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

hector fa mappa scan matcher acquisisce una posa amcl confronta tale posa e il cloud di punti del lidar con la mappa nota uwb resetta nel caso di scazzi

Come ottenere codice

google drive account link github e mini organizzazione
Invece per ottenere l'hardware non scriveteci, chiedete a Lorenzo Pollini metti mail

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.
FARE IMMAGINE CONCETTUALE
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 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

Per connettere e alimentare quanto qui sopra è stato installato:

- una custom pcb
- USB-HUB alimentato, che ci permette di utilizzare ulteriori porte usb senza far affidamento al Raspberry Pi per la loro alimentazione (che risulta inefficace per alimentare il lidar)

schede (PRIMO COLLEGAMENTO A RASPBERRY)

1.1 Primo collegamento a Raspberry

1.2 Ros master/slave

suggerimenti, ssh e rviz master/slave
descrizione alimentazione disegnino dei collegamenti batterie (come si ricaricano per dummies)

2 RPlidar

sensore lidar orientazione (asse zero insomma) del lidar
se voltaggio sotto 4.7V rischia di non funzionare bene
seriale slamtec "delicata" va stoppata per bene
pacchetto ros rplidarros
viene lanciato da rplidar_a3.launch
topic `/scan` letto da hector_slam produce mappa e da scan_tools produce la tf da laser odom a base_link

pacchetto ros già fornito da slamtec

3 Sistema Pozyx

comportamento fisico 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

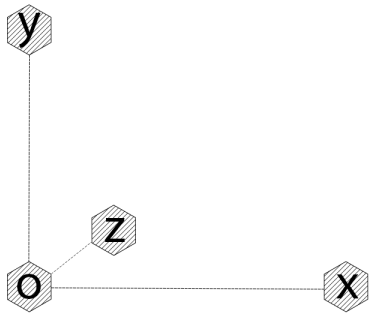

Figura 1: Disposizione ancore

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
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. 1. Assicurarsi di aver impostato correttamente le impostazioni WiFi (sez. 1.1) e di avere connessi alla stessa rete pc e Raspberry.

Connettersi **ssh** 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 **ABBIAMO FATTO IL FILE .SH DA CAMBIARE**:

```
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`:

VERIFICARE

```
11  <node name="map_server" pkg="map_server" type="map_server" args="/home
        /pi/charlie_ws/maps/NOME_MAPPA.yaml"/>
```

Quindi lanciare:

```
roslaunch charlie_launch localization.launch
```

Nello stesso file è possibile impostare di visualizzare rviz attraverso 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 di tracking **SCRIVERE NOME PER BENE** e selezionare nella lista oggetti: `Charlie` e `WAND NOME PER BENE`.

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 (richiede il posizionamento della wand 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 sottroiscriversi a tutti i topic ma quindi 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_bash.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:

```
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: se tale algoritmo fallisca, è mostrato su terminale l'errore;