SAPIENZA
UNIVERSITÀ DI ROMA

Applicazione SLAM con Orazio e AprilTag: rilevamento di landmark per la mappatura e la localizzazione nell'ambiente

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica

Corso di Laurea in Ingegneria Informatica e Automatica

Candidato

Lorenzo Nicoletti
Matricola 1797464

Relatore

Prof. Giorgio Grisetti

Anno Accademico 2019/2020

Tesi non ancora discussa

Applicazione SLAM con Orazio e AprilTag: rilevamento di landmark per la mappatura e la localizzazione nell'ambiente
Tesi di Laurea. Sapienza – Università di Roma

Questa tesi è stata composta con LaTeX e la classe Sapthesis.

Versione: 12 settembre 2020

Email dell'autore: nicoletti.1797464@studenti.uniroma1.it

A mia madre,
mio padre e mio fratello

Sommario

Questo documento costituisce l'elaborato scritto proposto dal candidato per la prova finale del Corso di Laurea in Ingegneria Informatica e Automatica. Al presente va affiancato il codice sviluppato dallo studente a conclusione del percorso triennale di studi.

Ringraziamenti

Un sentito ringraziamento al mio relatore di tesi, il Prof. Giorgio Grisetti, per il sostegno e il supporto continuo e al tutor del corso di Sistemi Operativi, Irvin Aloise, per la disponibilità e i preziosi consigli.

Indice

Capitolo 1

Introduzione

La presente tesi sperimentale propone un possibile approccio risolutivo per uno dei problemi più diffusi nell'ambito dell'intelligenza artificiale e della robotica, "Simultaneous Localization And Mapping" o più comunemente "SLAM".

Per SLAM si intende il processo per cui un robot si muove in un ambiente sconosciuto, costruisce la mappa di tale ambiente ed è capace di localizzarsi all'interno di essa, coniugando dunque due tra i tre principali macro-ambiti di studio del settore, il Mapping e la Localization.

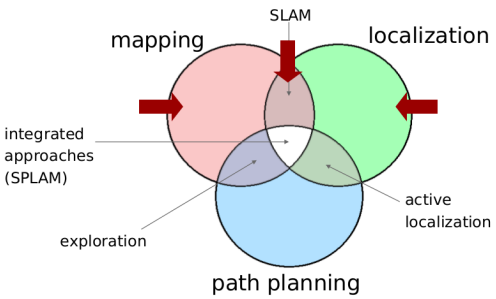


Figura 1.1. Area di studio SLAM

L'esperimento che si vuole proporre prevede l'ausilio di una base mobile robotica e di sensori montati su questa per poter effettuare misurazioni nello spazio e ricavarne informazioni utili alla risoluzione del problema prefissato. In particolare, la piattaforma adottata è MARRTino, completamente sviluppata nei laboratori de 'La Sapienza'.

Il sensore richiesto è la camera, in grado di riprendere il contesto d'azione del robot, modificata in modo tale da poter riconoscere particolari landmark collocati nell'ambiente, stimandone la posizione rispetto alla base mobile. Segue di conseguenza un'integrazione con la libreria AprilTag che si occupa dello sviluppo e dei possibili scopi applicativi di questi landmark. L'obiettivo da conseguire è ricreare il più fedelmente possibile l'ambito in cui l'agente robotico lavora, sviluppando un sistema robusto e coinvolgendo diverse nozioni studiate nel corso del percorso formativo svolto dallo studente.

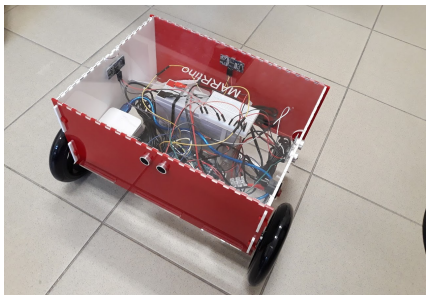


Figura 1.2. Esempio di piattaforma robotica MARRTino

Capitolo 2

Related work

Il lavoro propedeutico alla scrittura del codice ha richiesto una analisi accurata e una comprensione approfondita dei diversi stub di programmazione e delle librerie utilizzate nel corso dell'esperimento, nonché uno studio basilare delle nozioni teoriche riguardanti le trasformazioni geometriche nel piano e nello spazio e il cambio di coordinate tra diversi sistemi di riferimento. È stato, inoltre, necessario un approfondimento sull'utilizzo dei dispositivi nel sistema operativo Ubuntu, in modo da garantirne l'accesso e la loro seguente manipolazione. In particolare, il dispositivo utilizzato è la camera, in grado di fornire uno stream video in tempo reale dell'ambiente circostante. Nei prossimi paragrafi segue una analisi accurata di ognuno di questi componenti che sono stati coinvolti durante la procedura sperimentale della tesi.

2.1 MARRTino e Orazio

MARRTino è una semplice base mobile costruita per scopi didattici, dotata di sensori e attuatori, capace di interagire con l'ambiente circostante e ricavare informazioni da esso.

La gestione della piattaforma avviene tramite una scheda ATMega2560 collegata tramite una porta USB al PC host. La board, regolando il segnale PWM[1], controlla l'H-Bridge, dispositivo che fornisce ai motori la corrente e il voltaggio necessari al loro movimento. Di conseguenza, gli encoder destro e sinistro produrranno un codice digitale variabile a seconda della posizione angolare dei loro assi rotanti. Il robot comunica con il PC tramite l'EIA RS-232, un'interfaccia seriale a bassa velocità di trasmissione per lo scambio di dati tra dispositivi digitali.

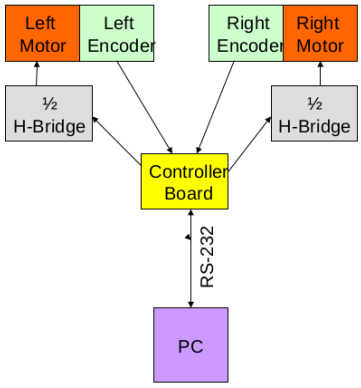


Figura 2.1. Componenti di MARRTino

[1] Acronimo di Pulse-Width Modulation, è un tipo di modulazione digitale che permette di ottenere una tensione media variabile dipendente dalla durata dell'impulso di ingresso e dell'intero periodo (duty cycle).

Orazio è il firmware caricato sulla scheda ATMega2560 utilizzata e che si occupa della gestione della piattaforma mobile nella sua interezza[2]. È un'architettura stratificata costituita da diversi sottosistemi che comunicano tra loro. Ognuno di questi moduli è in esecuzione sulla board di controllo e si occupa di un determinato compito all'interno dell'ambiente di Orazio. I sottosistemi presenti sono:

- System, si occupa della comunicazione che avviene tramite seriale e implementa un watchdog che interviene in caso di anomalie come misura di sicurezza;
- Joints, per il controllo dei quattro giunti di cui dispone la base mobile utilizzata;
- Drive, modulo che controlla la piattaforma e che gestisce le informazioni riguardanti le velocità rotazionale e traslazionale della base con la sua odometria nel tempo;
- Sonar, per la gestione dei sonar montati sulla piattaforma;
- Servo, che si occupa del controllo dei servo-scanner con le relative informazioni acquisite;
- IMU, interfaccia per la gestione dell'Unità di Misura Inerziale che monitora la dinamica del mezzo durante il movimento nello spazio.

Ogni sottosistema:

- ha uno stato;
- ha dei parametri memorizzati nella EEPROM, la memoria della scheda ATMega2560;
- può ricevere comandi;
- può essere abilitato o meno;
- può inviare aggiornamenti.

La tesi sperimentale, tuttavia, non ha previsto l'utilizzo di tutti questi moduli. Sono stati impiegati difatti solo i sottosistemi 'System' e 'Drive' per poter comunicare con la scheda, inviare comandi per impostare le velocità desiderate e ricevere aggiornamenti per quanto riguarda l'odometria nel corso dell'esperimento.

Orazio prevede l'invio e la ricezione di packets durante la comunicazione tra l'host e il controller, ciascuno dei quali è caratterizzato da un identificativo unico e contiene diversi tipi di informazione a seconda della propria categoria di appartenenza. In particolare, pacchetti inviati lato PC riguardano il settaggio di alcuni parametri come le velocità rotazionale e traslazionale desiderate, mentre lato robot si evidenziano packets contenenti lo stato dei sottosistemi come l'odometria corrente o la velocità misurata.

Tutte queste informazioni sono facilmente accessibili da un programma client (implementato in linguaggio C) che salva in determinate variabili il contenuto dell'ultima occorrenza di ogni tipologia di pacchetto ricevuto.

[2]Il codice sorgente completo dello stub di Orazio è disponibile presso la repository GitLab https://gitlab.com/srrg-software/srrg2_orazio.

2.2 Camera

In ambiente Linux, i dispositivi sono visti come file[3] e pertanto sono accessibili e manipolabili con l'ausilio di un file descriptor (fd). Tuttavia, alcuni tra i dispositivi collegabili necessitano per il loro corretto funzionamento il trasferimento di un'enorme quantità di dati e di conseguenza un ingente numero di accessi a memoria, i quali sono estremamente onerosi e costituiscono spesso il collo di bottiglia prestazionale del sistema.

Per ovviare a questa situazione, è stato introdotto il meccanismo del Direct Memory Access (DMA). Costituisce infatti un'alternativa al protocollo generalmente usato per gestire gli accessi a memoria, il Programmed I/O (PIO), il quale richiede l'intervento della CPU per trasferire un byte alla volta. L'overhead introdotto risulterebbe ingestibile per la macchina, impegnando inutilmente la CPU. L'ausilio del DMA permette invece alle periferiche di accedere direttamente in memoria per scambiare dati, in lettura e/o scrittura, senza coinvolgere l'unità di controllo per ogni singolo byte trasferito ma generando un solo interrupt per comunicare l'avvenuto trasferimento dell'intero blocco dati richiesto.

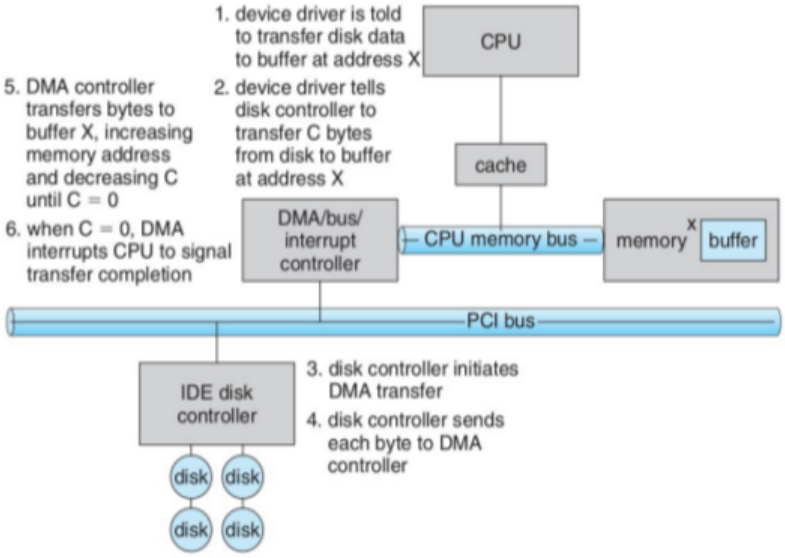


Figura 2.2. Meccanismo del DMA per gli accessi a memoria

Tra i dispositivi che fanno ampio uso del DMA della CPU vi è la camera. L'interazione con questo determinato device non è immediata come in altri casi, in quanto presenta un driver più complicato da gestire. Per controllare correttamente tale periferica, Linux offre una particolare API denominata "Video4Linux" (V4L[4]), utilizzata per la cattura di immagini, la registrazione audio e l'acquisizione di filmati.

Prima di poter essere pienamente operativa, la camera deve essere in grado di comprendere il proprio funzionamento e le capacità che può offrire come dispositivo.

[3] In linea con il "mantra" di Linux che afferma: "Everything is a file".

[4] La versione più aggiornata uscita in seguito è "Video4Linux2" (V4L2). Ogni riferimento nel seguito della relazione alla suddetta libreria si riferirà a questa seconda versione.

Risulta, di conseguenza, estremamente utile la syscall 'ioctl', particolare chiamata a sistema che, passati come argomenti un file descriptor, un codice di request ed eventuali parametri correlati, esegue per il dispositivo descritto da quel determinato fd la specifica richiesta domandata[5]. Utilizzata per interagire con il driver della periferica, ioctl permette di ricavare e impostare i parametri di quel determinato device.

La libreria V4L offre molteplici codici di richiesta per poter manipolare la camera, una volta aperta. Tra questi, le request effettuate nella tesi per poter usufruire del servizio di acquisizione delle immagini sono state:

- VIDIOC_QUERYCAP, per richiedere le capacità del dispositivo e in particolare se è in grado di supportare la cattura e lo streaming video;
- VIDIOC_CROPCAP e VIDIOC_S_CROP, per sapere se la camera supporta il cropping (il ritaglio) delle immagini;
- VIDIOC_S_FMT, in modo da impostare la larghezza e l'altezza dell'immagine espresse in pixels (rispettivamente 640 e 480) con relativo formato (jpeg);
- VIDIOC_REQBUFS, per allocare dei buffer di memoria per lo scambio dati con DMA e per inizializzare particolari strutture interne del driver;
- VIDIOC_QUERYBUF, che richiede lo stato dei buffer precedentemente allocati; in seguito a questa richiesta, i buffer sono mappati in memoria tramite l'invocazione della funzione 'mmap';
- VIDIOC_QBUF e VIDIOC_DQBUF, operazioni complementari per inserire o rimuovere un buffer dalla coda del device driver in modo da renderlo accessibile o meno;
- VIDIOC_STREAMON e VIDIOC_STREAMOFF, per avviare o terminare il processo di streaming.

L'invocazione di ioctl per ognuna di queste request permette di utilizzare la camera in modo adeguato, sostenendo il notevole scambio di dati con il dispositivo. Inoltre, l'utilizzo dei buffer, precedentemente mappati in memoria ed accessibili con rapide operazioni di DMA, consente di ricreare uno stream video in tempo reale dell'ambiente circostante inquadrato dalla camera.

2.3 OpenCV

OpenCV è una libreria software per la visione artificiale in tempo reale[6]. Il linguaggio di programmazione principalmente utilizzato per il suo sviluppo è il C++, ma è possibile interfacciarsi anche attraverso il C, Python e Java. L'architettura di OpenCV prevede la cooperazione di diversi sottosistemi o moduli che offrono all'utente molteplici servizi.

Tra questi moduli, nel corso dell'esperimento, sono stati utilizzati principalmente:

[5]Per ulteriori approfondimenti sul funzionamento della syscall ioctl si consulti la sezione due del Linux Programmer's Manual, accessibile tramite il comando "man 2 ioctl" da terminale.

[6]Per ulteriori approfondimenti sulla libreria si rimanda al sito ufficiale https://opencv.org.

- HighGUI, interfaccia per la manipolazione di finestre dotate di svariati tools grafici per la riproduzione video;
- ImgProc, per il processamento e l'elaborazione di immagini.

Precedentemente è stato esposto come il dispositivo della camera sfruttasse dei buffer allocati in memoria e li accedesse con il meccanismo del DMA. All'interno di questi buffer sono presenti tutti i dati necessari per una loro decodifica in immagini; in particolare, tramite le request di VIDIOC_QBUF e VIDIOC_DQBUF, è possibile ricreare in tempo reale uno streaming video della ripresa della camera, sfruttando le strutture dati offerte da OpenCV. Dal momento che l'intero sistema è stato sviluppato in linguaggio C, sono risultati particolarmente utili per questo scopo due costrutti di cui dispone la libreria:

- CvMat, una matrice Dense multicanale inizializzata con i dati presenti nei buffer;
- IplImage, che decodifica la precedente CvMat ricreando l'immagine originaria.

L'utilizzo di entrambi questi costrutti permette all'utente di ricreare in un unico flusso video il contesto d'azione che la camera sta riprendendo in quel determinato momento, decodificando e ricreando ciascun frame immagine per immagine.

2.4 AprilTag

AprilTag è un sistema visivo utilizzabile per una grande varietà di task tra cui realtà aumentata, robotica e calibrazione della camera. La libreria è implementata completamente in linguaggio C ed è progettata per essere facilmente inclusa in altre applicazioni[7]. Si basa sull'utilizzo di "tag" o landmark, i quali, simili a QR Code, una volta posizionati nello spazio circostante, sono individuabili dal software di rilevamento di cui dispone la libreria per poter ottenere una loro precisa identificazione, orientazione e posizione 3D.

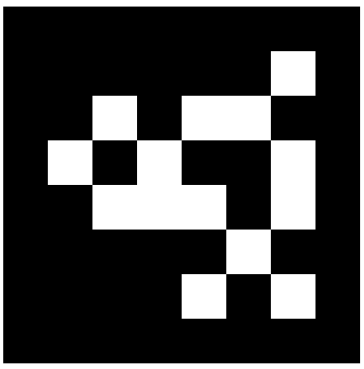


Figura 2.3. Esempio di landmark della libreria AprilTag

Il software risulta essere, inoltre, efficiente anche in condizioni di sovra illuminazione e in generale in situazioni avverse ad un immediato riconoscimento dei tag.

Ogni tag è disegnato per poter codificare un data payload molto piccolo (solitamente tra i 4 e i 12 bits), caratteristica che ne permette un rapido riconoscimento anche a distanze relativamente elevate. Inoltre, sono progettati per fornire una loro localizzazione estremamente accurata, calcolandone la posizione nello spazio rispetto alla camera utilizzata per il rilevamento.

[7] Il codice open source è disponibile presso la repository GitHub https://github.com/AprilRobotics/apriltag.

AprilTag offre all'utente sei diverse famiglie di tag, ciascuna delle quali dispone di diverse caratteristiche. La tesi si concentra esclusivamente sul riconoscimento, l'identificazione e la localizzazione dei tag appartenenti alla famiglia "36h11", tra i più semplici ma al tempo stesso tra i più diffusi per la varietà di applicazioni in cui possono essere utilizzati.

Uno dei principali vantaggi di cui la libreria dispone, e che è stato ampiamento sfruttato durante la scrittura del codice inerente, è l'integrazione che AprilTag offre con OpenCV. In particolare, a partire da una struttura IplImage, discussa precedentemente e che ricrea l'immagine di partenza, risulta immediato poter convertire il suo contenuto in una seconda struttura dati introdotta da AprilTag, la quale, richiedendo esclusivamente la larghezza, l'altezza e il puntatore ai dati dell'immagine, facilita il detector nel riconoscimento di eventuali tag contenuti proprio in quell'immagine.

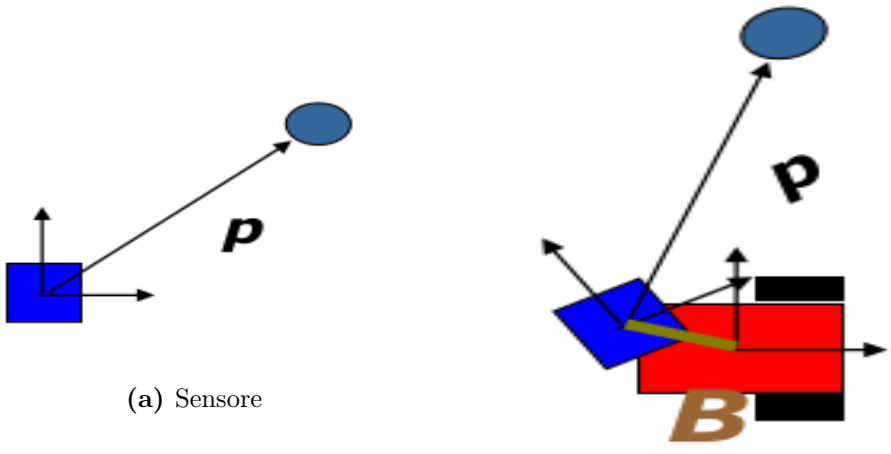
Tutti i rilevamenti, infine, sono collocati all'interno di un vettore di detection per rendere le seguenti operazioni di stima della posizione nello spazio molto più semplici e immediate.

2.5 Pose estimation

2.5.1 Trasformazioni nel piano e nello spazio

Una precisa stima della posizione del tag e del robot nel corso dell'esperimento rappresenta uno degli aspetti cruciali per ottenere dei risultati realistici.

Per ognuna di queste informazioni è richiesta la definizione di un sistema di riferimento (o "frame") globale che resti costante e invariato: tutte le posizioni saranno dunque espresse secondo le coordinate dettate da questo "world frame". Tuttavia, ottenere tali coordinate non è sempre immediato ma richiede delle considerazioni preliminari, ovvero un'analisi dei vari sistemi di riferimento coinvolti e le trasformazioni geometriche presenti tra questi.



(a) Sensore

(b) Robot

Figura 2.4. Rilevamento del landmark in diversi sistemi di riferimento

Infatti, nel momento in cui un sensore collocato su una base mobile robotica percepisce un determinato oggetto nello spazio, egli ne stima la posizione nel proprio sistema di riferimento. Tale posizione geometricamente è un vettore p di dimensione $m \times 1$, con m che definisce la cardinalità dello spazio vettoriale considerato.

Il sistema di riferimento del sensore però potrebbe non coincidere con quello della piattaforma robotica su cui è posizionato, motivo per cui le coordinate del vettore p non rappresentano una posizione valida dal punto di vista del robot.

È necessario dunque applicare una trasformazione geometrica per ricavare, a partire da p, le coordinate dell'oggetto individuato nel sistema di riferimento della base mobile.

Tale operazione è estremamente semplice e non richiede particolare complessità di calcolo. Preliminarmente, è necessario determinare la matrice di trasformazione B tra il sensore e il robot. Tale matrice è quadrata di dimensione $(m+1) \times (m+1)$ e racchiude al suo interno le informazioni sulla rotazione e sulla traslazione presente tra i due sistemi di riferimento coinvolti. In particolare, la struttura di B è del tipo:

$$B = \begin{pmatrix} R & t \\ 0 & 1 \end{pmatrix} \tag{2.1}$$

dove R e t rappresentano rispettivamente la matrice di rotazione $m \times m$ e il vettore di traslazione $m \times 1$. La presenza dell'ultima riga di B, $\begin{pmatrix} 0 & 1 \end{pmatrix}$, determina la struttura quadrata della matrice e permette l'applicazione della cosiddetta trasformazione affine. Inoltre, anche la matrice R è caratterizzata da una propria struttura a blocchi che, dato l'angolo di rotazione θ tra due frame, si configura come:

- nel piano:

$$R(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \tag{2.2}$$

- nello spazio:

  1. sull'asse delle x,

$$R(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix} \tag{2.3}$$

  2. sull'asse delle y,

$$R(\theta) = \begin{pmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{pmatrix} \tag{2.4}$$

  3. sull'asse delle z,

$$R(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.5}$$

Dato dunque il vettore p, questi è leggermente modificato determinando un nuovo vettore $\tilde{p}$ così composto:

$$\tilde{p} = \begin{pmatrix} p \\ 1 \end{pmatrix} \tag{2.6}$$

con '1' definito numero affine.

Eseguire la trasformazione necessaria richiede un semplice prodotto matriciale tra B e $\tilde{p}$, da cui si ricava quindi la posizione dell'oggetto nel sistema di riferimento del robot definita come:

$$p' = B \times \tilde{p} \tag{2.7}$$

Alternativamente, è possibile eseguire la medesima trasformazione, evitando di utilizzare i numeri affini, nel seguente modo:

$$p' = R \times p + t \tag{2.8}$$

a partire dunque dalla posizione p espressa nel sistema di riferimento del sensore e determinando la matrice di rotazione e il vettore di traslazione con il frame del robot, è possibile stimare la nuova posizione del rilevamento rispetto alla base mobile utilizzata.

A questo punto, in seguito alla definizione del sistema di riferimento del mondo ed eseguendo un ragionamento analogo al precedente, si ricava la posizione globale dell'oggetto.

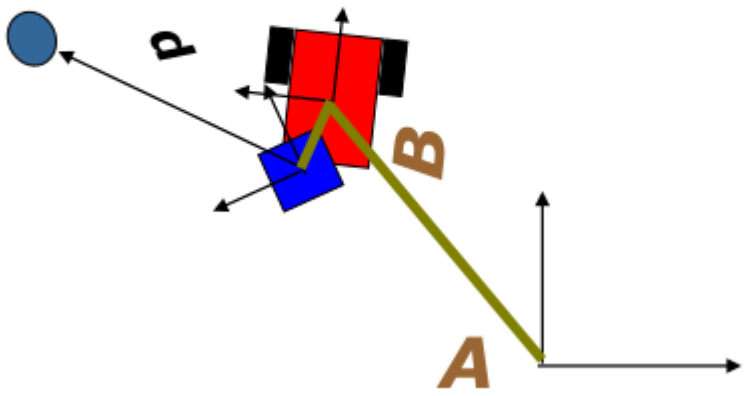


Figura 2.5. Rilevamento nel sistema di riferimento del mondo

Similmente, sarà presente tra il robot e il mondo una nuova matrice di trasformazione A, la cui struttura è analoga a quella di B, per cui le coordinate del punto rilevato nell'ambiente espresse nel sistema di riferimento del mondo sono determinate dall'equazione:

$$p'' = A \times p' = A \times B \times \tilde{p} \tag{2.9}$$

La concatenazione delle trasformazioni precedenti rende il calcolo della posizione finale estremamente semplice. Considerando infatti che solitamente la cardinalità

dello spazio vettoriale m è molto piccola, 2 nel piano e 3 nello spazio, le dimensioni delle matrici A e B sono limitate, di conseguenza la computazione è veloce e con bassa probabilità di errore, conseguendo quasi sempre una localizzazione del target estremamente accurata. Ovviamente è possibile anche effettuare le varie operazioni inverse per ciascuna delle formule analizzate: conoscendo ad esempio la posizione p' del tag, a partire dall'equazione (2.8), si ricava il vettore p tramite:

$$p = R^{-1} \times p' - R^{-1} \times t \tag{2.10}$$

dal momento che R è una matrice ortonormale, si implica che:

$$R^{-1} = R^T \tag{2.11}$$

in modo da agevolare il calcolo richiesto.

2.5.2 Applicazione della teoria nel contesto dell'esperimento

A fronte delle osservazioni sviluppate finora, si è evidenziato come ciascun sensore possegga un proprio sistema di riferimento nel quale esprime la posizione degli oggetti che rileva durante la sua esecuzione.

Di conseguenza, anche il sensore della camera avrà un proprio frame tridimensionale. Una prima stima degli AprilTag collocati nell'ambiente è dunque calcolata in questo sistema di riferimento: il software di detection offerto dalla libreria provvederà alla stima della posizione dei tag calcolando un vettore di dimensione 3×1 contenente le coordinate x, y e z del landmark individuato.

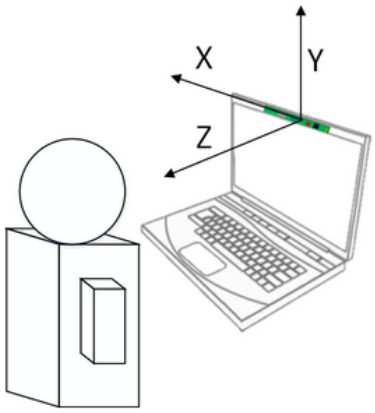


Figura 2.6. Oggetto rilevato dal sensore della camera

Come esposto precedentemente, tale frame non coincide con il sistema di riferimento della base robotica utilizzata durante l'esperimento.

È necessario quindi applicare una trasformazione geometrica nello spazio per poter convertire le coordinate del tag dal sistema della camera a quello del robot, individuando la matrice di rotazione R e il vettore di traslazione t tra questi due frame per poter calcolare in seguito la posizione p' tramite l'applicazione della formula (2.8).

Per quanto riguarda R, esaminando l'orientazione degli assi cartesiani si osserva come per comporre correttamente tale matrice sia necessario effettuare una doppia rotazione di un angolo θ pari a $-\pi/2$ sull'asse delle z e poi sull'asse delle y. La

formula seguente ne riassume lo svolgimento:

$$R = R_{\mathrm{y}}(-\frac{\pi}{2}) \times R_{\mathrm{z}}(-\frac{\pi}{2}) = \begin{pmatrix} \cos(-\frac{\pi}{2}) & 0 & -\sin(-\frac{\pi}{2}) \\ 0 & 1 & 0 \\ \sin(-\frac{\pi}{2}) & 0 & \cos(-\frac{\pi}{2}) \end{pmatrix} \times$$
$$\times \begin{pmatrix} \cos(-\frac{\pi}{2}) & -\sin(-\frac{\pi}{2}) & 0 \\ \sin(-\frac{\pi}{2}) & \cos(-\frac{\pi}{2}) & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \tag{2.12}$$
$$= \begin{pmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{pmatrix}$$

Il vettore di traslazione t è calcolato invece misurando la distanza che intercorre tra le due origini dei sistemi di riferimento della camera e del robot. Il vettore risultante sarà quindi:

$$t = \begin{pmatrix} t_{\mathrm{x}} \\ t_{\mathrm{y}} \\ t_{\mathrm{z}} \end{pmatrix} \tag{2.13}$$

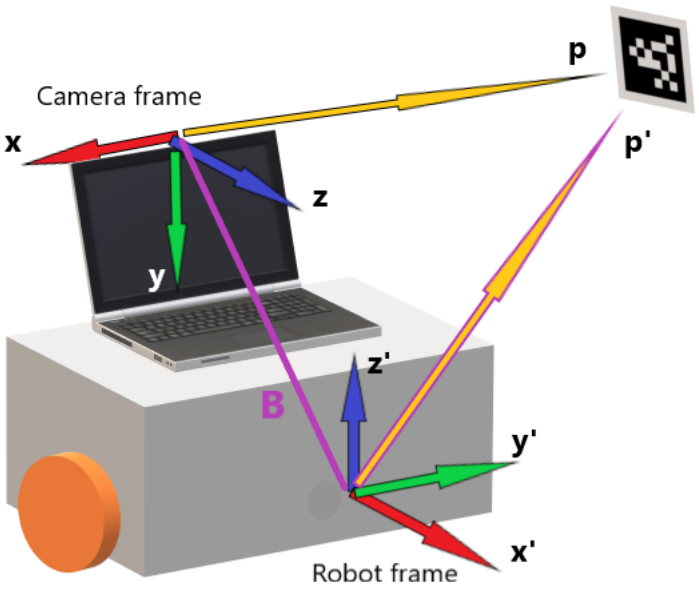


Figura 2.7. Ricostruzione modellistica della piattaforma utilizzata con evidenziazione dei sistemi di riferimento coinvolti

Applicando infine la (2.8), si stima la posizione del tag rispetto alla base mobile robotica.

Similmente, calcolare ora la posizione globale del tag segue un analogo ragionamento. Mentre precedentemente la matrice R e il vettore t tra camera e robot erano costanti in quanto i due sistemi di riferimento erano fissi nonostante la piattaforma fosse in movimento, ora questi due elementi variano con lo spostamento della base, richiedendo quindi un loro nuovo calcolo ogni qualvolta venga richiesta la localizzazione del tag nel mondo. Questa operazione, tuttavia, è estremamente immediata sfruttando le informazioni contenute nei packets di Orazio, dai quali è possibile ricondursi all'angolo θ e all'odometra del robot per il calcolo rispettivamente della rotazione e della traslazione rispetto al world frame durante l'esplorazione dell'ambiente. Applicare nuovamente la (2.8) permette la stima della posizione globale del tag.

Il calcolo della trasformata tra due posizioni consecutive del robot richiede invece un'ulteriore osservazione.

Dal momento che si vuole conoscere la posa corrente del robot rispetto al precedente frame, è necessario ora applicare l'operazione inversa a quella adottata finora, descritta dalla formula (2.10) combinata con la (2.11). Ciò implica la conoscenza delle due posizioni, precedente e corrente, della piattaforma e degli angoli θ' e θ" creatisi con il world frame. Come attuato in precedenza, è sufficiente consultare il server di Orazio per ottenere i pacchetti contenenti tali informazioni. A questo punto risalire alla trasformazione richiede solo semplici calcoli matematici tra matrici.

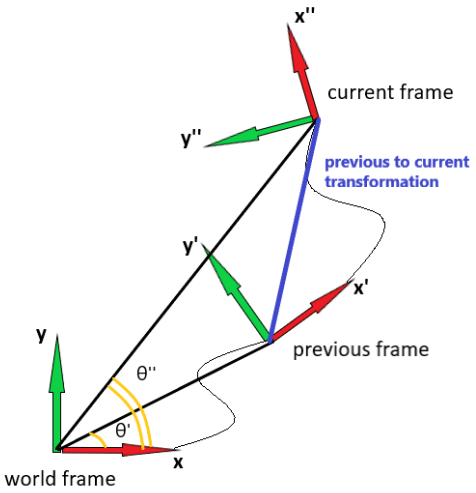


Figura 2.8. Ricostruzione modellistica del movimento del robot con seguente spostamento del proprio sistema di riferimento

Si osservi come ora la cardinalità dello spazio vettoriale si sia ridotta a 2: non è richiesta per gli scopi di questa tesi una considerazione del terzo asse cartesiano che misura l'altezza del robot e dei tag in quanto tutte le operazioni seguenti saranno effettuate unicamente sul piano, passando quindi da uno spazio tridimensionale a uno bidimensionale.

2.6 g2o

"g2o" è un framework per l'ottimizzazione di grafi che minimizza l'errore presente nelle varie misurazioni effettuate nel tempo[8]. Utilizzato nella robotica e in computer-

[8]Il codice sorgente è disponibile presso la repository GitHub https://github.com/OpenSLAM-org/openslam-g2o.

vision, offre soluzioni per diverse varianti di applicazioni SLAM. L'obiettivo generale è quello di trovare la configurazione ottima di parametri o variabili di stato che più esaurientemente spieghi l'insieme di misurazioni affette da rumore e, di conseguenza, da errore.

I dati che g2o processa in input sono grafi e quindi strutture formate da vertici e archi. In questo contesto, i vertici rappresentano la posizione del robot e dei tag espresse nel sistema di riferimento del mondo, pertanto tale posizione è anche globale. Gli archi invece possono collegare due posizioni consecutive del robot durante lo spostamento o in alternativa la posizione del robot e dei tag che sono rilevati in un determinato istante.

Per visualizzare tale struttura è disponibile un eseguibile denominato "g2o_viewer" che, dato in input un file con estensione ".g2o", costruisce il grafo relativo.

Un file g2o racchiude al suo interno tutte le informazioni riguardanti i vertici e gli archi del grafo corrispondente. Queste sono denominate come:

- VERTEX_XY, seguito da un identificativo (id) e da due coordinate nel piano, specifica la posizione globale di un tag descritto da quel determinato id;
- VERTEX_SE2, descrive la posizione globale del robot, unica per id e costituita dalle due coordinate nel piano e dall'angolo di rotazione creatosi rispetto al world frame;
- EDGE_SE2, arco tra due VERTEX_SE2 con relativa trasformazione tra i sistemi di riferimento coinvolti, seguito da alcuni valori di default;
- EDGE_SE2_XY, arco tra un VERTEX_SE2 e un VERTEX_XY che indica il rilevamento di un determinato tag effettuato in corrispondenza della posizione specificata, seguito da alcuni valori di default.

Un semplice esempio di file g2o con tutte le informazioni necessarie alla costruzione di un grafo è il seguente:

```
VERTEX_SE2 100 0.000000 0.000000 0.000000
VERTEX_SE2 101 -1.000000 2.000000 1.570796
EDGE_SE2 100 101 -1.000000 2.000000 1.570796 100 0 0 100 0 1000
VERTEX_XY 0 0.000000 5.000000
EDGE_SE2_XY 101 0 3.0000000 -1.000000 100 0 100
```

Figura 2.9. Esempio di semplice file g2o con informazioni su nodi e archi

Come si può osservare, contiene solo due posizioni del robot nello spazio e, in particolare, quando l'agente si trova nella seconda posizione rileva ad una certa distanza un landmark descritto dall'identificativo '0'. Utilizzando il visualizzatore grafico "g2o_viewer", il grafo corrispondente risulta essere semplicemente:

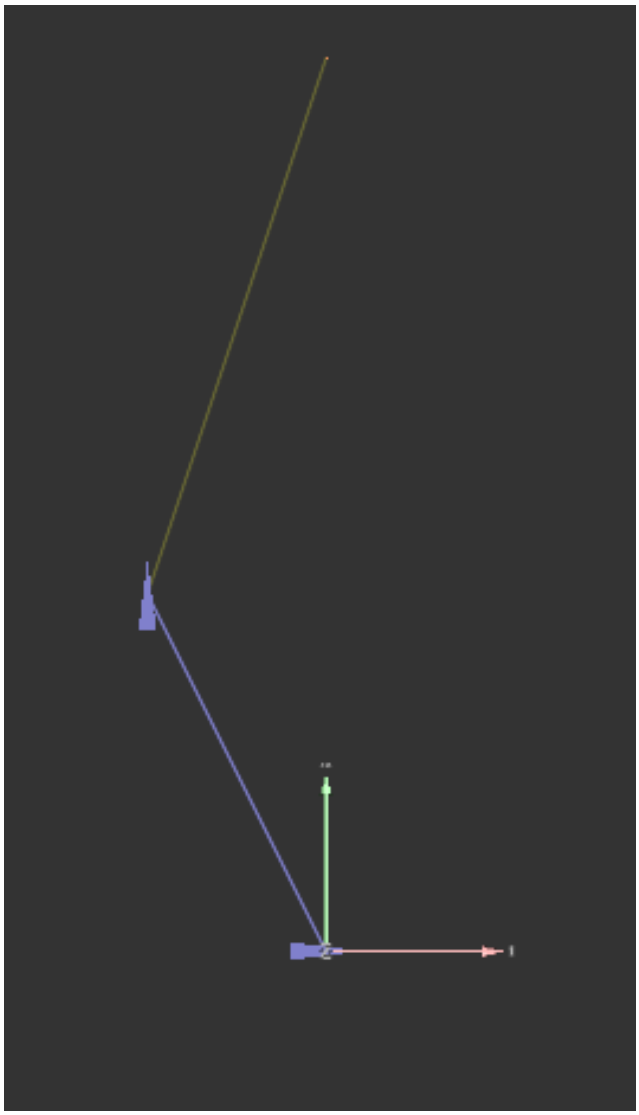


Figura 2.10. Grafo costruito a partire dal precedente file g2o illustrato

Capitolo 3

Metodo

L'approccio al problema affrontato vede la scomposizione dell'esperimento in due parti: la prima si occupa dell'acquisizione del dataset con le informazioni propedeutiche allo svolgimento della seconda, che consiste invece nel calcolo delle trasformazioni geometriche presenti tra i vari sistemi di riferimento coinvolti, con contestuale stesura del corrispondente file g2o. In particolare, la raccolta del dataset è effettuata dal programma client di Orazio che comunica con la base mobile utilizzata e che registra su un comune file di testo i dati sulla posizione del robot e dei tag nel tempo, mentre la loro seguente elaborazione è attuata da un parser che ricava le indicazioni per la stesura di un file g2o da cui si costruisce in seguito il relativo grafo.

Dal momento che una prima vera stima della posizione è effettuata dalla camera, è necessario preliminarmente individuare la configurazione ottima di tutti quei parametri che consentono una precisa localizzazione dei landmark, sottoponendo il device al processo di calibrazione. Successivamente, sarà possibile applicare il metodo elaborato per la risoluzione del problema che la tesi si è prefissato, eseguendo praticamente le nozioni esaminate finora.

3.1 Camera calibration

La calibrazione della camera è il processo di stima dei parametri intrinsechi ed estrinsechi del dispositivo: una configurazione di questi determina la precisione con la quale la periferica localizza nell'ambiente eventuali landmark rilevati. Risolve inoltre alcuni dei problemi presenti nella percezione di oggetti per opera di dispositivi digitali, come ad esempio la distorsione dell'immagine.

Fondamentalmente, la procedura termina con il calcolo della cosiddetta "camera matrix" la cui struttura è la seguente:

$$\begin{pmatrix} f_{\mathrm{x}} & 0 & c_{\mathrm{x}} \\ 0 & f_{\mathrm{y}} & c_{\mathrm{y}} \\ 0 & 0 & 1 \end{pmatrix} \tag{3.1}$$

Essa racchiude al suo interno i parametri che regolano la percezione del dispositivo, minimizzando le distorsioni e ricreando il più similmente possibile l'immagine originale.

I valori degli intrinsechi f_x, f_y, c_x e c_y differiscono da camera a camera, pertanto per garantire la corretta stima di una posizione è richiesto preventivamente sottoporre il dispositivo al processo di calibrazione.

I parametri f_x e f_y esprimono la lunghezza focale della camera e sono calcolati in pixels, mentre c_x e c_y costituiscono le coordinate del centro ottico, anch'esse espresse in pixels.

Solitamente, una buona calibrazione si evince dal fatto che i valori di questi ultimi due parametri siano circa pari alla metà della larghezza e dell'altezza del frame della camera. Ad esempio, il dispositivo utilizzato è caratterizzato da una risoluzione 640×480 e in seguito alla calibrazione le coordinate del centro ottico sono state stimate a 319.5×239.5, molto vicine dunque alle dimensioni ricercate.

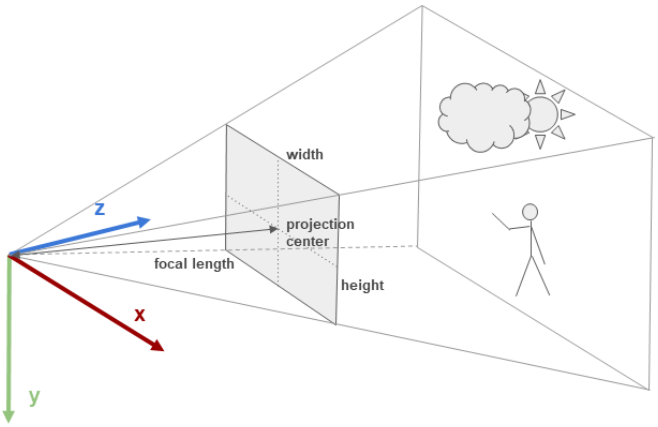
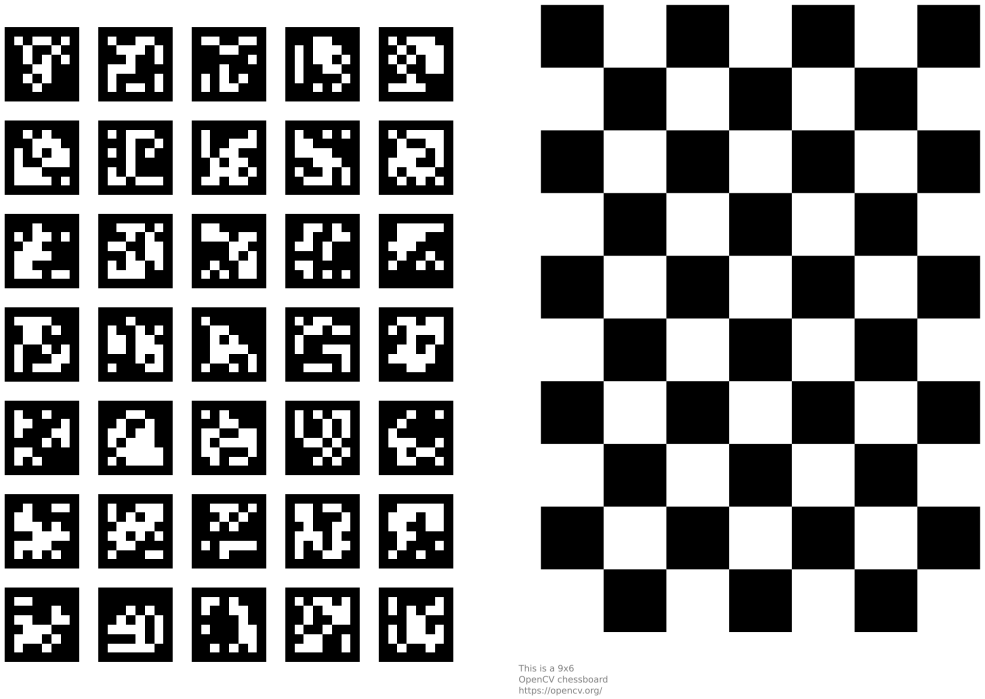


Figura 3.1. Percezione dell'ambiente da parte della camera e funzione dei parametri intrinsechi

La tesi ha previsto due diverse modalità per la calibrazione della camera che hanno condotto a risultati molto simili. Entrambe hanno implicato l'utilizzo di un pattern prefabbricato che, posizionato in diverse angolazioni nel campo visivo della periferica, ha permesso la stima dei parametri richiesti. Queste due modalità sono messe a disposizione direttamente dalle librerie grafiche adottate per il progetto, AprilTag e OpenCV. L'idea alla base di entrambe le procedure è sottoporre al device dei pattern creati appositamente per questi scopi, un mosaico di tag per AprilTag e una scacchiera per OpenCV, cercando di inquadrarli per coprire esaurientemente il campo visivo della camera per tutta la sua estensione, in modo che il dispositivo possa garantire ottime localizzazioni anche per quei tag che sono individuati ad esempio ai bordi dell'immagine o che sono caratterizzati da una determinata distorsione.

La definizione della matrice (3.1), e quindi il calcolo della lunghezza focale e del centro ottico della camera, permette al software di rilevamento dei tag di eseguire una loro stima realistica della posizione nel sistema di riferimento del device. Di conseguenza, la calibrazione risulta essere estremamente importante per una buona riuscita dell'esperimento e rappresenta il passo base per la risoluzione del problema da affrontare.



(a) Mosaico di AprilTag

(b) Chessboard di OpenCV

Figura 3.2. Pattern utilizzati per la calibrazione della camera

3.2 Acquisizione del dataset con Orazio

Lo stub di Orazio offre all'utente un semplice programma lato client per interagire con la piattaforma robotica adottata[9]. A partire da questo programma minimale sono stati rimossi tutti i moduli superflui ad eccezione dei sottosistemi 'System' e 'Drive', necessari per indicare la velocità della base e interrogare il robot per conoscere la sua odometria.

Per eseguire il primo task, viene utilizzato un secondo dispositivo in aggiunta a quello della camera che è il joystick. Tale device in Linux è estremamente semplice da manipolare e non richiede operazioni di 'ioctl' a differenza della camera: una volta aperto con una banale invocazione della funzione 'open', è possibile leggere sequenze di particolari strutture dati che contengono tutte le informazioni necessarie per indicare le velocità rotazionale e traslazionale desiderate[10].

Tali informazioni sono elaborate per poter essere racchiuse in un packets di Orazio di tipo DifferentialDriveControl: è sufficiente inizializzarne i campi per definire la velocità che la base mobile deve avere durante il movimento.

[9] Il programma è disponibile presso la repository GitLab di Orazio; in particolare al seguente link: https://gitlab.com/srrg-software/srrg2_orazio/-/blob/master/srrg2_orazio/src/host_test/orazio_client_test.c.

[10] Queste struct sono denominate "js_event" e sono utilizzabili in seguito all'inclusione dell'apposita libreria tramite #include <linux/joystick.h>.

In parallelo su un secondo thread d'esecuzione, Orazio si occupa anche di inizializzare la camera ed eseguire tutte le operazioni di I/O di cui questo dispositivo necessita. Successivamente, vi integra le librerie di AprilTag e OpenCV per avviare il software di riconoscimento dei tag presenti nel contesto d'azione. Con lo scopo di fornire un riscontro grafico di questi rilevamenti, all'interno del frame proiettato grazie ad OpenCV sono evidenziati tutti i tag ripresi e su questi è indicato anche l'identificativo, univoco per ciascun landamark individuato. Eseguendo a questo punto la funzione di stima della posizione offerta da AprilTag è possibile calcolare la posizione dei tag nel sistema di riferimento della camera, come di seguito mostrato:

```
typedef struct {
  PacketHeader header;
  float translational_velocity;
  float rotational_velocity;
} DifferentialDriveControlPacket;
```

Figura 3.3. Struttura di un Differential Drive Control packet di Orazio

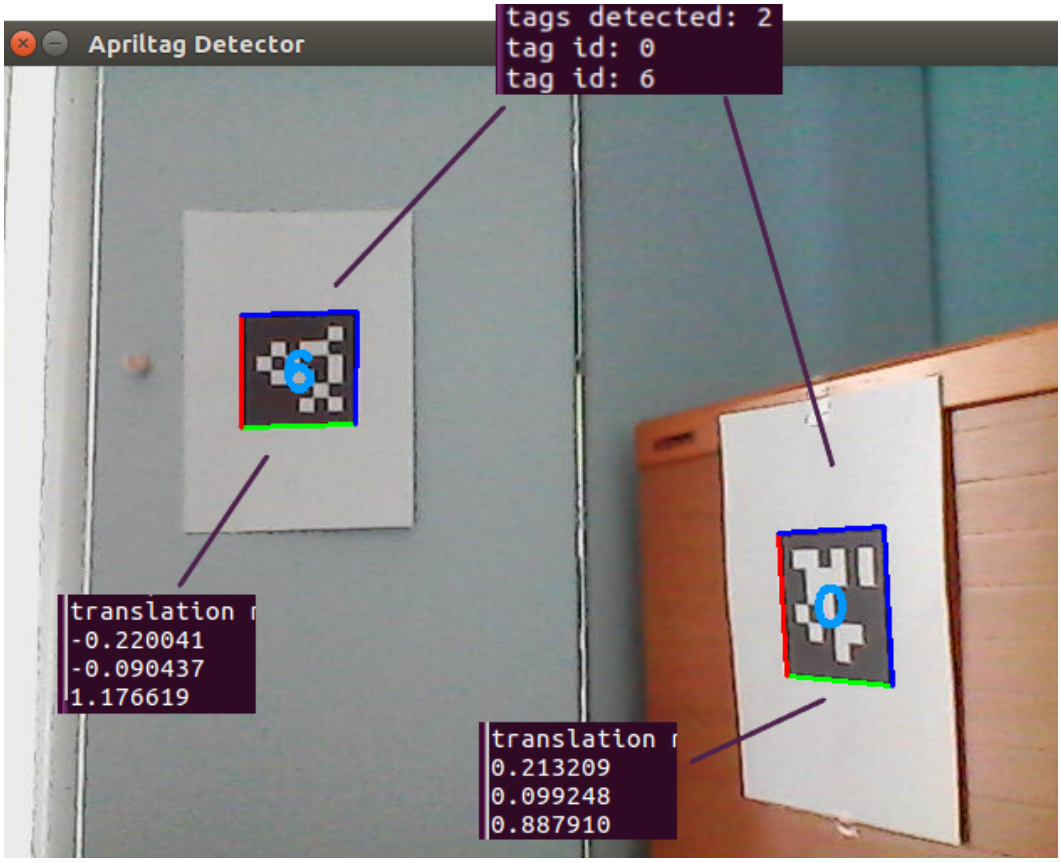


Figura 3.4. Esempio di rilevamento contemporaneo di due landmark con relative informazioni sui loro identificativi e posizioni nello spazio

A questo punto il sistema è pronto per la registrazione del dataset che sarà in fase successiva analizzato, elaborando un file g2o che descrive il grafo che ricrea l'ambiente di lavoro della piattaforma.

L'operazione di logging su file di testo è preceduta da un aggiornamento della variabile che contiene l'odometria corrente del robot.

Questa variabile consiste in un'istanza di una seconda tipologia di packets di Orazio denominata DifferentialDriveStatus. Tra i suoi vari campi sono di interesse per l'applicazione quelli riguardanti la posizione del robot nel piano, "odom_x" e "odom_y", e l'angolo creatosi con il sistema di riferimento del mondo "odom_theta". Per semplicità, il world frame coincide con la posizione iniziale della piattaforma al momento del suo avvio.

```
typedef struct {
  PacketHeader header;
  float odom_x, odom_y, odom_theta;
  float translational_velocity_measured;
  float rotational_velocity_measured;
  float translational_velocity_desired;
  float rotational_velocity_desired;
  float translational_velocity_adjusted;
  float rotational_velocity_adjusted;
  uint8_t enabled;
} DifferentialDriveStatusPacket;
```

Figura 3.5. Struttura di un Differential Drive Status packet di Orazio

Si noti come ora tutte le informazioni necessarie alla stesura del dataset sono a disposizione, di conseguenza con una banale operazione di scrittura su file è possibile registrarlo per sottoporlo all'elaborazione della fase successiva. In particolare, sono catalogate due categorie di dati:

- ODOM, che riporta la posizione globale del robot espressa nella terna x, y, θ;
- TAG, che contiene informazioni sull'id del tag rilevato e la sua posizione nel sistema di riferimento della camera.

Ogni dato è preceduto da un timestamp che indica quali tag sono individuati nel tempo in ogni determinata posizione della base: se i timestamp di un dato di tipo ODOM e di uno di tipo TAG coincidono, significa che il robot in quel punto dell'ambiente ha rilevato quel determinato tag. Un pratico esempio è il seguente:

```
<1593091715.991443> ODOM 0.703952 0.417587 1.211540
<1593091715.991443> TAG 1 -0.091154 -0.003077 1.675843
<1593091715.991443> TAG 6 -1.086601 0.101119 4.163209
```

Figura 3.6. Semplice esempio di file di log redatto dal client di Orazio

La coincidenza dei timestamp implica che in una determinata posizione l'agente robotico ha rilevato contemporaneamente due tag descritti dagli identificativi '1' e '6'.

A questo punto, muovendo la piattaforma a proprio piacere nello spazio, sono registrati tutti i suoi spostamenti con eventuali rilevamenti effettuati nel tempo. Una volta terminata l'acquisizione del dataset, si è pronti per ricavarne le informazioni necessarie per eseguire la parte finale del task prefissato: costruire un grafo g2o che ricrei il più fedelmente possibile il contesto d'azione.

3.3 Elaborazione dei dati e stesura del file g2o

La fase precedente ha visto la registrazione di tutte le odometrie del robot e le posizioni dei tag rilevati durante lo spostamento della base mobile nel tempo su un file di testo in cui ciascuna riga costituisce un dato di tipo ODOM o di tipo TAG. A questo punto, a fronte delle osservazioni preliminari esposte nel capitolo 2.5 sulle trasformazioni geometriche nel piano e nello spazio, è facile intuire come si disponga di tutte le informazioni necessarie alla creazione di un file g2o per poter costruire il grafo che ripropone il contesto in cui è avvenuta la registrazione del dataset.

Il file di log redatto da Orazio viene dato in input a un parser che si occupa della conversione in un secondo file da passare al framework di g2o. Il parser, inoltre, richiede anche determinati parametri, dei lower bound, che stabiliscono la minima variazione di spostamento della piattaforma che deve intercorrere tra due trasformazioni consecutive. Ciò implica che valori di questi parametri eccessivamente elevati ridurranno il numero di stime delle posizioni e di conseguenza anche la precisione con la quale si costruirà in seguito il grafo. I parametri, che rappresentano il minimo spostamento sull'asse delle x e delle y e la minima variazione di angolo θ tra due posizioni, di default sono impostati a:

- dx=0.2 m;
- dy=0.2 m;
- $d\theta$=$\frac{\pi}{4} = 0.7854$ (radianti).

Tuttavia, è possibile modificarli a proprio piacimento passandoli in input all'eseguibile del parser da linea di comando.

L'analisi effettuata scandisce il documento di log riga per riga, esaminandone la tipologia di dati che contiene ed eseguendo le trasformazioni geometriche necessarie. In particolare, se il dato corrente da analizzare è di tipo:

- ODOM, il parser verifica se la differenza tra le posizioni precedente e corrente raggiunge o supera almeno uno dei tre lower bound e, in caso affermativo, crea un nuovo VERTEX_SE2 e il conseguente EDGE_SE2 compilando il file g2o precedentemente inizializzato;
- TAG, dal momento che è richiesto un solo VERTEX_XY per ciascun tag, il parser si assicura preventivamente di aver già registrato tale landmark, o in alternativa crea il vertice che lo rappresenta, per poi costituire un nuovo EDGE_SE2_XY tra il tag e la posizione corrente della base.

Ovviamente, ogni creazione di un nuovo vertice o arco è preceduta dal calcolo delle trasformazioni tra i vari sistemi di riferimento della camera, del robot e del mondo. I calcoli che ne derivano sono effettuati adottando una struttura offerta da AprilTag, utile per la rappresentazione delle matrici e per un rapido svolgimento delle operazioni tra queste.

L'elaborazione termina nel momento in cui si raggiunge la fine del file di log stilato in precedenza da Orazio. Il risultato finale è un file g2o che racchiude i dati per una seguente rappresentazione del grafo corrispondente.

Capitolo 4

Procedura sperimentale e risultati

La tesi sperimentale ha previsto una applicazione pratica in un contesto reale per i ragionamenti finora condotti. L'acquisizione del dataset e la stesura del file di log da parte di Orazio è avvenuta esplorando un corridoio del Dipartimento di Ingegneria Informatica, Automatica e Gestionale "Antonio Ruberti" di Via Ariosto. Il sistema utilizzato era costituito da una base mobile su cui era collocato un computer portatile munito di webcam, in modo da garantire una corretta acquisizione dei frame della camera e il riconoscimento di eventuali tag ripresi. È stato utilizzato un joystick per comunicare con Orazio, tramite lo scambio di pacchetti, e per definire il movimento della piattaforma nell'ambiente circostante.

All'interno del corridoio del Dipartimento sono presenti degli armadi e delle panchine lungo le due pareti laterali, garantendo di conseguenza anche la percezione di ostacoli lungo il percorso.

Sono stati utilizzati quindici tag, posizionati più o meno uniformemente nell'ambiente.

L'esplorazione del robot è stata eseguita effettuando quattro giri completi descritti da movimenti circolari non sempre regolari, in modo da acquisire molteplici rilevazioni di tag contemporaneamente.

L'esperimento ha avuto una durata complessiva di circa otto minuti.

Al termine dell'osservazione, il file di log conteneva più di 36mila dati di cui circa 23mila di tipo ODOM e 13mila di tipo TAG. Ciascun landmark è stato individuato diverse centinaia di volte con una notevole differenziazione tra alcuni di loro: i tag più difficili da rilevare sono stati conteggiati tra le 350-400 volte, mentre altri evidentemente in posizioni più visibili hanno superato il migliaio di avvistamenti.

Nella pagina seguente viene riportata una piantina approssimativa del corridoio in cui è stato condotto il test, evidenziando i punti in cui i tag sono stati collocati nell'ambiente.

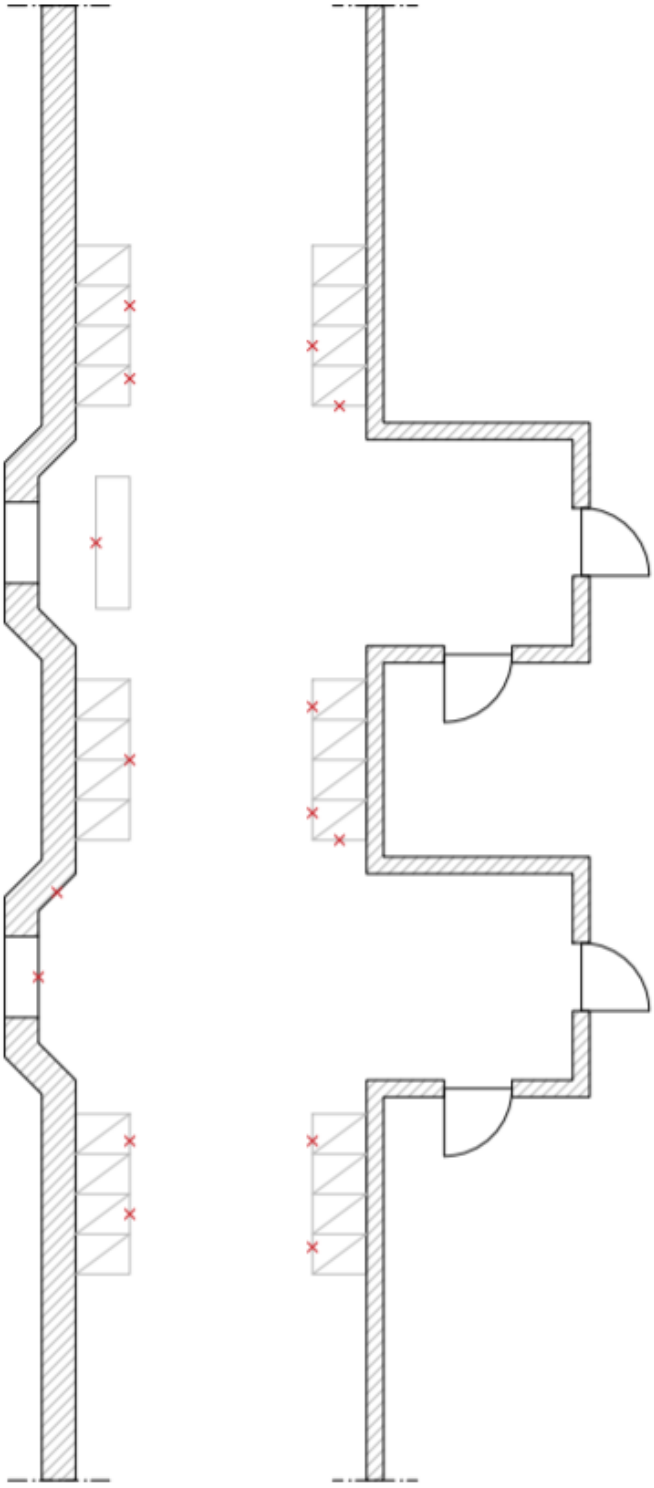


Figura 4.1. Ricostruzione approssimativa dell'ambiente dell'esperimento

A seguito dell'analisi del file di log da parte del parser, utilizzando i valori di default dei lower bound, è stato creato un secondo file di tipo g2o contenente 536 VERTEX_SE2 collegati da 535 EDGE_SE2, ovviamente 15 VERTEX_XY e un totale di 295 EDGE_SE2_XY.

La conclusione dell'esperimento ha visto il passaggio del file g2o al visualizzatore grafico, in modo da ottenere i grafi desiderati.

I risultati sperimentali ottenuti sono stati classificati a seconda se l'ottimizzazione offerta da g2o sia stata effettuata o meno.

Nelle due pagine seguenti sono mostrati i relativi grafi conseguiti in entrambe le circostanze. La non breve durata dell'esperimento e il movimento non sempre regolare della base mobile hanno introdotto un inevitabile errore di stima della posizione del robot, provocando di conseguenza uno sfasamento tra la posizione reale della base e quella invece mostrata nel grafo pre-ottimizzazione. Per gli stessi motivi, anche alcune localizzazioni dei tag risultano ambigue e confuse, allontanandosi dalla composizione originaria dell'ambiente d'esecuzione.

Tuttavia, il framework g2o si occupa proprio di ridurre l'errore presente tra le varie misurazioni. Pertanto, utilizzando il servizio di ottimizzazione di cui dispone, si è potuto constatare come questo errore sia stato minimizzato e reso quasi del tutto trascurabile, permettendo la riduzione del rumore e riproponendo la struttura del corridoio del Dipartimento in cui l'esperimento è stato svolto.

Come si può osservare, l'esperimento ha conseguito dei risultati estremamente soddisfacenti, ricreando in maniera realistica il contesto d'azione originario. La diversa profondità tra tag posizionati sulle pareti, sugli armadi e sulla panchina è stata colta ed evidenziata, così come non sono stati confusi tag tra loro molto vicini.

Il sistema è stato dunque capace di generare una mappatura di un ambiente reale a partire da misurazioni effettuate nello stesso con esiti più che positivi, dimostrando di poter garantire un corretto funzionamento in qualsiasi ambito applicativo venga utilizzato.

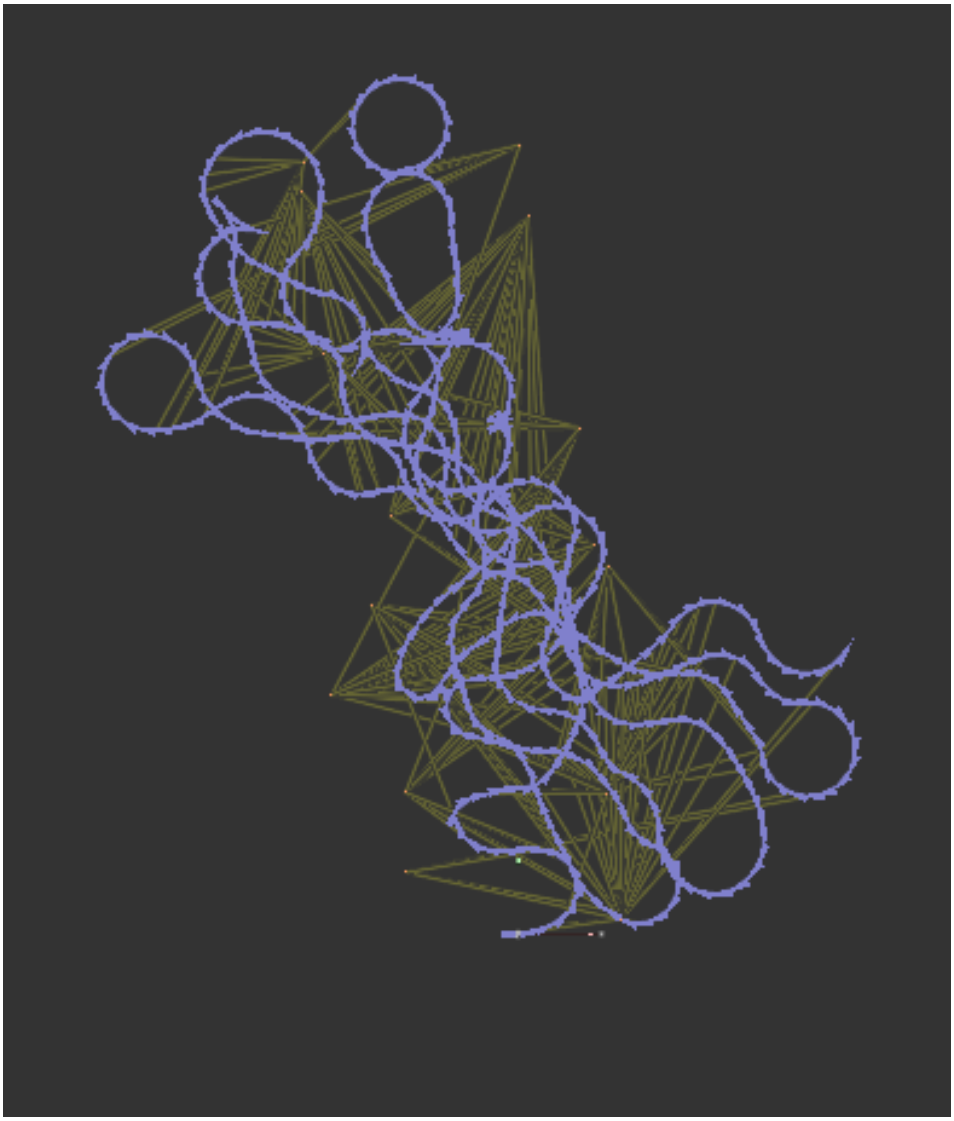


Figura 4.2. Risultati sperimentali prima dell'ottimizzazione di g2o

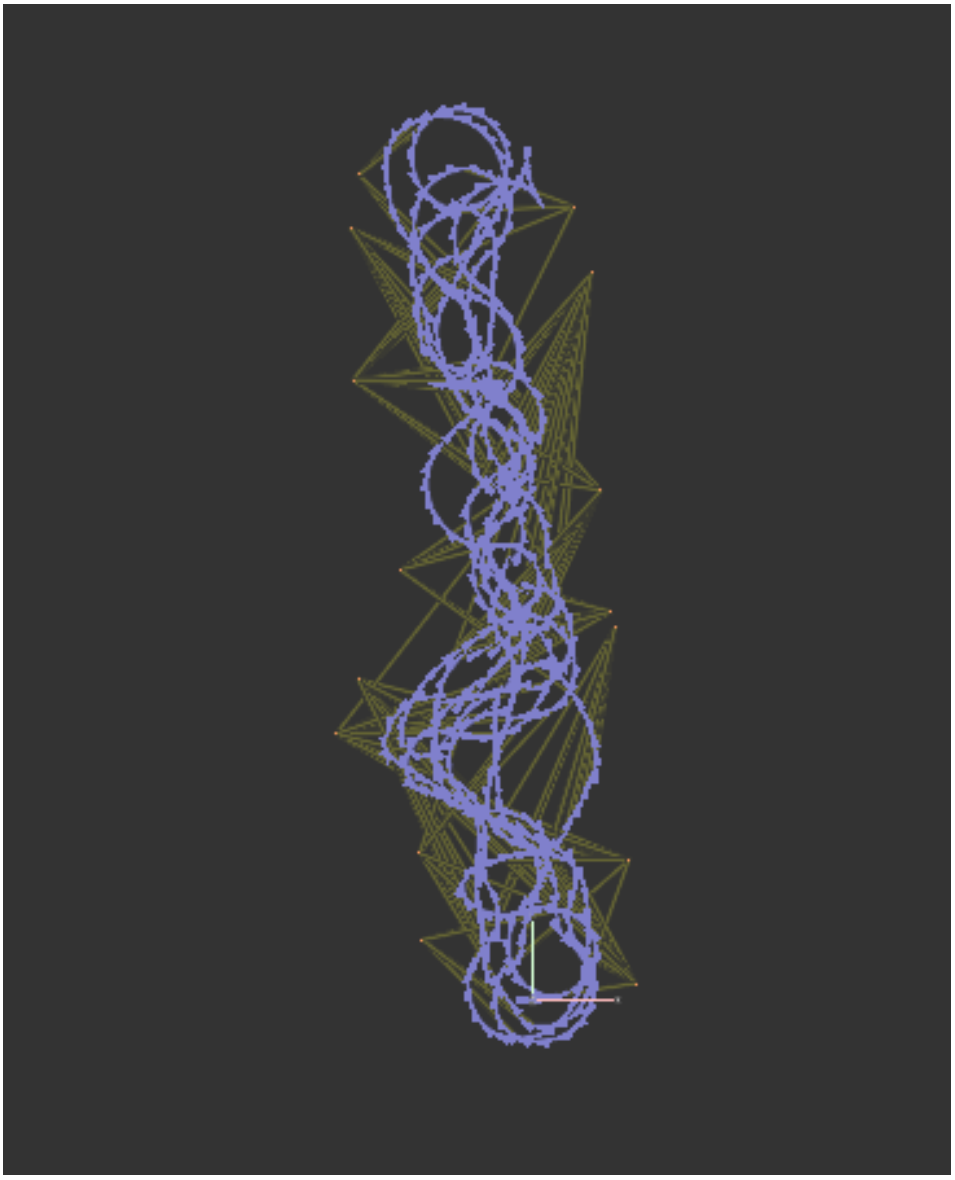


Figura 4.3. Risultati sperimentali in seguito all'ottimizzazione

Capitolo 5

Conclusioni

La tesi, delineandosi inizialmente come applicazione SLAM e coniugando discorsi teorici con osservazioni pratiche tramite l'esecuzione di un esperimento in un contesto reale, è stata capace di conseguire l'obiettivo prefissato, quello di generare la mappatura di un ambiente a partire da misurazioni in esso effettuate.

Possibili problemi riguardanti la coabitazione dei diversi stub e librerie di programmazione coinvolte come Orazio, AprilTag e OpenCV, sono stati analizzati e risolti sfruttando vantaggi di ciascuna di queste e in modo da garantirne un'integrazione costruttiva e fondamentale per la risoluzione del problema preposto.

Inoltre, le nozioni di studio preliminare che la tesi ha richiesto sono risultate cruciali per ottenere un esito pienamente positivo dell'esperimento.

Si è analizzato come difatti le conclusioni raggiunte abbiano permesso di realizzare in maniera soddisfacente un sistema capace di interagire correttamente con un ambiente stimolante per il complesso robotico utilizzato, ricavando in primo luogo i dati necessari per risolvere il task prefissato e in seguito analizzarli in modo da ricreare realisticamente l'ambiente in cui è stato eseguito.

In conclusione, il lavoro svolto ha collegato studi svolti durante il corso di laurea con altri di natura autonoma da parte dello studente, ha previsto un'applicazione pratica di questi con un'esperienza interattiva sul campo e ha permesso infine di conseguire degli ottimi risultati adempiendo esaustivamente agli obiettivi stabiliti agli inizi della discussione.

Capitolo 6

References

Grisetti, G. & Guadagnino, T. (2020) Introduction to Navigation using ROS.
Disponibile su:
https://gitlab.com/grisetti/lab_ia_gi_2019_20/-/tree/master/slides

Grisetti, G. & Guadagnino, T. (2020) Robot Sensors MARRtino.
Disponibile su:
https://gitlab.com/grisetti/lab_ia_gi_2019_20/-/tree/master/slides

Wikipedia. (2020) Modulazione di larghezza d'impulso.
Disponibile su:
https://it.wikipedia.org/wiki/Modulazione_di_larghezza_d'impulso

Wikipedia. (2019) Encoder (elettronica).
Disponibile su:
https://it.wikipedia.org/wiki/Encoder_(elettronica)

Wikipedia. (2020) EIA RS-232.
Disponibile su:
https://it.wikipedia.org/wiki/EIA_RS-232

Aloise, I. (2020) Devices.
Disponibile su:
https://gitlab.com/grisetti/sistemi_operativi_2019_20/-/tree/master/slides

Wikipedia. (2020) Video4Linux.
Disponibile su:
https://it.wikipedia.org/wiki/Video4Linux

Linux Programmer's Manual. ioctl(2).
Disponibile su:
https://man7.org/linux/man-pages/man2/ioctl.2.html

Linux Kernel Documentation. Video for Linux API.
Disponibile su:
https://www.kernel.org/doc/html/v4.10/media/uapi/v4l/v4l2.html

OpenCV documentation index.
Disponibile su:
https://docs.opencv.org

April Robotics Laboratory. AprilTag.
Disponibile su:
https://april.eecs.umich.edu/software/apriltag

AprilTag User Guide. (2020)
Disponibile su:
https://github.com/AprilRobotics/apriltag/wiki/AprilTag-User-Guide

Grisetti, G. (2020) A Compact Course on Linear Algebra.
Disponibile su:
https://gitlab.com/grisetti/lab_ia_gi_2019_20/-/tree/master/slides

OpenSLAM. g2o: A General Framework for Graph Optimization.
Disponibile su:
https://openslam-org.github.io/g2o.html

Camera calibration With OpenCV.
Disponibile su:
https://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/
camera_calibration.html

April Robotics Laboratory. Camera suite.
Disponibile su:
https://april.eecs.umich.edu/wiki/Camera_suite