



Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in
Informatica

ELABORATO FINALE

TITOLO

*Avalanche Disaster Inspection Tools:
Pianificazione Automatica Distribuita delle missioni di una flotta di
droni e robot mobili*

Supervisore
Palopoli Luigi

Laureando
Ruaro Alberto

Anno accademico 2021/2022

Ringraziamenti

Mi sento in dovere di dedicare questa pagina del presente elaborato alle persone che mi hanno supportato nella redazione dello stesso.

Innanzitutto, ringrazio il mio relatore Palopoli Luigi, sempre pronto a darmi le giuste indicazioni in ogni fase della realizzazione del progetto. Grazie a lui sono stato in grado di accrescere le mie conoscenze, le mie competenze e ho avuto la possibilità di acquisire un metodo di lavoro che sicuramente mi aiuterà in futuro.

Ringrazio la mia famiglia, i genitori Manuela e Patrizio, perché senza di loro non avrei mai potuto intraprendere questo percorso di studi e da sempre mi sostengono nella realizzazione dei miei progetti. Ringrazio i fratelli Sebastian e Davide che mi hanno sempre supportato durante tutte le sfide che ho dovuto fronteggiare per arrivare fino a qui.

Vorrei dire grazie anche a tutte le altre persone che sono sempre state presenti nella mia vita: comprendo zii e cugini, la mia fidanzata Giulia, con cui ho condiviso i momenti di gioia e difficoltà di questo cammino e di certo non mi posso dimenticare di tutti gli amici.

Infine, un grazie speciale ai colleghi Manuel e Nicola, con i quali ho condiviso questo progetto, per essere stati sempre molto collaborativi e per avermi dato i migliori suggerimenti nella realizzazione dell'elaborato.

INDICE

1 Sommario	5
2 Introduzione	6
3 Problema	8
3.1 Panoramica	10
4 Background	12
4.1 ROS2 Foxy su WSL Linux	12
4.2 Git & GitHub: Cooperation e Versioning	13
4.3 Visual Studio Code: editor per C++ e Python	14
5 Soluzione	15
5.1 Architettura	15
5.2 Specifiche sulla procedura	16
5.3 Modulo 1: Algoritmo K-Means	18
5.3.1 Generazione dei punti	18
5.3.2 Assegnazione punti al cluster	19
5.3.3 Calcolo dei nuovi centroidi e spostamento	20
5.4 Modulo 2: Algoritmo Voronoi	20
5.4.1 Conoscenza vicini	22
5.4.2 Regione di Voronoi	22
5.4.3 Calcolo del centroide	23
5.5 Modulo 3: Servizio ROS2	24
5.5.1 Nodi ROS2	24
5.5.2 Chiamata ad algoritmo di divisione aree	25
5.5.3 Calcolo altitudine	26
5.5.4 Traduzione punti	26
6 Analisi sperimentale e validazione	27
6.1 Parametri	27
6.2 Implementazione con le altre parti	28
6.3 Pro e contro tra Pianificazione Centralizzata e Distribuita	28

7 Conclusione e possibili future implementazioni	29
7.1 Collaborazione sistemi centralizzato e distribuito	29
7.2 Ulteriori strade	30
Bibliografia	31

1 Sommario

In questa tesi è esposto il lavoro svolto come tirocinio interno realizzato assieme ad altri due studenti e il professore Luigi Palopoli.

L'obiettivo è quello di monitorare e controllare la presenza di eventuali vittime dopo l'episodio di una valanga. Per fare ciò utilizziamo un insieme di agenti (con agenti si intendono robot terrestri e droni). Si è scelto di utilizzare questi due strumenti per massimizzare l'efficienza in quanto: i robot terrestri hanno una maggiore autonomia, ma può riscontrare difficoltà se il territorio è scosceso, mentre i droni, pur avendo un'autonomia minore, sono molto più rapidi, in quanto riescono a raggiungere la zona desiderata in meno tempo rispetto ai robot terrestri; essi sono anche più economici e riescono ad avere un campo visivo molto vasto.

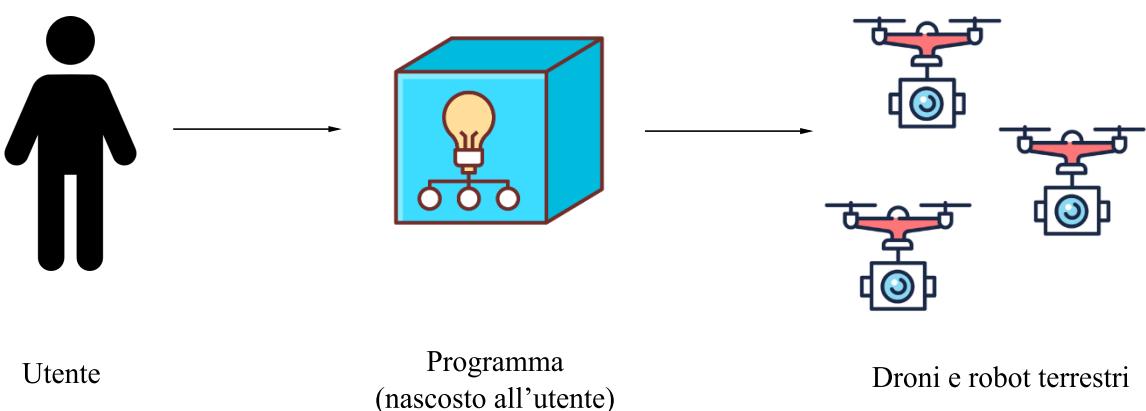
Essendo il territorio interessato dal progetto di montagna, i droni sono la scelta migliore in quanto risultano più efficaci date le loro notevoli prestazioni nei terreni poco pianeggianti.

Riassumendo, una volta che si conosce la posizione in cui è avvenuta la calamità, gli agenti coopereranno tra di loro per raggiungere il luogo il più velocemente possibile, calcolando la strada migliore e utilizzando le proprie risorse al meglio.

Più specificatamente, in queste pagine viene spiegata la parte del progetto che riguarda la **Pianificazione Automatica Distribuita di una flotta di droni e robot mobili**.

La pianificazione è automatica in quanto, sia droni sia robot mobili, sono comandati dal programma. In altre parole non si ha un controllo manuale degli agenti. L'utente ha il solo compito di inserire il luogo interessato, non preoccupandosi delle decisioni che verranno prese dai singoli dispositivi.

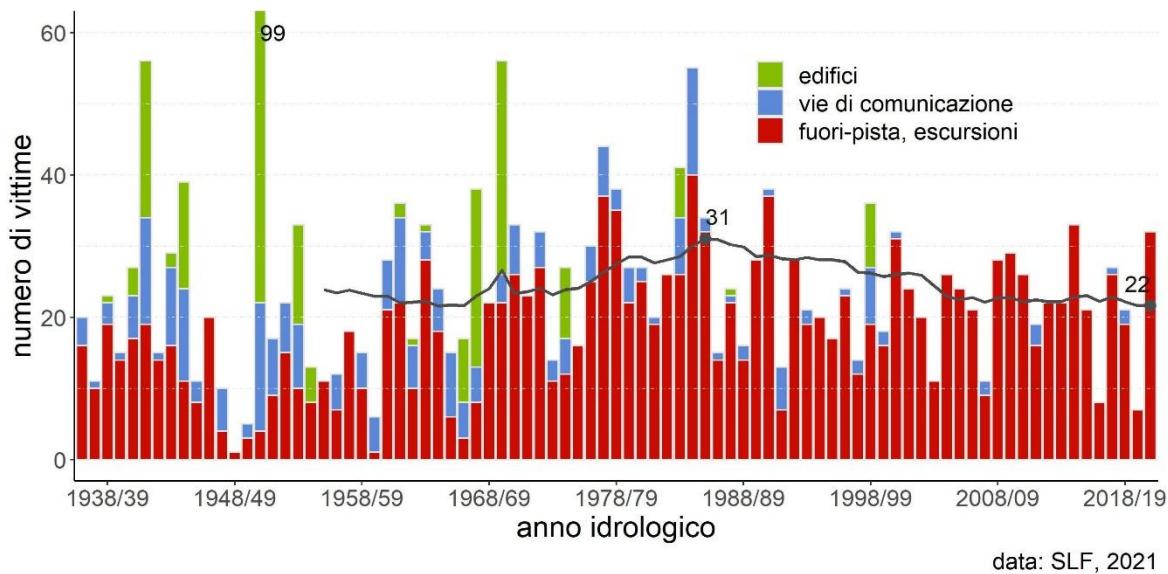
Inoltre, la pianificazione è distribuita: ogni drone, pur essendo in comunicazione con gli altri, prende delle decisioni in maniera del tutto autonoma, andando quindi ad eliminare il bisogno di un dispositivo centrale che coordina tutta l'operazione.



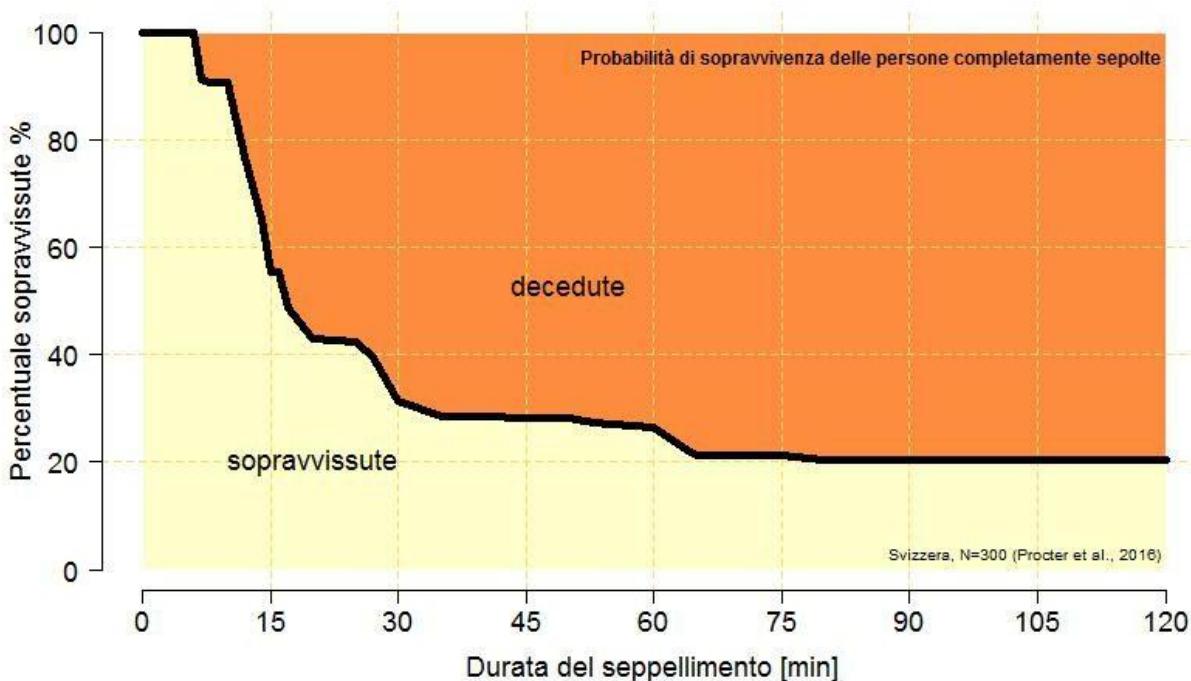
2 Introduzione

Facendo delle ricerche si può notare come le valanghe siano una preoccupazione costante. Esse spesso non sono prevedibili e basta un istante soltanto per far sì che si scateni un disastro di vasta portata. Il numero medio di vittime è più di 20 ogni anno, senza considerare i feriti e i dispersi.

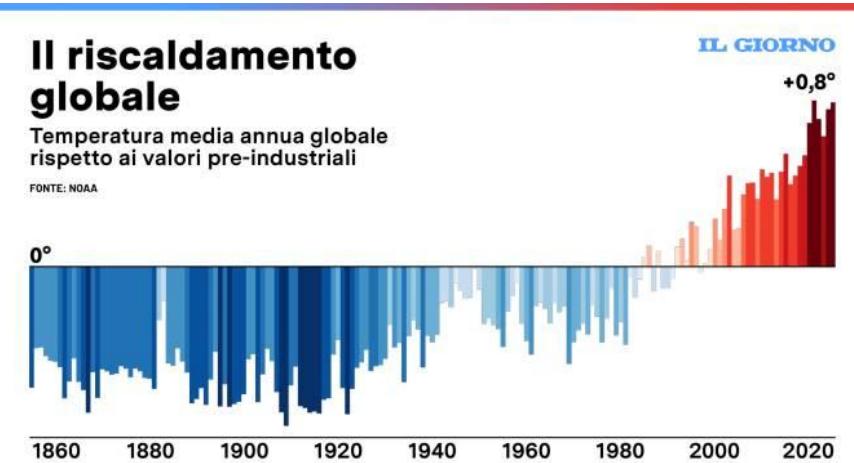
Il grafico sottostante mostra le vittime di valanghe solamente in Svizzera dal 1936/37.



Concentrandoci su questo fenomeno, un'altra cosa importante che bisogna tenere in considerazione è il tempo trascorso dal momento che si verifica la valanga a quando la vittima viene trovata. Il grafico che segue mostra molto chiaramente che le probabilità che una persona sopravviva dopo essere stata seppellita dalla neve, diminuiscono esponenzialmente in relazione ai minuti che passano prima di estrarla.



Per concludere, non possiamo non tenere in considerazione tutte le problematiche che riscontriamo al giorno d'oggi. Stiamo vivendo in piena crisi climatica: gas serra, deforestazione e scioglimento dei ghiacciai, che portano ad un innalzamento della temperatura, che produce, di conseguenza, un drastico aumento del rischio di valanghe.



Date le considerazioni fatte precedentemente possiamo dedurre quanto, in queste situazioni, la prontezza e la velocità siano importanti per quanto riguarda il soccorso delle vittime. Sfruttare tutta la tecnologia che abbiamo a disposizione, è sicuramente un ottimo modo per aumentare considerevolmente la percentuale di sopravvissuti.

Questo progetto ha quindi l'obiettivo di ideare un sistema di monitoraggio del territorio in cui si verifica una calamità meteorologica come una valanga e del soccorso di eventuali vittime. Come abbiamo visto prima, gli agenti devono essere in grado di svolgere questo lavoro nel più breve tempo possibile in modo da garantire il maggior numero di sopravvissuti. In tal caso utilizzeremmo entrambe le tipologie di dispositivi in modo tale che ognuno possa sfruttare i propri punti forti, cosicché la resa sia massima e vengano ridotti sia i tempi sia i costi.

Per una migliore comprensione, possiamo scomporre il compito in alcuni punti:

1. Posizionamento dei robot;
2. Interfaccia utente semplice dove selezionare il punto di maggiore probabilità;
3. Trovare la strada più veloce per il robot terrestre;
4. Decollo dei droni;
5. Comunicazione e divisione dinamica delle zone dei droni;
6. Rilevazioni nei punti trovati di maggiore probabilità;

Il programma è progettato in modo tale che possa riuscire ad affrontare tutte le sfide che gli si presentano dinanzi, che brevemente sarebbero i punti sopra elencati. In altre parole, permettere ad un utente di selezionare il punto con maggiore probabilità di presenza di vittime e automaticamente quest'ultimo viene poi comunicato ai droni e ai robot per procedere con il compito. Per di più, tutto il codice deve essere chiaro e documentato, per permettere eventuali futuri aggiornamenti ed estensioni, come ad esempio l'aggiunta di nuove funzionalità o miglioramenti di costo e tempo.

Il progetto è stato diviso in tre parti per rendere più agevole il lavoro da parte di un gruppo di tre studenti:

- Pianificazione Automatica Centralizzata delle missioni;
- Pianificazione Automatica Distribuita delle missioni
- Implementazione hardware in ambiente simulativo

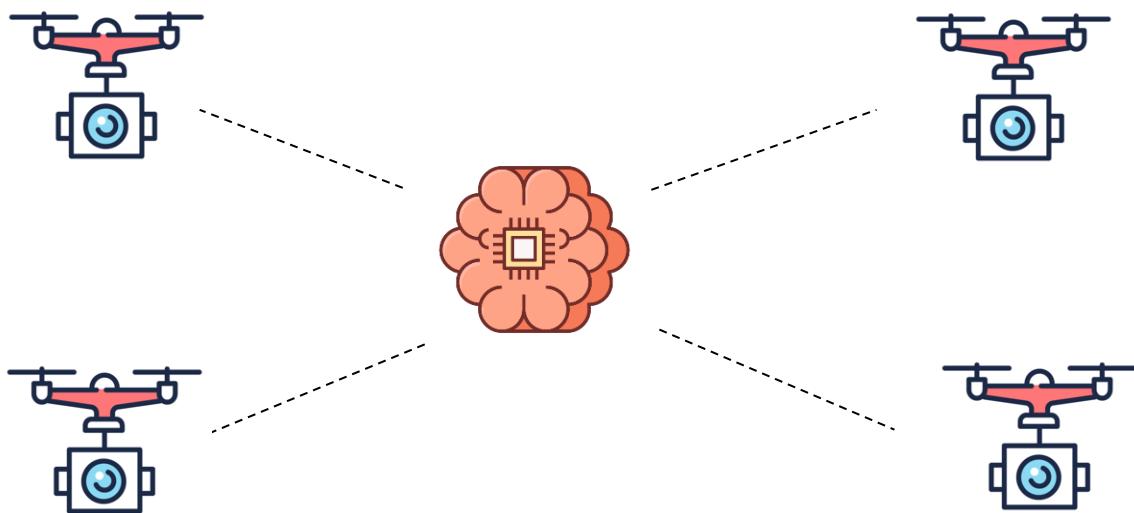
La seconda è l'oggetto di questa tesi e successivamente sarà successivamente trattata in maniera dettagliata. Le altre due sono state assegnate ad altri due studenti.

3 Problema

Il progetto è scomposto in 3 blocchi. I primi due riguardano il modo con cui avviene la comunicazione e la divisione dell'area tra gli agenti, ovvero centralizzata o distribuita, mentre il terzo blocco riguarda l'implementazione di questi due punti in un ambiente simulativo.

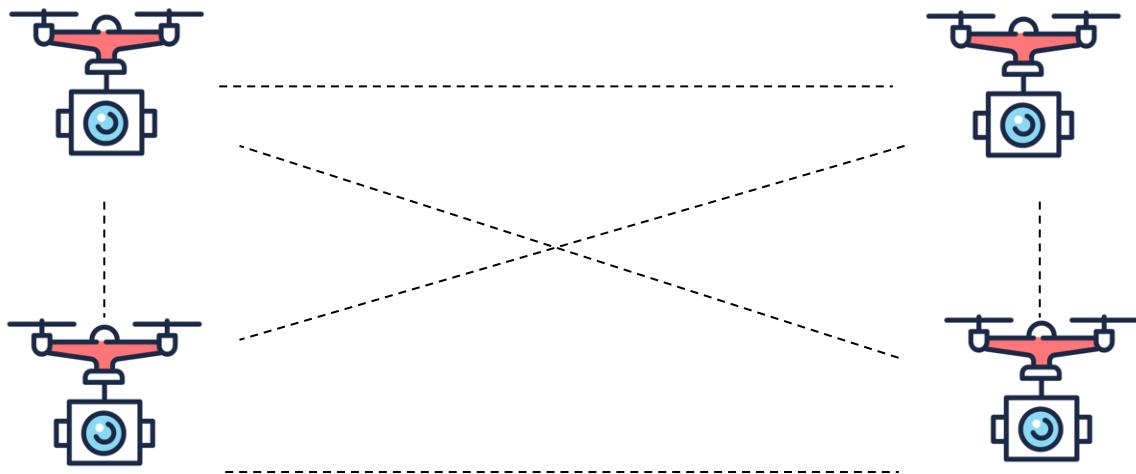
1. PIANIFICAZIONE E COMUNICAZIONE CENTRALIZZATA

Il primo approccio prevede che la comunicazione e la pianificazione del compito sia centralizzato. Con questo si intende che è presente un robot centrale, il quale è l'unico che ha la possibilità di comunicare con gli altri agenti e prendere decisioni su che cosa far fare alla flotta di droni e comunicare a ogni singolo robot quali sono le azioni da svolgere. Con questo approccio, quindi, i droni non sono a conoscenza della posizione degli altri simili.



2. PIANIFICAZIONE E COMUNICAZIONE DISTRIBUITA

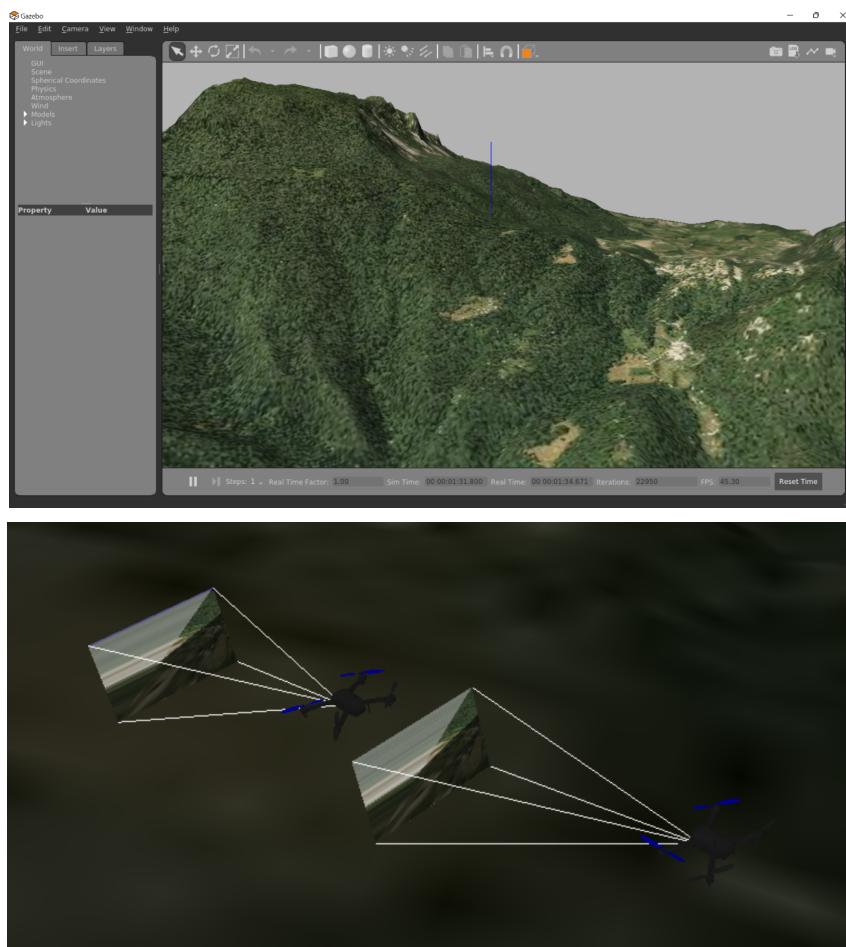
Il secondo approccio, invece, prevede che tutto il sistema sia decentralizzato. Questo comporta l'eliminazione di un nodo centrale che coordina, conosce il territorio e prende tutte le decisioni. Ogni agente, quindi, comunica con gli altri per capire in quale direzione muoversi al fine di massimizzare la copertura dell'area e rilevare l'altezza da terra per tenersi alla giusta quota. Per fare ciò, ogni drone può richiedere la posizione degli altri e a sua volta comunicare la propria, prendendo successivamente delle decisioni in maniera autonoma su quali azioni andare a svolgere.



3. SIMULAZIONE

In sostanza, una volta implementati i primi due blocchi, c'è bisogno di eseguire dei test al software e valutare quale sia il migliore algoritmo in maniera non più teorica, ma pratica. Partendo dall'interfaccia utente per selezionare i punti interessati, andiamo poi a testare e far comunicare tra loro i robot, utilizziamo i sensori, come ad esempio quello a ultrasuoni per valutare l'altezza da terra, scattiamo foto e le visualizziamo.

Qui sotto rispettivamente le foto dell'ambiente di simulazione Gazebo e due droni Iris.



3.1 Panoramica

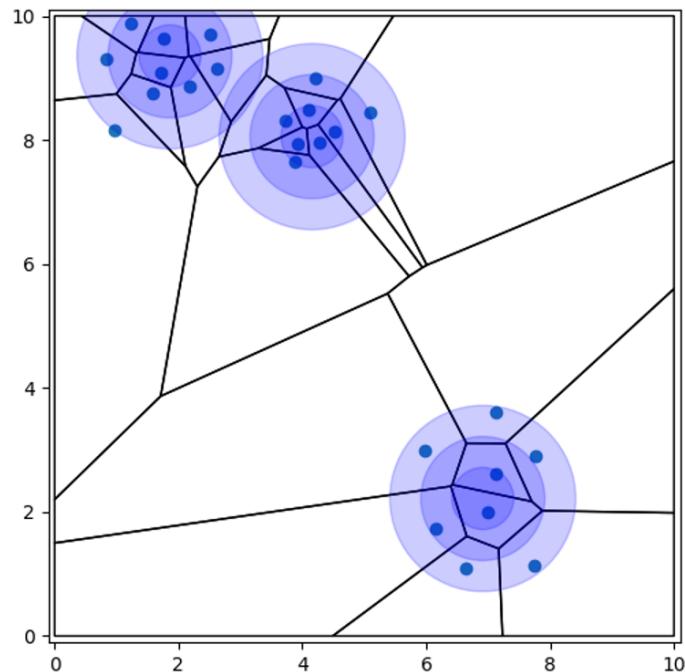
Nella figura numero 2 è presente la panoramica della struttura completa del progetto. Come già anticipato nell'introduzione e nel problema, il lavoro è stato diviso in 3 blocchi. Nel dettaglio:

(Parte a sinistra) Questa parte corrisponde alla gestione centralizzata, nella quale per partire avviene la creazione della mappa, la scelta dei percorsi migliori per ogni drone per poi successivamente collegarli ad un servizio per la comunicazione fisica tra gli agenti.

A. (Parte a destra) Questa parte mostra la gestione distribuita, o decentralizzata. Come si può vedere dal primo blocco, si utilizza l'algoritmo di Voronoi per calcolare le posizioni dinamicamente. E' questo algoritmo di Voronoi a calcolare, ad ogni step per ogni drone, la cella di Voronoi, ovvero il suo "centroide" e quindi la posizione verso la quale muoversi. Per fare ciò, come viene indicato nel secondo blocco, si ha quindi un continuo e costante rilevamento della posizione degli altri agenti e dell'altezza da terra. Questo permette dunque l'adattamento della posizione di ogni drone.

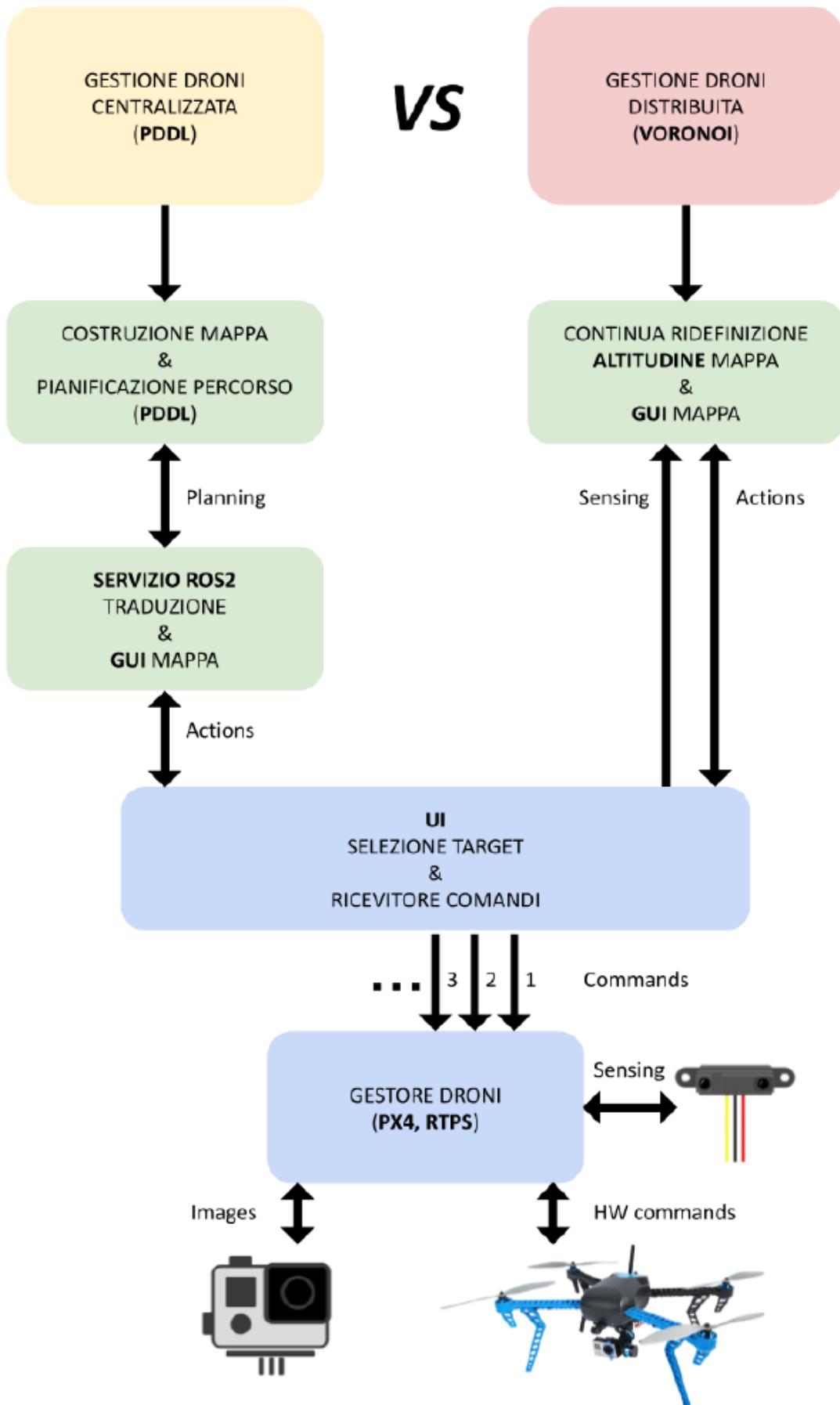
Nella figura 1, invece, è rappresentata una ipotetica situazione 2D in cui sono presenti 3 punti di interesse, rappresentati dai dischi blu, e 25 droni, rappresentati dai puntini blu. Come si può notare l'area è divisa in 25 sottoaree create dall'algoritmo di Voronoi e ognuna di queste si trova sotto il coordinamento di un drone.

Fig. 1



- (Parte in basso) Per concludere, l'ultima parte riguarda la parte più pratica e implementativa. Qui avviene l'esecuzione dei comandi utilizzati dalle sezioni precedenti, la lettura di sensori e telecamere.

Fig. 2



4 Background

In questo segmento saranno elencati e descritti tutti gli strumenti utilizzati per la pianificazione distribuita.

4.1 ROS2 Foxy su WSL Linux



Il Robot Operating System (ROS) è un insieme di librerie software e strumenti per la creazione di applicazioni robot, partendo da driver e algoritmi all'avanguardia fino ad arrivare ai potenti strumenti di sviluppo.

ROS Fornisce le stesse funzioni offerte da un sistema operativo come ad esempio l'astrazione dell'hardware e il controllo dei dispositivi.

Un insieme di processi all'interno di ROS si possono rappresentare in un grafo come dei nodi che possono ricevere e inviare i messaggi provenienti da e verso altri nodi e/o sensori.

Uno dei punti più forti di questa tecnologia è la possibilità di creare più nodi comunicanti tra di loro, nonostante siano scritti con linguaggi diversi (in questo caso C++ e Python).

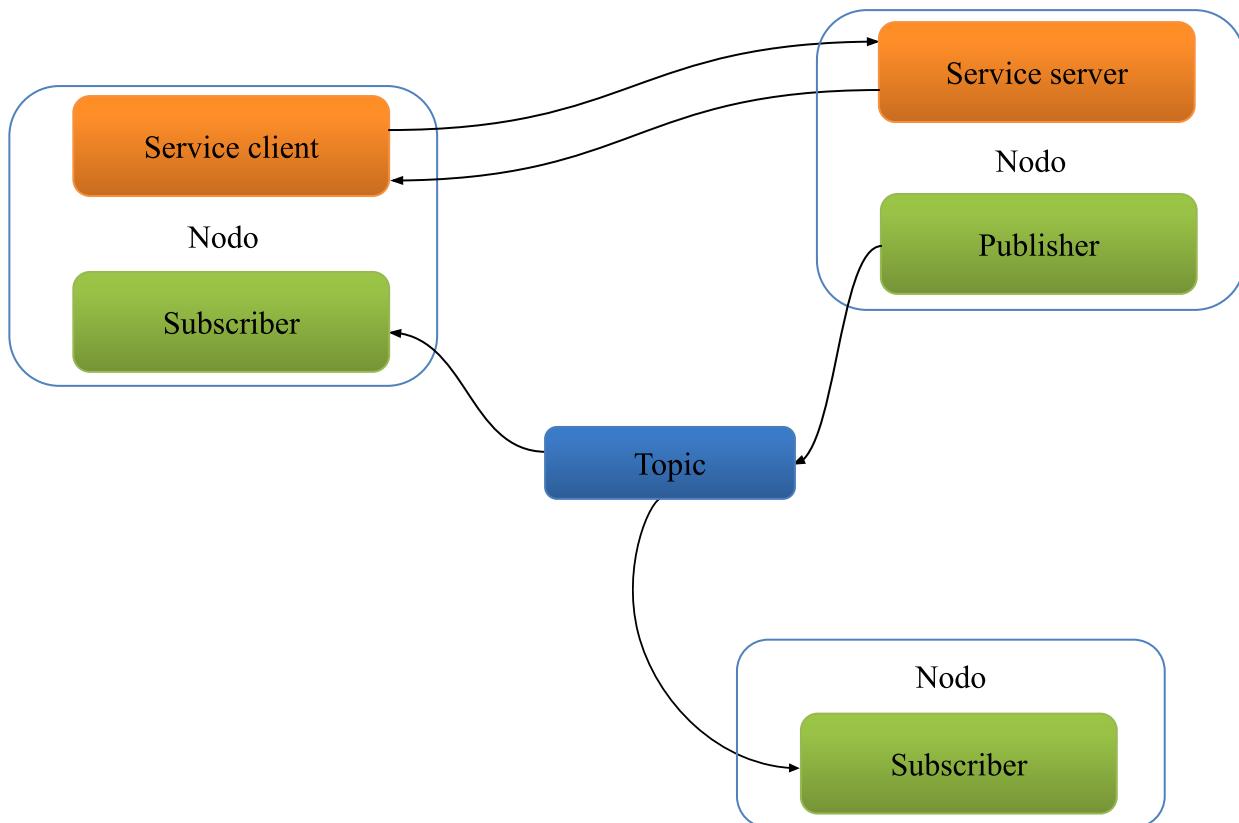
VANTAGGI

- Gratis: infatti tutto il sistema ROS è distribuito in maniera completamente gratuita.
- Open-source: offre possibilità veramente ampie lasciando quindi piena libertà di modifica e distribuzione di software sviluppati per questo ambiente, questo porta ad un altro vantaggio, la...
- Community: la *community* di ROS è veramente ampia e questo permette di trovare quasi sempre la soluzione a problemi e/o difficoltà che si possono riscontrare, dato che altre persone le hanno già incontrati precedentemente.
- Astrazione: con ROS possiamo sviluppare programmi (nodi) scritti in linguaggi diversi che però comunicano tra di loro.

Per capire meglio il funzionamento di ROS, si può dividere in punti:

- NODE: Il nodo è un processo che esegue calcoli all'interno dell'applicazione. I nodi comunicano tra di loro attraverso *topics* e *services*.
- TOPIC: I *topics* sono bus denominati su cui i nodi si scambiano messaggi. In generale, essi non sono a conoscenza con chi stanno comunicando. Invece, i nodi interessati ai dati si iscrivono al *topic* rilevante; i nodi che generano i dati pubblicano sul *topic* pertinente.
- SERVICE: Il *service* si basa sul modello di pubblicazione/sottoscrizione. Questo è definito da una coppia di messaggi: uno per la richiesta e uno per la risposta. Ogni nodo può pubblicare e/o sottoscriversi ad uno o più servizi, offerti a loro volta da altri nodi.

Nella figura seguente è rappresentata la struttura di alcuni nodi ROS con relativi *topics* e *services*.



4.2 Git & GitHub: Cooperation e Versioning



Git è un software per il controllo di versione distribuito in locale, utilizzabile da interfaccia a riga di comando.

GitHub è una piattaforma di hosting e permette quindi di caricare il proprio codice online, e renderlo disponibile ad altri utenti. Questi strumenti combinati insieme offrono due principali caratteristiche:

- Cooperazione: con il codice pubblico online si ha, con immediata facilità, la possibilità che più persone lavorino simultaneamente sullo stesso codice.
- Versionamento: Git offre la possibilità di tenere traccia di ogni modifica fatta, miglioramenti o risoluzione di errori, documentando ogni cambiamento e offrendo la possibilità di ripristinare versioni precedenti in qualsiasi momento.

In aggiunta, con questi due strumenti possiamo documentare il codice e renderlo pubblico, per esempio nel caso in cui altri utenti creino ulteriori implementazioni o miglioramenti.

La figura successiva mostra come si presenta un progetto GitHub.

The screenshot shows a GitHub repository page for 'albertor2000/Distributed_Coverage_Network'. The repository is public and has 1 branch and 0 tags. The README files show initial commits for Documentation, agent_interfaces, agents_pos_service, interfaces, robots_control, and README.md. The 'About' section indicates no description, website, or topics provided. The 'Releases' section shows no releases published, with a link to 'Create a new release'. The 'Packages' section shows no packages published, with a link to 'Publish your first package'. The 'Languages' section displays a chart showing the distribution of code across various languages: TeX (18.3%), Makefile (16.3%), HTML (14.2%), CMake (13.2%), Python (13.1%), C++ (9.1%), and Other (15.6%).

4.3 Visual Studio Code: editor per C++ e Python



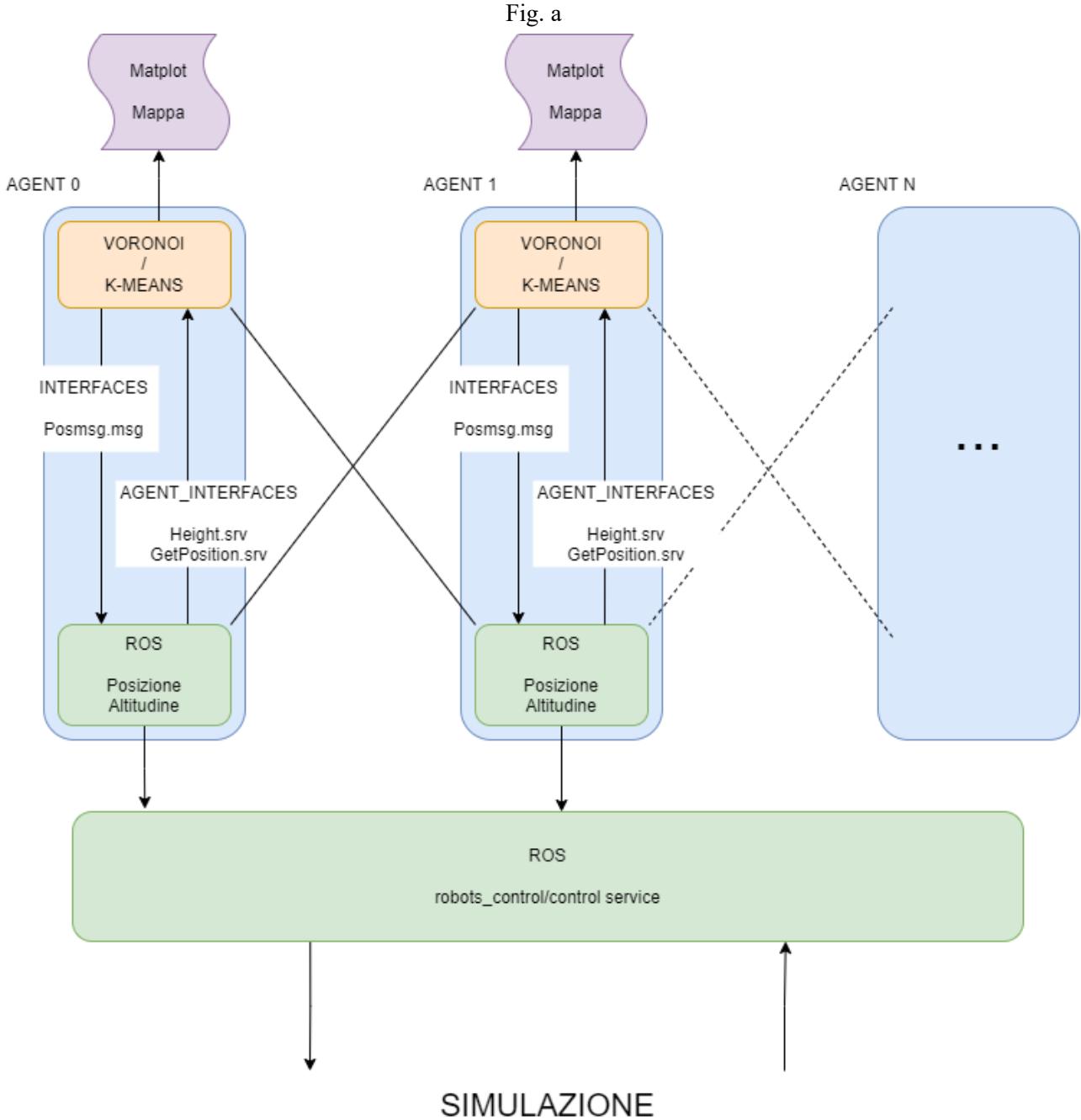
Come già scritto sopra, ROS2 utilizza i linguaggi C++ e Python per la creazione dei nodi utilizzati dal software. Per sviluppare quest'ultimo abbiamo bisogno di un ambiente affidabile, che permette la scrittura di codice in entrambi i linguaggi e eseguibile in Linux.

Visual Studio Code è un editor di codice sorgente sviluppato da Microsoft per Windows, Linux e macOS. È stato scelto di utilizzare questo framework perché include il supporto per debugging, ha la possibilità di aggiungere un numero vastissimo di estensioni per migliorare la qualità del codice e soprattutto ha un controllo per Git integrato.

5 Soluzione

5.1 Architettura

Andiamo ora ad approfondire l'architettura l'architettura riguardante la **pianificazione automatica distribuita**.



Come già detto precedentemente e come si può vedere dallo schema, non è presente nessun nodo centrale, proprio perché la comunicazione è completamente distribuita e in questo caso è rappresentata dai segmenti che collegano tra di loro i vari agenti.

5.2 Specifiche sulla procedura

Di seguito sono esposti i vari step che ogni agente deve compiere per completare il suo lavoro:

1. Prima di tutto l'agente deve capire in quale punto è localizzato. Per fare ciò deve chiedere all'apposito servizio la propria posizione, facendo uso dell'interfaccia ROS2 *agents_interfaces*. Il messaggio scambiato è composto da 4 parametri, 1 di richiesta e 3 di risposta:
 - a. *agent_id*: (Richiesta) Specifica di quale agente si vuole conoscere la posizione
 - b. *x*: (Risposta) Coordinata x attuale dell'agente.
 - c. *y*: (Risposta) Coordinata y attuale dell'agente.
 - d. *z*: (Risposta) Coordinata z attuale dell'agente.
2. Sempre tramite ROS2 viene chiesto al sensore ad ultrasuoni del drone la distanza dal terreno, questo per far sì che l'altezza venga aggiornata in caso questa non sia quella desiderata.
3. A questo punto, una volta che l'agente conosce dove si trova nello spazio, deve capire come muoversi. Utilizza quindi l'algoritmo scelto, K-Means o Voronoi, per andarsi a calcolare la destinazione e poi dirigersi verso quella posizione.
4. Per un maggiore controllo e monitoraggio da parte dell'utente è opportuno che quest'ultimo riceva costantemente un riscontro da parte del sistema. Per questo motivo, ad ogni iterazione dell'algoritmo usato nel punto precedente, viene salvata un'immagine che mostra:
 - a. Se viene utilizzato K-Means: la posizione di tutti gli agenti, e la posizione dei *punti di controllo*, colorati in maniera differente per simboleggiare che appartengono a droni diversi. (Fig. b)
 - b. Se viene utilizzato Voronoi: la posizione degli agenti, la posizione dei punti di maggiore probabilità di trovare degli individui inseriti dall'utente e i limiti di ogni cella di Voronoi (Fig. c)

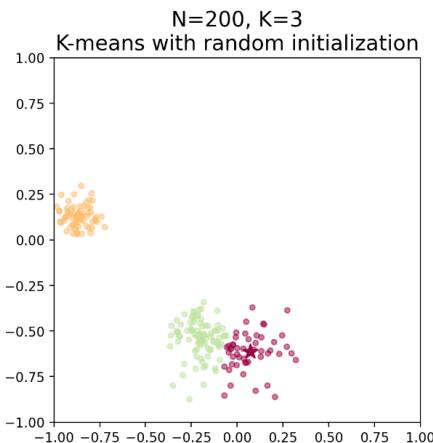


Fig. b

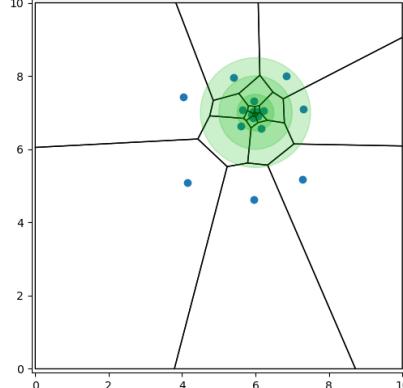


Fig. c

5. Come mostrato dalla figura a, una volta calcolata la destinazione, viene passata tramite messaggio al nodo ROS2 *robots_control* che si occupa di mandare i comandi per muovere il drone.

6. Per finire, il nodo ROS2 *robots_control* invia costantemente messaggi all'agente per farlo procedere in maniera continua verso la direzione calcolata nel punto 3. Il messaggio utilizzato è composto dalle coordinate x, y e z.

Per una maggiore comprensione possiamo dividere l'architettura in 3 moduli, i quali verranno dettagliatamente spiegati successivamente. I primi due riguardano l'algoritmo utilizzato per la divisione delle aree di copertura tra i vari agenti, rispettivamente K-Means e Voronoi. Entrambi lavorano su due dimensioni, in quanto l'altezza verrà calcolata tramite il sensore ad ultrasuoni. Il terzo modulo invece si occupa della comunicazione con ROS. Nel dettaglio:

1. **DIVISIONE AREA CON K-MEANS:**

Il primo approccio utilizzato per la divisione della copertura è l'utilizzo di un algoritmo K-Means: si vanno a generare un numero discreto di punti, che chiameremo *punti di controllo*, posizionati con una distribuzione gaussiana intorno ai punti segnalati dall'utente di maggiore probabilità di trovare vittime. Intuitivamente, più si è vicini al punto segnalato, maggiore sarà la quantità di *punti di controllo*. L'obiettivo che l'algoritmo si propone è di minimizzare la varianza totale intra-gruppo; infatti ogni gruppo viene identificato mediante un centroide o punto medio. L'algoritmo segue una procedura iterativa: inizialmente crea k partizioni, nel quale k rappresenta il numero di droni, e assegna i punti d'ingresso a ogni partizione casualmente; in seguito calcola il centroide di ogni gruppo; costruisce in seguito una nuova partizione associando ogni punto d'ingresso al gruppo il cui centroide è più vicino; concludendo con il ricalcolo dei centroidi per i nuovi gruppi e via dicendo, fino a che l'algoritmo non converge.

2. **DIVISIONE AREA CON VORONOI:**

Il secondo approccio fa uso dell'algoritmo di Voronoi. In questo caso all'avvio del software, ogni agente comunica con tutti gli altri e scambia informazioni sulla reciproca posizione, in modo che ogni robot sia a conoscenza della posizione di tutti gli altri.

A questo punto, ogni agente calcola la propria regione di Voronoi, determina il centroide e lo utilizza per trovare la direzione verso la quale muoversi. Dopo un intervallo di tempo, chiede nuovamente la posizione degli agenti per riuscire a ricalcolare la propria regione, il centroide e così via, finché non converge o raggiunge il numero massimo di iterazioni.

3. **SERVIZIO ROS2:**

Come ultimo passo ci si va a collegare con l'ambiente esterno utilizzando quindi ROS2, con uno scambio di messaggi, ignorando come e con che codice è stato scritto il software di ogni agente, permettendo così una comunicazione agevolata. Questi messaggi sono la posizione che dovrà essere successivamente inoltrata agli agenti per far loro capire verso quale punto devono spostarsi. Oltre a ciò, ROS2 viene utilizzato anche per raccogliere i dati dai sensori degli agenti: ad esempio il sensore ad ultrasuoni per trovare la distanza dal terreno e mantenerla costante, assieme anche alla posizione in tempo reale.

Attraverso ROS2 abbiamo anche la capacità di rilevare eventuali errori e/o imperfezioni nel software, andando quindi poi ad apportare eventuali migliorie e implementazioni.

5.3 Modulo 1: Algoritmo K-Means

Il primo approccio adottato per la divisione dell'area di copertura in sottoaree consiste nell'utilizzo di un algoritmo di K-Means. I punti di forza di questa procedura sono:

- Relativamente semplice da implementare;
- Garantisce la convergenza, infatti conoscendo che i *punti di controllo* sono distribuiti in maniera gaussiana, sappiamo che esiste un centroide;
- Efficace, in quanto lavoriamo su un piano bi-dimensionale e utilizziamo la distanza euclidea come parametro per valutare il cluster, portando quindi a calcoli molto veloci e accurati;
- Inoltre, è davvero flessibile, dato che si adatta al numero di punti di interesse selezionati dall'utente.

Descrizione

Dato un numero di punti di interesse (x_1, x_2, \dots, x_n), K-Means punta a partizionare la n osservazioni in k (nel nostro caso, il numero di agenti), clusters $S = \{S_1, S_2, \dots, S_k\}$ in modo da minimizzare la somma delle radici (ovvero la varianza) all'interno del cluster:

$$\arg \min_{\mathbf{S}} \sum_{i=1}^k \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2$$

Dove $\boldsymbol{\mu}_i$ è il centroide.

Entriamo ora nel dettaglio passo passo su come questo algoritmo è implementato per il nostro obiettivo:

1. Generazione dei punti
2. Assegnazione punti ai clusters
3. Calcolo centroidi
4. Spostamento verso il centroide

5.3.1 Generazione dei punti

L'idea alla base per implementare questo algoritmo al nostro specifico caso è la seguente: dopo che si è venuti a conoscenza dei punti di maggiore probabilità di trovare persone, si genera un numero n discreto, scelto a priori, di punti distribuiti con probabilità gaussiana intorno al punto di interesse.

Il numero n di punti deve essere scelto con cura infatti può variare di molto le prestazioni del nostro algoritmo:

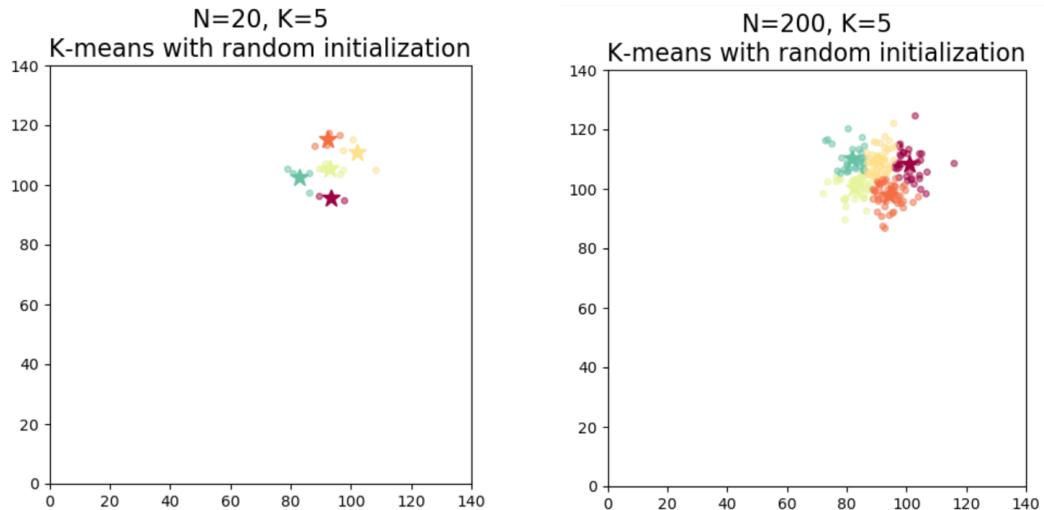
maggiori sono i punti, più precisa e corretta sarà la posizione dei nostri agenti, ma, d'altra parte, maggiore sarà il tempo necessario per trovare il centroide.

K-Means, infatti, è NP-completo e fissati k e n , la sua complessità è

$$O(n^{dk+1})$$

Le figure successive mostrano come la precisione della posizione del centroide cambia in relazione al numero n . La prima immagine mostra un esempio in cui ci sono $n=20$ punti, e vediamo che vengono divisi tra 5 agenti, in questo modo ogni agente può calcolarsi il centroide solamente per i suoi 4 punti.

Invece, nel secondo esempio vengono usati $n=200$ punti, mostrando quindi che la posizione dei centroidi è notevolmente più accurata.



5.3.2 Assegnazione punti al cluster

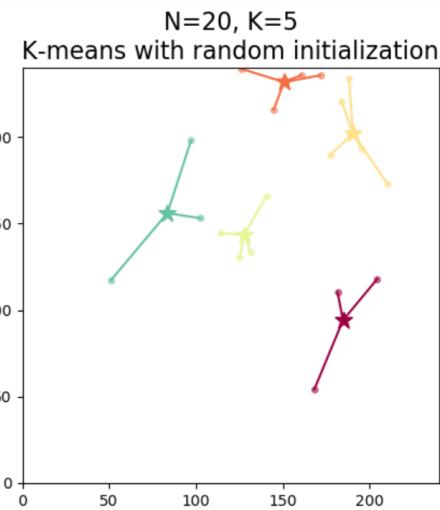
Come già anticipato, ogni *punto di controllo* deve essere associato al proprio cluster, il quale è a sua volta associato ad un drone. La regola alla base è che ogni punto viene associato al centroide più vicino, ovvero quello che ha la minore distanza euclidea, cioè

$$d_e = \sqrt{punto\ di\ interesse^2 + centroide^2}$$

Più dettagliatamente

$$d_e = \sqrt{(x_{punto\ di\ interesse} - x_{centroide})^2 + (y_{punto\ di\ interesse} - y_{centroide})^2}$$

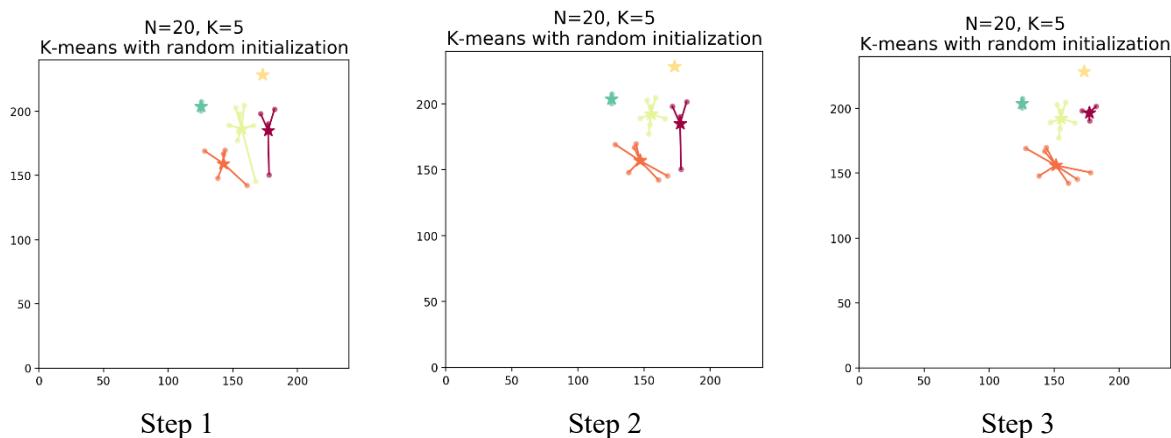
Come si può vedere dalla figura sottostante, la distanza minima è rappresentata dalle linee, colorate in base al cluster di appartenenza.



5.3.3 Calcolo dei nuovi centroidi e spostamento

Si è arrivati quindi alla parte centrale di questo algoritmo: ora che ogni *punto di controllo* è associato ad un cluster, è il momento di calcolare il nuovo centroide. Con una procedura iterativa, per ogni cluster, si va a trovare un punto c che porta ad avere la media minore tra tutte le distanze d_{cp} , dove p simboleggia un *punto di controllo* del cluster in considerazione.

Nell'esempio seguente, si nota come ad ogni iterazione, i cluster diventano sempre più compatti verso il proprio centroide, diminuendo quindi la distanza media tra tutti i punti.



5.4 Modulo 2: Algoritmo Voronoi

L'approccio più conveniente, però, è l'algoritmo di Voronoi, o meglio, una variazione dell'algoritmo di Lloyd, che garantisce una migliore suddivisione dell'area di copertura in sottoaree.

L'algoritmo di Lloyd è una procedura iterativa per trovare delle aree di punti equidistanti in uno spazio euclideo, dividendo questo spazio in sezioni. Similmente a K-Means, la procedura di Lloyd trova ripetutamente il centroide di ogni partizione per poi dividere nuovamente l'area in sezioni basandosi su quale di questi baricentri è più vicino.

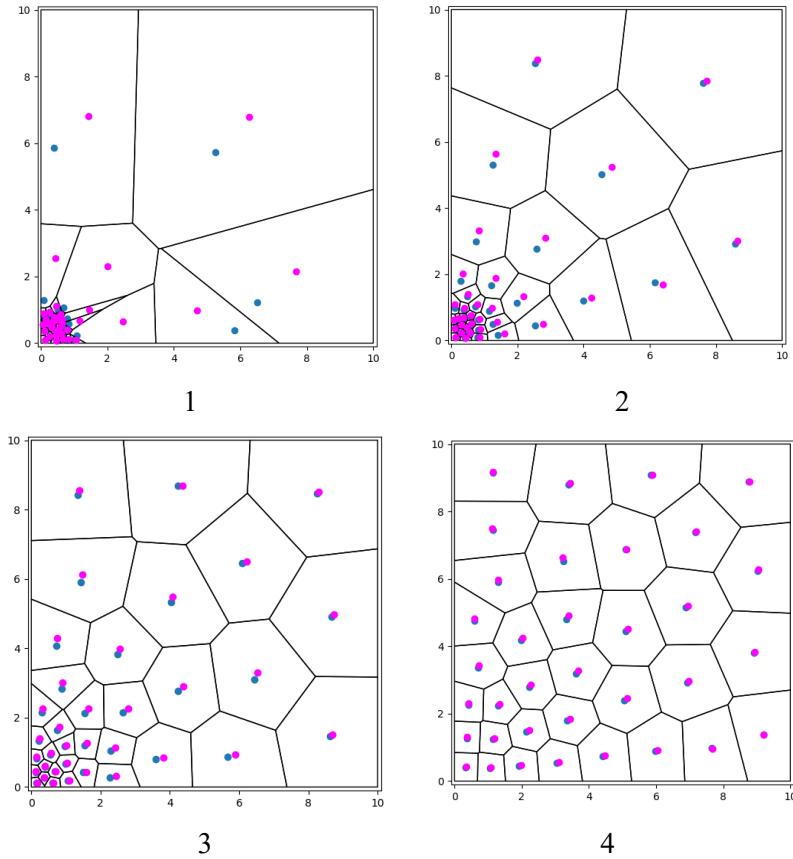
L'algoritmo di Lloyd tiene in considerazione tutti i limiti che possono avere gli agenti. Si è scelto questo algoritmo perché è:

- **Adattivo:** l'algoritmo di copertura fornisce una rete con la capacità di affrontare ambienti mutevoli, attività di monitoraggio e cambiamenti della topologia della rete (a causa di partenze, arrivi, guasti o malfunzionamenti).
- **Distribuito:** Le decisioni e quindi il comportamento di ogni veicolo dipende solo dalla posizione dei suoi vicini. Inoltre, l'algoritmo non richiede un grafo di comunicazione a topologia fissa, ovvero le relazioni di vicinato possono cambiare continuamente con la rete che evolve. I vantaggi degli algoritmi distribuiti sono la scalabilità e robustezza.
- **Asincrono:** L'algoritmo permette l'implementazione asincrona. Ciò significa che può essere implementato in una rete composta da agenti che si evolvono a velocità differenti, con differenti computazioni e capacità di comunicazione. Inoltre, la procedura di Lloyd non richiede una

sincronizzazione globale e le proprietà di convergenza sono conservative, anche se le informazioni sui veicoli vicini si propagano con un certo ritardo. Un vantaggio dell'asincronismo è una comunicazione ridotta al minimo.

- Verificabile Asintoticamente Corretto: perché garantisce una discesa monotona della funzione di costo per quanto riguarda il compito di rilevamento. Asintoticamente, è garantito che la rete di robot arriva alla convergenza verso i punti di copertura ottimale.

I punti usati per il calcolo delle regioni di Voronoi sono rappresentati in blu, mentre in rosa sono i centroidi di ogni partizione.



Nel caso di copertura di uno spazio con agenti, l'algoritmo scomposto in 3 passi consiste in:

1. Si calcola la cella di Voronoi per ogni agente;
2. Si trova il centroide (o baricentro) della cella;
3. Ogni agente si sposta verso il baricentro.

Come anticipato prima, si utilizzerà una versione modificata dell'algoritmo di Lloyd: quest'ultimo dispone tutti i punti in maniera equidistante tra loro, mentre nel nostro caso abbiamo bisogno che essi siano concentrati nei luoghi di maggiore probabilità di trovare vittime.

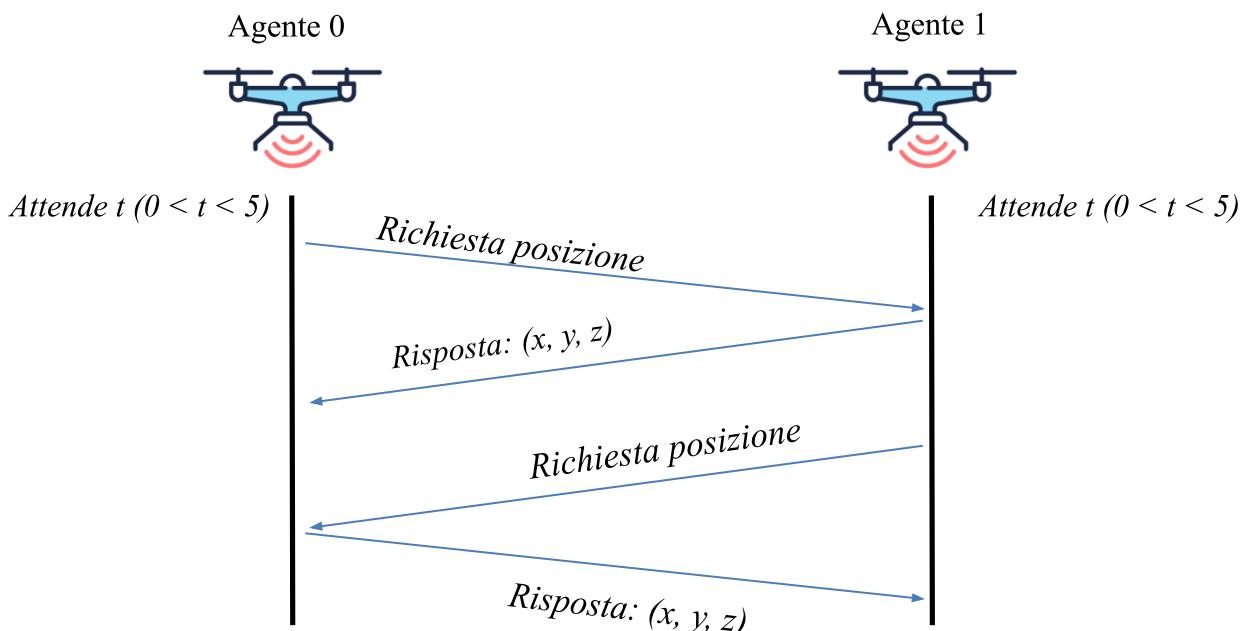
5.4.1 Conoscenza vicini

Prima ancora di applicare l'algoritmo di Voronoi, ogni agente ha bisogno di conoscere la posizione di tutti gli altri droni. Si introduce un tempo casuale t compreso tra 0 e 5 secondi per evitare che due robot si chiedano reciprocamente la posizione e creino, di conseguenza, un deadlock, dato che entrambi stanno aspettando che l'altro invii la posizione prima di procedere.

Il messaggio di risposta è composto da 3 valori:

- x : coordinata x della posizione dell'agente a cui è arrivata la richiesta;
- y : coordinata y della posizione dell'agente a cui è arrivata la richiesta;
- z : coordinata z della posizione dell'agente a cui è arrivata la richiesta.

Esempio scambio di messaggi per la posizione tra due agenti:



Possiamo procedere con il calcolo della...

5.4.2 Regione di Voronoi

Una volta conosciuta quindi la posizione di tutti gli agenti, si può iniziare a calcolare la propria regione di copertura, ricordando che ogni agente calcola la sua. Il nostro obiettivo è quello di ridurre al minimo l'area di copertura di ogni robot, in modo tale da aumentare le prestazioni dei sensori, avendo questi ultimi, una capacità limitata.

Si può facilmente dedurre che la copertura ottimale Q è la partizione di Voronoi $V(P)$, con P l'insieme delle posizioni degli agenti, generata dai punti (p_1, \dots, p_n) .

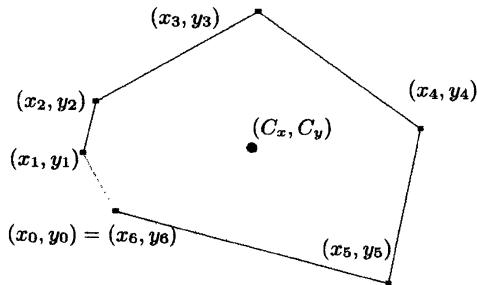
$$V_i = \{q \in Q \mid \|q - p_i\| \leq \|q - p_j\|, \forall j \neq i\}$$

Per una più chiara comprensione si ricordano alcuni concetti rilevanti:

- L'insieme delle regioni $\{V_1, \dots, V_n\}$ è chiamato diagramma di Voronoi;
- Il diagramma di Voronoi è generato dai punti $\{p_1, \dots, p_n\}$;
- Quando due regioni V_i e V_j sono adiacenti, quindi che condividono un confine, il generatore p_i è chiamato vicino di p_j .

5.4.3 Calcolo del centroide

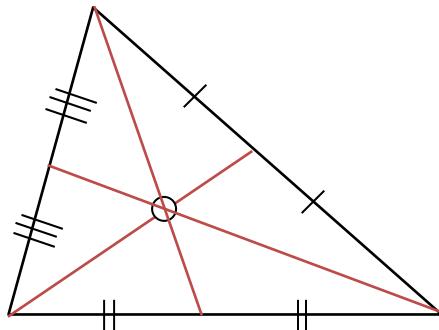
A questo punto tutta l'area da coprire è stata partizionata, questo vuol dire che ogni agente ha calcolato la propria regione di Voronoi ed è quindi ora di procedere con il calcolo del baricentro, o *centroide*.



Prendiamo come riferimento la figura sopra nella quale sono rappresentati i punti che costituiscono la regione e il centroide.

Assumiamo che la regione V_i sia un poligono convesso, anche chiamato politopo in R^2 , con N_i vertici etichettati $\{(x_0, y_0), \dots, (x_{N(i)-1}, y_{N(i)-1})\}$. È conveniente definire $(x_{N(i)}, y_{N(i)}) = (x_0, y_0)$.

Graficamente, il baricentro si trova in questo modo:



Si utilizzano le seguenti formule per calcolare le coordinate del centroide:

$$M_{V_i} = \frac{1}{2} \sum_{k=0}^{N_i-1} (x_k y_{k+1} - x_{k+1} y_k)$$

$$C_{V_i, x} = \frac{1}{6M_{V_i}} \sum_{k=0}^{N_i-1} (x_k + x_{k+1})(x_k y_{k+1} - x_{k+1} y_k)$$

$$C_{V_i, y} = \frac{1}{6M_{V_i}} \sum_{k=0}^{N_i-1} (y_k + y_{k+1})(x_k y_{k+1} - x_{k+1} y_k).$$

Dove Mv_i rappresenta la massa della regione e $Cv_{i,x}$ e $Cv_{i,y}$ rispettivamente la coordinata x e y del baricentro. Si ricorda che si lavora in un piano bi-dimensionale in quanto la terza dimensione non fa parte dell'algoritmo, ma essa viene ugualmente monitorata e aggiornata per mantenerla costante durante tutto il processo.

Successivamente si spiegherà la parte relativa all'invio di queste coordinate tramite un servizio ROS2, per fare in modo che i droni si muovano.

5.5 Modulo 3: Servizio ROS2

Si è arrivati alla conclusione, il momento di unire tutte le parti viste fino ad ora. Entrambi gli approcci del modulo 2 e 3 ora comunicheranno con i servizi ROS2 per permettere ai robot di ricevere le azioni da seguire, sotto forma di messaggi, come già detto in maniera dinamica.

5.5.1 Nodi ROS2

Come primo passo dobbiamo sicuramente avviare i nodi con i quali andremo a comunicare tramite *services* e *topics*.

I nodi utilizzati sono i seguenti:

- *Px4 Autopilot*: Sistema open source di pilota automatico per velivoli autonomi. Il nodo mette a disposizione tutti i servizi necessari per comandare i robot, leggere i valori dati dai sensori e attuare procedure.
- *Robots_control*: Questo nodo (o pacchetto) è quello che permette di interfacciarsi con l'ambiente simulativo, in questo caso, oppure con robot fisici nel caso di un'operazione nel mondo reale.

Questo pacchetto ha a disposizione le seguenti azioni:

- a) azione 1: *move_drone (drone id, x, y, z)* permette di inviare messaggi in maniera continua al drone per farlo muovere costantemente verso la posizione specificata;
- b) azione 2: *move_rover (rover id, x, y, z)* invia messaggi al rover per indicare verso quale direzione muoversi;
- c) azione 3: *takeoff (drone id, z)* in questo passaggio inizia la procedura di decollo del drone, occorre disarmarlo ed eseguire tutti i controlli e calibrazioni necessarie per intraprendere il volo;
- d) azione 4: *landing()* fa in modo che il drone atterri atterri sulla corretta coordinata, andando quindi ad abbassarsi progressivamente finché il sensore non rileva che ha raggiunto il suolo. A quel punto inizia la procedura di landing, offerta dal nodo Px4.
- e) azione 5: *take_picture (drone id, x, y, z)* segnala al drone di iniziare la routine per fotografare la zona interessata (indicata da x , y , z). La routine consiste nell'andare nella posizione

specificata, scattare la prima foto, ruotare di 180° e concludere scattando la seconda fotografia.

- *Agents_pos_services*: Questo pacchetto contiene i nodi drone del nostro sistema, chiamati *twin_<n>*. Questi ultimi processano gli algoritmi di K-Means o Voronoi per il calcolo della posizione successiva. Questo pacchetto raccoglie tutte le informazioni necessarie dai nodi offerti da Px4, come la posizione e l'altitudine.

Vediamo ora come avviare questi nodi sopra elencati per inizializzare l'ambiente di simulazione con tutti i servizi richiesti. I comandi, in ordine, sono da eseguire ognuno in un terminale diverso:

Questi primi due comandi servono rispettivamente per avviare la simulazione in Gazebo e Px4 e per inizializzare i droni.

```
> ros2 launch start0.launch.py
```

```
> ros2 launch start1.launch.py
```

Avviamo anche il servizio *ROS2 robots_control control* che fa da ponte tra i nodi che calcolano la divisione dell'area e i messaggi dei comandi ai droni, con il comando:

```
> ros2 run robots_control robot
```

Infine, si possono lanciare anche i nodi *ROS2 agents_pos_service twin_<n>*:

```
> ros2 run agents_pos_service twin_0
```

5.5.2 Chiamata ad algoritmo di divisione aree

Una volta avviato tutto il sistema, siamo pronti per procedere alla divisione dell'area di copertura con algoritmo di Voronoi.

Il drone, quindi, chiede al servizio della posizione dove si trova, utilizzando l'interfaccia *agents_interfaces GetPosition.srv*, e riceve come risposta le sue coordinate attuali.

```
int64 id
---
float64 x
float64 y
```

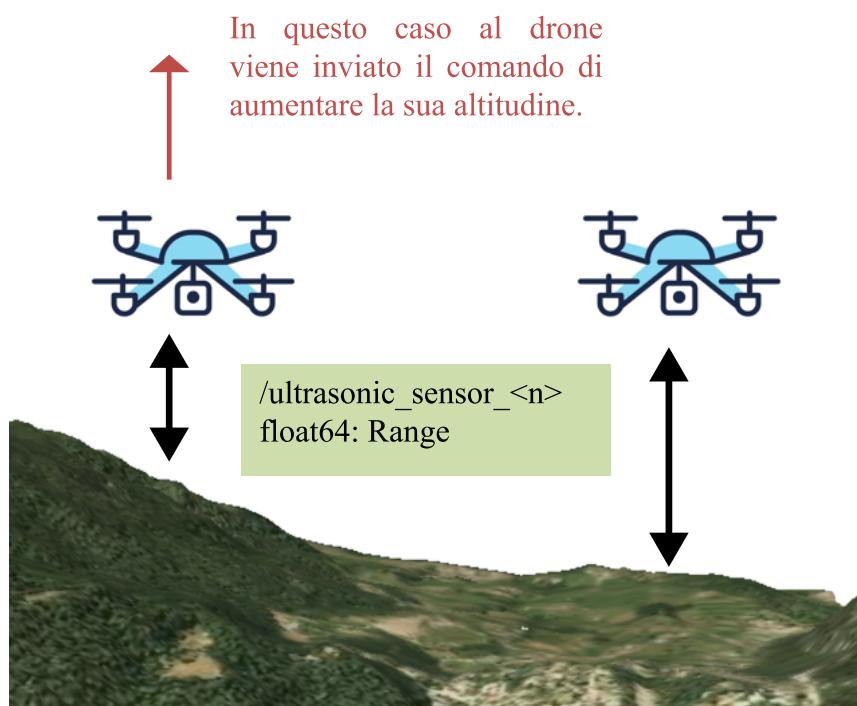
Procede allora, come spiegato nel capitolo precedente, a calcolarsi la propria area, il proprio centroide e di conseguenza la direzione verso la quale muoversi.

5.5.3 Calcolo altitudine

Manca solamente da conoscere la coordinata z , ovvero l'altitudine verso la quale alzarsi o abbassarsi. Per fare ciò, il nodo *twin_<n>*, fa una richiesta al *topic /ultrasonic_sensor_<n>*, il quale risponde restituendo l'altezza in metri alla quale si trova in questo momento.

Si ricorda che, non conoscendo la mappa a priori, dato che può essere stata modificata dopo l'evento di una valanga, l'agente deve continuare a monitorare il terreno per tenersi in quota.

Come mostrato nella figura seguente, il terreno della montagna può variare velocemente e quindi è opportuno controllare sempre a che altezza si trova il drone.



5.5.4 Traduzione punti

L'algoritmo di Voronoi utilizzato in questo progetto lavora solamente su numeri interi, più specificatamente nell'intervallo $[0, 10]$, mentre i droni lavorano su una mappa con coordinate x e y comprese nell'intervallo $[-1024, 1024]$.

Proprio per questo motivo è necessario convertire i punti di riferimento utilizzati dall'algoritmo di Voronoi in coordinate vere e proprie adatte ai droni. Si utilizzano quindi due funzioni:

1. *VoronoiToMap(coordinate)*: questa funzione si occupa di convertire la destinazione calcolata da Voronoi in coordinate leggibili prima che queste vengano spedite al drone;
2. *MapToVoronoi(coordinate)*: viceversa, converte le coordinate ricevute dal mondo esterno in punti utilizzabili dall'algoritmo di divisione delle aree.

6 Analisi sperimentale e validazione

Durante tutto lo svolgimento di questo progetto, sono stati svolti continui test e analisi per rivelare eventuali malfunzionamenti o problematiche. Sono stati eseguiti dei test anche per trovare i parametri migliori, spiegati meglio nel punto successivo: come, ad esempio, il numero di punti da generare per l'algoritmo di K-Means, intorno al punto di maggiore probabilità di vittime.

6.1 Parametri

I parametri sono entità che influiscono in maniera rilevante sulle prestazioni del software. Data la loro importanza, scegliere i giusti parametri non è sempre così facile. I parametri più importanti affrontati in questo progetto sono:

- Agenti utilizzati:

Per la simulazione, è stato scelto di utilizzare due droni e un robot terrestre, questo per trovare il giusto compromesso tra simulare una situazione il più reale possibile e le prestazioni limitate del pc, essendo la simulazione molto costosa a livello computazionale. Infatti, se il numero di agenti risulta elevato, le prestazioni calano drasticamente, andando a causare dei malfunzionamenti.

Tuttavia, il software realizzato è scalabile, ed è infatti in grado di gestire un numero veramente considerevole sia di droni che di robot terrestri.

- Tempo di attesa per aggiornamento destinazione:

Ai droni, una volta in volo, viene data la prima destinazione verso la quale dirigersi. Da quel momento, dopo un intervallo di tempo, viene ricalcolata, richiedendo la posizione agli altri agenti e gli altri valori. Anche qui è stato d'obbligo trovare il giusto parametro, perché:

- o Se il tempo di attesa prima della posizione successiva è troppo **elevato**, il drone, volando verso un'eventuale direzione sbagliata, ci metterebbe troppo tempo per poi ricalcolare la traiettoria e aggiustare il verso . Invece...
- o Se il tempo di attesa è troppo **breve**, si rischia che un drone abbia troppe richieste da gestire, portando così ad un rischio maggiore di deadlock, oppure a non svolgere più le proprie operazioni poiché le risorse sono impegnate.

- Mappa:

La mappa ha un ruolo fondamentale nello svolgimento dei test: si è dunque puntato ad utilizzare una sezione della mappa molto variegata, comprendente pianura, collina e montagna. È stata scelta la zona est di Povo, Trento, in quanto offre tutto quello citato appena prima, con pendenze anche molto ripide , creando così la possibilità dell'evento di valanghe, simulando inoltre l'impossibilità per il robot terrestre di percorrere certi tratti e vedere pertanto il suo comportamento in quel caso.

I dati della mappa sono stati presi online dalla piattaforma WebGIS.

6.2 Implementazione con le altre parti

Una volta sviluppata la parte centrale di questa tesi, ovvero la Pianificazione distribuita, ci si è mossi per testare questa parte nell'ambiente simulativo.

Grazie all'accurata documentazione e allo strumento GitHub, l'integrazione è avvenuta senza riscontrare troppe difficoltà.

La parte che ha richiesto più lavoro è stata passare da un planning statico ad uno dinamico, ciò significa che non si andava a pianificare tutto il percorso fin dall'inizio, conoscendo già la strada da percorrere a priori. Difatti, essendo che la pianificazione distribuita lavora in maniera dinamica, avendo un continuo scambio di informazioni con l'ambiente esterno, e prendendo decisioni sulla base di queste, c'è stata la necessità di aggiornare man mano la nuova destinazione da raggiungere.

6.3 Pro e contro tra Pianificazione Centralizzata e Distribuita

QUALITA' DELLA PIANIFICAZIONE CENTRALIZZATA:

1. Il sistema centrale comunica con un terminale il quale si limita a ricevere azioni da compiere, senza prendere decisioni, portando questo ad essere più economico dato che non ha bisogno di elevate prestazioni di calcolo e comunicazione. Esiste la possibilità di migliorare di gran lunga le prestazioni aumentando la potenza di calcolo e comunicazione del nodo centrale.
2. Non ha bisogno di ricevere informazioni dall'ambiente, in quanto è tutto calcolato e deciso all'avvio del software, dato che ci si basa su dati già verificati e affidabili. Questo diminuisce ulteriormente il rischio di malfunzionamenti da parte di sensori.
3. Calcoliamo a priori la strada che ogni agente deve compiere, evitando quindi l'obbligo che ci sia comunicazione tra i vari agenti; facendo così si limita drasticamente la possibilità di intoppi o malfunzionamenti.
4. La pianificazione della strada da percorrere è statica, scelta a priori e sicuramente quella più veloce; al contrario della pianificazione distribuita, la quale copre meglio la zona ma è più lenta a raggiungere la destinazione.
5. La strada calcolata a priori permette di andare il più velocemente possibile alla zona interessata, in quanto non serve monitorare ed esplorare tutto il campo, ma si va direttamente alla destinazione. Inoltre, essendo che viene trovata la strada più corta, non serve scavalcare tutta la montagna, ma se meno dispendioso, la si aggira.

QUALITA' DELLA PIANIFICAZIONE DISTRIBUITA:

1. Voronoi consente una partizione dell'area di copertura in maniera equa e veloce tra i diversi droni. Per di più, prevede anche la possibilità di non indicare i punti, e in quel caso, gli agenti si vanno automaticamente a suddividere sopra tutta l'area, minimizzando la sezione di controllo di ogni robot per aumentare l'efficacia del sistema.

2. Non ha bisogno di un link centrale che impedisce ordini, comporta quindi di essere meno ingombrante e più adattivo. Per merito di quest'ultima caratteristica, il sistema è veramente affidabile per quanto riguarda problematiche che possono insorgere mentre si sta svolgendo il compito, come ad esempio se un drone esaurisce l'autonomia o smette di funzionare oppure, al contrario, si vogliono aggiungere ulteriori agenti nel mezzo del monitoraggio per aumentare la qualità di ricerca.
3. Questa metodologia è conveniente nel caso in cui le coordinate avute riguardanti il terreno non siano corrette, oppure siano presenti ostacoli non identificati a priori o, più probabilmente, il terreno è mutato dopo la valanga. Il sistema distribuito, grazie ai sensori, permette un'ottima flessibilità e quindi una maggiore affidabilità ad evitare possibili schianti degli agenti, situazione che causerebbe notevoli malfunzionamenti nel caso di un sistema centralizzato.
4. Ha la possibilità di modificare la strada che sta percorrendo in base ad eventi che possono accadere durante lo svolgimento della missione. Non ha quindi bisogno che sia ricalcolato un intero percorso prima di procedere.

7 Conclusione e possibili future implementazioni

Tenendo in considerazione la situazione e la pericolosità delle valanghe, dato anche il rischio sempre maggiore di verificarsi episodi di questo tipo, causato dal cambiamento climatico in atto, il progetto si può descrivere come moderno e all'avanguardia.

Credo sia un dovere, nonché un'opportunità, sfruttare e sviluppare le nuove tecnologie per migliorare la sicurezza delle persone. Sicuramente la prevenzione è l'arma migliore, ma questo non vuol dire che sia infallibile, e quando catastrofi di questo tipo accadono, bisogna essere pronti per diminuire le possibilità che si trasformi in un tragico episodio che vede coinvolte le vite di persone.

Per concludere, si può affermare con sicurezza che il progetto ha risolto le problematiche esposte. Andando a tenere in considerazione diversi approcci, con la possibilità di scegliere la soluzione migliore per ogni ambiente ed evento.

Lavorando a lungo su questo progetto, sono emerse varie idee per possibili sviluppi futuri. Essendo il sistema veramente versatile e facile da implementare, può essere usato in occasioni veramente varie.

Possiamo partire dalla...

7.1 Collaborazione sistemi centralizzato e distribuito

Osservando dalla sezione precedente tutti i punti forti di ogni sistema, può venire in mente di creare un sistema ibrido, unendo la pianificazione centralizzata con quella distribuita. Con questa modalità, ogni sistema sfrutterebbe i suoi punti di forza, andando migliorare l'efficacia di tutta la tecnologia.

Un'idea che potrebbe venire in mente, è quella di calcolare a priori la destinazione di ogni agente per coprire l'area in maniera equa tra tutti i droni, dove questi saranno più concentrati nel punto di maggiore probabilità di trovare sopravvissuti. Per fare ciò si può utilizzare l'algoritmo di Voronoi, andando ad anticipare la posizione dei vari agenti e, di conseguenza, calcolare subito la convergenza del sistema.

Una volta trovata la destinazione, si utilizzerà il planning centralizzato per calcolare la strada più breve che ogni agente seguirà per arrivare a destinazione. In questa maniera si può essere sicuri che l'area di copertura sia divisa equamente tra tutti gli agenti, e questi arrivino nel minor tempo possibile nel punto preciso.

Intanto che i robot sono in viaggio per raggiungere la destinazione, si può utilizzare nuovamente il sistema distribuito per permettere ai droni di scambiarsi continui messaggi sulla reciproca posizione. Questo rende possibile conoscere in un breve intervallo di tempo se un agente ha smesso di funzionare (es. non invia più la sua posizione agli altri oppure invia la stessa posizione più volte, facendo capire che è rimasto bloccato), permettendo agli altri velivoli di coprire la sua area, garantendo così una copertura completa.

7.2 Ulteriori strade

I possibili futuri che può avere questo progetto sono moltissimi:

- Tipologie di robot: La nostra soluzione prevede l'utilizzo di robot terrestri e mobili, ma grazie alla flessibilità di ROS con lo scambio di messaggi e Px4 per il controllo automatico, nulla vieta la possibilità di integrare altri robot a questo progetto, così che siano incrementate le prestazioni di tutto il sistema. Si può pensare a velivoli acquatici oppure, spingendoci un po' oltre, un sistema di monitoraggio con satelliti in orbita (es. per rilevare incendi, tsunami o altri eventi).
- Situazioni: La versatilità della soluzione di questo progetto è un suo grande vantaggio: la tecnologia utilizzata può infatti essere utilizzata per altri ambienti, completamente distaccati da quello montuoso. Può essere impiegata per il ritrovamento di vittime dopo un disastro chimico, quindi implementare un algoritmo di Lloyd che tiene conto di ostacoli e ambienti chiusi; oppure, in un ambiente di guerra per la messa in sicurezza di una zona. Le situazioni possono essere tendenzialmente infinite.
- Intelligenza artificiale: Concentrandoci invece, non sul movimento, ma sul rilevamento delle persone, si può allenare un modello per riconoscere persone in situazioni dove altre tecnologie riscontrerebbero delle difficoltà.

Queste sopra elencate sono solamente alcune delle possibili strade future che questo progetto può avere, i modi per aumentare le prestazioni o aggiungere nuove funzionalità sono veramente limitati solamente dalla fantasia umana!

Bibliografia

- [1] Cortes, J., Martinez, S., Karatas, T., & Bullo, F. (n.d.). Coverage control for Mobile Sensing Networks. *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*. <https://doi.org/10.1109/robot.2002.1014727>
- [2] <https://www.slf.ch/it/valanghe/incidenti-e-valanghe-catastrofiche/statistiche-pluriennali.html>
- [3] <https://www.ilgiorno.it/cronaca/riscaldamento-globale-clima-grafico-1.6992416>
- [4] Boldrer, M., Palopoli, L., & Fontanelli, D. (2022). A unified Lloyd-based framework for multi-agent collective behaviours. *Robotics and Autonomous Systems*, 156, 104207. <https://doi.org/10.1016/j.robot.2022.104207>
- [5] <http://wiki.ros.org/en>
- [6] <https://gazebosim.org/docs>
- [7] <https://matplotlib.org/stable/index.html>
- [8] <https://doxygen.nl/>
- [8] https://en.wikipedia.org/wiki/K-means_clustering

Link GitHub con il codice codice del progetto:

Pianificazione Automatica Distribuita: https://github.com/albertor2000/Distributed_Coverage_Network