

Applied Machine Vision

N-View Reconstruction

Matthew Dailey

Information and Communication Technologies
Asian Institute of Technology



Readings for these lecture notes:

- Hartley, R., and Zisserman, A. (2004), *Multiple View Geometry in Computer Vision*, Cambridge University Press, Chapters 18–19.
- Triggs, B., McLauchlan, P., Hartley, R., and Fitzgibbon, A. (1999), Bundle adjustment — A modern synthesis, *Vision Algorithms: Theory and Practice*, Springer-Verlag.
- Tomasi and Kanade, Sturm and Triggs, Pollefeys et al., Davison et al., Klein and Murray, Lourakis and Argyros, Kerl, Sturm, and Cremers, Engel SVO, LSD SLAM, Mur-Artal, Montiel, and Tardós.

These notes contain material © Hartley and Zisserman (2004) and others.

- 1 Introduction
- 2 Bundle adjustment
- 3 Factorization
- 4 Resectioning
- 5 Metric upgrade and auto-calibration
- 6 LSD SLAM
- 7 ORB SLAM
- 8 VI-ORB SLAM

Introduction

Reconstruction from N views

We have seen two-view projective and metric reconstruction techniques. As we move to many views, however, what can we do?

In this part we consider how to obtain a sparse reconstruction given a **sequence** of images.

This is where Hartley and Zisserman's book becomes a little obsolete, as there have been many new developments since 2004.

A brief tour of the history:

- 1 1970's: The term **bundle adjustment** emerges in the photogrammetry literature, referring to simultaneous optimization of parameters of a set of cameras and a set of points observed by those cameras.
- 2 1992: Tomasi and Kanade show how to use SVD to factor the observation matrix to estimate a sequence of cameras and collection of 3D points. Limited to affine cameras with no missing points.
- 3 1996: Sturm and Triggs show how to use iterative factorization to obtain a projective reconstruction. Other factorization methods refine the technique.
- 4 2004: Pollefeys et al. combine keyframe selection, SfM, BA, resectioning, loop closure, autocalibration, and 3D mesh techniques to obtain textured 3D models from videos obtained with hand-held cameras.

Most offline 3D reconstruction methods use a pipeline similar to Pollefeys' approach, with manual intervention.

More recently, SfM (computer vision) and SLAM (robotics) techniques are starting to converge.

- 1 2006: Davison et al. introduce the first real-time monocular SLAM method, called **MonoSLAM**.
- 2 2007: Klein and Murray introduce **PTAM** (Parallel Tracking and Mapping) aimed at augmented reality applications.
- 3 2009: Lourakis and Argyros introduce **SBA** (Sparse BA), an efficient open source bundle adjustment library, bringing fast BA to the masses.
- 4 2013: Kerl, Sturm, and Cremers introduce **DVO SLAM** (Dense Visual SLAM) for RGB-D cameras.

As compute power increases, we are seeing more incremental real time methods with excellent results.

- 1 2014: Engel, Schöps, and Cremers introduce **SVO**, a semi-direct method for monocular visual odometry.
- 2 2014: Engel, Stückler, and Cremers (2015) introduce LSD SLAM (Large-Scale Direct Monocular SLAM), the first dense monoSLAM method.
- 3 2015: Mur-Artal, Montiel, and Tardós introduce ORB-SLAM, the most robust feature-based monoSLAM method today, combining the basic approach of PTAM with ORB features, BA, and loop closure techniques.

LSD SLAM brings the idea of dense photometric alignment from RGBD to RGB, providing richer maps than ORB-SLAM at higher computational cost.

Today, work is continuing on improving the robustness and efficiency of visual SLAM systems.

Several methods now obtain real time results on smartphones or embedded platforms like the Odroid XU4 or NVidia TX2.

However, it is still very difficult for state-of-the-art methods to keep track of a set of features during rotations and fast relative motion.

Visual-inertial SLAM systems attempt to combine IMU readings (linear acceleration and rotational velocity) with vision.

Knowing approximately how the camera has moved since the last keyframe gives us a better idea of where to look for features in the next frame.

Some visual-inertial SLAM systems:

- Christian Forster's Ph.D. thesis (2016) demonstrates combination of SVO with IMU preintegration to achieve very accurate VISLAM. No open source implementation.
- 2016: Forster, Zhang, Gassner, Welberger, and Scaramuzza introduce **SVO-2**, a faster, more accurate version of SVO incorporating Forster's IMU priors. Implementation is commercial (no open source).
- Raúl Mur-Artal, and Juan D. Tardós (2017) introduce **VI-ORB**, a visual-inertial version of ORB-SLAM using Forster's IMU preintegration. The authors did not release an open source version, but there is a community developed version on github.
- Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart and Paul Timothy Furgale (2015) introduce **OKVIS**, "Keyframe-based visualinertial odometry using nonlinear optimization." Open-source version maintained by the author available on github.

Interesting project for this class: VI-ORB or OKVIS experiments on Pixhawk or Android...

Outline

- 1 Introduction
- 2 Bundle adjustment**
- 3 Factorization
- 4 Resectioning
- 5 Metric upgrade and auto-calibration
- 6 LSD SLAM
- 7 ORB SLAM
- 8 VI-ORB SLAM

Bundle adjustment

The idea

Given a set of unknown 3D points \mathbf{X}_j viewed by a set of cameras with unknown projection matrices P^i at image points \mathbf{x}_j^i , we seek to find the camera matrices P^i and 3D points \mathbf{X}_j minimizing the reprojection error

$$\min_{\hat{P}^i, \hat{\mathbf{X}}_j} \sum_{ij} d(\hat{P}^i \hat{\mathbf{X}}_j, \mathbf{x}_j^i)^2$$

Iterative minimization of this cost function is called **bundle adjustment** because it involves adjusting the bundle of rays between each camera center and the set of 3D points.

As formulated above, we need outlier removal prior to nonlinear least squares optimization. However, Triggs, McLauchlan, Hartley, and Fitzgibbon (1999) argue for a formulation with a more general cost function allowing robust estimation with the outliers included.

Bundle adjustment

Problems with bundle adjustment

There are two big **problems** with bundle adjustment:

- It needs a good **starting point** to begin optimization.
- There can be a **large number of parameters** involved in the minimization. For n points viewed by m cameras we have $3n + 11m$ parameters. This makes the matrices used by Levenberg-Marquardt prohibitively large.¹

¹Remember the linear system we have to solve on each iteration of LM?

$$\left[\mathbf{J}_f^T(\mathbf{P})\mathbf{J}_f(\mathbf{P}) + \mu\mathbf{I} \right] \delta\mathbf{P} = -\mathbf{J}_f^T(\mathbf{P})\mathbf{f}(\mathbf{P}).$$

Bundle adjustment

Solutions to the problems with bundle adjustment

Solutions of the first problem (the initial solution) generally involve linear methods such as **factorization**.

Solutions to the second problem (large parameter matrix):

- Reduce n and/or m , by using a subset of the points or partitioning the views [**suboptimal**].
- Interleave estimation of camera matrices with estimation of 3D points [guaranteed to converge but **slow**].
- Use a sparse minimization routine:
 - Download the sba (sparse bundle adjustment) open source software from <http://www.ics.forth.gr/~lourakis/sba/>
 - (For Debian) install the liblapack-dev package, and add `-llapack` to your gcc command line.
 - Call `sba_motstr_levmar()` to estimate motion (P_i 's) and structure (X).

Outline

- 1 Introduction
- 2 Bundle adjustment
- 3 Factorization**
- 4 Resectioning
- 5 Metric upgrade and auto-calibration
- 6 LSD SLAM
- 7 ORB SLAM
- 8 VI-ORB SLAM

Factorization

Affine factorization

Tomasi and Kanade's algorithm is a maximum likelihood reconstruction in the case of **affine** cameras.

It requires that **all points** be seen in **all views**.

For affine cameras we write $\mathbf{x} = (x, y)^T$ and $\mathbf{X} = (X, Y, Z)^T$. Then we have the projection equation

$$\mathbf{x} = \mathbf{M}\mathbf{X} + \mathbf{t}$$

We seek cameras $\{\mathbf{M}^i, \mathbf{t}^i\}$ and 3D points \mathbf{X}_j such that the distance between estimated and predicted image points is minimized:

$$\min_{\mathbf{M}^i, \mathbf{t}^i, \mathbf{X}_j} \sum_{ij} (\mathbf{x}_j^i - \hat{\mathbf{x}}_j^i)^2 = \min_{\mathbf{M}^i, \mathbf{t}^i, \mathbf{X}_j} \sum_{ij} (\mathbf{x}_j^i - (\mathbf{M}^i \mathbf{X}_j + \mathbf{t}^i))^2.$$

Factorization

Affine factorization

If we choose the centroid of the points to be the origin of the coordinate system, then we can estimate the **translation vectors** \mathbf{t}^i easily.

Under affine projection, the origin of the coordinate system is projected to $(0, 0)^T$ in the image. This means that \mathbf{t} needs to translate the projection of the origin to the mean of the observed image points:

$$\mathbf{t}^i = \frac{1}{n} \sum_j \mathbf{x}_j^i.$$

To simplify the remaining calculations, we set $\mathbf{t}^i = \mathbf{0}$ and adjust the points \mathbf{x}_j^i accordingly.

Factorization

Affine factorization

Now we arrange the adjusted image points in the $2m \times n$ **measurement matrix**

$$W = \begin{bmatrix} \mathbf{x}_1^1 & \mathbf{x}_2^1 & \cdots & \mathbf{x}_n^1 \\ \mathbf{x}_1^2 & \mathbf{x}_2^2 & \cdots & \mathbf{x}_n^2 \\ \cdots & \cdots & \ddots & \cdots \\ \mathbf{x}_1^m & \mathbf{x}_2^m & \cdots & \mathbf{x}_n^m \end{bmatrix}.$$

We want to find M^i and \mathbf{X}_j such that

$$W = \begin{bmatrix} M^1 \\ M^2 \\ \cdots \\ M^m \end{bmatrix} [\mathbf{X}_1 \quad \mathbf{X}_2 \quad \cdots \quad \mathbf{X}_n].$$

Factorization

Affine factorization

Since M and X are rank 3, their product is rank 3.

Since in general W will not be rank 3 due to measurement noise, we replace it with the best possible reconstruction, the matrix \hat{W} which is rank 3 and closest to W in Frobenius norm.

Such a matrix \hat{W} can be computed easily using the SVD $UDV^T = W$ by truncating U and V to three columns to get \hat{U} and \hat{V} , then truncating D to a 3×3 matrix \hat{D} , then finally letting $\hat{W} = \hat{U}\hat{D}\hat{V}^T$.

Since \hat{W} minimizes $\|W - \hat{W}\|_F$, $\hat{M} = \hat{U}$ and $\hat{X} = \hat{V}^T$ is a maximum likelihood reconstruction.

So we see that a straightforward application of the SVD gives us an **optimal reconstruction** in the case of **affine cameras**.

Factorization

Projective factorization

The affine factorization method doesn't work for **projective reconstruction**, so any serious projective distortion will introduce error into the reconstruction.

Sturm and Triggs (1996), however, pointed out that if we knew the **projective depth** of each point, then the structure and motion (camera matrices) could be estimated correctly by factorization.

We have $\mathbf{x}_j^i = P^i \mathbf{X}_j$. With a projective camera we have homogeneous points and there is an implicit constant factor which we can make explicit as $\lambda_j^i \mathbf{x}_j^i = P^i \mathbf{X}_j$ with $\mathbf{x}_j^i = (x_j^i, y_j^i, 1)^T$.

Factorization

Projective factorization

If we write the depths explicitly and all points are visible in all images, we can write the problem in terms of a **scaled measurement matrix**:

$$W = \begin{bmatrix} \lambda_1^1 \mathbf{x}_1^1 & \lambda_2^1 \mathbf{x}_2^1 & \cdots & \lambda_n^1 \mathbf{x}_n^1 \\ \lambda_1^2 \mathbf{x}_1^2 & \lambda_2^2 \mathbf{x}_2^2 & \cdots & \lambda_n^2 \mathbf{x}_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_1^m \mathbf{x}_1^m & \lambda_2^m \mathbf{x}_2^m & \cdots & \lambda_n^m \mathbf{x}_n^m \end{bmatrix} = \begin{bmatrix} P^1 \\ P^2 \\ \vdots \\ P^m \end{bmatrix} [\mathbf{X}_1 \quad \mathbf{X}_2 \quad \cdots \quad \mathbf{X}_n]$$

Since P and X are each rank 4, W will be rank 4 and the **rank 4 factorization** from the SVD will give a valid P and X .

Factorization

Projective factorization

The key to projective factorization is how to choose the **projective depths**² λ_j^i ?

- We could use another reconstruction method to estimate λ_j^i .
- We can also initialize them arbitrarily, e.g. $\lambda_j^i = 1$, then **interleave** factorization with depth estimation.

Though there is no guarantee that the iterative method will converge to a global minimum, it is widely used in practice.

²The λ_j^i are called projective depths because in a Euclidean frame they would be the lengths of the projections of the scene points onto the cameras' principal axes. ▶

Factorization

Projective factorization

The algorithm works well in practice, but it is important to know **what cost function** is it minimizing.

By using rank 4 decomposition, it turns out the algorithm is minimizing

$$\|W - \hat{W}\|_F^2 = \sum_{ij} \|\lambda_j^i \mathbf{x}_j^i - \hat{\lambda}_j^i \hat{\mathbf{x}}_j^i\|^2 = \sum_{ij} (\lambda_j^i x_j^i - \hat{\lambda}_j^i \hat{x}_j^i)^2 + (\lambda_j^i y_j^i - \hat{\lambda}_j^i \hat{y}_j^i)^2 + (\lambda_j^i - \hat{\lambda}_j^i)^2$$

When the λ_j^i are close to each other, we see that the cost function is an **approximation to a scaled geometric distance**.

Factorization

Projective factorization

Because the projective reconstruction cost function involves λ_j^i , and the projective factorization method minimizes geometric error when $\forall i, j, \lambda_j^i = 1$, we would like to keep the λ_j^i as close to 1 as possible.

If we scale P and X , we obtain

$$(\alpha^i \beta_j \lambda_j^i) \mathbf{x}_j^i = (\alpha^i P^i) (\beta_j \mathbf{X}_j)$$

which means we can replace the projective depths by multiplying the i th row or j th column of W by an arbitrary factor.

One normalization method producing good results in practice is to renormalize, on every iteration, the rows and columns of W so they have unit norm.

As always, the image points should be normalized by isotropic scaling before beginning.

Factorization

Projective factorization

Projective factorization: Objective

Given a set of n image points seen in m views:

$$\mathbf{x}_j^i; i = 1, \dots, m, \quad j = 1, \dots, n$$

compute a projective reconstruction.

Factorization

Projective factorization

Projective factorization: Algorithm

- (i) Normalize the image data using isotropic scaling.
- (ii) Set projective depths $\lambda_j^i = 1$ or use some other method to estimate them.
- (iii) Normalize the depths λ_j^i by multiplying rows and columns by constant factors. One way is to do a pass setting the norm of each row to 1 then a pass setting the norm of each column to 1.
- (iv) Form the $3m \times n$ scaled measurement matrix W and obtain P^i and X_j from the rank 4 factorization of W .
- (v) Reproject the points into each image to obtain new estimates of the depths and repeat from step (iii) until convergence.

Factorization

Factorization approaches to N -view reconstruction

Factorization methods for reconstruction have received a great deal of attention over the last 10 years.

See Tang and Hung (2006) for a method based on the projective factorization idea that **allows missing points** and is also **guaranteed to converge** to a local minimum.

Outline

- 1 Introduction
- 2 Bundle adjustment
- 3 Factorization
- 4 Resectioning**
- 5 Metric upgrade and auto-calibration
- 6 LSD SLAM
- 7 ORB SLAM
- 8 VI-ORB SLAM

Resectioning

Motivation

The factorization methods just outlined are **batch** methods.

They cannot be used to perform **on-line** 3D estimation without modification.

The **resectioning method** first gets an initial 3D reconstruction using, for example, the SVD of \mathbf{E} , then repeats the following steps for frame i :

- Get correspondences between already-estimated 3D points $\{\mathbf{X}_j\}_{j \in 1 \dots n}$ and new 2D points $\{\mathbf{x}_k\}_{k \in 1 \dots m}$.
- Use the correspondences to estimate P^i . This is called **resectioning**.
- Use triangulation to estimate new 3D points using correspondences between frame i and frame $i - 1$.

Resectioning method

Challenges in resectioning

The main challenges in resectioning are **accumulated error**, **outliers**, and **degenerate conditions**.

To reduce accumulated error, we can periodically use bundle adjustment to find a globally consistent, minimum error configuration of the cameras and points.

To mitigate the effect of outliers, we use RANSAC or other robust estimators and outlier rejection.

To avoid degenerate conditions, we apply **keyframe selection**, in which we choose the images we use for reconstruction and resectioning specifically to avoid degenerate configurations of the cameras and points.

This leads to the general algorithm, on the next page.

Resectioning method

The algorithm

Reconstruct from an image sequence: Algorithm II

- (i) Compute **interest points** in each image using, e.g., SIFT.
- (ii) Extract **keyframes** from the image sequence in which significant motion separates successive keyframes.
- (iii) Obtain 2-frame reconstruction from keyframes 1 and 2.
- (iv) Perform **resectioning** on remaining images.
- (v) **Bundle adjust** the cameras and 3D structure for the complete keyframe sequence to obtain a maximum likelihood projective reconstruction.

Resectioning method


Keyframe selection

Keyframe selection **improves the runtime performance** of 3D reconstruction, helps improve **accuracy**, and helps avoid **degeneracy**.

Ahmed, Dailey, Landabaso, and Herrero (2010)³ experimented with a variety of criteria for robust key frame extraction.

Given the first keyframe, we apply **correspondence ratio** and **degeneracy avoidance** constraints to eliminate inappropriate candidate keyframes.

Then we select the best successor keyframe from the remaining candidate keyframe set by maximizing an objective function.

³Ahmed, M.T., Dailey, M.N., Landabaso, J.L., and Herrero, N. (2010), Robust key frame extraction for 3D reconstruction from video streams. In *International Conference on Computer Vision Theory and Applications (VISAPP)*. 

Resectioning method

Keyframe selection

The **correspondence ratio** R_c is a proxy for the baseline:

$$R_c = \frac{T_c}{T_f},$$

where T_c is the number of inlier correspondences and T_f is total number of features found.

Low values of R_c indicate low overlap between two frames, in turn indicating long baselines.

Long baselines are desired for triangulation accuracy, but if the number of corresponding points is insufficient, camera motion estimation accuracy suffers.

Thus, the correspondence ratio is constrained to lie between an upper threshold T_1 and a lower threshold T_2 .

Resectioning method

Keyframe selection

To avoid degenerate cases, the **geometric robust information criterion (GRIC)** score is used to measure the goodness of fit for a homography or the fundamental matrix.

$$GRIC = \sum_i \rho(e_i^2) + \lambda_1 dn + \lambda_2 k,$$

where $i = 1 \dots n$, $\rho(e_i^2)$ is a robust function

$$\rho(e_i^2) = \min\left(\frac{e_i^2}{\sigma^2}, \lambda_3(r - d)\right)$$

of the residual e_i over the n correspondences, σ is the assumed standard deviation of the error, d is the model dimension, k is the model degrees of freedom, r is the dimension of the data, $\lambda_1 = \log(r)$, $\lambda_2 = \log(rn)$, and λ_3 limits the residual error.

The GRIC constraint is to **reject** any candidate keyframes for which the **homography** model has a **lower GRIC score** than the **fundamental matrix**.

Resectioning method

Keyframe selection

After filtering by correspondence ratio and degeneracy constraints, we select as the next keyframe the frame that maximizes certain selection criteria incorporating

- **Normalized GRIC difference:**

$$f_G(i, j) = \frac{GRIC_H(i, j) - GRIC_F(i, j)}{GRIC_H}.$$

The difference is maximized when the fundamental matrix model is much better than the homography model.

- **Point-to-epipolar line cost (PELC):**

$$f_{GP}(i, j) = w_G f_G(i, j) + w_P (\sigma - PELC(i, j)),$$

where i is the current keyframe and j is a candidate next keyframe. PELC tends to be high when correspondences are not accurate.

Resectioning method

Resectioning

The camera matrix P^i is computed from 3D-2D correspondences and the DLT algorithm.

In practice, 3D reconstruction using **long** video sequence suffers from **accumulated error**. Some issues are:

- **Matching existing 3D to 2D points** in an additional frame is not perfect. Sometimes, already-existing features are treated as new points. This arises when the camera moves back and forth or when a point becomes occluded and then visible again.
- Resectioning is highly sensitive to **outliers**. The camera estimate can be completely wrong if we estimate P^i in the presence of outliers.


Resectioning method

Example from AIT Vision and Graphics Lab

Atima Tharatipyakul worked on solving these problems by finding 2D-3D point pairs using multiple frames then using RANSAC in camera estimation to discard 2D-3D correspondences outliers.⁴

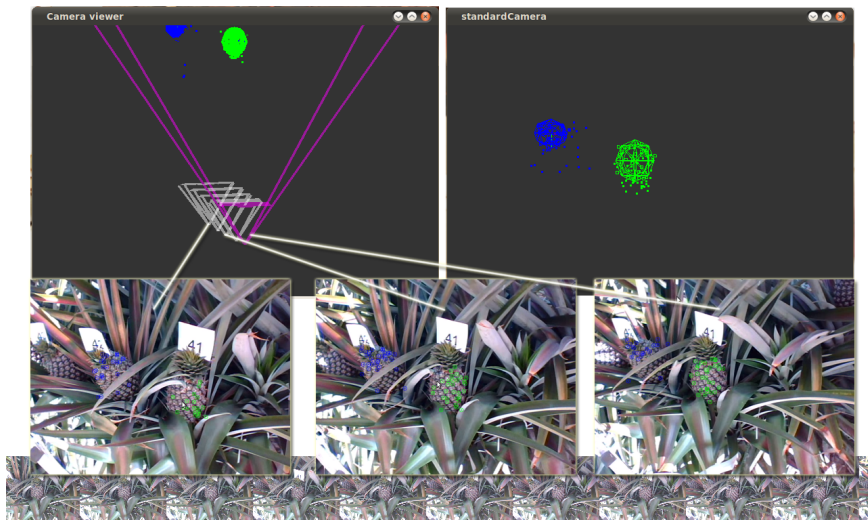
Next slide: example results from resectioning method in Atima's work, with 5 keyframes selected from a sequence of images containing pineapples.

3 of the keyframes are shown, along with camera estimates, point cloud estimates, and fruit ellipsoid estimates.

⁴Tharatipyakul, A., *3D Visualization from Video Sequence for Agricultural Field Inspection Robot*, Master's thesis, Asian Institute of Technology, 2011. 

Resectioning method

Example from AIT Vision and Graphics Lab



Outline

- 1 Introduction
- 2 Bundle adjustment
- 3 Factorization
- 4 Resectioning
- 5 Metric upgrade and auto-calibration**
- 6 LSD SLAM
- 7 ORB SLAM
- 8 VI-ORB SLAM

Metric upgrade and auto-calibration

Introduction

As we already know, if K is known we can directly obtain metric reconstructions. For two frames:

- For first pair of keyframes, estimate F .
- Calculate E from F and K .
- Factor E to obtain P' as described in H&Z Section 9.6.

For N frames, we factor E for the first pair of keyframes, then use incremental resectioning and bundle adjustment.

If K is **unknown** or **changing** during the image sequence, however, we need a method for **auto-calibration**.

Here we look at some of the techniques for auto-calibration from image correspondences over an image sequence.

Metric upgrade and auto-calibration

Idea of auto-calibration

The idea of auto-calibration:

- Obtain a projective factorization $W = PX$.
- Estimate a homography H such that $W = (PH)(H^{-1}X)$ is a metric reconstruction.
- $P^i H$ is metric if it can be decomposed as $K^i [R^i \mid \mathbf{t}^i]$ where the K^i are consistent with a priori constraints (the same across the image sequence, only focal length changing, etc.).

Metric upgrade and auto-calibration

Direct vs. stratified methods

There are two main approaches to auto-calibration for general motion:

- **Direct** estimation of H .
- **Stratified** reconstruction, beginning with an affine reconstruction (which identifies π_∞) followed by metric reconstruction.

Here we only consider direct methods, though stratified methods have some advantages, mainly that there is a linear solution for metric reconstruction once π_∞ is known (see text for details).

Metric upgrade and auto-calibration

Framework

We seek H such that $P_M^i = P^i H = K^i [R^i \mid \mathbf{t}^i]$ for $i = 1, \dots, m$.

Since we don't care about the absolute frame, we assume $P^1 = [I \mid \mathbf{0}]$ and that therefore $P_M^1 = K^1 [I \mid \mathbf{0}]$.

In general, H takes the form

$$H = \begin{bmatrix} A & \mathbf{t} \\ \mathbf{v}^T & k \end{bmatrix}.$$

Since $P_M^1 = P^1 H = [I \mid \mathbf{0}]$, we can infer that $A = K^1$ and $\mathbf{t} = \mathbf{0}$.

Since H is necessarily non-singular we can assume $k = 1$ to obtain

$$H = \begin{bmatrix} K^1 & \mathbf{0} \\ \mathbf{v}^T & 1 \end{bmatrix}.$$

Metric upgrade and auto-calibration

Framework

The plane at infinity π_∞ is $(0, 0, 0, 1)^T$ in an affine or metric frame, and H^{-T} is the point/plane transform from the metric frame to the projective frame. This means we can derive

$$\pi_\infty = \begin{pmatrix} -(K^1)^{-T} \mathbf{v} \\ 1 \end{pmatrix}.$$

Finally we can write

$$H = \begin{bmatrix} K & \mathbf{0} \\ -\mathbf{p}^T K & 1 \end{bmatrix}, \pi_\infty = (\mathbf{p}^T, 1)^T.$$

Metric upgrade and auto-calibration

Framework

For the rest of the cameras ($i = 2, \dots, m$), we write $P^i = [A^i \mid \mathbf{a}^i]$.

Using $P_M^i = P^i H$, we can obtain

$$K^i R^i = (A^i - \mathbf{a}^i \mathbf{p}^T) K^1$$

and (since R^i is orthogonal),

$$K^i K^{iT} = (A^i - \mathbf{a}^i \mathbf{p}^T) K^1 K^{1T} (A^i - \mathbf{a}^i \mathbf{p}^T)^T.$$

These are the **basic auto-calibration equations**. If we know K^1 , \mathbf{p} , and P^i , we can calculate P_M^i .

Metric upgrade and auto-calibration

The DIAC and the absolute dual quadric

The matrix $K^i K^{iT}$ is the **dual image of the absolute conic (DIAC)** ω^{*i} .

The DIAC is the projection of the **absolute dual quadric** Q_∞^* :

$$K^i K^{iT} = \omega^{*i} = P^i Q_\infty^* P^{iT}$$

If we know Q_∞^* , we can calculate K^i directly (by Cholesky decomposition of $P^i Q_\infty^* P^{iT}$).

Metric upgrade and auto-calibration

Framework

Important facts:

- The **absolute conic** Ω_∞ is a conic on π_∞ containing the intersection of all circles and spheres with π_∞ .
- The **absolute dual quadric** Q_∞^* is a rank 3 dual quadric whose envelope is the set of planes tangent to Ω_∞ .
- The absolute dual quadric is invariant under similarity transforms and is just $\text{diag}(1,1,1,0)$ in a metric frame.
- The absolute dual quadric's null space is π_∞ .

Metric upgrade and auto-calibration

Auto-calibration based on Q_{∞}^*

Auto-calibration based on Q_{∞}^* : Objective

Given a set of matched points across several views and constraints on the calibration matrices K^i , compute a metric reconstruction of the points and cameras.

Metric upgrade and auto-calibration

Auto-calibration based on Q_{∞}^*

Auto-calibration based on Q_{∞}^* : Algorithm

- (i) Compute a projective reconstruction from a set of views, resulting in cameras P^i and points X .
- (ii) Use $\omega^{*i} = P^i Q_{\infty}^* P^{iT}$ along with constraints to estimate Q_{∞}^* .
- (iii) Decompose Q_{∞}^* as $H\tilde{I}H^T$ where $\tilde{I} = \text{diag}(1,1,1,0)$.
- (iv) Apply H^{-1} to X and H to P^i to obtain a metric reconstruction of the point and cameras.
- (v) Use iterative least squares to improve the solution.

Metric upgrade and auto-calibration

Auto-calibration based on Q_{∞}^*

As a nice example of linear constraints on Q_{∞}^* , see Pollefeys et al. (2004), Visual modeling with a hand-held camera, *IJCV* 59(3).

As a nice example of more sophisticated methods to enforce the rank 3, positive semidefiniteness, and “chirality” constraints on Q_{∞} , see Chandraker et al. (2007), Autocalibration via rank-constrained estimation of the absolute dual quadric, In *Proceedings of CVPR*.

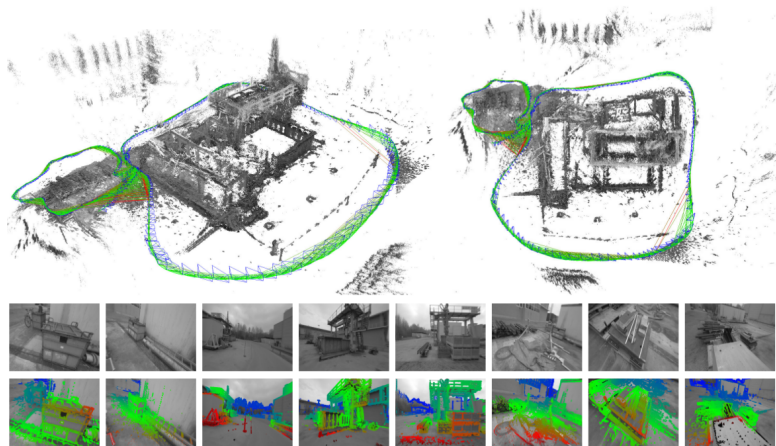
Outline

- 1 Introduction
- 2 Bundle adjustment
- 3 Factorization
- 4 Resectioning
- 5 Metric upgrade and auto-calibration
- 6 LSD SLAM**
- 7 ORB SLAM
- 8 VI-ORB SLAM

LSD SLAM

Introduction

Engel, Stückler, and Cremers' (2015) method **LSD SLAM** (Large-Scale Direct) is among the best of the new monocular SLAM methods.



Sample LSD-SLAM results (Engel, Schöps, and Cremers, 2014)

LSD-SLAM principles:

- Attempt to use all information in the image (not just features).
- Extract a series of “keyframes” from the video stream and find relationship between each pair of camera poses using graph optimization.
- Pairwise optimization of the pose graph minimizes **photometric error** between the two keyframes.
- Keyframes are paired with **inverse depth maps** that maintain a mean and variance (inverse confidence) in the inverse depth at each pixel.

Most SfM methods are **feature based**.

They first estimate camera positions and 3D point positions using sparse feature extraction, factorization of E , resectioning, and bundle adjustment.

A dense model can then be made through mesh construction and texture mapping.

Direct methods scrunch the 2-step process, finding the camera poses and depth maps that align images directly without feature extraction.

LSD SLAM

Notation and representation of poses

Camera must be **calibrated**. All points are converted to normalized camera coordinates (undistorting and multiplying by K^{-1}).

Camera poses, which are normally represented by a rotation matrix and translation vector packed into a 4×4 homography matrix, are instead represented by elements of the Lie algebra $\mathfrak{se}(3)$, written as **6-element vectors**.

Similarity transformations, normally represented by a scalar scale factor, a rotation matrix, and a translation vector, are instead represented by elements of $\mathfrak{sim}(3)$, written as **7-element vectors**.

Images are aligned by minimization of **photometric error**

$$E(\xi) = \sum_i (I_{\text{ref}}(\mathbf{p}_i) - I(\omega(\mathbf{p}_i, D_{\text{ref}}(\mathbf{p}_i), \xi)))^2$$

where

- ξ is a possible transformation between the cameras imaging I_{ref} and I ,
- $\omega(\mathbf{p}, d, \xi)$ projects point \mathbf{p} with inverse depth d in the reference camera's frame into the transformed camera frame,
- $D_{\text{ref}}(\mathbf{p})$ is the inverse depth of 2D point \mathbf{p} .

The parameters of ξ can be estimated using an initial guess and Gauss-Newton minimization.

Occlusions, reflections, and moving objects would introduce outliers.

Rather than remove outliers explicitly, we re-weight each point \mathbf{p}_i on each iteration, down-weighting large residual errors:

$$E(\boldsymbol{\xi}) = \sum_i w_i(\boldsymbol{\xi}) (l_{\text{ref}}(\mathbf{p}_i - l(\omega(\mathbf{p}_i, D_{\text{ref}}(\mathbf{p}_i), \boldsymbol{\xi})))^2.$$

Defining $r_i(\boldsymbol{\xi})$ to be the residual $l_{\text{ref}}(\mathbf{p}_i - l(\omega(\mathbf{p}_i, D_{\text{ref}}(\mathbf{p}_i), \boldsymbol{\xi})))$, the Gauss-Newton update becomes

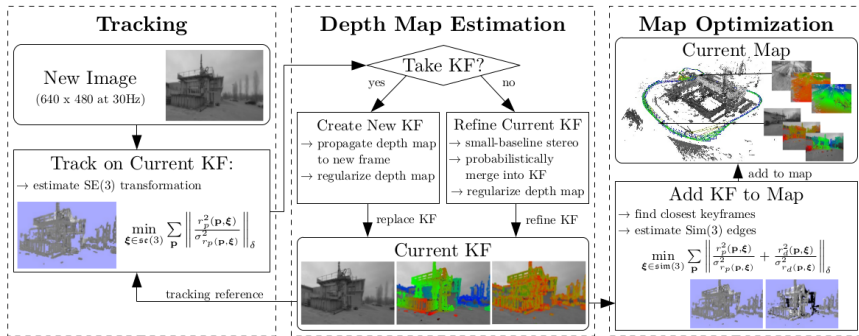
$$\delta \boldsymbol{\xi}^{(n)} = -(\mathbf{J}^T \mathbf{W} \mathbf{J})^{-1} \mathbf{J}^T \mathbf{W} \mathbf{r}(\boldsymbol{\xi}^{(n)}),$$

where \mathbf{J} is the Jacobian of the residuals r_i with respect to the changes in $\boldsymbol{\xi}$.

LSD SLAM

Overview

Overview of the processing pipeline:



Overall LSD-SLAM flow (Engel, Schöps, and Cremers, 2014)

We begin with an initial keyframe with a **random** depth map and a large variance.

Early translations of the camera enable convergence to a nearly correct depth map.

The map is a **pose graph** of keyframes.

Keyframe \mathcal{K}_i :

- Image $I_i : \Omega_i \rightarrow \mathbb{R}$
- Inverse depth map $D_i : \Omega_{D_i} \rightarrow \mathbb{R}^+$
- Variance of the inverse depth $V_i : \Omega_{D_i} \rightarrow \mathbb{R}^+$

The depth map and variance are only defined for a subset of all pixels $\Omega_{D_i} \subset \Omega_i$ that have sufficient intensity gradient.

Edges ξ_{ji} :

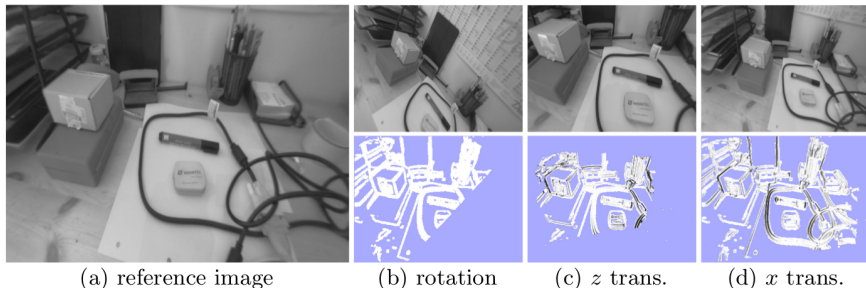
- Similarity transform $\xi_{ji} \in \text{sim}(3)$
- Covariance Σ_{ji} over ξ_{ji}

LSD SLAM

Depth estimation

Initial D starts out undefined for points with small gradient (purple color) and random with high variance for points with sufficient gradient.

Camera rotation gives no additional depth information; translation increases confidence/decreases variance for points with parallax under the translation.



Inverse depth map over camera transformations (Engel, Schöps, and Cremers, 2014).

From each keyframe, we perform short-term tracking, finding the relative pose $\xi_{ji} \in \mathfrak{se}(3)$ minimizing the **robust normalized photometric residual**

$$E_p(\xi_{ji}) = \sum_{\mathbf{p} \in \Omega_{D_i}} \left\| \frac{r_p^2(\mathbf{p}, \xi_{ji})}{\sigma_{r_p(\mathbf{p}, \xi_{ji})}^2} \right\|_{\delta}$$

where $\|\cdot\|_{\delta}$ is a robust norm and $\sigma_{r_p(\mathbf{p}, \xi_{ji})}^2$ is an estimate of residual uncertainty based on depth map uncertainty $V_i(\mathbf{p})$ and assumed constant image intensity noise σ_I .

A new keyframe is created from the most recent tracked image when the local motion ξ_{ji} exceeds a threshold.

Since actual depths are unknown, the depth map for each keyframe is scaled so that the mean inverse depth is 1.

The inverse depth map for the current keyframe is updated using the motion for each new tracking frame.

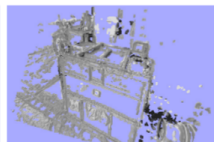
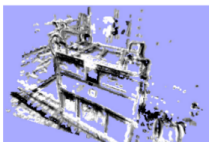
Each new keyframe is aligned with the previous keyframe to minimize both photometric residual and depth residual using image alignment over $\text{sim}(3)$ — we explicitly calculate the **relative depth** between keyframes.

LSD SLAM

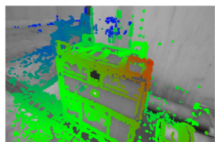
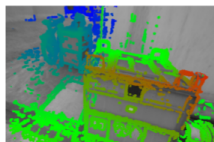
Keyframe alignment



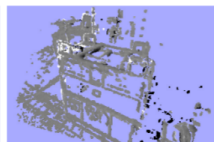
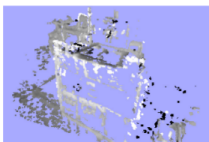
(a) camera images I



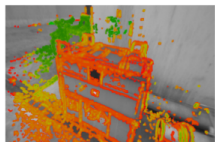
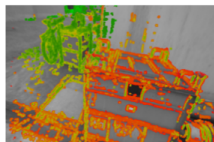
(d) normalized photometric residual r_p/σ_{r_p}



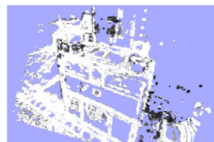
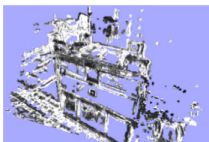
(b) estimated inverse depth maps D



(e) normalized depth residual r_d/σ_{r_d}



(c) inverse depth variance V



(f) robust Huber weights

Direct keyframe alignment on $\text{sim}(3)$ (Engel, Schöps, and Cremers, 2014)

Each time a new keyframe is selected, we compare with the nearest 10 keyframes looking for a **loop closure**: a previous keyframe close enough to the current frame to create a new edge in the pose graph.

Drift in a large loop might mean the new keyframe is too different from the best old keyframe for convergence.

There are a few tricks that help in convergence, but one of the best is a **course-to-fine** approach starting with downscaled 20×15 images.

Pose graph optimization runs continuously in the background.

LSD SLAM

Loop closure

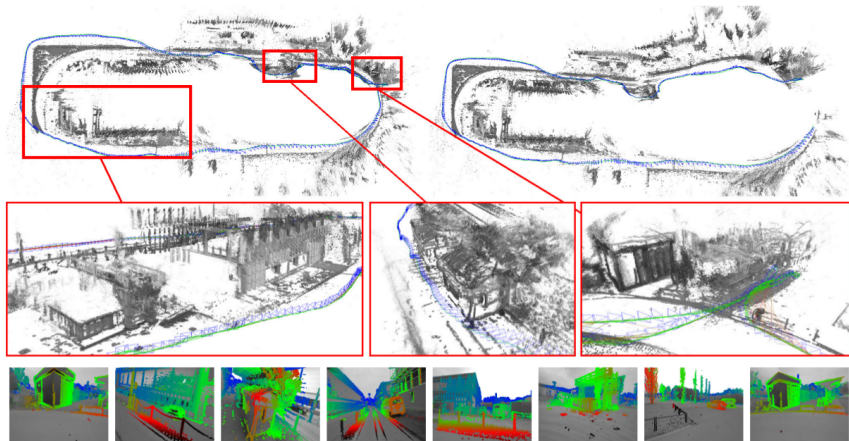


Fig. 7: Loop closure for a long and challenging outdoor trajectory (after the loop closure on the left, before on the right). Also shown are three selected close-ups of the generated pointcloud, and semi-dense depth maps for selected keyframes.

LSD-SLAM runs in real time on smartphones in odometry-only configurations (no large-scale map).

The full LSD-SLAM runs in real time on moderate CPUs (not requiring a GPU).

ORB-SLAM runs faster than LSD-SLAM and gives more accurate trajectories. But the resulting point clouds are sparse.

The near-term open problem is fast, accurate, dense 3D modeling from monocular cameras.

Outline

- 1 Introduction
- 2 Bundle adjustment
- 3 Factorization
- 4 Resectioning
- 5 Metric upgrade and auto-calibration
- 6 LSD SLAM
- 7 ORB SLAM**
- 8 VI-ORB SLAM

- ORB-SLAM is a visual SLAM method
- In 2016, ORB-SLAM was introduced as one of the most versatile and accurate monocular SLAM methods to date.
- ORB-SLAM is based on the main ideas of
 - Parallel tracking and mapping (PTAM) by Klein and Murray
 - Place recognition by Gálvez-López and Tardós
 - Scale-aware loop closing by Strasdat et. al
 - Covisibility information by Strasdat et. al

ORB-SLAM's aims:

- More efficient, simple and reliable system than existing visual SLAM methods.
- Real-time operation in large-environments.
- Real-time loop closing based on optimization.
- Real-time relocalization with significant invariance to viewpoint and illumination.
- Improving tracking robustness and enhance lifelong operation.

- Similar to other visual SLAM methods, ORB-SLAM needs a feature detector and descriptor to extract and match feature points from sequence of images.
- ORB-SLAM needs a feature detector and descriptor that can be used in mapping, tracking, and place recognition processes at real time.
- ORB detector and descriptor are fast enough to compute and match features while having good invariance to viewpoint.
- Therefore, ORB detector and descriptor are chosen for ORB-SLAM method.

- ORB-SLAM consists of three threads working in parallel.
 - Tracking thread
 - Mapping thread
 - Loop closing thread
- Bag of words place recognition is also an essential feature.
- The feature creates visual vocabulary from the keyframe it has been seen and stores in the database. If there are changes done to keyframes, the database is updated according to the changes.
- DBoW2 module is integrated to ORB-SLAM method for loop detection, relocalization, and keyframe culling.

ORB-SLAM

Introduction

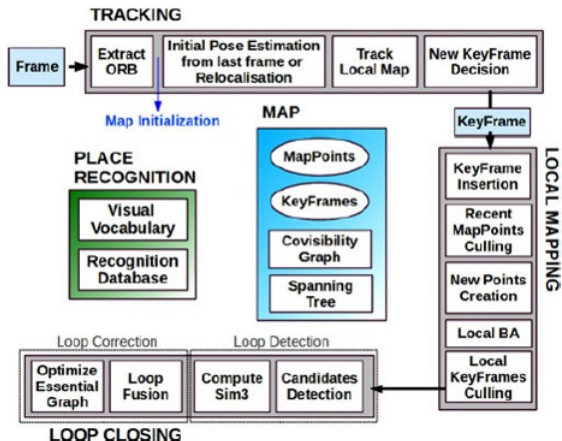


Figure: ORB-SLAM system overview

- Covisibility graph is an undirected weighted graph. Each node is a keyframe and an edge between two keyframes exists if they share observations of the same map points.
- Covisibility graph is too dense and error prone for loop closing operation. An idea of essential graph is proposed in ORB-SLAM method.
- Essential graph is a subset of covisibility graph that retains all nodes but less edges, still preserving a strong network that yields accurate results.
- Essential graph contains spanning trees, subsets of edges from the covisibility graph with high covisibility, and the loop closure edges, resulting in a strong network of cameras.

ORB-SLAM

Introduction

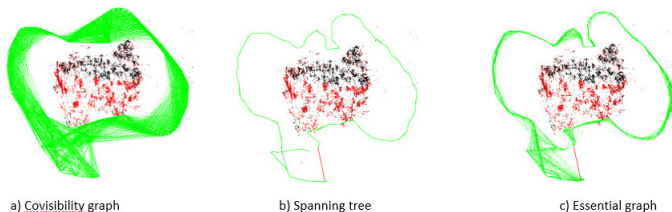


Figure: Graphs in ORB-SLAM method. a) A covisibility graph. b) A spanning tree. c) An essential graph.

- Tracking thread is responsible for the following tasks
 - Tracking
 - Map initialization
 - Relocalization when tracking is lost
 - Track local map
 - New keyframe decision

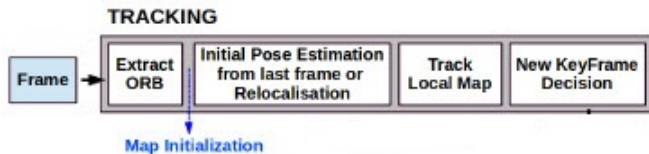


Figure: Tracking thread overview

- Tracking thread is a process that track a set of points through successive camera frames, and using these tracks to triangulate their 3D position.
- Camera pose of each frame is estimated from the relative pose of feature points between the current frame and previous frame.
- In ORB-SLAM, tracking thread extracts and uses ORB features from every images in the video sequence.

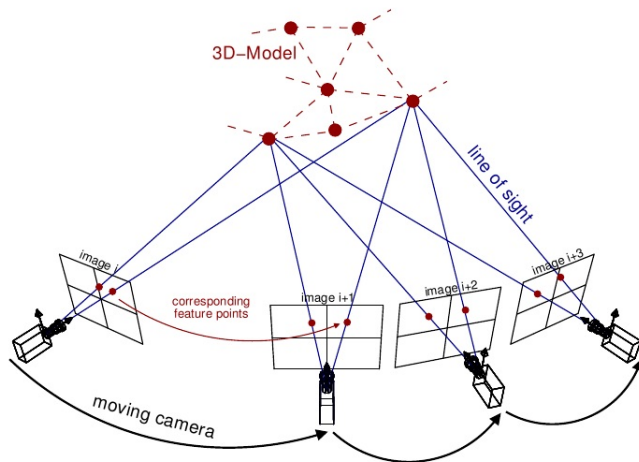


Figure: Point triangulation. from <http://www.theia-sfm.org/sfm.html>

- Map initialization is the first task of the tracking thread. It triangulates initial 3D correspondences with acceptable accuracy from initial image sequences.
- Matched features $x_c \leftrightarrow x_r$ between the current frame f_c and the reference frame f_r are searched to generate initial correspondences.
- Reset reference frame if not enough matches are found.

- Homography matrix H_{cr} and fundamental matrix F_{cr} are calculated. Tracking thread uses one of these matrices in map initialization process.
- Homography matrix is used if the scene is planar. Fundamental matrix is selected if the scene is nonplanar.
- The map is initialized if the selected matrix passes motion hypotheses tests and gives low reprojection error.
- Bundle adjustment is performed to refine initial correspondences and camera poses.

ORB-SLAM

Tracking thread: Map initialization

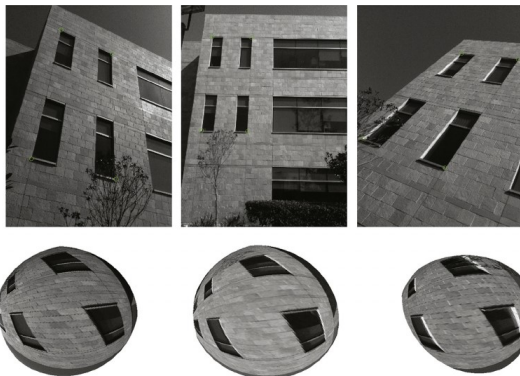


Figure: Planar scene. from https://www.researchgate.net/figure/259128816_fig2_Fig-7-Digital-images-of-a-planar-scene-Top-prior-to-projective-frame-registration

ORB-SLAM

Tracking thread: Map initialization

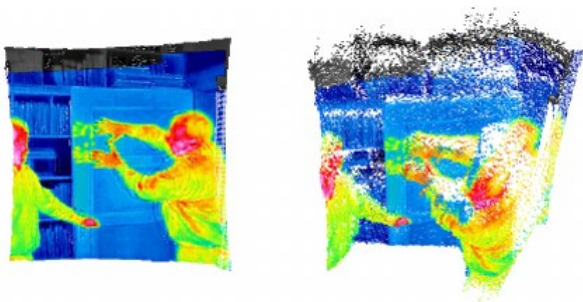


Figure: Non planar scene. from https://www.researchgate.net/figure/243459165_fig15_Figure-15-Frontal-view-onto-a-non-planar-scene-left-and-a-view-clearly-showing-the

- Tracking continues the process from map initialization.
- If tracking was successful for last frame, camera pose in current frame is predicted with constant velocity motion model.
- If not enough matches were found, tracking thread performs wider search of the map points around their position in the last frame.
- The pose is optimized with the found correspondences.

- If tracking is lost, the frame is converted into a bag of words and queried against a recognition database for keyframe candidates for global relocalization.
- Correspondences with ORB associated to map points in each keyframe are computed. RANSAC iterations are performed for each keyframe to find camera pose with *PnP* algorithm.
- If camera pose is found with enough inliers, guided search is performed to find more matches.
- The pose is then optimized and tracking procedure continues.

- Once estimated camera pose and an initial set of feature matches are obtained, the map into is projected to the frame and search more map point correspondences.
- The local map contains the set of keyframes K_1 , that share map points with the current frame, and a set K_2 with neighbors to the keyframes K_1 in the covisibility graph.
- Map points seen in K_1 and K_2 are searched in the current frame. If found, these map points are added into the current frame and optimized.

ORB-SLAM

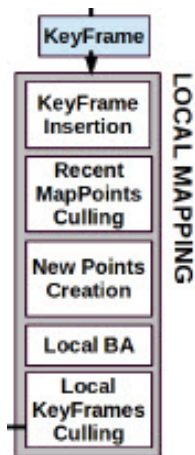
Tracking thread: New keyframe decision

- The current frame is verified as a keyframe if it meets the conditions
- The keyframe decided by the tracking thread is sent to the mapping thread to add into the map.
- Keyframes are inserted as fast as possible as it makes the tracking more robust to change in camera movements and rotations.
- Redundant keyframes will be culled in the mapping thread.

ORB-SLAM

Mapping thread

- Mapping thread is responsible for the following tasks
 - Keyframe insertion
 - Map point culling
 - New map point creation
 - Local bundle adjustment
 - Local keyframe culling



ORB-SLAM

Mapping thread: Keyframe insertion

- New keyframes decided by the tracking thread are added into the map.
- Nodes and edges in the covisibility graph are updated.
- Spanning tree and bag of word representation are updated.

- Map points must pass a restrictive test to ensure that they are trackable and not mistakenly triangulated.
 - The tracking must find the point in more than the 25 percent of the frames in which it is predicted to be visible.
 - If more than one keyframe has passed from map point creation, it must be observed from at least three keyframes.

- For each unmatched ORB in an individual frame, unmatched correspondences are searched with other unmatched point in other keyframes.
- Newly matched map points are accepted as the new points if they pass the following tests
 - Positive depth test in both cameras.
 - Parallax test.
 - Reprojection error test.
 - Scale consistency test.

ORB-SLAM

Mapping thread: Local bundle adjustment

- Local BA optimizes the currently processed keyframe (K_i), all keyframes connected to K_i in the covisibility graph (K_c), and all map points seen by K_i and K_c .
- Keyframes that see map points as K_i and K_c see are included in the optimization but remain fixed.
- Outlier are discarded in this process.

- As the complexity of bundle adjustment grows with the number of keyframes, deleting redundant keyframes benefits the system in lifelong operation.
- Keyframes that has 90 percent similar map points to its three previous keyframes are removed from the map.

- Loop closing takes the last keyframe processed by the mapping thread to detect and close the loop.
- Loop closing performs the following tasks
 - Loop candidates detection.
 - Similarity transformation estimation.
 - Loop fusion.
 - Essential graph optimization.

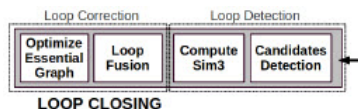


Figure: Loop closing overview.

ORB-SLAM

Loop closing thread: Loop candidates detection

- Similarity score is computed between the bag of word vector of the last keyframe K_i with all of its neighbors in the covisibility graph. The lowest score s_{min} is obtained.
- Recognition database is queried, keyframe whose score is lower than s_{min} are not considered as loop candidates.
- All keyframes whose are directly connected to K_i are also not considered as loop candidates.

- ORB-associated correspondences between the current keyframe and the loop candidate keyframes are calculated.
- RANSAC iterations are performed with each candidate to find a similarity transformation.
- If the similarity is found with enough inliers, the transformation is optimized and more correspondences are searched.
- After adding more correspondences, the transformation is optimized again. If the transformation is supported by enough inliers, that loop candidate is accepted.

- Once loop closing is detected and verified, two keyframes are merged together.
- The correction for this action must be done and propagated to their neighbor keyframes in covisibility graph so they can update their properties(i.e. recalculate transformation matrix, concatenated edges in covisibility graph).

ORB-SLAM

Loop closing thread: Essential graph optimization

- Pose graph optimization over the Essential graph is performed to effectively close the loop.
- Scale drift is corrected, each map point is transformed according to the correction.

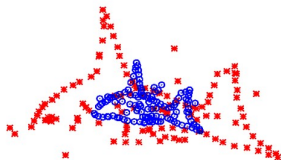


Figure: Scale drift.

ORB-SLAM

Example



Figure: ORB-SLAM example: tracking.

ORB-SLAM

Example

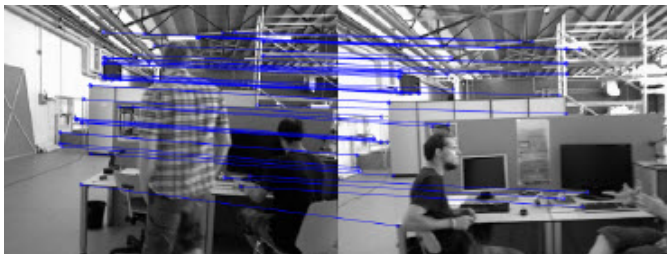


Figure: ORB-SLAM example: mapping.

ORB-SLAM

Example

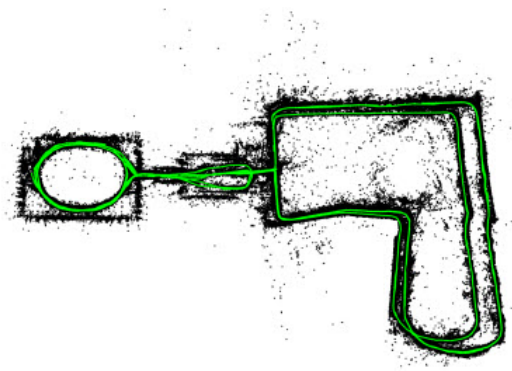


Figure: ORB-SLAM example: loop closing.

Outline

- 1 Introduction
- 2 Bundle adjustment
- 3 Factorization
- 4 Resectioning
- 5 Metric upgrade and auto-calibration
- 6 LSD SLAM
- 7 ORB SLAM
- 8 VI-ORB SLAM**

In 2017, the authors of ORB-SLAM (Raúl Mur-Artal and Juan D. Tardós) introduced “Visual-Inertial Monocular SLAM with Map Reuse.”

We know that all monocular SLAM methods have the limitation of **scale ambiguity**.

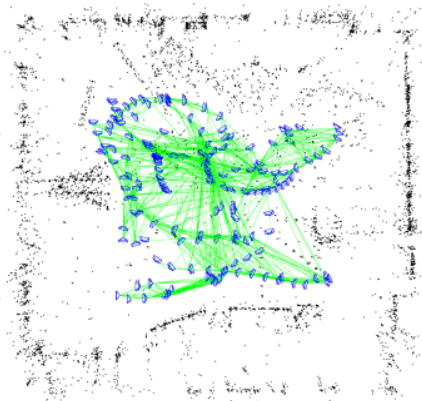
By adding information from GPS or an IMU, we can resolve that ambiguity.

Existing work:

- IMU preintegration by Lupton and Sukkarieh (2012)
- IMU preintegration mapped to $SO(3)$ by Forster et al. (2016)
- Factor graph representation by Indelman et al. (2013)
- ORB SLAM (2016)

VI-ORB SLAM

Overview



Mur-Artal and Tardós (2017), Fig. 1

ORB SLAM performs tracking for the current frame using a fixed map.

On the back end, local Bundle Adjustment (BA) optimizes a local window of keyframes.

Large loops are detected using place recognition then corrected using lightweight pose graph optimization followed by full BA.

Sample result on left.

VI-ORB SLAM

Avoiding biased partial solutions

Each step (tracking and local BA) fixes some states in their optimization.

This could bias the final solution, so it is important to get a good start.

To reduce any such bias, the authors implement a VI full BA that optimizes structure, camera poses, scale, velocities, gravity, and gyroscope and accelerometer biases.

But BA always needs an initial guess. The authors propose a piece-by-piece approach to obtain this solution (next slide).

Idea of initialization:

- Process a few seconds of video with regular ORB SLAM. This gives an initial solution for structure and some keyframe poses up to unknown scale.
- Compute bias of the gyroscope based on known orientation of the keyframes.
- Solve for scale and gravity without accelerometer bias.
- Using knowledge of the magnitude of gravity, solve for accelerometer bias, refining scale and gravity direction.
- Extract velocity for each keyframe.

The main requirement is that the sensor should be moved in such a way as to make all variables observable.

The VI odometry method is based on a few key aspects.

We assume a **pinhole camera**. Keypoints are extracted on original image, but coordinates are undistorted before being used.

There is an IMU that works as follows:

- Acceleration \mathbf{a}_b^k at time k in the IMU frame b .
- Angular velocity $\boldsymbol{\omega}_b^k$ at time k in the IMU frame b .
- Update frequency is on the order of 100 Hz.
- Accelerometer modeled as having slowly varying biases \mathbf{b}_a^k and \mathbf{b}_g^k for the accelerometer and gyroscope.
- Accelerometer is subject to gravity \mathbf{g}_w , which must be subtracted in order to compute motion.

Goal of the system: estimate state parameters and 3D points over time.

- \mathbf{R}_{wb}^k is the rotation of the IMU in the world frame at time k .
- ${}_w\mathbf{v}_b^k$ is the velocity in the world frame at time k .
- ${}_w\mathbf{p}_b^k$ is the position of the IMU in the world frame at time k .

Model is as follows:

$$\mathbf{R}_{wb}^{k+1} = \mathbf{R}_{wb}^k \exp\left(\left(\boldsymbol{\omega}_b^k - \mathbf{b}_g^k\right)\Delta_t\right)$$

$${}_w\mathbf{v}_b^{k+1} = {}_w\mathbf{v}_b^k + \mathbf{g}_w\Delta_t + \mathbf{R}_{wb}^k(\mathbf{a}_b^k - \mathbf{b}_a^k)\Delta_t$$

$${}_w\mathbf{p}_b^{k+1} = {}_w\mathbf{p}_b^k + {}_w\mathbf{v}_b^k\Delta_t + \frac{1}{2}\mathbf{g}_w\Delta_t^2 + \frac{1}{2}\mathbf{R}_{wb}^k(\mathbf{a}_b^k - \mathbf{b}_a^k)\Delta_t^2$$

$\exp(\cdot)$ is “exponential map” of rotations, a function mapping a twist vector $\mathbf{v} \in \mathbb{R}^3$ (denoting a rotation of $\|\mathbf{v}\|$ about the unit vector $\mathbf{v}/\|\mathbf{v}\|$) to a rotation matrix.

IMU measurements between two keyframes are preintegrated based on the work of Forster et al. (2016):

$$\mathbf{R}_{wb}^{i+1} = \mathbf{R}_{wb}^i \Delta_{R_{i,i+1}} \exp(\mathbf{J}_{\Delta R}^g \mathbf{b}_g^i)$$

$${}^w \mathbf{v}_b^{i+1} = {}^w \mathbf{v}_b^i + \mathbf{g}_w \Delta_{t_{i,i+1}} + \mathbf{R}_{wb}^i (\Delta_{\mathbf{v}_{i,i+1}} + \mathbf{J}_{\Delta \mathbf{v}}^g \mathbf{b}_g^i + \mathbf{J}_{\Delta \mathbf{v}}^a \mathbf{b}_a^i)$$

$${}^w \mathbf{p}_b^{i+1} = {}^w \mathbf{p}_b^i + \mathbf{v}_b^i \Delta_{t_{i,i+1}} + \frac{1}{2} \mathbf{g}_w \Delta_{t_{i,i+1}}^2 + \mathbf{R}_{wb}^i \left(\Delta_{\mathbf{p}_{i,i+1}} + \mathbf{J}_{\Delta \mathbf{p}}^g \mathbf{b}_g^i + \mathbf{J}_{\Delta \mathbf{p}}^a \mathbf{b}_a^i \right)$$

The idea is to separate the effect of the bias from the pre-integrated IMU measurements with a linear approximation.

The values $\Delta_{\mathbf{R}_{i,i+1}}$, $\Delta_{\mathbf{v}_{i,i+1}}$, and $\Delta_{\mathbf{p}_{i,i+1}}$ are the simple integration/concatenation of the transformations given by the IMU over the period between keyframe i and $i + 1$.

The preintegrations and Jacobians are computed iteratively as the IMU measurements arrive.

Finally, the camera and IMU are considered rigidly attached with transformation

$$\mathbf{T}_{cb} = \left[\mathbf{R}_{cb} \mid {}_c\mathbf{p}_b \right]$$

between their frames.

The transformation is estimated from the calibration method of Furgale et al. (2013) implemented in the open-source **Kalibr** tool.

VI-ORB SLAM

Modifying ORB-SLAM for VI odometry

As before, ORB-SLAM has 3 parallel threads: tracking, local mapping, and loop closing.

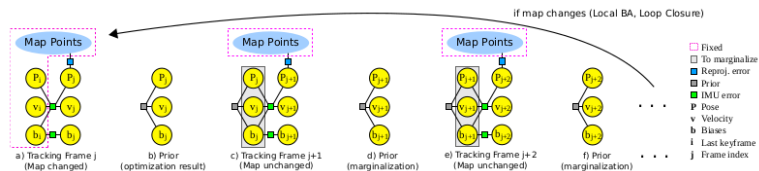
For the **tracking thread**:

- At each new frame, camera pose is predicted based on the IMU and the estimated pose from the previous frame.
- Visible keypoints are projected to the new frame then matched.
- Then the new frame's pose is optimized considering the matches and the IMU error.

VI-ORB SLAM

Modifying ORB-SLAM for VI odometry

Main idea for tracking thread:



Mur-Artal and Tardós (2017), Fig. 2.

As long as we are tracking successfully and the local mapping thread does not update 3D point locations, we continue to incrementally update the pose relative to the previous keyframe.

When the mapping thread updates the keyframe pose, we reset the current frame's pose relative to the keyframe that changed, then resume incremental updates.

[More on tracking thread]

The frame-changed optimization uses the parameter vector

$$\theta = \left\{ R_{wb,w}^j, \mathbf{p}_{b,w}^j, \mathbf{v}_b^j, \mathbf{b}_g^j, \mathbf{b}_a^j \right\}$$

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \left(\sum_k E_{\text{proj}}(k, j) + E_{\text{IMU}}(i, j) \right)$$

This is the sum of the reprojection error for all visible 3D points k and the IMU error from keyframe i to new frame j .

The frame-unchanged optimization uses a joint optimization of parameters for frame j and frame $j + 1$.

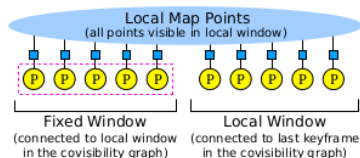
VI-ORB SLAM

Modifying ORB-SLAM for VI odometry

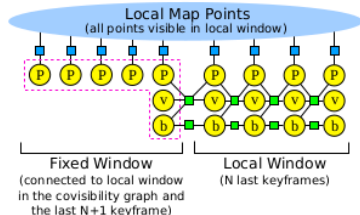
The **local mapping thread** performs a local BA every time a new keyframe is inserted.

An N keyframe window is optimized with all of the points observed over those keyframes (see right).

ORB-SLAM merely optimizes the poses and 3D points with reprojection error as the objective, whereas VI-ORB SLAM additionally adds IMU error terms for velocity and biases.



ORB-SLAM's Local BA



Visual-Inertial ORB-SLAM's Local BA

Mur-Artal and Tardós (2017), Fig. 3

VI-ORB SLAM

Modifying ORB-SLAM for VI odometry

The **loop closing thread** tries to reduce drift when returning to an already-mapped area.

The place recognition module matches recent keyframes with past keyframes.

If a rigid-body transformation can be found, then an optimization is carried out over the whole trajectory.

First is a pose-graph optimization that ignores IMU data, followed by a full BA that optimizes all velocities, biases, positions, and so on.

VI-ORB SLAM

Conclusion

Result: **accurate**, **real time**, **sparse** solution to the SLAM problem.

We learn that a monocular camera with IMU (and some sophisticated processing software) is sufficient for accurate localization and sparse point cloud mapping in real time.

No need for lasers, ultrasonic sensors, etc.!

See <https://www.youtube.com/watch?v=rdR50R8egGI> and other examples.

See <https://github.com/jingpang/LearnVIOORB> for community implementation based on the authors' ORB-SLAM 2 implementation.

Interesting project: get VI-ORB SLAM running on Android, using the smartphone IMU!

Another: compare VI-ORB-SLAM with OKVIS and SVO-2 (<https://www.youtube.com/watch?v=hR8uq1RTUfA&t=33s>).