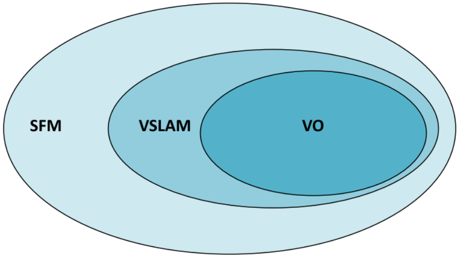
Self-Localization using SLAM

Saqib Azim, **Nito Takumi**

Tuesday 20th December, 2022

Visual Odometry (VO) vs Visual SLAM (vSLAM)

- Camera tracking (aka VO) focuses on estimating 3D camera motion sequentially (as new frame arrives) in real time
- VO only aims for local consistency of the camera trajectory
- SLAM aims for global consistency of the camera trajectory as well as the map



Applications of SLAM

- Robotics - Self-Driving Cars, Unmanned Aerial Vehicles, etc.
- Augmented and Virtual Reality applications

Robust SLAM systems becoming consumer products

- Google Tango (visual-inertial odometry, uses RGB-D sensor)
- Google ARCore
- Dyson 360 Eye
- Microsoft Hololens (uses RGB-D sensor)
- Oculus

vSLAM Standard Benchmark Datasets I

KITTI SLAM Dataset [8]

- 22 stereo image sequence (11 with gt trajectories + 11 without gt trajectories)
- Driving scenario in urban and highway environments
- Ground truth camera poses obtained using RTK-GPS

TUM-RGBD Dataset [6]

- 39 RGB-D image sequences (with 6-DoF camera poses) with accurate ground-truth
- Recorded using Microsoft kinect in different indoor scenes

TUM monoVO Dataset [17]

- 50 photometrically calibrated sequences, recorded in different (indoor+outdoor) environments, starting and ending at the same position
- Allows to evaluate tracking accuracy via accumulated drift from start to end, without requiring ground-truth for full sequence (only provides loop-closure-ground-truth)

EuRoC MAV Dataset [15]

- Visual-Inertial dataset, collected on-board a Micro Aerial Vehicle (MAV) in 3 different indoor environments

vSLAM Standard Benchmark Datasets II

- Contains 11 stereo-image sequences, synchronized IMU measurements, accurate motion and structure ground-truth

New College Vision and Laser Dataset [3]

- Contains 2.2-km stereo-image sequences from a robot traversing outdoor environments
- Contains several loops and fast rotations

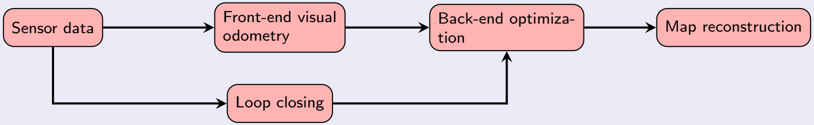
ICL-NUIM Dataset [11]

- Contains 8 ray-traced RGB-D sequences ($\approx$ 4.5 mins video) from 2 indoor environments

General vSLAM Framework

SLAM Algorithm - Main Components

1. System Initialization
2. Camera Motion/Pose Estimation
3. 3D Structure Estimation
4. Global Optimization
5. Loop Closure
6. Relocalization



Indirect (Feature-based) Methods

• MonoSLAM: Real-Time Single Camera SLAM [1]	[2007]
• Parallel Tracking and Mapping in small AR workspaces (PTAM) [2]	[2007]
• Scale Drift-aware large scale monocular SLAM [4]	[2010]
• Fast relocalisation and loop closing in keyframe-based SLAM [12]	[2014]
• **(SoTA)** ORB-SLAM: a versatile and accurate monocular SLAM system [14]	[2015]
• DynaSLAM: Tracking, Mapping and Inpainting in dynamic scenes [19]	[2018]
• OpenVSLAM: A versatile visual SLAM framework [20]	[2019]

Indirect (Feature-based) Methods

Indirect Methods

- Proceed in 2 steps -
  1. Raw sensor measurements $\xrightarrow{\text{preprocessed}}$ generate an intermediate repr. (extracting KPs and matching feature descriptors)
  2. computed intermediate values $\xrightarrow[\text{as}]{\text{interpreted}}$ noisy measurements in a probabilistic model to estimate geometry and camera motion
- Optimizes geometric error (since intermediate values are geometric quantities)
- Sparse 3D map reconstruction

Direct (Intensity-based) Methods

1. DTAM: Dense Tracking and Mapping in Real-Time [5] [2011]
2. Semi-Dense Visual Odometry for a Monocular Camera [7] [2013]
3. Semi-Dense Visual Odometry for AR on a Smartphone [13] [2014]
4. SVO: Fast Semi-Direct Monocular Visual Odometry [10] [2014]
5. Large-Scale Direct Monocular SLAM (LSD-SLAM) [9] [2014]
6. **(SoTA)** Direct Sparse Odometry (DSO) [16] [2016]

Direct (Intensity-Based) Methods

Direct Methods (categorization based on map density)

- Directly use actual sensor values (light received from a certain direction over a certain time-period) as measurements in a probabilistic model $\implies$ optimizes photometric error
- Directly exploits brightness information of all pixels in images $\implies$ Can be operated in more texture-less environments
- Lower performance than indirect methods with rolling shutter cameras (image sensors in smartphones and consumer cameras)
- completely dense or semi-dense map reconstruction
- Semi-dense: refrain from reconstructing complete surface, aim at using and reconstructing a subset (largely connected and well-constrained)
- More potential applications for robotics, than the sparse map generated by feature-based SLAM

Sparse vs Semi-Dense vs Dense Methods

Sparse Methods

- Use and Reconstruct only a selected set of independent points (traditionally corners)
- No notion of neighborhood, and geometry parameters (KP positions) are conditionally independent given the camera poses & intrinsics

Dense Methods

- Attempt to use and reconstruct all pixels in the 2D image domain
- Dense (or semi-dense) approaches exploit the connectedness of image region to formulate a **geometric prior** (typically favouring smoothness)

Semi-Dense Methods

- Refrain from reconstructing complete surface $\rightarrow$ Attempt to use and reconstruct a (largely connected + well-constrained) subset

Sparse/Dense + Direct/Indirect

Sparse+Indirect (most widely-used):

- Estimate 3D geometry from KP-matches $\rightarrow$ using geometric error without geometric prior. Examples - MonoSLAM [1], PTAM [2], ORB-SLAM [14]

Dense+Indirect:

- Estimate 3D geometry from dense optical flow $\rightarrow$ using geometric error (deviation from flow field) with geometric prior (smoothness of flow field). Examples - [18]

Dense+Direct:

- Employs photometric error + geometric prior to estimate dense or semi-dense geometry. Examples - DTAM [5], LSD-SLAM [9]

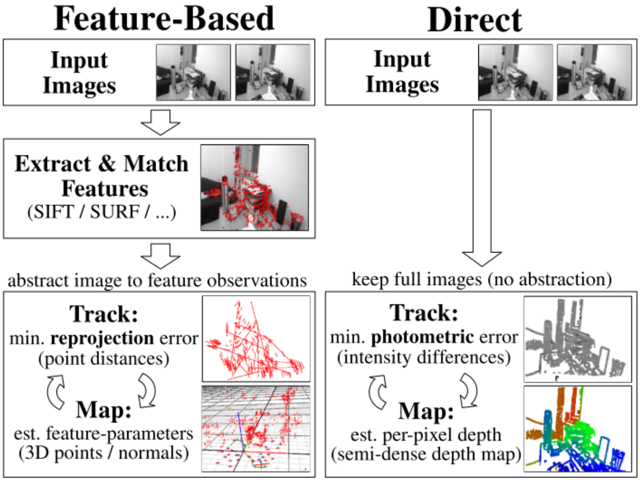
Sparse+Direct:

- Optimizes photometric error without geometric prior. Examples - DSO [16]

Hybrid:

- Direct formulation for initial alignment and to obtain corr. $\xrightarrow[\text{to}]{\text{switch}}$ an indirect formulation for joint model optimization. Examples - SVO [10]

Feature-based vs Direct Methods

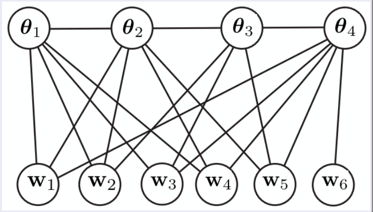


Feature-Based
Direct
Input Images
Input Images
Extract & Match Features (SIFT / SURF / ...)
abstract image to feature observations
keep full images (no abstraction)
Track: min. reprojection error (point distances)
Map: est. feature-parameters (3D points / normals)
Track: min. photometric error (intensity differences)
Map: est. per-pixel depth (semi-dense depth map)

Figure: Feature-based methods abstract images to features and discard all other information. Direct methods maps and tracks directly using image intensities

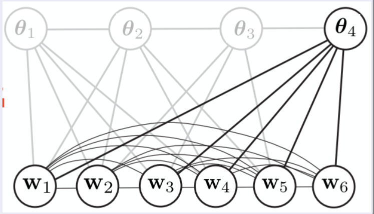
MonoSLAM - MRF

- vSLAM problem can be viewed as doing inference on a Markov Random Field (MRF)
- **Problem:** Becomes exponentially harder with time
- **Classic Solution:** pose BA as a filter (such as, EKF, particle filters, etc.)



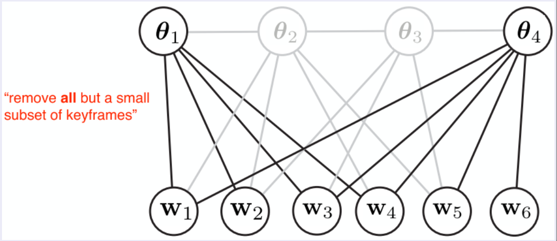
MonoSLAM - Filtering

- Eliminating prev. poses results in unwanted direct connections between 3D points
- **Problem1:** Waste computation time on frames with redundant information
- **Problem2:** Often times problematic (when device stops moving)



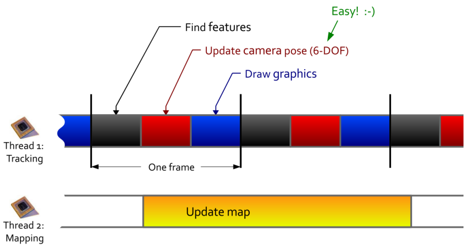
MonoSLAM - KF

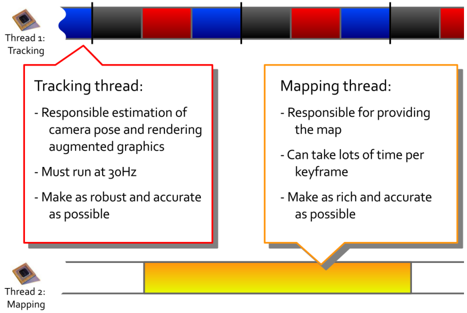
- Employ KF-BA
- Made popular by PTAM [2]



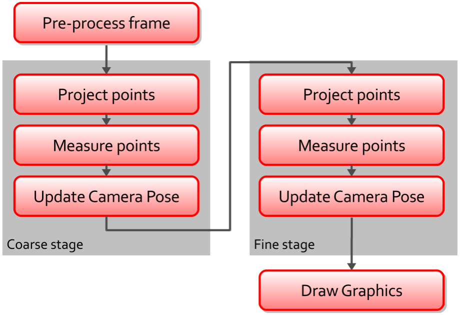
PTAM - Overview

- One of the foundations of many current SLAM research papers
- Designed to track handheld camera in small AR workspaces
- Updating entire map in real-time is expensive → Update Map only for KFs
- Separates camera tracking and map estimation tasks in parallel threads
  - Same Idea utilized in current SoTA vSLAM algorithms (e.g. ORB-SLAM)




Thread 1: Tracking
Tracking thread:
- Responsible estimation of camera pose and rendering augmented graphics
- Must run at 30Hz
- Make as robust and accurate as possible
Mapping thread:
- Responsible for providing the map
- Can take lots of time per keyframe
- Make as rich and accurate as possible
Thread 2: Mapping

PTAM - Tracking + Mapping



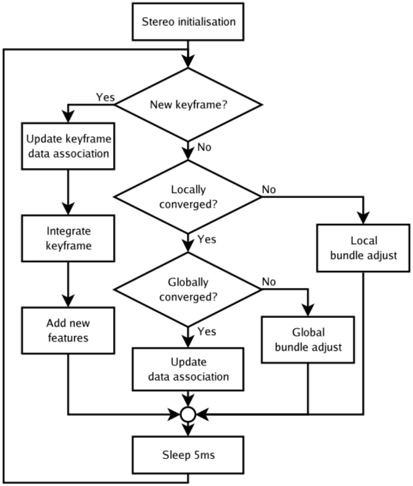
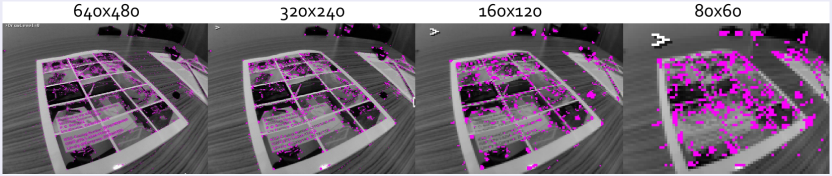


Figure 2: The asynchronous mapping thread. After initialisation, this thread runs in an endless loop, occasionally receiving new frames from the tracker.

PTAM - Tracking (Assumption: 3D Map points known and given)

Proprocessing step

$$\begin{tikzcd}
\boxed{\text{Input Frame}} \arrow[r] & \boxed{\text{Create four scaled image pyramid levels}} \arrow[r] & \boxed{\text{Detect FAST corners + BRIEF descriptors}}
\end{tikzcd}$$



PTAM - Tracking

Pose Update (computed iteratively)

Given: Set S of successful patch observations
Re-projection error for j^{th} 3D map point:

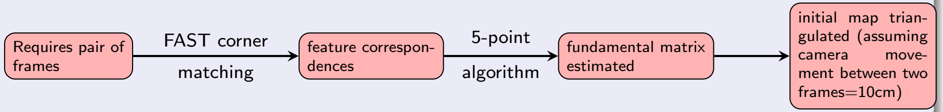
$$e_j = \begin{bmatrix} \hat{u}_j \\ \hat{v}_j \end{bmatrix} - \text{camproj}(exp_{se(3)}(\mu) E_{cw} p_j) \tag{1}$$

Minimize Tukey bi-weight re-projection error

$$\mu_{opt} = \arg\min_{\mu} \sum_{j \in S} Obj(\frac{e_j}{\sigma_j}, \sigma_T) \tag{2}$$

PTAM - Mapping

Stereo-based System Initialization

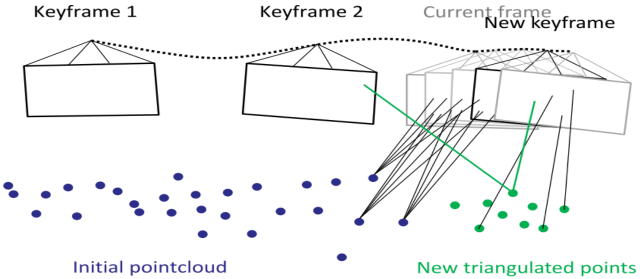

Requires pair of frames
FAST corner matching
feature correspondences
5-point algorithm
fundamental matrix estimated
initial map triangulated (assuming camera movement between two frames=10cm)

PTAM - Mapping

KF Insertion

Criteria for inserting new KF -

- Tracking quality should be good
- Minimum frame gap between consecutive KFs = 20 (hyperparameter)
- Camera must be minimum distance away from the nearest KF in the map
    - avoids stationary camera corrupting the map
    - ensures minimum stereo baseline for new features integration



PTAM - Mapping

Full Bundle Adjustment (BA)

- Joint optimization over all KF poses and all 3D map-points

Minimize re-projection error of all points in all KFs

$$\{\{\boldsymbol{\mu}_2..\boldsymbol{\mu}_N\},\{\boldsymbol{p}'_1..\boldsymbol{p}'_M\}\} = \underset{\{\{\boldsymbol{\mu}\},\{\boldsymbol{p}\}\}}{\operatorname{argmin}} \sum_{i=1}^{N} \sum_{j\in S_i} \operatorname{Obj}\left(\frac{|\boldsymbol{e}_{ji}|}{\sigma_{ji}}, \sigma_T\right)$$

- **Problem:** Computational complexity $= O(N^3)$ (system scales with cube of #KFs$= N$) $\implies$ expensive computation as map size increases
- **Solution:** Allow mapping thread to perform **Local BA**

Local Bundle Adjustment (LBA)

Minimize re-projection error of all 3D map-points in only a subset of KFs

$$\{\{\boldsymbol{\mu}_{x\in X}\},\{\boldsymbol{p}'_{z\in Z}\}\} = \underset{\{\{\boldsymbol{\mu}\},\{\boldsymbol{p}\}\}}{\operatorname{argmin}} \sum_{i\in X\cup Y} \sum_{j\in Z\cap S_i} \operatorname{Obj}(i,j)$$

- X: most recent KF $+$ $(N-1)$ KFs nearest to it in the map
- Z: all 3D map-points visible in any of the KFs $\in$ X
- Y: any KF in which a measurement of any point in Z has been made
- Optimizes - {pose of KFs in X} $+$ {all map points visible in X} using all measurements of points $\in Z$

PTAM - Achievements & Limitations

Achievements

- First KF-based monocular SLAM
- Bundle adjustment (BA) possible in real-time
- Still considered as a reference system

Limitations

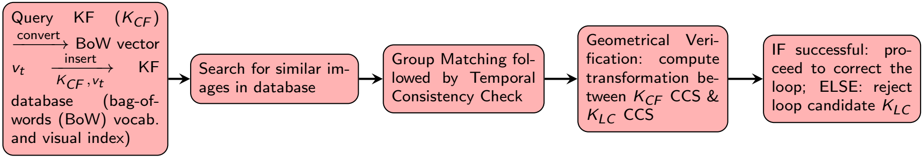
- Small scale operation
- Relocalization with little invariance to viewpoint
- No loop detection implemented
- Initialization requires human intervention

Loop Closure $\equiv$ Loop Detection $\xrightarrow[\text{by}]{\text{followed}}$ Loop Correction

- Loop constraints can be found by evaluating visual similarity between current frame and past frames
- If loop detected $\rightarrow$ compute similarity transformation (informs about drift accumulated in the loop) between current KF & loop KF
- Both sides of loop are aligned and duplicated points are fused
- Finally, pose-graph optimization over similarity constraints sim(3) performed to achieve global consistency

Loop Closure Detection (aka Place Recognizer)

Place Recognizer: Finding most similar image of a template image in a large image database (image retrieval problem)



Query KF (K_{CF}) $\xrightarrow{\text{convert}}$ BoW vector $v_t \xrightarrow[K_{CF}, v_t]{\text{insert}}$ KF database (bag-of-words (BoW) vocab. and visual index)

→ Search for similar images in database

→ Group Matching followed by Temporal Consistency Check

→ Geometrical Verification: compute transformation between K_{CF} CCS & K_{LC} CCS

→ IF successful: proceed to correct the loop; ELSE: reject loop candidate K_{LC}

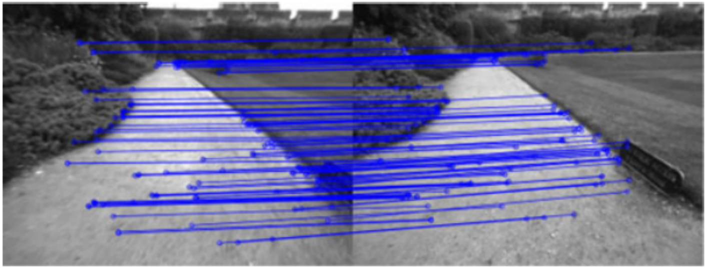


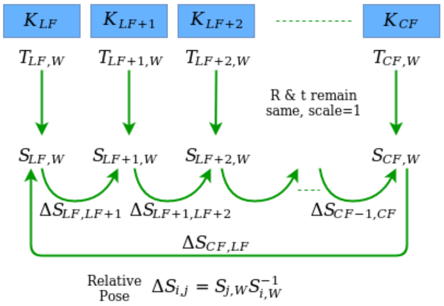
Fig. 4. Example of loop detected in the NewCollege sequence. We draw the inlier correspondences supporting the similarity transformation found.

Loop Closing Algorithm

Backend BA will optimize points + poses of the whole map $\rightarrow$ Due to drift accumulated, current map state is far from optimal $\rightarrow$ Compute an initial solution optimizing the pose graph from current KF K_{CF} to loop KF K_{LF} pose

- Convert all poses $T_{i,W} \in \mathrm{SE}(3)$ into $S_{i,W} \in \mathrm{sim}(3)$
- Compute relative transformation $\Delta S_{i,j}$ between consecutive poses
- Minimize residual error between pose $S_{i,W}$ & $S_{j,W}$ wrt constraint $\Delta S_{i,j}$ in the tangent space sim(3) and in minimal repr. $r_{i,j} = \left\{ \log_{\mathrm{sim}(3)}(\Delta S_{i,j}.S_{i,W}.S_{j,W}^{-1}) \right\}$
- Initialize: $r_{i,j} = 0 \ \forall i,j$ except loop residual $r_{CF,LF}$
- Optimize poses to distribute this error along the graph

$$\min \sum_{i,j} r_{i,j}^T \Lambda_{i,j}^{-1} r_{i,j}$$



Loop Closing Algorithm

After pose optimization, correct 3D points associated with them $\rightarrow$ For each 3D point x_j, select source KF pose $T_{i,W}$, re-map using optimized pose $S_{i,W}^{opt}$, $x_j^{opt} = (S_{i,W}^{opt})^{-1} T_{i,W} x_j$ $\rightarrow$ Convert optimized similarity transformation $S_{i,W}^{opt}$ back to 3D absolute transformation $T_{i,W}^{opt}$ $S_{i,W}^{opt} = \begin{pmatrix} sR & t \\ 0 & 1 \end{pmatrix} \rightarrow T_{i,W}^{opt} = \begin{pmatrix} R & \frac{1}{s}t \\ 0 & 1 \end{pmatrix}$

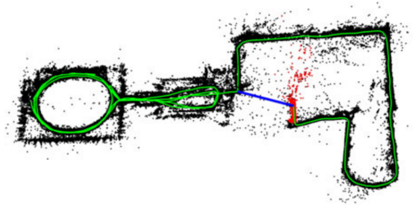


Figure: Map before loop closure. Loop Closure Match (in blue). Local Map for tracking at that moment (in red)

$\xrightarrow[\text{correction}]{\text{loop}}$

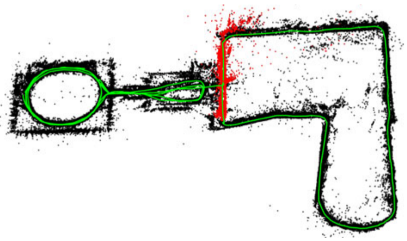


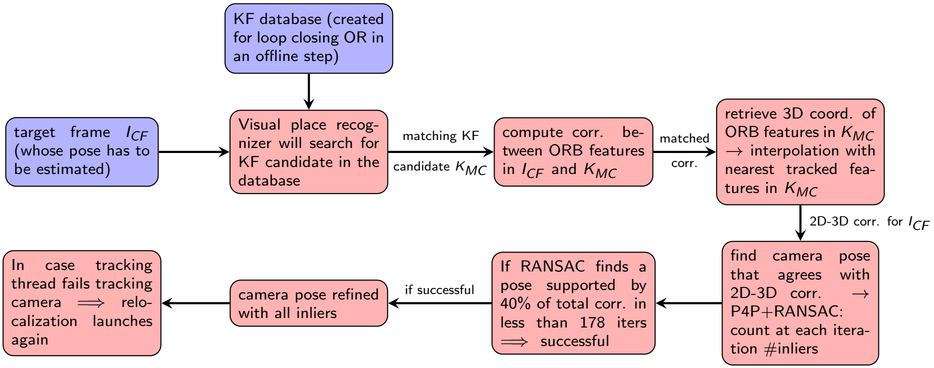
Figure: Map after loop closure. Local Map for tracking extended along both sides of the loop

Re-Localization

Situation 1: Pose estimation failed (tracking lost) due to occlusions or abrupt movements
Situation 2: Initialization to localize in a previous map

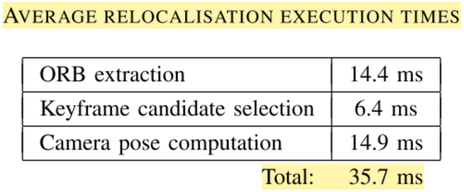
$\downarrow$

Solution: Perform Global Re-Localization



KF database (created for loop closing OR in an offline step)
target frame I_{CF} (whose pose has to be estimated)
Visual place recognizer will search for KF candidate in the database
matching KF candidate K_{MC}
compute corr. between ORB features in I_{CF} and K_{MC}
matched corr.
retrieve 3D coord. of ORB features in K_{MC} $\rightarrow$ interpolation with nearest tracked features in K_{MC}
2D-3D corr. for I_{CF}
find camera pose that agrees with 2D-3D corr. $\rightarrow$ P4P+RANSAC: count at each iteration #inliers
If RANSAC finds a pose supported by 40% of total corr. in less than 178 iters $\Longrightarrow$ successful
if successful
camera pose refined with all inliers
In case tracking thread fails tracking camera $\Longrightarrow$ relocalization launches again

Re-Localization



AVERAGE RELOCALISATION EXECUTION TIMES

ORB extraction	14.4 ms
Keyframe candidate selection	6.4 ms
Camera pose computation	14.9 ms
Total:	35.7 ms

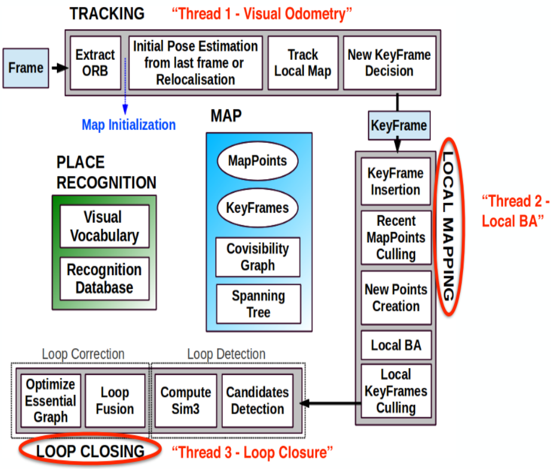
ORB-SLAM - Overview

Characteristics & Main Contributions

- Feature-based SLAM - ORB features used for all tasks (tracking, mapping, relocalization, loop closing)
- Real-time performance on CPU (3 parallel threads)
- Operates in small and large (indoor + outdoor) environments
- Generates compact, trackable map - only grows if scene content changes - allowing lifelong operation
- Use of covisibility graph - tracking & mapping focussed in local covisible area, independent of global map size
- Robust to severe motion clutter; Allows loop closing based on pose-graph optimization of Essesntial Graph (E_G)
- Allows relocalization with significant invariance to viewpoint & illumination $\rightarrow$ allows recovery from tracking failure and map reuse
- Full automatic initialization $\xrightarrow[\text{creation of}]{\text{permits}}$ initial map of planar + non-planar scenes
- Survival of the fittest strategy for 3D map points and KFs
- Powerful computer (e.g. i7) will ensure real-time performance and provide more stable and accurate results

ORB-SLAM - Overview

1. **Tracking:**
   - localizing camera every frame
   - deciding when to insert a new KF
2. **Local Mapping:**
   - processes new KFs + local BA
   - Triangulate new 3D map points: new corr. in new KF searched in connected KFs $\in C_G$
   - Culling policy to retain only high quality map points and KFs
3. **Loop Closing:**
   - search for loops with every KF
   - compute similarity transformation (tells drift accumulated in loop)
   - Both loop sides aligned $\rightarrow$ duplicated points fused
   - pose-graph optimization over Essential graph



ORB-SLAM - Covisibility Graph (C_G) & Essential Graph (E_G)

- Covisibility information between KFs (repr. as undirected weighted graph)
- Nodes: KFs; Edges exist between K_i & K_j, if they share observations of common map points; #common map points = edge weight (θ)
- From initial KF, Incrementally build spanning tree S_{C_G}(connected subgraph of C_G with minimal number of edges)
- New KF $\xrightarrow[\text{spanning tree}]{\text{insertion in}}$ linked to KF with most common point observations
- C_G can be very dense $\xrightarrow{\text{propose}}$ E_G (sparser subgraph of C_G) - retains all nodes, less edges
- E_G contains S_{C_G} + subset of edges from C_G with high covisibility + loop closure edges

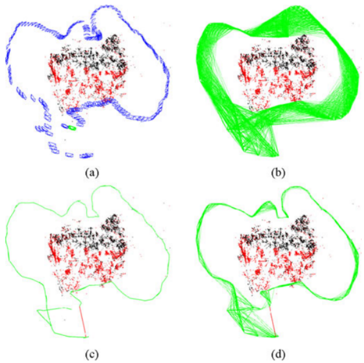


Fig. 2. Reconstruction and graphs in the sequence *fr3_long_office_household* from the TUM RGB-D Benchmark [38]. (a) Keyframes (blue), current camera (green), map points (black, red), current local map points (red). (b) Covisibility graph. (c) Spanning tree (green) and loop closure (red). (d) Essential graph.

ORB-SLAM - Automatic Map Initialization

- **Goal:** Compute relative pose between two frames to triangulate an initial set of 3D map points
- **Preferred qualities:** scene-independent (planar or non-planar) + No human intervention (to select a good 2-view config. with significant parallax)
- **Proposal:** Compute two geometrical models (in parallel) and recover the relative pose after selecting one model
  1. a homography (assuming a planar scene, nearly planar or with low parallax)
  2. a fundamental matrix (assuming a non-planar scene with enough parallax)
- Only initializes when it is certain that the 2-view config. is safe → avoids initializing corrupted map

find initial corr. $x_c \leftrightarrow x_r$ using ORB features between I_{CF} and I_{ref} → compute a homography ($x_c = H_{cr}x_r$) and a fundamental matrix ($x_c^T F_{cr} x_r = 0$) inside a RANSAC scheme. At each iter. compute a score for both models and keep the homography and fundamental matrix with highest score → After model selection, retrieve motion hypothesis associated. In case of homography, retrieve 8-motion hypothesis using [], otherwise retrieve 4-motion hypothesis with SVD method in [] → Triangulate the solutions

ORB-SLAM - Tracking (performed with every frame)

Initial pose estimation: use constant velocity model to predict pose and perform guided search of the map points observed in the last frame $\xrightarrow[corr.]{2D-3D}$ Pose optimization using PnP. If tracking lost, use global re-localization

$\xrightarrow[pose]{camera}$

Track Local Map: project local map (set of KFs K_1 which share map points with I_{CF} + set of KFs K_2 with neighbours to K_1 in C_G + reference KF $K_{ref} \in K_1$ sharing most map points with I_{CF}. Each map point in K_1 & K_2 searched for in I_{CF}. Camera pose optimized with all map points found in the frame

$\rightarrow$

New KF decision: Decide if I_{CF} used as new KF. Insert KFs as fast as possible (makes tracking more robust to challenging camera movements)

ORB-SLAM - Local Mapping (performed with every new KF K_i)

KF insertion: Update C_G (adding a new node K_i, updating edges from shared map points with other KFs), Update S_{C_G} (linking K_i with the KF with most points in common)

$\rightarrow$

Recent Map points Culling: Map points (in order to be retained), must pass restrictive test (ensures they are trackable and not wrongly triangulated) during first 3 KFs after creation.

$\rightarrow$

New Map Point Creation: Triangulating matching ORB feature pairs from connected KFs in C_G

$\downarrow$

Local BA: Optimizes current KF K_i + all KFs connected to it in C_G K_c + all map points seem by those KFs. All other KFs that see those points but not connected to currently processed KF included in optimization but remain fixed

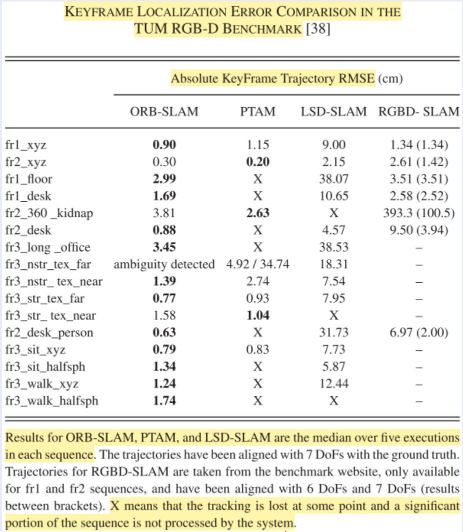
$\leftarrow$

Local KF culling: In order to maintain compact map repr., local mapping detects redundant KFs & deletes them (WHY? BA complexity grows with KFs + enable lifelong operation in the same environment as the #KFs will remain bounded, unless the visual content in scene changes)

ORB-SLAM - Experiments & Insightful Results

TUM RGB-D dataset: Localization Accuracy

- Typical Accuracy Indoors = 1-3 cm
- Accuracy (in open trajectories): ORBSLAM $\equiv$ PTAM
- Accuracy (in closed trajectories): ORBSLAM $>$ PTAM (trivial)
- Accuracy (overall): ORB-SLAM & PTAM $>$ LSD-SLAM & RGBD-SLAM
- Accuracy (in dynamic scenes): ORBSLAM $\gg$ LSDSLAM



KEYFRAME LOCALIZATION ERROR COMPARISON IN THE TUM RGB-D BENCHMARK [38]

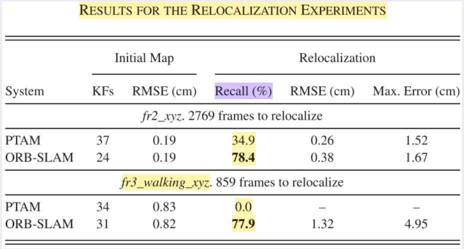
	Absolute KeyFrame Trajectory RMSE (cm)			
	ORB-SLAM	PTAM	LSD-SLAM	RGBD- SLAM
fr1_xyz	**0.90**	1.15	9.00	1.34 (1.34)
fr2_xyz	0.30	**0.20**	2.15	2.61 (1.42)
fr1_floor	**2.99**	X	38.07	3.51 (3.51)
fr1_desk	**1.69**	X	10.65	2.58 (2.52)
fr2_360 _kidnap	3.81	**2.63**	X	393.3 (100.5)
fr2_desk	**0.88**	X	4.57	9.50 (3.94)
fr3_long _office	**3.45**	X	38.53	–
fr3_nstr_tex_far	ambiguity detected	4.92 / 34.74	18.31	–
fr3_nstr_ tex_near	**1.39**	2.74	7.54	–
fr3_str_tex_far	**0.77**	0.93	7.95	–
fr3_str_ tex_near	1.58	**1.04**	X	–
fr2_desk_person	**0.63**	X	31.73	6.97 (2.00)
fr3_sit_xyz	**0.79**	0.83	7.73	–
fr3_sit_halfsph	**1.34**	X	5.87	–
fr3_walk_xyz	**1.24**	X	12.44	–
fr3_walk_halfsph	**1.74**	X	X	–

Results for ORB-SLAM, PTAM, and LSD-SLAM are the median over five executions in each sequence. The trajectories have been aligned with 7 DoFs with the ground truth. Trajectories for RGBD-SLAM are taken from the benchmark website, only available for fr1 and fr2 sequences, and have been aligned with 6 DoFs and 7 DoFs (results between brackets). X means that the tracking is lost at some point and a significant portion of the sequence is not processed by the system.

ORB-SLAM - Experiments & Insightful Results

TUM RGB-D dataset: Re-localization Accuracy

- System able to localize from very different viewpoints and under **moderate dynamic changes**



RESULTS FOR THE RELOCALIZATION EXPERIMENTS

	Initial Map		Relocalization		
System	KFs	RMSE (cm)	Recall (%)	RMSE (cm)	Max. Error (cm)
		fr2_xyz. 2769 frames to relocalize			
PTAM	37	0.19	34.9	0.26	1.52
ORB-SLAM	24	0.19	**78.4**	0.38	1.67
		fr3_walking_xyz. 859 frames to relocalize			
PTAM	34	0.83	0.0	–	–
ORB-SLAM	31	0.82	**77.9**	1.32	4.95

- Accuracy (re-localization): ORBSLAM $\gg \approx 2\times$PTAM

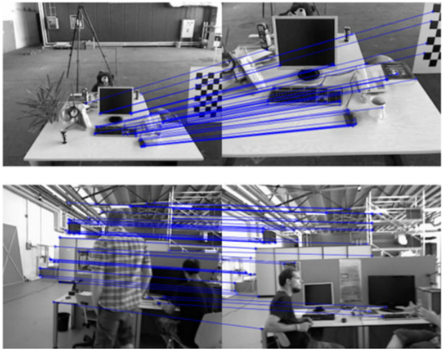


Fig. 8. Example of challenging relocalizations (severe scale change, dynamic objects) that our system successfully found in the relocalization experiments.

ORB-SLAM - Experiments & Insightful Results

TUM RGB-D dataset: Lifelong Operation (requirement for SLAM system)

- Map grows with scene content changes, but not with time

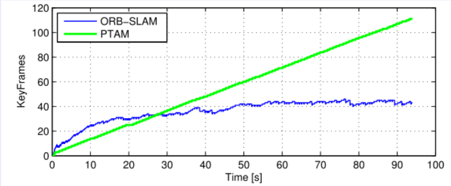


Fig. 9. Lifelong experiment in a static environment where the camera is always looking at the same place from different viewpoints. PTAM is always inserting keyframes, while ORB-SLAM is able to prune redundant keyframes and maintains a bounded-size map.

ORB-SLAM - Experiments & Insightful Results

KITTI Odometry Dataset

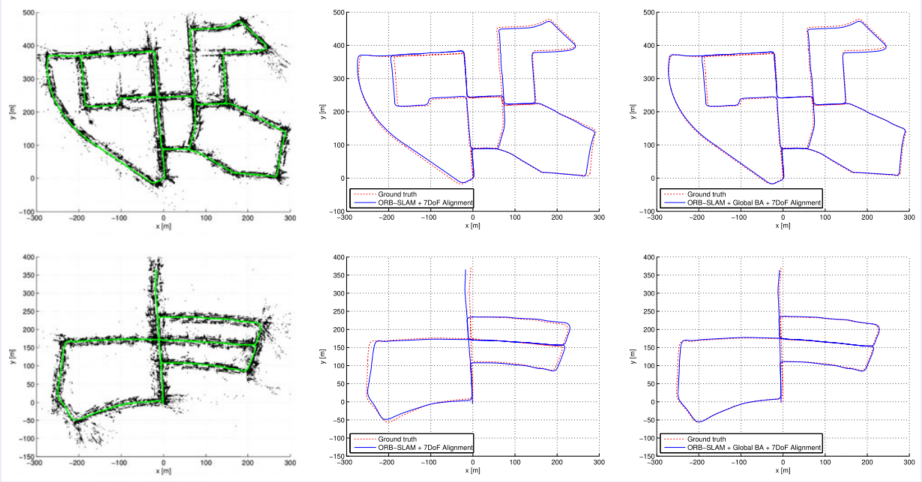


Figure: Left: Points + KF trajectory, Center: Trajectory without Global BA, Right: Trajectory with full BA

ORB-SLAM - Conclusions & Summary

Limitations

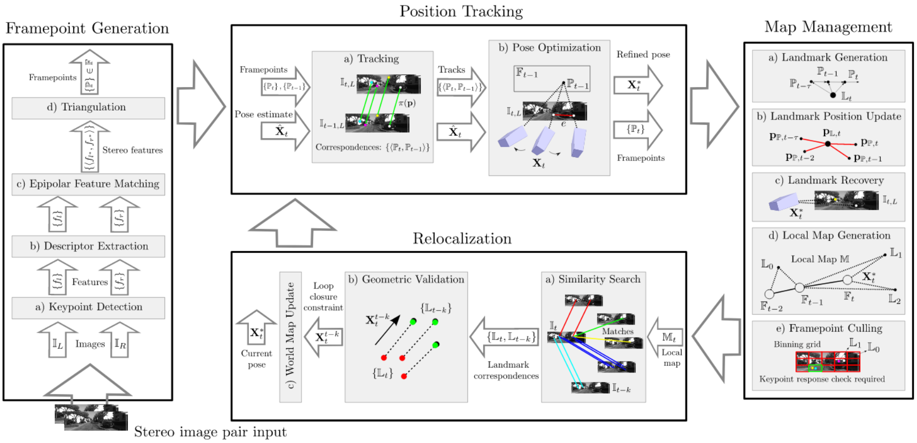
- Monocular $\rightarrow$ absolute scale is unknown
- Needs texture: will fail with large plain walls
- Map is too sparse for interaction with the environment
- Can only deal with moderate dynamic scene changes

Resources

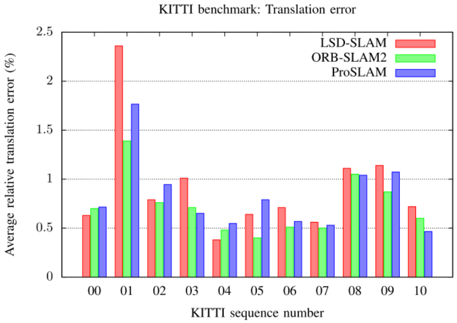
- Source Code: https://github.com/raulmur/ORB_SLAM2
- Project Webpage: http://webdiis.unizar.es/~raulmur/orbslam/

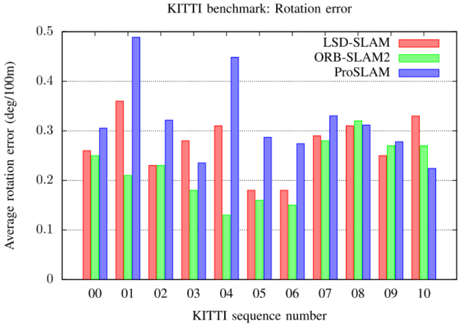
ProSLAM (Programmers SLAM)

- lightweight, stereo-based, substantially lower computational requirements
- Online, real-time, single-core CPU
- Designed for easy understanding and implementation (LOL!!!)



ProSLAM-Comparisons





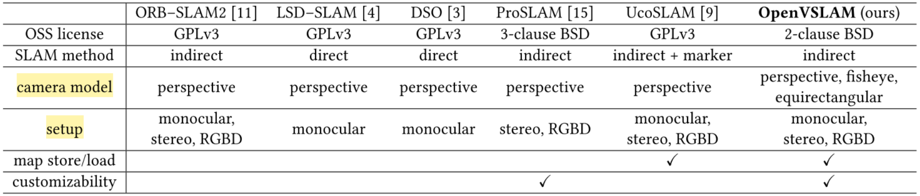
- Translation error: Comparable with competing methods
- Rotation error: Weaker compared to competing methods (because of NO BA or loop closing)

OpenVSLAM ≡ ORBSLAM + UcoSLAM + ProSLAM

Characteristics & Main Contributions:

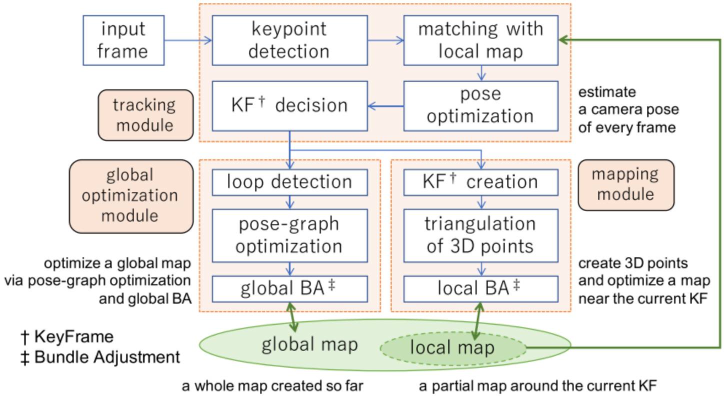
- Indirect Method of visual SLAM
- Framework with high usability and extensibility (Monocular, stereo and RGB-D)
- Created maps can be stored and re-loaded (for localization with pre-built maps)
- Can deal with various camera models (perspective, fisheye, equirectangular)

Table 1: Comparison of several open-source visual SLAM frameworks.



	ORB–SLAM2 [11]	LSD–SLAM [4]	DSO [3]	ProSLAM [15]	UcoSLAM [9]	**OpenVSLAM** (ours)
OSS license	GPLv3	GPLv3	GPLv3	3-clause BSD	GPLv3	2-clause BSD
SLAM method	indirect	direct	direct	indirect	indirect + marker	indirect
camera model	perspective	perspective	perspective	perspective	perspective	perspective, fisheye, equirectangular
setup	monocular, stereo, RGBD	monocular	monocular	stereo, RGBD	monocular, stereo, RGBD	monocular, stereo, RGBD
map store/load					✓	✓
customizability				✓		✓

OpenVSLAM - Overview



input frame
keypoint detection
matching with local map
tracking module
KF † decision
pose optimization
estimate a camera pose of every frame
global optimization module
loop detection
pose-graph optimization
global BA ‡
KF † creation
triangulation of 3D points
local BA ‡
mapping module
optimize a global map via pose-graph optimization and global BA
create 3D points and optimize a map near the current KF
† KeyFrame
‡ Bundle Adjustment
global map
local map
a whole map created so far
a partial map around the current KF

Figure 2: Main modules of OpenVSLAM: tracking, mapping, and global optimization modules.

OpenVSLAM - Experiments & Insightful Results

Tracking Accuracy: OpenVSLAM $\sim$ ORBSLAM

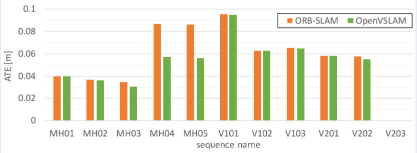


Figure 3: Absolute trajectory errors on the 11 sequences in EuRoC MAV dataset (monocular). Lower is better.

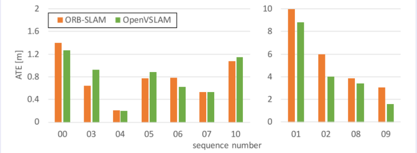


Figure 5: Absolute trajectory errors on the 11 sequences in KITTI Odometry dataset (stereo). Lower is better.

Tracking Time: OpenVSLAM faster than ORBSLAM

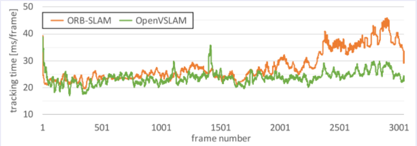


Figure 4: Tracking times on the `MH_02` sequence of EuRoC MAV dataset (monocular). The table shows mean and median tracking times on each of the two frameworks. The graph shows the change in tracking times. Lower is better.

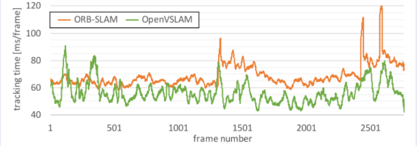


Figure 6: Tracking times on the sequence number `05` of KITTI Odometry dataset (stereo). The table shows mean and median tracking times on each of the two frameworks. The graph shows the change in tracking times. Lower is better.

Dealing with Dynamic Scenes

Overview

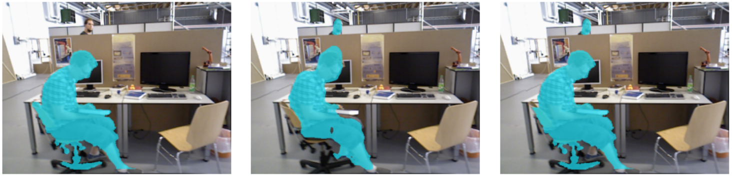
- **Assumption:** (in most current SLAM approaches) Static Environment $\Longrightarrow$ Not capable of handling dynamic scenarios
- Poses challenge for both tracking and mapping
- Detecting and Dealing with dynamic scenes required to estimate **stable**, **long-term (lifelong) re-usable maps**
- Two types of dynamic scene situations:
  1. Living movement: Person, cat, dog, etc.
  2. Non-living movement: parked car, buses, chair, etc.

Challenges

1. Mitigate the influence of dynamic objects on pose estimation
2. Prevent tracking algorithm from using matches belonging to dynamic objects
3. Prevent mapping algorithm from including dynamic objects as part of 3D map
4. How to complete the 3D map part occluded (temporally) by dynamic object ?

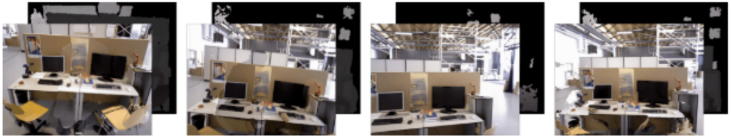
DynaSLAM - Overview

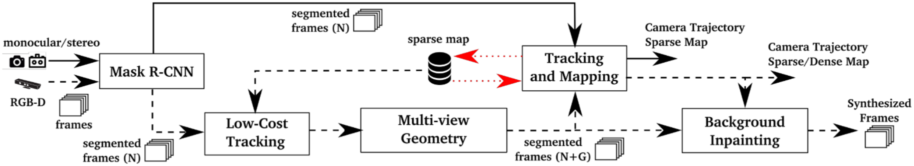
- Most existing SLAM approaches and datasets assume static environment (scene rigidity)
- Built on top of ORB-SLAM2, Adds motion segmentation approach (making it robust in dynamic environments)
- Compatible with Monocular, Stereo and RGB-D configurations
- System tracks camera + creates static, reusable 3D map
- In-painting of frame background occluded by dynamic objects (only for RGB-D frames)
- Detection of moving objects can be done by -
    - deep learning
    - multi-view geometry
    - combination of deep learning and multi-view geometry


(a) Using Multi-view Geometry. (b) Using Deep Learning. (c) Using Geometry and Deep Learning.

DynaSLAM - Technical Details

(a) Input RGB-D frames with dynamic content.


(b) Output RGB-D frames. Dynamic content has been removed. Occluded background has been reconstructed with information from previous views.



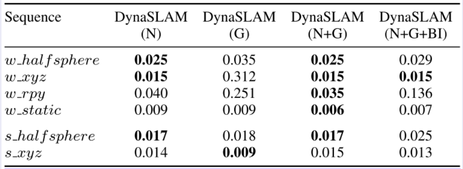
DynaSLAM - Technical Details

- Uses MaskRCNN for pixel-wise segmentation of (a priori) dynamic objects or scenes
- DynaSLAM $\equiv$ front-end dynamic scene detection and removal $\xrightarrow[\text{by}]{\text{followed}}$ ORBSLAM2

DynaSLAM - Experiments & Insightful Results

TUM RGB-D dataset: Different variants of DynaSLAM for 6 sequences

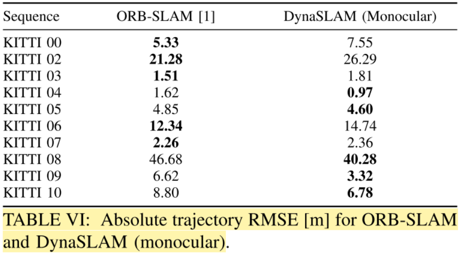
- Dynamic object detection + segmentation
  - N: Mask RCNN
  - G: multi-view geometry
  - (N+G): Mask RCNN + multi-view geometry
  - (N+G+BI): background in-painting before tracking and mapping
- System (N+G) - **most accurate in most sequences**



Sequence	DynaSLAM (N)	DynaSLAM (G)	DynaSLAM (N+G)	DynaSLAM (N+G+BI)
$w\_halfsphere$	**0.025**	0.035	**0.025**	0.029
w_xyz	**0.015**	0.312	**0.015**	**0.015**
w_rpy	0.040	0.251	**0.035**	0.136
w_static	0.009	0.009	**0.006**	0.007
$s\_halfsphere$	**0.017**	0.018	**0.017**	0.025
s_xyz	0.014	**0.009**	0.015	0.013

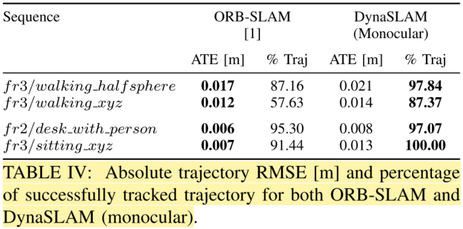
TABLE I: Absolute trajectory RMSE [m] for several variants of DynaSLAM (RGB-D).

DynaSLAM - Experiments & Insightful Results



Sequence	ORB-SLAM [1]	DynaSLAM (Monocular)
KITTI 00	**5.33**	7.55
KITTI 02	**21.28**	26.29
KITTI 03	**1.51**	1.81
KITTI 04	1.62	**0.97**
KITTI 05	4.85	**4.60**
KITTI 06	**12.34**	14.74
KITTI 07	**2.26**	2.36
KITTI 08	46.68	**40.28**
KITTI 09	6.62	**3.32**
KITTI 10	8.80	**6.78**

TABLE VI: Absolute trajectory RMSE [m] for ORB-SLAM and DynaSLAM (monocular).



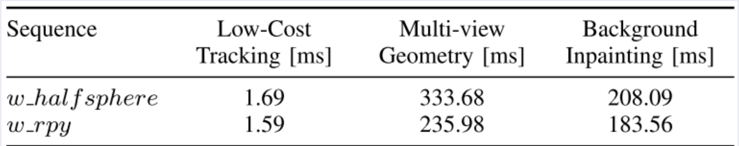
Sequence	ORB-SLAM [1]		DynaSLAM (Monocular)	
	ATE [m]	% Traj	ATE [m]	% Traj
$fr3/walking\_halfsphere$	**0.017**	87.16	0.021	**97.84**
$fr3/walking\_xyz$	**0.012**	57.63	0.014	**87.37**
$fr2/desk\_with\_person$	**0.006**	95.30	0.008	**97.07**
$fr3/sitting\_xyz$	**0.007**	91.44	0.013	**100.00**

TABLE IV: Absolute trajectory RMSE [m] and percentage of successfully tracked trajectory for both ORB-SLAM and DynaSLAM (monocular).

DynaSLAM - Experiments & Insightful Results

Timing Analysis

- **Not optimized for real-time performance on CPU**
  - Uses MaskRCNN for dynamic object segmentation (195 ms/frame on NVIDIA Tesla M40 GPU)



Sequence	Low-Cost Tracking [ms]	Multi-view Geometry [ms]	Background Inpainting [ms]
$w\_halfsphere$	1.69	333.68	208.09
w_rpy	1.59	235.98	183.56

TABLE VII: DynaSLAM average computational time [ms].

- Additional delay due to multi-view geometry and background inpainting

DynaSLAM - Conclusions & Summary

Future Extensions

- Real-time performance
- More realistic synthesized background inpainted RGB frames using (may be) GANs

Resources

- Source Code: https://github.com/BertaBescos/DynaSLAM
- Project Webpage: https://bertabescos.github.io/DynaSLAM/

DSO - Overview

Characteristics & Main Contributions

- **Direct** and **Sparse** approach to monocular VO (does not depend on KP features)
- Runs in real-time on CPU
- Joint optimization of photometric error over all model parameters (camera poses, camera intrinsics, inverse depth values)
- Geometric Repr. - 3D points as inverse depth in a ref. frame (1 DoF)
- Integrates full photometric calibration, accounting for exposure time, lens vignetting and non-linear response function
- Omit geometric prior $\rightarrow$ evaluate photometric error for each pixel over a small neighbourhood
- Incrementally eliminating old states to maintain real-time performance
- Incorporating photometric calibration increases performance compared to brightness constancy assumption
- Outperforms other SoTA approaches (direct+indirect), in terms of robustness and accuracy

DSO - Geometric Prior

Used by other direct methods (e.g. DTAM, LSD-SLAM)

Advantages

- Makes 3D reconstruction denser, locally more accurate and visual appealing

Drawbacks

- Can have unwanted effects on BA problem
- Can introduce bias $\rightarrow$ reduce long-term, large-scale accuracy

DSO - Model Formulation

Error function depends on following variables -

- Pixel's inverse depth - d_p
- Camera intrinsics - K
- Poses of involved frames - T_i, T_j
- Brightness transfer function params - a_i, a_j, b_i, b_j

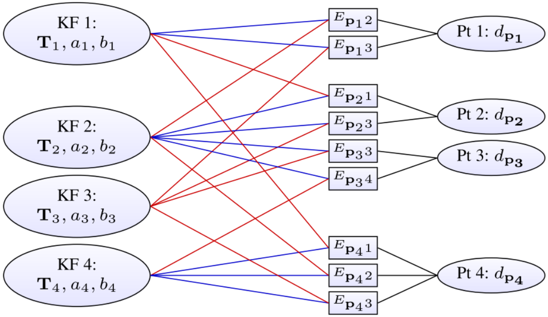


Figure: 4 KFs + 4 points (1 in KF1, 2 in KF2, 1 in KF4); Energy term $E_{p_i j}$ depends on point p_i host frame (blue), frame the point is observed in (red), p_i's inverse depth (black), and K

DSO - VO Front-End

- Decides which points, frames are used and in which frames a point is visible
- Initialization for new params (reqd. for minimizing highly non-convex energy function)
- Decide when a point or frame should be marginalized (eliminated)

Frame Management

Always keeps window of N_f active KFs
Step 1: Every new frame is tracked wrt reference frames (active KFs)
Step 2: New tracked frame $\rightarrow$ discarded or used to create new KF
Step 3: Once new KF and new points created $\rightarrow$ total photometric error optimized $\rightarrow$ Marginalize one or more frames

DSO - VO Front-End

3D Point Management

Always keeps fixed number of N_p active points (equally distribute across space and active frames). 3D point-cloud density directly corr. to N_p
Step 1: Identify N_p candidate points in each new KF
Step 2: Candidate points not immediately added to optimization $\xrightarrow{\text{instead}}$ tracked in subsequent frames $\xrightarrow{\text{generate}}$ coarse depth value
Step 3: Choose a number of candidate points (from across all frames in the optimization window) to be added to optimization window

DSO - Experiments & Insightful Results

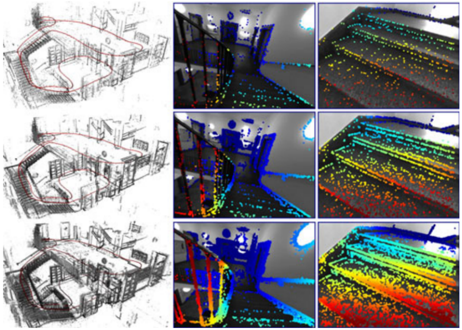


Figure: (left) 3D point-cloud density; (right) N_p active points projected into most recent KF for $N_p = 500$(top), 2000(middle), 10000(bottom)

DSO - Experiments & Insightful Results

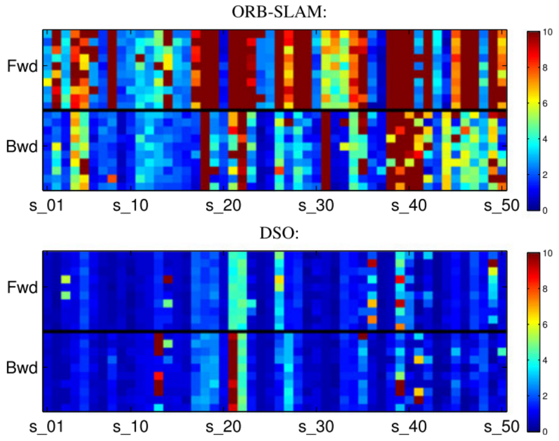
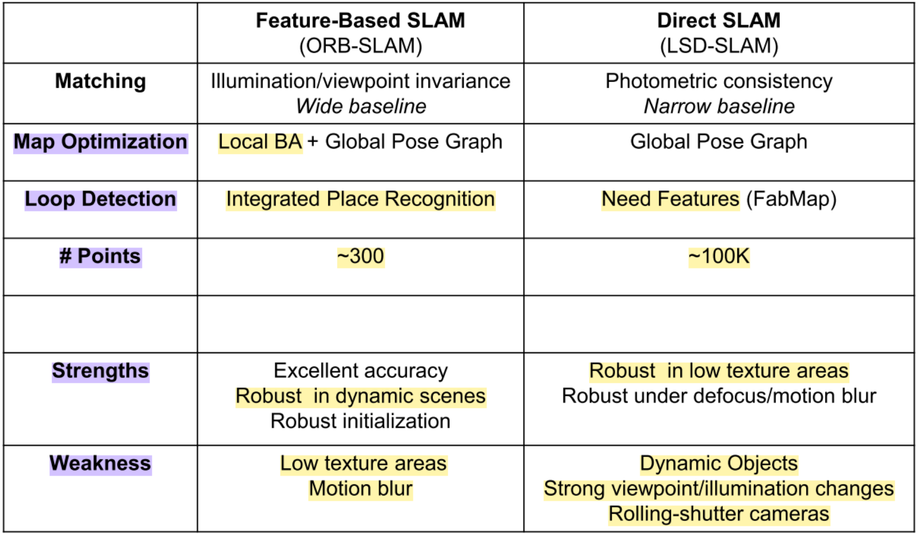


Figure: Error values for TUM monoVO dataset [17]

DSO - Conclusions & Summary

- DSO with loop closure: [video]
- Source Code: [github]

Sparse/Feature-based vs Dense/Direct Methods



	Feature-Based SLAM (ORB-SLAM)	**Direct SLAM** (LSD-SLAM)
Matching	Illumination/viewpoint invariance *Wide baseline*	Photometric consistency *Narrow baseline*
Map Optimization	Local BA + Global Pose Graph	Global Pose Graph
Loop Detection	Integrated Place Recognition	Need Features (FabMap)
# Points	~300	~100K
Strengths	Excellent accuracy Robust in dynamic scenes Robust initialization	Robust in low texture areas Robust under defocus/motion blur
Weakness	Low texture areas Motion blur	Dynamic Objects Strong viewpoint/illumination changes Rolling-shutter cameras

Sparse/Feature-based vs Dense/Direct Methods

Feature-based Method Advantages

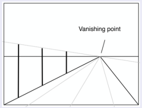


Figure: 1. Easier transition from images to geometry

Figure: 2. Wide-baseline Matching feasible (**Using Invariant Descriptors**)

Figure: 3. Illumination invariance (**Using Invariant Descriptors**)

- More robust to photometric + geometric distortions (e.g., originating from poor intrinsic calibration or imprecise lense or rolling shutter) compared to direct approach

Feature-based Method Limitations (or Challenges)

- Creates only a sparse 3D map of the world
- Does not sample across all image data
- Robustness/Invariance to photometric variations $\xrightarrow[\text{a cost}]{\text{comes at}}$ discarding potentially valuable information in those variations

Sparse/Feature-based vs Dense/Direct Methods

Direct Method Advantages

- Can be more useful for tasks (e.g. 3D dense map creation, AR-VR, etc.) than just camera localization
- More robust to photometric noise and achieves superior accuracy on well-calibrated data
- Can be correctly operated in more texture-less environments

Direct Method Limitations

- Performance affected by rolling shutter, autogain, auto-exposure artifacts (if not correctly modeled)
- Lower performance compared to indirect methods with rolling-shutter cameras (smartphones, consumer cameras, etc.)
- Assumes surface reflectance model $\rightarrow$ In real scenes produces its own artifacts
- Computationally more demanding compared to feature-based methods

Self-Localization - Proposed Method

SLAM-based localization using smartphones in real-world environment - still an ongoing work

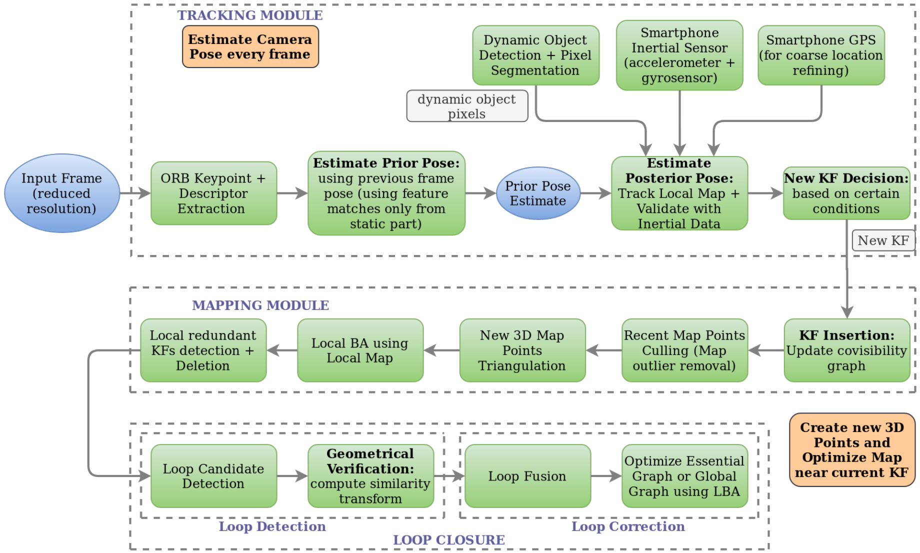
Priorities for self-localization using smartphones

- Tracking camera frames more important compared to 3D map creation
- Real-time, online, minimal in-place (without cloud support) computation, minimal memory usage

Tracking (every frame)

- Use smartphone IMU (acceleration + gyrosensor), GPS (for coarse location refining) and visual data for camera pose estimation
- Prior pose estimated using appropriate velocity model and previous camera pose estimate
- Dynamic object pixels discarded before posterior pose estimation
- Combine IMU data with prior pose estimate for improving tracking robustness

Self-Localization - Proposed Method Overview



TRACKING MODULE
Estimate Camera Pose every frame
Dynamic Object Detection + Pixel Segmentation
Smartphone Inertial Sensor (accelerometer + gyrosensor)
Smartphone GPS (for coarse location refining)
dynamic object pixels
Input Frame (reduced resolution)
ORB Keypoint + Descriptor Extraction
Estimate Prior Pose: using previous frame pose (using feature matches only from static part)
Prior Pose Estimate
Estimate Posterior Pose: Track Local Map + Validate with Inertial Data
New KF Decision: based on certain conditions
New KF
MAPPING MODULE
KF Insertion: Update covisibility graph
Recent Map Points Culling (Map outlier removal)
New 3D Map Points Triangulation
Local BA using Local Map
Local redundant KFs detection + Deletion
Loop Candidate Detection
Geometrical Verification: compute similarity transform
Loop Fusion
Optimize Essential Graph or Global Graph using LBA
Loop Detection
Loop Correction
LOOP CLOSURE
Create new 3D Points and Optimize Map near current KF

Self-Localization - Proposed Method

Implementation Details on Smartphones

- **Major Challenge:** Online dynamic object detection + computation limitation
- Most consumer cameras have Rolling Shutter (RS) $\xrightarrow{\text{introduces}}$ distortions
- Ignoring RS distortions gives good results in practice + saves computation time [13]
- Tracking and mapping can be performed at different frame resolution (suitable compromise)
- **Map Creation:** Offline one-time process for the desired environment
- **Map Selection:** GPS can be used for selecting appropriate map from the map database
- **Localization:** Performed with selected map using combination of place recognizer and IMU data

Self-Localization - Proposed Method

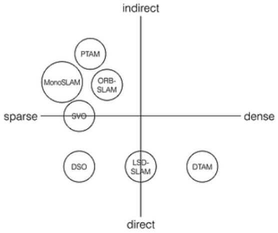
Preliminary Results using OpenVSLAM [20] in static environments

- Frame size (676x656) with fps $= 30$
- Mean tracking time $\approx$ 25 ms $\implies$ real-time performance
- CPU RAM usage for only localization (with given map) $\approx$ 1.9 GiB
- Total CPU memory usage $\approx$ 4 GiB
- Memory and RAM usage can be optimized further without accuracy compromise

Comparison of Representative (popular) Algorithms

Table 1 Comparison of representative algorithms

	Method	Map density	Global optimization	Loop closure
Mono-SLAM [26]	Feature	Sparse	No	No
PTAM [15]	Feature	Sparse	Yes	No
ORB-SLAM [38]	Feature	Sparse	Yes	Yes
DTAM [43]	Direct	Dense	No	No
LSD-SLAM [21]	Direct	Semi-dense	Yes	Yes
SVO [51]	Semi-direct	Sparse	No	No
DSO [53]	Direct	Sparse	No	No
KinectFusion [57]	RGB-D	Dense	No	No
Dense visual SLAM [23]	RGB-D	Dense	Yes	Yes
ElasticFusion [66]	RGB-D	Dense	Yes	Yes
SLAM++ [60]	RGB-D	Dense	Yes	Yes



Open Problems in vSLAM

Purely rotational motion (PRM)

- **Problem**: Disparities cannot be observed $\rightarrow$ mapping cannot continue during PRM with monocular vSLAM
- PRM - Not a problem for RGB-D vSLAM (since tracking and mapping can be done using obtained depth maps)
- Approach 1: Homography-based tracking (for PRM) and 6-DoF tracking (for other camera motion)
- Approach 2: Two types of 3D point repr. (depending on camera motion)
  - Points which can be observed with large disparities repr. as 3D points
  - Otherwise repr. as 3D rays

Rolling shutter (RS) distortion

- Shutter type - Important for accurate camera pose estimation
- RS - Each captured image row taken by different camera poses
- Most vSLAM algorithms assume global shutter; Most consumer cameras employ RS
- Interpolation-based approach used to estimate RS camera pose and trajectory

Open Problems in vSLAM

Failsafe SLAM & Recovery

- Current SLAM solvers vulnerable to presence of outliers due to non-convex optimization involved

Map Initialization

- To obtain accurate initial map $\rightarrow$ baseline should be wide
- After system launches $\rightarrow$ ideal camera motion (not always possible)
- Proposed solutions: User-friendly initialization, using reference objects (markers)

Dynamic Scenes

- Ex - indoor scene with many moving humans, street scene with moving cars, etc.

Open Problems in vSLAM

Tracking and Mapping in Degenerate Environments

Lack of discrete/unique features or poor lighting $\rightarrow$ Most vSLAM algorithms will fail with limited features

Related Online Resources

- OpenSLAM Org
- Awesome SLAM: A curated list of SLAM resources
- RTAB-Map: Real-Time Appearance-Based Mapping (RGB-D, Stereo, Lidar Graph-based SLAM)
- Kimera: an open-source library for real-time metric-semantic localization and mapping (Monocular camera not supported yet)
- g2o: a general framework for graph optimization
- evo: Python package for the evaluation of odometry and SLAM

References I

A. J. Davison et al. "MonoSLAM: Real-Time Single Camera SLAM". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.6 (June 2007), pp. 1052–1067. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2007.1049.

Georg Klein and David Murray. "Parallel Tracking and Mapping for Small AR Workspaces". In: *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*. Nara, Japan, Nov. 2007.

Mike Smith et al. "The New College Vision and Laser Data Set". In: *The International Journal of Robotics Research* 28.5 (2009), pp. 595–599. DOI: 10.1177/0278364909103911.

Hauke Strasdat, J. M. M. Montiel, and Andrew J. Davison. "Scale Drift-Aware Large Scale Monocular SLAM". In: *Robotics: Science and Systems*. 2010.

R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. "DTAM: Dense tracking and mapping in real-time". In: *2011 International Conference on Computer Vision*. Nov. 2011, pp. 2320–2327. DOI: 10.1109/ICCV.2011.6126513.

J. Sturm et al. "A Benchmark for the Evaluation of RGB-D SLAM Systems". In: *Proc. of the International Conference on Intelligent Robot Systems (IROS)*. Oct. 2012.

J. Engel, J. Sturm, and D. Cremers. "Semi-Dense Visual Odometry for a Monocular Camera". In: *IEEE International Conference on Computer Vision (ICCV)*. Sydney, Australia, Dec. 2013.

References II

A Geiger et al. "Vision meets robotics: The KITTI dataset". In: *The International Journal of Robotics Research* 32.11 (2013), pp. 1231–1237. DOI: 10.1177/0278364913491297.

J. Engel, T. Schöps, and D. Cremers. "LSD-SLAM: Large-Scale Direct Monocular SLAM". In: *European Conference on Computer Vision (ECCV)*. Sept. 2014.

C. Forster, M. Pizzoli, and D. Scaramuzza. "SVO: Fast semi-direct monocular visual odometry". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. May 2014, pp. 15–22. DOI: 10.1109/ICRA.2014.6906584.

A. Handa et al. "A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. May 2014, pp. 1524–1531. DOI: 10.1109/ICRA.2014.6907054.

Raul Mur-Artal and Juan Tardos. "Fast Relocalisation and Loop Closing in Keyframe-Based SLAM". In: *Proceedings - IEEE International Conference on Robotics and Automation* (June 2014), pp. 846–853. DOI: 10.1109/ICRA.2014.6906953.

T. Schöps, J. Engel, and D. Cremers. "Semi-Dense Visual Odometry for AR on a Smartphone". In: *International Symposium on Mixed and Augmented Reality*. Sept. 2014.

Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardos. "ORB-SLAM: A Versatile and Accurate Monocular SLAM System". In: *IEEE Transactions on Robotics* 31.5 (Oct. 2015), pp. 1147–1163. ISSN: 1941-0468. DOI: 10.1109/tro.2015.2463671. URL: http://dx.doi.org/10.1109/TRO.2015.2463671.

References III

Michael Burri et al. "The EuRoC micro aerial vehicle datasets". In: *The International Journal of Robotics Research* (2016). DOI: 10.1177/0278364915620033.

Jakob Engel, Vladlen Koltun, and Daniel Cremers. "Direct Sparse Odometry". In: *CoRR* abs/1607.02565 (2016). arXiv: 1607.02565. URL: http://arxiv.org/abs/1607.02565.

Jakob Engel, Vladyslav C. Usenko, and Daniel Cremers. "A Photometrically Calibrated Benchmark For Monocular Visual Odometry". In: *CoRR* abs/1607.02555 (2016). arXiv: 1607.02555. URL: http://arxiv.org/abs/1607.02555.

R. Ranftl et al. "Dense Monocular Depth Estimation in Complex Dynamic Scenes". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016, pp. 4058–4066. DOI: 10.1109/CVPR.2016.440.

Berta Bescós et al. "DynaSLAM: Tracking, Mapping and Inpainting in Dynamic Scenes". In: *CoRR* abs/1806.05620 (2018). arXiv: 1806.05620. URL: http://arxiv.org/abs/1806.05620.

Shinya Sumikura, Mikiya Shibuya, and Ken Sakurada. "OpenVSLAM". In: *Proceedings of the 27th ACM International Conference on Multimedia* (Oct. 2019). DOI: 10.1145/3343031.3350539.

Thank You !!!