

Documentazione - Progetto di Robotica

Augello Andrea Castiglione Francesco Paolo La Martina Marco

Università degli studi di Palermo

9 gennaio 2021

1 Introduzione

2 Gestione dei nodi ROS

3 TIAGo Iron

4 Modello del moto e posizionamento

5 Object recognition

6 Posizione dei target

7 Pianificazione del moto

8 Possibili modifiche

Section 1

Introduzione

La pandemia del coronavirus SARS-CoV-2 ha dato una forte spinta alla ricerca sia nel campo sanitario che informatico, mettendo in evidenza forti carenze dal punto di vista infrastrutturale.

Al momento, considerando la limitata disponibilità del vaccino alle masse, uno dei migliori modi di evitare la contrazione del coronavirus è di evitarne l'esposizione. Il distanziamento sociale si configura di conseguenza come un prerequisito per una significativa riduzione del numero di infetti, come evidenziato da simulazioni di un sistema ad agenti [1]. Un problema chiave si configura di conseguenza come il controllo del rispetto delle norme di distanziamento all'interno degli spazi chiusi.

Obbiettivo

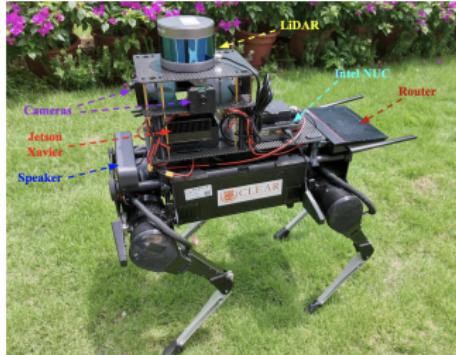
L'obbiettivo del progetto è quello di sviluppare un robot con lo scopo di **evitare assembramenti in ambienti indoor** e di invitare a **rispettare le norme sul distanziamento sociale**.

Nella dimostrazione presentata il nostro robot rileva le persone nella stanza e individua i possibili assembramenti. In seguito alla fase di rilevazione si sposterà verso l'assembramento evitando gli ostacoli e, arrivato, esorterà le persone al rispetto del distanziamento sociale.

Stato dell'arte

Un robot che si occupa di far rispettare il distanziamento sociale è quello proposto nell'articolo [2]. Sono stati usati un TurtleBot 2, una camera RGB-D e CCTV per il rilevamento degli assembramenti, una camera termica per rilevare la temperatura corporea e un lidar 2-D per evitare le collisioni.

Un altro esempio è quello proposto nell'articolo [3]. In questo caso sono state usate 2 camere e CCTV per il rilevamento degli assembramenti e un lidar 3-D per evitare le collisioni.



Setup

OS	Ubuntu 18.04 Ubuntu 20.04
ROS version	melodic noetic
Webots	R2020b revision 1
OpenCV [4]	4.x
Imutils [5]	0.5.3
Matplotlib [6]	3.3.3
Numpy [7]	1.17.2
Scikit-learn [8]	0.21.3
Target hardware	Raspberry Pi 3B+

1 Introduzione

2 Gestione dei nodi ROS

3 TIAGo Iron

4 Modello del moto e posizionamento

5 Object recognition

6 Posizione dei target

7 Pianificazione del moto

8 Possibili modifiche

Section 2

Gestione dei nodi ROS

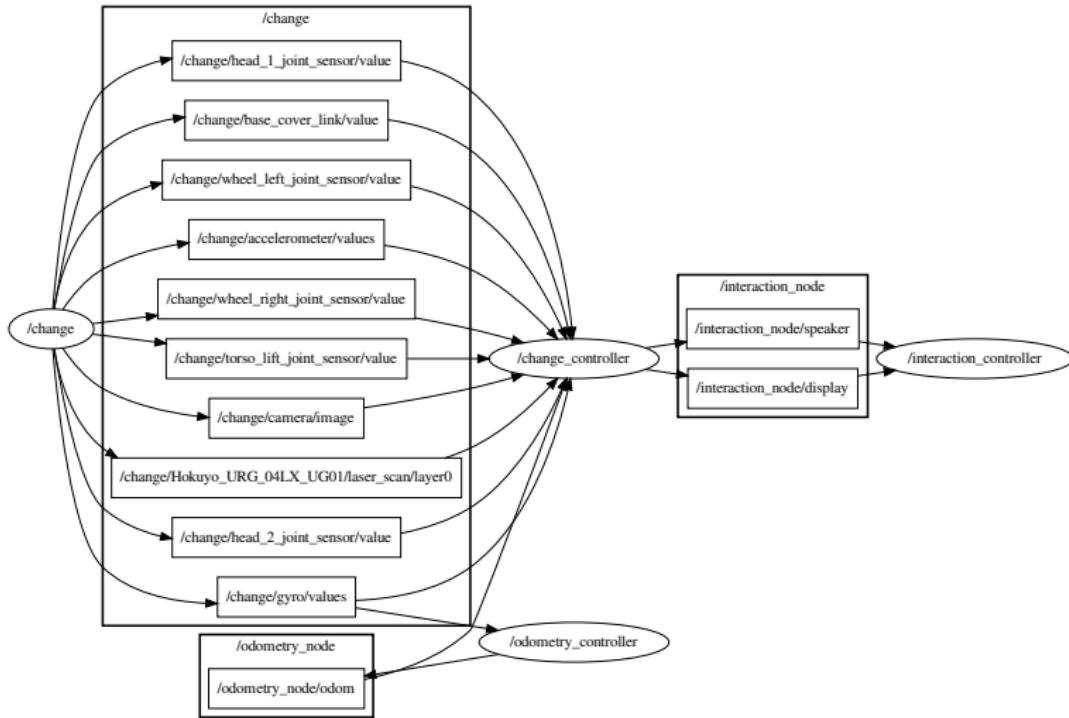


Figura: Architettura dei nodi ros, ottenuta tramite *rqt*

Webots node

Questo nodo si occupa solamente di lanciare Webots, e di impostare il valore del clock di ROS in base al tempo della simulazione, in modo da potere effettuare le integrazioni del tempo correttamente.

Change controller node

Questo è il nodo che si occupa della gran parte della elaborazione. Oltre ad arbitrare sui comportamenti da assumere, gestisce più moduli che si occupano di:

- acquisire i dati dai sensori
- mandare i comandi ai motori
- gestire il movimento, quindi rotazioni e traslazioni
- acquisire e analizzare le immagini dalla camera

Interaction node

Questo nodo ha il compito di gestire le interazioni audio/video. Ogni messaggio che viene riprodotto, prima in lingua italiana e poi inglese, viene anche visualizzato testualmente sullo schermo in italiano, inglese e cinese. I possibili comportamenti assunti dal robot sono:

- Salutare all'avvio
- Mostrare sul display immagini che esortano a rispettare il distanziamento sociale
- Riprodurre un messaggio audio che invita a rispettare il distanziamento sociale quando rileva un assembramento o quando scansiona l'ambiente

Odometry node

Il nodo che si occupa dell'odometria si occupa di stimare la posizione del robot, come spiegato approfonditamente nella sezione 4. In generale ciò che fa è integrare costantemente i valori del giroscopio e della velocità delle ruote per condividere posizione e orientamento del robot.

1 Introduzione

2 Gestione dei nodi ROS

3 TIAGo Iron

4 Modello del moto e posizionamento

5 Object recognition

6 Posizione dei target

7 Pianificazione del moto

8 Possibili modifiche

Section 3

TIAGo Iron

Il robot scelto per l'obiettivo proposto è il **TIAGo Iron**, un robot umanoide a due ruote con torso e testa ma senza braccia articolate [9].

Il datasheet del **TIAGo** [10] indica la presenza di speaker e display, non presenti nel modello Webots [11], che sono quindi stati aggiunti.

La camera del **TIAGo** è RGB-D ma il modello Webots ne è sprovvisto, di conseguenza è stata utilizzata una camera monoscopica RGB.

L'IMU utilizzata ha 6 gradi di libertà.

Il modello Webots del **TIAGo** presenta un lidar (Hokuyo URG-04LX-UG01 [12]) che, conformemente al datasheet, ha un range di 5.6 m ed un FOV di 240° (agli estremi è parzialmente occluso).



- 1 Introduzione
- 2 Gestione dei nodi ROS
- 3 TIAGo Iron
- 4 Modello del moto e posizionamento
- 5 Object recognition
- 6 Posizione dei target
- 7 Pianificazione del moto
- 8 Possibili modifiche

Section 4

Modello del moto e posizionamento

Orientamento

Il modello del moto è caratterizzato da rotazioni e traslazioni. Per le rotazioni ci basiamo sui dati forniti dal giroscopio, il quale fornisce una velocità angolare. Calcoliamo quindi l'angolo di rotazione effettuando un'integrazione discreta dei campioni con interpolazione lineare del primo ordine (Eq. 1).

$$\theta_i = \sum_{j=1}^i \frac{\omega_{j-1} + \omega_j}{2} (t_j - t_{j-1}) \quad (1)$$

Noto l'angolo corrente e l'angolo target utilizziamo un controllore proporzionale per raggiungere l'angolo desiderato.

Spostamento

Per effettuare lo spostamento lineare utilizziamo il controllore PID (Proporzionale-Integrale-Derivativo) delle ruote fornito da Webots, che richiede un angolo di rotazione target per ogni ruota. Utilizziamo quindi l'angolo di rotazione corrente, e il diametro delle ruote per calcolare la posizione delle ruote necessaria al fine di ottenere lo spostamento desiderato (Eq. 2).

$$\text{targetAngle} = \text{currentAngle} + 2\pi \frac{\text{distance}}{2\pi \cdot \text{diameter}} \quad (2)$$

Posizionamento I

Per stimare il posizionamento, inizialmente abbiamo utilizzato i dati dell'accelerometro [14], in quanto le ruote possono essere soggette a slittamento fornendo una misura imprecisa. Al segnale dell'accelerometro veniva applicato un integrale doppio per ottenere lo spostamento lineare(Eq. 3). Questa integrazione veniva effettuata solamente nel momento in cui al robot veniva dato il comando di muoversi.

$$\begin{cases} \mathbf{v}_i &= \sum_{j=1}^i \frac{\mathbf{a}_{j-1} + \mathbf{a}_j}{2} (t_j - t_{j-1}) \\ \mathbf{s}_i &= \sum_{j=1}^i \frac{\mathbf{v}_{j-1} + \mathbf{v}_j}{2} (t_j - t_{j-1}) \end{cases} \quad (3)$$

In un secondo momento abbiamo ritenuto opportuno utilizzare un nodo ROS che si occupasse di condividere costantemente la posizione e l'orientamento del robot, anche per predisporre il sistema per un eventuale algoritmo di SLAM. Ciò implica una integrazione costante dei valori del giroscopio e dell'accelerometro.

Il giroscopio non ha causato nessun problema a differenza dell'accelerometro che invece, essendo soggetto a rumore, rilevava accelerazioni diverse da 0 anche da fermo. Queste accelerazioni, essendo

Posizionamento II

integrate costantemente, producevano degli spostamenti significativi anche da fermi.

Abbiamo ritenuto necessario cambiare approccio, decidendo di utilizzare gli encoders delle ruote per determinare gli spostamenti.

In particolare abbiamo integrato la velocità lineare del robot, calcolata a partire dal raggio R e le velocità angolari $u[t]$ delle ruote, fornite da un servizio ROS, e aggiornando così la posizione. Avremmo anche potuto utilizzare queste informazioni per ricavarci l'orientamento del robot, ma abbiamo ritenuto più opportuno utilizzare le informazioni forniteci dal giroscopio. La posizione del robot è data da:

$$\mathbf{P}_i = [x_i \quad y_i]^T \quad (4)$$

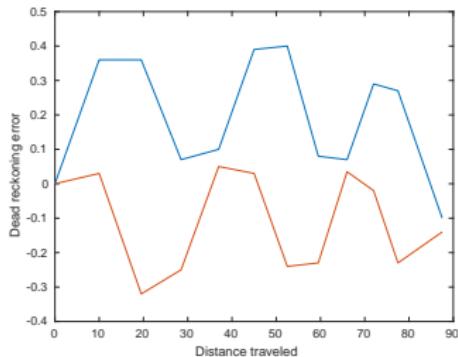
$$v_i = \frac{R(u_{r,i} + u_{l,i})}{2} \quad (5)$$

Questi valori sono aggiornati ad ogni intervallo di campionamento utilizzando la velocità lineare e la velocità angolare del robot [15].

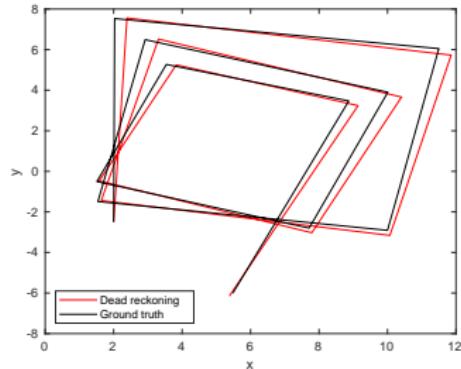
Posizionamento III

$$\mathbf{P}_i = \sum_{j=1}^i \begin{bmatrix} v_i \cos(\theta_i) \\ v_i \sin(\theta_i) \end{bmatrix} \cdot (t_j - t_{j-1}) \quad (6)$$

La $\omega[t]$ è data dall'interpolazione lineare descritta nell'equazione (1).



(a) Errore nella stima della posizione



(b) Errore nella stima della traiettoria

Abbiamo misurato le performance della stima di posizione e i risultati sono ritenuti soddisfacenti per raggiungere l'obiettivo proposto. In Fig. 3a sono mostrati i risultati delle misurazioni effettuate. In Fig. 3b viene mostrata chiaramente la differenza fra la traiettoria reale del **TIAGO** e la traiettoria calcolata con l'odometria.

Collision avoidance

Il **TIAGo** è in grado di rilevare gli ostacoli grazie all'utilizzo di un sensore lidar. Nel nostro caso d'uso, la rilevazione degli ostacoli è infatti imprescindibile per raggiungere l'obiettivo proposto in maniera soddisfacente.

Nell'immagine seguente viene mostrata la zona nella quale, se viene indicata dal lidar la presenza di un ostacolo, il **TIAGo** si ferma per ragioni di sicurezza al fine di evitare danni a persone e/o oggetti.

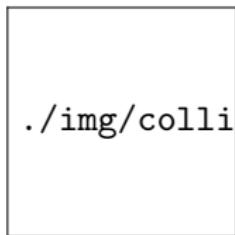


Figura: Collision detection

- 1** Introduzione
- 2** Gestione dei nodi ROS
- 3** TIAGo Iron
- 4** Modello del moto e posizionamento
- 5** Object recognition
- 6** Posizione dei target
- 7** Pianificazione del moto
- 8** Possibili modifiche

Section 5

Object recognition

Campionamento delle immagini

Il FOV della camera è di 57° , quindi per ricoprire 360° è necessario effettuare 7 campionamenti. Il settimo campionamento, come si vede in figura 5, è sovrapposto al primo per 39° .

È possibile che un individuo si trovi in una zona di confine tra due campioni, e che quindi non sia correttamente identificabile in nessuna delle due immagini in cui appare parzialmente. Mitighiamo questo problema effettuando una rudimentale operazione di image mosaicing [16] e campioniamo l'immagine così ottenuta ad intervalli di 28° .

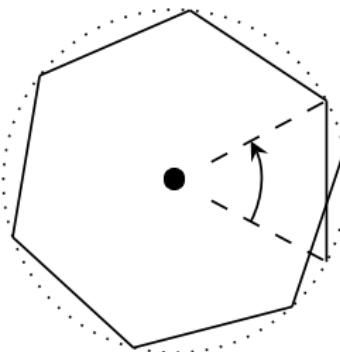


Figura: Campionamento delle immagini

Yolo I

Al fine di riconoscere le persone è stato necessario utilizzare sistemi di **object recognition**. A tal fine abbiamo valutato le performance di YOLOv3 (you only look once), YOLOv3-tiny, HoG (Histogram of oriented gradients), HoG + SVM (support vector machines) + NMS (non maxima suppression). In seguito a vari test su HoG abbiamo ritenuto essere problematica la larghezza delle bounding boxes fornite, in quanto, per motivazioni che verranno chiarite nel paragrafo successivo, vogliamo che queste ultime siano il più possibili vicine alla reale larghezza delle persone. YOLOv3, nonostante sia stato addestrato su foto di persone reali (e non modelli 3D) fornisce risultati soddisfacenti, in seguito al fine-tuning degli iperparametri della rete. Tuttavia, considerando le caratteristiche hardware del robot mobile, abbiamo optato per l'uso di YOLOv3-tiny, il quale risulta essere significativamente più efficiente (approssimativamente del 442% [17]), sacrificando in termini di precisione ma comunque sufficientemente preciso per il nostro obiettivo. La tabella 3 [18], mostra l'architettura di YOLOv3-tiny. La tabella 1 mostra un paragone fra le versioni di YOLO e le rispettive versioni tiny [17]. Inoltre è rilevante in tal senso un paragone fra YOLOv3 e YOLOv3-tiny in

Yolo II

termini di mAP (mean average precision) e FLOPS (floating-point operations per second) addestrate sul dataset COCO, come illustrato dalla tabella 2, i cui dati provengono dal sito di YOLO [19]:

Model	Number of n-layers	FLOPS	FPS	map value	Dataset
YOLOv1	26.00	Not given	45.00	63.50	VOC-data
YOLOv1-tiny	9.00	Not given	155.00	52.80	VOC-data
YOLOv2	32.00	62.95	40.00	48.20	COCO-data
YOLOv2-tiny	16.00	05.42	244.00	23.60	COCO-data
YOLOv3	106.00	140.70	20.00	57.80	COCO-data
YOLOv3-tiny	24.00	05.57	220.00	33-20	COCO-data

Tabella: Paragone fra le versioni di YOLO con rispettive versioni tiny

Yolo III

Model	mAP	FLOPS	FPS
YOLOv3-320	51.5	38.97 Bn	45
YOLOv3-416	55.3	65.86 Bn	35
YOLOv3-608	57.9	140.69 Bn	20
YOLOv3-tiny	33.1	5.56 Bn	220
YOLOv3-spp	60.6	141.45 Bn	20

Tabella: Paragone fra le versioni di YOLOv3

Yolo IV

Layer	Type	Filters	Size/Stride	Input	Output
0	Convolutional	16	3×3/1	416×416×3	416×416×16
1	Maxpool		2×2/2	416×416×16	208×208×16
2	Convolutional	32	3×3/1	208×208×16	208×208×32
3	Maxpool		2×2/2	28×208×32	104×104×32
4	Convolutional	64	3×3/1	104×104×32	104×104×64
5	Maxpool		2×2/2	104×104×64	52×52×64
6	Convolutional	128	3×3/1	52×52×64	52×52×128
7	Maxpool		2×2/2	52×52×128	26×26×128
8	Convolutional	256	3×3/1	26×26×128	26×26×256
9	Maxpool		2×2/2	26×26×256	13×13×256
10	Convolutional	512	3×3/1	13×13×256	13×13×512
11	Maxpool		2×2/1	13×13×512	13×13×512
12	Convolutional	1024	3×3/1	13×13×512	13×13×1024
13	Convolutional	256	1×1/1	13×13×1024	13×13×256
14	Convolutional	512	3×3/1	13×13×256	13×13×512
15	Convolutional	255	1×1/1	13×13×512	13×13×255
16	YOLO				
17	Route 13				
18	Convolutional	128	1×1/1	13×13×256	13×13×128
19	Up-sampling		2×2/1	13×13×128	26×26×128
20	Route 19 8				
21	Convolutional	256	3×3/1	13×13×384	13×13×256
22	Convolutional	255	1×1/1	13×13×256	13×13×256
23	YOLO				

Tabella: Architettura di YOLOv3-tiny

- 1** Introduzione
- 2** Gestione dei nodi ROS
- 3** TIAGo Iron
- 4** Modello del moto e posizionamento
- 5** Object recognition
- 6** Posizione dei target
- 7** Pianificazione del moto
- 8** Possibili modifiche

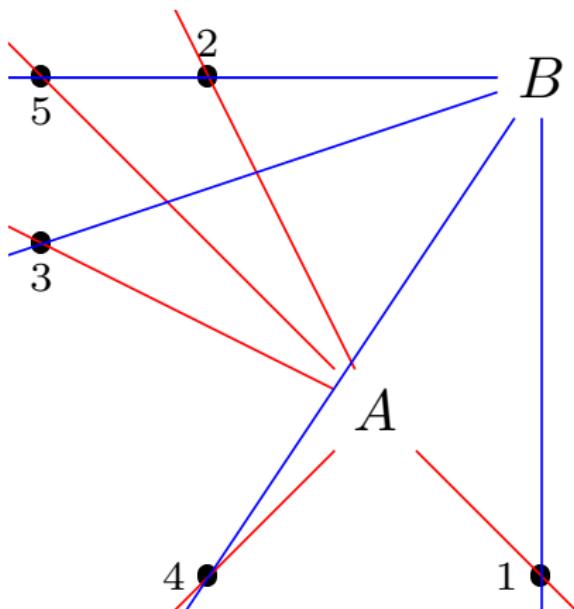
Section 6

Posizione dei target

Triangolazione

La triangolazione come metodo di individuazione delle persone presenta dei problemi nel nostro scenario:

- Occlusione delle persone: se due persone (5 e 2) sono una dietro l'altra lungo una retta immaginaria che le congiunge al robot (B), quest'ultimo non sarà in grado di individuare la più distante.
- Imputazione delle osservazioni: quando il robot effettua scan successivi non è in grado di dedurre quali osservazioni derivano dalla stessa persona. Non è possibile determinare quali intersezioni corrispondono ad osservazioni reali e quali sono spurie.



Calcolo della distanza I

L'altezza percepita dell'oggetto non è un indicatore affidabile della sua distanza dal robot in quanto parte dell'oggetto potrebbe essere occlusa o non presente nel frame. Inoltre classificatori quali HoG tendono a produrre ROI significativamente più alte dell'oggetto. La larghezza del torso, invece, è meno suscettibile a tali problemi, e non dipende dalla posizione (ad esempio seduto o alzato). Dobbiamo tuttavia ipotizzare che il torso abbia forma cilindrica introducendo quindi degli errori se l'obiettivo non sta guardando la camera. La posizione orizzontale dell'oggetto relativa alla camera può influenzare la larghezza percepita diminuendola quando aumenta la distanza dal centro dell'immagine per via delle distorsioni introdotte dall'ottica della camera. Ipotizzando che la camera abbia un FOV (field of view) di 2α e sia distante d dall'oggetto, la massima distanza orizzontale che un punto dell'immagine potrebbe avere dal centro del piano dell'immagine sarebbe $a = d \tan \alpha$ (Fig. 6).

Calcolo della distanza II

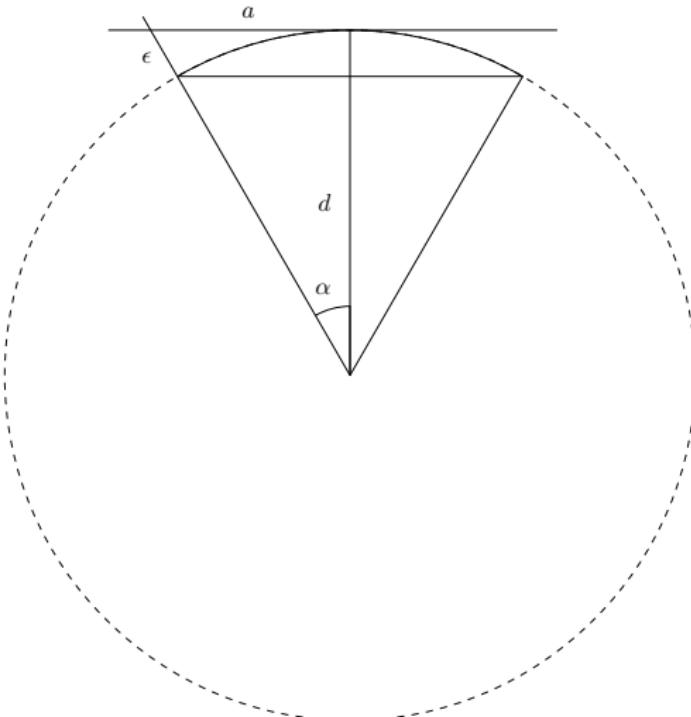


Figura: Errore nella stima della distanza con una linearizzazione a tratti della circonferenza

Calcolo della distanza III

Ignorare la prospettiva significa effettuare un'approssimazione lineare del primo ordine e trattare il punto come se si trovasse su una circonferenza di raggio d centrata sulla camera. Di conseguenza consideriamo il punto come se fosse più vicino di quanto non sia realmente, commettendo l'errore mostrato nell' Eq. 7. Con una camera con FOV di 1 radiante quale quella del TIAGo il massimo errore causato dalla linearizzazione è quindi una sottostima del 13.9%.

$$\epsilon = \sqrt{a^2 + d^2} - d = \sqrt{(d \tan \theta)^2 + d^2} - d = d \left(\sqrt{\frac{1}{\cos^2 \alpha}} - 1 \right) = d (\sec \alpha - 1) \quad (7)$$

Sotto tali ipotesi possiamo quindi calcolare la distanza di un oggetto come mostrato in Eq. 8.

$$object\ distance(m) = \frac{f(m) \times real\ width(m) \times image\ width(pixels)}{object\ width(pixels) \times sensor\ width(m)} \quad (8)$$

Calcolo della distanza IV

Poiché stiamo utilizzando un simulatore non è nota la larghezza del sensore da utilizzare per l'eq. 8. Abbiamo ovviato a tale problema posizionando il robot ed un oggetto dalle dimensioni note in posizioni note e abbiamo utilizzato questi dati insieme a delle misure in pixel nell' eq. 9. Abbiamo così stimato le dimensioni del sensore virtuale da utilizzare nei calcoli successivi.

$$\text{sensor width}(m) = \frac{f(m) \times \text{real width}(m) \times \text{image width}(pixels)}{\text{object width}(pixels) \times \text{object distance}(m)} \quad (9)$$

Scarto dei duplicati I

Durante la fase di individuazione delle persone vengono individuate varie bounding box corrispondenti al medesimo individuo. Di conseguenza è stato necessario effettuare una fase di clustering al fine di scartare le bounding box duplicate. L'algoritmo di clustering utilizzato è DBSCAN (Density based scan) [20], i cui parametri principali sono **eps**, ovvero la massima distanza fra due punti affinché vengano considerati appartenenti a un cluster (da non confondere con la massima distanza fra i punti di un cluster), **min_samples**, ovvero il numero minimo di punti affinché un cluster sia valido (nel nostro caso è uguale a 1 in quanto non vogliamo scartare ROI) ed infine la metrica di distanza. Come di evince dalla figura 7, la metrica utilizzata considera la lunghezza di un arco di circonferenza con raggio corrispondente alla distanza fra i due punti ed angolo α corrispondente all'angolo fra i due punti rispetto alla posizione del robot.

Scarto dei duplicati II

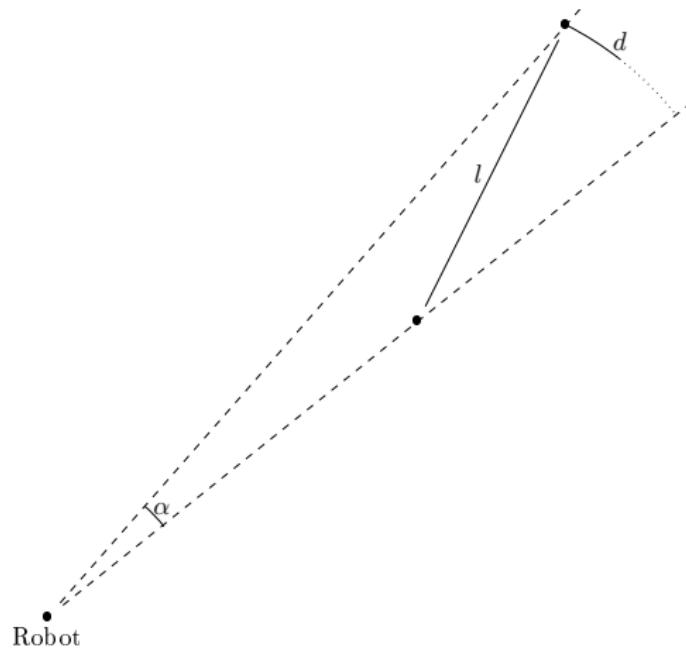
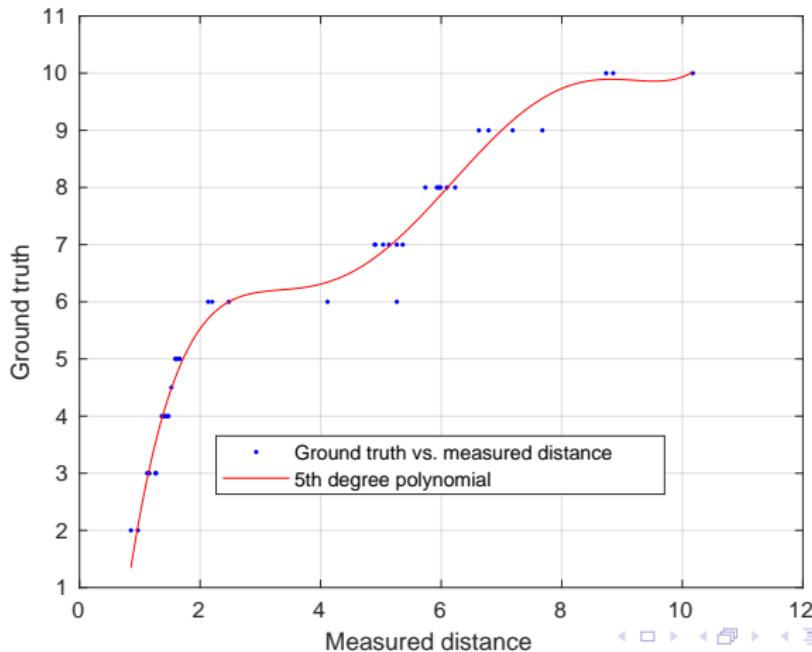


Figura: Non maxima suppression

Correzione dei valori

Per migliorare la stima sulla distanza abbiamo paragonato le reali distanze dei target con le stime effettuate dal sistema ottenendo il polinomio interpolante

$0.003116 * x^5 - 0.09722 * x^4 + 1.124 * x^3 - 5.908 * x^2 + 14.5 * x - 7.367$, che approssima la funzione di correzione della stima.



Le bontà della stima della distanza e dell'angolo si evince dalle Fig. 8

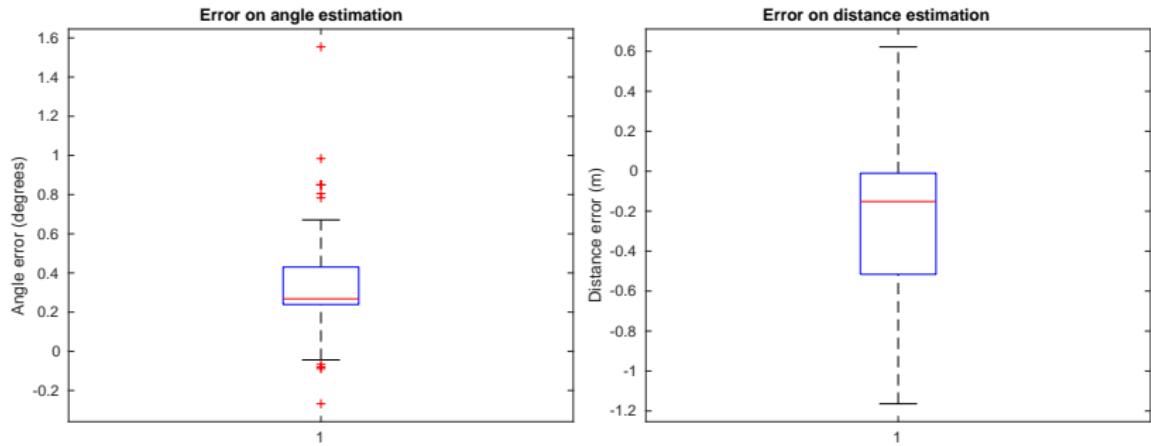


Figura: Box plot relativi all'errore su angolo e distanza

Matrice di covarianza degli errori su angolo e distanza:

$$V = \begin{bmatrix} 0.163 & -0.0397 \\ -0.0397 & 0.0752 \end{bmatrix} \quad (10)$$

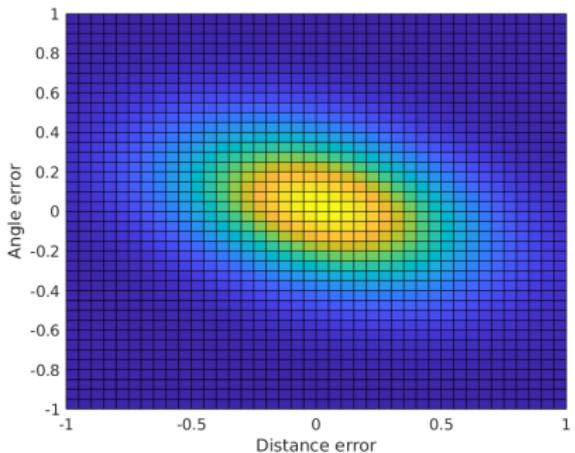
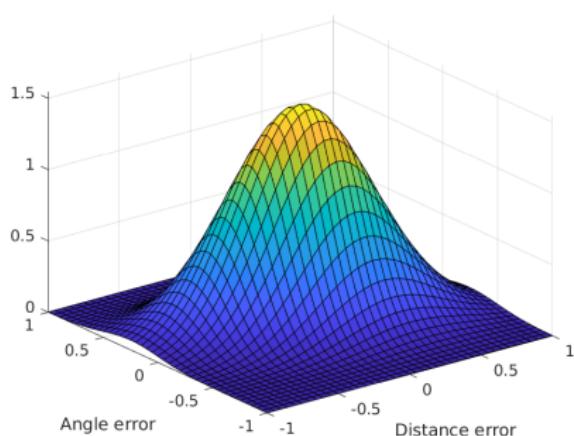


Figura: Distribuzione gaussiana multivariata con covarianza data nell'equazione 10

Le relazioni geometriche sulla distanza degli oggetti esposte nella sezione 2 assumono che il target abbia la stessa larghezza in ogni posa e che sia possibile ottenere dal classificatore delle bounding box estremamente aderenti all'oggetto. Dato che queste assunzioni non sono rispettate nella nostra applicazione la distanza dell'oggetto sarà soggetta ad errore non trascurabile. Per questa ragione e per i problemi legati alla triangolazione abbiamo abbandonato la rappresentazione basata su oggetti e abbiamo fatto ricorso ad un filtro di occupazione bayesiano (BOF) [21].

Nelle classiche metodologie di tracciamento, il problema dell'associazione dei dati (data association), ovvero di associare un oggetto o_t al tempo t con un oggetto o_{t+1} al tempo $t+1$, e della stima dello stato si configurano come classici problemi da risolvere. Nel filtro di occupazione bayesiano, il problema dell'associazione dei dati viene superato in quanto viene gestito da un livello di astrazione superiore. Il concetto di oggetto viene difatti riformulato da proprietà più utili quali occupazione o rischio, che vengono stimate direttamente per ogni cella utilizzando sia osservazioni dai sensori che conoscenze pregresse. Le caratteristiche di incertezza legate ai sensori vengono descritte, in questo modello, attraverso le probabilità di occupazione.

Al fine di trasformare le osservazioni ottenute in una probabilità che le persone si trovino effettivamente nella posizione indicata è stato

necessario definire una funzione densità di probabilità. La distribuzione normale, sebbene relativamente appropriata nel nostro scenario, è computazionalmente onerosa. La griglia di occupazione ha diverse migliaia di celle per cui va calcolata una pdf per ogni osservazione. Come si evince dalla fig. 10 con un numero così elevato di iterazioni i tempi di esecuzione non sarebbero accettabili, è stato quindi necessario ricorrere ad una approssimazione.

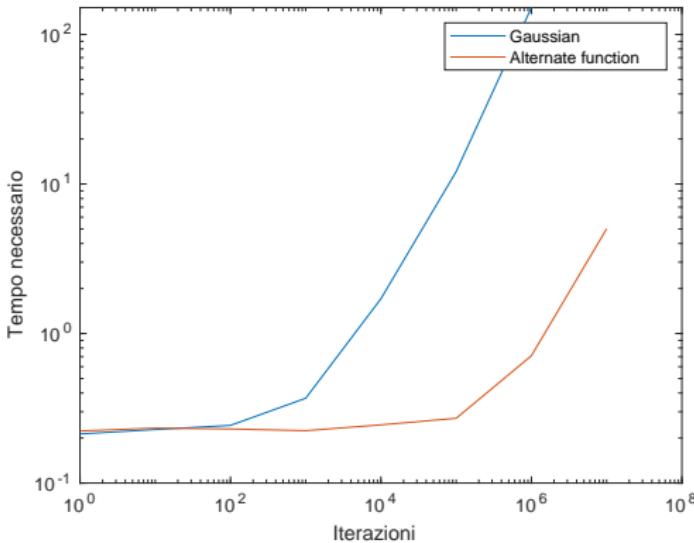


Figura: Benchmark funzione densità di probabilità (probability density function, funzione)

Un'approssimazione di largo uso è la distribuzione triangolare. Quest'ultima, tuttavia, presenta delle caratteristiche non desiderabili per il nostro caso d'uso. Una proprietà desiderabile della nostra funzione, difatti, sono le "fat tails", ovvero le code della distribuzione devono essere spesse e la distribuzione non deve tagliare nettamente. Quando, in seguito ad un'osservazione, non abbiamo alcuna osservazione di conferma nello scan successivo, non è infatti desiderabile che la probabilità di trovare una persona in quel punto scenda a zero. Inoltre, in una prima osservazione, un oggetto potrebbe essere occluso ed essere rilevato solo successivamente, quindi se in quella zona la probabilità di rilevare un target scendesse a zero non sarebbe possibile integrare correttamente questa nuova informazione. Ulteriori complicazioni sorgerebbero in caso di target in movimento.

Essendo la scansione una operazione estremamente costosa non è nemmeno possibile affidarsi esclusivamente all'aggiunta di rumore alla griglia di occupazione confidando in una eventuale convergenza.

Al fine di ottene un'approssimazione di una gaussiana adatta al nostro scenario, per modellare la probabilità che data l'occupazione della cella in posizione x si ottenga l'osservazione z è stato utilizzato un funzionale ispirato al guadagno del filtro di Butterworth.

$$p(z|x) = \frac{K}{1 + d(x, z)^4} \quad (11)$$

Nell'equazione 11 è mostrato il funzionale utilizzato. In figura 11 si mostra un confronto della forma di questa funzione rispetto ad una gaussiana con media nulla e deviazione standard unitaria nel caso monodimensionale.

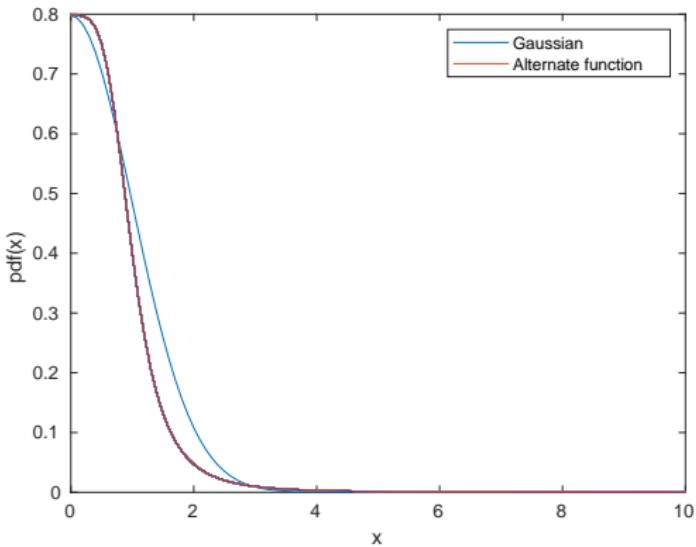


Figura: Forma pdf

Nell'equazione 11 K indica un parametro di scala per ottenere CDF unitaria. d è una funzione per il calcolo della distanza della cella x dalla stima di posizione z .

La distanza euclidea porterebbe a formare aree ad alta probabilità di forma circolare. Questo però non si adatta bene al nostro modello di errore del sensore: l'angolo dell'oggetto rispetto al robot è noto con una precisione molto elevata, mentre la maggior parte dell'incertezza si concentra nella distanza. Calcoliamo quindi la distanza non come distanza euclidea tra le coordinate cartesiane della cella e dell'osservazione, ma come norma L^2 delle coordinate polari, opportunamente normalizzate per tenere conto della diversa incertezza sulla misura di angolo e distanza.

L'incertezza sulla distanza è ricavata dalle misure di calibrazione effettuate. Per l'angolo è stato seguito un approccio differente: utilizzare una varianza calcolata a partire da una serie di osservazioni porterebbe ad assegnare maggiore probabilità ad oggetti lontani. In un setup come quello in figura 12, con due target rilevati, rispettivamente a distanza d_1 e d_2 , le aree ad alta probabilità relative ad entrambi i punti avranno lunghezza l e ampiezze α_1 e α_2 .

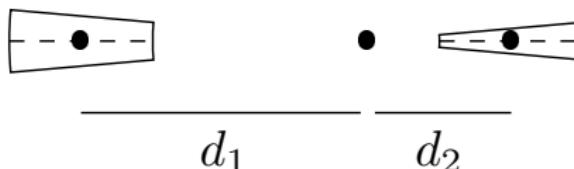


Figura: Area della regione probabile a seconda della distanza.

Come mostrato nell'equazione 12, con queste condizioni la dimensione dell'area ad alta probabilità è direttamente proporzionale alla distanza e all'ampiezza dell'angolo. Se quindi α_1 coincidesse con α_2 ad oggetti distanti verrebbero associate aree molto più grandi.

$$\begin{aligned} A_1 &= \pi((d_1 + l/2)^2 - (d_1 - l/2)^2) \frac{\alpha_1}{2\pi} = \alpha_1 d_1 l \\ A_2 &= \pi((d_2 + l/2)^2 - (d_2 - l/2)^2) \frac{\alpha_2}{2\pi} = \alpha_2 d_2 l \\ \frac{A_1}{A_2} &= \frac{\alpha_1 d_1}{\alpha_2 d_2} \end{aligned} \tag{12}$$

Per evitare questo problema, dopo avere individuato un valore α^* appropriato questo viene scalato per un coefficiente inversamente proporzionale alla distanza della cella, quindi $\alpha_1 = \alpha^*/d_1$ e $\alpha_2 = \alpha^*/d_2$, da cui $A_1 = A_2$.

Questa misura di distanza è in accordo con l'intuizione derivata dal fatto che un oggetto più lontano occuperà una porzione più piccola dell'immagine, di conseguenza ci sono meno posizioni valide che la ROI associata all'oggetto può assumere, e quindi l'intervallo di valori che può assumere l'angolo è ridotto.

Dato un insieme di osservazioni ottenute da una scansione Z_i aggiorniamo il belief precedente rispetto ad ogni cella della griglia di

occupazione come mostrato nell'equazione 13.

$$p(x_i|Z_i) = p(x_{i-1}) \cdot \sum_{z \in Z_i} p(z|x) \quad (13)$$

Al termine di ogni aggiornamento della mappa l'intera griglia viene inoltre normalizzata per avere somma unitaria. Questa formula è valida sotto le seguenti assunzioni:

- In uno stesso scan osservazioni separate si riferiscono ad oggetti distinti. In altre parole date due osservazioni z_1 e z_2 queste non hanno intersezione, quindi

$$p(z_1 \cup z_2) = p(z_1) + p(z_2) - p(z_1 \cap z_2) = p(z_1) + p(z_2)$$

Questa assunzione è ragionevole ricordando che lo scan avviene ruotando intorno al centro del robot e che viene effettuato uno scarto dei duplicati tenendo conto dell'angolo, quindi non possono esserci due osservazioni distinte derivanti dall'occupazione della stessa posizione nella mappa.

- In assenza di nuove osservazioni la stima dello stato del sistema rimane invariata: $\overline{bel}(x_i) = bel(x_{i-1})$

Il robot non si muove abbastanza velocemente da poter inseguire e raggiungere assembramenti di breve durata, ha senso quindi limitarsi a considerare situazioni prevalentemente stazionarie e assumere consistenza dell'ambiente tra uno scan e l'altro. ▶ Inoltre gli scan



sarebbero comunque troppo infrequenti per ottenere stime significative sul movimento degli oggetti.

In pratica applicando direttamente queste formule si introdurrebbe un errore nell'interpretazione delle informazioni: se per qualche ragione in uno scan non venisse rilevato nessun oggetto, Z sarebbe l'insieme vuoto. In mancanza di nuovi dati si potrebbero tentare due approcci: resettare le stime o lasciarle del tutto invariate. Nessuno di questi approcci si dimostra soddisfacente:

- Lasciare invariato il belief a seguito di multiple scansioni senza successo porterebbe a non notare che tutte le persone nell'ambiente in cui ci si trova sono andate via, continuando a considerare valide tutte le posizioni precedenti.
- Dall'altro lato, un approccio troppo drastico quale immediatamente scartare tutte le precedenti stime porterebbe a perdere informazioni utili a causa di occlusioni temporanee (e.g. il robot entra in una stanza vuota).

Per gestire questa situazione effettuiamo uno smoothing degli istogrammi prima di ogni update essendoci una diminuzione della certezza delle misure. Aggiungiamo inoltre del rumore. In questo modo in assenza di osservazioni ci sarà una tendenza al ritorno ad uno stato iniziale, ma sarà graduale in modo da evitare di scartare troppo rapidamente le informazioni pregresse.

Al fine di individuare le zone con alta probabilità di contenere persone abbiamo utilizzato un approccio derivante dall'elaborazione delle immagini. In primo luogo separiamo i punti nella mappa in punti ad alta probabilità di essere occupati da persone e punti con bassa probabilità. Per effettuare questa sogliatura abbiamo utilizzato il metodo Otsu [22], il quale applica un thresholding automatico calcolato minimizzando la varianza dei valori di intensità all'interno delle classi e massimizzandola fra classi differenti all'immagine in input.

Da questa sogliatura otteniamo una mappa binaria in cui sono evidenziate le celle ad alta probabilità. Applicando l'algoritmo in [23], implementato in OpenCV, estraiamo le regioni ad alta probabilità contigue ed i loro contorni. Per ogni regione verrà selezionato come centro il punto con maggiore probabilità (Fig. 13).

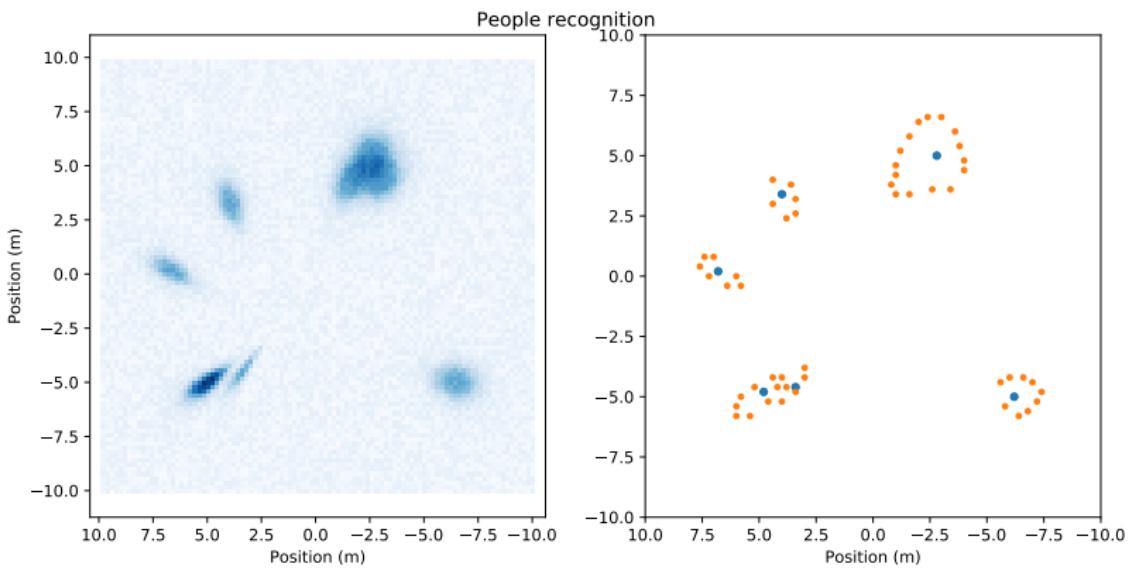


Figura: Segmentazione immagine: a sinistra i valori grezzi del BOF, a destra i contorni delle regioni (giallo) e i centri (blu)

1 Introduzione

2 Gestione dei nodi ROS

3 TIAGo Iron

4 Modello del moto e posizionamento

5 Object recognition

6 Posizione dei target

7 Pianificazione del moto

8 Possibili modifiche

Section 7

Pianificazione del moto

Modalità esplorazione

Quando il robot entra in modalità esplorazione il suo comportamento è di esplorazione casuale della mappa. Il robot continua ad esplorare ed effettuare scan periodici fino a quando non viene rilevato un obiettivo. In tal caso il robot entra in modalità campi di potenziale. Se il robot incontra un ostacolo nel suo cammino ruota di 90° nella direzione con più spazio e continua l'esplorazione casuale.

Tangent bug [24]

Quando il robot entra in modalità bug effettuiamo uno scan lidar. Poiché lavoriamo con valori discreti, confrontiamo i valori ottenuti con una soglia al fine di individuare le discontinuità nel profilo degli oggetti nel range. Analizziamo le discontinuità del segnale per individuare i bordi degli ostacoli (Fig. 14). Il robot si muove verso il punto al quale corrisponde l'angolo più vicino a quello del target, che ci farà allontanare meno dall'obiettivo.

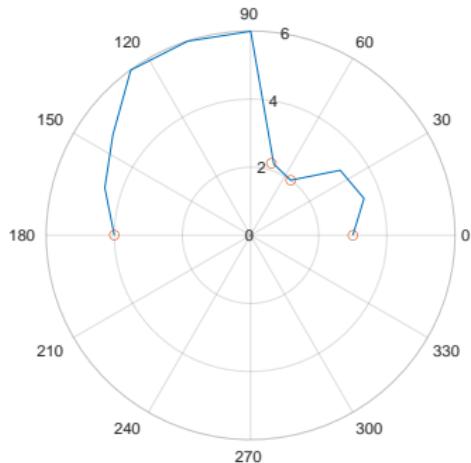


Figura: Profilo dell'ostacolo

Una soluzione al classico problema di muoversi in uno spazio evitando le collisioni è l'uso di un modello basato sui campi di potenziale. Questi schemi richiedono la definizione di potenziali attrattivi e repulsivi.

Tuttavia, questi metodi presentano una problematica significativa: la presenza di un minimo locale causerà una forza totale sul punto nulla, e di conseguenza il robot si fermerà in una posizione non ideale.

Il potenziale attrattivo è centrato sul target, ed è modellato come un potenziale quadratico, la forza attrattiva è quindi lineare rispetto alla distanza (Eq. 14).

Gli ostacoli generano invece un potenziale repulsivo. Questo ha un raggio d'azione limitato, la forza repulsiva è massima per un ostacolo a distanza nulla, e decresce in modo monotono fino ad annullarsi se la distanza dell'ostacolo supera una soglia d_t con la legge descritta nell'equazione 15. La maggior parte dei potenziali repulsivi descritti in letteratura dipendono esclusivamente dalla distanza dall'ostacolo, interferendo con il moto anche se questo avviene parallelamente all'ostacolo. Per ovviare a questo problema prendiamo in considerazione solo gli ostacoli situati in un FOV frontale al robot con ampiezza α calcolata per permettere di muoversi parallelamente ad una superficie mantenendo una distanza di sicurezza h (Eq. 16, Fig. 15):

$$F_{att}(\mathbf{x}) = k_a \cdot |\mathbf{x} - \mathbf{x}_d| \quad (14)$$

$$F_{rep}(obstacleDistance) = \begin{cases} k_r \cdot \left(\frac{1}{obstacleDistance^2} - \frac{1}{d_t^2} \right) & obstacleDistance \leq d_t \\ 0 & d > d_{thresh} \end{cases} \quad (15)$$

$$\alpha = 2 \sin^{-1} \frac{h}{d_t} \quad (16)$$

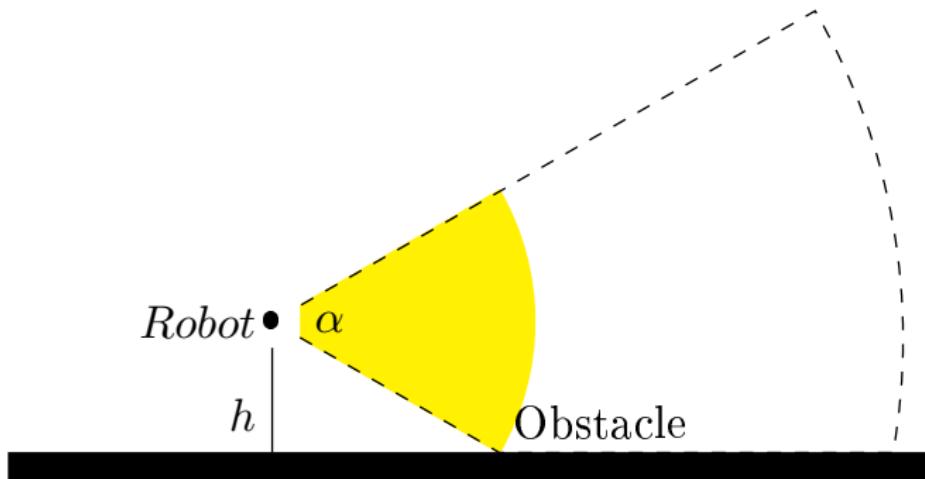


Figura: FOV frontale del robot

L'approccio seguito consiste in due comportamenti calcolati indipendentemente: il comportamento traslatorio e il comportamento rotazionale.

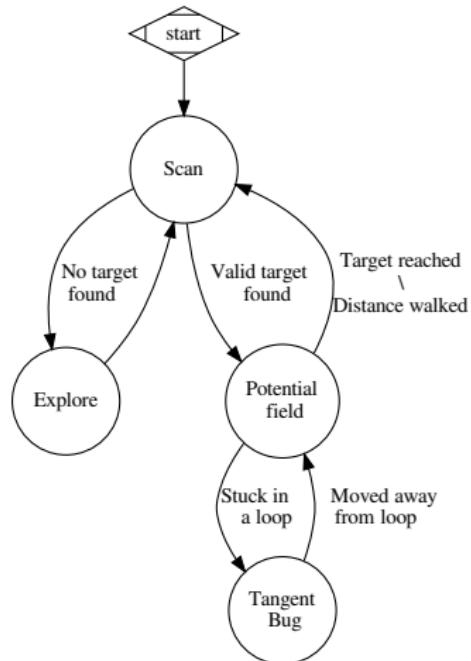
La traslazione viene calcolata in funzione dello spazio di movimento a disposizione e dall'angolo rispetto al target secondo l'equazione 17, ottenuta per interpolazione del profilo desiderato.

$$d = 0.5 + \frac{0.9 \cdot obstacleDistance - 0.5}{1 + \left| \frac{6\omega}{\pi} \right|} \quad (17)$$

$$\omega = \begin{cases} \omega^* = \theta - \angle(\mathbf{F}_{att} - \mathbf{F}_{rep}) & -\Omega \leq \omega^* \leq \Omega \\ -\Omega & \omega^* < -\Omega \\ \Omega & \omega^* < \Omega \end{cases} \quad (18)$$

Scheduling dei comportamenti

In seguito all'avvio, **Change** effettua uno scan dell'ambiente. Nel caso in cui non vengano rilevati obiettivi entra in modalità esplorazione e, dopo aver percorso una determinata distanza, effettua nuovamente uno scan dell'ambiente. Nel caso in cui venga rilevato un obiettivo il robot entra in modalità campi di potenziale e si muove verso l'obiettivo, effettuando nuovi scan nel caso in cui abbia raggiunto l'obiettivo o percorso una determinata distanza. Se, in modalità campi di potenziale, il **TIAGO** rimane bloccato in un loop, entra in modalità Tangent Bug, fin quando non esce dal loop, e ritorna nella modalità campi di potenziale.



- 1** Introduzione
- 2** Gestione dei nodi ROS
- 3** TIAGo Iron
- 4** Modello del moto e posizionamento
- 5** Object recognition
- 6** Posizione dei target
- 7** Pianificazione del moto
- 8** Possibili modifiche

Section 8

Possibili modifiche

Riportiamo di seguito possibili modifiche da apportare:

- Aggiungere un algoritmo per effettuare SLAM, in modo da poter pianificare il moto con algoritmi come A* [25] o RRT [26]
- Migliorare la modalità Tangent Bug
- Utilizzare una depth-cam per stimare con più precisione la distanza delle persone
- Tracciamento delle persone in real-time

Riferimenti bibliografici I

-  Petrônio CL Silva, Paulo VC Batista, Hélder S Lima, Marcos A Alves, Frederico G Guimarães, and Rodrigo CP Silva.
Covid-abs: An agent-based model of covid-19 epidemic to simulate health and economic effects of social distancing interventions.
Chaos, Solitons & Fractals, 139:110088, 2020.
-  Adarsh Jagan Sathyamoorthy, Utsav Patel, Yash Ajay Savle, Moumita Paul, and Dinesh Manocha.
Covid-robot: Monitoring social distancing constraints in crowded scenarios, 2020.
-  Tingxiang Fan, Zhiming Chen, Xuan Zhao, Jing Liang, Cong Shen, Dinesh Manocha, Jia Pan, and Wei Zhang.
Autonomous social distancing in urban environments using a quadruped robot.
arXiv preprint arXiv:2008.08889, 2020.

Riferimenti bibliografici II

-  G. Bradski.
The OpenCV Library.
Dr. Dobb's Journal of Software Tools, 2000.
-  A Rosebrock.
Imutils.
-  J. D. Hunter.
Matplotlib: A 2d graphics environment.
Computing in Science & Engineering, 9(3):90–95, 2007.
-  Charles R. Harris, K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez del R'io, Mark Wiebe, Pearu Peterson, Pierre G'erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant.
Array programming with numpy.

Riferimenti bibliografici III

Nature, 585(7825):357–362, September 2020.

-  Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al.

Scikit-learn: Machine learning in python.

Journal of machine learning research, 12(Oct):2825–2830, 2011.

-  Jordi Pages, Luca Marchionni, and Francesco Ferro.
Tiago: the modular robot that adapts to different research needs.
In *International workshop on robot modularity, IROS*, 2016.

-  Tiago datasheet.

-  Webots.

Webots user guide: Pal robotics' tiago iron.

-  Webots.
Lidar specifications.

Riferimenti bibliografici IV

-  Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard.
Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling.
In *Proceedings of the 2005 IEEE international conference on robotics and automation*, pages 2432–2437. IEEE, 2005.
-  Kurt Seifert and Oscar Camacho.
Implementing positioning algorithms using accelerometers.
Freescale Semiconductor, pages 1–13, 2007.
-  S. Maeyama, N. Ishikawa, and S. Yuta.
Rule based filtering and fusion of odometry and gyroscope for a fail safe dead reckoning system of a mobile robot.
In *1996 IEEE/SICE/RSJ International Conference on Multisensor Fusion and Integration for Intelligent Systems (Cat. No.96TH8242)*, pages 541–548, 1996.

Riferimenti bibliografici V



Debabrata Ghosh and Naima Kaabouch.

A survey on image mosaicing techniques.

Journal of Visual Communication and Image Representation,
34:1–11, 2016.



P. Adarsh, P. Rathi, and M. Kumar.

Yolo v3-tiny: Object detection and recognition using one stage improved model.

In *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, pages 687–694, 2020.



He, Huang, Wei, Li, and Guo.

Tf-yolo: An improved incremental network for real-time object detection.

Applied Sciences, 9:3225, 2019.



Joseph Redmon and Ali Farhadi.

Yolov3: An incremental improvement.
arXiv, 2018.

Riferimenti bibliografici VI

-  Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al.
A density-based algorithm for discovering clusters in large spatial databases with noise.
In *Kdd*, volume 96, pages 226–231, 1996.
-  MK Tay, Kamel Mekhnacha, Manuel Yguel, Christophe Coue, Cédric Pradalier, Christian Laugier, Th Fraichard, and Pierre Bessiere.
The bayesian occupation filter.
In *Probabilistic Reasoning and Decision Making in Sensory-Motor Systems*, pages 77–98. Springer, 2008.
-  Nobuyuki Otsu.
A threshold selection method from gray-level histograms.
IEEE transactions on systems, man, and cybernetics, 9(1):62–66, 1979.

Riferimenti bibliografici VII

-  Satoshi Suzuki et al.
Topological structural analysis of digitized binary images by border following.
Computer vision, graphics, and image processing, 30(1):32–46, 1985.
-  I. Kamon, E. Rivlin, and E. Rimon.
A new range-sensor based globally convergent navigation algorithm for mobile robots.
In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 1, pages 429–435 vol.1, 1996.
-  Akshay Guruji, Himansh Agarwal, and Deep Parsadia.
Time-efficient a* algorithm for robot path planning.
Procedia Technology, 23:144–149, 12 2016.
-  S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller.
Anytime motion planning using the rrt*.
In *2011 IEEE International Conference on Robotics and Automation*, pages 1478–1483, 2011.