

Test e implementazione di Orbslam3 su ROS (1)

Alex Ardelean
Giuseppe Gallo

Abstract—Il VSLAM ha suscitato notevole interesse negli ultimi anni, soprattutto per le sue applicazioni real time. A seguito dell'algoritmo PTAM, che implementa in parallelo mapping e tracking, ORBSLAM ne ha aumentato notevolmente le performance implementando loop closing, relocalization e inizializzazione automatica. I vantaggi nell'utilizzo di ORBSLAM sono dati principalmente dalle feature ORB, estratte velocemente ed utilizzate per tutti i task. In quest'ottica, l'algoritmo è stato testato su varie rosbag, con differenti condizioni di luminosità e su due mappe diverse. Al fine di valutarne le performance, sono stati poi variati gli iperparametri di ORB con differenti combinazioni, confrontando i risultati con metodi qualitativi e quantitativi. L'invarianza delle performance all'aumento del numero di feature ORB, così come l'estrema varianza dei risultati sotto le stesse condizioni e parametri, hanno portato alla luce la necessità di utilizzare di un approccio statistico, per tenere conto dell'aleatorietà intrinseca dell'algoritmo. Si è tuttavia confermato in generale il degradamento delle performance al diminuire delle condizioni di illuminazione ambientale.

I. INTRODUCTION

Il processo di Visual SLAM coinvolge l'uso di sensori visivi, come le telecamere, per consentire a un dispositivo mobile, come un robot o un veicolo autonomo, di navigare in un ambiente sconosciuto mentre simultaneamente costruisce una mappa di tale ambiente e determina la sua posizione all'interno di essa. L'interesse per la tecnologia Visual SLAM è cresciuto notevolmente di recente, soprattutto a causa delle sue capacità di essere eseguita in tempo reale. L'utilizzo di tecniche di ottimizzazione come il Bundle Adjustment sono cruciali per ottenere risultati accurati e robusti nella localizzazione e mappatura simultanea visuale in tempo reale.

Una delle prime applicazioni real-time del BA è stata effettuata in [1] per risolvere un problema di visual odometry. L'algoritmo PTAM [2] è stato uno dei primi sistemi di VSLAM in tempo reale. La caratteristica distintiva di PTAM è stata la sua capacità di eseguire il tracciamento e la mappatura in parallelo. In altre parole, mentre il sistema traccia la posizione della telecamera nell'ambiente, simultaneamente costruisce una mappa dell'ambiente circostante.

L'idea di base di ORB-SLAM [3] condivide alcuni principi chiave con PTAM, esso infatti è un sistema SLAM monoculare feature-based che opera real-time sia in ambienti indoor che outdoor. Il sistema è robusto al motion clutter, e a differenza di PTAM implementa il loop closing, la relocalization e l'inizializzazione automatica. Il sistema utilizza le stesse features per tutti i task che svolge. Le features e i keyframe sono selezionati con la strategia survival of the fittest al fine di garantire robustezza e buone prestazioni in ambienti dinamici per lunghi periodi. Le tecniche basate su keyframe sfruttano un sottoinsieme di immagini per effettuare Bundle Adjustment

e risultano essere più efficienti computazionalmente e più accurate come dimostrato in [1] e [2].

A. ORB

Essendo ORB-SLAM un metodo VSLAM indiretto rileva e traccia punti di interesse nell'immagine (features) per calcolare la posizione della camera. Al fine di poter essere eseguito in real-time ORB-SLAM utilizza un feature extractor molto veloce chiamato ORB [4]. ORB è basato su:

- FAST [5] per determinare i keypoint.
- BRIEF [6] come descrittore per le features.

Per ottenere l'invarianza di scala ORB genera più versioni dell'immagine facendo downsampling e eseguendo FAST su tutte le versioni, ottenendo così features su più scale. Per ottenere l'invarianza rispetto a rotazioni si calcola la direzione della feature e la si ruota ad una direzione normale.

B. Inizializzazione automatica della mappa

L'inizializzazione automatica della mappa ha come obiettivo quello di calcolare la posa relativa tra due frame c e r per triangolare un insieme iniziale di punti della mappa. Per fare ciò ORB-SLAM prevede una prima fase di model selection in cui, in base a due frame di riferimento, viene scelto il modello da utilizzare per l'inizializzazione. Nel caso in cui la scena presenta una vista planare si utilizza il modello Homography che grazie alla Homography Matrix H_{cr} permette di esprimere la relazione tra le due immagini ($x_c = H_{cr}x_r$). Nel caso in cui la scena sia non planare invece, si utilizza la Fundamental Matrix F_{cr} per esprimere la relazione tra i punti nelle immagini ($x_c^T F_{cr} x_r = 0$). In base alle corrispondenze tra features trovate nelle due immagini si calcolano F_{cr} e H_{cr} e un loro score associato S_F e S_H che esprime la qualità con cui i due modelli rappresentano la scena. Nel caso in cui la quantità $R_H = \frac{S_H}{S_H + S_F} > 0.45$ viene scelto il modello Homography, altrimenti viene scelta la Fundamental Matrix. Una volta selezionato il modello è possibile sfruttarlo per determinare il movimento della camera tra i due frame e usare il full BA per raffinare la ricostruzione iniziale.

C. Tracking

Il tracking della posa della camera viene effettuato frame by frame con il motion-only BA. Si usa un modello a velocità costante per predire una posa iniziale della camera. Sfruttando tale stima iniziale è possibile proiettare i punti della mappa nel frame corrente per effettuare BA. Per limitare la complessità si proietta solo una mappa locale individuata grazie al Covisibility Graph, ovvero un grafo non orientato che descrive la similitudine tra i keyframe. Ogni nodo è un keyframe e gli

archi tra i nodi hanno un peso θ pari al numero di punti in comune (almeno 15). Il modulo di tracking si occupa anche della selezione dei nuovi keyframe, che vengono scelti in base ad un criterio temporale e di features individuate.

D. Local Mapping

Il modulo di Local Mapping processa i nuovi keyframe ed effettua il local BA per una ricostruzione ottima della mappa nel vicinato della posa della camera. Il local BA ottimizza il keyframe corrente K_i , tutti i keyframe connessi ad esso nel covisibility graph \mathcal{K}_c e tutti i punti visti da questi keyframe. I keyframe che vedono questi punti (ma non connessi) vengono coinvolti ma non ottimizzati. Il modulo di Local Mapping si occupa anche della rimozione di keyframe e keypoint ridondanti.

E. Loop Closing

Il modulo di Loop Closing cerca un loop in ogni nuovo keyframe. Se viene rilevato un loop si calcola una trasformazione di similarità per valutare il drift accumulato. I due lati del loop vengono allineati e i punti doppi vengono fusi. Si effettua inoltre un pose graph optimization sull'essential graph (sottografo sparso del covisibility graph) al fine di ottenere una consistenza globale. La struttura del grafo è quella del covisibility graph, tuttavia il numero di archi è minore. Gli archi contenuti sono quelli dello spanning tree (calcolato a partire dalla posa iniziale) insieme agli archi con covisibilità elevata $\theta \geq 100$ e gli archi di loop closing.

I moduli di Tracking, Local Mapping e Loop Closing possono essere eseguiti in parallelo in 3 thread diversi, sfuttando così l'idea base di PTAM che ha reso possibile la sua esecuzione real-time.

II. PROJECT OBJECTIVE

L'obiettivo primario di questo progetto è eseguire un'analisi critica del funzionamento di ORB-SLAM3 (versione ROS) su specifici dataset, con lo scopo di valutarne l'efficacia in differenti scenari. In particolare, l'attenzione sarà rivolta alla stima della posa e alle variabili che possono influenzarla.

I dataset utilizzati consistono nello Scenario A:

- *Corridor_A_D_127_T1*

e nello Scenario B:

- *Corridor_B_L*
- *Corridor_B_D_40_T2*
- *Corridor_B_D_85_T3*

In cui le lettere *A* e *B* indicano gli scenari in cui si trova il robot, *L* indica che il robot si trova in condizioni di luminosità ambientale buona, *D_n* indica che il robot si trova in condizioni di luminosità ambientale scarsa e *n* è il valore di intensità luminosa delle luci artificiali utilizzate che può variare tra 0 e 255.

Si esamineranno i risultati ottenuti dall'esecuzione di ORB-SLAM3 su ciascun dataset, con una valutazione critica della precisione nella stima della posizione e dell'orientamento.

Si analizzerà come i parametri di hypertuning influenzano le prestazioni di ORB-SLAM3, con una valutazione attenta dei cambiamenti nei risultati al variare di tali parametri.



(a) *Corridor_B_L*



(b) *Corridor_B_D_40_T2*



(c) *Corridor_B_D_85_T3*



(d) *Corridor_A_D_127_T1*

Fig. 1: Campioni delle rosbag utilizzate

III. CHOSEN SOLUTIONS

La soluzione adottata consiste nell'utilizzo di ORB-SLAM3 nell'ambiente ROS per valutare le sue prestazioni in condizioni di luce variabile.

Per replicare scenari realistici e garantire la flessibilità nell'esecuzione di ORB-SLAM3, si è deciso di utilizzare rosbag, uno strumento di registrazione e riproduzione dei dati ROS. Questo approccio consente di simulare dati in tempo reale, facilitando la riproduzione di diverse condizioni ambientali e di movimento per valutare le prestazioni di ORB-SLAM3 in scenari variabili.

La versione mono di ORBSLAM è iscritta al topic `/camera/image_raw` su cui si aspetta di ricevere i dati della camera in tempo reale. Le rosbag utilizzate invece pubblicano i dati della camera sul topic `/bluefox_camera/image_raw`. Si è quindi effettuato un rimappaggio di tali topic nel launchfile, al fine di garantire il corretto funzionamento:

```
<remap from="/camera/image_raw"
      to="/bluefox_camera/image_raw"/>
```

I topic su cui pubblica ORBSLAM sono:

- `/orb_slam3/camera_pose`: posa stimata per la camera;
- `/orb_slam3/tracking_image`: immagine contenente anche i punti chiave estratti con ORB;
- `/orb_slam3/tracked_points`: punti chiave nella sliding window;
- `/orb_slam3/all_points`: punti chiave della mappa;
- `/orb_slam3/kf_markers`: posa dei keyframe;

La versione ORBSLAM di ROS presenta i seguenti parametri da configurare:

- `voc_file`: percorso del file di vocabolario richiesto da ORB-SLAM3. Il vocabolario bag-of-words viene creato durante la fase di addestramento del sistema, dove vengono estratte e catalogate le feature da un insieme di immagini di riferimento. Durante l'esecuzione dell'algoritmo, le nuove immagini vengono confrontate con il vocabolario per identificare le corrispondenze tra le feature presenti nell'immagine corrente e quelle presenti nelle immagini precedenti.
- `settings_file`: file contenente i parametri intrinseci della camera, i parametri di hypertuning di ORB e alcuni parametri di visualizzazione.
- `enable_pangolin`: abilita/disabilita l'interfaccia grafica Pangolin di ORB-SLAM3

Il modulo ROS di ORBSLAM mette inoltre a disposizione i seguenti servizi:

- `/orb_slam3/save_map`: permette di salvare la mappa;
- `/orb_slam3/save_traj`: permette di salvare la traiettoria stimata da ORBSLAM nel formato TUM;

Per l'analisi dei dati prodotti da ORB-SLAM3, si è scelto di utilizzare lo strumento `evo_traj`. Questo tool offre un'interfaccia a riga di comando e fornisce metriche di valutazione della traiettoria stimata. Per migliorare l'accuratezza dell'analisi, `evo_traj` fa uso del metodo Umeyama per stimare i parametri estrinseci delle traiettorie prodotte da ORB-SLAM3, consentendo così di allineare la traiettoria Ground Truth a quella stimata da ORBSLAM individuando anche la scala

corretta. Dato che i timestamp delle traiettorie ground truth e quelle prodotto da ORB-SLAM sono espressi su scale diverse, per poter essere processate da `evo_traj` devono essere corretti. Per fare ciò si è scritto lo script python `timestamp_correction.py`. Per eseguire lo script python è sufficiente digitare da riga di comando

```
python timestamp_correction.py
      traj_file_ref
      traj_file_to_correct
```

- `traj_file_ref` è il file TUM della traiettoria ground truth;
- `traj_file_to_correct` è il file TUM della traiettoria prodotta da ORB-SLAM;

Dato che `evo_traj`, a seguito della correzione con Umeyama, non allinea le origini delle traiettorie in modo automatico, si è scritto lo script python `align_origin.py` che le allinea grazie ad una rototraslazione individuata a partire dalla prima posa nelle due traiettorie. Questo procedimento ha permesso di effettuare un confronto anche sul drift che si ottiene tra la traiettoria ground truth e quella stimata da ORB-SLAM. Per eseguire lo script python è sufficiente digitare da riga di comando

```
python align_origin.py ref_traj
      traj_to_align
```

in cui:

- `ref_traj` è il file TUM della traiettoria ground truth;
- `traj_to_align` è il file TUM della traiettoria corretta tramite il tool `evo_traj`;

Un aspetto cruciale della metodologia adottata è la variazione dei parametri di hypertuning di ORB-SLAM3. Questa scelta consente di esplorare il comportamento del sistema in risposta a diverse configurazioni, valutando l'effetto di tali parametri sulle prestazioni dell'algoritmo di SLAM. Durante l'implementazione delle soluzioni proposte, sono emerse alcune problematiche. Ad esempio, la configurazione ottimale dei parametri di hypertuning è risultata non banale, richiedendo un processo di sperimentazione iterativo. Si è infine arrivati alla conclusione che essendo ORBSLAM non completamente deterministico, la determinazione dei parametri di hypertuning avrebbe richiesto l'utilizzo di una metodologia Monte Carlo. Inoltre la determinazione di tali parametri sarebbe stata fine a se stessa dato che probabilmente per ogni scenario i parametri di hypertuning ottimi potrebbero essere diversi.

Dato che la procedura di analisi dei dati prodotti da ORB-SLAM doveva essere ripetuta per un numero elevato di esperimenti si è scelto di automatizzarla tramite lo script `run.bat`. Tale script assume di avere nel suo stesso path:

- Una cartella nominata `1. origin data` contenente i file delle traiettorie ground truth e quelle generate da ORB-SLAM.
- Una cartella nominata `map_gt` contenente i file relativi alla mappa in formato ROS.
- Lo script `align_origin.py`.
- Lo script `timestamp_correction.py`

Per ogni traiettoria esegue quindi i seguenti step:

- Crea la cartella 2. *correct_timestamp_data* contenente le traiettorie con i timestamp corretti, in modo da poter essere elaborati da *evo_traj*.
- Crea la cartella 3. *scaled_rotated_and_translated_data* contenente le traiettorie corrette tramite il tool *evo_traj* con il metodo Umeyama.
- Crea la cartella 4. *aligned_origin_data* contenente le traiettorie allineate tramite lo script *align_origin.py*.
- Crea la cartella 5. *results* contenente i risultati ottenuti dal confronto delle traiettorie ground truth con quelle generate da ORB-SLAM.

IV. EXPERIMENTS

A. Esperimento 1

Come primo esperimento si sono utilizzate le rosbag *Corridor_B_L.bag* e *Corridor_B_D_40_T2.bag*. Il formato rosbag consente agli sviluppatori di registrare i dati generati da un sistema ROS durante un'esecuzione e di riprodurli in un secondo momento. Questa funzionalità è utile per il debugging, il testing e lo sviluppo di algoritmi senza la necessità di avere accesso al sistema robotico fisico in ogni istante. Le due rosbag in questione presentano i dati video relativi al percorso di un robot all'interno di un corridoio. In particolare la prima presenta delle buone condizioni di illuminazione, mentre la seconda è stata catturata in condizioni di illuminazione molto scarse e grazie all'ausilio di una fonte di illuminazione artificiale posta sul robot.

Si è quindi voluto confrontare l'errore compiuto da ORB-SLAM rispetto al Ground Truth nei due dataset al variare dei parametri di hypertuning:

- *ORBextractor.nFeatures*;
- *ORBextractor.scaleFactor*;
- *ORBextractor.nLevels*;
- *ORBextractor.iniThFAST* e *ORBextractor.minThFAST*;

In particolare si sono fatti variare i parametri come mostrato nella tabella I al fine di generare 7 scenari.

Case	nFeatures	scaleFactor	nLevels	iniThFAST	minThFAST
1	1250	1.2	8	20	5
2	750	1.2	8	20	5
3	1750	1.2	8	20	5
4	1250	1.53	4	20	5
5	1250	1.123	12	20	5
6	1250	1.2	8	10	2
7	1250	1.2	8	30	10

TABLE I: Tabella parametri esperimento 1

Case	<i>Corridor_B_L.bag</i>		<i>Corridor_B_D_40_T2.bag</i>	
	APE	RPE	APE	RPE
1	0.170	0.117	1.549	0.049
2	0.162	0.093	3.121	0.113
3	0.196	0.166	0.849	0.089
4	0.130	0.120	3.166	0.309
5	0.145	0.128	8.976	0.297
6	0.233	0.108	2.621	0.292
7	0.157	0.119	4.804	0.201

TABLE II: RMSE dell'APE e dell'RPE nei dataset *Corridor_B_L.bag* e *Corridor_B_D_40_T2.bag* eseguiti con gli iperparametri in tabella I

Dalle figure 2 e 3 è possibile notare che per quanto riguarda il caso di condizioni di illuminazione buone i parametri di hypertuning non hanno influenzato notevolmente le performance di ORB-SLAM. Nel caso di condizioni di illuminazione scarsa invece è possibile notare che:

- la variazione di *ORBextractor.iniThFAST* e *ORBextractor.minThFAST* nei casi 6 e 7 ha portato ad una traiettoria completamente errata, per cui probabilmente i valori nei casi 1-5 sono i più adatti, utilizzare una soglia troppo alta o troppo bassa infatti potrebbe portare a keypoint non significativi;
- la variazione di *ORBextractor.scaleFactor*; e *ORBextractor.nLevels* ha portato ad una traiettoria peggiore rispetto ai casi 1-3. In particolare l'aumento del parametro *ORBextractor.nLevels* ha portato i risultati peggiori fra tutti gli esperimenti;
- nei casi 1-3 è possibile notare che all'aumentare di *ORBextractor.nFeatures* aumenta la precisione con cui ORB-SLAM stima la traiettoria;

B. Esperimento 2

Dato che il precedente esperimento ha portato alla conclusione che la stima della traiettoria prodotta da ORB-SLAM migliori all'aumentare del parametro *ORBextractor.nFeatures*, si è deciso di effettuare un ulteriore esperimento in cui si valutano le performance di ORB-SLAM al variare di tale parametro. Nella tabella III sono riassunti i parametri utilizzati.

Case	nFeatures	scaleFactor	nLevels	iniThFAST	minThFAST
1	1000	1.2	8	20	5
2	2000	1.2	8	20	5
3	3000	1.2	8	20	5
4	4000	1.2	8	20	5
5	5000	1.2	8	20	5

TABLE III: Tabella parametri esperimento 2

Case	<i>Corridor_B_D_40_T2.bag</i>	
	APE	RPE
1	0.648	0.084
2	3.511	0.146
3	2.113	0.135
4	2.072	0.131
5	2.612	0.126

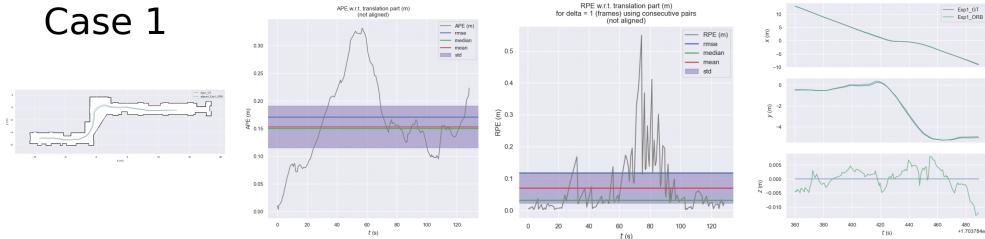
TABLE IV: RMSE dell'APE e dell'RPE nel dataset *Corridor_B_D_40_T2.bag* eseguito con gli iperparametri in tabella III

C. Esperimento 3

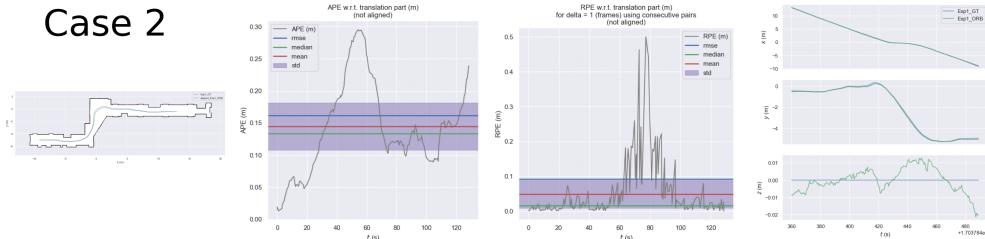
Dato che l'esperimento 2 non ha portato alcuna evidenza sul fatto che la stima di ORB-SLAM migliori all'aumentare di *ORBextractor.nFeatures* si è deciso di effettuare un ulteriore esperimento facendo variare *ORBextractor.nFeatures* nell'intervallo 750-2000. Nella tabella V sono riassunti i parametri utilizzati.

Nemmeno l'esperimento 3 ha portato ad una evidenza sul fatto che la stima di ORB-SLAM migliori all'aumentare di *ORBextractor.nFeatures*. Si è però notato che le performance

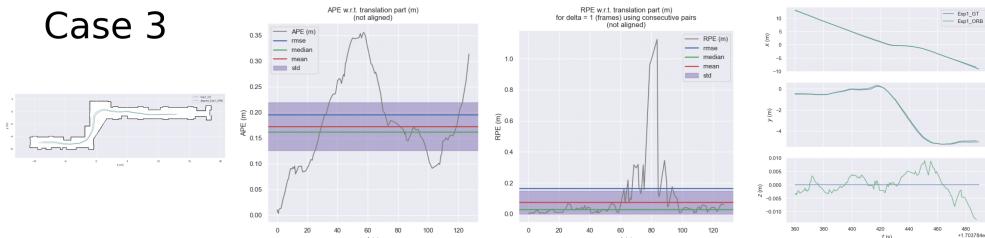
Case 1



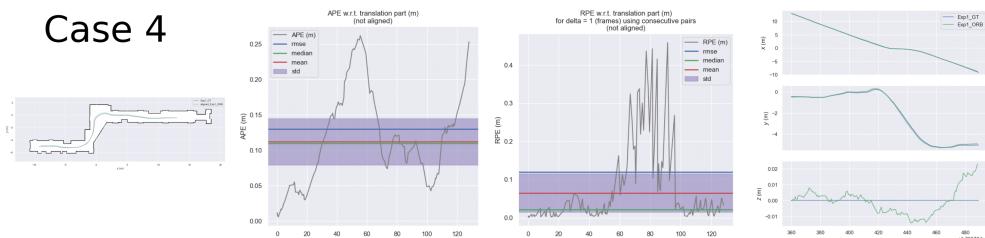
Case 2



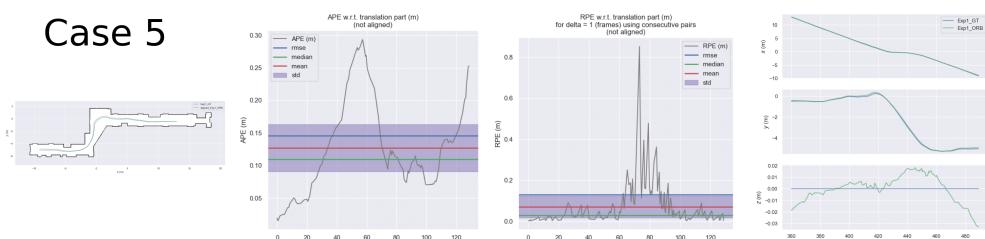
Case 3



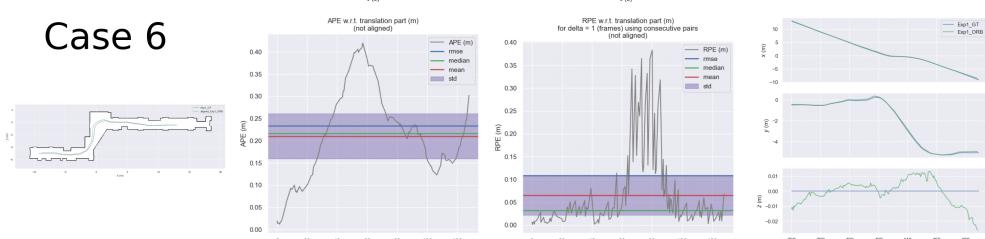
Case 4



Case 5



Case 6



Case 7

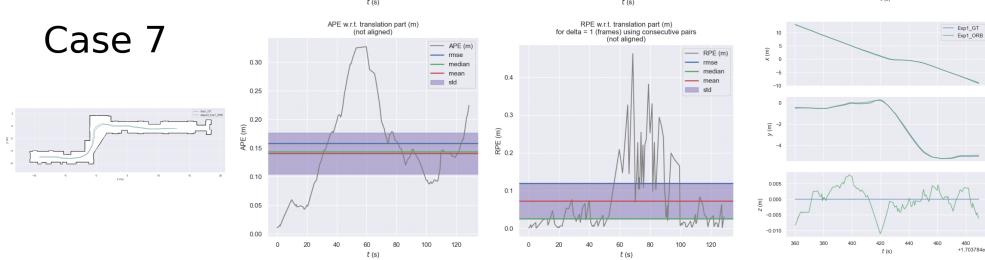
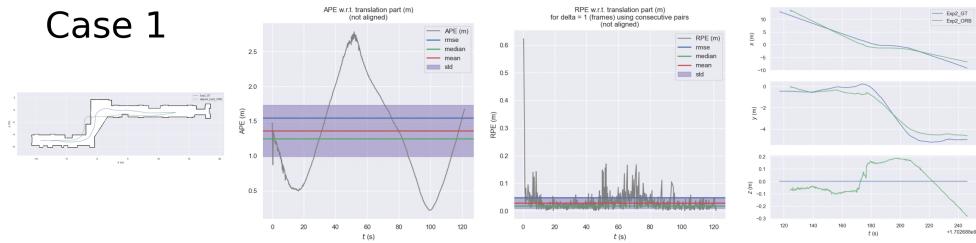
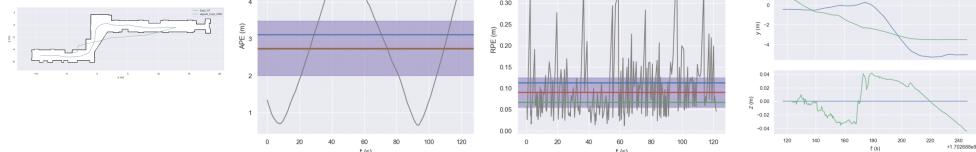


Fig. 2: Experimento 1 (dataset *Corridor_B_L.bag*)

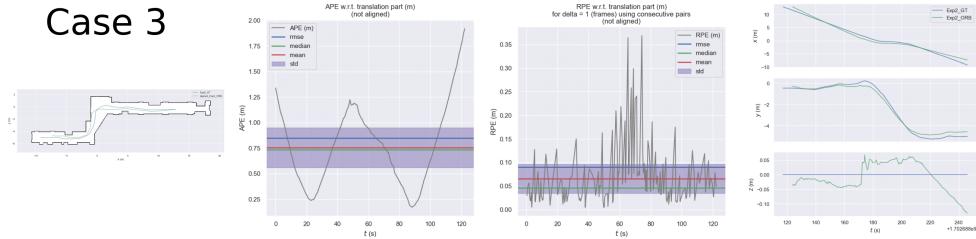
Case 1



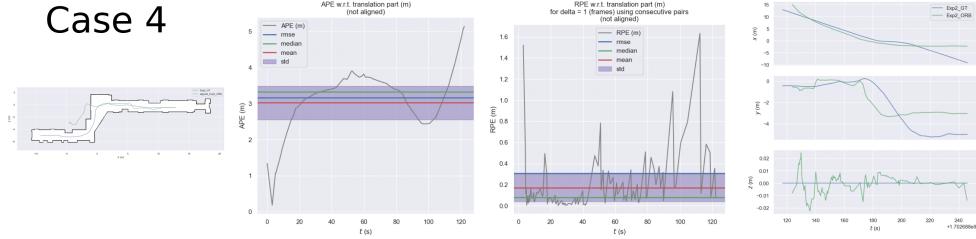
Case 2



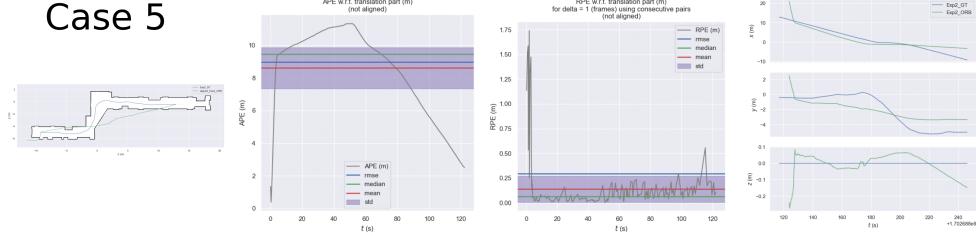
Case 3



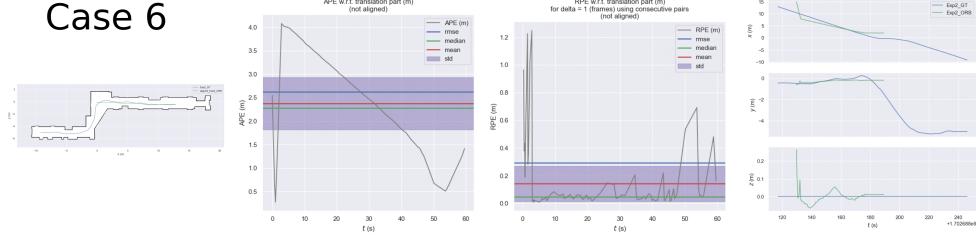
Case 4



Case 5



Case 6



Case 7

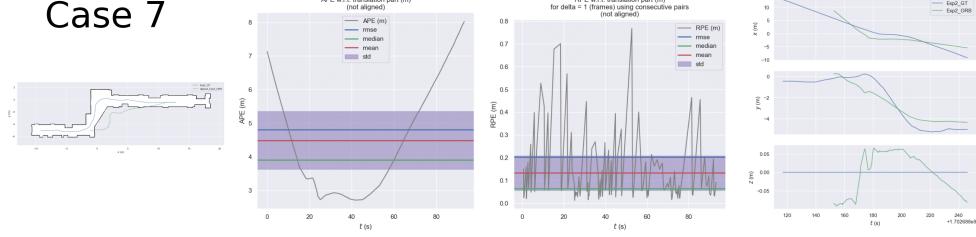


Fig. 3: Experimento 1 (dataset *Corridor_B_D_40_T2.bag*)

dipendono molto dalla specifica esecuzione. In particolare, nell'esperimento 1 si è visto che per $ORBextractor.nFeatures=1750$ si sono ottenute prestazioni molto migliori che nel caso dell'esperimento 3 con lo stesso parametro. In tutti gli esperimenti però si è notato che si ottengono performance buone con il parametro $ORBextractor.nFeatures$ nel range 1000-1250.

Case	nFeatures	scaleFactor	nLevels	iniThFAST	minThFAST
1	750	1.2	8	20	5
2	1000	1.2	8	20	5
3	1250	1.2	8	20	5
4	1500	1.2	8	20	5
5	1750	1.2	8	20	5
6	2000	1.2	8	20	5

TABLE V: Tabella parametri esperimento 3

<i>Corridor_B_D_40_T2.bag</i>		
Case	APE	RPE
1	2.217	0.111
2	0.862	0.086
3	2.596	0.131
4	3.518	0.155
5	2.215	0.115
6	3.155	0.132

TABLE VI: RMSE dell'APE e dell'RPE nel dataset *Corridor_B_D_40_T2.bag* eseguito con gli iperparametri in tabella V

D. Esperimento 4

L'esperimento 4 consiste nell'eseguire ORB-SLAM sul dataset *Corridor_B_D_85_T3*.

<i>Corridor_B_D_85_T3.bag</i>		
Case	APE	RPE
1	1.944	0.127

TABLE VII: RMSE dell'APE e dell'RPE nel dataset *Corridor_B_D_85_T3*

E. Esperimento 5

L'esperimento 5 consiste nell'eseguire ORB-SLAM sul dataset *Corridor_A_D_127_T1*. Rispetto agli esperimenti precedenti, il corridoio non presenta curve. Ciò ha compensato la scarsa luminosità (7) dando buoni risultati, come è possibile vedere in tabella VIII.

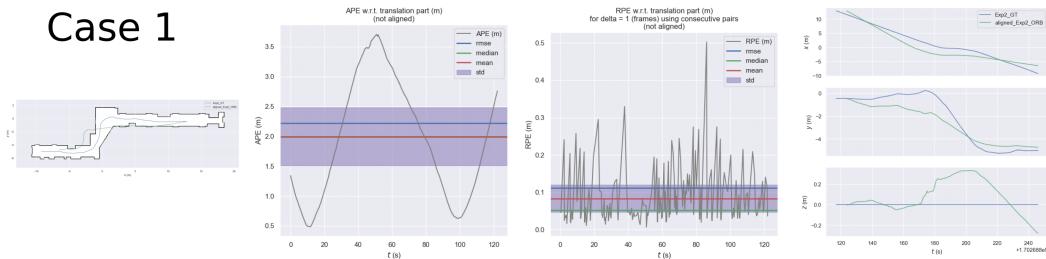
<i>Corridor_A_D_127_T1.bag</i>		
Case	APE	RPE
1	0.253	0.061

TABLE VIII: RMSE dell'APE e dell'RPE nel dataset *Corridor_A_D_127_T1*

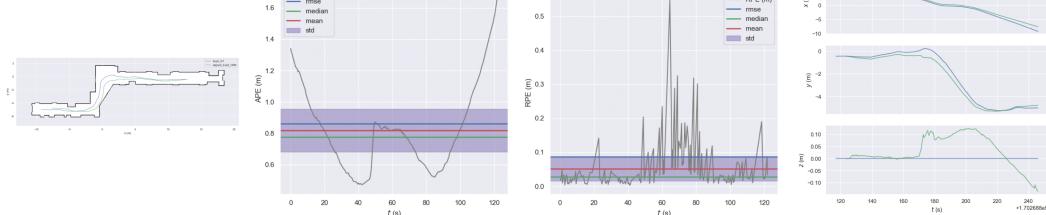
REFERENCES

- [1] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd, "Real time localization and 3d reconstruction," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 1, 2006, pp. 363–370.
- [2] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, 2007, pp. 225–234.
- [3] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.
- [4] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571.
- [5] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7–13, 2006. Proceedings, Part I* 9, Springer, 2006, pp. 430–443.
- [6] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5–11, 2010, Proceedings, Part IV* 11, Springer, 2010, pp. 778–792.

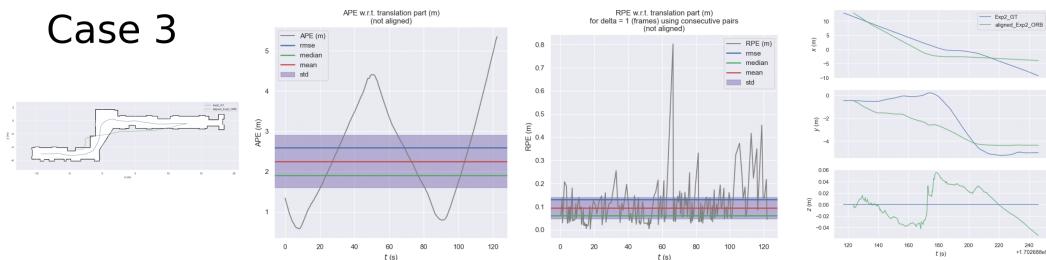
Case 1



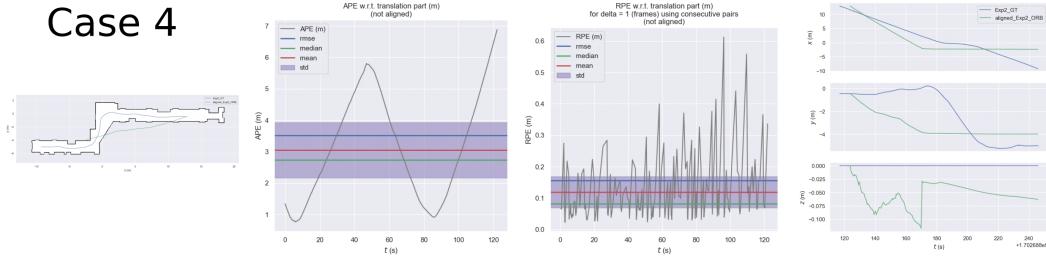
Case 2



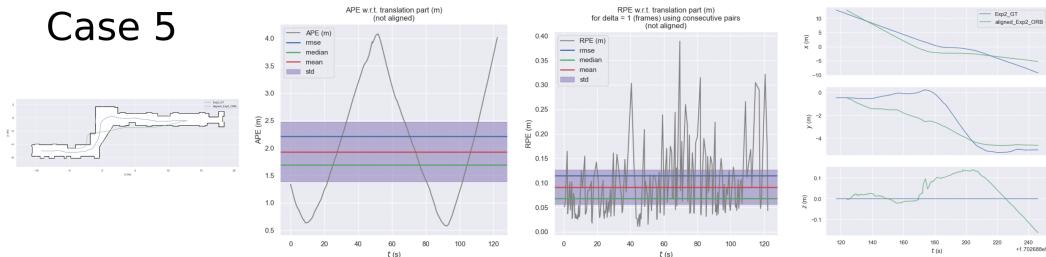
Case 3



Case 4



Case 5



Case 6

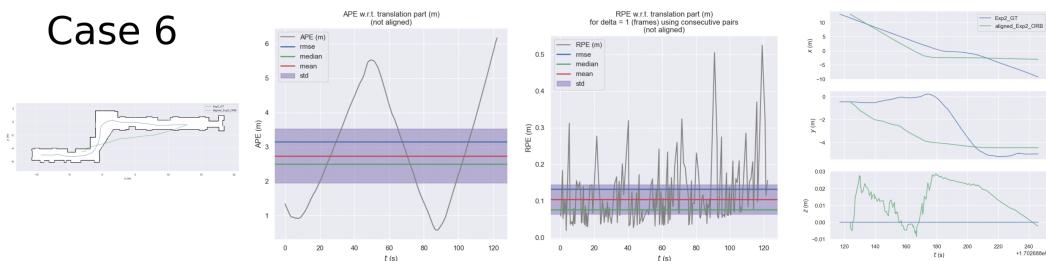


Fig. 5: Experimento 3 (dataset *Corridor_B_D_40_T2.bag*)

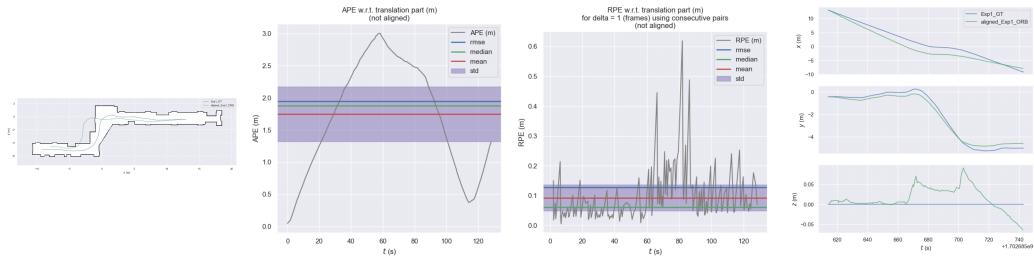


Fig. 6: Experimento 4 (dataset *Corridor_B_D_85_T3*)

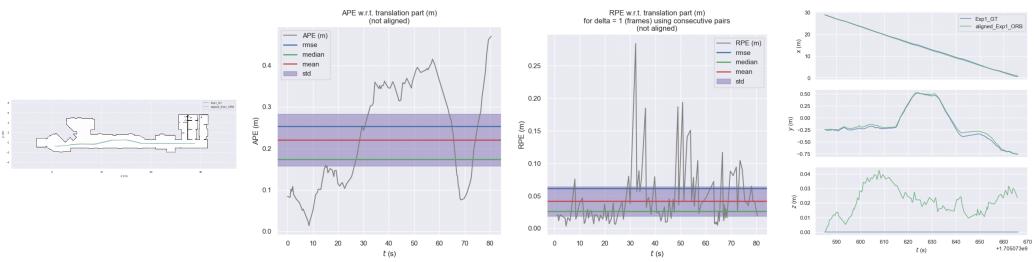


Fig. 7: Experimento 5 (dataset *Corridor_A_D_127_T1*)