Vision Algorithms for Mobile Robotics

Lecture 10
Multiple View Geometry 4

Davide Scaramuzza
http://rpg.ifi.uzh.ch
Lab Exercise – This afternoon

Intermediate VO integration for mini projects:
• problem statement
• details about what can/needs to be done
• we will show some of best examples from last years
• we will go through FAQ such as what can be added to get up to +0.5 mark

Image sequence → Feature detection → Feature matching (tracking) → Motion estimation → Local optimization
Outline

• Bundle Adjustment
• SFM with $n$ views
2-View Bundle Adjustment (BA)

- Non-linear, joint optimization of structure, $P^i$, and motion $R, T$
- Commonly used after least square estimation of $R$ and $T$ (e.g., after 8- or 5-point algorithm)
- Optimizes $P^i, R, T$ by minimizing the **Sum of Squared Reprojection Errors**:

$$P^i, R, T = \arg\min_{P^i, R, T} \sum_{i=1}^{N} \|p_1^i - \pi(P^i, K_1, I, 0)\|^2 + \|p_2^i - \pi(P^i, K_2, R, T)\|^2$$
2-View Bundle Adjustment (BA)

- Non-linear, joint optimization of structure, $P^i$, and motion $R, T$
- Commonly used after least square estimation of $R$ and $T$ (e.g., after 8- or 5-point algorithm)
- Optimizes $P^i, R, T$ by minimizing the **Sum of Squared Reprojection Errors**:

$$P^i, R, T = \arg\min_{P^i, R, T} \sum_{i=1}^{N} \left\| p_1^i - \pi(P^i, K_1, I, 0) \right\|^2 + \left\| p_2^i - \pi(P^i, K_2, R, T) \right\|^2$$

**Good to know:**
- Like in the formula, we typically assume the first camera as the world frame, but it's arbitrary
- Occasionally, the residual terms are weighted
- In order to not get stuck in local minima, the initial values of $P^i, R, T$ should be close to the optimum
- Can be minimized using **Levenberg–Marquardt** (more robust than Gauss-Newton to local minima)
- **Can be modified to also optimize the intrinsic parameters**
- Implementation details in Exercise 9

What is the key difference with the reprojection error minimization seen in previous lectures?
$n$-View Bundle Adjustment (BA)

- Non-linear, joint optimization of structure, $P^i$, and camera poses $C_1 = [I, 0]$, $..., C_k = [R_k, T_k]$
- Minimizes the Sum of Squared Reprojection Errors across all views

$$P^i, C_2, ..., C_k = \arg \min_{p^i, C_2, ..., C_k} \sum_{k=1}^{n} \sum_{i=1}^{N} \|p^i_k - \pi(P^i, K_k, C_k)\|^2$$

- NB: we assume the first camera as the world frame, that’s why $C_1 = [I, 0]$
Huber and Tukey Norms

• To prevent that large reprojection errors can negatively impact the optimization, a more robust norm $\rho(\cdot)$ is used instead of the $L_2$:

$$P_i, C_2, ..., C_k = \arg\min_{P_i, C_2, ..., C_k} \sum_{k=1}^{n} \sum_{i=1}^{N} \rho(p^i_k - \pi(P^i_k, K_k, C_k))$$

• $\rho(\cdot)$ is a robust cost function (Huber or Tukey) to alleviate the contribution of wrong matches:

- **Huber norm:**
  $$\rho(x) = \begin{cases} 
  x^2 & \text{if } |x| \leq k \\
  k(2|x| - k) & \text{if } |x| > k 
  \end{cases}$$

- **Tukey norm:**
  $$\rho(x) = \begin{cases} 
  \alpha^2 & \text{if } |x| \geq \alpha \\
  \alpha^2 \left(1 - \left(1 - \left(\frac{x}{\alpha}\right)^2\right)^3\right) & \text{if } |x| < \alpha 
  \end{cases}$$

These formulas are not asked at the exam but their plots and meaning is asked 😊
Outline

- Bundle Adjustment
- $n$-views SFM
$n$-View Structure From Motion

- Compute initial structure and motion using either:
  - **Hierarchical SFM**
  - **Sequential SFM** $\rightarrow$ Visual Odometry (VO)
- Refine simultaneously structure and motion through BA
Hierarchical SFM applied to random internet images

- Reconstruction from 150,000 images from Flickr associated with the tags “Rome”
- 4 million 3D points. Cloud of 496 computers. 21 hours of computation!


Most influential paper of 2009

State of the art software: [COLMAP]
Hierarchical SFM

1. Extract and match features between nearby frames
2. Build clusters consisting of 2 nearby frames
Hierarchical SFM

1. Extract and match features between nearby frames
2. Build clusters consisting of 2 nearby frames
3. Extract topological tree (e.g., count number of SIFT matches)
4. Start from the terminal nodes
   1. Compute 2-view SFM and build 3D model (point cloud)
5. Iterate according to tree structure:
   1. Merge new view by running 3-point RANSAC between 3D model and 3rd view
   2. Merge near-by models using 5.1
   3. Bundle adjust

The circle ○ corresponds to the creation of a stereo-model, the triangle △ corresponds to applying PNP, the diamond ⋄ corresponds to a fusion of two partial independent models.
$n$-View Structure From Motion

- Compute initial structure and motion using either:
  - Hierarchical SFM
  - **Sequential SFM** $\rightarrow$ Visual Odometry (VO)
- Refine simultaneously structure and motion through BA
Sequential SFM (also called Visual Odometry (VO))

- Initialize structure and motion from 2 views (bootstrapping)
- For each additional view
  - Determine pose (localization)
  - Extend structure, i.e., extract and triangulate new features (mapping)
  - Refine structure and motion through Bundle Adjustment (BA) (optimization)
VO Flow Chart: review (Lecture 01)

• VO computes the camera path incrementally (pose after pose)

Image sequence

Feature detection

Feature matching (tracking)

Motion estimation

Local optimization

**Front-end:** outputs the *relative pose* between the *last two frames*

**Back-end:** “*adjusts*” the relative poses among *multiple recent frames*
VO Flow Chart: review (Lecture 01)

- VO computes the camera path incrementally (pose after pose)

Image sequence → Feature detection → Feature matching (tracking) → Motion estimation → Local optimization

Features tracked over multiple recent frames overlaid on the last frame
VO Flow Chart: review (Lecture 01)

• VO computes the camera path incrementally (pose after pose)

Image sequence

Feature detection

Feature matching (tracking)

Motion estimation

<table>
<thead>
<tr>
<th></th>
<th>2D-2D</th>
<th>3D-3D</th>
<th>3D-2D</th>
</tr>
</thead>
</table>

Local optimization
VO Flow Chart: review (Lecture 01)

- VO computes the camera path incrementally (pose after pose)

\[ P_i, c_2, ..., c_k = \arg\min_{P_i, c_2, ..., c_k} \sum_{k=1}^{n} \sum_{i=1}^{N} \|p_i^k - \pi(P_i, K_k, c_k)\|^2 \]

Example: Bundle Adjustment:

Or Pose-Graph Optimization (see later)
2D-to-2D (already seen: Lecture 08)

Motion from 2D-to-2D feature correspondences

• Both feature **correspondences** $f_{k-1}$ and $f_k$ are specified in **image coordinates** (2D)

• The **minimal-case** solution involves **5** feature correspondences

• Popular algorithms: **5- and 8-point algorithms**
3D-to-2D (already seen: Lecture 03)

Motion from 3D-to-2D feature correspondences (i.e., Perspective from $n$ Points: PnP problem)

- $f_{k-1}$ is specified in 3D and $f_k$ in 2D
- **Minimal case:**
  - DLT algorithm: minimal case: 6 points from 3D objects, or 4 from planar objects
  - P3P algorithm: minimal case: 3 points (+1 for disambiguation)
  - EPNP algorithm: for more than 4 points
• **Motion from 3D-to-3D feature correspondences** (also known as point cloud registration problem)
  - Both $f_{k-1}$ and $f_k$ are specified in 3D. To do this, it is necessary to first triangulate 3D points (e.g. use a stereo camera)

• The **minimal-case** solution involves 3 **non-collinear** correspondences

• Popular algorithm: [Arun’87]

Arun, Huang, Blostein, “Least-Squares Fitting of Two 3-D Point Sets,” Transactions on Pattern Analysis and Machine Intelligence (PAMI), 1987. [PDF]
# Motion Estimation: Recap

<table>
<thead>
<tr>
<th>Type of correspondences</th>
<th>Monocular</th>
<th>Stereo</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D-2D</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>3D-2D</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3D-3D</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Case Study: Monocular VO (i.e., single camera VO)

This pipeline was initially proposed in PTAM (Parallel Tracking & Mapping) [Klein, ISMAR’07]

Case Study: Monocular VO (i.e., single camera VO)

Bootstrapping (i.e., initialization)
• Initialize structure and motion from 2 views: e.g., **5- or 8-point RANSAC**
• Refine structure and motion (**Bundle Adjustment**)
• How far should the two frames (i.e., keyframes) be?
Skipping frames (Keyframe Selection)

- When frames are taken at nearby positions compared to the scene distance, 3D points will exhibit large uncertainty

Small baseline $\rightarrow$ large depth uncertainty

Large baseline $\rightarrow$ small depth uncertainty
Skipping frames (Keyframe Selection)

- When frames are taken at nearby positions compared to the scene distance, 3D points will exhibit large uncertainty.
- One way to avoid this consists of **skipping frames** until the average uncertainty of the 3D points (normalized by the average distance from the scene) falls below a certain threshold. The selected frames are called **keyframes**.
- **Rule of the thumb:** add a keyframe when \( \frac{\text{keyframe distance}}{\text{average-depth}} > \text{threshold (~10-20 %)} \)
Case Study: Monocular VO (i.e., single camera VO)

Localization

• Given a 3D point cloud (map), determine the pose of each additional view
• What algorithm is used?
• How far from the last keyframe can we use it for?

<table>
<thead>
<tr>
<th>Motion estimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D-2D</td>
</tr>
<tr>
<td>3D-2D</td>
</tr>
<tr>
<td>3D-3D</td>
</tr>
</tbody>
</table>

Initial pointcloud

Keyframe 1

Keyframe 2

Current frame
Case Study: Monocular VO (i.e., single camera VO)

Localization

• Given a 3D point cloud (map), determine the pose of each additional view

Video of Oculus Insight (the VIO used in Oculus Quest): built by former Zurich-Eye team, today Oculus Zurich. Dr. Christian Forster (Oculus Zurich & co-founder of Zurich-Eye) will give a lecture on Nov. 26
Case Study: Monocular VO (i.e., single camera VO)

Extend Structure (i.e., mapping)

- Extract and triangulate new features
- Is it necessary to do this at every frame or can we just do it at keyframes?
- What are the pros and cons?

Initial pointcloud  
New triangulated points  
Keyframe 1  
Keyframe 2  
New keyframe
VO: putting all pieces together

• Let the relative motion $t_k$ from images $I_{k-1}$ to image $I_k$

$$t_{k,k-1} = \begin{bmatrix} R_{k,k-1} & T_{k,k-1} \\ 0 & 1 \end{bmatrix}$$

• Concatenate adjacent transformations to recover the current pose:

$$C_n = C_{n-1} t_{n,n-1}$$

• Optimize over the last $m$ poses to refine the trajectory (Pose-Graph or Bundle Adjustment)

Sliding-window bundle adjustment
Optimization

VO flowchart:

1. Image sequence
2. Feature detection
3. Feature matching (tracking)
4. Motion estimation
5. Local optimization

Example: Bundle Adjustment:

\[ p_i, c_2, ..., c_k = \arg\min_{p_i, c_2, ..., c_k} \sum_{k=1}^{n} \sum_{i=1}^{N} \| p_k^i - \pi(p_i, K_k, C_k) \|^2 \]

Or Pose-Graph Optimization (see later)
Pose-Graph Optimization

• So far we assumed that the transformations are between consecutive frames

• However, transformations can also be computed between non-adjacent frames $T_{ij}$ (e.g., when features from previous keyframes are still observed). They can be used as additional constraints to improve cameras poses by minimizing the following:

$$\{C_1, \ldots, C_n\} = \arg\min_{\{C_1, \ldots, C_n\}} \sum_i \sum_j \|C_i - C_j t_{ij}\|^2$$

• For efficiency, only the last $m$ keyframes are used

• Gauss-Newton or Levenberg-Marquadt are typically used to minimize it. For large graphs, efficient open-source tools: g2o, GTSAM, SLAM++, Google Ceres
Bundle Adjustment (BA)

• Similar to pose-graph optimization but it also optimizes 3D points

\[ \rho(p) \] is the Huber or Tukey norm

• Gauss-Newton or Levenberg-Marquadt are typically used to minimize it. For large graphs, efficient open-source tools: g2o, GTSAM, SLAM++, Google Ceres
Bundle Adjustment vs Pose-graph Optimization

- BA is more precise than pose-graph optimization because it adds additional constraints (landmark constraints).

- But more costly: $O((qN + ln)^3)$ with $N$ being the number of points, $n$ the number of cameras poses and $q$ and $l$ the number of parameters for points and camera poses. Workarounds:
  - A small window size limits the number of parameters for the optimization and thus makes real-time bundle adjustment possible.
  - It is possible to reduce the computational complexity by just optimizing over the camera parameters and keeping the 3-D landmarks fixed, e.g., (motion-only BA).
Loop Closure Detection (i.e., Place Recognition)

• **Relocalization problem:**
  - During VO, tracking can be lost (due to occlusions, low texture, quick motion, illumination change)

• **Solution:** Re-localize camera pose and continue

• **Loop closing problem**
  - When you go back to a previously mapped area:
    • Loop detection: to avoid map duplication
    • Loop correction: to compensate the accumulated drift
  - In both cases you need a place recognition technique

We will address place recognition in Lecture 12
VO vs. Visual SLAM (recap from Lecture 01)

• **Visual Odometry**
  • Focus on incremental estimation
  • **Guarantees local consistency** (i.e., estimated trajectory is locally correct, but not globally, i.e. from the start to the end)

• **Visual SLAM** (Simultaneous Localization And Mapping)
  • **SLAM = visual odometry + loop detection & closure**
  • **Guarantees global consistency** (the estimated trajectory is globally correct, i.e. from the start to the end)

Image courtesy of [Clemente et al., RSS’07]
Open Source Monocular VO and SLAM algorithms

- PTAM
- ORB-SLAM
- SVO
- LSD-SLAM
- DSO

**Indirect methods:** Minimize the feature reprojection error

**Direct methods:** Minimize the feature photometric error
PTAM: Parallel Tracking and Mapping

- Monocular only
- **Feature based**
  - FAST corners + patch descriptors
  - Minimizes reprojection error
  - Jointly optimizes poses & structure (sliding window)
- First to propose **keyframe-based VO**
- First to propose **alternation of localization** (i.e., camera tracking) and **mapping**
- Tracking and mapping running in **two independent threads**: updated map is used by localization thread asynchronously, as soon it becomes available
- Includes:
  - Relocalization
  - No global optimization, only local
- **Real-time (30Hz)**, however global optimization is not done in real time but asynchronously every once in a while

ORB-SLAM

• Supports both **monocular and stereo** cameras

• **Feature based**
  • FAST corners + ORB descriptors
  • ORB: binary descriptor, very fast to compute and match (Hamming distance)
  • Jointly optimizes poses & structure (sliding window)

• **Same workflow as PTAM** (keyframe based, alternation of localization and mapping as independent threads)

• Includes:
  • Loop closing
  • Relocalization
  • Final optimization

• **Real-time (30Hz)**, however global optimization is not done in real time but asynchronously every once in a while

Mur-Artal, Montiel, Tardos, ORB-SLAM: Large-scale Feature-based SLAM, IEEE Transactions on Robotics (T-RO), 2015. [PDF, code, videos]
Indirect vs Direct Methods

- **Indirect methods**
  1. Extract & match features + 3-point RANSAC
  2. Bundle Adjust by minimizing the Reprojection Error:

  \[
  p^i, R, T = \arg \min_{p^i, R, T} \sum_{i=1}^{N} \rho \left( p^i_k - \pi(p^i, K, R, T) \right)
  \]

- **Direct methods**
  1. No feature extraction & no RANSAC needed. Instead, directly minimize Photometric Error:

  \[
  p^i, R, T = \arg \min_{p^i, R, T} \sum_{i=1}^{N} \rho \left( l^i_{k-1}(p^i_{k-1}) - l^i_k \left( \pi(p^i, K, R, T) \right) \right)
  \]

What are their pros and cons?

Irani, Anandau, *All about direct methods*, Springer’99. [PDF]
Indirect vs Direct Methods

- **Indirect methods**
  1. Extract & match features + 3-point RANSAC
  2. Bundle Adjust by minimizing the **Reprojection Error**:

\[
P^i, R, T = \arg \min_{P^i, R, T} \sum_{i=1}^{N} \rho \left( p^i - \pi \left( P^i, K, R, T \right) \right)
\]

- **Direct methods**
  1. No feature extraction & no RANSAC needed. Instead, directly minimize **Photometric Error**:

\[
P^i, R, T = \arg \min_{P^i, R, T} \sum_{i=1}^{N} \rho \left( I_k - \pi \left( P^i, K, R, T \right) \right)
\]

  ✓ Can cope with large frame-to-frame motions (large basin of convergence)
  × Slow due to costly feature extraction and matching
  × Matching Outliers (RANSAC)

✓ All information in the image can be exploited (higher accuracy, higher robustness to motion blur and weak texture (i.e., weak gradients))
✓ Increasing camera frame-rate reduces computational cost per frame (no RANSAC needed)
× Very sensitive to intial value → limited frame-to-frame motion (small basin of convergence)

Irani, Anandau, *All about direct methods*, Springer’99. [PDF](#)
Direct Methods: Dense vs Semi-dense vs Sparse

Dense

Semi-Dense

Sparse

300'000+ pixels

~10,000 pixels

~2,000 pixels

Direct Methods: Dense vs Semi-dense vs Sparse

**Dense**
- Live incremental reconstruction of a large scene
- Texture mapped model
- Inverse depth solution
- 300'000+ pixels
- DTAM [Newcombe ‘11], REMODE [Pizzoli’14]

**Semi-Dense**
- LSD-SLAM [Engel’14]
- ~10,000 pixels
- e.g., 120 feature patches × (4×4 pixels per patch)

**Sparse**
- SVO [Forster’14], DSO [Engel’17]
- ~2,000 pixels
- SVO with a single camera on Euroc dataset

Direct Methods: Dense vs Semi-dense vs Sparse

- What is the influence of the motion baseline on the convergence rate of direct methods?

For small motion baselines, $\|T\|$, the photometric error is usually small.

Direct Methods: Dense vs Semi-dense vs Sparse

• What is the influence of the motion baseline on the convergence rate of direct methods?
• We can use photorealistic simulation to answer this question and generate thousands of data

Findings:
• Dense and Semi-dense behave similarly
• Weak gradients are not informative for the optimization
• Dense methods are only useful with motion blur, defocus, and weak-texture regions
• Sparse methods behave equally well as dense or semi-dense for small motion baselines

LSD-SLAM

- Supports both **monocular** and **stereo** cameras
- **Direct** (photometric error) + **Semi-Dense** formulation
  - 3D structure represented as semi-dense depth map
  - Minimizes **photometric error**
  - Separately optimizes poses & structure (sliding window)
- **Same workflow as PTAM** (keyframe based, alternation of localization and mapping as independent threads)
- Includes:
  - Loop closing
  - Relocalization
  - Final optimization
- **Real-time (30Hz)**, however global optimization is not done in real time but asynchronously every once in a while

Engel, Schoeps, Cremers, *LSD-SLAM: Large-scale Semi-Dense SLAM*, European Conference on Computer Vision (ECCV), 2014. [PDF, code, videos].
DSO

• Supports both **monocular** and **stereo** cameras

• **Direct (photometric error) + Sparse** formulation
  • 3D structure represented as **sparse large gradients’ depth map**
  • Minimizes **photometric error**
  • Jointly optimizes poses & structure (sliding window)
  • Incorporates photometric correction to compensate exposure time change \((\Delta t_{k-1}, \Delta t_k)\)

\[
p^i, R, K = \arg \min_{p^i,R,K} \sum_{i=1}^{N} \rho \left( I_{k-1}(p^i_{k-1}) - \frac{\Delta t_{k-1}}{\Delta t_{k}} I_k \left( \pi(p^i, K, R, T) \right) \right)
\]

• **Same workflow as PTAM** (keyframe based, alternation of localization and mapping as independent threads)

• **Real-time (30Hz)**, however global optimization is not done in real time but asynchronously every once in a while

SVO

- Supports both **monocular, stereo, and multi camera** systems as well as omnidirectional models (fisheye and catadioptric)
- Combines **indirect + direct methods**
  - **Direct** (minimizes photometric error)
    - Used for frame-to-frame motion estimation
    - Corners and edgelets
    - Jointly optimizes poses & structure (sliding window)
  - **Indirect** (minimizes reprojection error)
    - Frame-to-Keyframe pose refinement
- **Mapping**
  - **Probabilistic depth** estimation (heavy tail Gaussian distribution)
- **Same workflow as PTAM** (keyframe based, alternation of localization and mapping as independent threads)
- **Faster than real-time (up to 400Hz): 400 fps** on i7 laptops and **100 fps** on smartphone PCs (Odroid (ARM), NVIDIA Jetsons)

SVO

- Supports both **monocular, stereo, and multi camera** systems as well as omnidirectional models (fisheye and catadioptric)
- Combines **indirect + direct methods**
  - **Direct** (minimizes photometric error)
  - Used for frame-to-frame motion estimation
  - **Corners and edgelets**
  - **Jointly optimizes poses & structure** (sliding window)
  - **Indirect** (minimizes reprojection error)
  - **Frame-to-Keyframe** pose refinement

- **Mapping**
  - **Probabilistic depth** estimation (heavy tail Gaussian distribution)
- **Same workflow as PTAM** (keyframe based, alternation of localization and mapping as independent threads)
- **Faster than real-time (up to 400Hz): 400 fps** on i7 laptops and **100 fps** on smartphone PCs (Odroid (ARM), NVIDIA Jetsons)

### Processing times of ORB-SLAM, LSD-SLAM, DSO, SVO

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>St.D.</th>
<th>CPU@20 fps</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVO Mono</td>
<td>2.53</td>
<td>0.42</td>
<td>55 ±10%</td>
</tr>
<tr>
<td>ORB Mono SLAM (No loop closure)</td>
<td>29.81</td>
<td>5.67</td>
<td>187 ±32%</td>
</tr>
<tr>
<td>LSD Mono SLAM (No loop closure)</td>
<td>23.23</td>
<td>5.87</td>
<td>236 ±37%</td>
</tr>
<tr>
<td>DSO</td>
<td>20.12</td>
<td>4.03</td>
<td>181 ±27%</td>
</tr>
</tbody>
</table>

Processing times in milliseconds

Applications of SVO and its surrogates

- DJI products (VIO front-end)
- Magic Leap
- Oculus
- Huawei
- ...

Position error: 5 mm, height: 1.5 m – Down-looking camera

Speed: 4 m/s, height: 3 m – Down-looking camera

Robustness to dynamic scenes (down-looking camera)

Automatic recovery from aggressive flight

More here: http://rpg.ifi.uzh.ch/svo2.html
Dacuda 3D (now Magic Leap Zurich)

• Fully immersive VR (running on iPhone 6)
• 6DoF Head tracking by SVO
Startup: “Zurich-Eye” – Today: Facebook-Oculus Zurich

- **Vision-based Localization and Mapping** systems for mobile robots
- Born in Sep. 2015, became **Facebook-Oculus Zurich** in Sep. 2016. Today, **100 employees**.
Startup: “Zurich-Eye” – Today: Facebook-Oculus Zurich

- **Vision-based Localization and Mapping** systems for mobile robots
- Born in Sep. 2015, became **Facebook-Oculus Zurich** in Sep. 2016. Today, **100 employees**.
- In 2018, Zurich-Eye launched **Oculus Quest**: [https://youtu.be/xwW-1mbemGc](https://youtu.be/xwW-1mbemGc)
We will have a lecture by Christian Forster, from Oculus Zurich, on November 26
How can we evaluate the accuracy of VO/SLAM algorithms?

• Idea: compare the estimated trajectory against ground truth trajectory (from GPS, motion tracking systems), but the key question is what error metric should be used?

• Issues:
  • Different reference frames
  • Different scale
  • Different times stamps

• Naïve solution (not used anymore): Maybe align the first poses and measure the end-pose error?

• Not repeatable:
  • Most VIOs are non-deterministic (e.g., RANSAC, multithreading) → every time you run your VIO on the same dataset, you get different results

• Not meaningful:
  • sensitive to the trajectory shape (the number of turns of a trajectory greatly affects the end-pose error)
  • does not capture the error statistics
Metric 1: Absolute Trajectory Error (ATE)

- **Step 1:** align the estimated trajectory to the ground truth from the start to the end using a similarity transformation (i.e., $R, T, s$) by minimizing the sum of square position errors

$$R, T, s = \arg\min_{R,T,s} \sum_{k=0}^{n} \| \hat{C}_k - sRC_k - T \|^2$$

- **Step 2:** compute Root Mean Square Error (RMSE) after alignment:

$$RMSE = \sqrt{\frac{\sum_{k=1}^{n} \| \hat{C}_k - sRC_k - T \|^2}{n}}$$

- **Pros and cons:**
  - ✓ Single-number metric
  - ✓ Captures the global error (accuracy of the global trajectory)
  - ✗ Does not capture the relative error (accuracy of the local trajectory estimate)

---

Metric 2: Relative Trajectory Error (RTE)

- Computes **error statistics** of sub-trajectories of specified lengths

- **Pros and cons:**
  - ✓ Informative statistics: captures the relative error (accuracy of the local trajectory estimate)
  - ✗ Complicated to compute and rank, but the good news is that there is code for it 😊 (toolbox, link below)

---

Things to remember

• Hierarchical SFM
• VO flowchart
  • Monocular VO
  • Stereo VO
  • Keyframe selection
• Bundle adjustment vs pose-graph optimization
• Indirect vs direct methods
• Dense vs semi-dense vs sparse
• Popular open-source VO algorithms
• ATE and RTE trajectory evaluation metrics
Readings


Understanding Check

Are you able to answer the following questions:

• Bundle Adjustment and Pose Graph Optimization. Mathematical expressions and illustrations. Pros and cons.

• Are you able to describe hierarchical and sequential SFM for monocular VO?

• What are keyframes? Why do we need them and how can we select them?

• Are you able to define loop closure detection? Why do we need loops? How can we detect loop closures? (make link to other lectures)

• Are you able to describe the differences between feature-based methods and direct methods?

• Sparse vs semi-dense vs dense. What are their pros and cons?

• Are you able to provide a list of the most popular open source VO and VSLAM algorithms?

• Difference between SFM, VO, SLAM (see also lecture 01)

• How do we evaluate the accuracy of visual odometry? What are ATE and RTE, how are they computed and what do they capture?