la possibilità di visualizzare rviz attraverso la connessione ssh con il "Compressed X11 Forwarding ", scommentando la riga corrispondente di rviz. Questa opzione è però 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 direttamente dal terminale del pc il comando:

```
roslaunch charlie_remote rviz_remote.launch
```

Così da aprire rviz da pc, ma sfruttando il master su raspberry. 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"
```

Si consiglia di farlo una volta dato il comando del waypoint da raggiungere con rviz. Per fermarli, similmente, è sufficiente pubblicare il messaggio complementare, sempre sul topic `start\_and\_stop`:

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

Sistema Vicon

Ovviamente, per poter utilizzare il sistema Vicon è necessario trovarsi nella stanza del volo del DII. Per quanto concerne 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)`. Connettere poi il computer fisso, a sua volta attaccato fisicamente al sistema Vicon (che sarà il server Vicon), alla stessa rete in cui abbiamo già raspberry e pc. In seguito, modificare nel file di lancio `charlie_remote/launch/vicon_charlie.launch` l'ip del server Vicon nella seguente riga:

```
6     <arg name="server" default="IP_SERVER"/>
```

Per avviare i dialoghi tra ros e il sistema Vicon, eseguire da pc il file di lancio `vicon_charlie.launch`con il comando:

```
roslaunch charlie_remote vicon_charlie.launch
```

Salvare poi la trasformazione tra vicon e uwb (all'utente sarà richiesto di posizionare la wand, in successione, sulle verie 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, in quanto verrebbero salvati molti elementi non necessari al nostro esperimento. Si consiglia quindi di lanciare il seguente comando da pc (dopo essersi spostati nella cartella desiderata nella quale salvare i file `.bag`), contenente la selezione da noi fatta in merito ai topic ai quali sottoscriversi per ottenere tutti e soli i dati di nostro interesse:

```
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 ad 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 ancore Pozyx

L'autocalibrazione si basa sullo script `python3` "autocalibration_ransac.py". Affinché possa 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 innanzitutto utilizzare un metro col quale misurare la distanza relativa tra le coppie di ancore della rete. Queste misure andranno poi inserite manualmente nelle opportune variabili `r01`, `r02`, `r03`, `r12`, `r13` ed `r23`, che rappresentano per l'appunto le distanze tra le rispettive antenne. Per quanto riguarda invece la calibrazione automatica, lo script prevede una prima fase nella quale si ha l'acquisizione dei dati necessari, seguita dall'esecuzione l'algoritmo ransac. Infine, viene utilizzato l'algoritmo algebrico per determinare le coordinate effettive delle ancore. L'algoritmo ransac permette di rimuovere 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:

- Le coordinate dei dispositivi all'accensione del sistema, prima che sia effettuata la nuova calibrazione.
- Il risultato del settaggio dei parametri UWB della rete.
- Il risultato dell'algoritmo ransac per la distanza tra le ancore senza outliers.
- Il risultato dell'algoritmo algebrico per determinare le coordinate delle ancore: in caso di fallimento dell'algoritmo, è mostrato a 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 potremmo dire che 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, con queste informazioni, una propria lista interna dei dispositivi della rete.
Per maggiori approfondimenti, 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).