



UNIVERSITA' DEGLI STUDI DI PADOVA
Dipartimento di Ingegneria dell'Informazione

Corso di Laurea in Ingegneria Informatica

Relazione per la prova finale

Implementazione algoritmo di Path Planning su NVIDIA Jetson Orin tramite simulazione remota

Relatore: Prof. Damiano Varagnolo

Laureando: Stefano Piccolo
Matricola n.: 2073988

ANNO ACCADEMICO 2025/2026



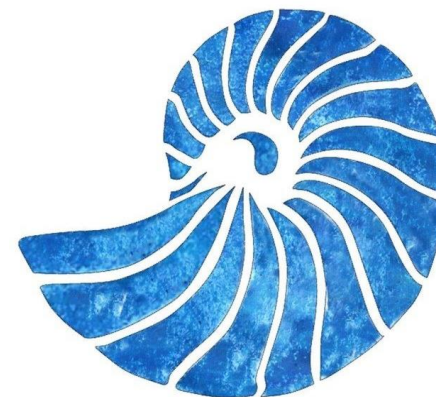
Introduzione

Gruppo di ricerca

Questo progetto è stato svolto insieme al gruppo studentesco **Nautilus** presso l'Università degli Studi di Padova. Il gruppo si occupa principalmente di sistemi autonomi e intelligenti utilizzati in ambienti marini, nello specifico di veicoli/droni **USV (Unmanned Surface Vehicle)** e alla loro integrazione con tecnologie embedded ad alte prestazioni.

Le ricerche e i progetti hanno come finalità:

- **campionamento** di parametri fisici/chimici dell'acqua
- **cattura di immagini e video** subacquei e in superficie
- **monitoraggio** dell'ambiente marino in generale



In particolare, il progetto si concentra sul veicolo **Blueboat**, un drone dotato di due scafi paralleli fissati da due semiasse, come in figura.

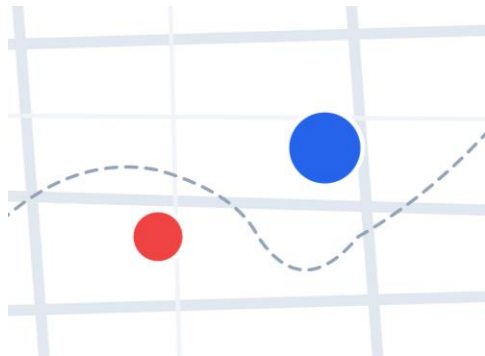


Obbiettivo del progetto

Lo scopo dei droni USV è quello di **muoversi liberamente** in un ambiente marino e **analizzarlo in modo efficace**, superando gli ostacoli che potrebbero rallentarli (o frenarli completamente) e impedirli di raccogliere i dati adeguatamente.

Esistono diversi tipi di fattori che possono fare questo:

- **ostacoli statici:** scogli, boe di segnalazione, imbarcazioni, piattaforme petrolifere, reti da pesca, fauna marina...
- **ostacoli dinamici:** correnti marine intense, venti forti e condizioni meteorologiche avverse...



Conoscendo questi fattori:

l'obbiettivo del progetto è quello di implementare un **algoritmo** in grado di raggiungere una destinazione precisa evitando gli ostacoli durante il percorso, e simularlo in un ambiente virtuale.





Strumenti utilizzati

La simulazione viene elaborata su un dispositivo e visualizzata graficamente su un altro per alleggerire il carico di lavoro sull'hardware, risparmiando fino al **25% di utilizzo della CPU**.

Per farlo sono state usate le seguenti tecnologie Hardware/Software:



NVIDIA Jetson Orin Nano

Un sistema integrato progettato per applicazioni di IA e robotica che richiedono un basso consumo energetico, dotato di GPU con **1024 CUDA core***, CPU con 6 core ARM con prestazioni massime fino a **67 TFLOPS**, **8 GB di RAM LPDDR5**.

PC/Laptop

Un computer con sistema operativo Ubuntu 22.04 LTS dotato di una GPU con supporto al driver grafico OpenGL 3.3 o superiore, necessario per la visualizzazione della simulazione.



*Compute Unified Device Architecture, core più leggeri adatti per numerose operazioni parallele



Strumenti utilizzati



ROS2 Humble (Robot Operating System)

Un framework open source scritto in **Python** e **C++** che fornisce librerie e strumenti per gestire comunicazione e azioni tra i vari componenti di un Robot (in questo caso di un drone).

Ros2 si basa sul meccanismo di gestione dei **topic**, usando un'architettura di comunicazione tra nodi **publisher/subscriber**:

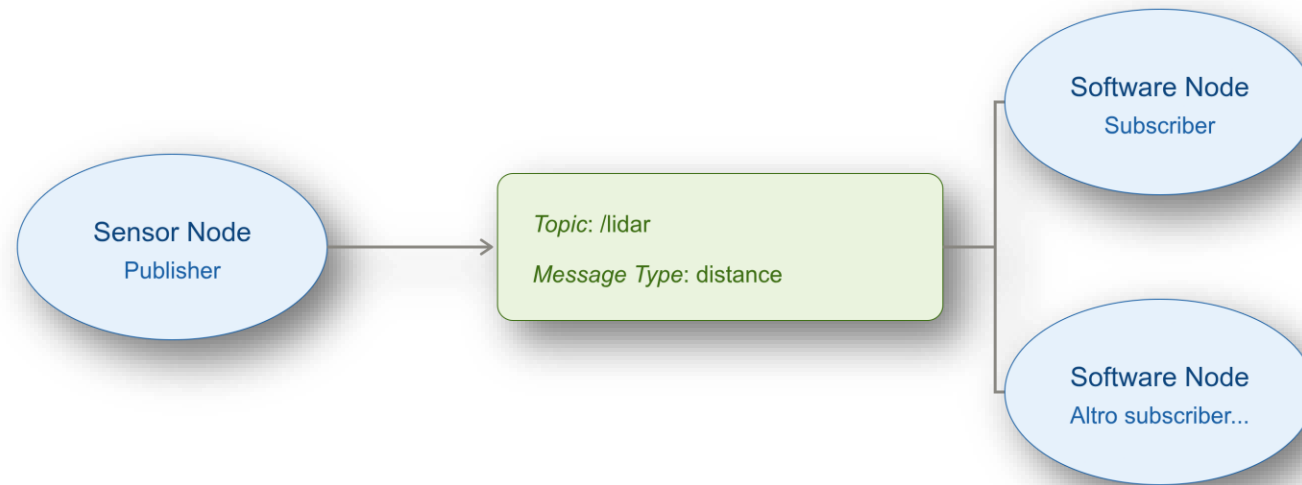
- **Topic**: il canale di comunicazione tipizzato dove passano esclusivamente messaggi di un tipo specifico, sono identificati da un nome (es. "/camera/image" è il canale usato per passare le immagini della videocamera);
- **Publisher**: un nodo che pubblica i messaggi su un topic senza conoscere chi lo riceverà;
- **Subscriber**: un nodo che si "iscrive" a un topic per ricevere i messaggi pubblicati..



Strumenti utilizzati

Funzionamento di ROS

Il drone è dotato di un sensore lidar per il rilevamento degli ostacoli, e di un programma che gestisce l'algoritmo di Path-Planning per calcolare la prossima manovra



- **il sensore**, tramite un nodo publisher, pubblica sul topic "/lidar" i messaggi con la posizione aggiornata di eventuali ostacoli;
- **il programma**, tramite un nodo subscriber, si mette in ascolto sul topic ed elaborare il messaggio.



Strumenti utilizzati



Gazebo Harmonic

Software di simulazione robotica open source per simulare in modo realistico robot, sensori e ambienti fisici. Dotato di un motore fisico per calcolare la "fisica" del robot (collisioni, gravità, attriti) e un sistema di **rendering avanzato**.

Consente di integrare diversi sensori come **telecamere**, **GPS**, **lidar**, **trasduttori di forza e telecamere termiche** oltre alla possibilità di modificare l'ambiente virtuale, modificando un file in formato .xml del progetto (immagine a destra), oppure con il sistema drag-and-drop dell'interfaccia grafica.



```
<model name="obstacle_1">
  <static>true</static>
  <pose>9 7 0 0 0 0</pose>
  <link name="link">
    <collision name="collision">
      <geometry>
        <cylinder>
          <radius>1.5</radius>
          <length>3.0</length>
        </cylinder>
      </geometry>
    </collision>
  </link>
</model>
```



Struttura della simulazione

Collegamento Jetson/PC

Il collegamento è gestito dalla libreria di comunicazione **gz_transport**, che in una simulazione normale (senza dispositivi esterni) invia in tempo reale i modelli di Gazebo sulla porta **localhost**.



Nel nostro caso, utilizziamo **cycloneDDS**, un programma che permette di trovare i nodi ROS esterni presenti nella rete LAN. I requisiti del collegamento sono:

```
<GzTransport>
  <Unicast>
    <HostIp>JETSON_IP</HostIp>
    <Port>11811</Port>
  </Unicast>
  <Unicast>
    <HostIp>YOUR_IP</HostIp>
    <Port>11811</Port>
  </Unicast>
</GzTransport>
```

- entrambi i dispositivi devono essere sulla **stessa rete locale**
- devo conoscere gli **indirizzi ip** e le **interfacce di rete** dei dispositivi
- impostare una **porta di ascolto** comune

Tutti i requisiti sono definiti nei rispettivi file di configurazione.



Struttura della simulazione

Ambiente Virtuale e movimento

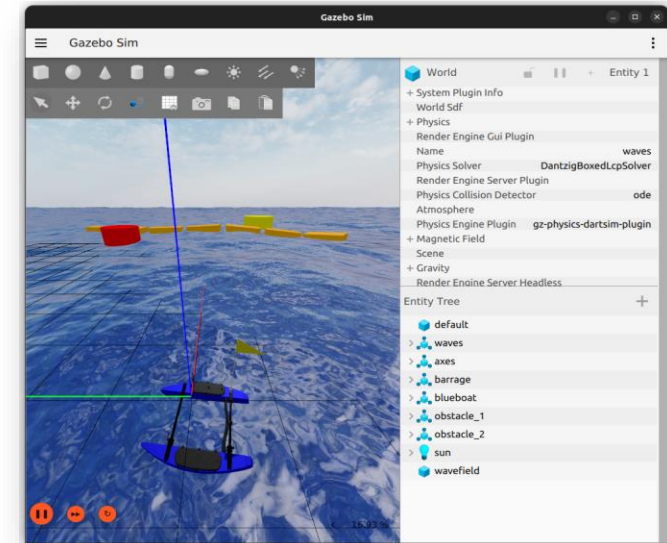
Modelli

L'ambiente virtuale è interamente descritto da un file **sdf (Simulation Description Format)** chiamato waves.sdf, che definisce i modelli di:

- Blueboat e i rispettivi sensori (lidar, GPS, accelerometro)
- onde e acqua
- ostacoli (boe, barriere artificiali)

Plugin comandi

Per il controllo della barca si usa **ArduPilot**, un plugin open source direttamente compatibile con Ros e Gazebo che, tramite dei comandi inseriti da un terminale e inviati su un porta di rete, permette di impostare le coordinate di una destinazione.





Calcolo traiettoria Path-Planning

Il programma di Path-Planning esegue 3 blocchi principali, eseguiti con una frequenza di 10Hz:

- lettura sensoriale del lidar
- calcolo geometrico della direzione verso il goal
- invio comandi di movimento ad ArduPilot

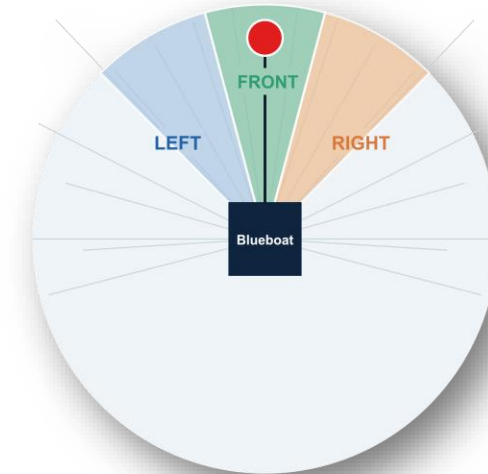
Il range del lidar è diviso in 3 spicchi principali **left**, **right** e **front** con un raggio massimo di 30 metri e con un angolo di 30 gradi.

Algoritmo

Quando uno ostacolo viene individuato si applica uno sterzo proporzionato allo spicchio in cui si trova e si regola la velocità .

Left/Right: sterzo di 15 gradi nella direzione opposta

Front: sterzo di 25 gradi nel lato con più spazio



$$h_{new} = h_{corrente} \pm 25^\circ \text{ (in radianti)}$$

$$v_x = V_{avoid} \cdot \cos(h_{new})$$

$$v_y = V_{avoid} \cdot \sin(h_{new})$$



Calcolo traiettoria Path-Planning

E se dopo una manovra vado in una direzione diversa rispetto al goal, oppure incontro un' ostacolo non rilevato?

Per continuare a seguire il goal dopo ogni manovra vengono utilizzate le coordinate **GPS** (latitudine, longitudine), inserite tramite la console di ArduPilot all'inizio della simulazione.

L'algoritmo calcola quindi la differenza tra le sue **coordinate attuali** e quelle del **goal** e calcola l'angolo di manovra normalizzato in radianti

$$\begin{aligned}\Delta lat &= lat_{goal} - lat_{now} \\ \Delta lon &= lon_{goal} - lon_{now} \\ \beta &= atan2(\Delta lon, \Delta lat) \rightarrow \text{normalizzato in } [0^\circ, 360^\circ]\end{aligned}$$

atan2 è una variante della funzione arcotangente che distingue i punti del piano su tutti i 4 quadranti, restituendo un valore dell'intervallo $(-\pi, \pi]$.

Nel caso il drone dovesse scontrarsi con un' ostacolo l'algoritmo segue le specifiche della tabella.

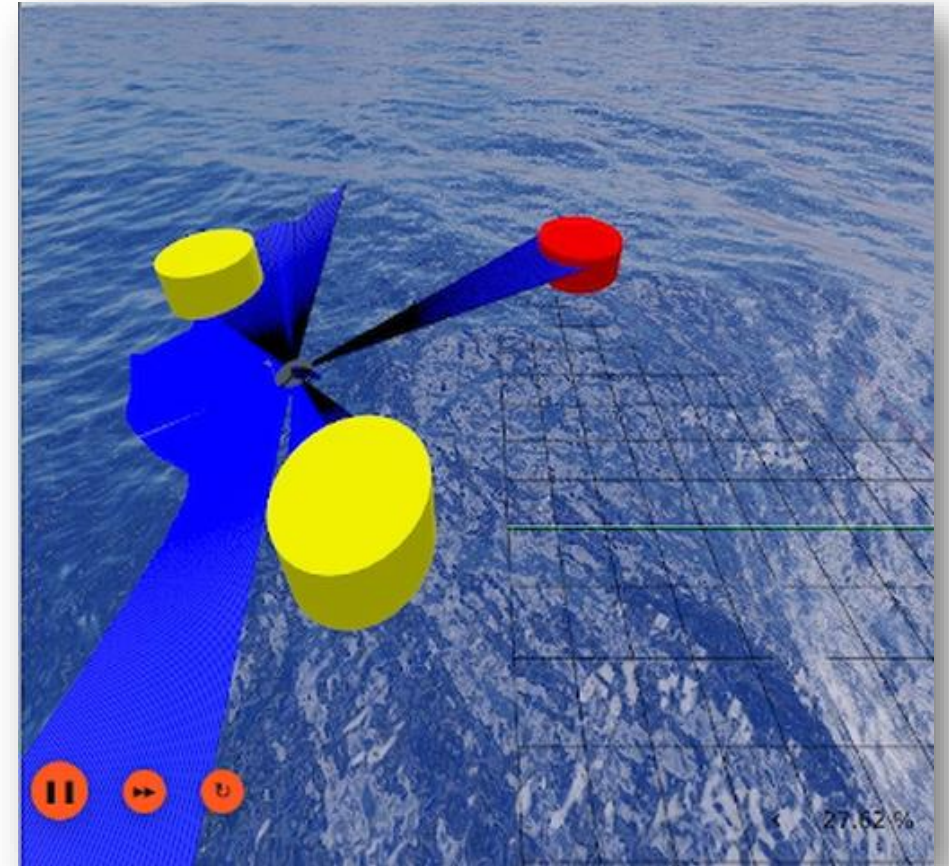
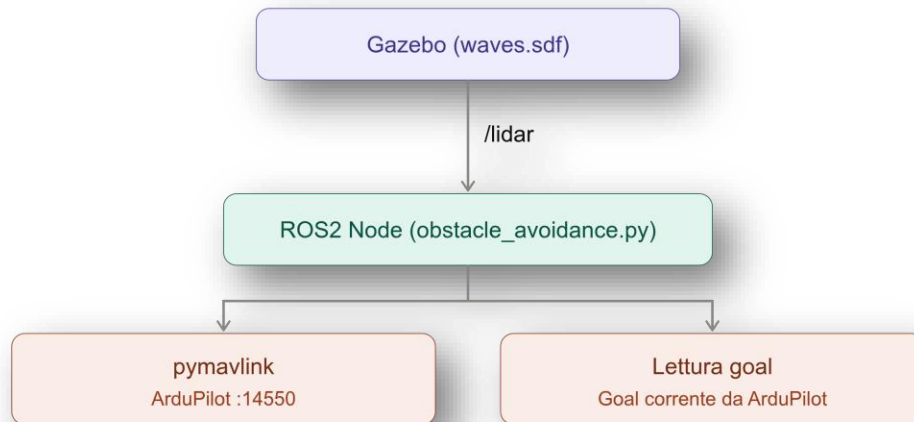
Priorità	Condizione	Azione	Velocità
1 — GOAL	$d_{goal} < 2 \text{ m}$	Stop, reset goal	0 m/s
2 — EMERGENZA	$d_{front} < 1.5 \text{ m}$	Stop completo $v_x=v_y=0$	0 m/s
3 — EVASIONE	$d_{front} < 4.0 \text{ m}$	Sterza $\pm 25^\circ$	0.5 m/s
4 — CROCIERA	nessuna condizione	Punta al goal	1.0 m/s



Simulazione finale

Nella simulazione finale, una volta inserite le coordinate di destinazione, si vedrà la BlueBoat muoversi tra gli ostacoli che rileva tramite i raggi del lidar fino a che non avrà raggiunto il goal.

La **struttura finale**, composta dai componenti descritti, è definita dallo schema sottostante.





Considerazioni Finali

Anche se l'obiettivo del progetto è stato raggiunto, il progetto può essere migliorato aggiungendo delle caratteristiche nuove o migliorando quelle già presenti, ad esempio:

- l'algoritmo implementato è un **Path-Planning locale**, che non memorizza il percorso fatto. Aggiungere un sistema di tracciamento del percorso aumenterebbe l'efficienza
- Nel caso si incontri un corridoio stretto l'algoritmo rischia di attuare come unica soluzione un dietrofront, anche quando questo non è necessario

