UNIVERSITÀ DI BRESCIA
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione

Laboratorio di Robotica Avanzata
Advanced Robotics Laboratory

Corso di Robotica Mobile
(Prof. Riccardo Cassinis)

Where Am I – Localizzazione di un robot mediante codici QR

Elaborato di esame di:

Patrizio Bertoni, Michael Maghella, Giovanni Richini

Consegnato il: **14 luglio 2014**

Sommario

Dato un flusso continuo di immagini provenienti da una telecamera montata su un robot mobile, si vuole individuare la presenza almeno due codici QR in una vista a 180 gradi sul piano orizzontale. Stimate le posizioni polari relative ad ogni singolo codice QR, grazie a informazioni aggiuntive sulle posizioni cartesiane assolute degli stessi si vuole localizzare assolutamente il robot sul piano cartesiano.

1. Introduzione

Il lavoro riprende un elaborato precedentemente sviluppato da Nicola Zaghen ed Alex Zampieri durante l'A.A. 2012/2013, mirato al riconoscimento di codici QR e alla seguente stima dell'angolo di prospettiva e della distanza relativa all'osservatore. La continuazione del suddetto progetto, tra l'altro proposta dai suoi stessi autori come possibile sviluppo futuro, si propone l'obiettivo di pilotare la telecamera montata sul robot mobile Morgul all'interno del settore circolare anteriore (per 180° complessivi) e ricercare un numero sufficiente di codici QR al fine di poter localizzare il robot all'interno di un mondo modellizzato da un piano cartesiano.

Successivamente a un'analisi teorica, descritta nella sezione 2, il lavoro svolto si suddivide nelle seguenti fasi:

1. Riprogettazione software del lavoro precedente;
2. Implementazione della soluzione proposta;
3. Testing finale per la validazione dei risultati ottenuti.

Per approfondire la teoria relativa ai codici QR e le tecniche software per riconoscerli mediante apposite librerie di acquisizione ed elaborazione delle immagini, si rimanda alla relazione del precedente elaborato disponibile al seguente link: http://www.ing.unibs.it/~arl/docs/projects/Sen_10.pdf.

Nota sulla relazione: è stato mantenuto quanto possibile il linguaggio italiano. Gli sporadici termini inglesi utilizzati lo sono in quanto intraducibili e/o utilizzati come identificatori nel codice sorgente, totalmente sviluppato in lingua inglese.

1.1. GitHub

Per lo sviluppo ed il versioning del codice si è optato per il servizio di web-hosting accessibile all'indirizzo https://github.com, nato per ospitare repository di tipo Git. La scelta è stata ulteriormente motivata dall'impossibilità di avere giorni d'incontro fissi. Su GitHub è attualmente mantenuto il progetto *whereami*, e in accordo con la filosofia *opensource*, chiunque può accedervi e effettuare delle *pull requests*.

All'indirizzo *https://github.com/nzaghen/qrlocate* è disponibile il repository del progetto originario di Zaghen, dal quale, tramite *forking*, è nato il progetto *whereami*.

1.2. Sistema destinatario

Il programma è attualmente portato su un netbook Acer Aspire One, dotato di processore Intel Atom 1.60GHz, 512MB di RAM e sistema operativo Debian, kernel 2.6.32.5-686, versione "Squeeze". Il calcolatore è interfacciato con una videocamera Philips *SPC1005NC* con risoluzione di 1.3 MP reali e 5 MP virtuali ottenuti tramite interpolazione.

2. Il problema affrontato

Segue un'analisi teorica del problema della localizzazione con vincoli reali, quali i limiti fisici della qualità dei dati a disposizione (i fotogrammi del flusso video), e delle risorse offerte dal calcolatore che ospita il programma.

2.1. Esplorazione del settore circolare

La camera installata sul robot mobile è posizionata su di un piccolo servomotore elettrico capace di farne ruotare l'asse verticale in uno spazio di 180°, esattamente 90° a destra più altrettanti a sinistra rispetto all'asse del robot.

È possibile muovere il motorino ad un angolo a piacere richiamando un apposito programma presente sul robot, *morgulservo*, specificando i gradi a cui la telecamera deve posizionarsi.

2.2. Gestione dei codici QR noti e riconosciuti

Innanzitutto, *whereami* necessita di imparare una quantità minima di fatti sufficiente a potersi localizzare. Nella sezione 3.1.2 questo verrà descritto più ampiamente.

Nella sua esecuzione, il programma dovrà inoltre ricercare e salvare le informazioni relative ai codici QR rilevati. La ricerca, effettuata attraverso l'elaborazione dei fotogrammi costituenti il flusso video generato dalla videocamera avverrà in tempo reale ed è perciò necessaria una logica intelligente che rispetti i requisiti di precisione (efficacia) e velocità di esecuzione (efficienza).

I dati acquisiti dovranno pertanto essere memorizzati in strutture dati adeguate, se necessario create appositamente.

2.3. Il problema della localizzazione

La risoluzione del problema dell'auto-localizzazione permette ad un robot di riconoscere in maniera autonoma la propria posizione, e quindi di rispondere alla domanda "Where am I?".

Una volta ispezionato l'ambiente che lo circonda, il robot calcola la propria posizione, utilizzando i QR rilevati come punti di riferimento.

3. La soluzione adottata

Uno dei metodi per risolvere il problema della localizzazione assoluta di un robot mobile è utilizzare dei punti di riferimento (definiti come *Landmark*, da qui in avanti) che il robot può riconoscere ed utilizzare per calcolare la sua posizione.

Vista la natura dell'elaborato scelto come base per il progetto *whereami*, l'utilizzo di codici QR per il ruolo dei marcatori geografici ne è stata banale conseguenza. Ogni QR contiene un messaggio che lo identifica univocamente, mentre nella memoria del programma è definita una tabella di tutti i possibili QR noti (e quindi riconoscibili), insieme alle relative coordinate spaziali, conosciute a priori.

Nella fase di ricerca, quando il programma riconosce un QR, verifica innanzitutto che questo sia presente tra i QR conosciuti. Se ciò è vero, e se il QR è in posizione centrale rispetto all'asse verticale dell'immagine (la ragione di ciò sarà motivata in 3.2.), il programma salva la distanza del landmark dal robot e l'angolo attuale di rotazione della telecamera.

Una volta terminata la fase di ispezione, se il robot ha collezionato un numero sufficiente di dati, procede con il calcolo della propria posizione.

3.1. Riprogettazione software

Il precedente progetto, *qrlocate*, è nato come codice principalmente scritto nel linguaggio di programmazione C, ma come dimostra il diagramma UML delle classi a 3.1.1, si è cercato di seguire un paradigma orientato agli oggetti. Il codice di *whereami* è stato quindi prevalentemente sviluppato in C++. Il codice *qrlocate* è fortemente imperativo: vi si trova un'alternanza continua di processi per l'acquisizione e l'elaborazione dei singoli fotogrammi del flusso video, al fine di riconoscervi la presenza di un codice QR e stimarne quindi distanza e angolo rispetto alla camera.

Dopo una sessione di analisi del presente problema, si è deciso di implementare un automa a stati finiti, nominato *ExplorerFSM*, essendo questo un modello che permette di descrivere con precisione ed in modo formale il comportamento del programma desiderato.

Per comprendere la struttura del software prodotto, presentiamo *ExplorerFSM* nella sezione 3.1.3, nelle cui sottosezioni ne verranno definiti, in modo dettagliato, ogni stato e transizione.

3.1.1. Diagramma delle classi

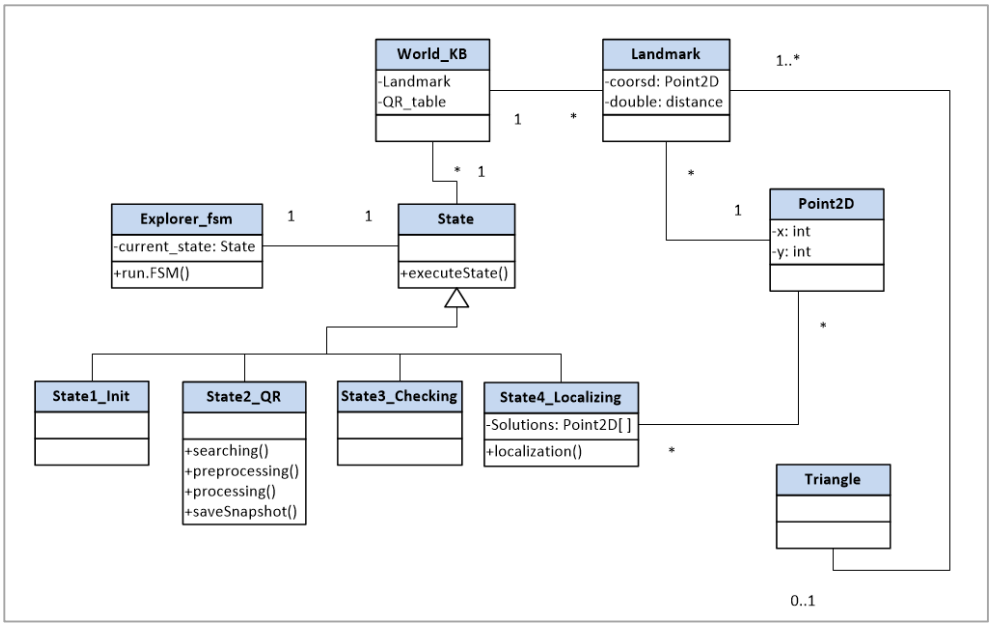

Fig. 1 - Figura 1 - Diagramma UML delle classi.

3.1.2. Base di conoscenza del mondo

Per poter memorizzare i QR da elaborare si è scelto di implementare una struttura dedicata, indipendente dalla macchina a stati e ad essa conosciuta.

Denominata *WorldKB*, essa contiene, sotto forma di un array, le informazioni relative a tutti i possibili codici QR che fungono da *Landmark*.

All'inizio del programma le informazioni *statiche* (identificativo alfanumerico e coordinate cartesiane sul piano) dei *Landmark* dovranno essere dunque acquisite, come descritto meglio nella sezione 4.3.2.

Oltre ai suddetti attributi statici, un *Landmark* ne prevede altri detti *dinamici*, che saranno memorizzati al momento del riconoscimento del relativo QR:

1. Distanza tra l'obiettivo fotografico e il codice;
2. Angolo, rispetto a quello iniziale della videocamera, in cui il codice è stato rilevato in posizione centrale rispetto al fotogramma.

3.1.3. Macchina a stati

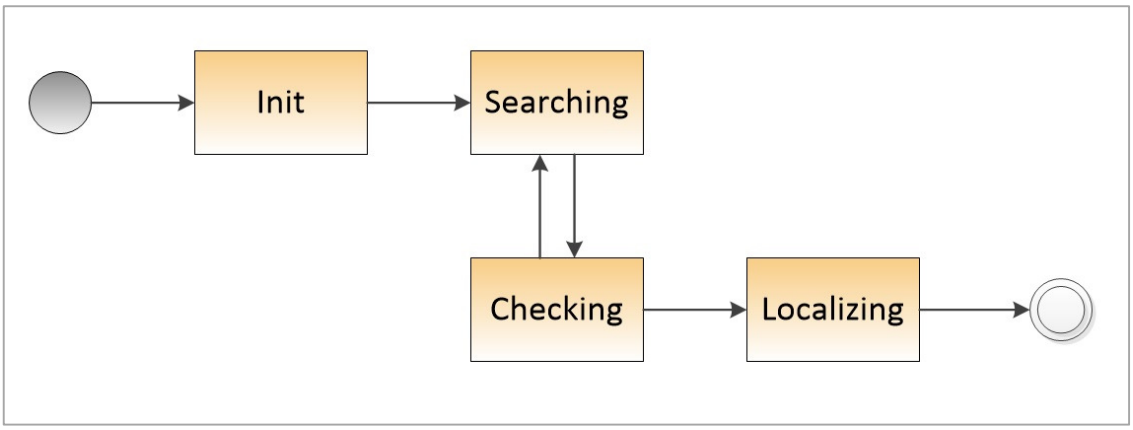

Fig. 2 - Figura 2 - Automa a stati finiti.

3.1.3.1 Init

Il primo stato serve a inizializzare il robot a una situazione predefinita. Viene pertanto portata la telecamera all'angolazione di partenza, solitamente fissata a -90°.

3.1.3.2 Searching

La ricerca può essere definita come il cuore della macchina a stati.

Grazie all'implementazione di thread vengono svolti due compiti paralleli:

1. Gestione del movimento in senso orario, dalla posizione iniziale a quella finale, alternando una rotazione di una certa quantità angolare a un certo periodo di riposo del thread, necessario perché la telecamera possa percepire un'immagine stabile e non mossa;
2. Elaborazione dell'immagine acquisita dalla telecamera, alla ricerca della presenza di codici QR ivi contenuti.

Per migliorare l'efficienza del programma, la ricerca è attuabile in due modalità; una grossolana, *rough*, e una fine, *fine*. I parametri del programma che variano tra le due modalità sono i seguenti, che verranno descritti in 4.3.2:

1. Granularità dello spostamento angolare;
2. Tempo di riposo del programma dopo ogni spostamento angolare;

3. Numero di tentativi per cui ciascun fotogramma viene elaborato.

L'idea è di esplorare quanto più possibile il settore circolare grossolanamente, rallentando con l'entrata in modalità fine (quindi diminuendo i parametri 1°,3° e aumentando il 2° dell'elenco precedente) solo quando si rileva la vicinanza di un codice QR non ancora centrato, nella parte destra del fotogramma.

La centralità del QR rispetto all'asse verticale è fondamentale per l'accuratezza della localizzazione, essendo un'ipotesi dell'algoritmo descritto in 3.2.

Appena il QR viene riconosciuto centralmente, oppure è ormai sfuggito all'ispezione (trovandosi quindi nella parte sinistra dell'immagine, essendo la scansione effettuata in senso orario) viene ripristinata la modalità *rough*.

In Figura 3 - Diagramma di flusso, logica Searching., la logica appena descritta è mostrata attraverso un diagramma di flusso.

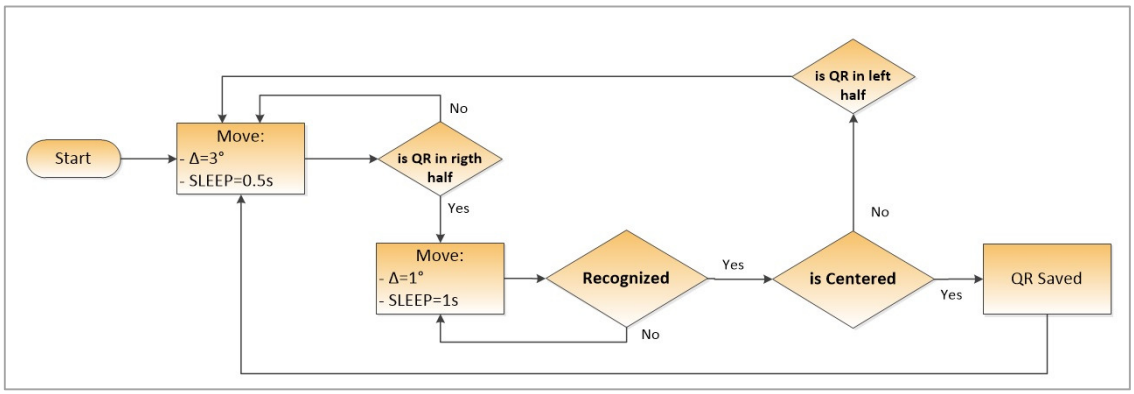

Fig. 3 - Figura 3 - Diagramma di flusso, logica Searching.

La ricerca termina quando la camera si trova nella posizione finale (parametrizzata) del settore circolare da esplorare, solitamente fissata a +90°.

Al fine di limitare i problemi dovuti alla luminosità e alle ombre dell'ambiente, il fotogramma (a tre canali cromatici) acquisito viene prima convertito in una scala di grigio e successivamente filtrato per generare un'immagine binaria con una soglia di ripartizione tra bianco e nero, come descritto in 4.3.3.

In Figura 4 possiamo notare i miglioramenti dovuti al filtraggio, a sinistra troviamo l'immagine unicamente convertita in scala di grigi mentre, a destra, quella filtrata.

Fig. 4 - Figura 4 - Confronto immagine con e senza binarizzazione

3.1.3.3 Checking

Questo controllo viene sicuramente raggiunto al termine dello stato di *Searching*. Se si dispone di un numero sufficiente di *Landmark* riconosciuti (almeno due) si passa immediatamente allo stato *Localizing*. In caso contrario il programma termina, informando l'utente che non sono stati individuati un numero sufficiente di QR per effettuare la localizzazione.

3.1.3.4 Localizing

Questa fase finale contiene l'algoritmo centrale di localizzazione, *core*, che implementa il problema descritto in 3.2.

Il suo compito è la computazione dell'algoritmo *core*, date in ingresso tutte le possibili combinazioni di coppie senza ripetizione sull'insieme di tutti e soli i *Landmark* della *WorldKB* che sono stati rilevati. Ognuna di queste coppie è descritta nel programma dalla classe *Triangle*, che contiene i riferimenti ai due *Landmark* interessati e le coordinate alle quali il robot dovrebbe trovarsi. Una volta esaurita questa enumerazione, viene infine viene calcolata una media di tutte le $\binom{n}{k}$ soluzioni ottenute.

3.2. Risoluzione del problema geometrico

Presentiamo ora la risoluzione del problema geometrico:

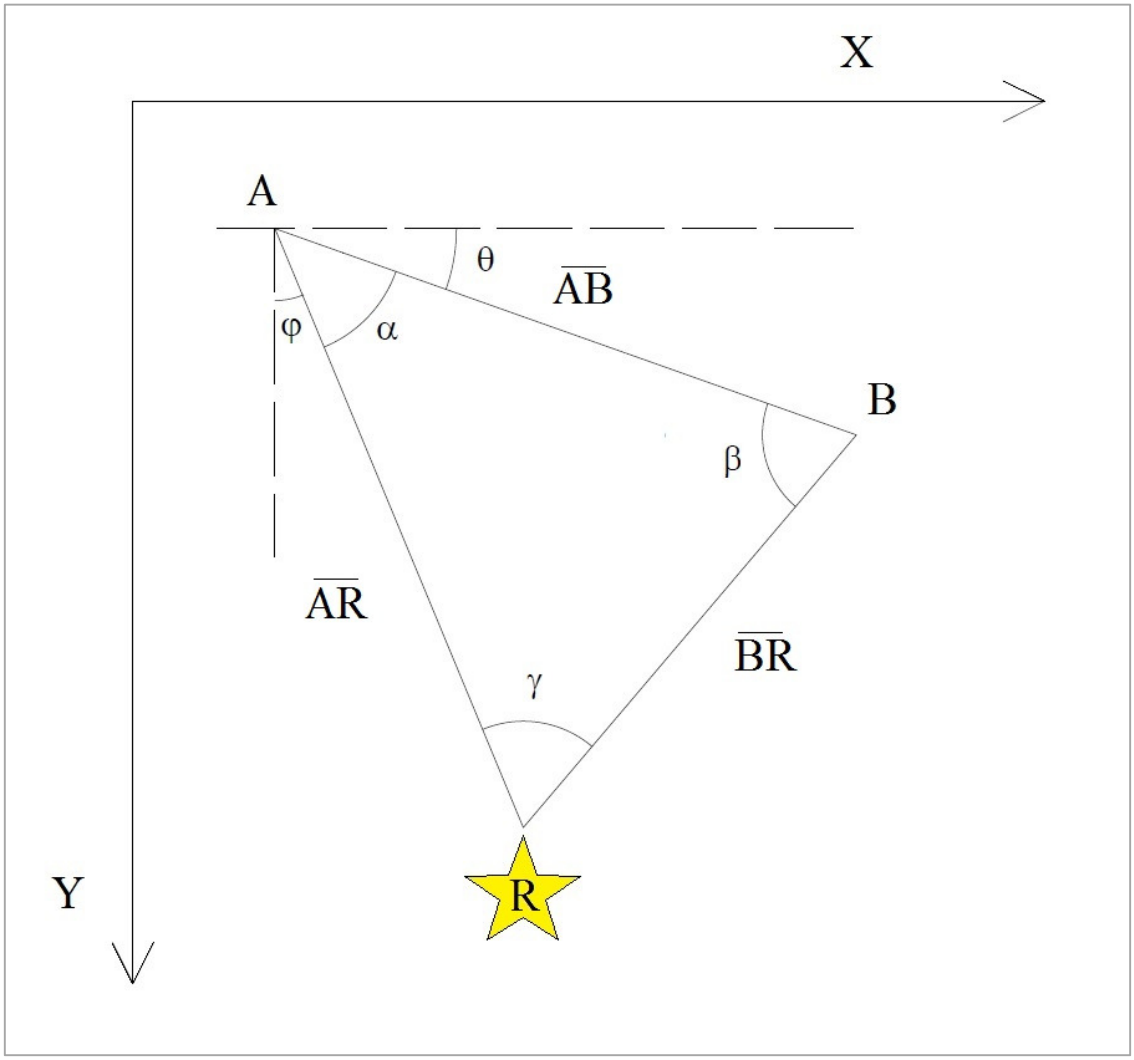

Fig. 5 - Figura 5 - Modellizzazione del problema

Dati:

$$A = (x_A, y_A), \quad \delta_A$$
$$B = (x_B, y_B), \quad \delta_B$$

Dove δ_A e δ_B sono gli angoli in cui la telecamera ha centrato i codici QR nei punti A e B.

Si deve calcolare:

$$R = (x_R, y_B)$$

Grazie al teorema dei seni:

$$\alpha = \sin^{-1}\left(\frac{\overline{BR}}{\overline{AB}}\sin\gamma\right)$$

Essendo:

$$\gamma = \delta_B - \delta_A > 0$$
$$\overline{AB} = \sqrt{(y_B - y_A)^2 + (x_B - x_A)^2}$$

Adesso:

$$\theta = \tan^{-1}\frac{y_B - y_A}{x_B - x_A}$$

$$\varphi = \frac{\pi}{2} - \alpha + \theta$$

$$\Delta_{AR}^{x} = \overline{AR}\sin\varphi$$

$$\Delta_{AR}^{y} = \begin{cases} \overline{AR}\cos\varphi\,, & \theta \geq \alpha \\ -\overline{AR}\cos\varphi\,, & \theta < \alpha \end{cases}$$

Pertanto

$$x_R = x_A + \Delta_{AR}^{x}$$
$$y_R = x_A + \Delta_{AR}^{y}$$

3.3. Risultati sperimentali

In Tabella 1 tabulata la conoscenza a priori del mondo dei test sperimentali svolti.

Tabella 1

	X	Y
QR 1	0	1180
QR 2	1250	0
QR 3	3000	720

Le simulazioni riportate in Tabella 3 sono state condotte con due differenti configurazioni; seguono in Tabella 2 i relativi file dei parametri, ove le differenze sono state evidenziate.

Tabella 2

Configurazione 1	**Configurazione 2**
CAMERA=0 STARTANGLE=-90 STEPANGLEROUGH=3 ENDANGLE=90 STEPSLEEPROUGH=0.1 INITSLEEP=5 CENTER_TOL=60 NTRY=2 STEPANGLEFINE=1 STEPSLEEPFINE=0.5 NTRYFINE=5 BWTRESH=100	CAMERA=0 STARTANGLE=-90 STEPANGLEROUGH=3 ENDANGLE=90 STEPSLEEPROUGH=1 INITSLEEP=5 CENTER_TOL=60 NTRY=2 STEPANGLEFINE=1 STEPSLEEPFINE=1 NTRYFINE=5 BWTRESH=100

Tabella 3

	X	Y
Posizione Robot reale	1250	1180
Configurazione 1		
QR 1, QR 2	1372	1270
QR 2, QR 3	1318	1310
QR 3, QR 1	1336	1293
Valore medio	1323	1277
Errore	92	111
Configurazione 2		
QR 1, QR 2	1297	1152
QR 2, QR 3	1308	1280
QR 3, QR 1	1316	1234
Valore medio	1307	1222
Errore	57	42

Per ottenere una misura di bontà delle localizzazioni, abbiamo stimato l'errore che viene commesso rispetto al valore medio di ciascuna delle coordinate ottenute.

La errore commesso dalla configurazione 1 è di 92 mm per l'asse X e di 111 mm per l'asse Y, mentre per la configurazione 2 gli errori si riducono a 57 mm e 42 mm.

I risultati ottenuti dalle diverse simulazioni effettuate si possono pertanto ritenere soddisfacenti.

La configurazione 2 è più precisa, in quanto viene dato più tempo alla telecamera per acquisire l'immagine, che risulta più nitida e ferma.

4. Modalità operative

4.1. Componenti necessari

Al fine di utilizzare correttamente il programma è necessario che vi sia installato lo strumento essenziale per l'elaborazione delle immagini: OpenCV. La guida per l'installazione può essere trovata al seguente link: http://docs.opencv.org/doc/tutorials/introduction/linux_install/linux_install.html

4.2. Modalità di installazione

Come già descritto il programma risiede su di un repository esterno, è necessario perciò effettuare un'operazione di clonazione per poterlo scaricare sul proprio calcolatore attraverso il seguente comando:

git clone https://github.com/pbertoni/whereami

Una volta scaricato è necessario compilarlo al fine di poterlo eseguire, posizionarsi nella cartella ed invocare l'apposito comando:

cd whereami

make whereami

Una volta che la compilazione è andata a buon fine eseguire il programma:

./whereami

4.3. Modalità di taratura

Per questioni di flessibilità e comodità, e per prevenire situazioni in cui potrebbe non esserci la possibilità di ricompilare il programma, il maggior numero possibile di parametri del programma viene letto da specifici file testuali di configurazione, dai nomi e dalla struttura però ben rigidi.

4.3.1. Calibrazione della videocamera

Come descritto nell'elaborato *qrlocate* di N. Zaghen & A. Zampieri, il programma *calibration* produceva, nel suo stesso direttorio, un documento chiamato *out_camera_data.yml*, contenente le informazioni necessarie a un'ottimizzazione della videocamera per la riduzione delle varie aberrazioni grafiche.

4.3.2. Conoscenza a priori del mondo esplorato

All'interno di *StaticKB.txt* sono memorizzati, nella forma grammaticale:

QR (*label*, *x_coord*, *y_coord*)

I singoli segnali geografici (*Landmark*) rappresentanti posizioni note nel mondo, in corrispondenza delle quali dovrà esistere un QR-code avente come messaggio l'identificativo alfanumerico *label*.

4.3.3. Parametri di funzionamento

Infine, dentro a *parameters.txt* verranno letti i seguenti parametri di funzionamento:

1. Identificativo della videocamera che diventerà sorgente di fotogrammi. Se è presente una sola videocamera, sia integrata che esterna, il valore *0* (zero) dovrebbe correttamente portare ad essa, supposta la compatibilità con OpenCV. Con più di una telecamera, è necessario fare delle prove o documentarsi meglio sulla suddetta libreria grafica;
2. Angolo iniziale di posizionamento della videocamera;
3. Granularità dello spostamento angolare in modalità *grossa*;
4. Angolo finale di posizionamento della videocamera;
5. Tempo di riposo del programma dopo ogni spostamento in modalità *grossa*;
6. Tempo di riposo del programma subito dopo lo spostamento iniziale;

7. Tolleranza di decentramento, come l'intervallo di pixel entro il quale la differenza tra il centro del fotogramma e quello del QR al suo interno.
8. Numero di tentativi per cui ciascun fotogramma viene elaborato, in modalità *grossa*;
9. Granularità dello spostamento angolare in modalità *fine*;
10. Tempo di riposo del programma dopo ogni spostamento in modalità *fine*;
11. Numero di tentativi per cui ciascun fotogramma viene elaborato, in modalità *fine*.
12. Valore di soglia, tra 0 e 255, per filtrare il fotogramma in scala di grigio e crearne una versione binaria in bianco e nero.

Nota bene: tutti gli angoli sono trecentosessantesimali, tutti i tempi sono in secondi.

5. Conclusioni e sviluppi futuri

È stato realizzato è un sistema software che, grazie a una telecamera rotante, è in grado di scandire un settore circolare alla ricerca codici QR. Una volta analizzato un nuovo codice noto nella base di conoscenza, è in grado di estrarne:

1. Contenuto informativo che lo identifichi;
2. Distanza a cui esso si trova rispetto all'obiettivo fotografico
3. Angolo di rotazione, a partire dall'angolo iniziale, a cui è stato trovato e centrato.

Grazie a questo può auto-localizzarsi nello spazio.

Una volta sviluppato il programma si è effettuata una verifica di funzionamento attraverso diverse simulazioni.

L'elaborato potrebbe essere esteso e migliorato, senza particolari sforzi, in diversi aspetti.

Se ne propongono due:

1. Efficienza: una ricerca binaria, sostituita alla logica di regolazione grossa/fine nella ricerca, potrebbe diminuire il tempo di esecuzione;
2. Efficacia: potendo disporre di versioni più recenti di OpenCV rispetto a quella utilizzata nella sperimentazioni, si sfrutterebbe al massimo la risoluzione offerta dalla videocamera in dotazione. Questo porterebbe però a elaborazioni ben più gravose, controbilanciando i miglioramenti nella dimensione dell'efficienza.

Un ulteriore aspetto di miglioramento potrebbe derivare da un possibile confronto tra la modellizzazione matematica qui implementata (trigonometrica) e quella basata sulla tecnica degli archi capaci.

Bibliografia

[1] Nicola Zaghen ed Alex Zampieri, *qrlocate*: https://github.com/nzaghen/qrlocate

[2] OpenCV, documentazione e tutorial: http://docs.opencv.org/

[3] Documentazione C++: http://www.cplusplus.com/reference/

Indice