Documentazione - Progetto di Robotica

Augello Andrea Castiglione Francesco Paolo La Martina Marco

10 gennaio 2021

Indice

1 Introduzione

L'esplosione della 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 miglior modi di evitare la contrazione del coronavirus è di evitarne l'esposizione, oltre all' utilizzo di mascherine e dispositivi di protezione personale e la frequente igienizzazione delle mani. 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 [**?**]. Un problema chiave si configura di conseguenza come il controllo del rispetto delle norme di distanziamento all'interno degli spazi chiusi. In tal senso l'utilizzo di un robot evita l'inutile esposizione da parte di un operatore all'agente patogeno.

1.1 Nome

Il team ha scelto come nome del robot **Change** in onore di **Chang'e 4** [**?**], la missione parte della seconda fase del programma cinese di esplorazione lunare, durante il quale è andato a buon fine il primo atterraggio morbido sulla faccia nascosta della luna.

2 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 sposta verso l'assembramento evitando gli ostacoli e, arrivato, esorta le persone al rispetto del distanziamento sociale.

3 Stato dell'arte

Allo stato dell'arte, non sono presenti molte soluzioni al problema proposto, poiché l'esplosione della pandemia del coronavirus SARS-CoV-2 è relativamente recente. Sono state trovate solamente 2 proposte:

- Utilizzare un TurtleBot 2, con una camera RGB-D e CCTV per il rilevamento degli assembramenti, una camera termica FLIR C3 per rilevare la temperatura corporea e un lidar 2-D per evitare le collisioni. L'elaborazione delle immagini provenienti da CCTV è eseguita in un laptop con una CPU Intel i7 7th generation e una GPU Nvidia GTX1060, mentre il resto dell'elaborazione è eseguita su una CPU Intel i9 8th generation e una GPU Nvidia RTX2080 montate sul robot. [**?**]
- Un Laikago con 2 camere laterali e CCTV per il rilevamento degli assembramenti e un lidar 3-D per evitare le collisioni. L'elaborazione riguardante il modulo visivo è effettuata in una NVIDIA Jetson AGX Xavier, il resto è eseguito in una CPU Intel i5 8259U. Il rilevamento delle persone viene effettuato tramite la rete YOLO. [**?**]

4 Setup

OS	Ubuntu 18.04 Ubuntu 20.04 Raspberry Pi OS Buster
ROS version	melodic noetic
Webots	R2020b revision 1
Target hardware	Raspberry Pi 3B+

La seguente è una lista delle librerie utilizzate nel nostro progetto ed una breve spiegazione della loro funzione e rilevanza:

- opencv 4.x, una libreria per la computer vision, usata per operazioni di segmentazione [**?**];
- imutils 0.5.3, che include funzioni per semplici operazioni di image processing quali traslazioni, rotazioni, ridimensionamento. Utilizzato inoltre per effettuare Non Maxima Suppression(NMS) [**?**];
- sklearn 0.21.3, una libreria per il machine learning comprendente algoritmi di clustering quali DBSCAN [**?**];
- numpy 1.17.2, una libreria che fornisce supporto per array multidimensionali, matrici ed operazioni matematiche per lavorare su detti array [**?**];
- matplotlib 3.3.3, una libreria per creare visualizzazioni di dati (statiche, dinamiche, interattive). Usata anche per calcolare la posizione di punti rispetto a poligoni [**?**];
- math, una libreria che fornisce funzioni matematiche definite dallo standard C [**?**].

5 TIAGo Iron

Il robot scelto per l'obbiettivo proposto è il **TIAGo Iron**.
Il **PAL Robotics TIAGo Iron** [**?**] è un robot umanoide a due ruote con torso e testa ma senza braccia articolate. Il modello è una piattaforma modulare mobile che permette l'interazione fra esseri umani e robot.
Il datasheet del **TIAGo** [**?**] indica la presenza di speaker e display, tuttavia questi non sono presenti nel modello Webots [**?**]. Abbiamo dunque ritenuto necessario per il nostro scopo aggiungere uno **speaker** e un **display** con corrispondente solido di supporto al modello. La camera del **TIAGo**, come indicato dal datasheet, è RGB-D. Il modello Webots ne è sprovvisto, di conseguenza è stata utilizzata una camera monoscopica RGB. Inoltre è stato necessario contattare gli sviluppatori del **TIAGo** per chiedere informazioni circa le dimensioni esatte delle **ruote** in quanto tale informazione è omessa dal datasheet. Ci è stato comunicato che le ruote del **TIAGo** hanno diametro di $200\,mm$. Utilizzando tale valore nei calcoli odometrici, abbiamo ottenuto valori differenti dalle misurazioni. Abbiamo quindi attentamente letto il file *proto* del modello di Webots, e abbiamo scoperto che nel modello le ruote hanno un diametro di $3\,cm$ inferiore.

L'IMU utilizzata ha 6 gradi di libertà ed è composta delle seguenti componenti:

1. giroscopio;
2. accelerometro;

Abbiamo ritenuto non necessario aggiungere il **magnetometro** in quanto in uno scenario reale sarebbe stato soggetto ad interferenze (significativamente più di un giroscopio), specialmente in un ambiente con molti oggetti metallici (quale potenzialmente lo scenario di utilizzo del nostro robot).

Il modello Webots del **TIAGo** presenta un lidar, ovvero il modello Hokuyo URG-04LX-UG01 [**?**] che, come specificato nel datasheet, ha un range di $5.6\,m$ ed un FOV di $240°$ (agli estremi è parzialmente occluso). In seguito ad analisi sperimentale abbiamo dedotto che l'occlusione del lidar causa malfunzionamenti del package GMapping [**?**], utilizzato per implementare le funzionalità di SLAM.

6 Ambiente

Abbiamo considerato opportuno analizzare e studiare il package **webots_ros** [**?**] al fine di raggiungere una comprensione più profonda sulle metodologie per interfacciare i nodi ROS con il controller ROS standard per Webots. Inoltre è risultato necessario approfondire la documentazione ROS [**?**] al fine di installare e configurare l'ambiente di sviluppo ROS e per capire i concetti fondamentali relativi ai nodi e topics. Infine abbiamo impostato l'interfaccia ROS su Webots seguendo la documentazione cyberbotics rilevante [**?**].

7 ROS

7.1 Descrizione

ROS [**?**] è un meta sistema operativo open-source. Fornisce tutti i servizi che caratterizzano un sistema operativo quali l'astrazione hardware, il controllo dei dispositivi a basso livello, l'implementazione delle funzionalità più usate, lo scambio di messaggi fra processi e la gestione dei pacchetti. Fornisce inoltre librerie e strumenti per scrivere, ottenere e eseguire codice su più computer. Il "grafo" runtime è una rete peer-to-peer di processi (potenzialmente distribuiti su più macchine) collegati attraverso l'infrastruttura ROS.

ROS implementa vari stili di comunicazione, fra i quali streaming di dati asincrono attraverso i **topics** e comunicazione sincrona attraverso i **servizi**.

I topics sono bus con identificativi attraverso i quali i nodi scambiano messaggi. I topics hanno una semantica di publish/subscribe anonima, che separa la produzione di informazioni dal loro utilizzo. In generale, i nodi non sanno con chi comunicano. I nodi interessati a dei dati si iscrivono (subscribe) al topic rilevante. I nodi che generano dati pubblicano (publish) al topic rilevante. Possono esserci più publishers e subscribers per ogni topic. I topic sono pensati per un flusso unidirezionale di informazione. I nodi che devono utilizzare chiamate a procedure remote, ad esempio ricevere la risposta ad una richiesta, dovrebbero usare i servizi. Un servizio è definito da una coppia di messaggi: una richiesta ed una risposta. Un nodo ROS offre un servizio attraverso una stringa nome e un client chiama il servizio mandando la richiesta ed aspettando la risposta. Le librerie per i client solitamente presentano tale interazione allo sviluppatore come se fosse una procedura a chiamata remota. I servizi sono definiti usando i file **srv**, che sono compilati in codice sorgente da una libreria del client ROS.

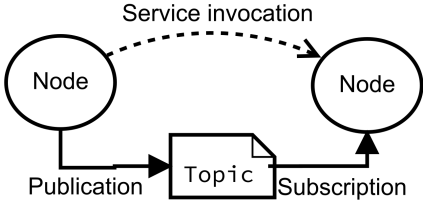

Figura 1: Concetti di base ROS

7.2 Bug

Nel file RosSpeaker.cpp, parte del controllore ROS per Webots [**?**], è stato riscontrato un bug. Il metodo per verificare che si sta riproducendo un suono da file audio ed il metodo per verificare che il robot sta parlando presentavano difatti le funzioni callback invertite. In seguito al nostro avviso via email il bug è stato risolto nel successivo aggiornamento, come si evince dal commit GitHub rilevante [**?**], riga 32-34.

7.3 Gestione dei nodi

7.3.1 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.

7.3.2 Change 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

7.3.3 Interaction node

Questo nodo ha il compito di gestire le interazioni audio/video ed è in esecuzione sul Raspberry. In particolare viene sfruttato il display per caricare della immagini e il sintetizzatore vocale dello speaker per riprodurre messaggi audio. 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 (riportato anche testualmente sul display) che invita a rispettare il distanziamento sociale quando rileva un assembramento o quando scansiona l'ambiente

7.3.4 Odometry node

Il nodo che si occupa dell'odometria, eseguito sul Raspberry, si occupa di stimare la posizione del robot, come spiegato approfonditamente nella sezione 8.3. In generale ciò che fa è integrare costantemente i valori del giroscopio e della velocità delle ruote per condividere posizione e orientamento del robot.

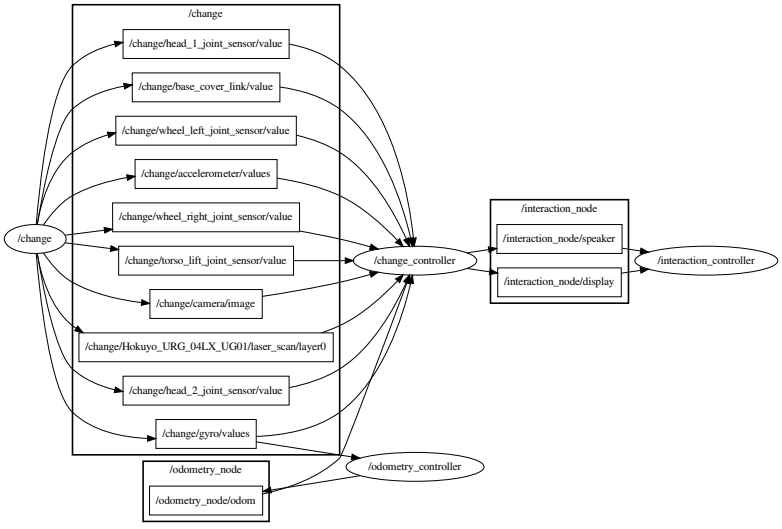

Figura 2: Architettura dei nodi ROS, ottenuta tramite *rqt*

8 Modello del moto e posizionamento

8.1 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).

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

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

8.2 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).

$$targetAngle = currentAngle + 2\pi \frac{distance}{2\pi \cdot diameter} \tag{2}$$

8.3 Posizionamento

Per stimare il posizionamento, inizialmente abbiamo utilizzato i dati dell'accelerometro [**?**], 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 per mitigare il drift.

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

In un secondo momento abbiamo ritenuto opportuno utilizzare un nodo ROS che si occupasse di aggiornare e 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 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_i 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:

$$\mathbf{P}_i = \begin{bmatrix} x_i & y_i \end{bmatrix}^T \tag{4}$$

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

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

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

La θ_i è data dall'interpolazione lineare descritta nell'equazione (1).

Abbiamo misurato le performance della stima di posizione e i risultati sono ritenuti soddisfacenti per raggiungere l'obbiettivo proposto. In Fig. 3 sono mostrati i risultati delle misurazioni effettuate.

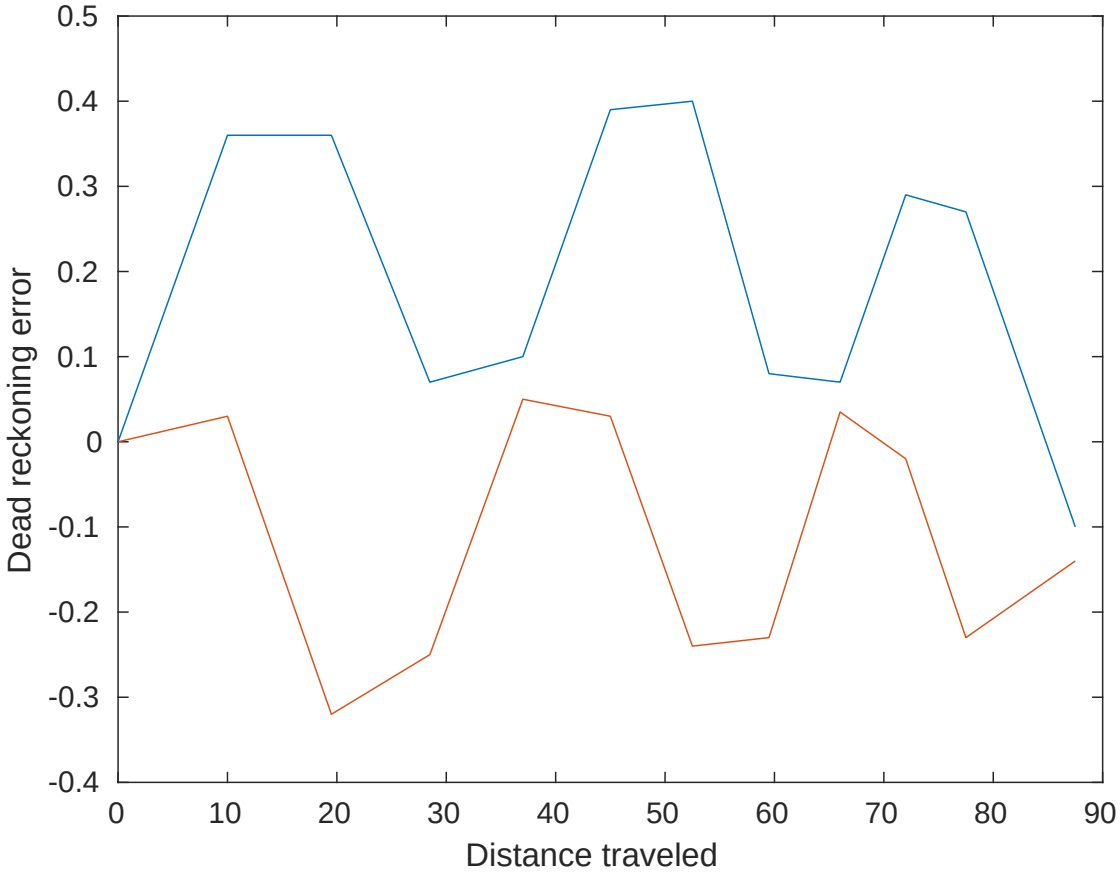

Figura 3: Errore nella stima della posizione

In Fig. 4 viene mostrata chiaramente la differenza fra la traiettoria reale del **TIAGo** e la traiettoria calcolata con l'odometria.

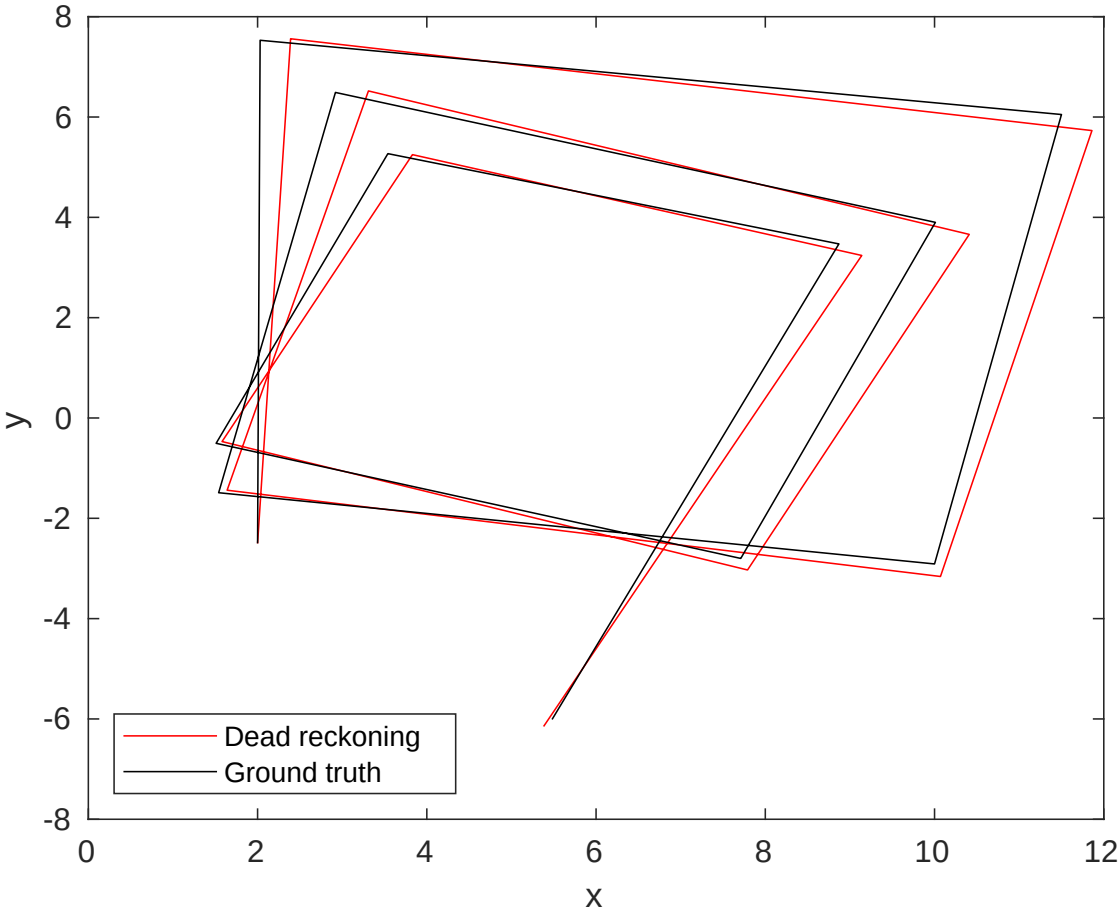

Figura 4: Errore nella stima della traiettoria

9 Prevenzione delle collisioni

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 difatti imprescindibile per raggiungere l'obbiettivo 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.

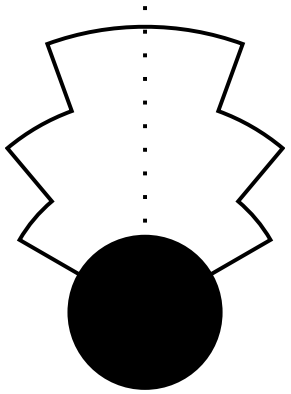

Figura 5: Collision avoidance

10 Object recognition

10.1 Campionamento delle immagini

Il FOV della camera è di 57°, di conseguenza per ricoprire 360° è stato necessario effettuare 7 campionamenti. Il settimo campionamento, come si evince dalla figura 6, è sovrapposto al primo per una porzione di scena pari a 39° coincidente col primo campionamento.

È però possibile che un individuo si trovi in una zona di confine tra due campioni, e che quindi non risulti correttamente identificabile in nessuna delle due immagini in cui è parzialmente contenuto. Per mitigare questo problema effettuiamo una rudimentale operazione di image mosaicing [**?**] e campioniamo l'immagine così ottenuta ad intervalli di 28°.

La rotazione ha una precisione limitata, inferiore alla granularità con cui possiamo ottenere informazioni sull'orientamento del robot. Nel corso di una scansione l'errore sulle rotazioni può accumularsi e risultare significativo. Per questo motivo compensiamo la differenza tra la posizione effettiva stimata del robot e la posizione che dovrebbe occupare attraverso una rotazione della testa.

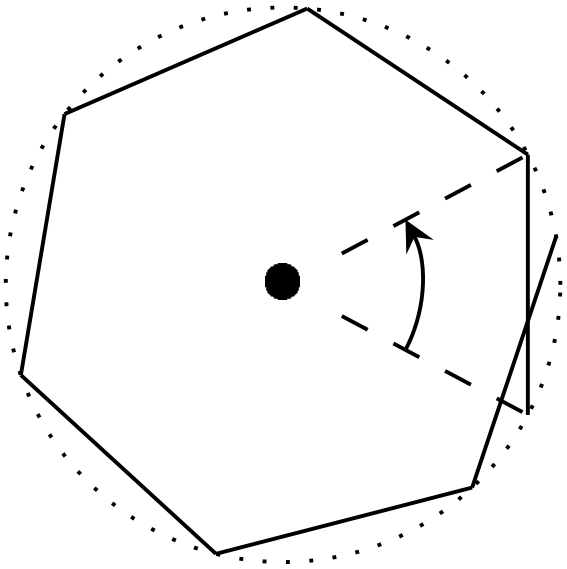

Figura 6: Campionamento delle immagini

10.2 YOLO

Al fine di riconoscere le persone è stato necessario utilizzare sistemi di **object recognition**. Per fare ciò abbiamo valutato le performance di YOLOv3 (you only look once), YOLOv3-tiny, HoG (Histogram of oriented gradients), HoG + SVG (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% [**?**]), sacrificando in termini di precisione ma comunque sufficientemente preciso per il nostro obbiettivo. La tabella 3 [**?**], mostra l'architettura di YOLOv3-tiny. La tabella 1 mostra un paragone fra le versioni di YOLO e le rispettive versioni tiny [**?**]. Inoltre è rilevante in tal senso un paragone fra YOLOv3 e YOLOv3-tiny in 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 [**?**]:

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

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 2: Paragone fra le versioni di YOLOv3

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 3: Architettura di YOLOv3-tiny

10.3 Triangolazione

La triangolazione come metodo di individuazione delle persone, sebbene teoricamente possibile, presenta dei problemi nel nostro scenario. In primo luogo prendiamo in esame l'occlusione delle persone. Ad esempio, 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 persona 5.

Utilizzando una rappresentazione basata su oggetti si presenta anche il problema dell'imputazione delle osservazioni. Quando il robot effettua scan successivi, non sarebbe altrimenti in grado di dedurre quali osservazioni derivano dalla stessa persona. Poiché nel nostro scenario abbiamo più persone in una stanza, non si potrebbe determinare quali intersezioni delle rette corrispondono ad osservazioni reali o se si tratta di intersezioni spurie.

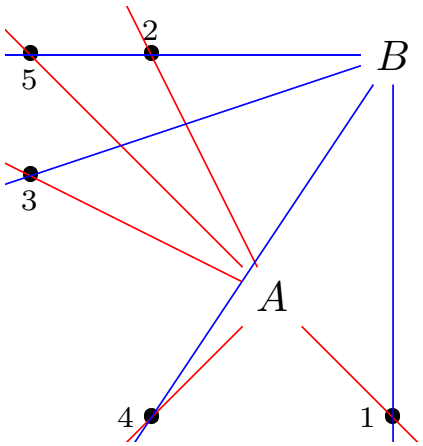

Figura 7: Triangolazione

10.4 Calcolo della distanza

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'obbiettivo 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. 8).

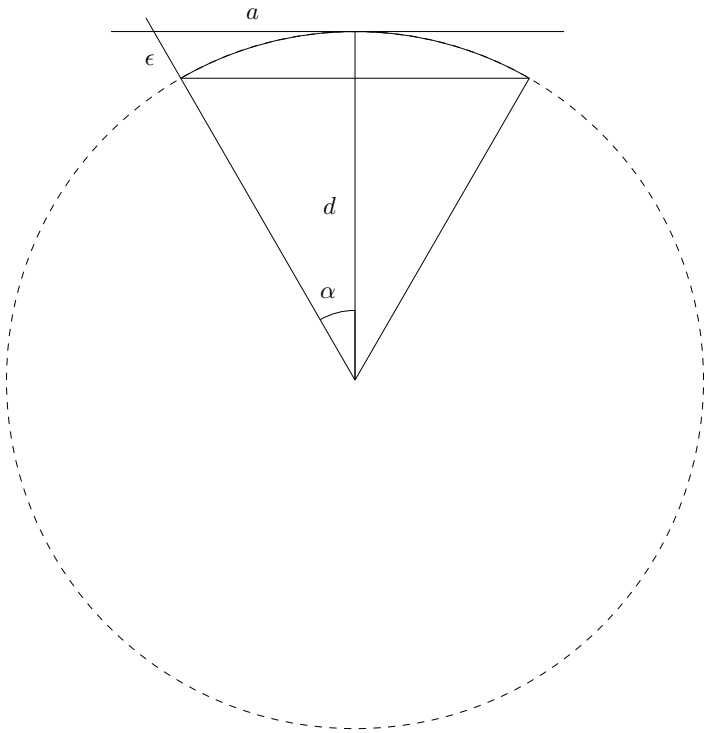

Figura 8: Errore nella stima della distanza con una linearizzazione a tratti della circonferenza

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%.

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

Sotto tali ipotesi possiamo quindi calcolare la distanza di un oggetto come mostrato in Eq. 8 dove f è il fuoco.

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

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.

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

10.5 Scarto dei duplicati

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) [**?**], 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 si evince dalla figura 9, 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.

Abbiamo ritenuto opportuno utilizzare l'implementazione dell'algoritmo fornita da **sklearn** [**?**].

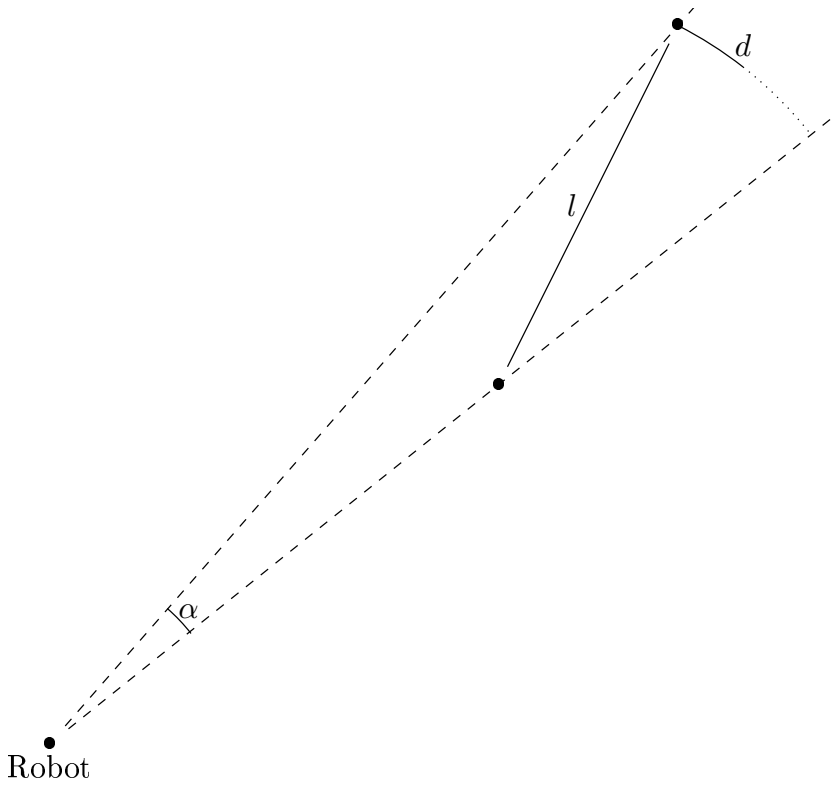

Figura 9: Non maxima suppression

11 Posizione dei target

11.1 Correzione dei valori

Al fine di migliorare la stima sulla distanza dei target abbiamo ritenuto utile paragonare le reali distanze dei target con le stime effettuate dal nostro sistema. In seguito ad analisi quantitative abbiamo ottenuto il polinomio $0.003116 * x^5 - 0.09722 * x^4 + 1.124 * x^3 - 5.908 * x^2 + 14.5 * x - 7.367$, che approssima la funzione di correzione della stima. Il grafico è mostrato nella Fig. 10

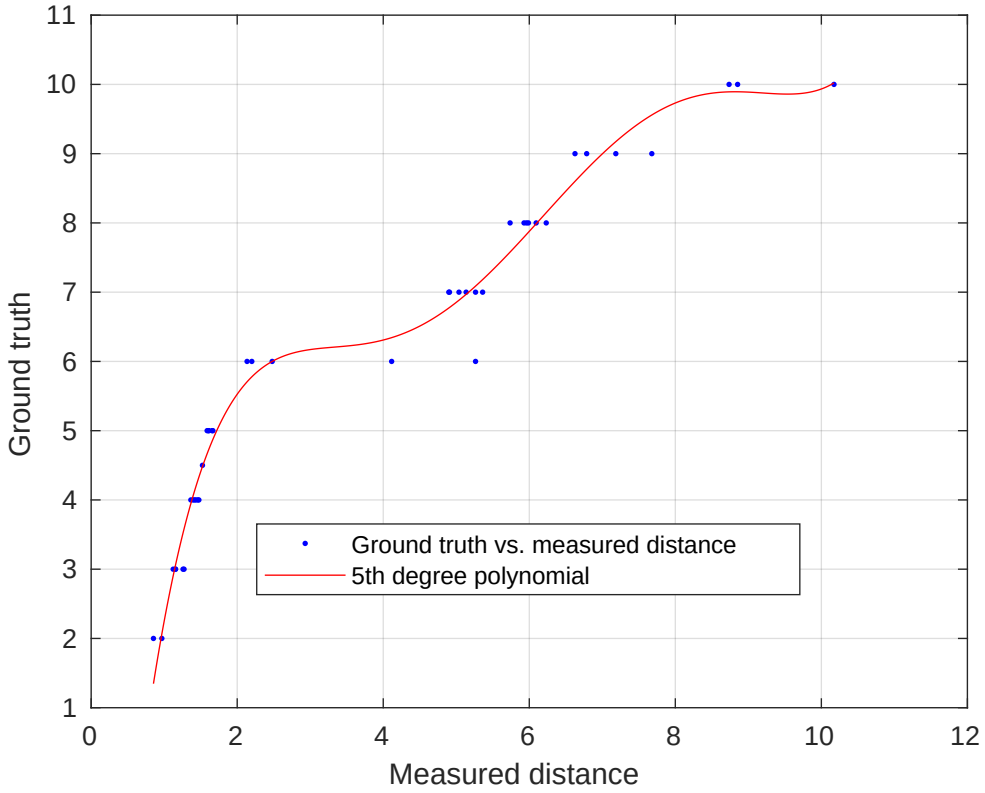

Figura 10: Polinomio interpolante

Le performance del polinomio nella correzione della stima della distanza e dell'angolo si evince dalle Fig. 11

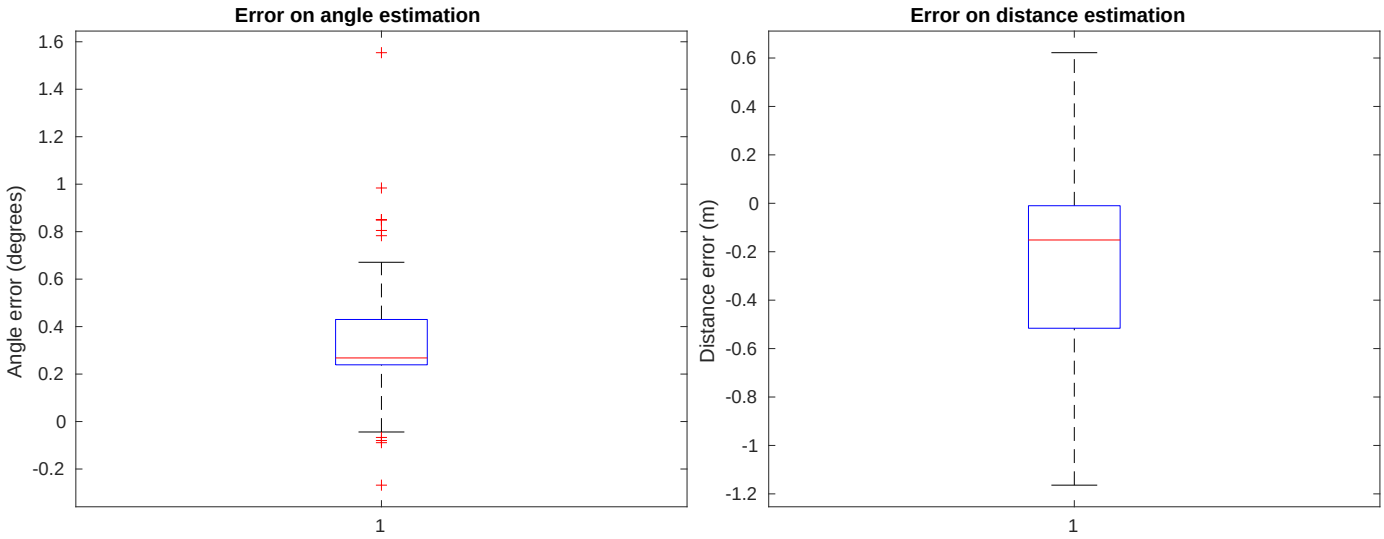

Figura 11: Box plot relativi all'errore su angolo e distanza

La matrice di covarianza è mostrata nell'equazione 10

$$V = \begin{bmatrix} 0.163 & -0.0397 \\ -0.0397 & 0.0752 \end{bmatrix} \tag{10}$$

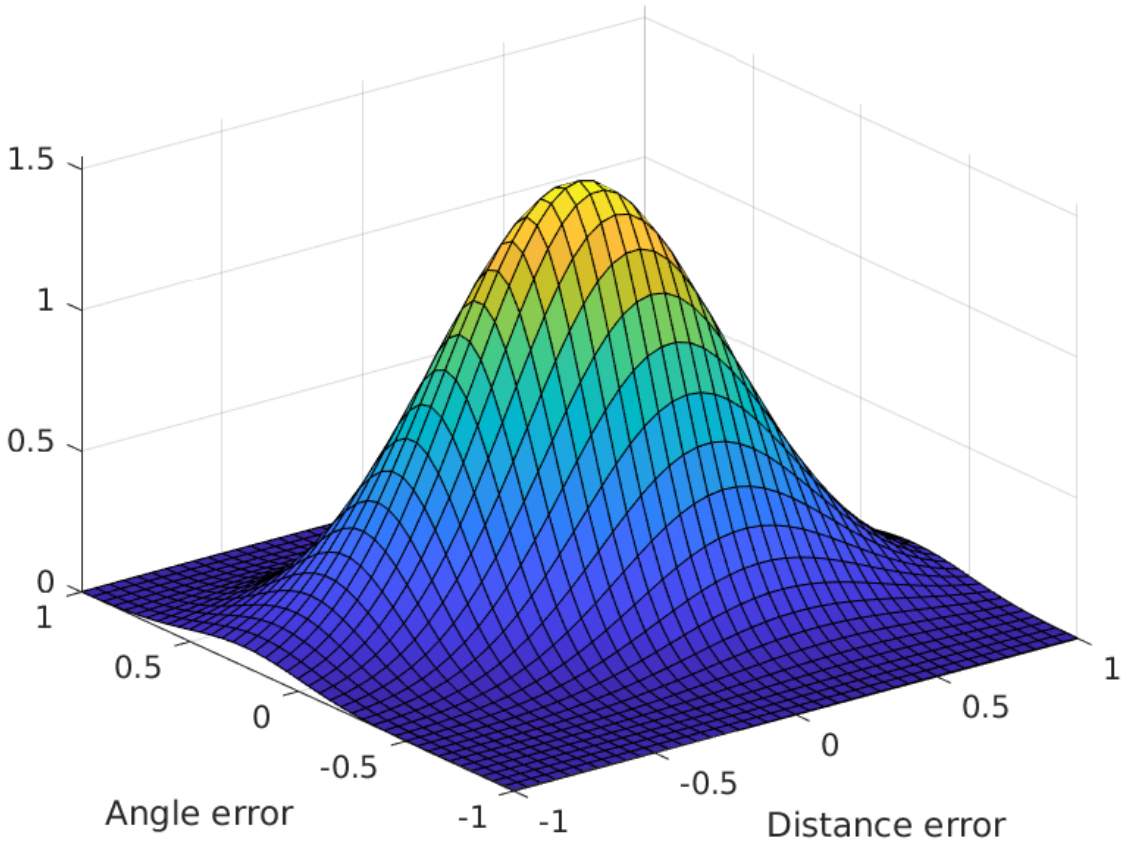

Figura 12: Distribuzione gaussiana multivariata con covarianza data nell'equazione 10 in 3 dimensioni

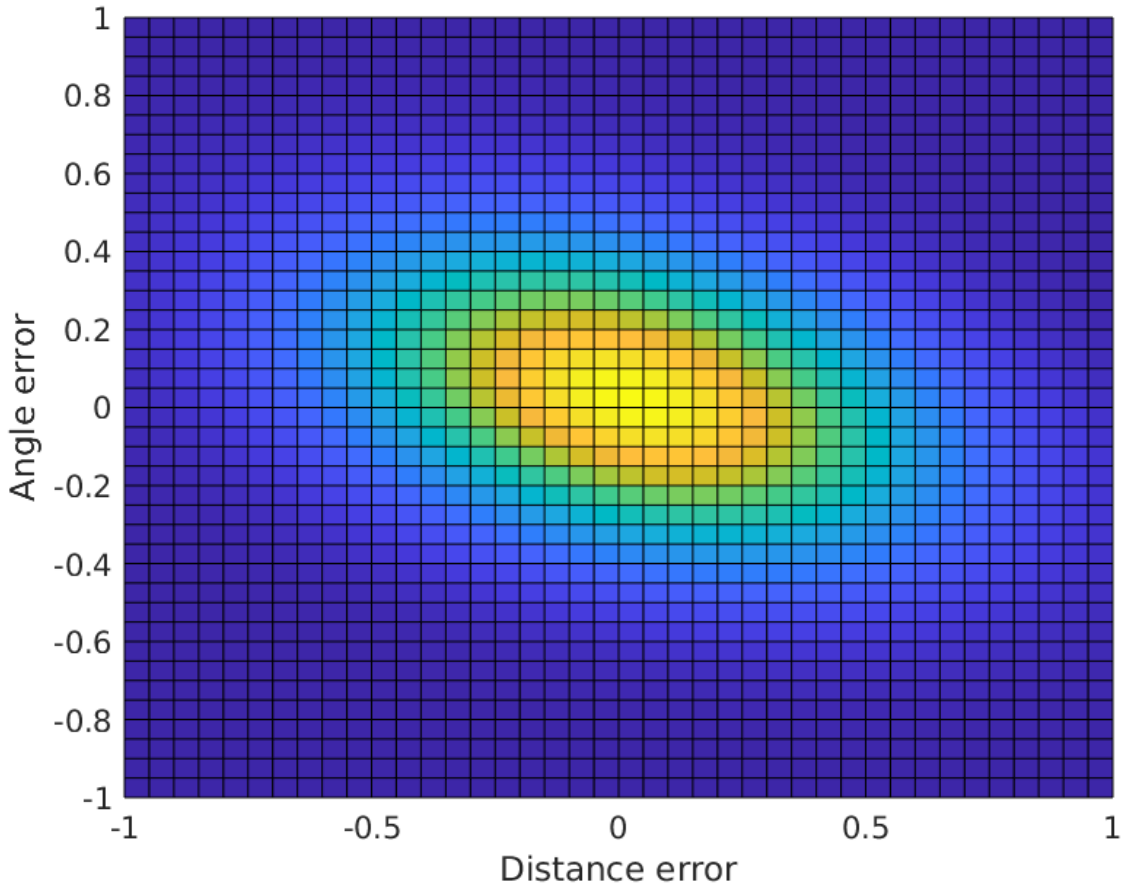

Figura 13: Distribuzione gaussiana multivariata con covarianza data nell'equazione 10 in 2 dimensioni

11.2 Modello probabilistico

Le relazioni geometriche sulla distanza degli oggetti esposte nella sezione 10.4 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. La correzione apportata con il polinomio interpolante migliora la precisione del posizionamento, ma i risultati ottenuti rimangono poco soddisfacenti. Inoltre basarsi esclusivamente su un singolo scan per determinare la posizione dei target non sfrutta il fatto che la camera è situata su una piattaforma mobile [**?**]. 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) [**?**].

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. 14 con un numero così elevato di iterazioni i tempi di esecuzione non sarebbero accettabili, è stato quindi necessario ricorrere ad una approssimazione.

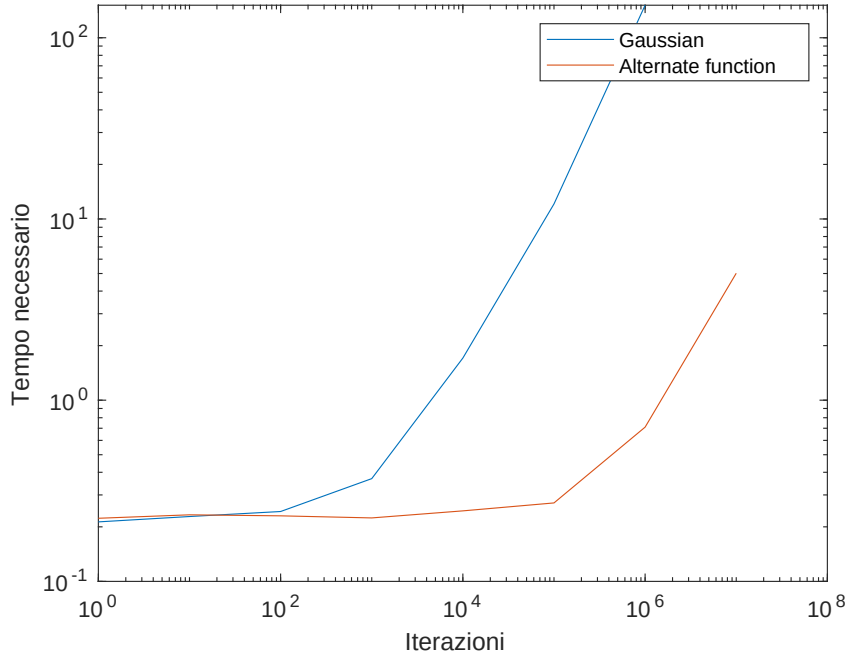

Figura 14: Benchmark funzione densità di probabilità (probability density function, funzione)

La distribuzione triangolare è spesso utilizzata per approssimare una gaussiana, tuttavia, presenta delle caratteristiche non volute 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, dopo una prima osservazione, questa non viene confermata nello scan immediatamente successivo, non è 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 ottenere un'approssimazione di una gaussiana adatta al nostro scenario, per modellare la probabilità che data l'occupazione della cella in posizione $\mathbf{x}$ si ottenga l'osservazione $\mathbf{z}$, è stato utilizzato un funzionale ispirato al guadagno del filtro di Butterworth.

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

Nell'equazione 11 è mostrato il funzionale utilizzato. In figura 15 si mostra un confronto della forma di questa funzione rispetto ad una gaussiana con media nulla e deviazione standard unitaria nel caso monodimensionale.

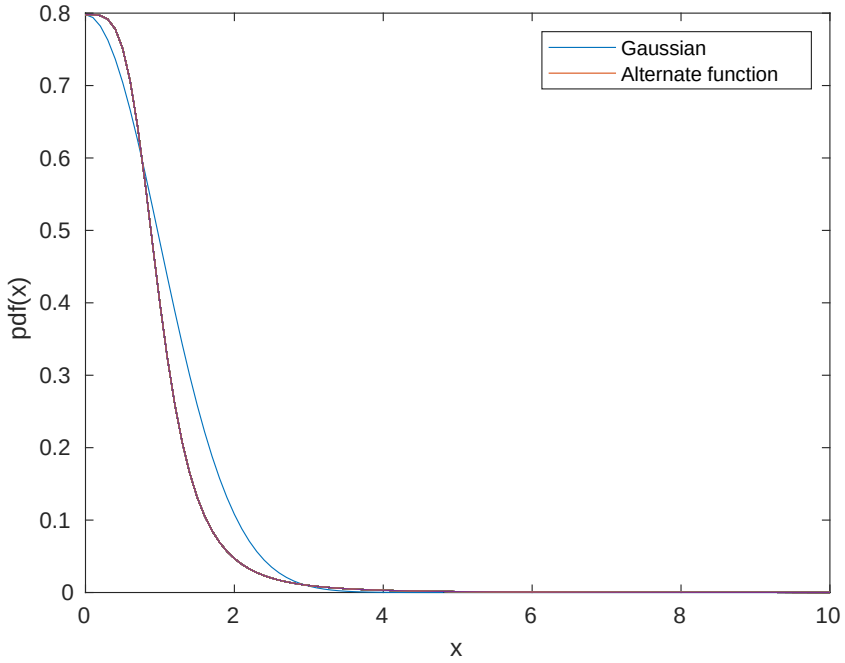

Figura 15: Forma pdf

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

La distanza euclidea porterebbe a formare aree ad alta probabilità di forma circolare. Questo però non si adatterebbe 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 16, 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.

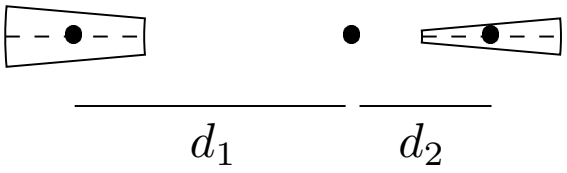

Figura 16: Area della regione probabile a seconda della distanza.

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.

$$p(\mathbf{x}_i|Z_i) = p(\mathbf{x}_{i-1}) \cdot \sum_{\mathbf{z} \in Z_i} p(\mathbf{z}|\mathbf{x}_i) \tag{13}$$

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_i 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 [**?**], 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 [**?**], 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. 17).

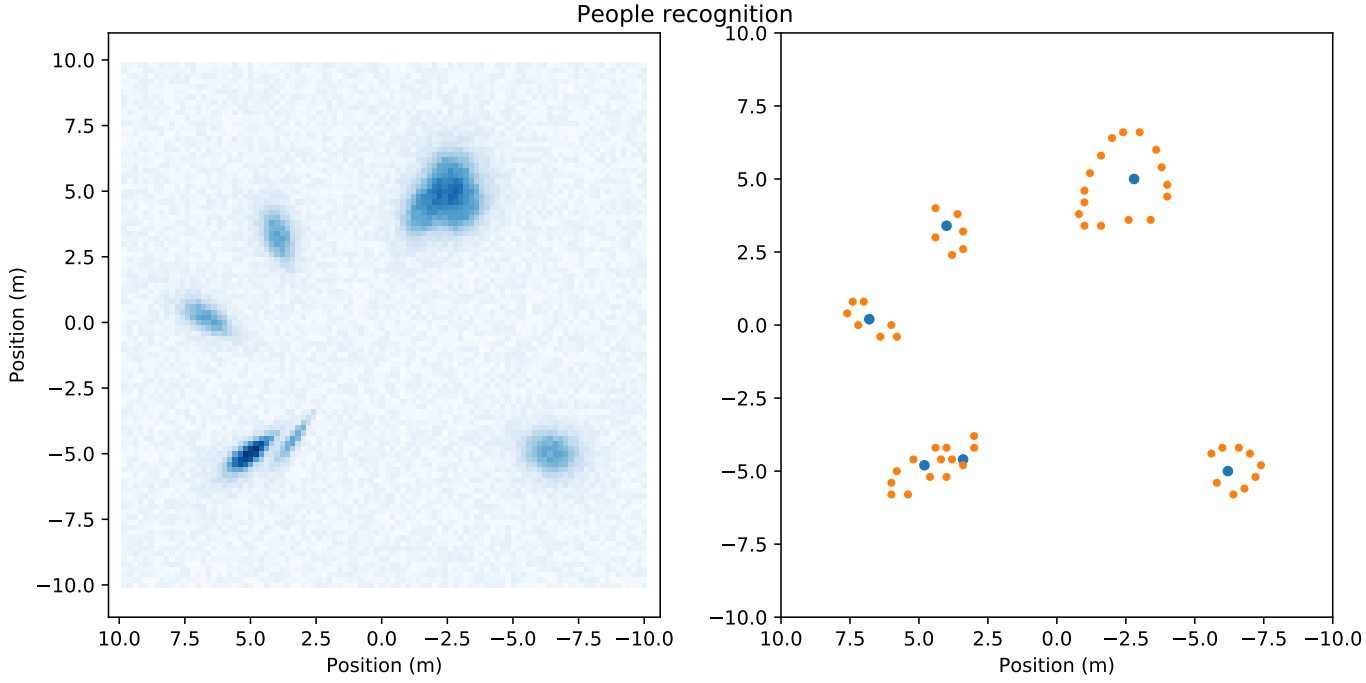

Figura 17: Segmentazione immagine: a sinistra i valori grezzi del BOF, a destra i contorni delle regioni (giallo) e i centri (blu)

12 Scheduling dei comportamenti

12.1 Modalità di movimento

12.1.1 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 obbiettivo. 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.

12.1.2 Bug mode

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 successivamente le discontinuità del segnale, corrispondenti ai bordi degli ostacoli, come si evince dalla fig. 18. Da tali punti viene calcolato l'angolo fra la retta che li congiunge con l'obbiettivo. Il robot si muove infine verso il punto al quale corrisponde l'angolo minore, che ci farà allontanare meno dall'obbiettivo. Se il profilo dell'ostacolo è una circonferenza, la retta fra il punto con angolo minimo e l'obbiettivo è tangente all'ostacolo, da cui il nome dell'algoritmo Tangent Bug [?].

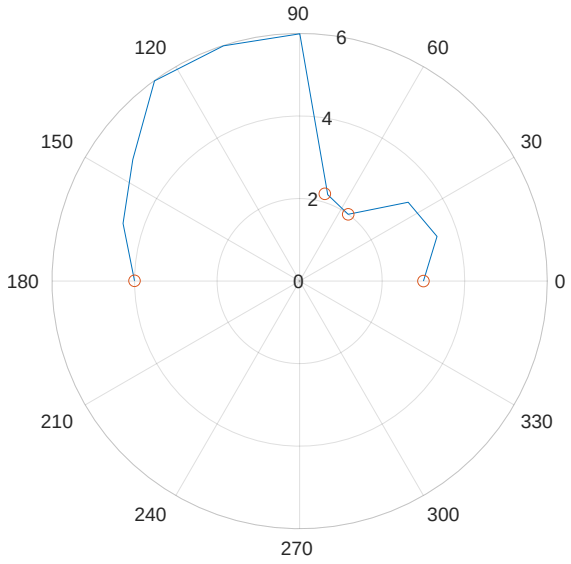

Figura 18: Profilo dell'ostacolo

12.1.3 Campi di potenziale

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).

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

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.

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

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. 19):

$$\alpha = 2\sin^{-1}\frac{h}{d_t} \tag{16}$$

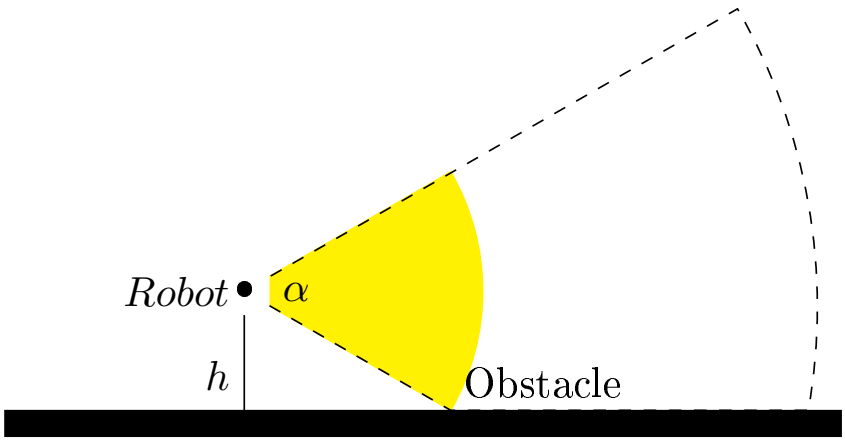

Figura 19: FOV frontale del robot

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. L'angolo viene invece determinato dalla somma vettoriale delle forze attrattive e repulsive. La rotazione viene inoltre sogliata per evitare cambi di direzione troppo bruschi(Eq. 18).

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

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

12.2 Automa a stati finiti

Nella figura 20 viene riportato l'automa a stati finiti che descrive i comportamenti del robot. In seguito all'avvio, **Change** effettua uno scan dell'ambiente. Nel caso in cui non vengano rilevati obbiettivi entra in modalità esplorazione, avviando l'esplorazione casuale e, dopo aver percorso una determinata distanza, effettua nuovamente uno scan dell'ambiente. Nel caso in cui venga rilevato un obbiettivo il robot entra in modalità campi di potenziale e si ferma nel caso in cui abbia raggiunto l'obbiettivo 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.

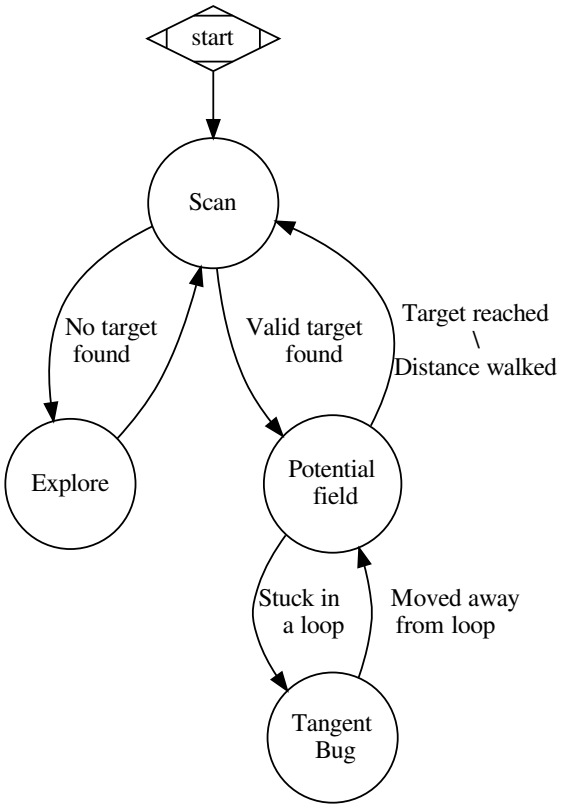

Figura 20: Automa a stati finiti

13 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* [**?**] o RRT [**?**]
- Migliorare la modalità Tangent Bug
- Utilizzare una depth-cam per stimare con più precisione la distanza delle persone
- Tracciamento delle persone in real-time

14 Conclusioni

Durante lo sviluppo del progetto sono state affrontate numerose tematiche trattate durante il corso di *Robotica*. L'esperienza di sviluppo ha permesso al team di apprezzare l'importanza di solide fondamenta teoriche unite ad una significativa esperienza pratica. In particolar luogo abbiamo riscontrato problemi concernenti vari ambiti, dalla robotica probabilistica all'esplorazione degli ambienti, la cui ricerca di risoluzione ci ha portato ad approfondire le fondamenta teoriche della materia. La performance raggiunta, nonostante le tempistiche, risulta soddisfacente.