

Project log - Robotica

Augello Andrea

Castiglione Francesco Paolo

La Martina Marco

2 gennaio 2021

Indice

1 Setup	1
2 Nome	1
3 Ambiente	2
4 ROS	2
4.1 Bug	2
5 Dipendenze	2
6 Obbiettivo	2
7 TIAGo Iron	2
8 Modello del moto e posizionamento	3
9 Object recognition	3
9.1 Campionamento delle immagini	3
9.2 Yolo	3
9.3 Scarto dei duplicati	4
10 Posizione dei target	4
10.1 Triangolazione	4
10.2 Calcolo della distanza	5
10.3 Modello probabilistico	6
11 Pianificazione del moto	9
11.1 Modalità di movimento	9
11.1.1 Exploration mode	9
11.1.2 Bug mode	9
11.1.3 Campi di potenziale	9
11.2 Pianificazione	9

1 Setup

OS	Ubuntu 18.04 Ubuntu 20.04
ROS version	melodic noetic
Webots	R2020b revision 1
Target hardware	Raspberry Pi 4B Raspberry Pi 3B+

2 Nome

Il team ha scelto il nome **Change** in onore di **Chang'e 4** [1], 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.

3 Ambiente

Abbiamo considerato opportuno analizzare e studiare il package **webots_ros** [2] 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 [3] al fine di installare e configurare l'ambiente ROS ed inoltre per capire i concetti fondamentali relativi ai nodi e topics. Infine abbiamo impostato l'interfaccia ROS su Webots seguendo la documentazione cyberbotics rilevante [3].

4 ROS

4.1 Bug

Logical values did not allow callbacks.

5 Dipendenze

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 [4];
- imutils, che include funzioni per semplici operazioni di image processing quali traslazioni, rotazioni, ridimensionamento. Utilizzato inoltre per effettuare Non Maxima Suppression(NMS) [5];
- sklearn, una libreria per il machine learning comprendente algoritmi di clustering quali DBSCAN [6];
- numpy, una libreria che fornisce supporto per array multidimensionali, matrici ed operazioni matematiche per lavorare su detti array [7];
- matplotlib, una libreria per creare visualizzazioni di dati (statiche, dinamiche, interattive) [8];
- math, una libreria che fornisce funzioni matematiche definite dallo standard C [9].

6 Obbiettivo

L'obiettivo del robot è di **evitare assembramenti in ambienti indoor**.

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.

7 TIAGo Iron

Il robot scelto per l'obiettivo proposto è il **TIAGo Iron**.

Il **PAL Robotics TIAGo Iron** [10] 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 raggio di 200 mm . Utilizzando tale valore nei calcoli odometrici, abbiamo ottenuto valori largamente differenti dalle misurazioni. Abbiamo quindi dedotto sperimentalmente che il raggio del modello Webots è di 3 cm più lungo.

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 **lidar** del **TIAGo**, come indicato nel datasheet [11], ha un range di 5.6 m ed un FOV di 240° (ricordiamo che agli estremi è parzialmente occluso).

8 Modello del moto e posizionamento

Il modello del moto è caratterizzato da rotazioni e traslazioni. Per ottenere l'angolo di rotazione ci basiamo sui dati ottenuti dal giroscopio, il quale misura il moto rotazionale fornendo una velocità angolare. Per ottenere l'angolo di rotazione effettuiamo quindi un'integrazione discreta dei campioni con interpolazione lineare del primo ordine. Per effettuare lo spostamento lineare utilizziamo il controllore PID (Proporzionale-Integrale-Derivativo) delle ruote fornito da Webots, che richiede l'angolo di rotazione corrente, il diametro delle ruote e fornisce l'angolo di rotazione necessario al fine di ottenere lo spostamento desiderato.

$$targetAngle = currentAngle + 2\pi \frac{distance}{2\pi \cdot diameter} \quad (1)$$

A causa di possibili imprecisioni calcoliamo la stima dello spostamento lineare utilizzando l'accelerometro. Al segnale dell'accelerometro viene applicato un integrale doppio per ottenere lo spostamento lineare.

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.

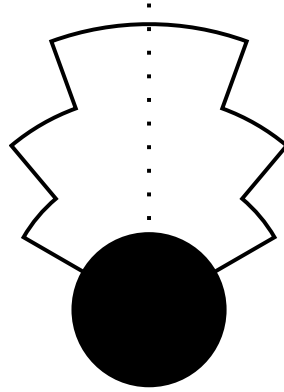


Figura 1: Collision detection

9 Object recognition

9.1 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 2, è sovrapposto al primo per una porzione di scena pari a 39° coincidente col primo campionamento.

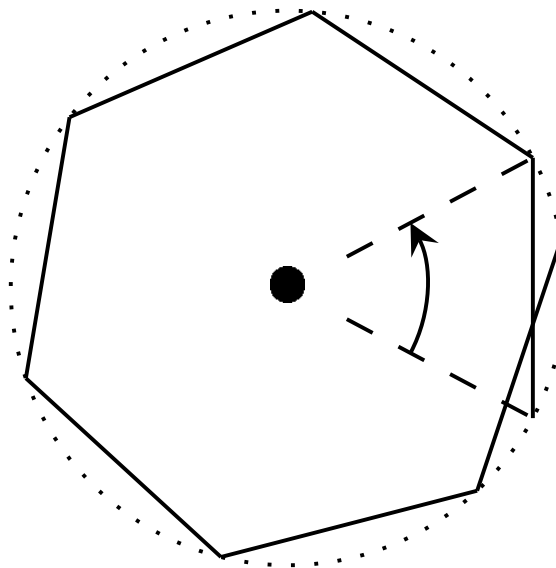


Figura 2: Campionamento delle immagini

9.2 Yolo

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 + 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 finetuning degli iperparametri parametri 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, sacrificando in termini di precisione ma comunque sufficientemente preciso per il nostro obiettivo. Ecco un paragone fra YOLOv3 e YOLOv3-tiny in termini di mAP (mean average precision) e FLOPS (floating-point operations per second), come illustrato dalla tabella seguente, i cui dati provengono dal sito di YOLO [12]:

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

9.3 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) [13], 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 3, 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** [6].

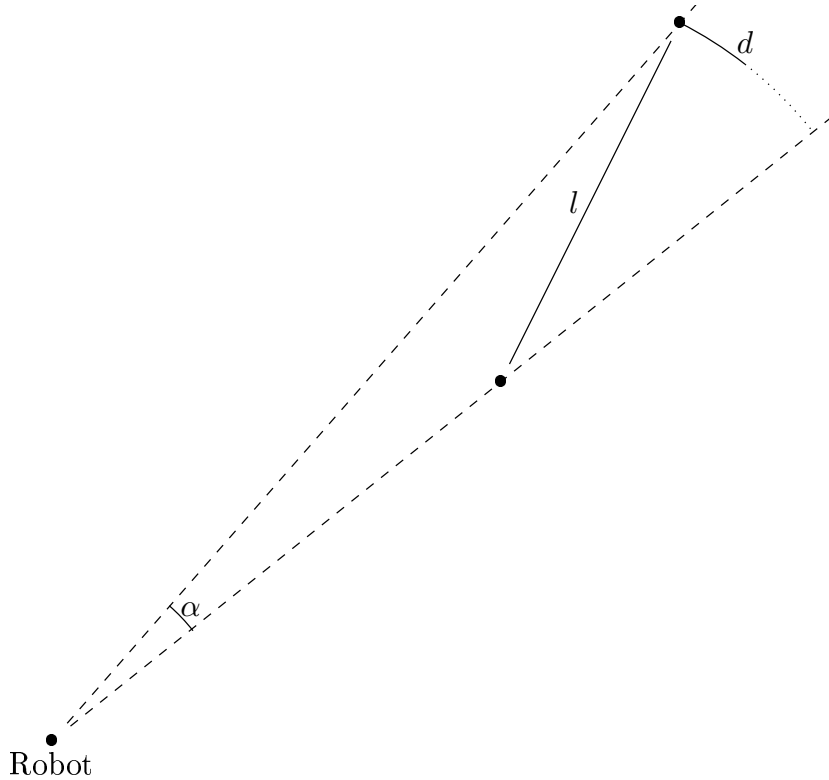


Figura 3: Non maxima suppression

10 Posizione dei target

10.1 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. Inoltre, quando il robot effettua scan successivi, non sarebbe in grado di dedurre quali osservazioni derivano

dalla stessa persona. Infatti, poiché nel nostro scenario abbiamo più persone in una stanza, non è possibile dedurre quali intersezioni delle rette corrispondono ad osservazioni reali o se si tratta di intersezioni spurie.

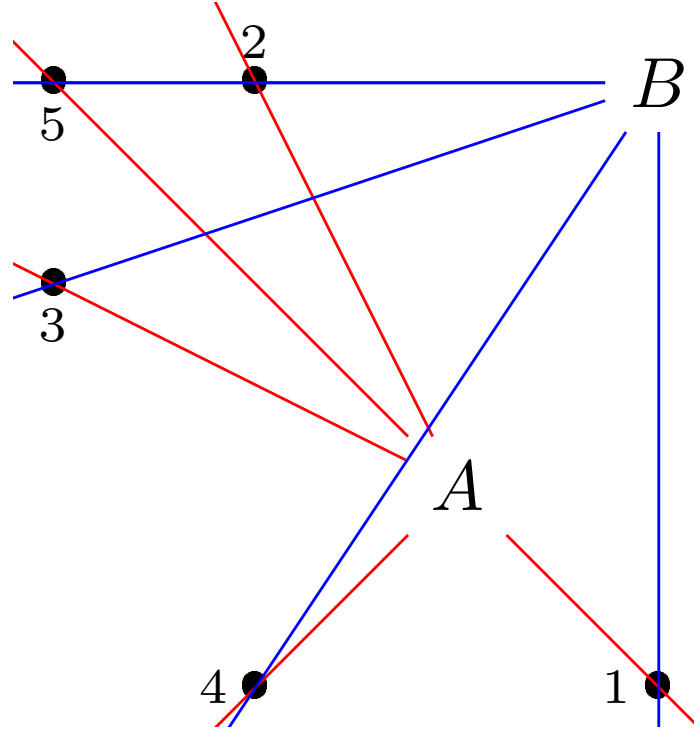


Figura 4: Triangolazione

10.2 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. 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. 5).

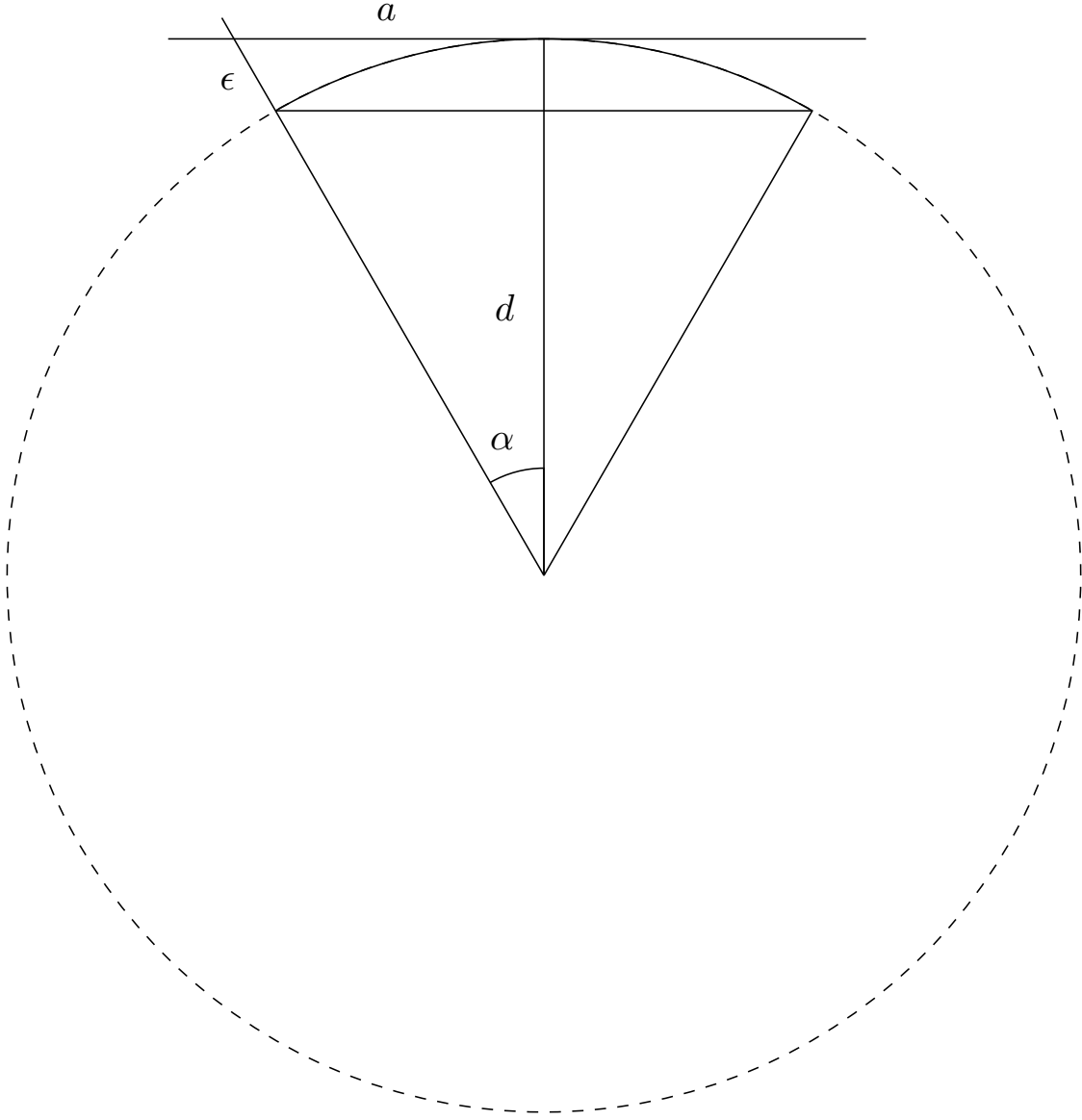


Figura 5: Errore nella stima della distanza con una linearizzazione a tratti della circonferenza

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. 2. 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%.

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

Sotto tali ipotesi possiamo quindi calcolare la distanza di un oggetto come mostrato in Eq. 3

$$object\ distance(m) = \frac{f(m) \times real\ width(m) \times image\ width(pixels)}{object\ width(pixels) \times sensor\ width(m)} \quad (3)$$

Poiché stiamo utilizzando un simulatore non è nota la larghezza del sensore da utilizzare per l'eq. 3. 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. 4. Abbiamo così stimato le dimensioni del sensore virtuale da utilizzare nei calcoli successivi.

$$sensor\ width(m) = \frac{f(m) \times real\ width(m) \times image\ width(pixels)}{object\ width(pixels) \times object\ distance(m)} \quad (4)$$

10.3 Modello probabilistico

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, come si evince dalla fig. 6.

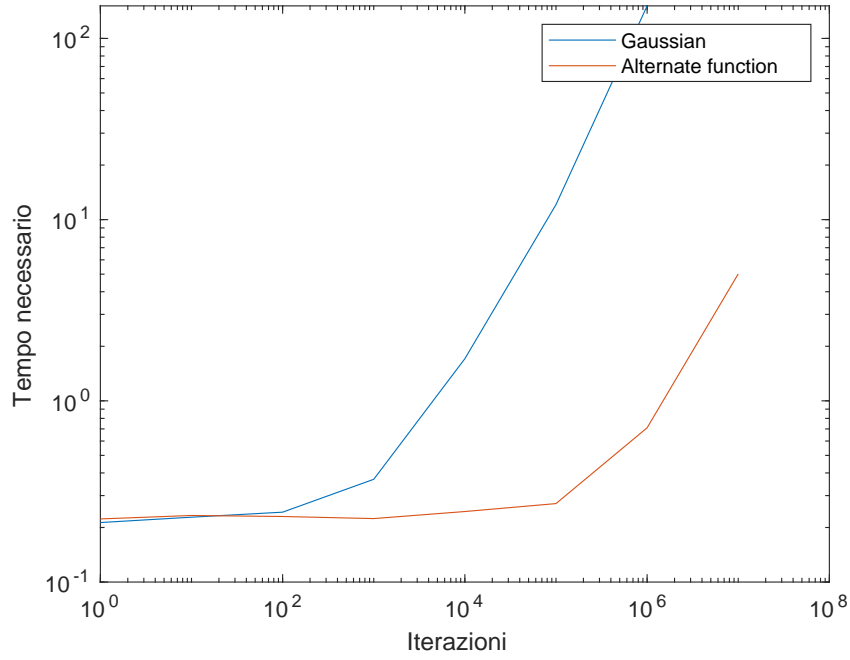


Figura 6: Benchmark funzione densità di probabilità (probability density function, funzione)

Al fine di ottenere un'approssimazione di una gaussiana applicabile nel nostro scenario, è stato utilizzato un funzionale ispirato al filtro di Butterworth. Un'approssimazione di largo uso è la distribuzione triangolare. Quest'ultima, tuttavia, presenta delle caratteristiche non ottimali 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 vogliamo infatti che la probabilità di trovare una persona in quel punto scenda e rimanga a zero.

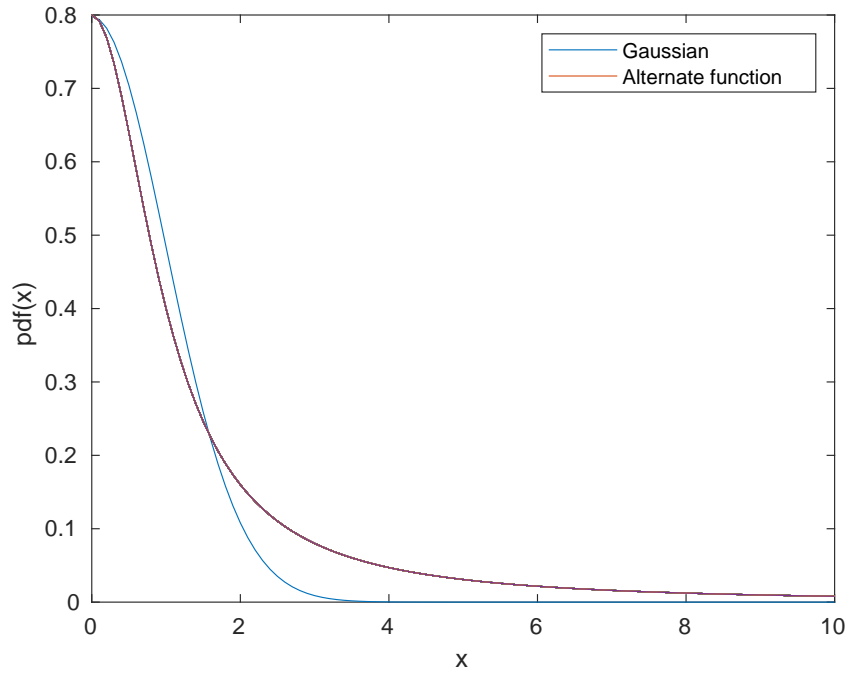


Figura 7: Forma pdf

Al fine di individuare gli eventuali cluster abbiamo utilizzato un approccio derivante dall'elaborazione delle immagini. In primo luogo abbiamo utilizzato [14], 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. Dall'immagine binaria restituita in output da otsu estraiamo i contorni con un metodo fornito da OpenCV per utilizzare l'algoritmo in [15].

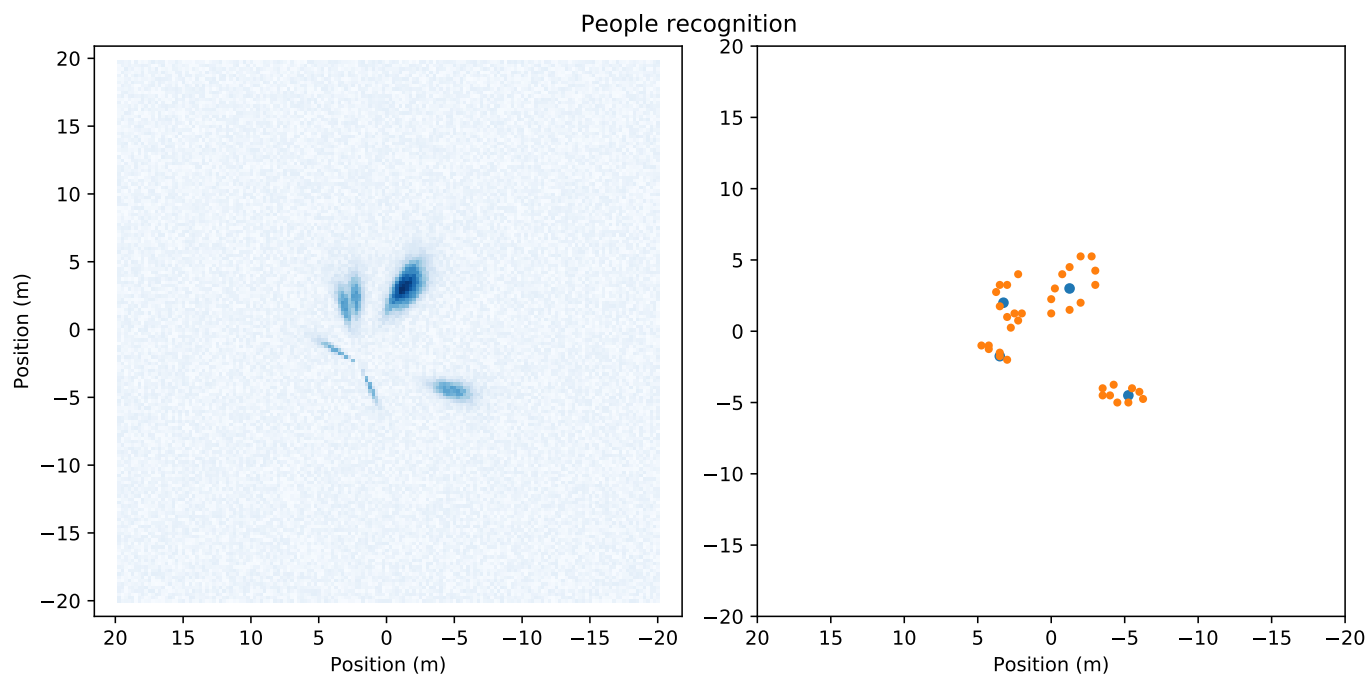


Figura 8: Segmentazione immagine

11 Pianificazione del moto

11.1 Modalità di movimento

11.1.1 Exploration mode

11.1.2 Bug mode

11.1.3 Campi di potenziale

11.2 Pianificazione

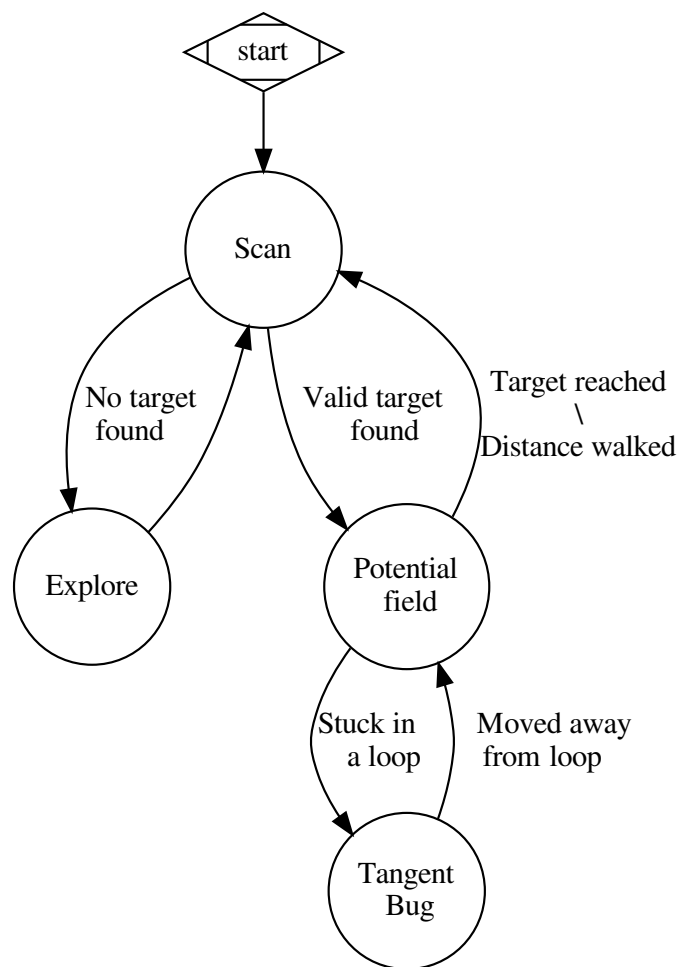


Figura 9: Automa a stati finito (Finite state automata)

Riferimenti bibliografici

- [1] Hannah Devlin and Kate Lyons. Far side of the moon: China's chang'e 4 probe makes historic touchdown, Jan 2019.
- [2] Webots. Webots user guide: Tutorial 8: Using ros.
- [3] Ros documentation from ros.org.
- [4] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
- [5] A Rosebrock. Imutils.
- [6] 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.
- [7] 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. *Nature*, 585(7825):357–362, September 2020.
- [8] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.
- [9] math - mathematical functions.
- [10] Webots. Webots user guide: Pal robotics' tiago iron.
- [11] Tiago datasheet.
- [12] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.
- [13] 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.
- [14] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.
- [15] 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.