ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA
Sede di Forlì

Corso di Laurea in
INGEGNERIA AEROSPAZIALE
Classe L-9

ELABORATO FINALE DI LAUREA
In Meccanica del Volo

SVILUPPO DI UN SISTEMA DI MOTION PLANNING TRAMITE SENSORI OTTICI PER NAVIGAZIONE AUTONOMA IN AMBIENTI CONFINATI

CANDIDATO
Andrea Dal Pozzo

RELATORE
Prof. Fabrizio Giulietti

Anno Accademico 2023/2024
Sessione I – Luglio 2024

Ai miei genitori, per aver
sempre creduto in me.

Ai miei amici e colleghi, che ho avuto
la fortuna di avere accanto in questa
avventura.

A Giuseppe, sempre presente compagno di
viaggi mentali e preziosissimo amico.

Ad Alessandro, per
l'infinito sostegno che mi ha dato e
per l'amore che continua a darmi.

Sommario

Indice delle figure 4
Indice delle abbreviazioni 6
1. Abstract 7
2. Sistemi di navigazione autonoma 8
2.1. Controllo automatico di sistemi LTI 11
2.2. Sistemi di Data Fusion e controllori probabilistici 13
2.3. Assetto e posizione tramite unità inerziali 16
2.4. Assetto e posizione tramite unità ottiche 21
i. Feature Extraction and Visual-Inertial Odometry 23
ii. Local and Global Mapping and Optimization 24
iii. Loop Closure and Global Optimization 25
3. Architettura della piattaforma 27
3.1. Hardware utilizzato 27
3.2. Il sistema ROS 31
3.3. Il sistema di mappatura 33
3.4. Algoritmo di navigazione 35
i. Inizializzazione 37
ii. Ricezione della mappa e applicazione della sfocatura Gaussiana 38
iii. Pianificazione del percorso 42
iv. Controllore di posizione 46
4. Risultati sperimentali 50
5. Conclusioni 56
Ringraziamenti 58
Bibliografia 59
Appendice 1: Algoritmo di navigazione 61
Appendice 2: Procedura di installazione del sistema ROS 66
Appendice 3: Disegno tecnico Supporto Realsense 67

Indice delle figure

Figura 1: Sistema GN&C del rover della missione EXOMARS, Airbus Defence&Space ... 8
Figura 2: Struttura di un sistema di controllo in retroazione ... 11
Figura 3: Analisi della stabilità tramite root locus [31] ... 12
Figura 4: Architettura multisensore per ADCS [3] ... 13
Figura 5: Schematizzazione della stima dello stato attraverso filtro di Kalman ... 14
Figura 6: Rappresentazione tramite angoli di Eulero ... 17
Figura 7: Determinazione di assetto tramite sensori inerziali [29] ... 19
Figura 8: Struttura di un accelerometro MEMS ... 19
Figura 9: Andamento qualitativo del drift in un'unità inerziale ... 20
Figura 10: Formulazione del problema SLAM ... 21
Figura 11: Struttura di un algoritmo SLAM [21] ... 22
Figura 12: Riconoscimento dei landmark [30] ... 23
Figura 13: Generazione dell'odometria visivo-inerziale ... 23
Figura 14: Aggiunta delle features alla mappa globale ... 24
Figura 15: Loop closure ... 25
Figura 16: Scansione volumetrica interna del laboratorio CIRI Aerospaziale ... 26
Figura 17: LeoRover ... 27
Figura 18: Intel RealSense D455 Depth Camera ... 28
Figura 19: Determinazione della distanza tramite metodo stereografico [23] ... 29
Figura 20: Diagramma del flusso dati del rover ... 30
Figura 21: Tipico scambio di informazioni in un sistema ROS ... 31
Figura 22: Esempio di catena di acquisizione ottica ... 33
Figura 23: Funzionamento del nodo RTABMAP [11] ... 34
Figura 24: Difetti nella generazione del percorso ... 38
Figura 25: Convoluzione 2D [22] ... 39
Figura 26: Andamento e discretizzazione della parte positiva della distribuzione normale 39
Figura 27: Dettaglio della generazione dell'offset per altre configurazioni ... 40
Figura 28: Generazione del percorso prima e dopo l'applicazione del filtro ... 40
Figura 29: Sistema di riferimento al termine delle trasformazioni ... 41
Figura 30: A* in mappa di test, bersaglio singolo e bersagli multipli sequenziali ... 43
Figura 31: Composizione della B-spline a partire dalle singole componenti [24] ... 45

Figura 32: Variazione delle caratteristiche della spline interpolante al variare del parametro di smoothing 45
Figura 33: Schema a blocchi di un controllore PID 46
Figura 34: Passaggio da immagine acquisita, a nuvola di punti 3D, a mappa 2D 50
Figura 35: Interfaccia RVIZ e del nodo di navigazione 51
Figura 36: Sovrimpressione della traiettoria seguita dal rover con quella generata dal controllore..... 52
Figura 37: Comandi generati dal controllore di direzione..... 53
Figura 38: Comandi generati dal controllore di posizione 53

Indice delle abbreviazioni

AHRS	Attitude and Heading Reference System
ADCS	Attitude Determination and Control System
CIRI	Centro Interdipartimentale di Ricerca Industriale
DAP	Digital Auto Pilot
DCM	Direction Cosine Matrix
EKF / UKF	Extended / Unscented Kalman Filter
FPS	Frames Per Second
GDOP	Geometric Dilution Of Precision
GDL	Gradi Di Libertà
GNC	Guidance, Navigation and Control
GNSS	Global Navigation Satellite System
HIL	Hardware In the Loop
IMU	Inertial Measurement Unit
LIDAR	Light Detection And Ranging
LTI	Linear Time-Invariant systems
LTS	Long Term Support
LORAN	Long Range Navigation
MEMS	Micro Electro-Mechanical System
OOP	Object-Oriented Programming
PID	Controllore Proporzionale-Integrale-Derivativo
RGB-D	Red, Green, Blue – Depth
ROS	Robot Operating System
RPI	Raspberry PI
RTAB-MAP	Real-Time Appearance-Based Mapping
RVIZ	Robot Visualization
SLAM	Simultaneous Localization And Mapping
SIL	Software In the Loop
TOF	Time Of Flight
UI	User Interface
VIO	Visual Inertial Odometry
VOR	VHF Omnidirectional Ranger

1. Abstract

I progressi nei sistemi di acquisizione ed elaborazione dati hanno permesso lo sviluppo di sistemi di navigazione ottica sempre più affidabili e di semplice implementazione.

Si esaminerà in questa tesi un'applicazione di un sistema di localizzazione e navigazione autonomo funzionante tramite metodi ottici. A tal proposito si è realizzata una piattaforma mobile autonoma per navigazione planare indoor, da qui in poi denominata rover.

Sfruttando le informazioni rese disponibili da una telecamera di profondità è possibile procedere alla realizzazione di una mappa tridimensionale dell'ambiente in esame. Effettuata manualmente la scansione, il sistema ne estrae una sezione bidimensionale, sulla quale è possibile indicare il punto da raggiungere. In autonomia viene determinata quindi la traiettoria ottimale da seguire, minimizzando la distanza da percorrere e tenendo conto della presenza di ostacoli. Infine, un controllore di velocità di bordo viene impiegato per attuare la navigazione del rover verso la destinazione.

Dal momento che gli ostacoli non vengono rilevati in tempo reale, l'architettura del software è detta di pianificazione del moto (motion planning): al momento, una volta ricavata la traiettoria questa non è più modificabile. Si descrivono nel capitolo conclusivo alcuni spunti per sviluppi futuri del sistema di navigazione, tra cui l'implementazione di un sottosistema di evitamento ostacoli (obstacle avoidance).

Il software è stato interamente sviluppato in linguaggio Python 3.12 ed eseguito su hardware off-the-shelf. La sperimentazione è stata svolta presso la struttura del CIRI Aerospaziale di Forlì per un periodo di circa quattro mesi.

Il sistema completo si è dimostrato essere sufficientemente prestante per svolgere piccole missioni in relativa autonomia sia in ambienti limitati che all'aperto su terreni regolari, è richiesta tuttavia un'ulteriore rifinitura prima di poter ottenere un prodotto totalmente autonomo.

2. Sistemi di navigazione autonoma

Si definisce navigazione lo studio dell'evoluzione del moto di un veicolo nel tempo. La necessità di un sistema automatico di navigazione può essere dovuta a molteplici fattori: può essere necessario pilotare veicoli che richiedano una risposta estremamente rapida come missili o veicoli spaziali; oppure veicoli operanti in ambienti ostili o remoti, dove la presenza di un operatore sul campo non sia possibile o dove il ritardo di trasmissione sia tale da non consentire un pilotaggio da remoto.

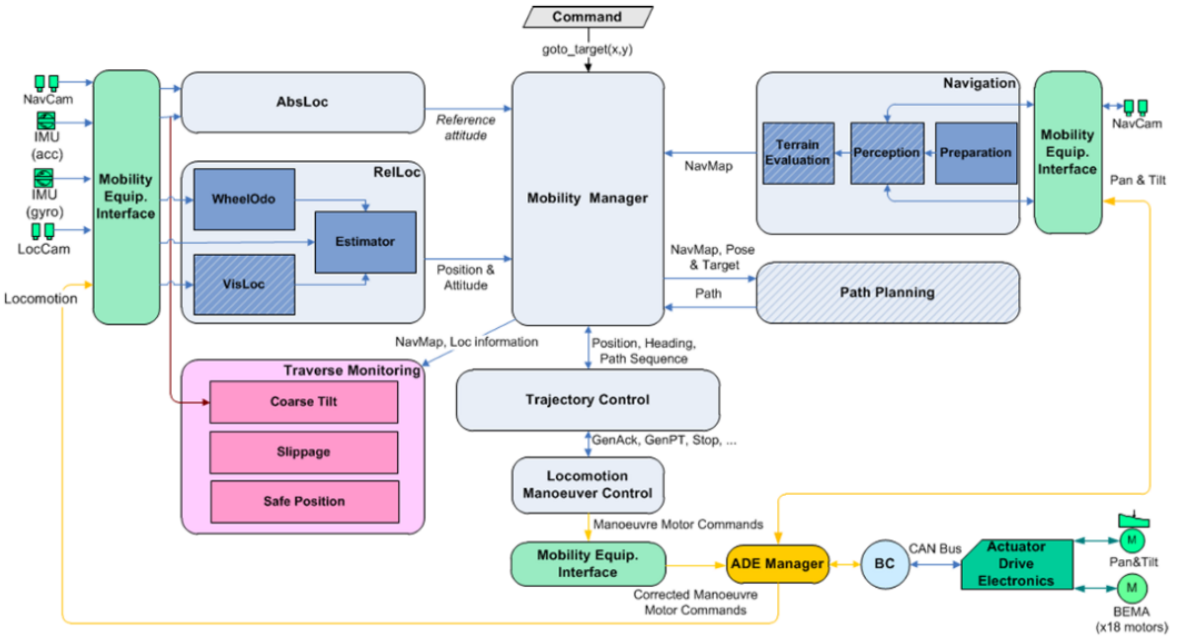


Figura 1: Sistema GN&C del rover della missione EXOMARS, Airbus Defence&Space

I primi sistemi di controllo automatico sono apparsi durante la prima rivoluzione industriale sotto forma di regolatori di giri per motori a vapore: uno dei dispositivi più diffusi fu il regolatore centrifugo di Watt a masse rotanti. Con l'avvento dei sistemi elettronici a valvole prima e a transistor poi, le dimensioni delle unità di controllo si sono ridotte a sufficienza da poterne considerare l'impiego su veicoli e velivoli. Uno dei principali esempi è rappresentato dal sistema DATAR della marina reale canadese per la tracciatura autonoma di bersagli, entrato in funzione nel 1953 [1].

Da allora sono stati sviluppati sistemi di controllo automatico sempre più complessi fino ad arrivare ai moderni DAP (Digital Auto Pilot) computerizzati, operanti non più tramite un controllore elettronico discreto ma attraverso un software. L'impiego di sistemi digitali consente di manipolare in tempo reale ed in maniera automatica i parametri di funzionamento per adattare il controllore alle esigenze della missione prevista.

Un ulteriore vantaggio della digitalizzazione è dovuto alla semplicità di collaudo dei sistemi: è possibile realizzare delle simulazioni generando un modello digitale del sistema controllato e collaudando il software in ambiente virtuale. Questo approccio è detto SIL (software in the loop) e consente il monitoraggio istante per istante dei parametri di funzionamento. Nel caso non fosse possibile realizzare un modello completo del sistema se ne possono modellare solo delle parti, comandando direttamente le rimanenti. Si ha di conseguenza un metodo di test detto HIL (hardware in the loop).

Le componenti alla base di un sistema di navigazione automatico sono dette di Guida, Navigazione e Controllo. Le tre azioni svolgono compiti complementari per la generazione e l'inseguimento di una traiettoria prefissata [2].

- Guida: determinazione del percorso ottimale per raggiungere un obiettivo a partire dalla posizione attuale e dalla conoscenza della mappa del volume di navigazione. Si determinano l'assetto e le caratteristiche del moto desiderato istante per istante, come i valori di velocità, accelerazione e derivate successive della posizione, l'orientazione spaziale e altri parametri, il tutto sulla base dei vincoli progettuali e delle capacità degli attuatori. Altri sistemi secondari complessi dotati di movimento, come bracci robotici o utensili specifici, solitamente sono governati da un sistema di pianificazione parallelo in costante comunicazione con il sistema primario.
- Navigazione: sulla base dei dati ricavati dall'insieme di sensori installati sul veicolo si determina il relativo vettore di stato, ovvero le componenti dell'accelerazione, della velocità, della posizione e l'assetto nello spazio ricavate istante per istante. Questo modulo, chiamato anche di determinazione di assetto (ARS, Attitude Reference System), comprende il sottosistema di acquisizione, di condizionamento e di elaborazione dei dati.
- Controllo: ottenuti i dati di orientazione spaziale e della missione da seguire, vengono generati i comandi necessari agli attuatori per modificare lo stato del veicolo. Il controllo è tipicamente effettuato in catena chiusa misurando il feedback dei sensori. Il sottosistema di controllo comunica con il nodo di navigazione per assicurarsi che non si creino delle situazioni di auto-compenetrazione o impatto con ostacoli, segnalando eventuali conflitti all'unità di guida per la progettazione di una traiettoria alternativa.

I sensori più diffusi per il calcolo dell'assetto sono basati sulla misura diretta del moto del veicolo. Dette unità inerziali (IMU, Inertial Measurement Unit), sono composte principalmente da accelerometri, per la misura delle accelerazioni inerziali lungo i tre assi di traslazione e da giroscopi, per la misura delle velocità angolari.

Questo tipo di determinazione dell'assetto basata su misurazioni ripetute è detta di dead reckoning, o navigazione stimata. Nei sistemi più comuni le unità inerziali sono spesso accoppiate con sensori barometrici per la determinazione della quota e, se applicabile, di magnetometri per ricavare l'orientazione assoluta rispetto ai poli magnetici terrestri.

Per applicazioni astronautiche a quota elevata sono di interesse anche i sistemi astro-inerziali, che ricavano l'assetto analizzando la posizione relativa delle stelle fisse.

Altri sistemi di navigazione si basano sul ritardo di trasmissione delle onde radio (ad esempio, LORAN-C per navigazione marittima, i sistemi GNSS o gli impianti VOR aeroportuali), altri ancora sul calcolo della distanza da determinati oggetti tramite onde sonore, radio o luminose (rispettivamente SONAR, RADAR e LIDAR).

La metodologia impiegata nel progetto presentato da questa tesi impiega il riconoscimento in tempo reale delle caratteristiche ambientali basandosi sull'analisi ottica dei fotogrammi.

Il campo della navigazione tramite sensori ottici è relativamente giovane rispetto agli altri metodi di navigazione ma dimostra una flessibilità e adattabilità molto elevate. La necessità di un sistema ottico nasce in applicazioni per le quali non è possibile fare riferimento a strutture satellitari o ad altra infrastruttura pesante ma dove è comunque richiesta una qualità elevata dei dati di posizionamento. Grazie al progredire della tecnologia sono disponibili sul mercato telecamere di profondità e computer per l'elaborazione dati relativamente economici in confronto ai sistemi radio (in particolare radar e radiofari) ed estremamente compatti, impiegabili anche in velivoli a peso ridotto.

Nei paragrafi seguenti verrà valutata la teoria dei sistemi di navigazione e controllo per sensori tradizionali e ottici. Dal momento che la generazione della traiettoria ottimale è un processo che deve essere adattato al particolare sistema in uso, il modulo di guida sarà trattato direttamente nei paragrafi ii e iii del capitolo 3.4.

2.1. Controllo automatico di sistemi LTI

Si definiscono le basi dei sistemi di controllo automatico. Per sistema di controllo automatico si intende un insieme di strumenti e tecniche in grado di raggiungere un obiettivo prefissato, tipicamente la regolazione di una variabile in un impianto, in maniera automatica [5].

Nel progetto di tesi sono presenti due controllori automatici principali, uno dedito all'analisi del flusso ottico registrato dalla telecamera di navigazione e uno per il pilotaggio del rover attraverso il volume di controllo. In quest'ultimo caso, ottenuta una stima dello stato del sistema, si desidera procedere alla movimentazione del sistema autonomo. Si introduce a tale scopo un controllore automatico lineare tempo-invariante (LTI).

La condizione di linearità è rispettata se è valido il principio di sovrapposizione degli effetti: dati due segnali $x_1(t), x_2(t)$, per ogni coefficiente scalare reale a, b deve essere valida la relazione $a \cdot f(x_1(t)) + b \cdot f(x_2(t)) = f\big(a \cdot x_1(t) + b \cdot x_2(t)\big)$.

Un sistema è invece detto tempo-invariante se il comportamento del sistema è indipendente dal tempo. Data una variabile controllata $x(t)$ e la relativa funzione di uscita $y(t)$, il sistema è detto tempo-invariante se un ritardo nella funzione d'ingresso comporta uno stesso ritardo nella funzione d'uscita: $f\big(x(t+\ \delta)\big) = y(t+\ \delta)\ \forall\ x, \delta, t$.

La metodologia di controllo più rudimentale è detta a catena aperta, dove le variabili in ingresso vengono manipolate dal controllore e restituite direttamente in uscita.

L'estrema semplicità di questo tipo di architettura rende il controllo in catena aperta utilizzabile solamente per sistemi basilari e non critici: l'effetto dei disturbi sui canali di ingresso e uscita non viene compensato in alcun modo, la mancanza del controllo dello stato del sistema controllato non consente la correzione automatica degli errori di attuazione.

Per consentire un controllo accurato si possono impiegare sistemi retroazionati, in catena chiusa, in cui la variabile controllata non è funzione esclusivamente dello stato precedente ma anche di quello attuale.

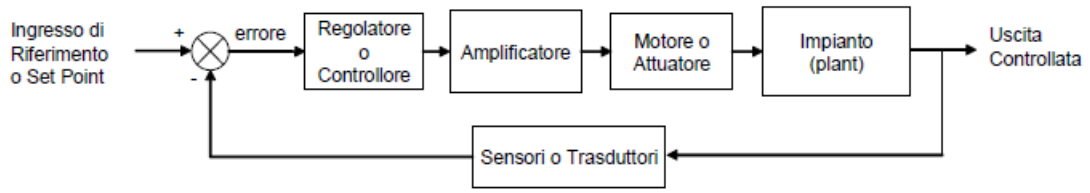


Figura 2: Struttura di un sistema di controllo in retroazione [4]

In un generico sistema di controllo automatico ad anello chiuso la risposta dell'impianto da controllare viene monitorata e confrontata con un valore di riferimento. Il valore ottenuto da questa differenza, detto errore, viene utilizzato per comandare la variabile di controllo e alterare la risposta del controllore. In un sistema a retroazione negativa, una perturbazione a valle del controllore viene riportata a monte invertita di segno. L'effetto complessivo è un'azione stabilizzante, che tende ad annullare l'errore a regime.

Per sistemi retroazionati LTI si dimostra che esiste sempre almeno uno stato di equilibrio indipendente dall'entità delle perturbazioni agenti sul sistema. In particolare, se tutti i poli della funzione di trasferimento sono a parte reale negativa, allora il sistema è asintoticamente stabile. A tal proposito è possibile valutare la stabilità esaminando il luogo delle radici del sistema.

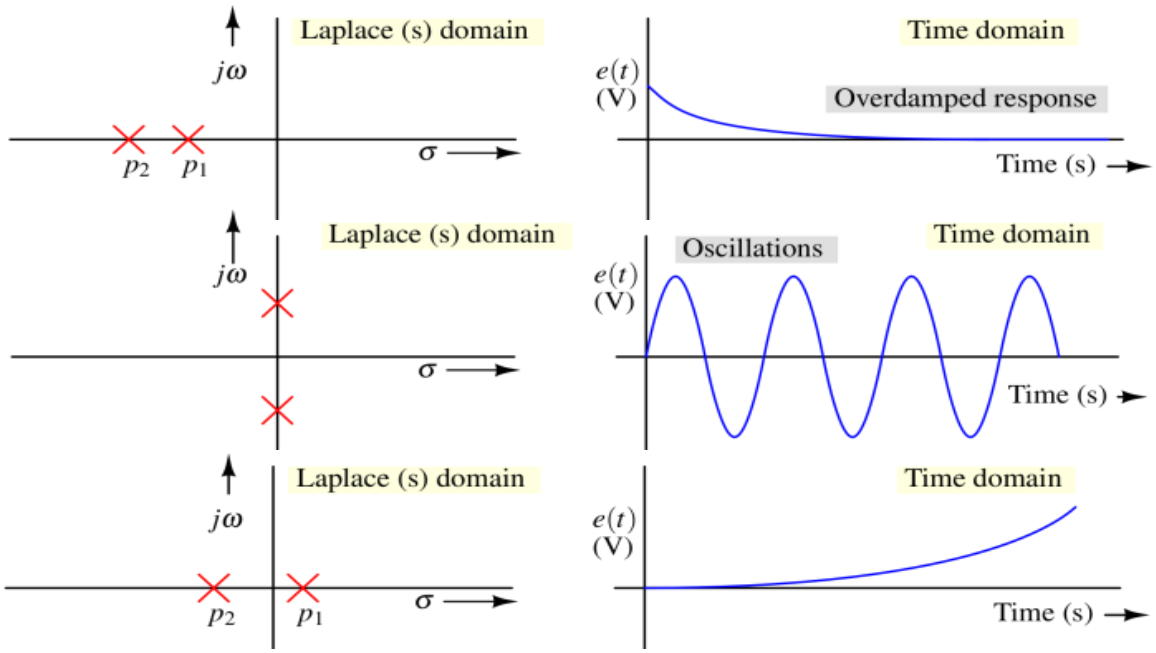


Figura 3: Analisi della stabilità tramite root locus [31]

Per il controllo della posizione e dell'orientazione si è optato per l'impiego di un controllore PID, il cui principio di funzionamento è illustrato nel dettaglio al termine del capitolo 3.4.

Data la relativa lentezza e regolarità del processo controllato, il controllore è approssimabile ad un sistema ILUL (Ingresso Limitato – Uscita Limitata), ovvero un sistema in cui ad ogni segnale di ingresso che non superi una certa soglia in modulo corrisponda un segnale limitato in uscita. In questi casi è dimostrabile che la condizione di stabilità asintotica è sufficiente per garantire la stabilità globale del sistema di controllo.

2.2. Sistemi di Data Fusion e controllori probabilistici

In un sistema robotizzato vengono usualmente impiegati più tipi di sensori allo stesso tempo per la misura di una stessa grandezza.

Oltre a fornire una resistenza ai guasti, attraverso questa ridondanza è possibile unire insieme le informazioni allo scopo di avere dei segnali nei quali la componente di errore complessiva (dovuta a rumore o ad altre fonti) viene diminuita. Questo approccio è detto di sensor o data fusion.

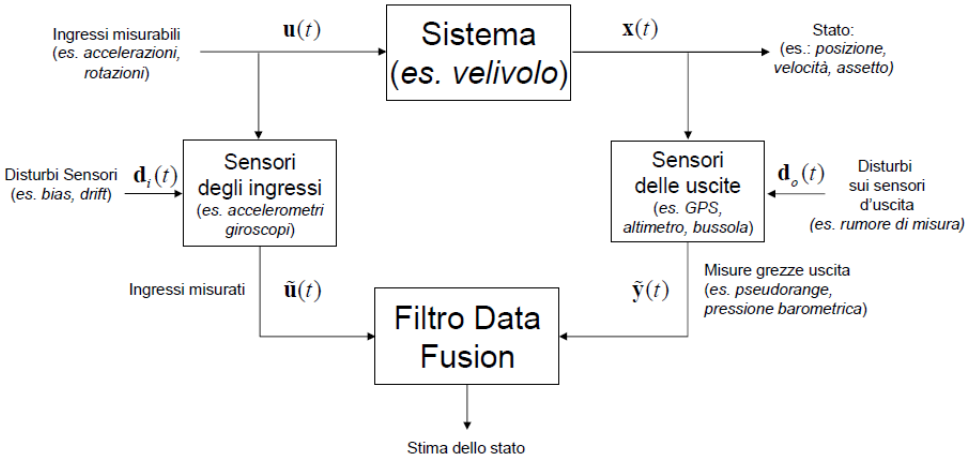


Figura 4: Architettura multisensore per ADCS [3]

Senza perdita di generalità, viene brevemente riportato il funzionamento dei principali sistemi di sensor fusion, limitando la stima dello stato per segnali monodimensionali. Per applicazioni reali il procedimento è estensibile in dimensioni di ordine maggiore.

A partire dalle misure grezze, provenienti direttamente dai sensori, un filtro di data fusion è in grado di effettuare una stima del valore reale misurato limitando errori e disturbi. Il filtro più diffuso a questo scopo è il filtro di Kalman.

Si ipotizza che i dati misurati siano segnali stocastici reali, discreti e finiti. Un filtro di Kalman è un filtro non-lineare e tempo-variante del tipo osservatore identità dello stato, in grado cioè di stimare il valore di stato $\widehat{x_n}$ a partire dalle variabili osservate ad un determinato istante y_n e dalla matrice di osservabilità Q [3]. Nella teoria dei controlli automatici, la matrice di osservabilità rappresenta una stima linearizzata del modello del sistema analizzato.

La relazione che lega lo stato alle variabili osservate è indicata come:

$$\widehat{x_n} = Q^{-1} y_n.$$

Per sistemi iterativi, si dimostra che per un sistema completamente osservabile (matrice di osservabilità a determinante non nullo) lo stimatore di Kalman è stabile e ottimo, mentre asintoticamente è sempre stabile.

Questo tipo di controllori è detto probabilistico, in quanto prende in esame non solo i dati forniti ma anche le caratteristiche statistiche di ognuno di essi. In uno stimatore di Kalman convenzionale le caratteristiche del sistema controllato vengono sintetizzate (tramite la matrice di osservabilità) in una relazione lineare a coefficienti reali: la stima dello stato diventa quindi un problema di interpolazione lineare, aggiornata passo per passo per rimuovere errori e allinearla alla condizione reale.

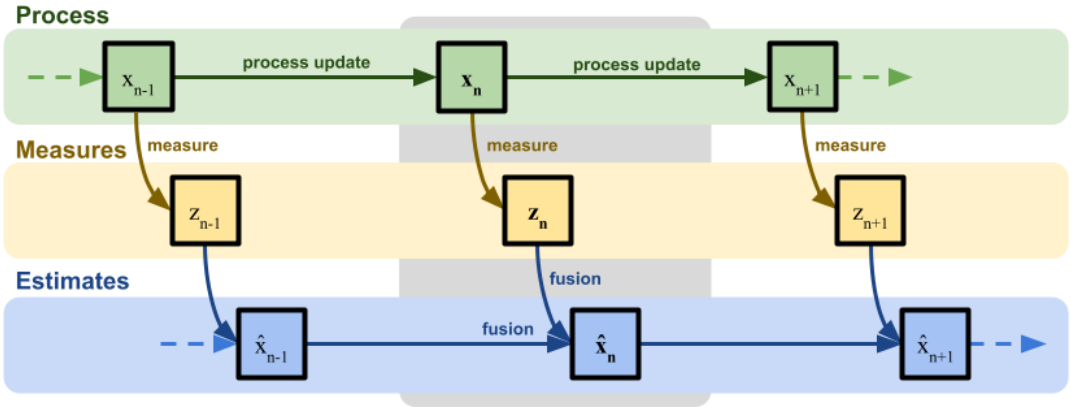


Figura 5: Schematizzazione della stima dello stato attraverso filtro di Kalman

Dal momento che la maggior parte dei processi fisici controllati nei sistemi GN&C è del tipo non lineare, l'operazione di linearizzazione delle caratteristiche porta inevitabilmente ad un accumulo significativo di errori di calcolo. Per ovviare al problema, l'implementazione globalmente più diffusa è la versione discreta del filtro di Kalman Esteso (EKF).

Nei sistemi EKF, ai dati rilevati dai sensori si aggiunge un'ulteriore informazione: il rumore dei processi misurati è un segnale reale di tipo aleatorio, tuttavia, trattandosi di processi continui, con ottima approssimazione il valore misurato segue la distribuzione normale. Di conseguenza, anche l'errore totale della combinazione dei due segnali dovrà seguire la distribuzione normale. Per questo motivo il filtro di Kalman è uno stimatore appartenente alla famiglia dei filtri Gaussiani [4].

Conoscendo quindi la matrice di covarianza dei due segnali è possibile correggere la stima iniziale per ottenere un modello pseudo-lineare, ottenuto tramite scomposizione in serie di Taylor, che rispetti i nuovi vincoli. Si tratta del passaggio computazionalmente più costoso,

in quanto è necessario procedere al calcolo della derivata prima del segnale nel punto di approssimazione.

È da notare che quanto detto perde di validità per segnali discontinui. Ogni volta che la misura è ripetuta nel loop di acquisizione, dal momento che si stanno rilevando segnali discreti, si avrà con elevata probabilità una discontinuità. La conseguente perdita di derivabilità comporta una degradazione nella stima. Ipotizzando un segnale sufficientemente regolare, il filtro può essere in grado di compensare questa azione procedendo per iterazioni successive e stabilizzandosi su un valore medio istantaneo.

Se si sta impiegando un sensore con una frequenza di aggiornamento particolarmente bassa, ma comunque con la garanzia che la variabile misurata sia regolare, può rendersi necessario applicare delle tecniche di interpolazione numerica, affinando artificialmente il segnale.

Nonostante il filtro di Kalman discreto non sia più uno stimatore ottimo, molto suscettibile a errori di modellazione del sistema, le performance più che ragionevoli e l'adattabilità lo hanno reso di fatto lo standard nei moderni sistemi di data fusion. Per segnali fortemente discontinui è utilizzata la versione UKF del filtro (Unscented Kalman Filter) che, al prezzo di un costo computazionale più elevato, sfrutta una campionatura selettiva per l'ottenimento di un segnale medio approssimabile come continuo.

Altri filtri largamente diffusi si basano su algoritmi di tipo prettamente probabilistico, come il filtro antiparticellare (particle filter), o sequenziale di Monte Carlo, basato sull'inferenza bayesiana [4]. Si presuppone in questi metodi che i segnali controllati non seguano la distribuzione Gaussiana, rendendo necessaria la modellazione della curva di probabilità analizzando lo storico dei dati misurati.

2.3. Assetto e posizione tramite unità inerziali

Si analizza ora la metodologia di misura più diffusa, grazie al costo ridotto e alla semplicità del metodo: la navigazione stimata inerziale.

Allo scopo di questa tesi, i sistemi di riferimento impiegati sono il sistema assi-corpo $\mathcal{F}_b$, solidale al veicolo, e il sistema di riferimento pseudo-inerziale tangente alla superficie terrestre $\mathcal{F}_t$. Data la scala dell'ambiente di navigazione non si compie un errore significativo introducendo l'ipotesi di terra piatta e non rotante, trascurando così gli effetti di trascinamento come l'accelerazione di Coriolis. All'avvio del rover i due sistemi di riferimento saranno coincidenti. La configurazione impiegata è del tipo strapdown, ovvero con l'unità inerziale solidale al telaio del veicolo, al contrario di sistemi stabilizzati giroscopicamente.

Per ricavare la posizione e l'assetto, nei sistemi basati su unità inerziali le accelerazioni e le velocità angolari vengono integrate iterativamente tramite metodi numerici.

Per quanto riguarda la posizione si può procedere all'integrazione diretta: nota l'accelerazione inerziale $a_i = \left(a_x, a_y, a_z\right)_{\mathcal{F}_t}$ e le condizioni iniziali temporali e cinematiche $t_0, (v_0, x_0)_{\mathcal{F}_t}$ si ricavano:

$$v_i = v_0 + \int_{t_0}^{t_0+\delta t} a_i \, dt$$

$$p_i = p_0 + \int_{t_0}^{t_0+\delta t} v_i dt = p_0 + \int_0^{\delta t} \left(v_0 + \int_0^{\delta t} a_i \, dt \right) dt$$

Se la frequenza di acquisizione dei dati $f = \frac{1}{\delta t}$ è sufficientemente elevata è possibile introdurre l'ipotesi di linearità per l'andamento dell'accelerazione. Trascurando gli effetti superiori al secondo ordine, al termine della linearizzazione si avrà quindi:

$$p_i = p_0 + v_0 \delta t + a_i \frac{\delta t^2}{2} + \mathcal{O}(\delta t^2)$$

Si noti che l'accelerazione integrata a_i è rappresentata rispetto al sistema di riferimento inerziale, tuttavia, i dati misurati dall'accelerometro sono espressi in relazione agli assi corpo. È necessario risalire all'orientazione spaziale relativa tra i sistemi di riferimento.

Si definisce a questo proposito la matrice di assetto, A, che identifica le componenti della base del sistema di riferimento mobile rispetto a quello fisso. Un vettore nello spazio, riferito al sistema di riferimento assi terra, può essere espresso in termini di assi corpo tramite l'applicazione della matrice di assetto:

$$b_{\mathcal{F}_b} = A \cdot b_{\mathcal{F}_t} \Rightarrow a_i = A^{-1} \cdot a_{i_{\mathcal{F}_b}}$$

Per indicare l'orientazione di un oggetto nello spazio sono spesso impiegati gli angoli di Eulero, ovvero una sequenza di tre successive rotazioni intorno agli assi del sistema di riferimento:

- angolo di rollio φ rispetto all'asse longitudinale,
- angolo di beccheggio θ rispetto all'asse laterale,
- angolo di imbardata ψ rispetto all'asse normale.

Quando, come in questo caso, le rotazioni avvengono sempre intorno ad assi differenti (321), si parla di rotazioni asimmetriche.

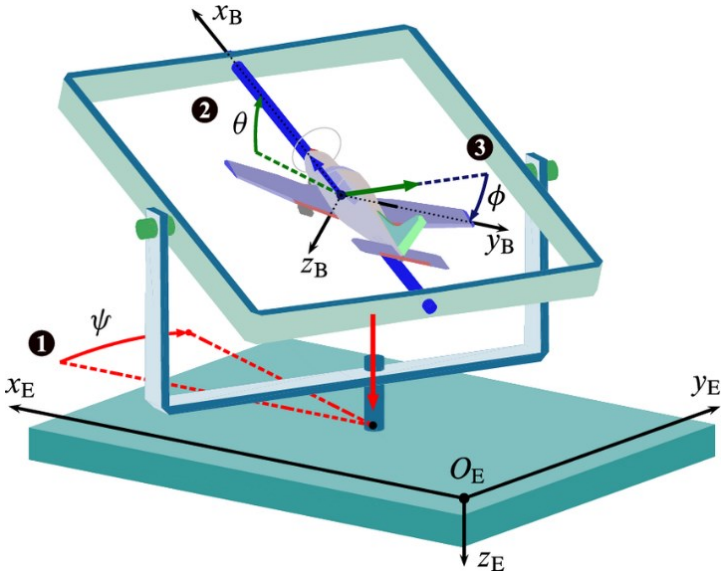


Figura 6: Rappresentazione tramite angoli di Eulero

La matrice di assetto può essere definita come la composizione di tre rotazioni elementari in termini degli angoli di Eulero. Per brevità, si indicano le operazioni trigonometriche seno, coseno e tangente con le diciture s, c, t.

$$A_{321} = R_1(\psi)R_2(\theta)R_3(\varphi) = \begin{bmatrix} c\psi c\theta & s\psi c\theta & -s\theta \\ s\theta s\varphi c\psi - s\psi c\varphi & c\varphi c\psi + s\theta s\varphi s\psi & s\varphi c\theta \\ s\varphi s\psi + s\theta c\varphi c\psi & s\theta c\varphi s\varphi - s\varphi c\psi & c\varphi c\theta \end{bmatrix}$$

Il problema principale dell'espressione dell'assetto tramite angoli di Eulero è la presenza di posizioni singolari: nel caso due assi rotanti di rotazione dovessero allinearsi, come ad esempio con una rotazione di 90° sull'asse di beccheggio, una rotazione sull'asse di rollio sarebbe indistinguibile da una sull'asse di imbardata, con la conseguente perdita di un grado di libertà. Con due righe uguali, la matrice di assetto in questo caso avrebbe determinante nullo, di conseguenza non sarebbe più possibile procedere al calcolo della matrice inversa.

Considerando invece l'evolversi del moto nel tempo, l'assetto in seguito a rotazioni successive sarà dato dalla moltiplicazione delle rispettive matrici di rotazione tra loro:

$$x_{\mathcal{F}_t} = A^{-1} x_{\mathcal{F}_b} = (A_1 A_2 \cdots A_n)^{-1} x_{\mathcal{F}_b}$$

Per rimuovere la presenza della singolarità e alleggerire il calcolo della matrice di rotazione (una composizione di n matrici comporta $27n$ moltiplicazioni di funzioni trigonometriche) la maggior parte degli algoritmi di determinazione di assetto sfrutta la rappresentazione tramite quaternioni, vettori a quattro componenti nella forma $\boldsymbol{q} = a + b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k}$, dove a, b, c, d rappresentano numeri reali e 1, i, j, k rappresentano i vettori unitari che formano la base dello spazio vettoriale. È rispettata l'identità $i^2 = j^2 = k^2 = ijk = -1$.

La relazione tra quaternioni e angoli di Eulero è calcolabile come:

$$q = \begin{bmatrix} c(\varphi/2)c(\theta/2)c(\psi/2) + s(\varphi/2)s(\theta/2)s(\psi/2) \\ s(\varphi/2)c(\theta/2)c(\psi/2) - c(\varphi/2)s(\theta/2)s(\psi/2) \\ c(\varphi/2)s(\theta/2)c(\psi/2) + s(\varphi/2)c(\theta/2)s(\psi/2) \\ c(\varphi/2)c(\theta/2)s(\psi/2) - s(\varphi/2)s(\theta/2)c(\psi/2) \end{bmatrix}$$

Nonostante con il passaggio in quaternioni si perda l'interpretazione fisica intuitiva, l'impiego dei quaternioni rimuove la singolarità, mentre la composizione di rotazioni successive diventa un problema di moltiplicazione tra vettori a quattro elementi (16 contro 27 operazioni), senza funzioni trigonometriche.

Tornando alla risoluzione del problema di assetto, misurati i ratei angolari rispetto agli assi principali attraverso l'unità inerziali, questi vengono integrati in maniera simile alle accelerazioni per ottenere gli angoli di Eulero. La conversione dalle velocità angolari misurate $[p, q, r]$ al rateo di variazione degli angoli di Eulero è ricavabile attraverso la risoluzione di un sistema di equazioni differenziali ordinarie:

$$\begin{bmatrix} \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & s(\varphi)/c(\theta) & c(\varphi)/c(\theta) \\ 0 & c(\varphi) & -s(\varphi) \\ 1 & s(\varphi)t(\theta) & c(\varphi)t(\theta) \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

A prescindere dal metodo impiegato, con le adeguate trasformazioni viene generata la matrice di assetto corretta e, applicandola finalmente alle misurazioni degli accelerometri, si possono ottenere anche le accelerazioni in assi terra [7] [6].

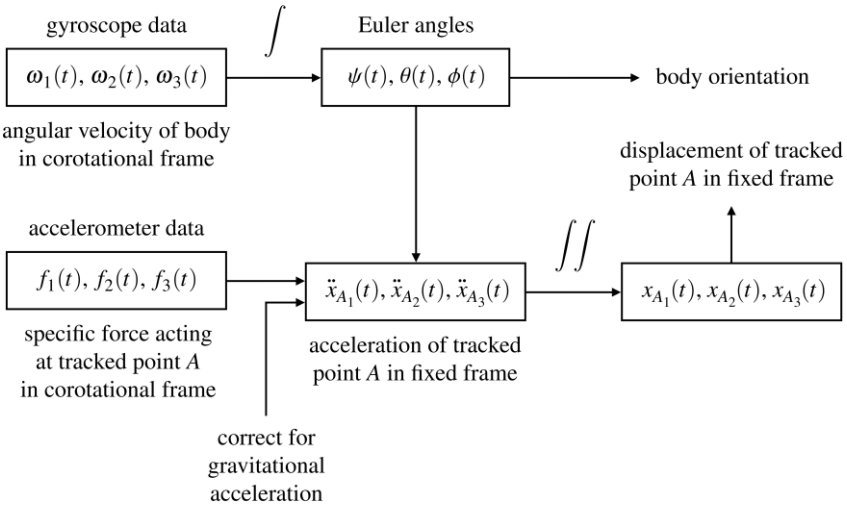


Figura 7: Determinazione di assetto tramite sensori inerziali [29]

I metodi di navigazione basati solamente su sistemi inerziali non sono in grado di fornire un'elevata precisione nel tempo. Vi sono infatti una serie di fattori che portano ad una progressiva degradazione della qualità dei dati ricavati, tra cui la deriva degli accelerometri e l'accumulo dell'errore di misurazione.

All'interno di un accelerometro si ha genericamente una massa oscillante libera di muoversi lungo l'asse di misura. Mentre in sistemi discreti si impiega un sensore di forza collegato alla massa, negli accelerometri integrati si fruttano tecnologie MEMS: una sottile lamina metallica libera di muoversi forma un'armatura in un sistema di condensatori. In seguito ad un movimento si avrà una variazione nel valore della capacità complessiva e, di conseguenza, del campo elettrico, direttamente proporzionale all'accelerazione subita.

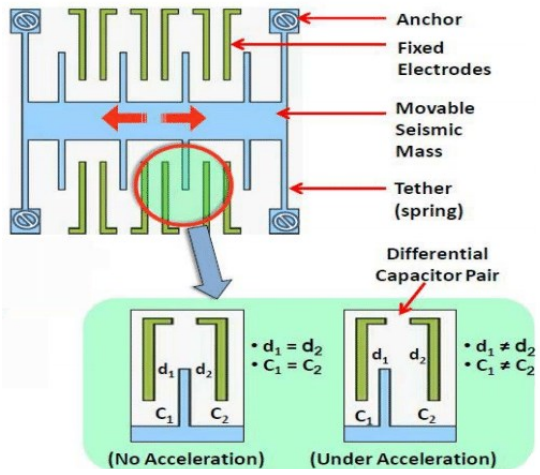


Figura 8: Struttura di un accelerometro MEMS

A causa di effetti dovuti a deformazioni o rapide vibrazioni, fluttuazioni di temperatura del dielettrico o di tensione durante il funzionamento, la misura della capacitanza interna può subire delle piccole variazioni risultanti in alterazioni nel valore di output.

Dal momento, inoltre, che le misurazioni avvengono ad intervalli discreti di tempo, le accelerazioni misurate non saranno totalmente fedeli alla realtà ma presenteranno un certo errore, che sarà integrato a sua volta. Con il tempo, l'accumulo di questi errori farà differire sensibilmente i dati osservati da quelli reali, portando ad un calo della qualità delle misure. Quest'effetto prende il nome di deriva, o drift.

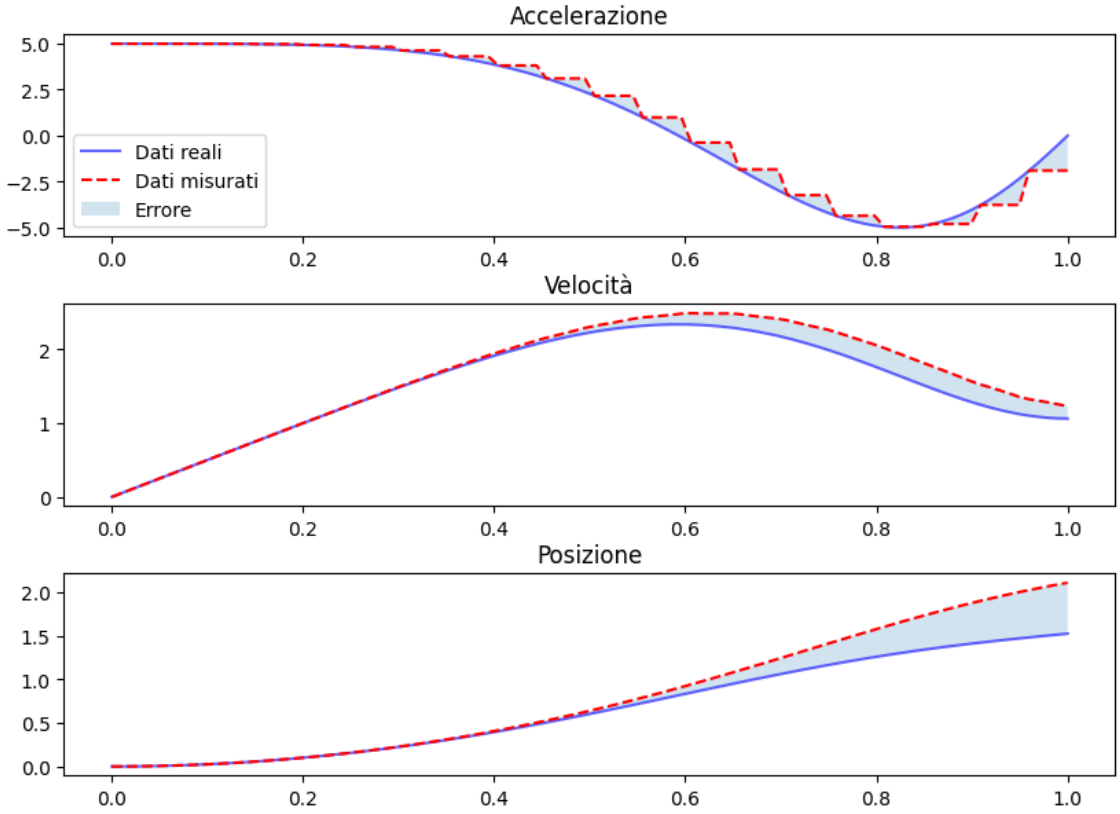


Figura 9: Andamento qualitativo del drift in un'unità inerziale

L'effetto può essere compensato ricalibrando ciclicamente le unità inerziali basandosi sulle informazioni provenienti da altri sensori ritenuti affidabili.

Nel software di controllo del progetto le operazioni di stima dell'assetto sono state eseguite tramite l'impiego del filtro spaziale di Madgwick [8], ora parte della suite di filtri Fusion della X-IO Technologies.

2.4. Assetto e posizione tramite unità ottiche

Il metodo di navigazione impiegato allo scopo di questa tesi è la localizzazione tramite metodi ottici. Al contrario dei classici metodi di localizzazione deterministici, la navigazione ottica è un problema di tipo probabilistico. Si esamina in questa sezione il funzionamento base del problema di localizzazione SLAM (Simultaneous Localization and Mapping) [10].

Il processo SLAM è un sistema grazie al quale si rende possibile la costruzione di una mappa dell'ambiente circostante al veicolo e la stima in contemporanea della posizione dello stesso mediante lo studio delle caratteristiche ambientali e l'identificazione di punti di riferimento (landmarks).

La seguente procedura è tratta da Durrant-Whyte et al. [11]

Si consideri un sistema robotico in movimento attraverso un ambiente. Si definiscono i vettori $\boldsymbol{x_k}, \boldsymbol{u_k}, \boldsymbol{m_i}$ *e* $\boldsymbol{z_{ik}}$ e i relativi set $\boldsymbol{X_{0:k}}, \boldsymbol{U_{0:k}}, \boldsymbol{m}, \boldsymbol{Z_{0:k}}$ che ne contengono lo storico.

Per ogni istante di tempo k, $\boldsymbol{x_k}$ rappresenta il vettore di stato del veicolo; $\boldsymbol{u_k}$ il vettore delle azioni di controllo applicate all'istante $k-1$, $\boldsymbol{m_i}$ rappresenta il vettore che descrive la posizione relativa dell'i-esimo landmark e, infine, $\boldsymbol{z_{ik}}$ identifica l'osservazione da parte del veicolo dell'i-esimo landmark all'istante di tempo k.

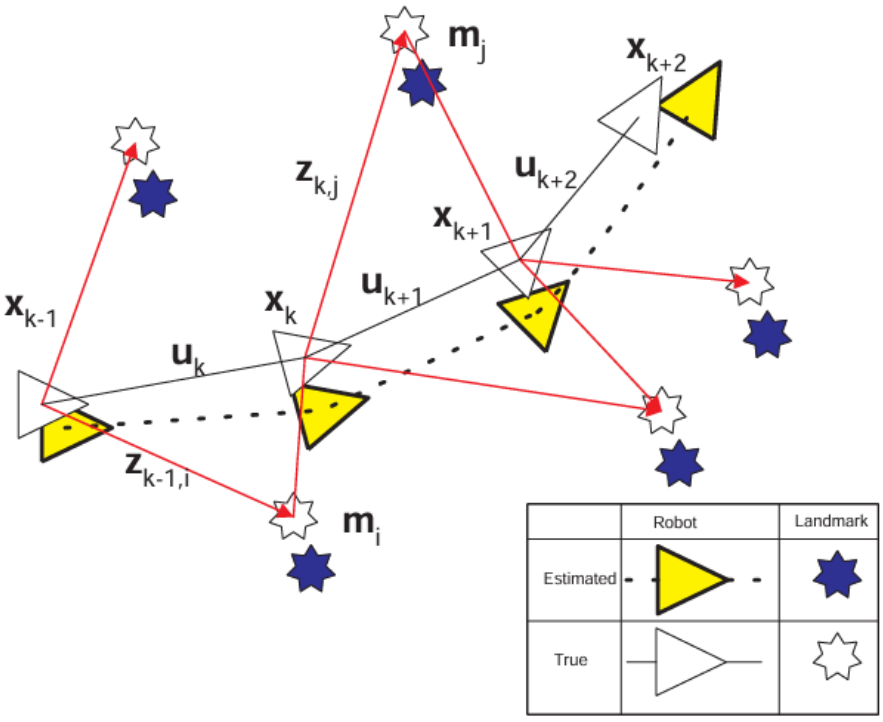


Figura 10: Formulazione del problema SLAM

La risoluzione del problema SLAM richiede l'identificazione della distribuzione di probabilità relativa alla posizione $P(\boldsymbol{x}_k, \boldsymbol{m} \mid \boldsymbol{Z}_{0:k}, \boldsymbol{U}_{0:k}, \boldsymbol{x}_0)$, aggiornata iterativamente per ogni istante k.

Il metodo impiegato nel progetto di tesi sfrutta un filtro di Kalman esteso per stimare la distribuzione di probabilità ed è detto EKF-SLAM.

Il moto del veicolo è descritto in funzione dello stato all'istante di tempo precedente:

$$P(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \Leftrightarrow \boldsymbol{x}_k = \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) + \boldsymbol{w}_k$$

Dove $\boldsymbol{w}_k$ è un segnale del tipo disturbo Gaussiano, a media nulla, di covarianza $\boldsymbol{Q}_k$.

Data quindi una sequenza di osservazioni si può ricavare la posizione del veicolo e dei punti di riferimento, stimando contemporaneamente l'errore complessivo di posizionamento. Dato un set di landmarks, la risoluzione tramite EKF-SLAM è convergente per misure ripetute.

Gli algoritmi di SLAM visivo sfruttano informazioni provenienti da una vasta gamma di sensori come fotocamere semplici o monoculari, molteplici sensori ottici ravvicinati in configurazione stereografica, oppure ad occhio composto (più fotocamere distribuite nello spazio 3D), insieme a telecamere infrarosse di profondità, sistemi lidar etc.

Il sensore impiegato nel progetto di tesi è del tipo stereografico e di profondità, per cui si parla di visual SLAM (vSLAM) RGB-D.

Si esamina ora in dettaglio la struttura dell'algoritmo di un sistema SLAM.

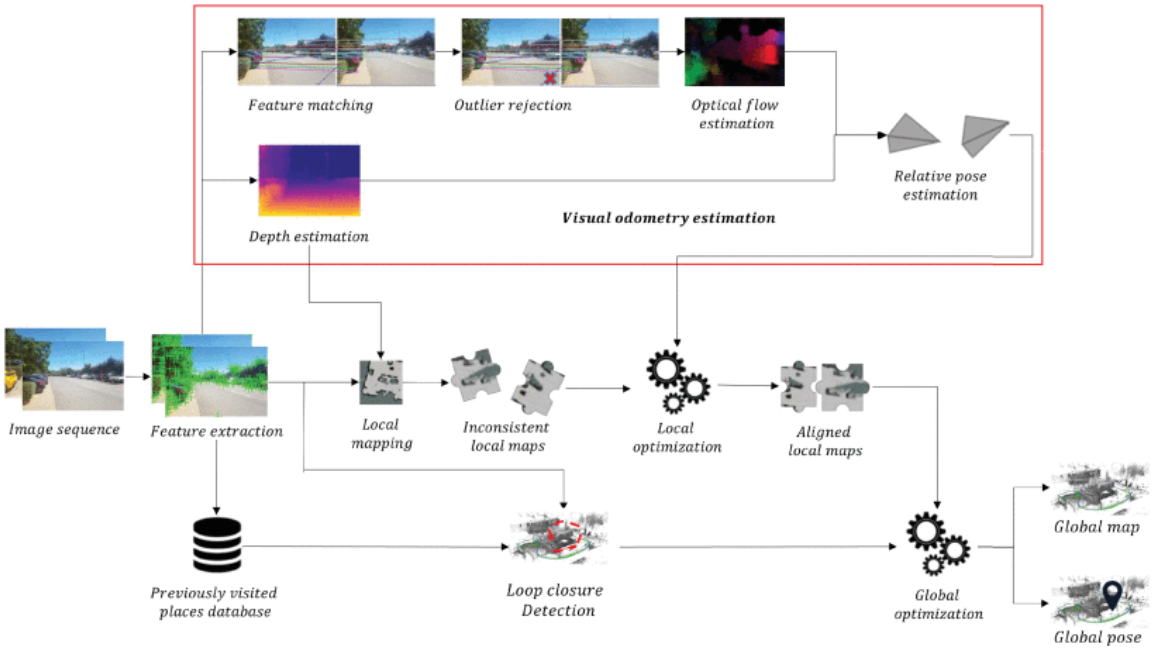


Figura 11: Struttura di un algoritmo SLAM [21]

i. Feature Extraction and Visual-Inertial Odometry

Per poter valutare la posizione del veicolo nello spazio, un sistema vSLAM analizza i fotogrammi ottenuti per ottenere un'indicazione del moto della telecamera relativo all'ambiente, che prende il nome di flusso ottico.

Per ogni frame acquisito, l'immagine viene pre-processata per ridurre il numore, rimuovere distorsioni e correggere il colore. Le immagini vengono quindi elaborate da un filtro di Feature Detection and Description: algoritmi di computer vision (come FAST, ORB o SIFT) determinano e catalogano i candidati punti di interesse dell'immagine, come spigoli o punti più o meno luminosi; che vengono filtrati per rimuovere outlier e punti di scarso interesse. Le coordinate dei punti di interesse sono inoltrate ad un modulo di Feature Matching per essere confrontate con il database interno dei landmark raccolti.

Fig 3.1: Original image

Fig 3.2: FAST keypoint

Fig 3.3: Non-maximum suppression

Figura 12: Riconoscimento dei landmark [30]

Finalmente, confrontando l'evoluzione temporale di tutte le traiettorie delle features esaminate, il modulo di stima del moto della telecamera (Camera Motion Estimator) può ricostruire la traiettoria seguita e l'orientazione spaziale attuale.

Per ottenere un posizionamento preciso e compensare gli errori dovuti alla covarianza non nulla caratteristica dei metodi probabilistici, l'odometria visiva appena ottenuta è fusa con l'odometria inerziale elaborata dai sensori IMU. Si parla quindi di odometria visivo-inerziale (VIO, Visual-Inertial Odometry). Possono essere fusi anche dati provenienti da sensori di altro genere.

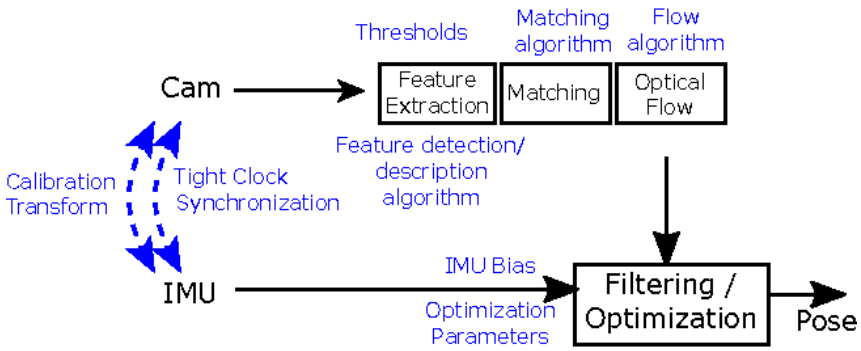


Figura 13: Generazione dell'odometria visivo-inerziale

ii. Local and Global Mapping and Optimization

Dai fotogrammi provenienti dalla telecamera vengono estratti regolarmente i cosiddetti fotogrammi chiave (keyframes), per supportare la creazione di una mappa locale. Ottenuta infatti la stima della posa del veicolo dal modulo visivo-inerziale, i landmark vengono distribuiti nello spazio per formare una mappa delle circostanze.

Sovrapponendo le informazioni di ogni singola mappa locale per ogni punto della traiettoria si ottiene finalmente la mappa globale dei punti di riferimento.

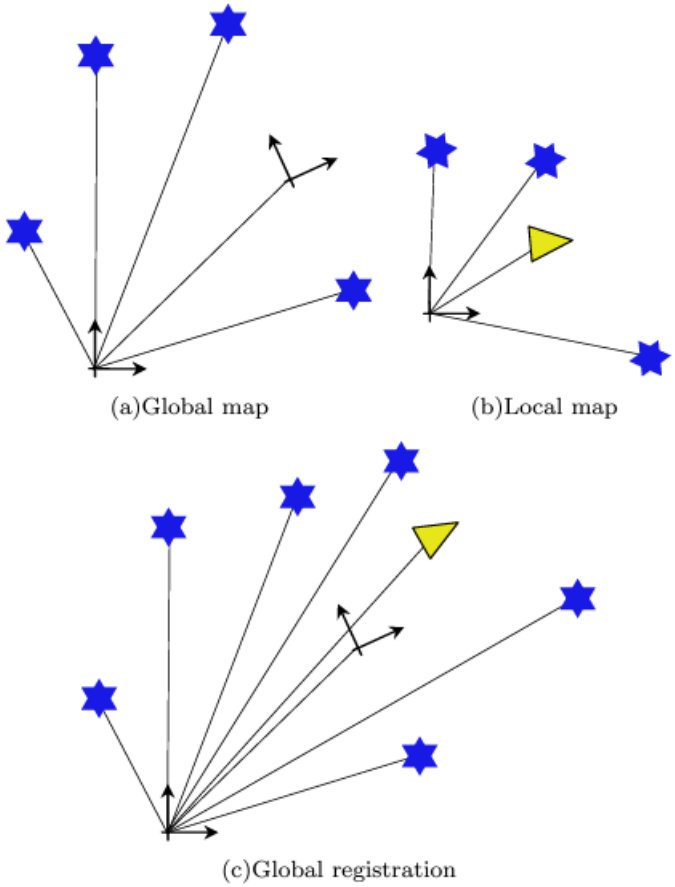


Figura 14: Aggiunta delle features alla mappa globale [11]

Il metodo di selezione dei fotogrammi chiave varia da algoritmo ad algoritmo ed è al momento uno degli spunti di sviluppo più prolifico. Le ricerche in questo ambito si concentrano al momento su metodi di catalogazione semantica dei keyframe, impiegando sistemi a reti neurali per la classificazione intuitiva delle features.

La mappa globale è quindi ottimizzata rimuovendo i punti di interesse ripetuti o la cui probabilità posizionale sia inferiore ad una determinata soglia.

iii. Loop Closure and Global Optimization

Terminata la generazione della mappa globale viene svolta l'operazione di loop closure.

A causa dell'accumulo dell'errore per colpa del drift degli accelerometri, o per la presenza di errori nella determinazione dei landmark si è osservato che l'odometria tenda a subire una perdita di precisione. In particolare, si ha un incremento nella norma della matrice di covarianza del veicolo. Questa perdita, detta diluizione geometrica di precisione (GDOP), risente negativamente sulla capacità dell'algoritmo di mappatura di generare una mappa fedele alla realtà circostante.

Per missioni di lunga durata può capitare quindi che la mappa tenda a disallinearsi, fino alla condizione di non riuscire a richiudersi su sé stessa nonostante sia seguita una traiettoria chiusa. Gli algoritmi di loop closure analizzano quindi i fotogrammi chiave in modo da consentire la distinzione tra punti differenti e stessi punti osservati da angolazioni diverse. Per punti del secondo caso, si fa riferimento al database dei landmark e la mappa viene distorta in modo da far coincidere la nuova misurazione con quella memorizzata.

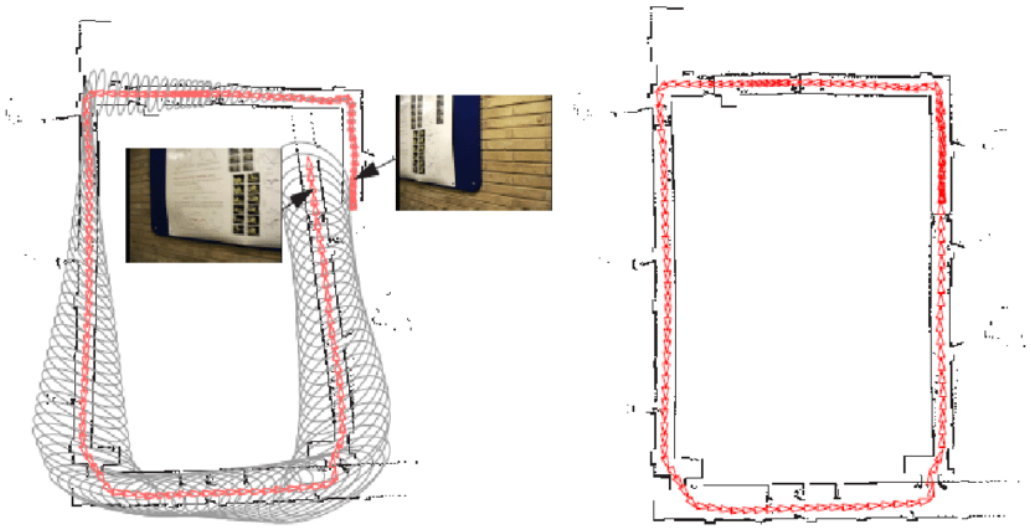


Figura 15: Loop closure

Si noti in Figura 15 la progressiva espansione degli ellissoidi di incertezza, rappresentati in grigio, indici dell'evoluzione della covarianza delle misure nel tempo.

La chiusura del loop ha il notevole vantaggio di ridurre l'incertezza del sistema, di conseguenza la navigazione vSLAM rappresenta un sistema stabile: in seguito ad un'interruzione del flusso dati visivo, osservando l'ambiente circostante il sistema è in grado di recuperare la posa corretta.

È da notare che il metodo di loop closure varia a seconda dell'algoritmo utilizzato: in sistemi Graph-SLAM la mappa è rappresentata come un grafo ordinato, permettendo il riconoscimento anche dei vincoli geometrici nell'immagine; nei sistemi EKF-SLAM invece questo avviene durante la fase di aggiornamento del filtro di Kalman.

A causa di questa differenza, i sistemi EKF-SLAM sono proni ad errori causati da falsi positivi in quanto perdono precisione in maniera estremamente rapida se i punti di interesse sono simili tra loro.

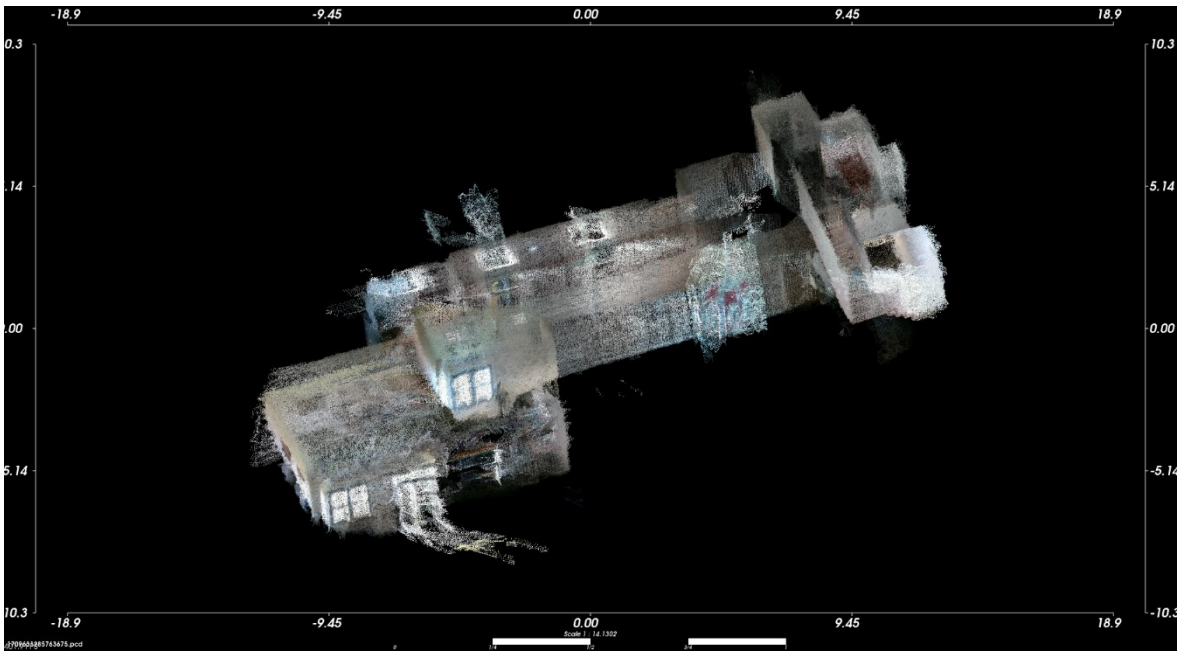


Figura 16: Scansione volumetrica interna del laboratorio CIRI Aerospaziale

Si prenda ad esempio la scansione volumetrica in Figura 16, effettuata tramite il modulo EKF-SLAM della telecamera impiegata nel progetto di tesi. La sezione del corridoio scansionata è sufficientemente fedele alla realtà e, nonostante alcune variazioni di luminosità, la presenza di elementi caratteristici e dal colore brillante, come un estintore portatile e una porta a vetri esterna, è stata sufficiente per compensare gli errori di misura. Non è stato invece possibile digitalizzare l'ala orientale del laboratorio, alla sinistra dell'immagine, in quanto il corridoio in quella direzione si presenta di colore omogeneo e praticamente privo di punti di interesse. Tentando la scansione si è sempre ottenuto un risultato insoddisfacente o completamente inutilizzabile.

3. Architettura della piattaforma

3.1. Hardware utilizzato

Il mezzo utilizzato per la sperimentazione consiste di un rover terrestre equipaggiato con una telecamera di profondità e un computer ausiliario di bordo.

Il rover in questione è un robot terrestre progettato dall'azienda polacca *fictionlab*, denominato LeoRover, sviluppato per ricercatori e università e largamente impiegato per collaudare sistemi di navigazione autonoma.

Figura 17: LeoRover

Il rover è provvisto di quattro ruote motorizzate, ognuna dotata di motoriduttore epicicloidale ed encoder rotativo per la determinazione della velocità di rotazione, montate su un supporto mobile per compensare le asperità del terreno. Il corpo principale è composto dall'alloggiamento della batteria e dal contenitore frontale, che ospita una telecamera a bassa risoluzione e l'unità di controllo. Entrambi i contenitori sono protetti contro getti d'acqua e completamente ermetici a fumo e polveri (IP66).

Un Raspberry PI 4B da 2 Gb costituisce l'unità di controllo centrale, associato ad una scheda di gestione (LeoCore) basata sul controllore STM32F4, che si occupa del controllo di batteria, sensori inerziali, encoder e motori.

Per quanto riguarda la connessione con l'esterno, il rover è dotato di una scheda wireless USB che funge da Access Point, per la comunicazione e il controllo tramite interfacce esterne.

Era inoltre presente in origine una porta USB micro-A dotata di tappo ermetico, che è stata sostituita da un cavo ethernet collegato alla relativa porta sulla scheda Raspberry PI. In questo modo è possibile collegare l'unità di controllo esterna tramite un canale ad elevate prestazioni, riducendo la latenza e il carico sulla rete wireless.

Per quanto riguarda l'acquisizione delle informazioni visive si è optato per l'uso della telecamera di profondità Intel RealSense D455.

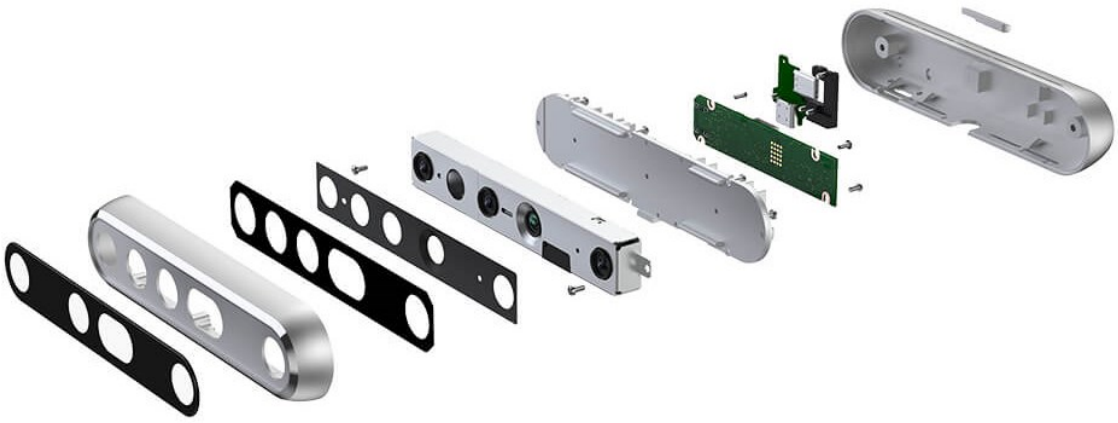


Figura 18: Intel RealSense D455 Depth Camera

Equipaggiata con due sensori RGB da 1 MP in configurazione stereoscopica e un sensore di profondità a laser infrarosso, è largamente impiegata in robotica per creare mappe tridimensionali dell'ambiente. È anche dotata di un'unità inerziale, utilizzata per ricavare la posizione nello spazio.

Il meccanismo di determinazione della profondità è basato su un duplice effetto. In primo luogo, tramite una tecnica di laser ranging: dopo aver illuminato il bersaglio con un impulso luminoso infrarosso, si misura il tempo necessario a rilevare la riflessione TOF (Time of flight). Nota la velocità della luce e il tempo di volo è triviale ricavare lo spazio percorso. Per ottenere la distanza per ogni punto dell'immagine il proiettore infrarosso genera sequenzialmente una griglia di punti distribuiti uniformemente sul piano dell'immagine e per ciascuno di essi ripete la misurazione del tempo.

In contemporanea al laser ranger, si sfrutta la configurazione stereoscopica: due sensori ottici identici sono posti ad una certa distanza tra di loro per osservare un oggetto posizionato entro il loro campo visivo. Stimando la differenza tra i fotogrammi acquisiti è possibile risalire agli angoli di deviazione e di conseguenza alla distanza dell'oggetto.

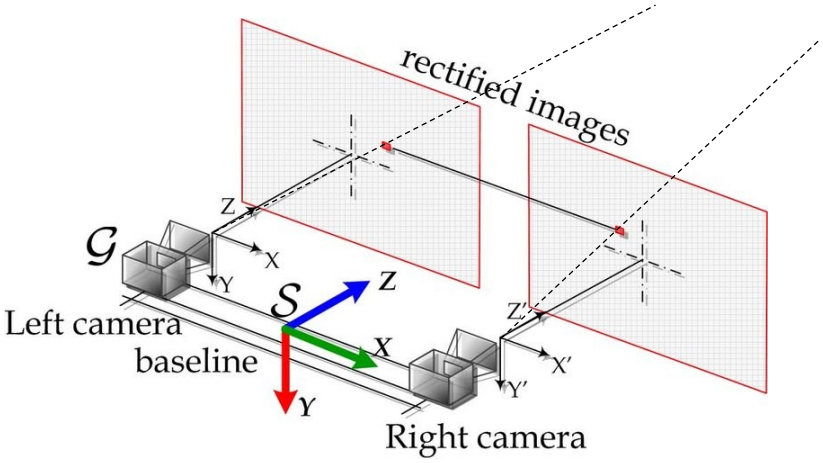


Figura 19: Determinazione della distanza tramite metodo stereografico [23]

I dati in uscita dalla telecamera sono trasmessi sotto forma di frame. Ciascun frame contiene le informazioni dei canali R, G e B dell'immagine, con un quarto canale D per la profondità (rappresentata come un'immagine in scala di grigi, dove un valore 0 indica un oggetto lontano). Oltre alle informazioni ottiche sono trasmesse anche le accelerazioni e velocità angolari misurate dall'unità inerziale e un pacchetto di metadati contenente informazioni come altezza e larghezza del frame, temperatura del processore etc... Considerando che in condizioni di massima operatività la telecamera è in grado di emettere oltre 60 fotogrammi al secondo si ha un'immensa mole di dati che deve essere analizzata dal computer di controllo.

Per evitare di saturare completamente le capacità di calcolo del computer Raspberry di bordo si è optato per l'utilizzo di un secondo computer Raspberry Pi 4B, provvisto di 8 Gb di memoria RAM. Sfruttando il collegamento ethernet si possono trasferire dati fino alla velocità massima di cinque Gbps.

Per agevolare le operazioni di test si è utilizzato temporaneamente un pc portatile DELL, con installato il sistema operativo Ubuntu 20.04 LTS.

Il dettaglio delle connessioni e delle interazioni tra le varie componenti è descritto nel diagramma di Figura 20.

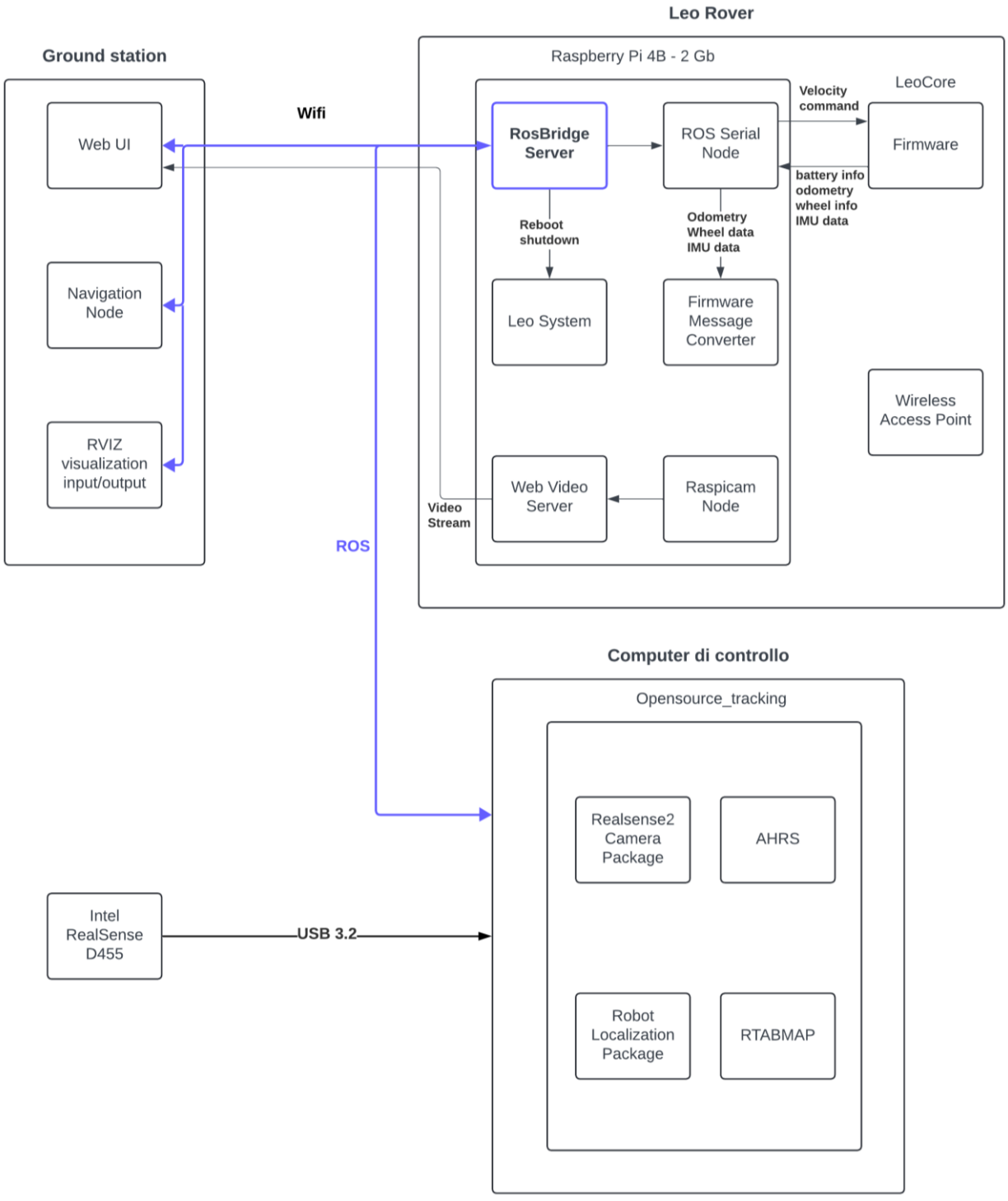


Figura 20: Diagramma del flusso dati del rover

3.2. Il sistema ROS

Convenzionalmente un sistema robotico è composto da un'unità centrale, che svolge le funzioni di calcolo primarie, e diverse periferiche che interagiscono con il mondo esterno. In configurazioni master-slave come questa il carico di lavoro è centrato sul processore principale che, oltre ad eseguire la sua funzione, deve anche coordinare il flusso di dati tra periferiche. All'aumentare della complessità del progetto è necessario aumentare la potenza di calcolo aggiungendo altre unità di controllo.

Il sistema ROS, Robot Operating System, nasce con lo scopo di uniformare i protocolli di comunicazione e favorire lo scambio di dati all'interno di un sistema robotizzato. Sviluppato nel 2007 da ricercatori dell'università di Stanford, è formato da un insieme di librerie e strumenti che permettono la realizzazione di sistemi di calcolo distribuito.

Il protocollo ROS è costituito da vari elementi. Allo scopo di questa tesi, si considerano in dettaglio i seguenti:

- Topic: bus di trasferimento dati utilizzato da vari nodi nella rete ROS per il trasferimento di dati.
- Packages: simile al concetto di "libreria" per altri linguaggi di programmazione, si tratta di un insieme di software precompilato atto a svolgere una determinata funzione.
- Nodi: processi eseguiti dentro l'architettura ROS. Sono software programmabili dall'utente e svolgono funzioni complesse.

I nodi si interfacciano alla rete ROS tramite l'uso di funzioni *publisher*, responsabili di emettere dati su un determinato topic, e funzioni *subscriber*, che ricevono i dati. A seconda delle applicazioni, publisher e subscriber possono essere impostati per leggere o scrivere dati in maniera continuativa (in modalità streaming) o saltuariamente su richiesta.

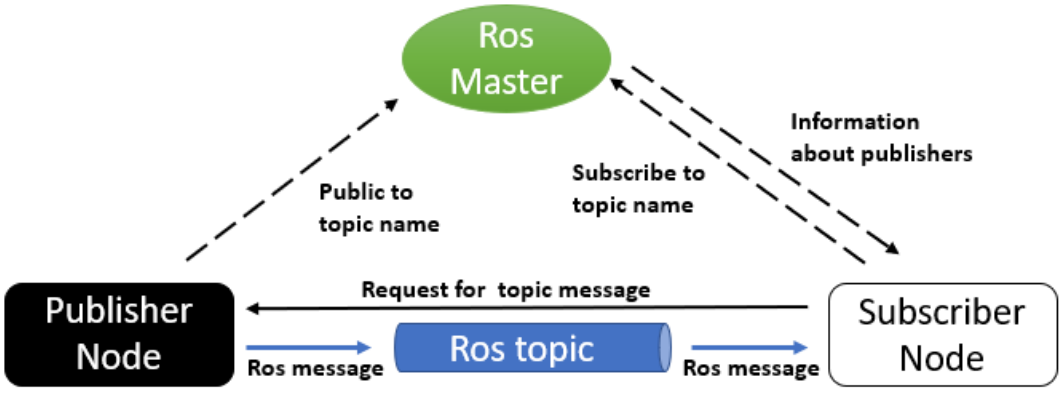


Figura 21: Tipico scambio di informazioni in un sistema ROS

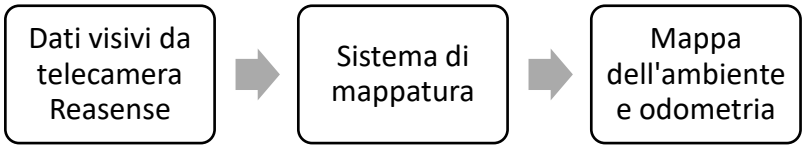
I tre calcolatori impiegati nel progetto comunicano tra loro tramite la rete wireless generata dal rover stesso e il collegamento ethernet. La versatilità del sistema ROS permette di interfacciare i tre computer semplicemente condividendo con il nodo server i relativi indirizzi IP: i messaggi sono condivisi in tempo reale a prescindere dal sistema operativo o dalla tipologia di computer.

Il coordinamento del traffico dati è affidato all'unità interna al rover, che ospita il server e viene denominata Master.

Per la sperimentazione è stato utilizzato ROS 1 Noetic Ninjemys. La scelta è dovuta a stringenti vincoli di compatibilità tra il nodo di input della telecamera (proprietario di Intel) e l'hardware del rover. Per facilitare la visualizzazione dei dati di odometria è stato impiegato l'ambiente RVIZ, un software open source sviluppato per rappresentare in maniera grafica lo stato spaziale di un sistema robotico in un ambiente 3D virtuale.

3.3. Il sistema di mappatura

L'elaborazione dei dati visivi viene effettuata dal computer ausiliario a bordo del rover tramite il sistema di mappatura.



Questo blocco è in realtà formato da quattro nodi, eseguiti parallelamente:

- Realsense2_camera
- Attitude and Heading Reference System (AHRS)
- Robot_localization
- Real-Time Appearance-Based Mapping (RTABMAP)

Il primo nodo, *Realsense2_camera*, gestisce l'ingresso dei dati visivi dalla telecamera di profondità.

Per elaborare l'elevata mole di dati (fino a 1172 Mbps con una risoluzione di 848x480@90fps per entrambi i sensori ottici) [13] l'algoritmo di post-processing è organizzato in maniera sequenziale: i frame attraversano la catena di elaborazione affrontando progressivamente ogni stadio.

Questo tipo di catena di acquisizione prende il nome di Pipeline. Nel sistema sviluppato si è utilizzata la pipeline di default sviluppata da Intel, riportata in Figura 22.

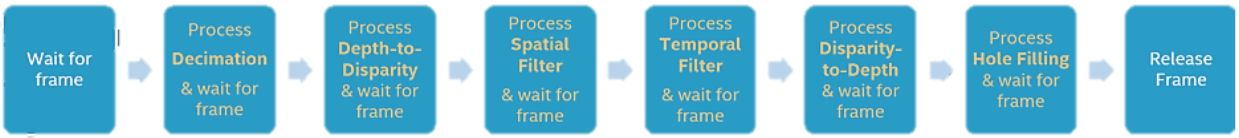


Figura 22: Esempio di catena di acquisizione ottica [28]

Tramite il filtro di decimazione, frame simili vengono "mediati" (si riduce il numero di frame effettivi da elaborare) e tramite il filtro temporale e spaziale si riduce l'errore dovuto a variazioni aleatorie nell'immagine a causa di bruschi cambiamenti di illuminazione o dovuti a vibrazioni meccaniche. Infine, la subroutine Hole-Filling corregge i difetti dell'immagine che possono essere dovuti a riflessioni del laser infrarosso. Questi difetti sono preponderanti in immagini di oggetti lucidi, lontani o quando il laser viene operato in regime di potenza ridotta. [10]

A concludere la sequenza di elaborazione dei dati ottici si trova il nodo *RTABMAP*. [11] Sviluppato inizialmente come libreria open-source, il sistema Real-Time Appearance-Based Mapping sfrutta le variazioni tra frame successivi per stimare il moto nello spazio del veicolo, generando in contemporanea una mappa tridimensionale dell'ambiente.

Allo scopo del progetto di tesi si considera come mappa la sezione bidimensionale del modello 3D realizzata con un piano parallelo al pavimento e posto all'altezza del sensore di profondità, fornita dal topic *2D Occupancy Grid*.

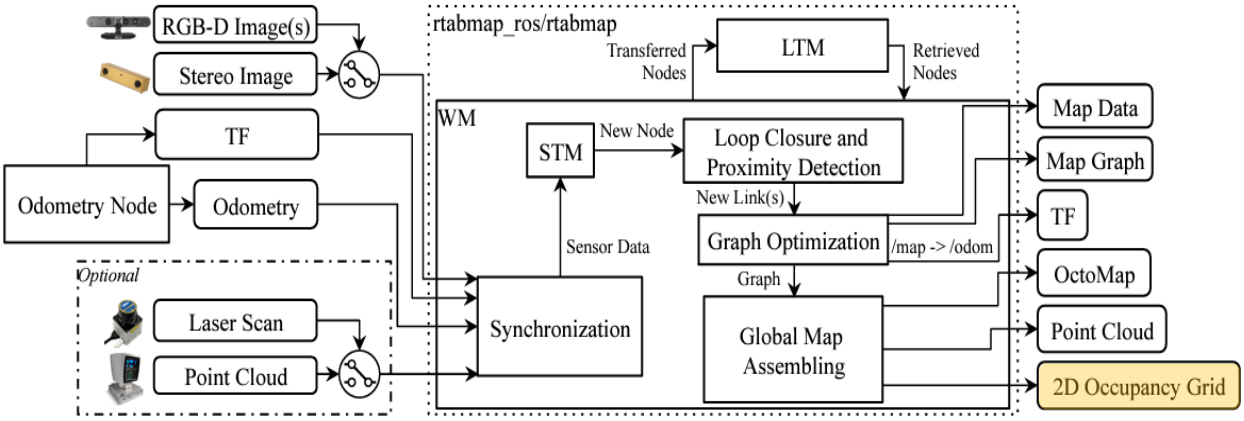


Figura 23: Funzionamento del nodo RTABMAP [11]

L'odometria inerziale è ricavata dal nodo *AHRS*: le accelerazioni e le velocità angolari misurate dall'accelerometro e dal giroscopio della telecamera vengono integrate dal filtro di Madgwick per ottenere lo stato spaziale completo. Il filtro può essere adattato a ricevere in ingresso anche dati forniti da un eventuale sensore di orientazione magnetica, ed è tarabile a seconda delle necessità di calibrazione dei sensori.

Per ottenere infine l'odometria visivo-inerziale si fruttano le tecniche di sensor fusion fornite dal package *robot_localization*, in particolare l'elaborazione tramite filtro di Kalman.

Si hanno finalmente le 15 componenti del vettore di stato:

$$\ddot{X}, \ddot{Y}, \ddot{Z} \quad Accelerazioni\ lineari$$

$$\dot{X}, \dot{Y}, \dot{Z} \quad Velocità\ lineari$$

$$X, Y, Z \quad Posizione$$

$$\dot{\varphi}, \dot{\theta}, \dot{\psi} \quad Velocità\ angolari$$

$$\varphi, \theta, \psi \quad Assetto\ angolare$$

3.4. Algoritmo di navigazione

In questa sezione verranno commentate le funzioni principali del software di navigazione, trascurando i segmenti di codice meno rilevanti. Il listato di codice completo è disponibile nell'Appendice 1.

Lo scopo dell'algoritmo è, in primo luogo, la determinazione del percorso ottimale congiungente la posizione attuale del rover con un punto sulla mappa richiesto dall'utente e, in secondo luogo, il calcolo dei comandi di velocità da inviare alla piattaforma mobile per compiere lo spostamento richiesto.

A grandi linee il funzionamento dell'algoritmo è dato dalla seguente sequenza di istruzioni:

1. Attesa della pubblicazione della mappa da parte del nodo di mappatura e relativa lettura
2. Applicazione di una sfocatura Gaussiana per generare una zona di offset intorno agli ostacoli
3. Attesa della pubblicazione delle coordinate del punto bersaglio
4. Calcolo del percorso ottimale
5. Interpolazione SPLINE e generazione dei waypoint intermedi
6. Controllo closed-loop dell'angolo di direzione del veicolo e della velocità lineare

Il nodo di navigazione è stato sviluppato in linguaggio Python 3.12. Come altri linguaggi largamente utilizzati nella progettazione di sistemi autonomi si tratta di un linguaggio di programmazione orientata agli oggetti (OOP).

In un linguaggio OOP alcune sezioni del software vengono raggruppate in strutture chiamate classi, ognuna dotata di un proprio set di dati da elaborare (attributi) e funzioni (metodi)[12].

L'algoritmo è composto da una classe principale, *navigation*, e da due classi secondarie *getTrajectory* e *PID*. Il principale vantaggio di un approccio object-oriented è la modularità: al posto di dichiarare ogni funzione nel corpo principale del software ogni classe può essere dotata di un proprio set di parametri e funzioni non più globali ma locali, inaccessibili al resto del codice, e può di conseguenza essere eseguita in più istanze contemporaneamente.

Si considerino a scopo di esempio le istruzioni seguenti:

```
211.      self.pid_ang = PID(0.75,0.1,0)
212.      self.pid_lin = PID(1,0.01,0)
```

Si desidera, durante l'inizializzazione della classe *navigation()*, che vengano generate due istanze della classe *PID* ma con parametri di tuning differenti. A seguito della dichiarazione si hanno funzionalmente delle copie della classe iniziale che ereditano le funzioni e le strutture dati originali, essendo però oggetti totalmente distinti tra loro.

Si noti inoltre come gli oggetti appena creati abbiano il prefisso *self.* posto prima del nome: in questo caso le istanze dei controllori sono inserite all'interno della struttura dati della classe *navigation()* e sono quindi richiamabili esternamente alla classe, al contrario di altre variabili locali che esistono solamente all'interno della classe.

i. Inizializzazione

In primo luogo, sono dichiarate le librerie in uso nel software.

```
 1. import rospy
 2. import tf
 3. import numpy as np
 4. import heapq
 5. from math import pi, atan2
 6. from scipy import interpolate
 7. from scipy.ndimage import gaussian_filter
 8. from geometry_msgs.msg import Twist, PoseStamped
 9. from nav_msgs.msg import Odometry, OccupancyGrid, Path

                  # Segue il corpo principale del software

320. if __name__ == '__main__':
321.     try:
322.         navigation()
323.     except rospy.ROSInterruptException:
324.         rospy.loginfo("Processo interrotto dall'utente.")
```

L'implementazione python del protocollo ros avviene tramite la libreria rospy, in congiunzione con *geometry_msgs* e *nav_msgs*. Per la manipolazione dei quaternioni si utilizza la libreria *tf*; per ultime *heapq, math, numpy* e *scipy* forniscono le funzioni necessarie all'elaborazione numerica dei dati.

Le ultime linee di codice sono in realtà le prime ad essere eseguite: all'avvio del nodo il sistema inizializza immediatamente la classe *navigation()*, responsabile del calcolo del percorso e della determinazione dei comandi di velocità.

```
131. class navigation():
132.
133.     def odom_callback(self, msg): ...
145.     def map_callback(self, msg): ...
166.     def wp_callback(self, msg): ...
173.     def publish_waypoints(self): ...
193.     def __init__(self):
194.         rospy.init_node('Navigation_node', anonymous=False, log_level=rospy.INFO)

239.         self.odom_sub = rospy.Subscriber("rtabmap/odom", Odometry,
             self.odom_callback)
240.         self.wp_sub = rospy.Subscriber("move_base_simple/goal", PoseStamped,
             self.wp_callback)
241.         self.map_sub = rospy.Subscriber("rtabmap/grid_map", OccupancyGrid,
             self.map_callback)
242.
244.         self.vel_pub = rospy.Publisher('cmd_vel', Twist, queue_size=10)
245.         self.waypoint_pub = rospy.Publisher('path', Path, queue_size=10)
```

Per abilitare la comunicazione con i topic del rover e della telecamera i primi elementi ad essere avviati sono i publisher ed i subscriber, seguono poi le variabili d'ambiente principali, modificabili dall'utente per alterare il comportamento del sistema.

ii. Ricezione della mappa e applicazione della sfocatura Gaussiana

La mappa viene ricevuta dalla relativa funzione callback tramite un messaggio del tipo *OccupancyGrid*. Il messaggio è ricevuto sotto forma di vettore 1D, note le dimensioni della mappa in termini di numero di caselle per lato si ricostruisce il dato come matrice in modo da ottenere una griglia bidimensionale. Ogni elemento della matrice rappresenta una porzione di spazio quadrata di 5 cm di lato. Il relativo stato è indicato tramite un numero intero da 0 a 100, indicante, in percentuale, la densità di probabilità che sia presente un ostacolo. Nell'applicazione attuale la casella assume solamente i valori estremi (normalizzati tra 0 e 1), indicando con -1 una zona della mappa non ancora scansionata.

Prima di poter considerare completa la fase di acquisizione è necessario compiere un ulteriore passaggio. L'algoritmo di pianificazione del percorso è incaricato di cercare, se esiste, il percorso più breve che congiunge due caselle; tuttavia, questo compito viene svolto considerando il rover come se fosse un punto materiale, di dimensione nulla. Vengono a generarsi quindi alcuni difetti nella generazione della traiettoria, in particolare:

- Per sua natura il percorso più breve passa radente agli ostacoli nel caso debba aggirarli: senza tolleranza posizionale il rover rischia di impattare contro di essi.
- Nell'esplorare la mappa l'algoritmo considera adiacenti alla casella attuale tutte e otto le confinanti: nella sfortunata evenienza che le quattro caselle ortogonali siano marcate come ostacoli ma non lo sia una sulla diagonale allora il sistema la considererà come candidata accettabile. Il rover dovrebbe attraversare quindi lo spazio tra due spigoli, un passaggio di dimensione nulla.

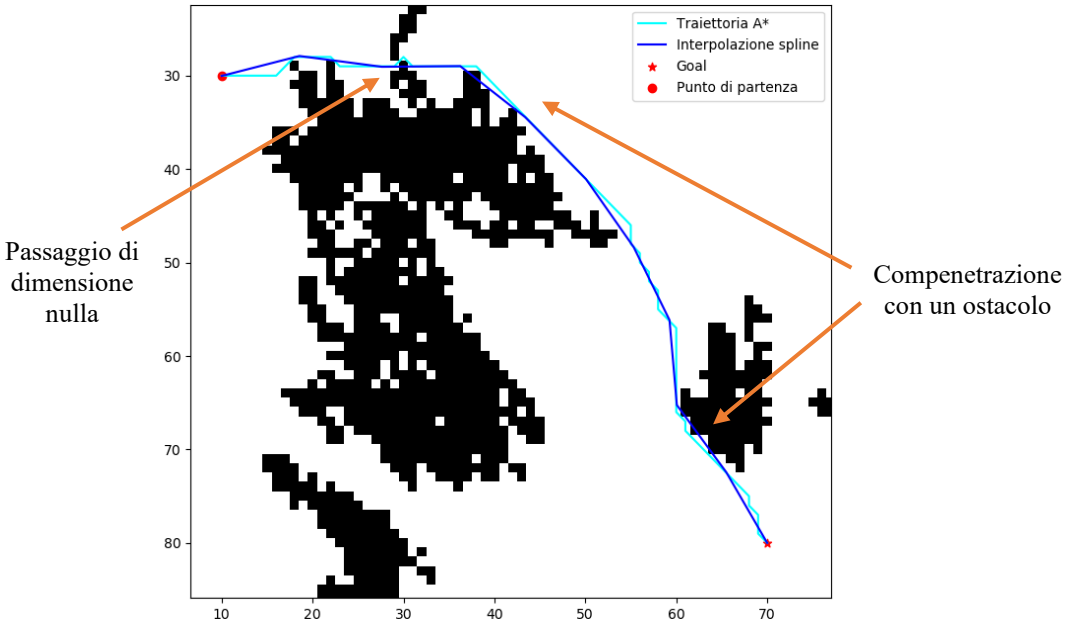


Figura 24: Difetti nella generazione del percorso

Per generare un margine, dal momento che ogni valore diverso da zero rappresenta un ostacolo (rappresentato in nero nell'immagine), si considera la mappa come immagine in bianco e nero, sulla quale applicare un filtro di sfocatura Gaussiana [10]. L'applicazione di un filtro su un'immagine è realizzata tramite l'operazione di convoluzione: si genera una matrice detta *kernel* che viene traslata lungo i due assi dell'immagine, considerando di volta in volta la media pesata dei valori della sottomatrice dell'immagine originale ottenuta dalla sovraimpressione con il kernel.

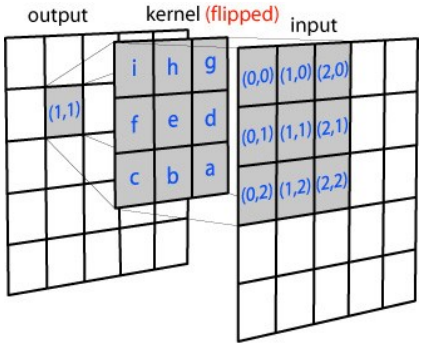


Figura 25: Convoluzione 2D [22]

Il risultato dell'operazione è un valore scalare, posto presso le coordinate dell'elemento centrale del kernel sull'immagine. Regolando i valori degli elementi del kernel è possibile ottenere vari effetti, sfruttando una distribuzione dei pesi dettata da una distribuzione normale si può ottenere l'effetto di sfocatura desiderato.

Dal momento che il sistema considera qualsiasi valore maggiore di zero come ostacolo allora i valori intermedi generati dalla mediazione verranno considerati ostacoli a loro volta. Si è ottenuto quindi un offset la cui estensione è direttamente controllata dal valore della variabile *sigma*, la deviazione standard.

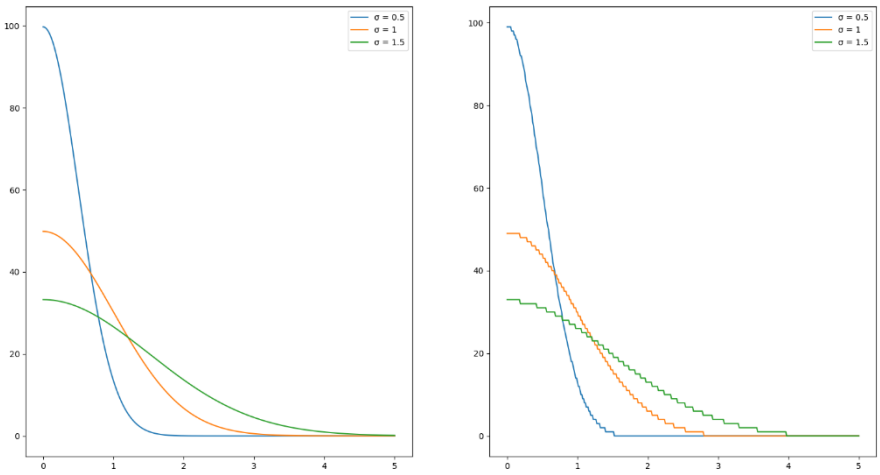


Figura 26: Andamento e discretizzazione della parte positiva della distribuzione normale

La mappa ottenuta in questo modo può finalmente essere inviata in sicurezza alla funzione di pianificazione del percorso.

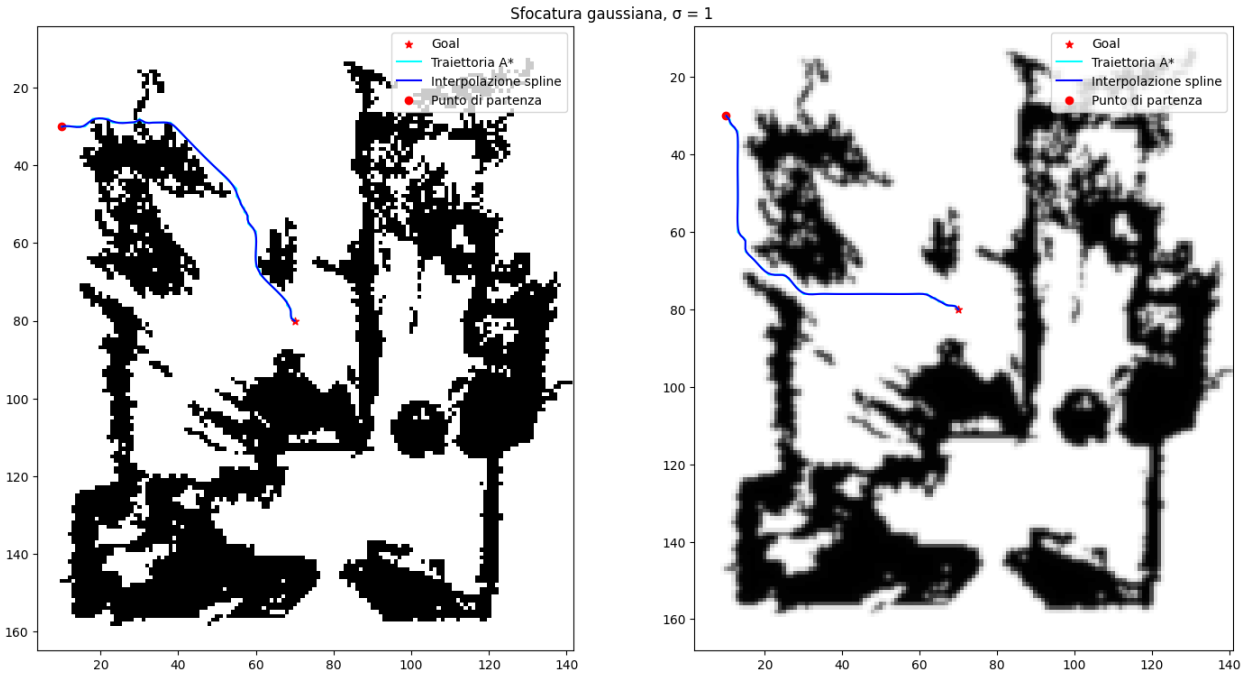


Figura 28: Generazione del percorso prima e dopo l'applicazione del filtro

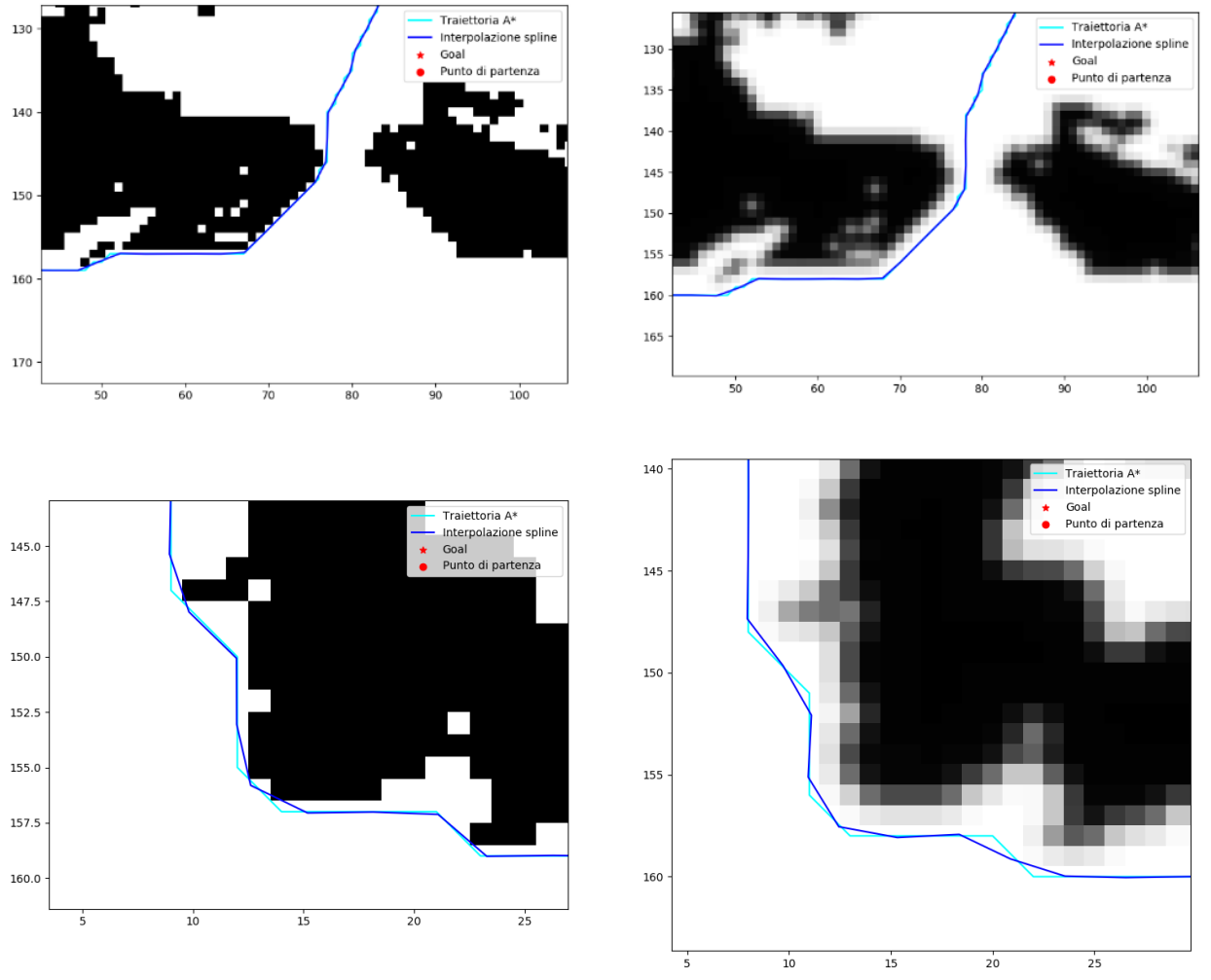


Figura 27: Dettaglio della generazione dell'offset per altre configurazioni

Terminata la fase di elaborazione della mappa, il sistema resta in stato di attesa dell'input da parte dell'utente del punto di destinazione. Questo viene fornito sotto forma di messaggio sul topic */move_base_simple/goal* ed è pubblicabile tramite l'interfaccia RVIZ interagendo con il comando "2D Nav Goal" e selezionando un punto sul piano della mappa.

Dal momento che la mappa viene elaborata sotto forma di matrice ogni elemento deve avere indici interi e positivi, questo è possibile però soltanto se l'origine del coincide con l'origine della mappa. Si effettuano quindi dei cambi di sistema di riferimento:

- All'avvio il rover è posto alle coordinate (0,0) nella zona centrale della mappa, mentre quest'ultima ha origine alle coordinate $\bar{O}_{map} = (-x_{map}, -y_{map})$. Si pone la nuova origine in queste ultime coordinate. La posizione dell'origine del rover sarà quindi in $\bar{O}_{rover} = (x_{map}, y_{map})$
- I waypoint generati sono forniti relativamente alla posizione del rover all'istante di generazione, per portarli nelle coordinate mappa vi si somma vettorialmente la posizione di origine: $\bar{P}_{map} = \bar{P}_{rover} + \bar{O}_{rover}$
- I sistemi di riferimento della mappa e del rover sono entrambi destrorsi, con la differenza però che il primo ha l'asse z diretto verso l'alto al contrario del secondo, che rispetta la convenzione NED per gli assi locali. Per ovviare questa differenza la mappa viene ruotata in senso orario di 90°.

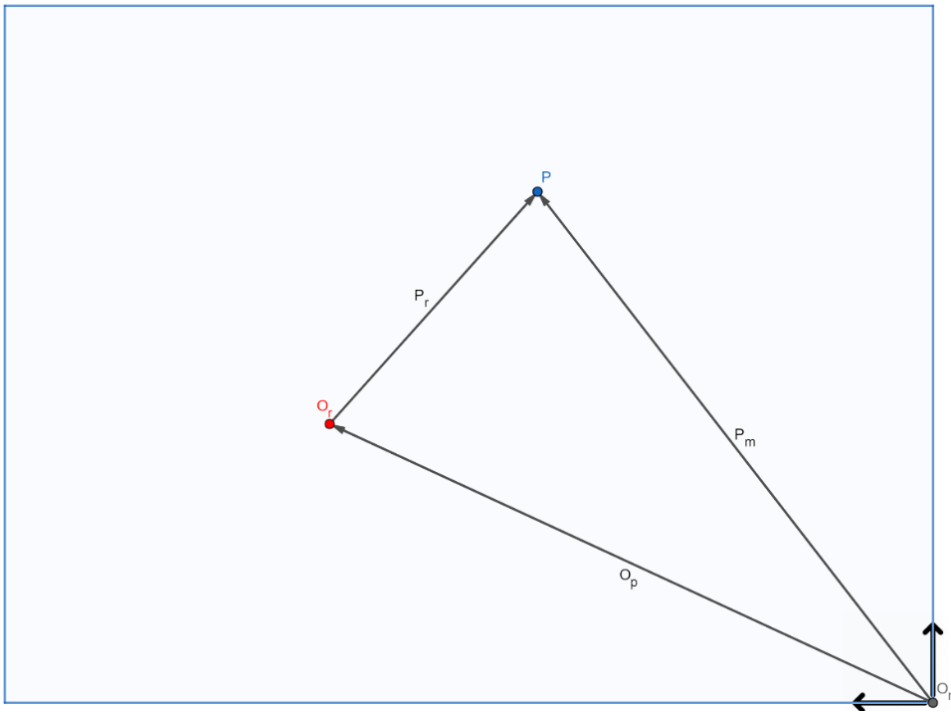


Figura 29: Sistema di riferimento al termine delle trasformazioni

iii. Pianificazione del percorso

Completata l'acquisizione della mappa e dei waypoint, è possibile procedere alla determinazione del percorso ottimale.

```
262.      self.pathfinder = getTrajectory(self.curveRes, self.curveSmoothness,
               self.mapSize, self.mapResolution, self.neighMode)
263.      self.pathfinder.calcPath(self.map, self.wp, self.pos_attuale)
```

Al momento dell'avvio della funzione *calcPath* le coordinate della posizione attuale e dei waypoint, già riferite al sistema di riferimento della mappa, vengono convertite da metri in unità di mappa semplicemente dividendole per la risoluzione (5 cm).

$$(x, y)[m] \rightarrow (x/res\,, y/res)$$

```
96.       def calcPath(self, mappa, wp, pos_attuale):
97.            pos = (round(pos_attuale[0]/self.mapResolution), self.mapSize[1] -
               round(pos_attuale[1]/self.mapResolution))
               # Conversione da metri in unità di mappa
98.            wp = (round(wp[0]/self.mapResolution), self.mapSize[1] -
               round(wp[1]/self.mapResolution))
99.            mappa = np.rot90(mappa, k=-1)            # Rotazione oraria di 90 gradi
106.           route = self.astar(mappa, wp, pos)
```

Terminata la conversione delle coordinate e la rotazione della mappa, l'algoritmo di pianificazione viene attivato.

Il problema di identificare il percorso più breve che collega due punti è del tipo *best-first search*. In questa categoria di problemi si esplora un grafo, struttura dati caratterizzata da un numero finito di vertici (nodi) e dagli spigoli che li collegano, espandendo la ricerca verso i nodi che successivamente minimizzano una data funzione.

La funzione minimizzata, detta funzione di costo o euristica, può essere decisa arbitrariamente a seconda dal problema da risolvere; nel caso in questione il costo è rappresentato dalla distanza euclidea tra la casella attuale e la casella bersaglio. Questo tipo di ricerca è chiamata ricerca a costo uniforme. La scelta del metodo di ricerca ottimale è ricaduta sull'algoritmo A^*. [13]

Il codice fu inizialmente implementato nel 1968 da alcuni ricercatori dell'istituto di ricerca di Stanford nell'ambito del progetto "Shakey", il primo robot mobile ad utilizzare un sistema decisionale autonomo per svolgere compiti complessi [14].

Partendo da un nodo selezionato, l'algoritmo valuta la distanza tra la posizione attuale e la meta sfruttando un'euristica definita a priori. Il nodo viene quindi aggiunto ad una lista

denominata “open”. La struttura dati della lista è del tipo heap: i nodi rimangono ordinati rispetto al valore assegnatogli al momento dell’inserimento.

Il nodo è quindi rimosso dalla cima della lista. A questo punto si possono realizzare tre possibili scenari:

- La lista era vuota e non è possibile rimuovere il nodo: non esiste un percorso che unisca le due coordinate scelte.
- L’ultimo nodo rimosso coincide con la meta: è stato trovato un percorso. Ricostruendo la sequenza dei nodi rimossi in precedenza si ottiene il percorso.
- L’ultimo nodo rimosso non coincide con la meta: si generano nuovi nodi per ogni casella segnata come “vicino” e l’algoritmo è reiterato partendo da questi ultimi.

I nodi il cui costo è già stato calcolato sono memorizzati in una seconda struttura “closed” per evitare che vengano ricalcolati nelle iterazioni successive. Il tempo di calcolo risparmiato in questo modo aumenta esponenzialmente all’aumentare della dimensione della matrice in ingresso.

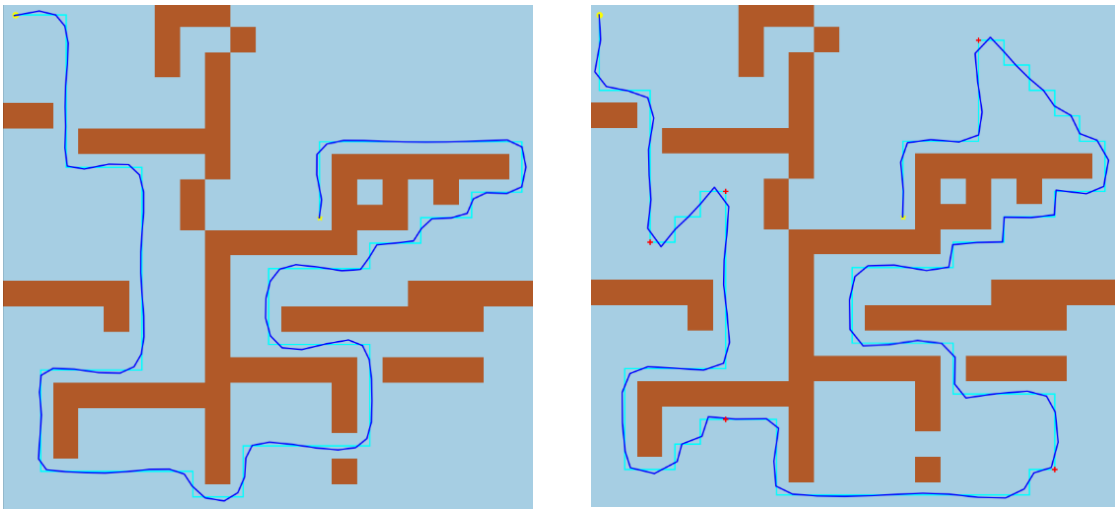


Figura 30: A in mappa di test, bersaglio singolo e bersagli multipli sequenziali*

A* è un algoritmo completo, ammissibile ed ottimale: per ogni grafo con numero di vertici finito e con spigoli di valore non-negativo l’algoritmo ha un tempo di elaborazione finito; se la metrica impiegata non sovrastima i costi allora il percorso ottenuto sarà sempre il percorso più breve. In sviluppi successivi può essere utile sostituire l’algoritmo di ricerca con variazioni che sfruttino il principio della bounded relaxation: si permette il raggiungimento di una soluzione sub-ottimale allo scopo di ottenere un percorso sufficientemente valido in un lasso di tempo minore.

Dal momento che il percorso ottenuto in output è formato da un'unione di un numero finito di segmenti si viene a generare una traiettoria continua ma pesantemente irregolare. Per evitare che il controllore di posizione generi variazioni troppo brusche la traiettoria viene approssimata tramite una funzione SPLINE interpolante.

```
113.     for i in (range(0,len(route))):
114.        x = route[i][0]
115.        y = route[i][1]
116.        x_coords.append(x)
117.        y_coords.append(self.mapSize[1] - y)
118.
119.        if self.smoothing > 0:
120.           tck, u = interpolate.splprep([x_coords,y_coords], s=self.smoothing)
               # Interpolazione spline per ottenere una traiettoria continua e liscia
121.           unew = np.arange(0, 1.01, self.resolution)
122.           self.out = interpolate.splev(unew, tck)
123.         else:
124.           self.out = [x_coords, y_coords]
```

Una curva SPLINE è una funzione continua formata dall'unione a tratti di più polinomi di grado prefissato, approssimante una curva passante per un numero intero di nodi [15].

Siano dati *k* nodi *t* distribuiti nello spazio. La funzione spline *B* sarà definita come:

$$B: [a, b] \in \mathbb{R} \rightarrow \mathbb{R}$$

Dove $[a, b] = \{[t_0, t_1[\cup [t_1, t_2[\dots \cup [t_{i-2}, t_{i-1}[\cup [t_{i-1}, t_i[\}$ rappresenta l'intervallo di interpolazione, definito a tratti.

Per ogni i-esimo sotto intervallo di $[a, b]$ si definisce un polinomio P_i di grado $n \in \mathbb{N}$ tale che:

$$P_i = \sum_{j=1}^{n} c_j x^j \quad limitata\ su\ [t_i, t_{i+1}[\,, \qquad c \in \mathbb{R}$$

La funzione si dice liscia di grado *s* se nei punti di giunzione degli intervalli è rispettata la continuità delle derivate dei polinomi fino al s-esimo ordine:

$$P_{i-1}^{(l)}(t_i) = P_i^{(l)}(t_i)\ \forall\ l = [0, s]$$

Di conseguenza $B \in \mathcal{C}^s(\mathbb{R})$.

Per l'approssimazione dei nodi generati tramite A* si è sfruttata una B-spline cubica *(n = 3)*.

Date le k coppie di coordinate (x_i, y_i), $i = [1, k]$ si generano due funzioni spline x e y definite rispetto alla coordinata curvilinea t e ottenute interpolando i singoli valori.

Si ha quindi una curva parametrica descritta come:

$$S: \mathbb{R} \to \mathbb{R}^2 \mid S(t) = \big(x(t), y(t)\big) \; con \; t \in [a, b]$$

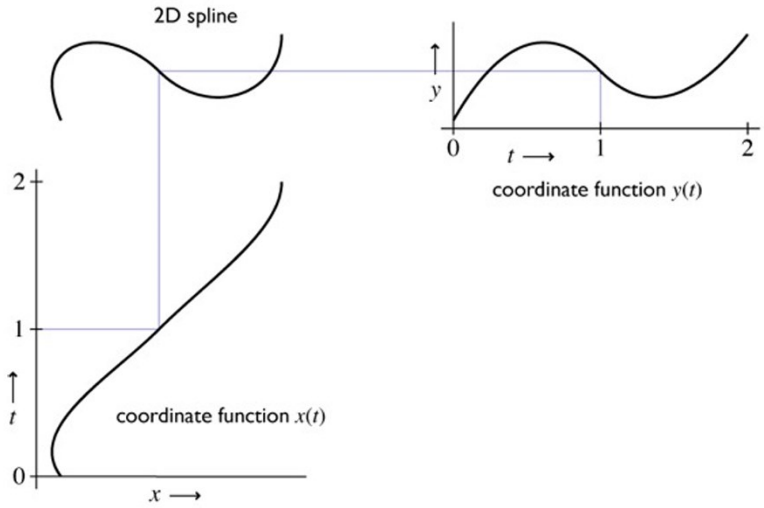


Figura 31: Composizione della B-spline a partire dalle singole componenti [24]

All'aumentare del grado di "liscezza" della funzione si ha un aumento sostanziale dell'errore rispetto alla traiettoria non interpolata. Si è verificato sperimentalmente che per navigazione in ambienti confinati un valore nell'intervallo $s \in [2,6]$ fornisce risultati accettabili. In Figura 32 si evidenzia in rosso la traiettoria originale e in blu quella interpolata.

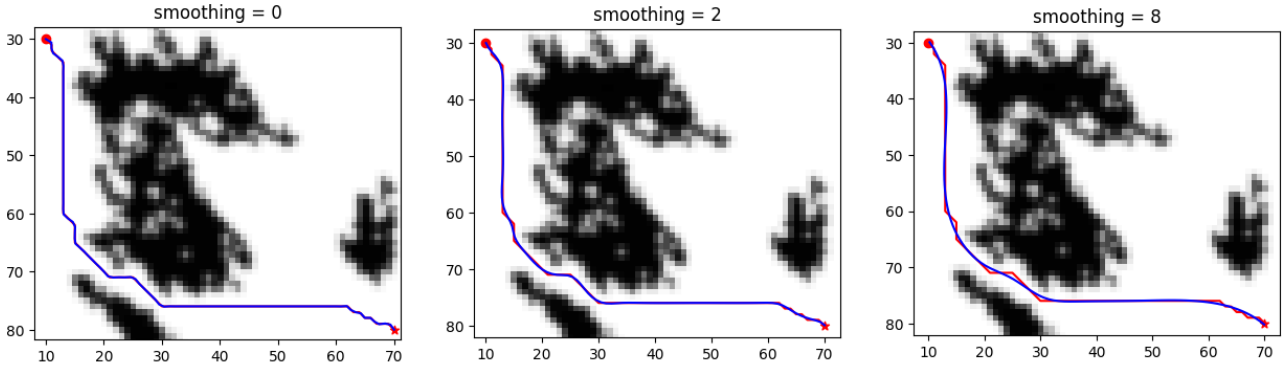


Figura 32: Variazione delle caratteristiche della spline interpolante al variare del parametro di smoothing

I coefficienti dei polinomi della spline vengono elaborati tramite il comando *splprep (spline preparation)*. Per ottenere un risultato discreto vengono generati 100 (1.01/*self.resolution*) punti equidistanti su cui ricavare i valori effettivi dei punti di passaggio tramite la funzione *splev (spline evaluation)*.

iv. Controllore di posizione

Ottenuti i punti di passaggio è necessario generare i comandi di velocità da inviare al controllore interno del rover.

Si generano due riferimenti di velocità: una velocità lineare, diretta lungo la direzione del moto, e una velocità angolare centrata sull'asse perpendicolare al pavimento e passante per il centro di massa del rover. Dal momento che tutte le ruote sono motorizzate è possibile compiere delle rotazioni sul posto, con un raggio di sterzata quasi nullo.

Si è optato per l'implementazione di un controllore closed-loop di tipo PID, Proporzionale-Integrale-Derivativo. In un controllore PID l'azione di controllo dipende da un'azione proporzionale all'errore, una proporzionale alla variazione dell'errore e una proporzionale all'integrale dello stesso.

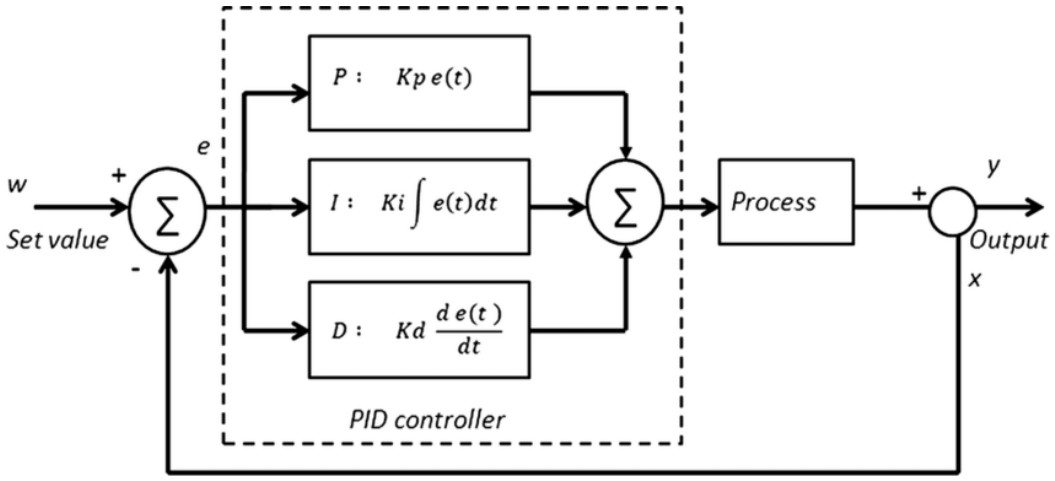


Figura 33: Schema a blocchi di un controllore PID

La funzione di trasferimento del sistema è composta dalla somma delle tre azioni:

$$y(t) = K_p \cdot e(t) + K_i \cdot \int_{t_0}^{t} e(t)dt + K_d \cdot \frac{d\big(e(t)\big)}{dt}$$

- L'azione proporzionale determina il guadagno statico del controllore e influenza l'errore di inseguimento per ogni frequenza;
- L'azione derivativa fornisce un'azione di stabilizzazione per sistemi del secondo ordine o oltre, è indice della prontezza del controllore e reagisce alle fluttuazioni istantanee dell'output;
- L'azione integrale fornisce uno storico dell'errore accumulato fino all'istante di valutazione e permette di annullare asintoticamente l'errore a regime.

L'implementazione realizzata sfrutta una discretizzazione del controllore PID:

$$out = K_p \cdot e + K_i \cdot (e + int) \cdot \Delta t + K_d \cdot \frac{e - e_{prev}}{\Delta t}$$

$$int = int + e$$

$$e_{prev} = e$$

Dove *int* è la variabile di accumulo che funge da azione integrale memorizzando gli stati precedenti. La funzione del controllore viene richiamata e aggiornata ciclicamente. È da considerare il fatto che nel codice si assume un tempo unitario tra richiami successivi della funzione per semplificare il calcolo, in realtà il Δt dipende dal tempo di esecuzione del resto del codice e delle subroutine di funzionamento interne al computer di controllo e quindi risulta variabile.

Nella pratica, a meno di qualche variazione sporadica, il lasso di tempo tra iterazioni successive resta sufficientemente stabile, la differenza di questo dall'unità viene compensata modificando i valori K_d e K_i. È comunque possibile utilizzare il Δt corretto misurando il tempo tra richiami successivi della funzione, basandosi sull'orologio di sistema.

```
13. class PID:
24.     def update(self, error):
25.         self.error = error
26.         self.p = self.Kp * self.error
27.         self.integral = self.Ki * (self.error + self.integral)
28.         self.integral = max(min(self.integral_max, self.integral),
                                self.integral_min)
29.         self.derivate = self.Kd * (self.error - self.derivate)
30.         out = self.p + self.integral + self.derivate
31.         return out
```

La velocità massima delle ruote del rover è limitata a causa delle caratteristiche meccaniche del sistema, ma viene ulteriormente ridotta via software per avere un margine di sicurezza.

In presenza di un errore considerevole può accadere che il contributo dell'azione integrale aumenti notevolmente fino a raggiungere i limiti di saturazione del sistema. In questo caso anche se l'errore dovesse iniziare a ridursi non si avrebbe una diminuzione dell'uscita del processo in quanto questa rimarrebbe nella zona di saturazione per il tempo necessario al termine integrale di ridursi entro margini accettabili.

L'effetto risultante sul sistema è la comparsa di un ritardo tra l'applicazione del comando e la risposta da parte del plant.

Per eliminare (o quantomeno ridurre) la latenza si utilizza una configurazione *anti-windup*: lo stato dell'integratore viene limitato entro un margine prestabilito (linea 28).

Le variabili controllate in questo caso non sono accoppiate tra loro: il controllo viene volto parallelamente su entrambe e le velocità calcolate in uscita sono semplicemente sommate vettorialmente tra loro.

Il sistema di controllo della posizione riceve come dati in ingresso la posizione attuale del rover, fornita dal nodo di localizzazione tramite l'odometria, e le coordinate del punto da raggiungere.

```
291.        # Calcolo errore angolare
292.        ang_target = atan2(prox_wp[1] - self.pos_attuale[1], prox_wp[0] -
                        self.pos_attuale[0])
293.        self.err_ang = self.yaw_attuale - ang_target
294.        v_ang = self.pid_ang.update(-self.err_ang)
295.
296.        # Calcolo errore di posizione
297.        dir = np.sign(prox_wp[0] - self.pos_attuale[0]) # +- 1
298.        self.err_pos = dir * np.sqrt((prox_wp[0] - self.pos_attuale[0]) ** 2 +
                         (prox_wp[1] - self.pos_attuale[1]) ** 2)
299.        v_lin = self.pid_lin.update(self.err_pos)
300.
301.        # Si limita l'output entro le capacità del rover
302.        v_ang = max(min(self.vbound_ang, v_ang), -self.vbound_ang)
303.        v_lin = max(min(self.vbound_lin, v_lin), -self.vbound_lin)
304.
305.        # Movimento verso il waypoint
306.        self.vel.angular.z = v_ang
307.        self.vel.linear.x = v_lin
```

Si calcola inizialmente l'errore direzionale: noti l'angolo di direzione χ (derivato dall'angolo di imbardata ψ) e le coordinate del punto, si ha:

$$\Delta\vartheta = \chi - atan2\left(\frac{y_{waypoint} - y_{attuale}}{x_{waypoint} - x_{attuale}}\right)$$

Si sfrutta il comando *atan2* per il calcolo dell'arcotangente in quanto preserva le informazioni sui segni date dall'orientazione del problema.

Per il calcolo dell'errore di posizione si stima semplicemente la distanza euclidea tra il rover e il waypoint:

$$\Delta x = \sqrt{\left(x_{wp} - x_{att}\right)^2 + \left(y_{wp} - y_{att}\right)^2}$$

I due errori sono quindi inviati ai rispettivi controllori PID per il calcolo dell'azione di controllo, che dopo essere stata limitata entro una velocità massima e minima è inviata al publisher per essere pubblicata sul topic dedicato.

Per quanto riguarda la taratura dei coefficienti K_p, K_i, K_d, sono disponibili numerose procedure empiriche o numeriche, come ad esempio la procedura Ziegler-Nichols [16].

Nella procedura Ziegler-Nichols si annullano i guadagni derivativo e integrale e si aumenta il guadagno proporzionale fino all'innescarsi di oscillazioni periodiche. Si misura a questo punto il periodo delle oscillazioni T_u e il guadagno massimo raggiunto K_u.

I parametri di taratura si ricavano con le seguenti formule:

Ziegler-Nichols	K_p	τ_{int}	τ_{der}
P	$K_u/_2$		
PI	$K_u/_{2.2}$	$T_u/_{1.2}$	
PID	$K_u/_{1.7}$	$T_u/_2$	$T_u/_8$

Ottenute le costanti di tempo è possibile ricavare i coefficienti.

Un controllore PID regolato tramite procedura Ziegler-Nichols porta ad avere una risposta del sistema smorzata critica, ovvero si ha un transitorio di risposta senza oscillazioni il più breve possibile.

Tra le altre procedure più impiegate si ricordano il metodo Cohen-Coon (sfruttando la risposta a ciclo aperto) e il metodo Tyreus-Luyben per sistemi a dinamica rapida.

L'impianto traente del rover è formato da quattro motori elettrici bipolari a magneti permanenti in grado di generare fino a 100 Ncm di coppia ad una velocità di circa 40 rpm [17]. Data l'inerzia ridotta del sistema si ha una risposta praticamente impulsiva alle variazioni di velocità, di conseguenza la dinamica della variazione dell'errore è estremamente rapida. Sarebbe stato sufficiente l'impiego di un controllore P, tuttavia, per ridurre l'errore a regime che caratterizza questo tipo di sistemi e per evitare oscillazioni, i controllori di velocità lineare e angolare sono stati impostati in configurazione PI.

I coefficienti impiegati nel progetto sono stati ricavati tramite la procedura Ziegler-Nichols e modificati iterativamente fino all'ottenimento di un comportamento stabile.

4. Risultati sperimentali

Si è verificato sperimentalmente il funzionamento dell'algoritmo di navigazione. Sono stati svolti vari esperimenti, viene riportato in questa tesi l'ultimo di essi. L'esperimento è stato svolto all'interno del laboratorio in condizioni di buona illuminazione, i dati forniti sono stati consistenti con quelli degli altri effettuati. In Figura 34 è riportata la sequenza che porta, a partire dall'immagine statica dell'ambiente e dai dati di profondità, all'ottenimento della mappa bidimensionale del piano di navigazione. La Figura 35 mostra invece l'interfaccia di controllo principale.

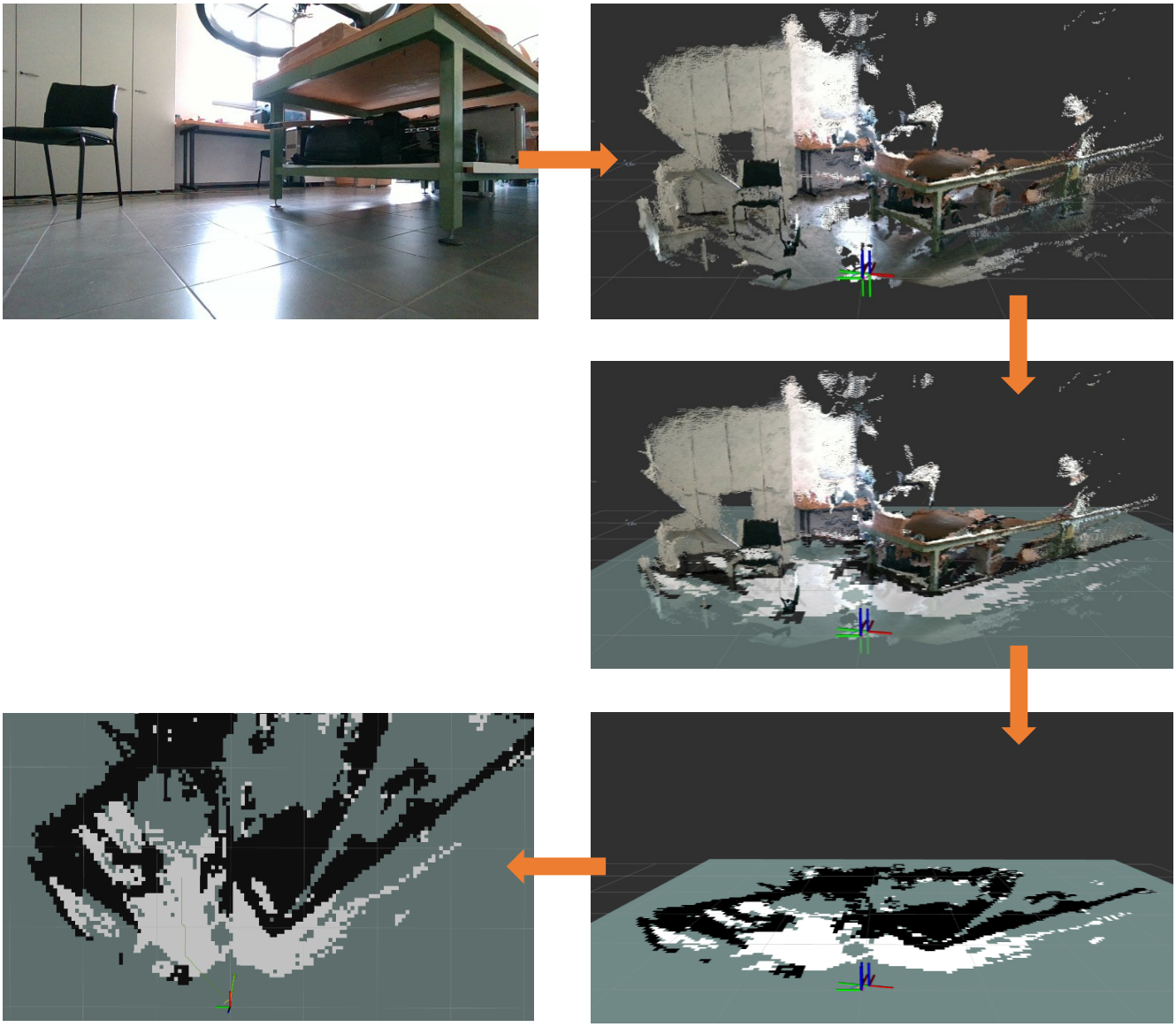


Figura 34: Passaggio da immagine acquisita, a nuvola di punti 3D, a mappa 2D

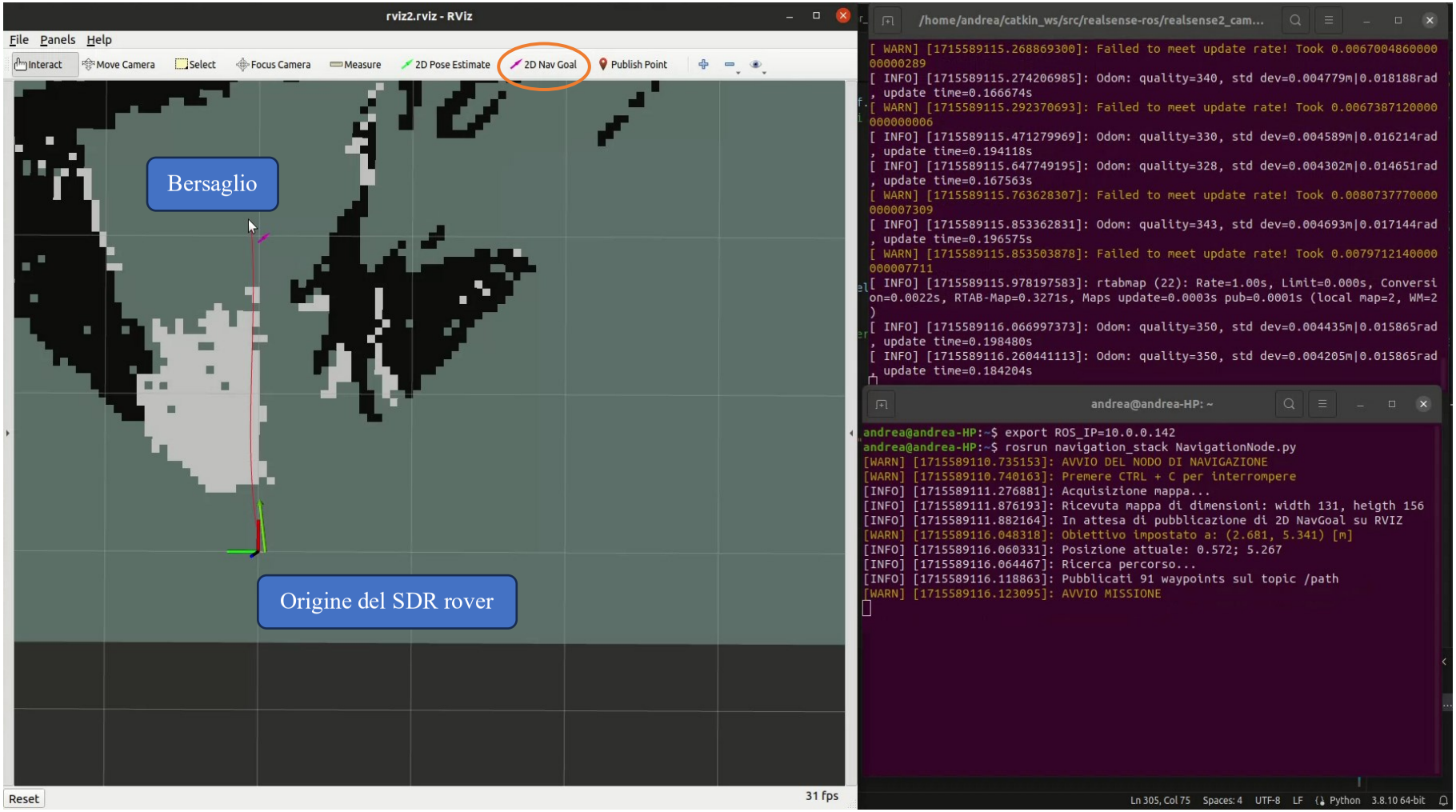


Figura 35: Interfaccia RVIZ e del nodo di navigazione

L’immagine in Figura 36 è stata ottenuta sovrapponendo alla mappa visualizzata in RVIZ fotogrammi successivi della registrazione dell’esperimento. La durata complessiva è stata di circa 13 secondi, con una distanza totale coperta di circa 1,88 metri.

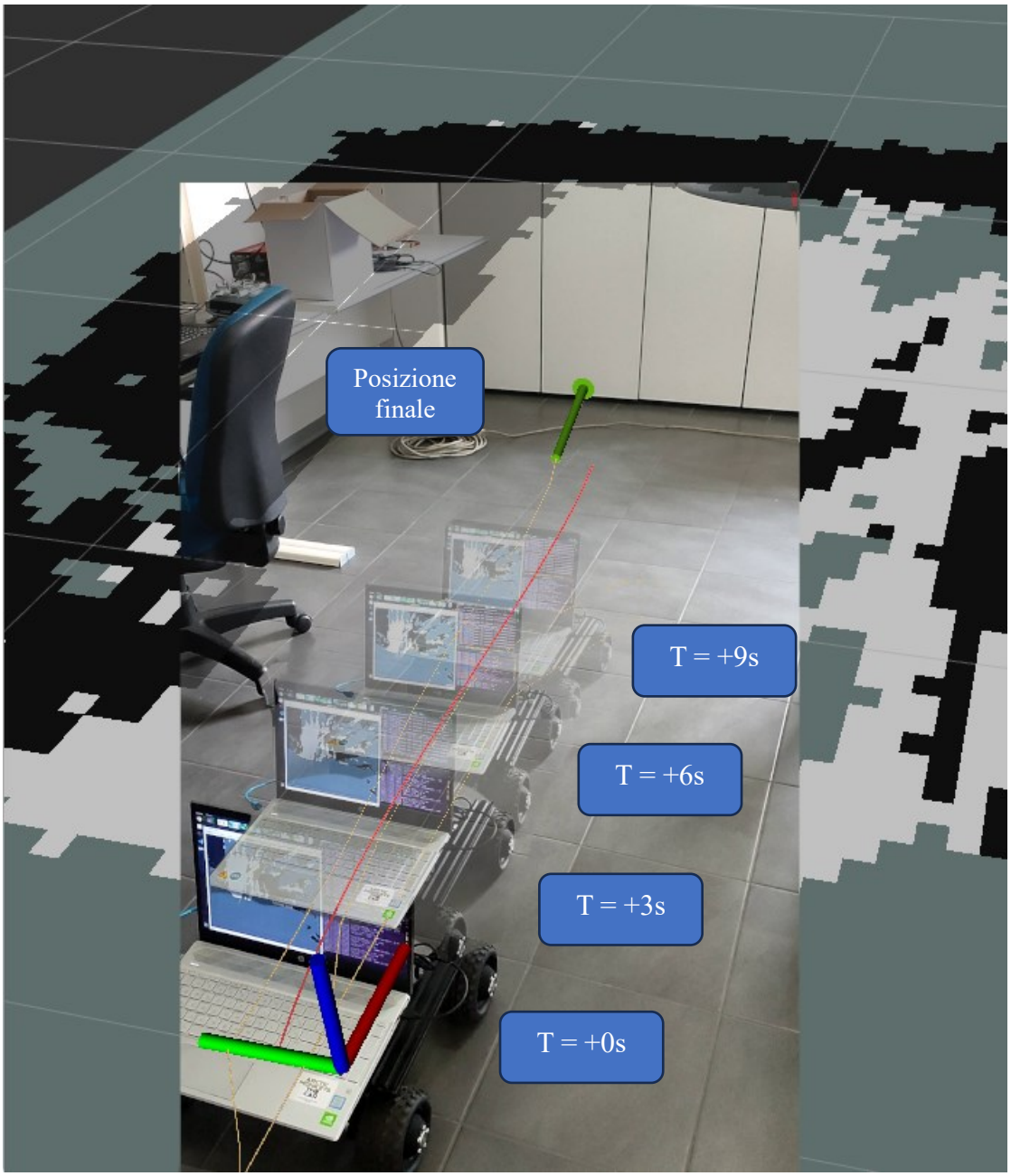


Figura 36: Sovrimpressione della traiettoria seguita dal rover con quella generata dal controllore

In rosso è indicata la traiettoria desiderata generata tramite A*, mentre in giallo vi è la traiettoria realmente percorsa. Si noti che le due traiettorie differiscono non per errori di calcolo ma per geometria: la linea gialla traccia il moto del centro di massa del rover, al contrario della linea rossa che considera il centro delle caselle della mappa. Dal momento che le caselle hanno una dimensione discreta il sistema considera la posizione più vicina al centro del rover istante per istante, portando a questa differenza.

La velocità massima lineare e angolare è stata limitata rispettivamente a 0,25 m/s e a 0,4 rad/s; la soglia di tolleranza dell'errore per il raggiungimento del waypoint è stata invece fissata a 15 cm. I guadagni dei controllori PID per la posizione e l'angolo sono stati impostati a (1, 0.01, 0) e (0.75, 0.1, 0) rispettivamente (P, I, D).

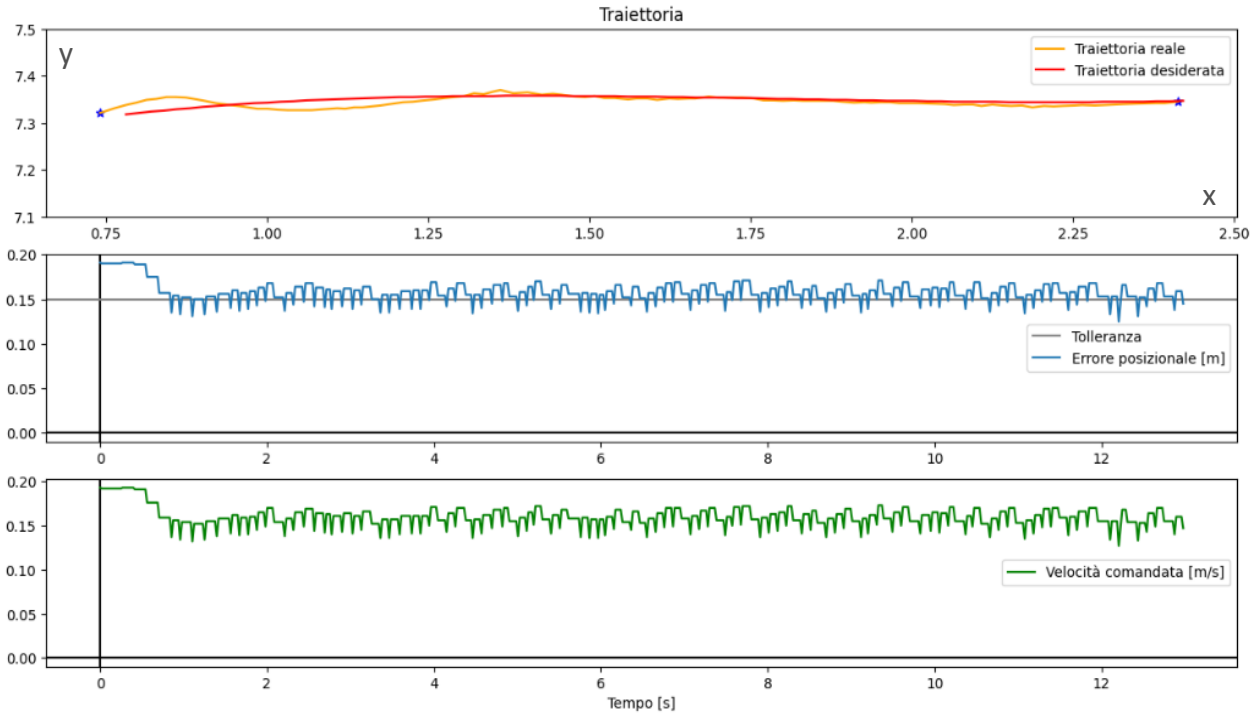


Figura 38: Comandi generati dal controllore di posizione

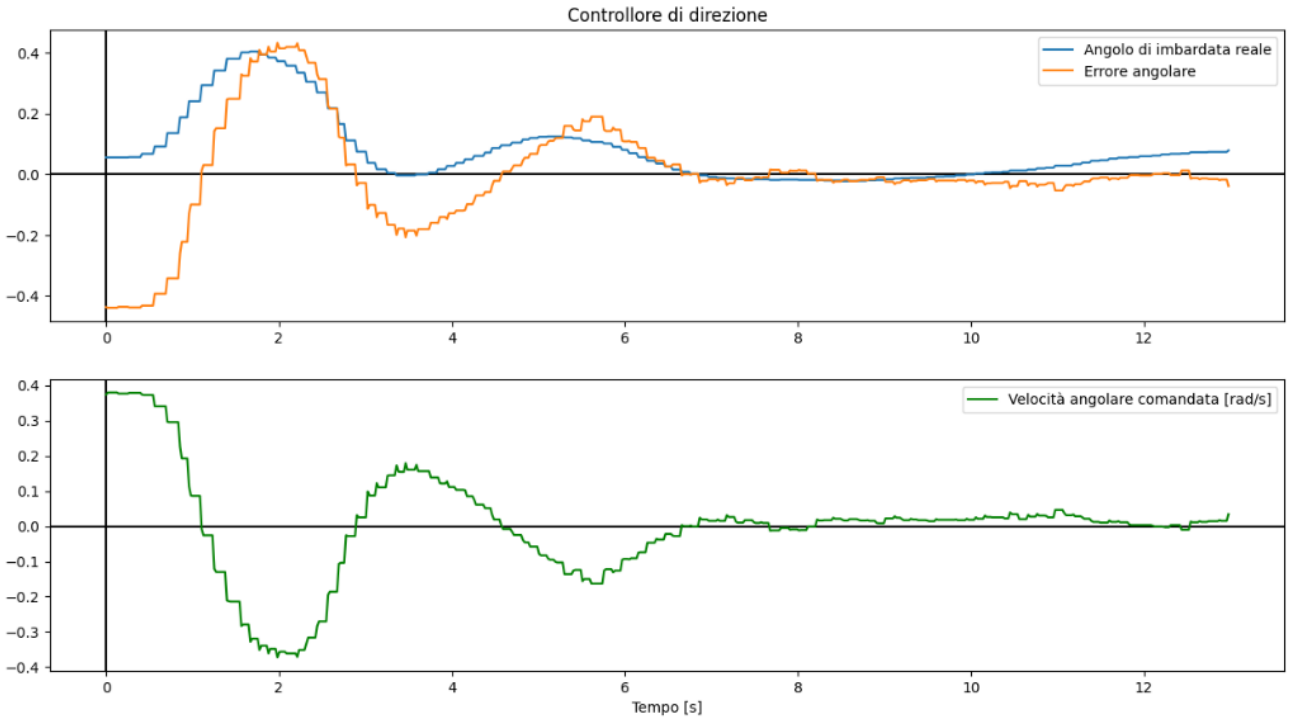


Figura 37: Comandi generati dal controllore di direzione

Ogni picco nell'errore o nella velocità comandata di Figura 38 coincide con il raggiungimento del waypoint desiderato: istantaneamente il controllore cambia il bersaglio e la ricerca viene aggiornata. L'ampiezza dei picchi diminuisce all'infittirsi dei punti che costituiscono la traiettoria mentre aumenta all'aumentare della tolleranza posizionale richiesta.

Dal momento che la precisione dell'odometria non è eccessivamente accurata, diminuendo troppo il valore della tolleranza è possibile sorpassare il waypoint selezionato. Questa situazione può essere esacerbata dal fatto che sotto una certa velocità limite (circa 0,1 m/s) il rover cessa di avere un comportamento lineare e, a seconda anche della carica residua della batteria e dell'attrito tra copertone e pavimento, può fermarsi del tutto o avere piccole oscillazioni nella velocità.

Nel caso un waypoint venisse sorpassato il sistema invertirà la direzione della velocità lineare per recuperare, con la possibilità tuttavia di avere delle oscillazioni centrate sul punto stesso. Infine, in questa situazione il calcolo dell'angolo di sterzata diventa instabile: a piccole variazioni di posizione corrispondono grandi variazioni nel valore dell'angolo. Cade quindi l'ipotesi di indipendenza tra l'errore angolare e quello lineare e il sistema di controllo smette di funzionare a dovere.

Il sistema si è dimostrato sufficientemente robusto da affrontare percorsi di lunghezza variabile e con curve moderate in maniera consistente, con prestazioni ridotte solo per virate maggiori di 90°. Si è rivelata invece sorprendentemente laboriosa la regolazione dei parametri dei controllori PID, che è necessario variare ogni qual volta che venga modificato il limite della velocità massima. È plausibile avere una piattaforma mobile a 0,25 m/s per operazioni di fino in ambienti ristretti ma non per operazioni lunghe o in campo aperto.

Sono stati osservati anche alcuni fenomeni negativi legati all'utilizzo di un sistema di navigazione puramente ottico. In primo luogo, la degradazione dell'odometria durante la fase di mappatura iniziale e durante la missione: all'avvio del sistema l'errore di posizione è pressoché nullo, già dopo pochi movimenti però la posizione reale inizia a differire sensibilmente da quella indicata dal sistema di odometria. Questa degradazione è di portata tale da rendere inutilizzabile il sistema di localizzazione dopo pochi esperimenti, richiedendo il riavvio del nodo *realsense2* e la cancellazione della mappa.

Un effetto di questa problematica è visibile nel momento in cui il sistema di input video dovesse rilevare un repentino cambio nell'ambiente osservato, tipicamente dovuto a un

movimento brusco o ad una temporanea interruzione del flusso dati. In teoria, se riportato in condizioni ideali, il sottoprocesso di loop closure dovrebbe riconoscere features già note dell'ambiente e recuperare l'orientamento relativo; questo però avviene piuttosto raramente, con la conseguenza di una perdita totale di affidabilità dei dati ottici che richiede il riavvio del nodo.

Dal momento che il nodo *Realsense2* impiega la tecnica EKF-SLAM, le cause sono riconducibili alla forte non-linearità ed agli errori di loop closure. EKF-SLAM è particolarmente suscettibile ad associazioni errate delle features che, a causa dell'accumulo dell'errore, aumentano in frequenza all'aumentare della distanza percorsa. Dove applicabile, è possibile ridurre l'errore distribuendo nell'ambiente di navigazione dei catadiottri o dei generici marcatori per favorire l'acquisizione dei landmark.

Un altro fenomeno osservato è stata la pesante dipendenza dalle condizioni di illuminazione ambientali: una brusca variazione di luminosità può essere interpretata dal sistema come movimento improvviso, portando alle conseguenze di cui sopra. Potrebbe essere utile sostituire il modulo ottico con un modulo ad infrarossi oppure installare un sistema di illuminazione sul rover stesso, soprattutto se ne si prevede l'impiego in spazi ristretti o angusti.

Infine, è d'obbligo ricordare che il nodo di navigazione deve essere eseguito esclusivamente sul computer di controllo del rover, o ancora meglio direttamente sul RPI nativo dello stesso. Sebbene la mappatura venga eseguita prima della fase di navigazione, il nodo di localizzazione ottica è utilizzato istante per istante per ricavare l'odometria. A causa dell'elevato flusso dati il canale wireless del rover tende a diventare instabile e presentare notevoli ritardi di comunicazione. Considerando anche che il nodo è sprovvisto di sistemi di conferma della ricezione dei dati per ridurre il carico si ha che, con il nodo in esecuzione sulla ground station, l'odometria ricevuta dal rover possa essere incompleta e notevolmente in ritardo. Considerando una velocità media di 0,2 m/s, con una latenza di 2 s (valore elevatissimo, ma non uno dei più alti registrati) si possono avere errori di posizionamento nell'ordine dei 40/80 cm. Il controllo, in queste condizioni, non è efficace.

In definitiva, per percorsi sufficientemente regolari eseguiti in ambienti provvisti di un certo "interesse" ottico il sistema di navigazione ha fornito risultati più che accettabili. È tuttavia consigliabile rivedere la parte di acquisizione dell'odometria e passare a sistemi di controllo di tipo multivariabile [18].

5. Conclusioni

Lo sviluppo di un sistema autonomo è stata una sfida complessa che ha presentato non pochi punti critici da risolvere. È stato necessario imparare un sistema di programmazione completamente nuovo, oltre ad una notevole quantità di teoria dei segnali e dei controlli automatici.

Si propongono alcuni spunti per gli sviluppi futuri:

- Sistema di esplorazione automatica: sviluppo di una routine di mappatura che, data un'estensione massima, ricava il percorso ottimale per mappare l'interezza dell'area scelta.
- Real-time obstacle avoidance: nel caso venga rilevata una variazione consistente nella mappa dell'ambiente si interrompa la navigazione per generare un segmento di traiettoria che, sostituito a quello bloccato da un ostacolo imprevisto, permetta di arrivare in tempo utile al bersaglio. Questo segmento di traiettoria deve rispettare le condizioni di continuità e derivabilità della traiettoria originale. Può essere impiegato un approccio come TOPP-RA [19].
- Sistema di salvataggio della mappa: implementare un sistema che, al termine della mappatura, salvi la mappa dell'ambiente in un file accessibile in un secondo momento. Alla successiva apertura del file il sistema deve essere in grado di risalire alla posizione attuale confrontando il frammento della nuova mappa osservata con la mappa salvata.
- Aggiunta di un modulo lidar/gps per incrementare la precisione dell'odometria.
- Misurazione del Δt per le istanze dei controllori PID, da eseguire sfruttando il comando *rospy.get_time* per accedere all'orologio interno del server ROS.
- Navigazione verso waypoint multipli: si è sviluppato in simulazione un sistema di navigazione che generasse una traiettoria continua sfruttando A* passante per più waypoint. Questi ultimi possono essere specificati allo stesso modo dell'inserimento target del sistema attuale, pubblicandoli quindi come lista su un topic, oppure accedendovi via file. Una versione rifinita di questo sistema dovrebbe rispettare continuità e derivabilità. Si può anche definire automaticamente l'ordine di passaggio dai waypoint cercando la distanza minore [20]. Il problema diventa del tipo di copertura dei vertici, è possibile risolverlo con procedure analitiche ma per ridurre il costo computazionale euristicamente si è verificato che è ben risolvibile con il metodo *simulated annealing*.

Grazie alla flessibilità di ROS e per come è estato sviluppato il sistema, questo impianto può essere adattato anche a velivoli. Sarebbe possibile convertire il sistema per il funzionamento 3D sfruttando la stessa tipologia di sensori, generando una mappa tridimensionale degli ostacoli considerando non più pixel ma voxel.

Per concludere, il sistema proposto si è rivelato in grado di effettuare una navigazione autonoma con una percentuale di successo e ripetibilità sufficientemente elevata.

Questo campo presenta molte possibilità e spunti per nuove ricerche. Ho potuto provare con mano la potenza e la flessibilità del sistema ROS e i limiti delle tecnologie di navigazione autonoma.

Ringraziamenti

Vorrei ringraziare innanzitutto il professor Fabrizio Giulietti per avermi permesso di mettermi in gioco e di mostrare le mie competenze attraverso un progetto complesso e stimolante.

Un ringraziamento è assolutamente dovuto alla dottoressa Giulia Bertolani, al dottor Daniele Fattizzo ed Elia Costantini, del laboratorio di Meccanica del Volo, per il loro più che prezioso supporto tecnico e morale durante lo sviluppo e il debug del sistema.

Inoltre, vorrei ringraziare i miei amici e colleghi del corso di studio per il supporto costante e per essere stati un'ottima fonte di dibattito e confronto.

Per ultima, ma non per importanza, ringrazio la mia famiglia per essermi stata accanto ed avermi appoggiato in ogni momento del mio percorso universitario e della mia vita.

Bibliografia

[1] J. Vardalas, «Early Digital Technology and the Navy,» [Online]. Available: https://ethw.org/Early_Digital_Technology_and_the_Navy.

[2] M. S. Grewal, L. R. Weill e A. P. Andrews, «Global Positioning Systems, Inertial Navigation and Integration,» John Wiley & Sons Inc., 2007.

[3] F. Golnaraghi e B. C. Kuo, Automatic control systems, John Wiley & Sons inc., 2010.

[4] M. Zanzi, Dispense del corso di Elaborazione Dati per la Navigazione.

[5] S. Thrun, W. Burgard e D. Fox, Probabilistic Robotics, The MIT Press, 2005.

[6] S. W. Smith, The Scientst and Engineer's Guide to Digital Signal Processing, San Diego, California: California Technical Publishing.

[7] B. L. Stevens e F. L. Lewis, Aircraft Control and Simulation, Hoboken, New Jersey: John Wiley & Sons Inc., 2003.

[8] C. Casarosa, Meccanica del Volo, Pisa: Pisa University press, 2013.

[9] S. Madgwick, «An efficient orientation filter for inertial and inertial/magnetic sensor arrays,» 2010.

[10] H. Durrant-Whyte e T. Bailey, «Simultaneous Localization and Mapping: Part I The Essential Algorithms».

[11] H. Durrant-Whyte e T. Bailey, Simultaneous Localization and Mapping (SLAM): Part II State of the Art.

[12] A. Grunnet-Jepsen e D. Tong, «Depth Post-Processing for Intel® RealSense™ Depth Camera D400 Series,» [Online]. Available: https://dev.intelrealsense.com/docs/depth-post-processing.

[13] P. Krejov e A. Grunnet-Jepsen, «Intel® RealSense™ Depth Camera over Ethernet,» [Online]. Available: https://dev.intelrealsense.com/docs/depth-camera-over-ethernet-whitepaper.

[14] R. C. Gonzales e R. E. Woods, Digital Image Processing, Pearson Education Limited, 2018.

[15] M. Labbè e F. Michaud, «RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation,» *Journal of Field Robotics,* vol. 36, n. 2, pp. 416-446, 2019.

[16] T. Budd, An Introduction to Object-Oriented Programming, Pearson College Div, 2001.

[17] D. Foead, A. Ghifari, M. B. Kusuma, N. Hanafiah e E. Gunawan, «A Systematic Literature Review of A* Pathfinding,» *Procedia Computer Science,* vol. 179, pp. 507-514, 2021.

[18] SRI International, «Shakey the Robot,» [Online]. Available: https://www.sri.com/hoi/shakey-the-robot/.

[19] M. S. Floater, An introduction to Spline Theory, Department of Mathematics, University of Oslo, 2023.

[20] J. G. Ziegler e N. B. Nichols, «Optimum Settings for Automatic Controllers,» Rochester, N. Y., 1942.

[21] Bühler Motors, «1.61.077.414 Geared Motor Datasheet,» [Online]. Available: http://www.mobot.pl/download/1_61_077_4xx_en.pdf.

[22] S. Skogestad e I. Postlethwaite, MULTIVARIABLE FEEDBACK CONTROL Analysis and Design, John Wiley & Sons, 2001.

[23] H. Pham e Q. C. Pham, «A New Approach to Time-Optimal Path Parameterization Based on Reachability Analysis,» vol. IEEE Transactions on Robotics 34, n. 3, pp. 645 - 659, June 2018.

[24] A. Dal Pozzo, «Relazione finale di tirocinio curriculare, Studio, analisi e implementazione di sensori non convenzionali e algoritmi GNC di applicazione in ambito aerospaziale,» Forlì, 2023.

[25] S. Mokssit, D. B. Licea, B. Guermah e M. Ghogho, «Deep Learning Techniques for Visual SLAM: A Survey,» [Online]. Available: https://ieeexplore.ieee.org/document/10054007.

[26] «Implementation of Gaussian Blur,» [Online]. Available: https://computergraphics.stackexchange.com/questions/39/how-is-gaussian-blur-implemented.

[27] S. A. R. Florez, V. Fremont, P. Bonnifait e V. Cherfaoui, «An Embedded Multi-Modal System for Object Localization and Tracking,» in *Conference: Intelligent Vehicles Symposium (IV), IEEE*, 2010.

[28] S. Marschner, «2D Spline Curves,» in *Cornell CS4620 - Lecture 16*, 2013.

[29] D. H. Titterton e J. L. Weston, Strapdown Inertial Navigation Technology, Stevenage, United Kingdom: The Institution of Electrical Engineers, 2004.

[30] k. Mohammed e S. Srichitra, «Feature Extraction and Description Pipeline in Visual SLAM,» 28 Nov 2023. [Online]. Available: https://ignitarium.com/feature-extraction-and-description-pipeline-in-visual-slam.

[31] Analog Circuit Design (ACD), «Stability of control systems,» 2024. [Online]. Available: https://analogcircuitdesign.com/stability-of-control-systems/.

Appendice 1: Algoritmo di navigazione

```
#!/usr/bin/env python

import rospy
import tf
import numpy as np
import heapq
from math import pi, atan2
from scipy import interpolate
from scipy.ndimage import gaussian_filter
from geometry_msgs.msg import Twist, PoseStamped
from nav_msgs.msg import Odometry, OccupancyGrid, Path

class PID:
    def __init__(self, P=0.0, I=0.0, D=0.0, derivate=0, integral=0, integral_max=10, integral_min=-
10):
        self.Kp = P
        self.Ki = I
        self.Kd = D
        self.derivate = derivate
        self.integral = integral
        self.integral_max = integral_max
        self.integral_min = integral_min
        self.error = 0.0

    def update(self, error):
        self.error = error
        self.p = self.Kp * self.error
        self.integral = self.Ki * (self.error + self.integral)
        self.integral = max(min(self.integral_max, self.integral), self.integral_min)
        self.derivate = self.Kd * (self.error - self.derivate)
        out = self.p + self.integral + self.derivate
        return out

    def setParam(self, set_P=0.0, set_I=0.0, set_D=0.0):
        self.Kp = set_P
        self.Ki = set_I
        self.Kd = set_D

class getTrajectory():
    def __init__(self, resolution, smoothing, mapsize, mapresolution, neighmode):

        if neighmode == 0:
            self.neighbors = [(0,1),(0,-1),(1,0),(-1,0)]     # Solo quelli ortogonali
        else:
            self.neighbors = [(0,1),(0,-1),(1,0),(-1,0),(1,1),(1,-1),(-1,1),(-1,-1)]

        self.resolution = resolution
        self.smoothing = smoothing
        self.out = []
        self.dist = []
        self.mapSize = mapsize
        self.mapResolution = mapresolution

    def dist_euclidea(self, x, y):
        rospy.logdebug(str(x))
        rospy.logdebug(str(y))
        return np.sqrt((y[0] - x[0]) ** 2 + (y[1] - x[1]) ** 2)

    def astar(self, array, start, goal):
        close_set = set()
        came_from = {}
        gscore = {start:0}
        fscore = {start:self.dist_euclidea(start, goal)}
        oheap = []
        heapq.heappush(oheap, (fscore[start], start))

        while oheap:
            current = heapq.heappop(oheap)[1]
            if current == goal:
                data = []
```

```
70.                 while current in came_from:
71.                     data.append(current)
72.                     current = came_from[current]
73.                 return data
74.
75.             close_set.add(current)
76.             for i, j in self.neighbors:
77.                 neighbor = current[0] + i, current[1] + j
78.                 tentative_g_score = gscore[current] + self.dist_euclidea(current, neighbor)
79.                 if 0 <= neighbor[0] < array.shape[0]:
80.                     if 0 <= neighbor[1] < array.shape[1]:
81.                         if array[neighbor[0]][neighbor[1]] == 1:
82.                             continue
83.                     else: # array bound y walls
84.                         continue
85.                 else: # array bound x walls
86.                     continue
87.
88.                 if neighbor in close_set and tentative_g_score >= gscore.get(neighbor, 0):
89.                     continue
90.                 if  tentative_g_score < gscore.get(neighbor, 0) or neighbor not in [i[1]for i in
                       oheap]:
91.                     came_from[neighbor] = current
92.                     gscore[neighbor] = tentative_g_score
93.                     fscore[neighbor] = tentative_g_score + self.dist_euclidea(neighbor, goal)
94.                     heapq.heappush(oheap, (fscore[neighbor], neighbor))
95.
96.     def calcPath(self, mappa, wp, pos_attuale):
97.         pos = (round(pos_attuale[0]/self.mapResolution), self.mapSize[1] -
           round(pos_attuale[1]/self.mapResolution))  # Conversione da metri in unità di mappa
98.         wp = (round(wp[0]/self.mapResolution), self.mapSize[1] - round(wp[1]/self.mapResolution))
99.         mappa = np.rot90(mappa, k=-1)                # Rotazione oraria di 90 gradi
100.
101.        rospy.loginfo("Ricerca percorso...")
102.        x_coords = []
103.        y_coords = []
104.        route = []
105.
106.        route = self.astar(mappa, wp, pos)
107.
108.        if route == None:
109.            rospy.logwarn("L'algoritmo di navigazione non ha trovato un percorso")
110.        else:
111.            #route = route[::-1]
112.
113.            for i in (range(0,len(route))):
114.                x = route[i][0]
115.                y = route[i][1]
116.                x_coords.append(x)
117.                y_coords.append(self.mapSize[1] - y)
118.
119.            if self.smoothing > 0:
120.                tck, u = interpolate.splprep([x_coords,y_coords], s=self.smoothing)
                       # Interpolazione spline per ottenere una traiettoria continua e liscia
121.                unew = np.arange(0, 1.01, self.resolution)
122.                self.out = interpolate.splev(unew, tck)
123.            else:
124.                self.out = [x_coords, y_coords]
125.
126.            self.dist = 0
127.            for i in range(len(self.out[0])-1):
128.              self.dist = self.dist +
                  self.dist_euclidea((self.out[0][i],self.out[1][i]),(self.out[0][i+1],self.out[1][i+1]))
129.              rospy.loginfo("Trovato un percorso di lunghezza: " + str(round(self.dist,3)) + " m")
130.
131. class navigation():
132.
133.     def odom_callback(self, msg):
134.     # Si ricavano i parametri di posizione e orientamento del rover rispetto all'origine della mappa
135.         quarternion = [msg.pose.pose.orientation.x, msg.pose.pose.orientation.y,
           msg.pose.pose.orientation.z, msg.pose.pose.orientation.w]
136.         (roll, pitch, yaw) = tf.transformations.euler_from_quaternion(quarternion)
137.         self.yaw_attuale = yaw
```

```
138.         self.pos_attuale = (msg.pose.pose.position.x - self.mapOrigin[0], msg.pose.pose.position.y -
             self.mapOrigin[1])                              # Posizione rispetto all'origine della mappa
139.
140.         self.counter += 1
141.         if self.counter == 50:  # Stampa odometria alla frequenza di 100/50 = 2Hz
142.             self.counter = 0
143.             self.trajectory.append([self.pos_attuale[0],self.pos_attuale[1]])
                 # Registrazione traiettoria seguita fin ora
144.
145.     def map_callback(self, msg):
146.         map = msg.data
147.         self.mapSize = (msg.info.width, msg.info.height)
             # RTABMAP genera una mappa divisa in quadretti da 5 cm di lato
148.         # (-1 non mappato, 0 libero, altro ostacolo)
149.         self.mapOrigin = (msg.info.origin.position.x, msg.info.origin.position.y) # Espresso in metri
150.         self.mapResolution = msg.info.resolution
151.         self.map = np.reshape(map, (self.mapSize[1], self.mapSize[0]))
152.
153.         if self.navMode == 0:
154.             self.map = np.where(self.map != 0, 1, 0)    # 0 - Navigazione solo su suolo certo
155.             self.map = gaussian_filter(self.map * 100, sigma=self.sigma)
                 # Sfocatura gaussiana: si aggiunge un offset agli ostacoli
156.             self.map = np.where(self.map != 0, 1, 0)
157.         elif self.navMode == 1:
158.             self.map = np.where(self.map > 0, 1, 0)     # 1 - Navigazione anche su suolo non mappato
159.             self.map = gaussian_filter(self.map * 100, sigma=self.sigma)
160.             self.map = np.where(self.map > 0, 1, 0)
161.         elif self.navMode == 2:
162.             self.map = np.where(self.map > 0, 0, 0)     # 2 - Navigazione ovunque, ostacoli compresi
163.         else:
164.             rospy.logfatal("Errore impostazione navMode")
165.
166.     def wp_callback(self, msg):
167.         data = msg.pose.position
168.         self.wp = []
169.         if data != []:
170.             self.wp = (data.x + self.pos_attuale[0], data.y + self.pos_attuale[1])
                 # I waypoint sono dati in relazione alla posizione del rover
171.             rospy.logwarn("Obiettivo impostato a: " +
            str((round(self.wp[0],self.logRoundingPrecision),
            round(self.wp[1],self.logRoundingPrecision))) + " [m]")
172.
173.     def publish_waypoints(self):
174.         # Pubblicazione della traiettoria da seguire sul topic /path
175.         msg = Path()
176.         msg.header.frame_id = "map"
177.         msg.header.stamp = rospy.Time.now()
178.         if self.pathfinder.out is not None:
179.             for i in range(self.dropIndex, np.shape(self.pathfinder.out)[1]):
180.                 pose = PoseStamped()
181.                 pose.pose.position.x = self.pathfinder.out[0][i] * self.mapResolution +
                        self.mapOrigin[0] # Metri
182.                 pose.pose.position.y = self.pathfinder.out[1][i] * self.mapResolution +
                    self.mapOrigin[1]
183.                 rospy.logdebug("x: " + str(pose.pose.position.x) + ", y: " +
                        str(pose.pose.position.y))
184.                 pose.pose.position.z = 0
185.                 pose.pose.orientation.x = 0
186.                 pose.pose.orientation.y = 0
187.                 pose.pose.orientation.z = 0
188.                 pose.pose.orientation.w = 1
189.                 msg.poses.append(pose)
190.         self.waypoint_pub.publish(msg)
191.         rospy.loginfo("Pubblicati {} waypoints sul topic /path".format(len(msg.poses)))
192.
193.     def __init__(self):
194.         rospy.init_node('Navigation_node', anonymous=False, log_level=rospy.INFO)
                                            # log_level=rospy.(DEBUG,INFO,WARN,ERR,FATAL)
195.         rospy.on_shutdown(self.stop)
196.         rospy.logwarn("AVVIO DEL NODO DI NAVIGAZIONE")
197.         rospy.logwarn("Premere CTRL + C per interrompere")
198.
199.         # Parametri di navigazione
200.         self.neighMode = 1              # Celle confinanti per la navigazione (0,1)
```

```
201.        self.curveRes = 0.01           # Numero di segmenti (curveRes^-1), default 0.01
202.        self.curveSmoothness = 2       # default 2
203.
204.        self.navMode = 1               # Modalità di lettura della mappa (0,1,2)
205.        self.sigma = 2                 # Deviazione standard per la sfocatura Gaussiana (se 0 allora
            non sfoca)
206.        self.dropIndex = 10            # Salta i primi n waypoint
207.
208.        # Parametri controllori di velocità
209.        self.enable = True             # Abilita il movimento
210.
211.        self.pid_ang = PID(0.75,0.1,0) # (P,I,D)
212.        self.pid_lin = PID(1,0.01,0)
213.
214.        self.toll_lin = 0.15           # Errore di 15 cm
215.        self.toll_ang = 0.1            # Errore di 0.1 rad (5.74°)
216.        self.angTreshold = pi/6        # Soglia di applicazione della velocità lineare
217.
218.        self.vbound_lin = 0.5         # Velocità massime
219.        self.vbound_ang = 0.4
220.
221.        #Inizializzazione variabili
222.        self.map = None
223.        self.mapSize = None
224.        self.mapOrigin = (0,0)
225.        self.mapResolution = 0.05
226.        self.wp = None
227.        self.vel = Twist()
228.        self.rate = rospy.Rate(100)
229.        self.pos_attuale = (0,0)
230.        self.yaw_attuale = 0.0
231.        self.err_pos = 0
232.        self.err_ang = 0
233.        self.counter = 0
234.        self.trajectory = list()
            # La traiettoria percorsa dal rover è immagazzinata qui, ad uso futuro
235.
236.        self.logRoundingPrecision = 3   # 3 cifre decimali
237.
238.        # Inizializzazione Subscribers
239.        self.odom_sub = rospy.Subscriber("rtabmap/odom", Odometry, self.odom_callback)
240.        self.wp_sub = rospy.Subscriber("move_base_simple/goal", PoseStamped, self.wp_callback)
241.        self.map_sub = rospy.Subscriber("rtabmap/grid_map", OccupancyGrid, self.map_callback)
242.
243.        # Inizializzazione Publishers
244.        self.vel_pub = rospy.Publisher('cmd_vel', Twist, queue_size=10)
245.        self.waypoint_pub = rospy.Publisher('path', Path, queue_size=10)
246.        rospy.sleep(0.5)
247.
248.        # «------------------------------------ Navigazione ------------------------------------»
249.        rospy.loginfo("Acquisizione mappa...")
250.        rospy.wait_for_message('rtabmap/octomap_grid', OccupancyGrid)
251.        if self.mapSize != []:
252.              rospy.loginfo("Ricevuta mappa di dimensioni: width " + str(self.mapSize[0]) + ", heigth
                  " + str(self.mapSize[1]))
253.        else:
254.             rospy.logerr("Nessuna mappa trovata")
255.        rospy.loginfo("In attesa di pubblicazione di 2D NavGoal su RVIZ")
256.        rospy.wait_for_message('move_base_simple/goal', PoseStamped)
             # Ricezione coordinate bersaglio
257.
258.        rospy.loginfo("Posizione attuale: " + str(round(self.pos_attuale[0],
            self.logRoundingPrecision))
259.        + "; " + str(round(self.pos_attuale[1],self.logRoundingPrecision)))
260.
261.        if self.mapSize != [] and self.wp != []:
262.              self.pathfinder = getTrajectory(self.curveRes, self.curveSmoothness, self.mapSize,
                  self.mapResolution, self.neighMode)
263.              self.pathfinder.calcPath(self.map, self.wp, self.pos_attuale)
                        # Dati mappa e waypoint, usa A* per trovare il percorso
264.
265.            if self.pathfinder.out != []:
266.                 self.publish_waypoints()
267.                 rospy.logwarn("AVVIO MISSIONE")
```

```
268.                 rospy.sleep(1)
269.                 d = np.shape(self.pathfinder.out)[1] - 1
270.
271.                 for j in range(self.dropIndex, d):
272.                    point = (((self.pathfinder.out[0][j] * self.mapResolution)),
                       ((self.pathfinder.out[1][j] * self.mapResolution)))
273.
274.                     while not rospy.is_shutdown() and self.enable:
275.                         self.updateSpeed(point)          # Determinazione dei comandi di velocità
276.                         self.vel_pub.publish(self.vel)
277.
278.                         if abs(self.err_pos) < self.toll_lin:
279.                                rospy.logwarn("Waypoint " + str(j + self.dropIndex) + "/" + str(d) + "
                               raggiunto")
280.                             break
281.
282.                 self.stop()
283.                 rospy.logwarn("MISSIONE CONCLUSA")
284.             else:
285.                 rospy.logfatal("ERRORE - PERCORSO NON VALIDO")
286.         else:
287.              rospy.logfatal("ERRORE - MAPPA O BERSAGLIO NON VALIDO")
288.
289.     def updateSpeed(self, prox_wp):
290.
291.         # Calcolo errore angolare
292.         ang_target = atan2(prox_wp[1] - self.pos_attuale[1], prox_wp[0] - self.pos_attuale[0])
293.         self.err_ang = self.yaw_attuale - ang_target
294.         v_ang = self.pid_ang.update(-self.err_ang)
295.
296.         # Calcolo errore di posizione
297.         dir = np.sign(prox_wp[0] - self.pos_attuale[0]) # +- 1
298.         self.err_pos = np.sqrt((prox_wp[0] - self.pos_attuale[0]) ** 2 + (prox_wp[1] -
                self.pos_attuale[1]) ** 2) * dir
299.         v_lin = self.pid_lin.update(self.err_pos)
300.
301.         # Si limita l'output entro le capacità del rover
302.         v_ang = max(min(self.vbound_ang, v_ang), -self.vbound_ang)
303.         v_lin = max(min(self.vbound_lin, v_lin), -self.vbound_lin)
304.
305.         # Movimento verso il waypoint
306.         self.vel.angular.z = v_ang
307.         self.vel.linear.x = v_lin
308.
309.           if self.vel_pub.get_num_connections() == 0:
310.               rospy.logerr("Nessuna connessione al topic /cmd_vel")
311.               self.stop()
312.               rospy.sleep(0.2)
313.
314.     def stop(self):
315.         self.vel.linear.x = 0
316.         self.vel.angular.z = 0
317.         self.vel_pub.publish(self.vel)
318.         rospy.sleep(1)
319.
320. if __name__ == '__main__':
321.     try:
322.         navigation()
323.     except rospy.ROSInterruptException:
324.         rospy.loginfo("Processo interrotto dall'utente.")
```

Appendice 2: Procedura di installazione del sistema ROS

[Nota: al momento il package robot_localization è supportato solo per sistemi con architettura AMD64, mentre sistemi come Raspberry Pi sono basati su ARM64. Può essere necessario ricompilare il package e le relative dipendenze tramite la modalità build from source (solo per utenti esperti).]

Installazione sul pc ground station/di bordo:

1. Installare Ubuntu 20.04 LTS in una partizione dedicata
   1.1. https://ubuntu.com/tutorials/install-ubuntu-desktop#1-overview
   1.2. https://ramith.fyi/setting-up-raspberry-pi-4-with-ubuntu-20-04-ros-intel-realsense/
2. Installare Ros Noetic (desktop o desktop-full)
   2.1. https://wiki.ros.org/noetic/Installation/Ubuntu
3. Installare la libreria LibRealSense (Assicurarsi di essere nel branch *ros1-legacy*)
   3.1. https://github.com/IntelRealSense/realsense-ros/tree/ros1-legacy
4. Installare il package SLAM
   4.1. https://github.com/IntelRealSense/realsense-ros/wiki/SLAM-with-D435i
5. Generare l'ambiente catkin su cui eseguire i packages di ROS.
   5.1. https://wiki.ros.org/catkin/Tutorials/create_a_workspace
   5.2. https://wiki.ros.org/ROS/Tutorials/CreatingPackage

Avvio del sistema:

6. Collegarsi alla rete wireless del rover
   6.1. https://docs.fictionlab.pl/docs/category/guides
   6.2. https://docs.fictionlab.pl/docs/leo-rover/guides/connect-to-rover-ap
   6.3. https://docs.fictionlab.pl/docs/leo-rover/guides/connect-to-network
7. In alternativa, collegarsi tramite cavo ethernet
8. Ricavare l'indirizzo IP del computer tramite il comando *ip address*

Ripetere i seguenti passaggi per ogni istanza della shell di sistema

9. *source /opt/ros/noetic/setup.bash*
10. *export ROS_MASTER_URI=http://master.localnet:11311*
11. *export ROS_IP=XX.X.X.XXX* (inserire IP ricavato al punto 8, diverso per ogni pc)

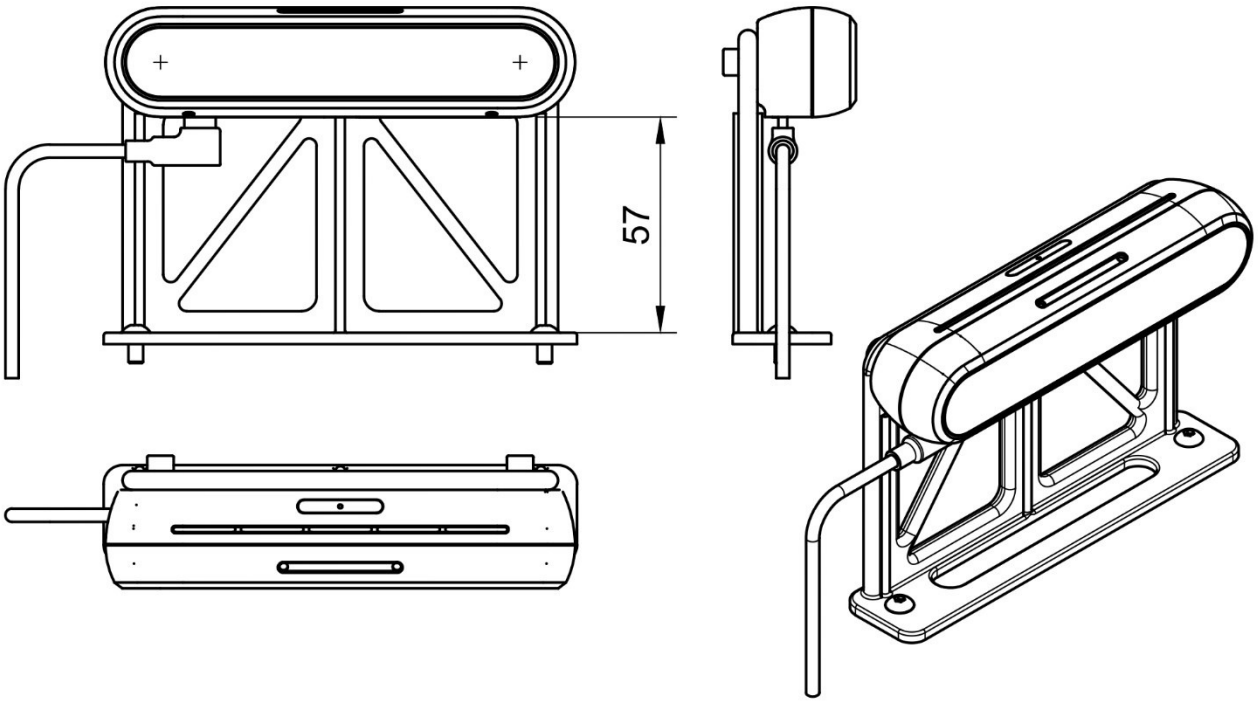
(A bordo del rover) In una schermata eseguire il nodo di mappatura

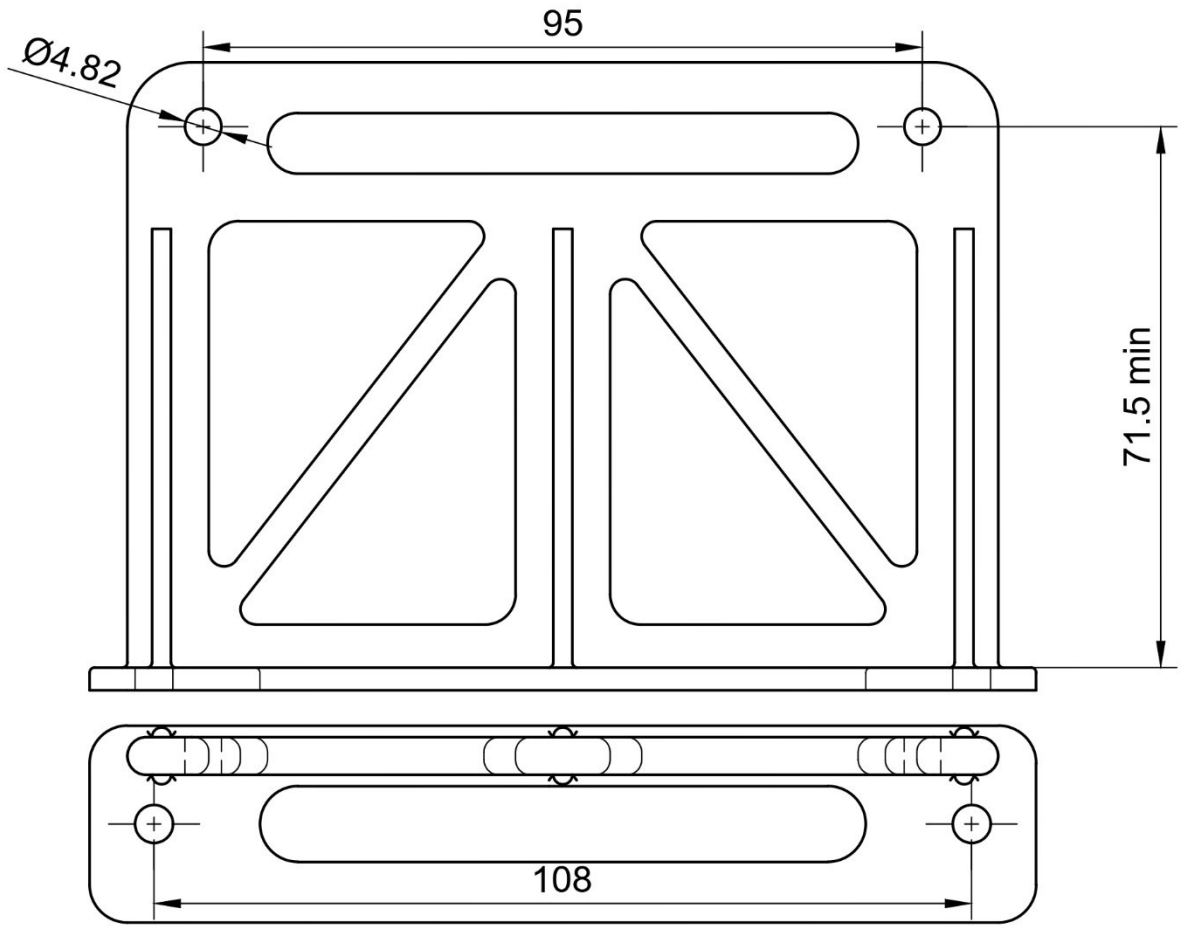
12. *roslaunch realsense2_camera opensource_tracking.launch*

(A bordo del rover) In una seconda schermata eseguire il nodo di navigazione

13. *source ~/catkin_ws/devel/setup.bash*
14. *rosrun navigation_stack NavigationNode.py*

In questo caso il software è contenuto all'interno del package *navigation_stack* nel file *NavigationNode.py*, ma può variare a seconda dell'installazione. Aprire l'interfaccia RVIZ sul pc ground station per visualizzare i dati e controllare il sistema da remoto.





Dept.	Technical reference	Created by	Approved by
		Andrea Dal Pozzo	
Tutti i fori ospitano viti M4		Document type	Document status: Final
		Title: Supporto RealSense	DWG No.
			Rev. 2 / Date of issue 05/04/2024 / Sheet 1/1