UNIVERSITA' POLITECNICA DELLE MARCHE

FACOLTA' DI INGEGNERIA INDUSTRIALE E SCIENZE MATEMATICHE

Corso di Laurea triennale in Ingegneria Meccanica

OTTIMIZZAZIONE DELLA PROCEDURA (SLAM) SIMULTANEOUS LOCALIZATION AND MAPPING PER LA DETERMINAZIONE DI CARATTERISTICHE GEOMETRICHE

OPTIMISING SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM) PROCESS IN ORDER TO MAP THE ENVIRONMENT

Relatore:

Prof. Castellini Paolo

Tesi di Laurea di:

Costante Simone

Correlatore:

Prof.ssa Martarelli Milena

ANNO ACCADEMICO **2019/ 2020**

Ai miei genitori

A mia sorella Valentina

Indice

Elenco delle figure

Elenco dei Codici

Introduzione

La robotica è una scienza interdisciplinare che a partire dagli anni ’50 ha subito uno sviluppo vertiginoso e oggigiorno si erige come struttura portante dell’industria manifatturiera. La versatilità di tale branca dell’ingegneria la ha resa indispensabile in numerosi ambiti come la medicina, l’esercito e l’assistenzialismo.

Lo sviluppo della robotica però non ha solo un risvolto meramente economico e pratico ma anche sociale: sono cospicue le situazioni in cui la robotica ha facilitato la vita dell’uomo nell’ultimo decennio. Ne è un esempio la situazione attuale in cui una pandemia mondiale, causata da un virus trasmissibile per via aerea, ha costretto l’imprenditoria a prediligere in molti casi l’acquisto di bracci robotici per ridurre la possibilità di assembramenti nelle aziende.

In questo contesto si inserisce la decisione di orientare la tesi verso tale disciplina, in particolar modo riguardo la capacità di un robot di rilevare lo spazio circostante e di sapersi localizzare all’interno di esso. Tale branca della ricerca è stata denominata SLAM, acronimo inglese che sta per *Simultaneous Localization And Mapping.* Questo problema computazionale trova nella vita di tutti i giorni numerose applicazioni come: la realtà aumentata, la guida autonoma e la robotica. Lo SLAM è diventato uno strumento indispensabile laddove altre tecnologie, molto meno complesse, falliscono. I sistemi GPS ad esempio, usati per la localizzazione, non possono essere utilizzati in luoghi inaccessibili e spesso non sono considerati abbastanza accurati. Inoltre non è sempre possibile avere una conoscenza a priori dell’ambiente in cui un robot verrà impiegato. Per i suddetti motivi è evidente come lo SLAM possa rappresentare un’alternativa valida a molte situazioni attualmente adottate.

Il progetto di tesi trattato in questo documento concerne l’analisi di un algoritmo ORB-SLAM, presente su Matlab, una piattaforma di calcolo scritta in C, basata su un linguaggio di programmazione creato dalla MathWorks. La trattazione passo per passo del codice è presentata nel capitolo 2, mentre nel primo capitolo viene

proposto un riepilogo introduttivo sullo stato dell'arte di tale disciplina e alcune ricerche correlate. Infine nel capitolo 3 è presente l'ottimizzazione che è stata fatta al codice, la quale permette all'utente di modificare con più facilità i parametri di ingresso dell'algoritmo, il calcolo dell'RMSE che indica la precisione con cui i risultati sono ottenuti e la rappresentazione della nuvola di punti estrapolata dall'algoritmo mediante analisi delle features contenute nelle immagini.

Capitolo 1

Stato dell'arte

Il termine SLAM è un acronimo che sta per Simultaneous Localization And Mapping: ovvero la capacità di un apparato, in genere un robot, di ricostruire la mappa di un ambiente sconosciuto mentre contemporaneamente si muove all'interno di esso, descrivendo una traiettoria. Tale problema è stato in principio affrontato da Hugh Durrant-Whyte e John J. Leonard, i quali si sono basati sui lavori precedenti di Smith, Self and Cheeseman [1]. In sostanza si tratta di un complesso algoritmo scomponibile in più fasi, le quali vengono costantemente migliorate e riprogettate a seconda del risvolto pratico auspicato. Nonostante i numerosi metodi utilizzati possiamo comunque distinguere un filo logico trasversale ad ogni algoritmo: in principio si effettua l'estrapolazione dei punti notevoli (chiamati in inglese "landmarks"), successivamente si confrontano i dati trovati, infine si ipotizza la posizione del robot e la conformazione dell'ambiente. La procedura SLAM può essere applicata, a seconda dei casi, sia per scene a due dimensioni sia a tre dimensioni purchè vengano adottati degli oppurtuni accorgimenti e vengano impostati in maniera corretta i parametri, come verrà ampiamente spiegato nei capitoli successivi.

Questo processo è necessariamente dotato di una componente *software,* contraddistinta da un cospicuo numero di algoritmi e di una componente *hardware* che invece può essere formata da uno o più sensori dedicati. A testimonianza di ciò la comunità scientifica nel corso degli anni ha presentato molte versioni diverse di SLAM tra le quali è necessario citare:

- Optical SLAM, che utilizza nella quasi totalità dei casi dei sensori LIDAR, acronimo dall'inglese *Laser Imaging Detection and Ranging*. Tali dispositivi

permettono il telerilevamento di oggetti o superfici attraverso un'impulso laser continuo.

- Visual SLAM, che predispone l'utilizzo di una o più fotocamere a seconda se si voglia prediligere una visione monoculare o stereoscopica.
- Radar SLAM, è una tecnica per molti aspetti simile a quella LIDAR, dove il rilevamento delle distanze dall'ambiente circostante è calcolato attraverso le onde radio emesse da un radar.
- Wifi-SLAM, uno degli approcci più innovativi il quale richiede un punto di accesso ad una comune rete wifi e che quindi non necessita di una ingombrante componentistica meccanica.

  E' doveroso notare come in tale elenco non sia presente l'utilizzo di un sistema GPS, questo perché l'utilizzo di un robot, ad esempio in campo militare, richiede un segnale costante e affidabile in ambienti spesso remoti e inaccessibili, cosa che un sistema GPS non potrebbe garantire.

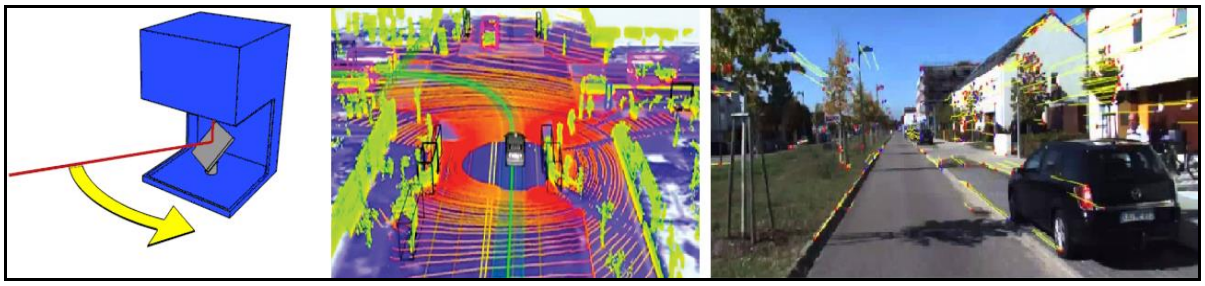

Figura 1 Esempi di SLAM, da destra verso sinistra: Lidar SLAM, Radar SLAM e Visual SLAM.

1.1 Ricerche in campo scientifico

La comunità scientifica negli ultimi 30 anni ha fatto progressi stupefacenti riguardo lo SLAM permettendo applicazioni su vasta scala nella vita reale e traghettando questa tecnologia anche a livelli industriali, in particolare per quanto riguarda la robotica. La peculiarità principale dello SLAM è la capacità di permettere a un robot di operare in ambienti di cui non si ha alcuna informazione sull'ambiente circostante e di conseguenza non si ha accesso ad una mappa preesistente. Ad esempio la robotica applicata all'industria manufatturiera prevede la conoscenza a priori degli spazi nei quali il braccio robotico andrà ad operare oppure se ciò non è possibile spesso è necessario comunque l'utilizzo di un sistema GPS che prevede un segnale costante emesso da un satellite. Ci si potrebbe chiedere il perché dello sviluppo di un programma così complesso quando in realtà per molti anni si è parlato di odometria, ovvero la tecnica che permette di stimare la posizione di un veicolo su ruote grazie ai sensori, allo spazio percorso dalle ruote e dall'angolo di sterzo. La differenza sostanziale tra un approccio SLAM e l'odometria è legata al concetto di *loop closure,* ossia la capacità di un dispositivo di riconoscere quando di ritorna su una posizione precedentemente acquisita, in modo tale da creare delle traiettorie chiuse. Questa tecnica permette al robot di riconoscere meglio la sua traiettoria e di poterla correggere qualora ci siano degli errori. Per quanto riguarda l'odometria invece gli errori, anche minimi, sono accumulati in maniera crescente con lo spostamento del robot, causando anche dopo alcuni metri degli errori per niente trascurabili.

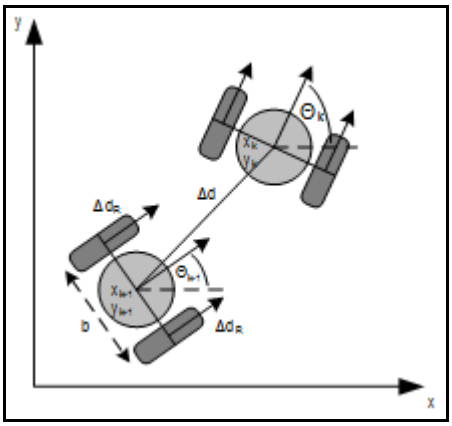

Figura 2 Odometria.

Lo sviluppo di questo algoritmo è iniziato nel 1986 e fino al 2004 (*classical age)* visto principalmente la creazione e l'ottimizzazione degli strumenti matematici alla base dello SLAM: *extended Kalman filters,* , *Rao–Blackwellized particle filters e infine maximum likelihood estimation.* Successivamente dal 2004 a oggi la comunità scientifica si è focalizzata più sullo studio dell'algoritmo vero e proprio *(algorithmic-analysis age)* [2]. Con il recente sviluppo di altre tecnologie lo SLAM ha subito l'influenza di altre discipline scientifiche come la *computer vision* e il *signal processing* rendendo le ricerche in questo campo ancora più complesse e multidisciplinari. Attualmente i limiti di questa procedura sono: la presenza di *outliers,* ovvero di punti delle dimensioni di alcuni pixel che presentano una qualità inferiore rispetto a quella necessaria per l'utilizzo, la presenza nell'ambiente di geometria troppo complesse che non permettono all'algoritmo di poterle estrapolare e la necessità di apparecchiature costose per quanto riguarda la sensibilità dei sensori e la velocità di computazione dei calcolatori.

1.2 ORB-SLAM

L'ORB-SLAM è una soluzione per il Simultaneous Localization And Mapping versatile e precisa ed è applicabile a fotocamere monoculari, stereoscopiche e RGB-D. Il termine ORB è un acronimo che sta per *Oriented fast and rotated brief,* ovvero una tecnica locale che permette di estrapolare le geometrie principali delle immagini. Le caratteristiche principali di questo algoritmo sono: è invariante alla rotazione e resistente al rumore, inteso come errore nella associazione dei dati. Inoltre nell'articolo [3] è dimostrato come esso sia molto più veloce rispetto al SIFT, un algoritmo molto utilizzato precedentemente. Il SIFT (*Scale-invariant feature transform*), progettato nel 1999 da David G. Lowe, permette anch'esso di rilevare ed estrarre alcuni punti interessanti dalle immagini nonostante queste ultime, cambiando visuale, varino la scala e l'illuminazione. Entrambi questi algoritmi utilizzano la posizione relativa dei punti rispetto ad altre strutture nelle immagini; qualora queste *feature* modifichino anche la loro geometria il riconoscimento fallirebbe.

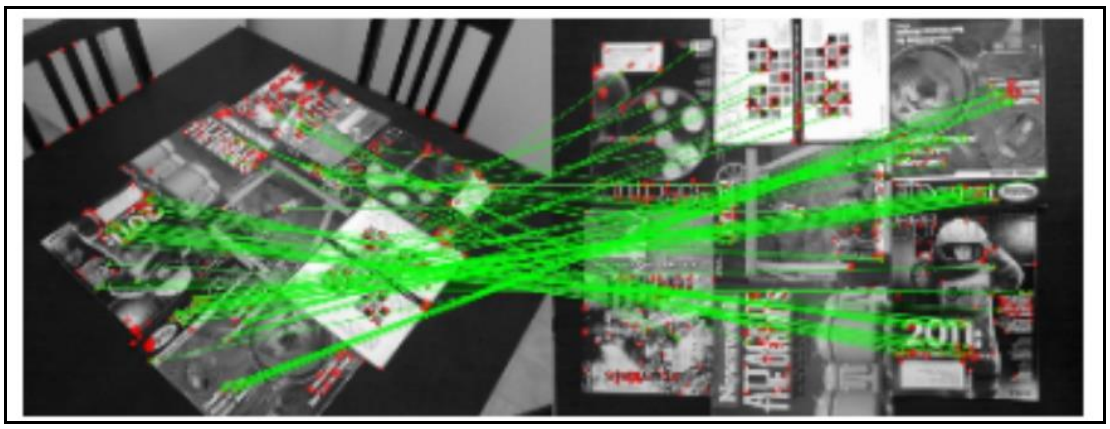

Figura 3 Estrapolazioni delle features dalle immagini tramite l'algoritmo ORB-SLAM.

La necessità di sostituire l'algoritmo SIFT non è nata principalmente da questi motivi ma dal fatto che non era applicabile in un gran numero di situazioni reali perché necessita una grande capacità di calcolo, di tempo di esecuzione e di corrente elettrica in termini energetici. Per i suddetti motivi non poteva essere applicata a *real-time system* e dispositivi dotati di batterie modeste come ad esempio i cellulari.

Capitolo 2

Codice Matlab

In questo capitolo è descritta passo per passo la procedura ORB-SLAM, la quale verrà suddivisa in quattro fasi principali: map initialization, tracking, local mapping e loop closure. Ognuna di queste fasi è strettamente legata a quella precedente, infatti il codice non solo svolge alcune operazioni in parallelo ma presenta anche una struttura ricorsiva. Inoltre è necessario, per avere una migliore comprensione dell'algoritmo, definire il significato di alcuni termini che verranno utilizzati spesso nel corso della descrizione della procedura:

- *Key frames*: immagini che contengono informazioni rilevanti per la localizzazione della fotocamera e per la definizione dell'ambiente; solitamente si tratta di oggetti con geometrie molto semplici e pronunciate come ad esempio degli spigoli.
- *Map points*: un insieme di punti in 3 dimensioni che rappresenta l'ambiente in cui si muove la fotocamera.
- *Feature*: caratteristica geometrica o cromatica dell'immagine che permette di distinguere un frame rispetto ad un altro.

La procedura si basa sull'inizializzazione di una *map points* provvisoria a partire da due frames con sufficiente cambio di visuale. Dopodiché ogni frame viene analizzato in maniera ricorsiva: nella fase di *tracking* vengono estrapolate le features e viene ipotizzata una posizione della fotocamera rispetto all'ambiente, confrontando il frame corrente con la mappa inizializzata precedentemente. Se il frame viene identificato come *key frame* allora nella fase di *local mapping* quest'ultimo viene utilizzato per aggiornare e ottimizzare la *map points*.

Infine nella fase *loop closure*, l'algoritmo cerca di stabilire se la camera abbia fatto un giro completo ritornando su un punto precedentemente acquisito. Qualora ciò accada vengono ottimizzate la posizione e la traiettoria della camera, le quali sono poi rappresentante su un grafico definitivo.

L'algoritmo ORB-Slam può essere illustrato come in figura[5]:

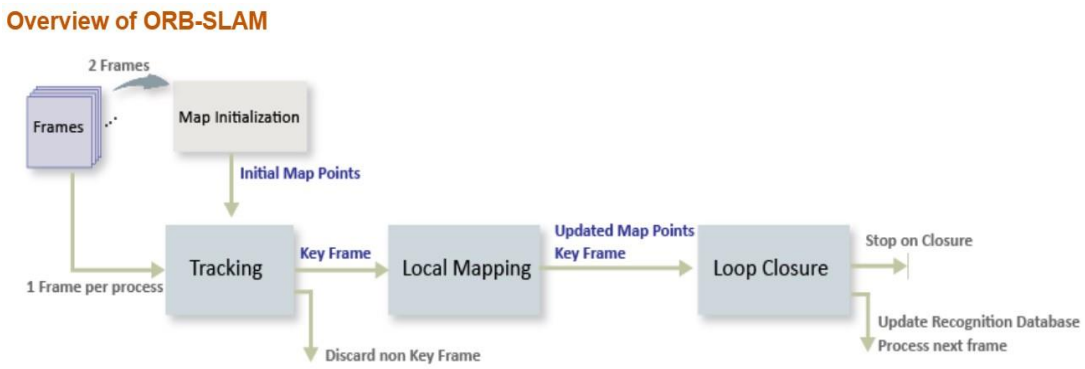

Figura 4 Diagramma Algoritmo ORB-Slam

Un altro fondamentale aspetto dello SLAM è la modalità con cui vengono estrapolate le features dalle immagini, poiché una cattiva rilevazione dei punti chiave detti "landmarks" costituisce una fonte cospicua di errori e di inaccuratezza. In questa trattazione è stato usato un algoritmo ORB, acronimo del termine *Oriented Fast and rotated brief,* ovvero uno strumento locale di rilevazione delle features presentato in [1].

2.1 Map initialization

Il processo preliminare di creazione della mappa di punti in tre dimensioni è fondamentale per l'accuratezza del risultato infatti è proprio grazie a questa fase che si determina la posizione inziale delle fotocamera e si inizializza la map points.

Nella prime righe di codice viene stabilito il percorso delle cartelle che verranno utilizzate, è importante definire quindi una cartella in cui saranno presenti in formato .mat sia le *funzioni helper*, la cui descrizione avverrà nel paragrafo 2.5 sia il *main,* ovvero la struttura principale del codice.

Inoltre vengono definiti i parametri intrinseci della fotocamera che vengono stimati con la calibrazione, come: la lunghezza focale e il punto di intersezione tra il piano dell'immagine e la retta di osservazione detto `principal point`. Questi parametri insieme alla risoluzione dell'immagine, che viene estrapolata in automatico, sono archiviati nella variabile *`intrincs.`*

In seguito grazie alla funzione ***helperDete*ctAndExtractFeatures** vengono rilevate le corrispondenze tra due frames con sufficiente cambio di visuale, l'algoritmo poi indicizza le features utilizzando la funzione `matchFeatures` che ha come input due parametri:

- `Matchtreshold:` la percentuale minima di punti per definire una solida corrispondenza tra due immagini. Il valore è compreso tra 0 e 100; in questo caso è stato utilizzato 70.

- `Maxratio:` la proporzione geometrica definita dal rapporto di forma.

```
% Map initialization loop
while ~isMapInitialized && hasdata(imds)
    currI = readimage(imds, currFrameIdx);

    [currFeatures, currPoints] = helperDetectAndExtractFeatures(currI,scaleFactor,...
                                        numLevels_DetectAndExtract, numPoints);

    currFrameIdx = currFrameIdx + 1;

    % Find putative feature matches
    indexPairs = matchFeatures(preFeatures, currFeatures, 'Unique', true, ...
        'MaxRatio', 0.7, 'MatchThreshold', 70);

    % If not enough matches are found, check the next frame
    minMatches = 100;
    if size(indexPairs, 1) < minMatches
        continue
    end

    preMatchedPoints  = prePoints(indexPairs(:,1),:);
    currMatchedPoints = currPoints(indexPairs(:,2),:);
```

Codice 1 Ciclo while - Map initialization (righe 115-131)

In questo ciclo while se il numero di features trovate tra le due immagini è maggiore della soglia minima `minMatches=100` si forma una nuvola di punti provvisoria altrimenti l'algoritmo passa al frame successivo. Qualora si riuscisse a definire una relazione tra due frames la mappa viene inizializzata e compare la seguente finestra:

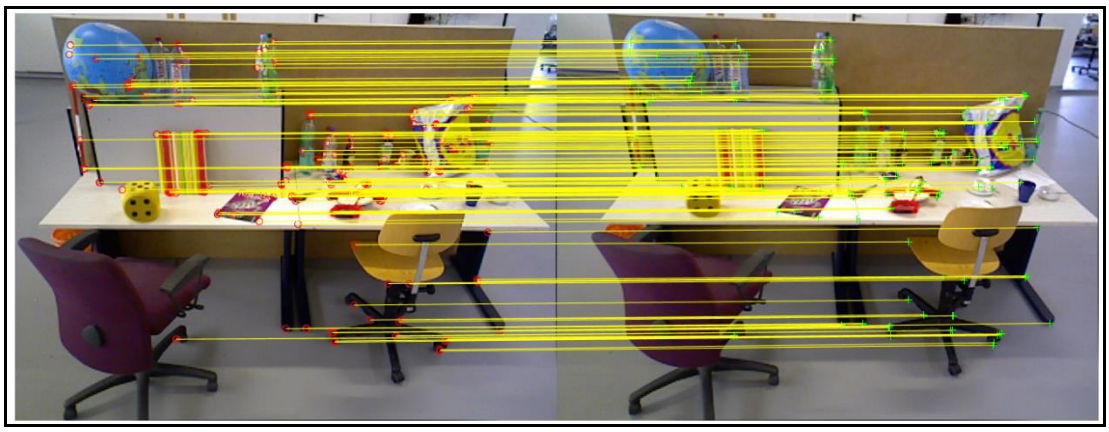

Figura 5 Riconoscimento delle features - Map initialization

Definita la corrispondenza tra le due immagini vengono utilizzati due modelli geometrici di trasformazione per localizzare i punti sulle immagini e proiettarli in uno spazio cartesiano 3-D. Se la scena è geometricamente planare si usa la funzione `estimateGeometricTransform` basata sul principio dell'omografia che permette di creare una relazione tra le coordinate dei punti in uno spazio a tre dimensioni e la loro proiezione attraverso un foro stenopeico. Diversamente se la scena non è planare si usa la funzione `estimateFundamentalMatrix.`

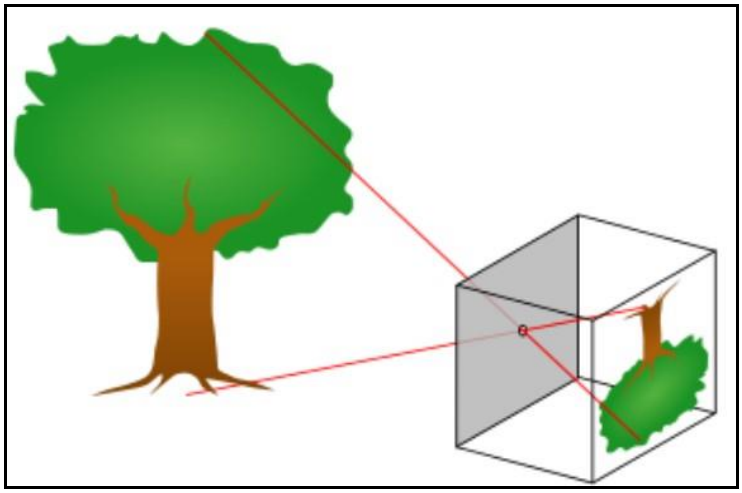

Figura 6 Proiezione attraverso foro stenopeico

Conseguentemente l'inizializzazione della mappa viene ultimata stimando la rotazione e la traslazione relativa della fotocamera tramite la funzione `relativeCameraPose` e localizzando nello spazio i punti trovati dal confronto tra i due frames attraverso la funzione `triangulate.` Bisogna inoltre prestare attenzione al fatto che le immagini sono state scattate da una fotocamera monoculare, la quale a differenza di una visione stereo non ci dà alcuna informazione riguardo la profondità. Per tale motivo è possibile stimare la traslazione della fotocamera solo attraverso un opportuno fattore di scala.

Dopo aver creato la map points si utilizzano `imageviewset` e `helperMapPointSet` per memorizzare i due key frames e la nuvola di punti ottenuta. Per ottenere una ricostruzione migliore dell'ambiente e della posizione della fotocamera si usa la funzione **bundleAdjustment** che permette di correggere errori di proiezione e geometrici come quelli di parallasse [3]

Infine il tutto viene mostrato all'utente tramite un visualizzatore in tempo reale dello stato della procedura. In figura sono illustrate in rosso la posizione della fotocamera e in blu e giallo i punti estrapolati.

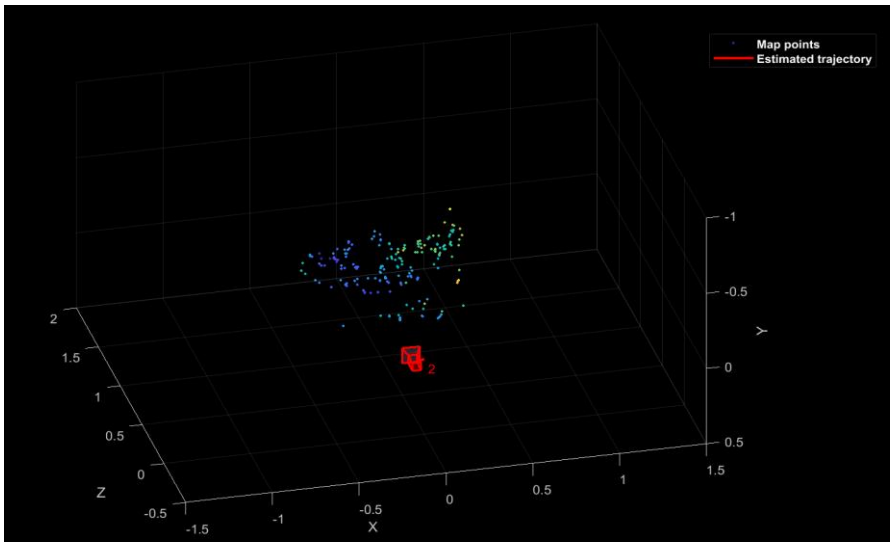

Figura 7 Map points e Fotocamera

2.2 Tracking

I processi di tracking, local mapping e loop closing sono eseguiti dal programma in parallelo e costituiscono la parte iterativa del codice. Il tracking ha lo scopo di localizzare la fotocamera in ogni frame e di decidere quando inserire un nuovo keyframe [3].

Ogni immagine viene processata secondo lo schema seguente:

1. Vengono estrapolate le ORB-feature per ogni frame e poi confrontate con le features dell'ultimo keyframe tramite la funzione `helperDetectAndExtractFeatures` la quale ha come input due parametri di natura geometrica (l'immagine è suddivisa in 8 livelli di scala `numLevels` con un fattore di scala `scaleFactor` di 1.2) e uno dipendente dalla risoluzione dell'immagine (per una risoluzione 480x640 vengono estratti 1000 punti `numPoints)`. Qualora si volesse aumentare la risoluzione è necessario modificare quest'ultimo valore ad almeno 2000 punti. Per ottenere una buona distribuzione dei punti estrapolati i livelli di scala sono divisi in griglie, ognuna delle quali deve contenere almeno 5 punti, evento non sempre possibile in quanto una mancanza di contrasto o di geometrie riconoscibili potrebbe non far rilevare alcun punto.

   A differenza di altre versione di slam, l'estrapolazione dei punti detta *fast corners* è molto più efficace e non necessita di grandi calcoli computazionali.

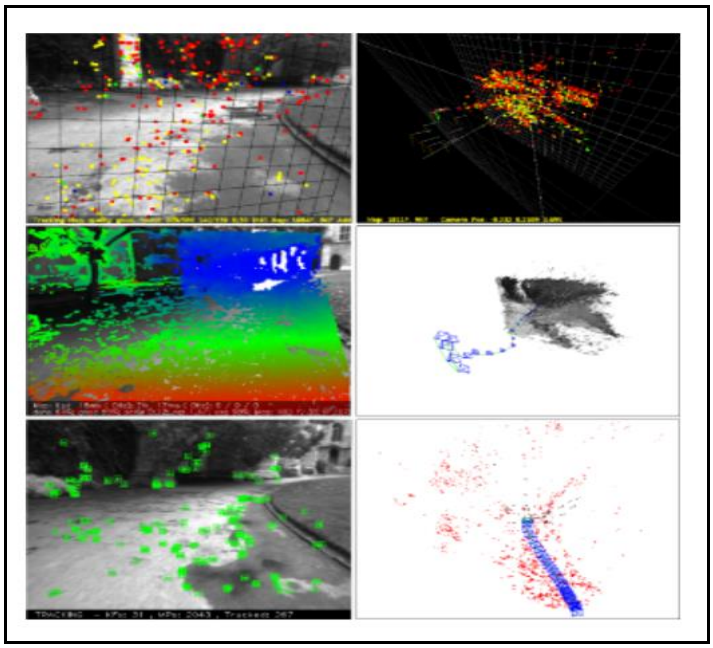

Figura 8 In ordine dall'alto verso il basso tre versioni di SLAM differenti : PTAM, LSD-SLAM e ORB-SLAM. Solo con l'ultimo algoritmo la mappa viene inizializzata in maniera consona, nonostante la scena non fosse perfettamente planare.

2. La posizione della fotocamera viene stimata tenendo conto di due possibili situazioni: il frame in questione presenta un buon numero di features in comune con quello precedente oppure se ciò non accade si parla di *motion model violated.* Nel primo caso il tracking ha buon esito, quindi viene usato un modello di spostamento a velocità costante della camera e vengono confrontati i punti rilevati con quelli del frame precedente per aggiornare la map points tramite la funzione *helperMatchFeaturesInRadius*. Nel secondo caso invece bisogna cambiare approccio: è necessario comparare il frame corrente con il database in cui sono archiviati tutti i key frames (tramite il metodo *bag of words* che verrà illustrato nel paragrafo 2.4), cercando di rilocalizzare la fotocamera rispetto all'ambiente grazie a un algoritmo non

iterativo chiamato *Perspective-n-Point* presente nella funzione *estimateWorldCameraPose.* Questo algoritmo è stato ideato da Vincent Lepetit, Francesc Moreno-Noguer e Pascal Fua presso il laboratorio di Computer Visione dell' École Polytechnique Fédérale de Lausanne (EPFL) [6] e consiste nel stimare la posizione della fotocamera partendo dalle corrispondenze tra punti 3D-to-2D (la soluzione prevede un complessità di calcolo lineramente crescente con il numero di punti). Infine grazie alla funzione *bundleAdjustmentMotion* viene migliorata la precisione della vista estrapolata.

3. Viene proiettata la nuvola di punti sul frame corrente per trovare delle somiglianze; siccome spesso la map points si presenta come un conglomerato di punti tutt'altro che modesto si utilizza solo parte della mappa da qui il nome di questa fase: *Track local map.*

4. Nell'ultima fase del tracking l'algoritmo deve valutare se definire il frame corrente come un nuovo key frame. Questa scelta spesso crea dei key frames ridondanti pur di velocizzare il più possibile il processo, ma ciò non costituisce un problema per la procedura in quanto nella fase di *local mapping* tali frames verranno rianalizzati e eventualmenti scartati. Affinchè un key frame possa essere considerato un candidato valido si devono rispettare le seguenti condizioni: devono essere passati almeno 20 frames dall'ultima rilocazizzazione globale o dall'ultimo inserimento di un key frame, nel frame corrente inoltre devono essere stati rilevati almeno 50 punti e meno del 90% dei punti già presenti nel database composto dai key frame di riferimento.

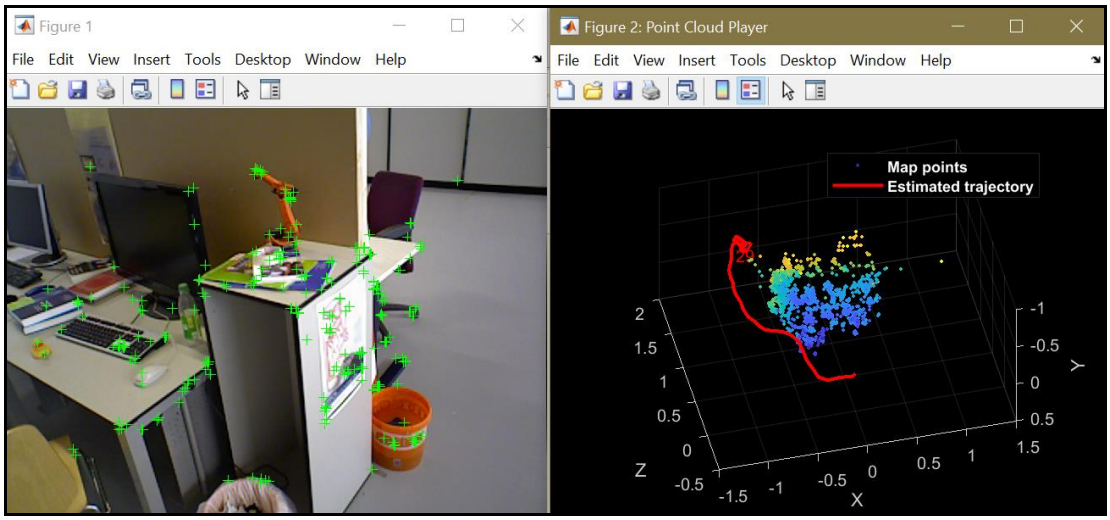

Figura 9 Illustrazione del processo di Tracking. A destra sono riportate sia la nuvola di punti sia la traiettoria della fotocamera in rosso.

2.3 Local mapping

Ogniqualvolta viene trovato un nuovo key frame il *covisibility graph* deve essere aggiornato. Questo grafico è costituito dai nodi che rappresentano i key frames, i quali vengono collegati due a due se i key frames condividono almeno 15 punti. Per migliorare la leggibilità del grafico solitamente ne viene creato un altro nel quale vengono rappresentanti solo i bordi tra due key frames che hanno in comune 100 punti che è denominato *essential graph.*

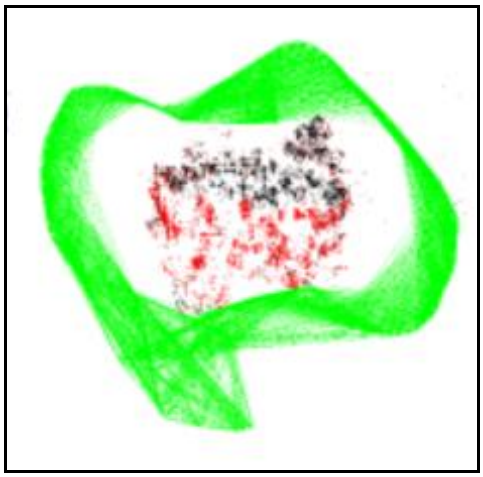

Figura 10 Covisibility Graph

Affinché la nuvola di punti abbia meno anomalie possibili, vengono eliminati i punti della mappa che sono rilevati da meno di 3 key frames tramite la funzione *helperCullRecentMapPoints.* Successivamente la nuova mappa di punti è aggiornata tramite la triangolazione delle ORB features contenute sia nel key frame corrente sia in quelli a lui connessi. In questo processo sono scartati i punti che non rispettano il vincolo della geometria epipolare, la parallasse e gli errori di riproiezione[7]. Per geometria epipolare si intende il modello della visione stereoscopica che permette di descrivere le relazioni e i vincoli geometrici tra due immagini 2D della stessa scena 3D, scattate da due fotocamere distinte[8].

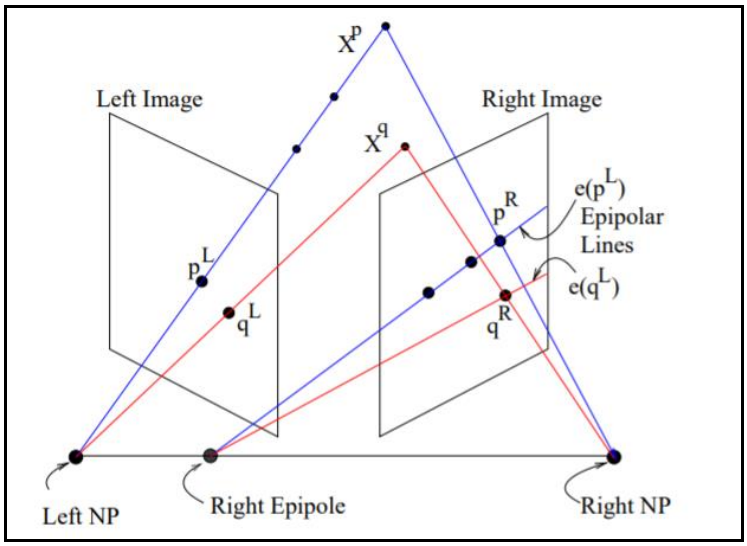

Figura 11 Schema Geometria Epipolare

Infine la funzione *helperLocalBundleAdjustment* migliora la posizione dei key frames, i bordi che li legano e i punti della mappa da loro osservati.

2.4 Loop Closure

Il processo di loop closure consiste nella capacità dell'algoritmo di riconoscere che la fotocamera sia ritornata in un punto precedentemente già visitato. E' una procedura estremamente critica che nello stato dell'arte attuale necessita ancora di perfezionamenti nonostante i numerosi approcci possibili. Nella seguente trattazione verrà utilizzato un approccio chiamato *bag of visual words,* il cui acronimo è BoVW benché le ultime ricerche siano propense all'adozione di un metodo basato sul *deep learning* in particolare sulle *convolutional neural networks* dette CNNs [8].

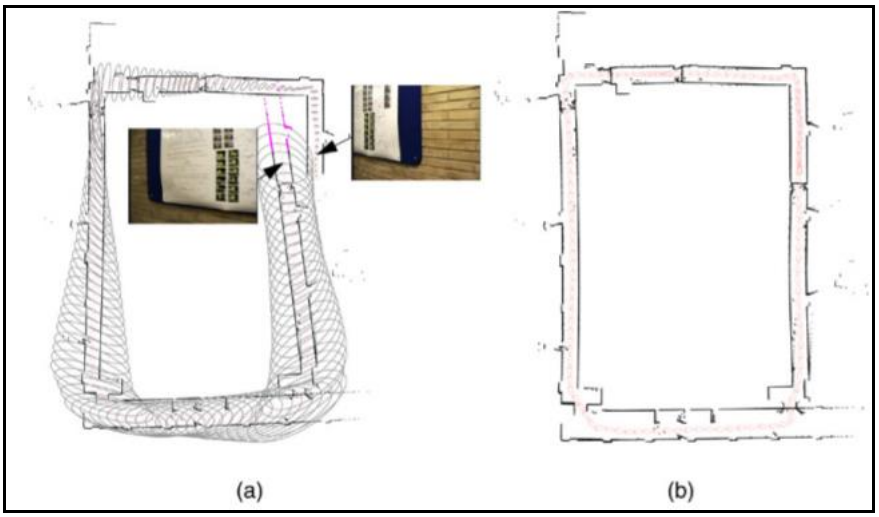

Figura 12 (a) Mappa prima del loop closure. (b) Traiettoria ricostruita dopo aver effettuato il loop closure. I triangoli rossi rappresentano le posizioni di un robot, mentre le ellissi verdi l'incertezza sulla traiettoria reale [10].

Il modello *bag of words* è una rappresentazione semplificata delle immagini usata in *natural language processing* e *information retrieval*. Questo approccio trova quindi utilità anche nella *computer vision*; i frames infatti possono essere classificati come documenti di testo in cui le caratteristiche delle immagini sono delle parole. Analogamente a una borsa che contiene degli oggetti in maniera disordinata, le bag of words sono vettori sparsi paragonabili a degli istogrammi archiviati in un database chiamato vocabolario [11].

In particolare il programma utilizzato da Matlab di cui la tesi è oggetto, utilizza una funzione chiamata *bagOfFeatures* che rappresenta un "vocabolario visivo" creato offline grazie a un ampio database di immagini, tramite la procedura *speeded up robust features* (SURF) che permette rilevare le features, classificare e ricostruire le immagini. Questa funzione ha inoltre come input il database che contiene le immagini necessarie per "allenare" l'algoritmo.

```
bag = bagOfFeatures(imds,'CustomExtractor', @helperSURFFeatureExtractorFunction);
```

Codice 2 Bag of features

La procedura quindi confronta le somiglianze tra il vettore *bag of words* del key frame i-esimo e tutti i key frames a lui vicini nel *covisibility graph*, dopodiché utilizza il database creato per escludere tutti i key frames ridondanti e quelli a lui direttamente connessi. Affinché un key frame sia considerato un candidato valido per definire un loop deve avere almeno altri 3 key frames consecutivi accettabili dalla procedura. Per chiudere un loop bisogna applicare una trasformazione geometrica di similitudine tra il key frame candidato per il loop e un key frame della traiettoria; ciò è possibile grazie al metodo iterativo di Horn [12], tenendo in considerazione che nel caso di *monocular Slam* si hanno 7 gradi di libertà: 3 di traslazione e di rotazione nello spazio e 1 dovuto al fattore di scala. Inoltre dalla trasformazione di similitudine è possibile stimare l'errore accumulato nel loop e confermare la correttezza della scelta del key frames candidato per il loop.

Infine nella fase di *Loop Fusion* vengono congiunti i punti duplicati e viene inserito un nuovo bordo che chiude la traiettoria percorsa dalla fotocamera presente nel *covisibility graph*.

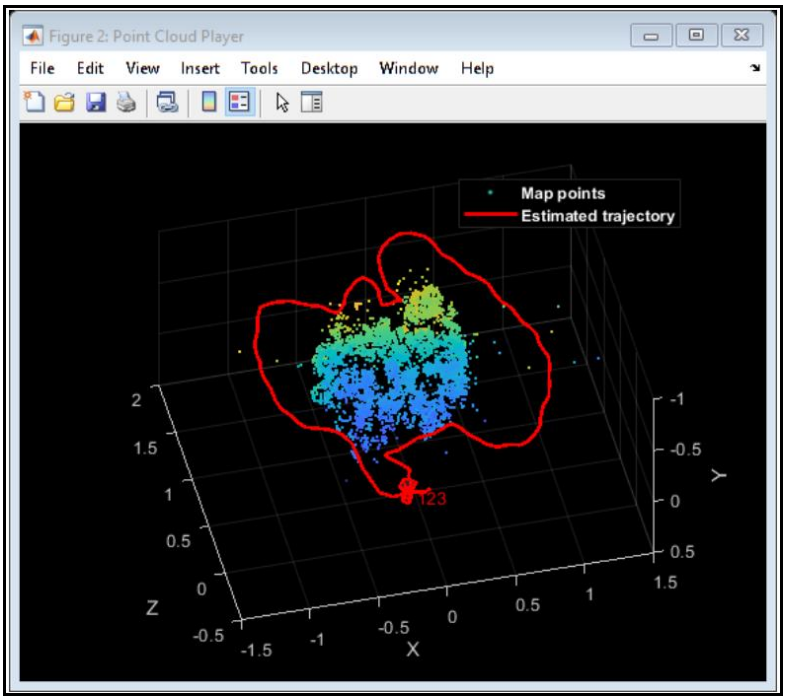

Figura 13 Inizio del processo di Loop Closure

2.5 Supporting Functions

In questo paragrafo sono descritte le funzioni più importanti presenti nel codice principale e che sono state precedentemente citate nella descrizione delle 4 fasi del programma.

2.5.1 helperDetectAndExtractFeatures

La funzione helperDetectAndExtractFeatures permette di rilevare e estrapolare le ORB features dalle immagini ed è stata utilizzata sia durante il processo di inizializzazione della mappa sia durante il processo di tracking.

```
function [features, validPoints] = helperDetectAndExtractFeatures(Irgb, varargin)
```

Codice 3 Struttura Funzione HelperDetectAndExtractFeatures

Tale funzione può essere adottata sia per immagini distorte sia per immagini non distorte; le immagini utilizzate in questo progetto rientrano nel secondo caso e per questo motivo non è stato utilizzato parte del codice che altrimenti sarebbe necessario.

```
if nargin > 1
    intrinsics = varargin{1};
end
Irgb  = undistortImage(Irgb, intrinsics);
```

Codice 4 Codice HelperDetectAndExtractFeatures per immagini distorte.

Le immagini in formato R vengono trasformate utilizzando sfumature di grigio in modo da accentuare le variazioni cromatiche e rendere le caratteristiche dell'immagine più facilmente rilevabili. Successivamente i punti sono estrapolati tramite la funzione *detectORBfeatures* che ha come input le immagini in tonalità di grigio, il fattore di scala e i livelli di scala. I punti trovati poi vengono selezionati in modo da avere una distribuzione uniforme e sono utilizzati per estrarre le ORB features.

2.5.2 helperTriangulateTwoFrames

Anche questa funzione ha un ruolo importante nella fase di map initialization, siccome permette di triangolare due frames per creare la mappa di punti. Dopo aver stimato la posizione e le corrispondenze tra due frames la funzione *triangulate* identifica la posizione dei punti 3D. Un punto triangolato è considerato valevole se

appare in due frames, se l'errore di riproiezione è basso e se la parallasse delle due viste è sufficientemente ampia.

```
function [isValid, xyzPoints, inlierIdx] = helperTriangulateTwoFrames(...
    pose1, pose2, matchedPoints1, matchedPoints2, intrinsics)
```

Codice 5 Funzione HelperTriangulateTwoFrames.

La funzione utilizza i parametri intrinseci della fotocamera, la posizione e i punti rilevati dei due frames per stabilire la posizione dei punti e la presenza di *inlier.* La definizione di *inliers* e *outliers* non viene spiegata in maniera esaustiva nel codice a disposizione per tale motivo si è fatto riferimento a [12]. In questo articolo un outlier è definito come una porzione di frame a livello dei pixel non identificabile o un oggetto che appare all'improvviso in un frames la cui presenza non è prevista.

In seguito i punti sono filtrati tenendo conto della loro direzione e dell'errore di riproiezionie: in questo caso il *minReprojectionError* è uguale a 1 per cui tutti i punti con errore superiore non sono considerati validi. Un altro parametro di selezione come detto in precedenza è la parallasse che in questa trattazione è assunta uguale a 3.

2.5.3 helperEstimateTrajectoryError

La funzione helperEstimateTrajectoryError permette di confrontare la traiettoria stimata grazie al processo di tracking con quella reale che viene fornita direttamente da Matlab (groundtruth.txt). Questo paragone permette all'utente di capire l'accuratezza della procedura ORB-SLAM; l'errore commesso è calcolato in metri (m) attraverso RMSE ovvero l'errore quadratico medio.

L’RMSE è calcolato a partire dalla componente di traslazione dell’errore relativo di posizione ($trans E_i$) che misura la precisione della traiettoria stimata entro un certo intervallo Δ come illustra questa formula tratta da un articolo scientifico [14].

$$\mathrm{RMSE}(\mathbf{E}_{1:n}, \Delta) := \left(\frac{1}{m} \sum_{i=1}^{m} \| trans(\mathbf{E}_i) \|^2 \right)^{1/2}$$

Capitolo 3

Ottimizzazione del Codice

In questo capitolo sono riportate tutte le modifiche fatte al codice Matlab con il fine di perfezionare l'algoritmo e facilitare l'utente nelle impostazioni dei parametri. La procedura infatti è sviluppata per qualsiasi tipo di immagini anche per risoluzioni superiori a quelle utilizzate. Inoltre con delle semplici modifiche è possibile elaborare un video anziché un numero rilevante di immagini registrate sequenzialmente. Per i suddetti motivi si è ritenuto necessario estrarre tutte le costanti presenti nell'algoritmo e di posizionarle in testa al codice, in modo tale da raggruppare tutti gli input da fornire al codice e rendere più facile la modifica di alcuni di essi come, ad esempio, la modifica della tipologia di fotocamera o delle soglieper la determinazione delle features. Difatti si è notato che variando alcune costanti si ottengono valori di RMSE diversi e pertanto è stata creata una tabella in cui riportare i risultati. Contrariamente a quanto ci si potesse aspettare si è notato che al variare del calcolatore utilizzato, nonostante i parametri rimanessero invariati, il risultato poteva essere leggermente diverso da quello ottenuto e per questo motivo sono state fatte delle prove con computer diversi. Infine negli ultimi due paragrafi è spiegato in maniera concisa come calcolare la traiettoria ottimizzata rispetto a quella reale e come rappresentare la nuvola di punti trovata in modo da poterla eventualmente migliorare escludendo i punti superflui.

3.1 Estrapolazione dei parametri

La necessità di estrapolare i parametri in input alle funzioni *helper* e nel codice principale è stata evidente fin da subito in quanto anche per semplici modiche dell'algoritmo era necessario ricercare le costanti all'interno del codice avendo come risultato una notevole perdita di tempo. In aggiunta il gran numero di applicazioni di

questo algoritmo richiede un continuo settaggio delle costanti, che spesso devono essere adattate in base alle caratteristiche intrinseche delle fotocamera, come ad esempio la risoluzione, che in questo caso era 480x640 e al tipo di ambiente che si deve mappare.

L'ottimizzazione quindi comprende una ricerca iniziale delle costanti all'interno del codice, le quali successivamente sono state commentate e poi inserite come input all'interno delle funzioni principali. Di seguito sono illustrati alcuni esempi tratti dalla funzione *helperDetectAndExtractFeatures:*

```
function [features, validPoints] = helperDetectAndExtractFeatures(Irgb, varargin)

scaleFactor = 1.2;
numLevels   = 8;
numPoints   = 1000;
```

Codice 6 Codice prima dell'estrapolazione dei parametri dalla funzione HelperDetectAndExtractFeatures.

```
function [features, validPoints] = helperDetectAndExtractFeatures(Irgb, scaleFactor,...
    numLevels_DetectAndExtract, numPoints, varargin)

% scaleFactor = 1.2;
% numLevels   = 8;
% numPoints   = 1000;
```

Codice 7 Estrapolazione dei parametri dalla funzione HelperDetectAndExtractFeatures.

I problemi principali riscontrati in questa procedura sono stati sostanzialmente tre:

1. In quasi tutte le funzioni sono presenti delle sotto-funzioni che a loro volta hanno come input dei parametri comuni ad altri *helper*. Ad esempio nella funzione helperTrackLocalMap è presente come sotto-funzione helperMatchFeaturesInRadius.

```
% Search radius depends on scale and view direction
searchRadius    = 4*ones(size(localFeatures, 1), 1);
searchRadius(viewAngles<3) = 2.5;
searchRadius    = searchRadius.*predictedScales;

indexPairs = helperMatchFeaturesInRadius(localFeatures, unmatchedFeatures, ...
    locaValidPoints, unmatchedValidPoints, projectedPoints, searchRadius, ...
    max(1, predictedScales/scaleFactor), predictedScales);
```

Codice 8 Funzione helperMatchFeaturesInRadius presente nella funzione principale helperTrackLocalMap.

2. E' necessario rinominare i parametri in quanto essi stessi hanno valori diversi a seconda della funzione in cui si trovano. In seguito sono riportati alcuni dei parametri posizionati all'inizio del codice principale e rinominati a seconda della loro posizione nell'algoritmo. Ad esempio la costante `minNumMatches` è uguale a 40 nel codice principale mentre nella funzione helperAddLoopConnections è uguale a 60, per tale motivo è stata rinominata `minNumMatches_AddLoop`.

```
% AddLoopConnection
minNumMatches_AddLoop = 60;
r_AddLoop = 3;
maxRatio_AddLoopConnection_matchFeatures = 0.9;
matchThreshold_AddLoopConnection_matchFeatures = 90;
Confidence_AddLoopConnection_estimateWorldCameraPose = 90;
MaxReprojectionError_AddLoopConnection_estimateWorldCameraPose = 6;
MaxIteration_AddLoopConnection_bundleAdjustmentMotion = 50;

% AddNewKeyFrame
coviThrehsold = 5;

% CheckLoopClosure
minNumCandidates = 3;
maxRange = 10;
NumResults_CheckLoopClosure_evaluateImageRetrieval = 10;
```

Codice 9 Elenco di alcuni paramentri presenti nel codice principale.

3. Qualora sia presente in input ad una funzione la variabile `varargin` è indispensabile posizionarla come ultimo input, altrimenti si incorre in un errore di sintassi quando si esegue il programma.

```
function [features, validPoints] = helperDetectAndExtractFeatures(Irgb, scaleFactor,...
    numLevels_DetectAndExtract, numPoints, varargin)
```

Codice 10 Variabile varargin come ultimo input della funzione.

3.2 Traiettoria e sensibilità RMSE

In questo paragrafo è descritto un aspetto ragguardevole della procedura: la precisione con cui l'errore viene espresso alla fine del loop closure. La traiettoria ottimizzata è confrontata con quella chiamata "ground truth" ovvero una traiettoria fornita dal programmatore che è stata misurata in maniera empirica quindi

considerabile con buona approssimazione simile a quella reale. Come descritto nel paragrafo 2.5.3 il valore dell'RMSE (errore quadratico medio o Root Mean Square) è calcolato tenendo conto dell'errore relativo di posizione della traiettoria, calcolata durante la fase di tracking rispetto a quella reale. Per tale motivo minore sarà l'RMSE, maggiore sarà l'accuratezza dell'algoritmo. Seguendo questa logica è importante trovare quell'insieme di valori per cui l'errore è minimizzato. Ciò è stato fatto modificando in maniera opportuna alcuni parametri presenti nel codice principale per poi eseguire numerose volte il programma. I risultati sono molteplici e sono stati riportati nella tabella che segue:

helperMatchFeaturesInRadius		helperTrackLastKeyFrame		
matchThreshold_MatchFeaturesInRadius	maxRatio_MatchFeaturesInRadius	matchThreshold_TrackLastKeyFrame_matchFeatures	maxRatio_TrackLastKeyFrame_matchFeatures	Results
100	1,5	30	0,9	errore estimateworldcamerapose
100	0,8	30	0,9	(m)=0.266
100	1	30	0,9	errore estimateworldcamerapose
100	0,6	30	0,9	errore estimateworldcamerapose
80	1,5	30	0,9	errore estimateworldcamerapose
60	1,5	30	0,9	errore estimateworldcamerapose
30	1,5	30	0,9	errore estimateworldcamerapose
30	1	30	0,9	errore estimateworldcamerapose
30	0,8	30	0,9	errore estimateworldcamerapose
30	0,5	30	0,9	exceeds array bounds 20000
60	0,5	30	0,9	errore estimateworldcamerapose
80	0,5	30	0,9	errore estimateworldcamerapose
80	0,8	30	0,9	(m)=1,135
120	0,8	30	0,9	(m)=0,513
160	0,8	30	0,9	errore estimateworldcamerapose
160	1,5	30	0,9	MinNumMatches results in disconnect graph
100	0,8	50	0,9	errore calcolo traiettoria approssiamata
100	0,8	70	0,9	(m)=1,89
100	0,8	100	0,9	(m)=1,89
100	0,8	30	0,7	errore estimateworldcamerapose
100	0,8	30	0,5	errore estimateworldcamerapose
100	0,8	60	0,7	errore estimateworldcamerapose
100	0,8	10	0,7	errore estimateworldcamerapose

Figura 14 Sensibilità RMSE 1: in verde sono stati evidenziati i risultati accettabili, mentre in giallo quelli più accurati.

All'inizio sono stati modificati 4 parametri:
`matchThreshold_MatchFeaturesInRadius` e `maxRatio_MatchFeaturesInRadius` presenti nella funzione helperMatchFeaturesInRadius e `matchThreshold_TrackLastKeyFrame_matchFeatures` e `maxRatio_TrackLastKeyFrame_matchFeatures` presenti nella funzione helperTrackLastKeyFrame. I primi due riguardano la precisione con cui è settata la

funzione che permette di rilevare le features nelle immagini entro un certo raggio mentre gli altri due descrivono la sensibilità con cui l'algoritmo nella fase di tracking stima la posizione della fotocamera rispetto all'ultimo key frame. Da questa ipotesi si è ottenuto un RMSE in metri di 0,266.

Successivamente mantenendo costanti i valori per cui si aveva l'errore quadratico medio minore sono stati modificati altri parametri delle due funzioni citate precedentemente:

helperTrackLastKeyFrame	
Confidence_TrackLastKeyFrame_estimateWorldCameraPose	
50	errore estimateworldcamerapose
110	errore deve essere minore di 100
r_TrackLastKeyFrame	
7	errore imageviewset
2	(m)=2.78
MaxReprojectionError_TrackLastKeyFrame_estimateWorldCameraPose	
7	errore estimateworldcamerapose
2	errore estimateworldcamerapose
MaxIteration_TrackLastKeyFrame_bundleAdjustmentMotion	
40	(m)=0.166
10	errore estimateworldcamerapose
70	errore estimateworldcamerapose

helperTrackLocalMap	
numLevels_TrackLocalMap	
8	errore estimateworldcamerapose
ScaleMinDistance_TrackLocalMap_removeOutlierMapPoints	
1,2	errore la traiettoria entra dentro la nuvola punti
0,5	errore la traiettoria entra dentro la nuvola punti
ScaleMaxDistance_TrackLocalMap_removeOutlierMapPoints	
1,4	(m)=0.259
1,8	errore estimateworldcamerapose
0,8	(m)=0.178
0,4	errore world image
MaxIteration_TrackLocalMap_bundleAdjustmentMotion	
40	(m)=0.11
70	errore estimateworldcamerapose

Figura 15 Sensibilità RMSE 2.

Come è possibile notare il valore più basso si è raggiunto con il parametro `MaxIteration_TrackLocalMap_bundleAdjustmentMotion` = 40 con il quale si ha un RMSE di 11 centimetri che è un risultato soddisfacente vista la qualità dell'attrezzatura utilizzata. Inoltre è possibile notare come l'algoritmo sia molto sensibile alle variazioni dei settaggi in quanto molti dei risultati ottenuti sono errori che compromettono il funzionamento dello SLAM.
Infine si è constatato, contrariamente a quanto si potesse pensare, che utilizzando un calcolatore diverso ma mantenendo invariate le impostazioni e la versione di Matlab si hanno risultati diversi in termini di RMSE. La prova è stata fatta con due computer

con potenza di calcolo diversa: il primo un ASUS VIVOBOOK S512JP-EJ032T con processore Intel Core i7, 16,00 GB di ram e scheda video NVIDIA GeForce MX330; il secondo un LENOVO IDEAPAD 320-15IKB con processore Intel Core i5 e 8,00 GB di ram e scheda grafica Intel UHD Graphics 620. Nel primo caso come detto in precedenza, il risultato è stato (m) = 0,11 mentre per quanto riguarda il LENOVO si ha un (m) = 1.884 che è possibile diminuire a (m) = 0,090 con un valore del parametro `MaxIteration_TrackLocalMap_bundleAdjustmentMotion` pari a 20.

Per concludere è necessario portare all'attenzione del lettore un altro errore ricorrente riguardo la traiettoria finale. Sporadicamente alla fine dell'esecuzione del programma, la procedura mostra un problema sul calcolo della traiettoria finale: la traiettoria visibilmente sbagliata è di colore viola ed è all'interno di uno sfondo bianco. In questi casi è necessario solo rieseguire il programma senza dover modificare alcun parametro.

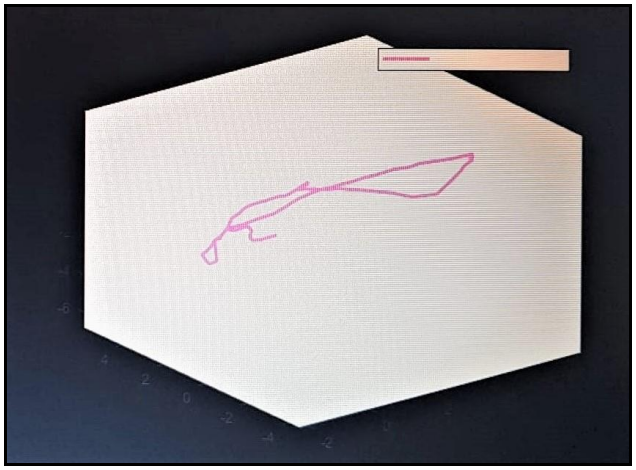

Figura 16 Errore calcolo traiettoria.

3.3 Nuvola di punti

In questo ultimo paragrafo è descritto come viene mostrato il risultato finale dell'algoritmo ORB-SLAM e di come è possibile rappresentare la mappa di punti trovata. La figura a sinistra mostra la nuvola di punti finale con attorno le tre traiettorie: in rosso quella stimata, in verde quella reale fornita dal programmatore e in viola quella ottimizzata dall'algoritmo. L'immagine a destra invece è stata inserita a posteriori e fa parte di quella logica di ottimizzazione del programma che rende più facilmente accessibile all'utente i dati in output dell'algoritmo. La figura a destra infatti ripropone la nuvola di punti trovata senza la presenza delle traiettorie e definendo delle dimensioni maggiori degli assi per avere una visione di insieme migliore.

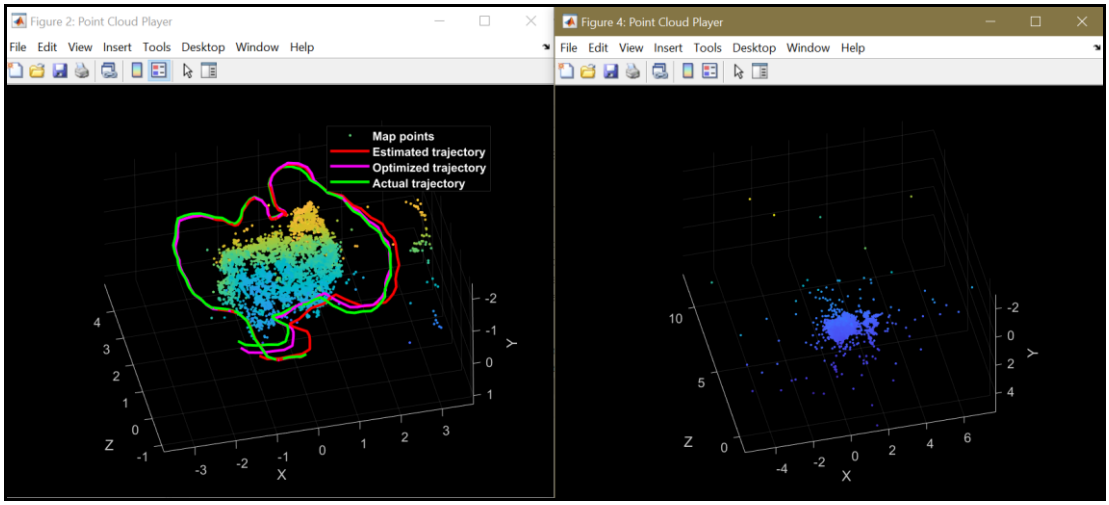

Figura 17 Risultato finale algoritmo ORB-SLAM a sinistra e proiezione della nuvola di punti a destra.

Per la proiezione della nuvola di punti è stato utilizzato il comando `pcplayer` che necessita come variabili in input la definizioni dei 3 assi cartesiani (con la possibilità

di indicarne anche la direzione) e se si vuole, lo spessore dei punti per renderli più visibili tramite il comando `markersize`.

Infine è utilizzata la funzione `pointCloud` che permette di trasformare i dati presenti nelle colonne della tabella allocata in `mapPointSet.Locations` in coordinate 3-D che poi possono essere rappresentate nel grafico precedente.

```
%% plot point cloud

player1 = pcplayer(XLim, YLim, ZLim, ...
                'VerticalAxis', 'y', 'VerticalAxisDir', 'down', 'MarkerSize', 30);
xlabel('X'),ylabel('Y'),zlabel('Z')
ptCloud = pointCloud(mapPointSet.Locations);
view(player1,ptCloud);
```

Figura 18 Codice per l'illustrazione finale della nuvola di punti.

Conclusione

In questo lavoro sono state analizzate le funzionalità e l'accuratezza di un sistema *monocular ORB-SLAM,* il quale nonostante la sua complessità rappresenta uno strumento molto utile nella rilevazione di scene a tre dimensioni e nella identificazione della traiettoria compiuta dalla fotocamera. Sicuramente la procedura presenta delle complicazioni non facilmente risolvibili, come ad esempio la difficoltà nell'estrapolazione delle features in ambienti in cui sono presenti delle geometrie particolari e le caratteristiche cromatiche degli oggetti non sono facilmente distinguibili. Tuttavia è bene ricordare che si tratta di un codice *open source* accessibile a chiunque voglia iniziare ad approcciare la teoria dello SLAM e che i risultati ottenuti rappresentano una conferma in termini positivi delle prestazioni di tale algoritmo.

Le problematiche principalmente riscontrate nell'apprendimento della struttura del codice sono: la chiarezza con cui sono stati definiti i parametri dell'algoritmo, la presenza di errori sporadici come quelli illustrati nel paragrafo 3.2 e l'incongruenza dei risultati qualora si usino due calcolatori diversi. Tuttavia la nuvola di punti trovata è conforme con la realtà e l'errore quadratico medio presenta dei valori assolutamente plausibili.

E' giusto riflettere anche sulla scelta di questa procedura piuttosto di una che utilizza una componentistica *hardware* diversa. La possibilità di utilizzare immagini scattate da un comune cellulare, senza dover disporre come nel caso del Lidar SLAM di sensori laser, rende la sperimentazione accessibile a chiunque. Per quanto riguarda invece il *software* in questo progetto si è utilizzato un algoritmo ORB piuttosto che uno SIFT o PTAM, la scelta in questo caso non è da attribuire principalmente ai vantaggi in termini di accuratezza ma alla reperibilità del codice stesso, il quale essendo presente sulla piattaforma Mathworks ne ha facilitato lo studio.

Sicuramente nei prossimi anni l'attenzione della comunità scientifica e dei ricercatori non verrà distolta dallo SLAM, in quanto il suo impiego nella robotica e nella realtà aumentata rappresenta una fonte inestimabile di possibilità. A conferma di ciò è possibile trovare *on-line* numerose varianti dell'algoritmo trattato in questa tesi e molteplici articoli scientifici su Reaserchgate, la maggior parte dei quali apre a nuove prospettive della ricerca in questo campo.

Bibliografia

[1] S. R. a. M. R. Blas, «SLAM for Dummies, a Tutorial Approach to Simultaneous Localization and Mapping,» [Online]. Available: https://web.archive.org/web/20111215093112/http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam_blas_repo.pdf

[2] L. C. H. C. Y. L. D. S. J. N. I. R. Cesar Cadena, «Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age,» december 2016. [Online]. Available: http://rpg.ifi.uzh.ch/docs/TRO16_cadena.pdf

[3] E. Rublee, V. Rabaud, K. Konolige e G. (. Bradski, «ORB: an efficient alternative to SIFT or SURF,» 2011. [Online]. Available: https://www.researchgate.net/publication/221111151_ORB_an_efficient_alternative_to_SIFT_or_SURF.

[4] MathWorks, «Monocular Visual Simultaneous Localization and Mapping,» 2019. [Online]. Available: https://it.mathworks.com/help/vision/examples/monocular-visual-simultaneous-localization-and-mapping.html.

[5] R. Mur-Artal, J. M. M. Montiel e J. D. Tardós, «ORB-SLAM: a Versatile and Accurate Monocular SLAM System,» [Online]. Available: https://www.arxiv-vanity.com/papers/1502.00956/.

[6] V. Lepetit, F. Moreno-Noguer e P. Fua, «EPnP: Accurate Non-Iterative O(n) Solution to the PnP Problem,» [Online]. Available: https://upcommons.upc.edu/bitstream/handle/2117/10327/moreno_ijcv2009.pdf.

[7] D.-W. Shin, «Introductory Level SLAM Seminar,» 18 gennaio 2018. [Online]. Available: https://www.slideshare.net/DongWonShin4/introductory-level-of-slam-seminar.

[8] A. J. e. D. Fleet, «Epipolar Geometry,» [Online]. Available: http://www.cs.toronto.edu/~jepson/csc420/notes/epiPolarGeom.pdf.

[9] X. Zhang, Y. Su e X. Zhu, «Loop closure detection for visual SLAM systems using convolutional neural network,» september 2017. [Online]. Available: https://www.researchgate.net/publication/320824241_Loop_closure_detection_for_visual_SLAM_systems_using_convolutional_neural_network.

[10] K. L. Ho e P. N. -. O. R. R. Group, «Loop closure detection in SLAM by combining visual and spatial appearance,» march 2006. [Online]. Available: http://www.robots.ox.ac.uk/~mobile/Papers/VisualLoopClosureSLAM.pdf.

[11] J. S. e. A. Zisserman, «Efficient visual search of videos cast as text retrieval,» [Online]. Available: https://www.di.ens.fr/~josef/publications/sivic09a.pdf.

[12] B. K. P. Horn, «Closed-form solution of absolute orientation using unit,» november 1986. [Online]. Available: https://www.cvl.iis.u-tokyo.ac.jp/~oishi/Papers/Alignment/Horn_Closed-form_JOSA1987.pdf.

[13] David Hasler, Luciano Sbaiz, Sabine Susstrunk and Martin Vetterli, «Outlier Modeling in Image Matching,» march 2003. [Online]. Available: https://www.researchgate.net/publication/37414066_Outlier_Modeling_in_Image_Matching

[14] Jurgen Sturm, «A Benchmark for the Evaluation of RGB-D SLAM Systems,» ottobre 2012. [Online]. Available: https://www.researchgate.net/publication/261353760_A_benchmark_for_the_evaluation_of_RGB-D_SLAM_systems

Appendice

```
%% Monocular Visual Simultaneous Localization and Mapping
close all
clear all
clc
dataFolder      = 'C:\Users\simon\Desktop\SlamCast\';
%% set path

imageFolder   = [dataFolder,'data/rgb/'];
imds          = imageDatastore(imageFolder);
dirname = 'C:\Users\simon\Desktop\SlamCast\frames';
% Inspect the first image
currFrameIdx = 1;
currI = readimage(imds, currFrameIdx);
himage = imshow(currI);

a = dir([imageFolder '/*.png']);
num_Frames = size(a,1);

%% Image properties
imageSize    = [480 640];
scaleFactor  = 1.2;

% AddLoopConnection
minNumMatches_AddLoop = 60;
r_AddLoop = 3;
maxRatio_AddLoopConnection_matchFeatures = 0.9;
matchThreshold_AddLoopConnection_matchFeatures = 90;
Confidence_AddLoopConnection_estimateWorldCameraPose =
90;
MaxReprojectionError_AddLoopConnection_estimateWorldCamer
aPose = 6;
MaxIteration_AddLoopConnection_bundleAdjustmentMotion =
50;

% AddNewKeyFrame
coviThrehsold = 5;

% CheckLoopClosure
minNumCandidates = 3;
maxRange = 10;
NumResults_CheckLoopClosure_evaluateImageRetrieval = 10;

% ComputeFundamentalMatrix
```

```
outlierThreshold_FundamentalMatrix = 4;
DistanceThreshold_ComputeFundamentalMatrix_estimateFundam
ental = 0.01;

% ComputeHomography
outlierThreshold_Homography = 6;
MaxDistance_ComputeHomography_estimateGeometricTransform
= 4;
Confidence_ComputeHomography_estimateGeometricTransform =
90;

% CreateNewMapPoints
minParallax = 3;
MaxRatio_CreateNewMapPoints_matchFeatures = 0.9;
MatchThreshold_CreateNewMapPoints_matchFeatures = 40;

% DetectAndExtractFeatures
numLevels_DetectAndExtract   = 8;
numPoints   = 1000;

% IsKeyFrame
MapPoints_IsKeyFrame = 80;
PercentagePointsKeyFrame_IsKeyFrame = 0.9;

% LocalBundleAdjustment
maxError = 6;
MaxIteration_LocalBundleAdjustment_bundleAdjustment =
100;

% MatchFeaturesInRadius
matchThreshold_MatchFeaturesInRadius = 100;
maxRatio_MatchFeaturesInRadius       = 0.8;

% TrackLastKeyFrame
r_TrackLastKeyFrame = 4;
maxRatio_TrackLastKeyFrame_matchFeatures = 0.9;
matchThreshold_TrackLastKeyFrame_matchFeatures = 30;
Confidence_TrackLastKeyFrame_estimateWorldCameraPose =
90;
MaxReprojectionError_TrackLastKeyFrame_estimateWorldCamer
aPose = 4;
MaxIteration_TrackLastKeyFrame_bundleAdjustmentMotion =
20;

% TrackLocalMap
numLevels_TrackLocalMap = 8;
```

```
MaxIteration_TrackLocalMap_bundleAdjustmentMotion = 40;
ScaleMinDistance_TrackLocalMap_removeOutlierMapPoints =
0.8;
ScaleMaxDistance_TrackLocalMap_removeOutlierMapPoints =
1.2;

% TriangulateTwoFrames
minParallax = 3;
minReprojError = 1;

%% 1***Map Initialization***

% Set random seed for reproducibility
rng(0);

focalLength    = [535.4, 539.2];    % in units of pixels
principalPoint = [320.1, 247.6];    % in units of pixels
imageSize      = size(currI,[1 2]);  % in units of pixels
intrinsics     = cameraIntrinsics(focalLength,
principalPoint, imageSize);

% Detect and extract ORB features
[preFeatures, prePoints] =
helperDetectAndExtractFeatures(currI,scaleFactor,
numLevels_DetectAndExtract, numPoints);

currFrameIdx = currFrameIdx + 1;
firstI       = currI; % Preserve the first frame

isMapInitialized  = false;

% Map initialization loop
while ~isMapInitialized && hasdata(imds)
    currI = readimage(imds, currFrameIdx);

    [currFeatures, currPoints] =
helperDetectAndExtractFeatures(currI,scaleFactor,
numLevels_DetectAndExtract, numPoints);

    currFrameIdx = currFrameIdx + 1;

    % Find putative feature matches
    indexPairs = matchFeatures(preFeatures, currFeatures,
'Unique', true, ...
        'MaxRatio', 0.7, 'MatchThreshold', 70);
```

```
    % If not enough matches are found, check the next
frame
    minMatches = 100;
    if size(indexPairs, 1) < minMatches
        continue
    end

    preMatchedPoints  = prePoints(indexPairs(:,1),:);
    currMatchedPoints = currPoints(indexPairs(:,2),:);

    % Compute homography and evaluate reconstruction
    [tformH, scoreH, inliersIdxH] =
helperComputeHomography(preMatchedPoints,
currMatchedPoints, outlierThreshold_Homography,...

MaxDistance_ComputeHomography_estimateGeometricTransform,
Confidence_ComputeHomography_estimateGeometricTransform);

    % Compute fundamental matrix and evaluate
reconstruction
    [tformF, scoreF, inliersIdxF] =
helperComputeFundamentalMatrix(preMatchedPoints,
currMatchedPoints,...
        outlierThreshold_FundamentalMatrix,
DistanceThreshold_ComputeFundamentalMatrix_estimateFundam
ental);

    % Select the model based on a heuristic
    ratio = scoreH/(scoreH + scoreF);
    ratioThreshold = 0.45;
    if ratio > ratioThreshold
        inlierTformIdx = inliersIdxH;
        tform          = tformH;
    else
        inlierTformIdx = inliersIdxF;
        tform          = tformF;
    end

    % Computes the camera location up to scale. Use half
of the
    % points to reduce computation
    inlierPrePoints  = preMatchedPoints(inlierTformIdx);
    inlierCurrPoints = currMatchedPoints(inlierTformIdx);
    [relOrient, relLoc, validFraction] =
relativeCameraPose(tform, intrinsics, ...
```

```
        inlierPrePoints(1:2:end),
inlierCurrPoints(1:2:end));

    % If not enough inliers are found, move to the next
frame
    if validFraction < 0.7 || numel(size(relOrient))==3
        continue
    end

    % Triangulate two views to obtain 3-D map points
    relPose = rigid3d(relOrient, relLoc);
    [isValid, xyzWorldPoints, inlierTriangulationIdx] =
helperTriangulateTwoFrames(...
        rigid3d, relPose, inlierPrePoints,
inlierCurrPoints, intrinsics, minReprojError,
minParallax);

    if ~isValid
        continue
    end

    % Get the original index of features in the two key
frames
    indexPairs =
indexPairs(inlierTformIdx(inlierTriangulationIdx),:);

    isMapInitialized = true;

    disp(['Map initialized with frame 1 and frame ',
num2str(currFrameIdx-1)])
end % End of map initialization loop

if isMapInitialized
    close(himage.Parent.Parent); % Close the previous
figure
    % Show matched features
    hfeature = showMatchedFeatures(firstI, currI,
prePoints(indexPairs(:,1)), ...
        currPoints(indexPairs(:, 2)), 'Montage');
else
    error('Unable to initialize map.')
end
%% Store Initial Key Frames and Map Points
% Create an empty imageviewset object to store key frames
vSetKeyFrames = imageviewset;
```

```
% Create an empty helperMapPointSet object to store 3D
map points
mapPointSet   = helperMapPointSet;

% Add the first key frame. Place the camera associated
with the first
% key frame at the origin, oriented along the Z-axis
preViewId     = 1;
vSetKeyFrames = addView(vSetKeyFrames, preViewId,
rigid3d, 'Points', prePoints,...
    'Features', preFeatures.Features);

% Add the second key frame
currViewId    = 2;
vSetKeyFrames = addView(vSetKeyFrames, currViewId,
relPose, 'Points', currPoints,...
    'Features', currFeatures.Features);

% Add connection between the first and the second key
frame
vSetKeyFrames = addConnection(vSetKeyFrames, preViewId,
currViewId, relPose, 'Matches', indexPairs);

% Add 3-D map points
[mapPointSet, newPointIdx] = addMapPoint(mapPointSet,
xyzWorldPoints);

% Add observations of the map points
preLocations   = prePoints.Location;
currLocations  = currPoints.Location;
preScales      = prePoints.Scale;
currScales     = currPoints.Scale;

% Add image points corresponding to the map points in the
first key frame
mapPointSet   = addObservation(mapPointSet, newPointIdx,
preViewId, indexPairs(:,1), ....
    preLocations(indexPairs(:,1),:),
preScales(indexPairs(:,1)));

% Add image points corresponding to the map points in the
second key frame
mapPointSet   = addObservation(mapPointSet, newPointIdx,
currViewId, indexPairs(:,2), ...
    currLocations(indexPairs(:,2),:),
currScales(indexPairs(:,2)));
```

```
%% Refine and Visualize the Initial Reconstruction
% Run full bundle adjustment on the first two key frames
tracks       = findTracks(vSetKeyFrames);
cameraPoses  = poses(vSetKeyFrames);

[refinedPoints, refinedAbsPoses] =
bundleAdjustment(xyzWorldPoints, tracks, ...
    cameraPoses, intrinsics, 'FixedViewIDs', 1, ...
    'PointsUndistorted', true, 'AbsoluteTolerance', 1e-
7,...
    'RelativeTolerance', 1e-15, 'MaxIteration', 50);

% Scale the map and the camera pose using the median
depth of map points
medianDepth   = median(vecnorm(refinedPoints.'));
refinedPoints = refinedPoints / medianDepth;

refinedAbsPoses.AbsolutePose(currViewId).Translation =
...
    refinedAbsPoses.AbsolutePose(currViewId).Translation
/ medianDepth;
relPose.Translation = relPose.Translation/medianDepth;

% Update key frames with the refined poses
vSetKeyFrames = updateView(vSetKeyFrames,
refinedAbsPoses);
vSetKeyFrames = updateConnection(vSetKeyFrames,
preViewId, currViewId, relPose);

% Update map points with the refined positions
mapPointSet = updateLocation(mapPointSet, refinedPoints);

% Update view direction and depth
mapPointSet = updateViewAndRange(mapPointSet,
vSetKeyFrames.Views, newPointIdx);

% Visualize matched features in the current frame
close(hfeature.Parent.Parent);
featurePlot = helperVisualizeMatchedFeatures(currI,
currPoints(indexPairs(:,2)));

% Visualize initial map points and camera trajectory %%
Plot map points
```

```
mapPlot     =
helperVisualizeMotionAndStructure(vSetKeyFrames,
mapPointSet);

% Show legend
showLegend(mapPlot);

%%2 ***Tracking***

% ViewId of the current key frame
currKeyFrameId    = currViewId;

% ViewId of the last key frame
lastKeyFrameId    = currViewId;

% ViewId of the reference key frame that has the most co-
visible
% map points with the current key frame
refKeyFrameId     = currViewId;

% Index of the last key frame in the input image sequence
lastKeyFrameIdx   = currFrameIdx - 1;

% Indices of all the key frames in the input image
sequence
addedFramesIdx    = [1; lastKeyFrameIdx];

isLoopClosed      = false;

%  Main loop
while ~isLoopClosed && hasdata(imds)   &&
currFrameIdx<=num_Frames
    currI = readimage(imds, currFrameIdx);
% currFrameIdx
    [currFeatures, currPoints] =
helperDetectAndExtractFeatures(currI,scaleFactor,
numLevels_DetectAndExtract, numPoints);

    % Track the last key frame
    [currPose, mapPointsIdx, featureIdx] =
helperTrackLastKeyFrame(mapPointSet, ...
        vSetKeyFrames.Views, currFeatures, currPoints,
lastKeyFrameId, intrinsics, imageSize,...
        scaleFactor,r_TrackLastKeyFrame,
matchThreshold_MatchFeaturesInRadius,
maxRatio_MatchFeaturesInRadius,...
```

```
        maxRatio_TrackLastKeyFrame_matchFeatures,
matchThreshold_TrackLastKeyFrame_matchFeatures,...

Confidence_TrackLastKeyFrame_estimateWorldCameraPose,
MaxReprojectionError_TrackLastKeyFrame_estimateWorldCamer
aPose,...

MaxIteration_TrackLastKeyFrame_bundleAdjustmentMotion);

    % Track the local map
    [refKeyFrameId, localKeyFrameIds, currPose,
mapPointsIdx, featureIdx] = ...
        helperTrackLocalMap(mapPointSet, vSetKeyFrames,
mapPointsIdx, ...
        featureIdx, currPose, currFeatures, currPoints,
intrinsics,imageSize, scaleFactor,
numLevels_TrackLocalMap,...
        matchThreshold_MatchFeaturesInRadius,
maxRatio_MatchFeaturesInRadius,...

MaxIteration_TrackLocalMap_bundleAdjustmentMotion,
ScaleMinDistance_TrackLocalMap_removeOutlierMapPoints,...

ScaleMaxDistance_TrackLocalMap_removeOutlierMapPoints);

    % Check if the current frame is a key frame.
    isKeyFrame = helperIsKeyFrame(mapPointSet,
refKeyFrameId, lastKeyFrameIdx, ...
        currFrameIdx, mapPointsIdx, MapPoints_IsKeyFrame,
PercentagePointsKeyFrame_IsKeyFrame);

    % Visualize matched features
    updatePlot(featurePlot, currI,
currPoints(featureIdx));

    if ~isKeyFrame
        currFrameIdx = currFrameIdx + 1;
        continue
    end

    % Update current key frame ID
    currKeyFrameId  = currKeyFrameId + 1;

%% 3 ***Local Mapping***

    % Add the new key frame
```

```
    [mapPointSet, vSetKeyFrames] =
helperAddNewKeyFrame(mapPointSet, vSetKeyFrames, ...
        currPose, currFeatures, currPoints, mapPointsIdx,
featureIdx, localKeyFrameIds, coviThrehsold);

    % Update view direction and depth
    mapPointSet = updateViewAndRange(mapPointSet,
vSetKeyFrames.Views, mapPointsIdx);

    % Remove outlier map points that are observed in
fewer than 3 key frames
    mapPointSet = helperCullRecentMapPoints(mapPointSet,
vSetKeyFrames, newPointIdx);

    % Create new map points by triangulation
    [mapPointSet, vSetKeyFrames, newPointIdx] =
helperCreateNewMapPoints(mapPointSet, vSetKeyFrames, ...
        currKeyFrameId, intrinsics, scaleFactor,
minParallax,
MaxRatio_CreateNewMapPoints_matchFeatures,...
        MatchThreshold_CreateNewMapPoints_matchFeatures);

    % Local bundle adjustment
    [mapPointSet, vSetKeyFrames] =
helperLocalBundleAdjustment(mapPointSet, vSetKeyFrames,
...
        currKeyFrameId, intrinsics, maxError,
MaxIteration_LocalBundleAdjustment_bundleAdjustment);

    % Visualize 3D world points and camera trajectory
    updatePlot(mapPlot, vSetKeyFrames, mapPointSet);

%% 4 ***Loop Closure***

    % Initialize the loop closure database
    if currKeyFrameId == 3
        % Load the bag of features data created offline
        bofData         = load('bagOfFeaturesData.mat');
        loopDatabase    =
invertedImageIndex(bofData.bof);
        loopCandidates  = [1; 2];

    % Check loop closure after some key frames have been
created
    elseif currKeyFrameId > 20
```

```
        % Detect possible loop closure key frame
candidates
        [isDetected, validLoopCandidates] =
helperCheckLoopClosure(vSetKeyFrames, currKeyFrameId, ...
            loopDatabase, currI, loopCandidates,
minNumCandidates, maxRange,
NumResults_CheckLoopClosure_evaluateImageRetrieval);

        if isDetected
            % Add loop closure connections
            [isLoopClosed, mapPointSet, vSetKeyFrames] =
helperAddLoopConnections(...
                mapPointSet, vSetKeyFrames,
validLoopCandidates, ...
                currKeyFrameId, currFeatures, currPoints,
intrinsics,imageSize,scaleFactor,
minNumMatches_AddLoop,...
                r_AddLoop,
matchThreshold_MatchFeaturesInRadius,
maxRatio_MatchFeaturesInRadius,
maxRatio_AddLoopConnection_matchFeatures,...

matchThreshold_AddLoopConnection_matchFeatures,
Confidence_AddLoopConnection_estimateWorldCameraPose,...

MaxReprojectionError_AddLoopConnection_estimateWorldCamer
aPose,
MaxIteration_AddLoopConnection_bundleAdjustmentMotion);
        end
    end

    % If no loop closure is detected, add the image into
the database
    if ~isLoopClosed
        fname=[dirname '\' num2str(currFrameIdx) '.png']
        imwrite(currI, fname)

        currds = imageDatastore(fname);
        addImages(loopDatabase, currds, 'Verbose',
false);

        loopCandidates= [loopCandidates; currKeyFrameId];

    end

    % Update IDs and indices
```

```
    lastKeyFrameId  = currKeyFrameId;
    lastKeyFrameIdx = currFrameIdx;
    addedFramesIdx  = [addedFramesIdx; currFrameIdx];
    currFrameIdx  = currFrameIdx + 1;

end
% End of main loop

%% Optimize the poses

minNumMatches      = 40;
vSetKeyFramesOptim = optimizePoses(vSetKeyFrames,
minNumMatches, 'Tolerance', 1e-16, 'Verbose', true);

% Plot the optimized camera trajectory
optimizedPoses  = poses(vSetKeyFramesOptim);
plotOptimizedTrajectory(mapPlot, optimizedPoses)

% Update legend
showLegend(mapPlot);

%% Compare with the Ground Truth

% Load ground truth
gTruthData = load('orbslamGroundTruth.mat');
gTruth     = gTruthData.gTruth;

% Plot the actual camera trajectory
plotActualTrajectory(mapPlot, gTruth(addedFramesIdx),
optimizedPoses);

% Show legend
showLegend(mapPlot);

% Evaluate tracking accuracy
helperEstimateTrajectoryError(gTruth(addedFramesIdx),
optimizedPoses);

%% plot point cloud
figure,
scatter3(mapPointSet.Locations(:,1),mapPointSet.Locations
(:,2),mapPointSet.Locations(:,3),'+');
% XLim = [-1.5 1.5];
% YLim = [-1 0.5];
% ZLim = [-0.5 2];
```

```
XLim = [min(mapPointSet.Locations(:,1))
max(mapPointSet.Locations(:,1))];
YLim = [min(mapPointSet.Locations(:,2))
max(mapPointSet.Locations(:,2))];
ZLim = [min(mapPointSet.Locations(:,3))
max(mapPointSet.Locations(:,3))];
player1 = pcplayer(XLim, YLim, ZLim, ...
                'VerticalAxis', 'y', 'VerticalAxisDir',
'down', 'MarkerSize', 30);
xlabel('X'),ylabel('Y'),zlabel('Z')
ptCloud = pointCloud(mapPointSet.Locations);
view(player1,ptCloud);
```