Lecture 13
Visual Inertial Fusion
(advanced)

Davide Scaramuzza
Lab Exercise 6 - Today

- Room ETH HG E 1.1 from 13:15 to 15:00
- Work description: Bundle Adjustment
Recall: VO Working Principle

1. Compute the relative motion $T_k$ from images $I_{k-1}$ to image $I_k$

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

2. Concatenate them to recover the full trajectory

   $C_n = C_{n-1}T_n$

3. An optimization over the last $m$ poses can be done to refine locally the trajectory (Pose-Graph or Bundle Adjustment)
Pose-Graph Optimization

- So far we assumed that the transformations are between consecutive frames.

- Transformations can be computed also 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:

$$C_k = \arg\min_{C_k} \sum_i \sum_j \|C_i - C_j T_{ij}\|^2$$

- 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

\[
X^i, C_k = \arg\min_{X^i, C_k} \sum_i \sum_k \rho \left( p_k^i - \pi(X^i, C_k) \right)
\]

- \( \rho_H() \) is a robust cost function (e.g., Huber or Tukey cost) to penalize wrong matches
- In order to not get stuck in local minima, the initialization should be close to the minimum
- Gauss-Newton or Levenberg-Marquadt are typically used to minimize it. For large graphs, efficient open-source tools: g2o, GTSAM, SLAM++, Google Ceres
Huber and Tukey Norms

Goal: penalize the influence of wrong matches (i.e., high reprojection error)

Huber norm:

\[
\rho(x) = \begin{cases} 
  x^2 & \text{if } |x| \leq k \\
  k(2|x| - k) & \text{if } |x| \geq k 
\end{cases}
\]

Tukey norm:

\[
\rho(x) = \begin{cases} 
  \alpha^2 & \text{if } |x| \geq \alpha \\
  \alpha^2 \left(1 - \left(1 - \left(\frac{x}{\alpha}\right)^2\right)^3\right) & \text{if } |x| \leq \alpha 
\end{cases}
\]
Bundle Adjustment vs Pose-graph Optimization

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

- But **more costly**: $O((qM + lN)^3)$ with $M$ and $N$ being the number of points and 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)*
Outline

- Introduction
- IMU model and Camera-IMU system
- Different paradigms
  - Closed-form solution
  - Filtering approaches
  - Maximum a posteriori estimation (non linear optimizers)
    - Fixed-lag Smoothing (aka sliding window estimators)
    - Full smoothing methods
- Camera-IMU extrinsic calibration and Synchronization
What is an IMU?

- Inertial Measurement Unit
  - Angular velocity
  - Linear Accelerations
What is an IMU?

- Different categories
  - Mechanical ($100,000)
  - Optical ($20,000)
  - MEMS (from 1$ (phones) to 1,000$)
  - ....

- For mobile robots: MEMS IMU
  - Cheap
  - Power efficient
  - Light weight and solid state
MEMS Accelerometer

A spring-like structure connects the device to a seismic mass vibrating in a capacity divider. A capacitive divider converts the displacement of the seismic mass into an electric signal. Damping is created by the gas sealed in the device.
MEMS Gyroscopes

- MEMS gyroscopes measure the Coriolis forces acting on MEMS vibrating structures (tuning forks, vibrating wheels, or resonant solids)
- Their working principle is similar to the haltere of a fly
- Haltere are small structures of some two-winged insects, such as flies. They are flapped rapidly and function as gyroscopes, informing the insect about rotation of the body during flight.
Why IMU?

- Monocular vision is scale ambiguous.
- Pure vision is not robust enough
  - Low texture
  - High dynamic range
  - High speed motion

Robustness is a critical issue: Tesla accident

“The autopilot sensors on the Model S failed to distinguish a white tractor-trailer crossing the highway against a bright sky.”
Why vision?

- Pure IMU integration will lead to large drift (especially cheap IMUs)
  - Will see later mathematically
  - Intuition
    - Integration of angular velocity to get orientation: error proportional to $t$
    - Double integration of acceleration to get position: if there is a bias in acceleration, the error of position is proportional to $t^2$
    - Worse, the actually position error also depends on the error of orientation.

```
<table>
<thead>
<tr>
<th>Grade</th>
<th>Accelerometer Bias Error [mg]</th>
<th>Horizontal Position Error [m]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1s</td>
</tr>
<tr>
<td>Navigation</td>
<td>0.025</td>
<td>0.13 mm</td>
</tr>
<tr>
<td>Tactical</td>
<td>0.3</td>
<td>1.5 mm</td>
</tr>
<tr>
<td>Industrial</td>
<td>3</td>
<td>15 mm</td>
</tr>
<tr>
<td>Automotive</td>
<td>125</td>
<td>620 mm</td>
</tr>
</tbody>
</table>
```

# Why visual inertial fusion?

**IMU and vision are complementary**

<table>
<thead>
<tr>
<th>Cameras</th>
<th>IMU</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓ Precise in slow motion</td>
<td>✓ Robust</td>
</tr>
<tr>
<td>✓ Rich information for other purposes</td>
<td>✓ High output rate (~1,000 Hz)</td>
</tr>
<tr>
<td>x Limited output rate (~100 Hz)</td>
<td>✓ Accurate at high acceleration</td>
</tr>
<tr>
<td>x Scale ambiguity in monocular setup.</td>
<td>x Large relative uncertainty when at low acceleration/angular velocity</td>
</tr>
<tr>
<td>x Lack of robustness</td>
<td>x Ambiguity in gravity / acceleration</td>
</tr>
</tbody>
</table>

In common: state estimation based on visual or/and inertial sensor is dead-reckoning, which suffers from drifting over time.  
(solution: loop detection and loop closure)
Outline

- Introduction
- IMU model and Camera-IMU system
- Different paradigms
  - Closed-form solution
  - Filtering approaches
    - Maximum a posteriori estimation (non linear optimizers)
      - Fixed-lag Smoothing (aka sliding window estimators)
      - Full smoothing methods
- Camera-IMU extrinsic calibration and Synchronization
IMU model: Measurement Model

- Measures angular velocity and acceleration in the body frame:

\[
\begin{align*}
\mathbf{B} \hat{\mathbf{\omega}}_{WB}(t) &= \mathbf{B} \mathbf{\omega}_{WB}(t) + \mathbf{b}^g(t) + \mathbf{n}^g(t) \\
\mathbf{B} \hat{\mathbf{a}}_{WB}(t) &= \mathbf{R}_{BW}(t)\left(\mathbf{w} \mathbf{a}_{WB}(t) - \mathbf{w} \mathbf{g}\right) + \mathbf{b}^a(t) + \mathbf{n}^a(t)
\end{align*}
\]

where the superscript \(^g\) stands for Gyroscope and \(^a\) for Accelerometer.

Notations:
- Left subscript: reference frame in which the quantity is expressed
- Right subscript \{Q\}{Frame1}{Frame2}: Q of Frame2 with respect to Frame1
- Noises are all in the body frame
IMU model: Noise Property

- Additive Gaussian white noise: \( n^g(t), \ n^a(t) \)

\[
E[n(t)] = 0 \\
E[n(t_1)n(t_2)] = \sigma^2 \delta(t_1 - t_2)
\]

\[
n[k] = \sigma_d w[k] \\
w[k] \sim N(0,1) \\
\sigma_d = \sigma / \sqrt{\Delta t}
\]

- Bias: \( b^g(t), \ b^a(t) \)

\[
\dot{b}(t) = \sigma_b w(t)
\]

\[
b[k] = b[k-1] + \sigma_{bd} w[k] \\
\sigma_{bd} = \sigma_b \sqrt{\Delta t} \\
w[k] \sim N(0,1)
\]

The biases are usually estimated with the other states
- can change every time the IMU is started
- can change due to temperature change, mechanical pressure, etc.

IMU model: Integration

Per component: \{t\} stands for \{B\}ody frame at time \(t\)

\[
p_{\text{wt}_2} = p_{\text{wt}_1} + (t_2 - t_1)v_{\text{wt}_1} + \int_{t_1}^{t_2} \left( R_{\text{wt}}(t)(\ddot{a}(t) - b^a(t)) + w g \right) dt^2
\]

\[
v_{\text{wt}_2} = v_{\text{wt}_1} + \int_{t_1}^{t_2} \left( R_{\text{wt}}(t)(a(t) - b^a(t)) + w g \right) dt
\]

- Depends on initial position and velocity
- The rotation \(R(t)\) is computed from the gyroscope

Rotation is more involved, will use quaternion as example:

\[
\dot{q}_{\text{wt}}(t) = \frac{1}{2} \Omega(\omega(t))q_{\text{wt}}(t) \quad \Rightarrow \quad q_{\text{wt}_2}(t_2) = \Theta(t_1, t_2)q_{\text{wt}_1}(t_1)
\]

\(\Theta(t_1, t_2)\) is the state transition matrix.

IMU model: Integration

Per component: \{t\} stands for \{B\}ody frame at time \(t\)

\[
p_{wt_2} = p_{wt_1} + (t_2 - t_1) v_{wt_1} + \int_{t_1}^{t_2} \int \left[ R_{wt}(t) \left( \ddot{a}(t) - b^a(t) \right) \right] + w g dt^2
\]

- Depends on initial position and velocity
- The rotation \(R(t)\) is computed from the gyroscope

There can be multiple cameras.
Outline

- Introduction
- IMU model and Camera-IMU system
- Different paradigms
  - Closed-form solution
  - Filtering approaches
    - Maximum a posteriori estimation (non linear optimizers)
      - Fixed-lag Smoothing (aka sliding window estimators)
      - Full smoothing methods
- Camera-IMU extrinsic calibration and Synchronization
Different paradigms

- **Loosely coupled:**
  - Treats VO and IMU as two separate (not coupled) black boxes
    - Each black box estimates pose and velocity from visual (up to a scale) and inertial data (absolute scale)

- **Tightly coupled:**
  - Makes use of the raw sensors’ measurements:
    - 2D features
    - IMU readings
    - More accurate
    - More implementation effort

- In the following slides, we will only see tightly coupled approaches
The Loosely Coupled Approach

- **Feature Extraction & matching**
- **IMU Integration**
- **VO**
- **Fusion**

**Inputs:**
- Images
- IMU measurements

**Outputs:**
- Refined Position
- Orientation
- Velocity

**Flow:**
1. Feature Extraction & matching
2. IMU Integration
3. VO
4. Fusion
The Tightly Coupled Approach

images

Feature Extraction & matching

IMU measurements

IMU Integration

2D features

Fusion

Position Orientation Velocity

Refined Position Orientation Velocity
Filtering: Visual Inertial Formulation

System states:

**Tightly coupled:** \( X = \begin{bmatrix} \dot{w}p(t); \dot{q}_{WB}(t); \dot{w}v(t); b^a(t); b^g(t); \dot{w}L_1; \ldots; \dot{w}L_K \end{bmatrix} \)

**Loosely coupled:** \( X = \begin{bmatrix} \dot{w}p(t); \dot{q}_{WB}(t); \dot{w}v(t); b^a(t); b^g(t) \end{bmatrix} \)

Corke, An Introduction to Inertial and Visual Sensing, IJRR’07
Outline

- Introduction
- IMU model and Camera-IMU system
- Different paradigms
  - Closed-form solution
  - Filtering approaches
    - Maximum a posteriori estimation (non linear optimizers)
      - Fixed-lag Smoothing (aka sliding window estimators)
      - Full smoothing methods
- Camera-IMU extrinsic calibration and Synchronization
Closed-form Solution (1D case)

- The absolute pose $x$ is known up to a scale $s$, thus

$$x = s \tilde{x}$$

- From the IMU

$$x = x_0 + v_0(t_1 - t_0) + \int_{t_0}^{t_1} a(t) dt$$

- By equating them

$$s \tilde{x} = x_0 + v_0(t_1 - t_0) + \int_{t_0}^{t_1} a(t) dt$$

- As shown in [Martinelli’14], for 6DOF, both $s$ and $v_0$ can be determined in closed form from a **single feature observation and 3 views**. $x_0$ can be set to 0.

Martinelli, Closed-form solution of visual-inertial structure from motion, International Journal of Computer Vision, 2014
Closed-form Solution (1D case)

\[
\begin{align*}
\overline{s x}_1 &= v_0(t_1 - t_0) + \int_{t_0}^{t_1} a(t) dt \\
\overline{s x}_2 &= v_0(t_2 - t_0) + \int_{t_0}^{t_2} a(t) dt
\end{align*}
\]

\[
\begin{bmatrix}
\overline{s x}_1 \\
\overline{s x}_2
\end{bmatrix} =
\begin{bmatrix}
(t_0 - t_1) \\
(t_0 - t_2)
\end{bmatrix}
\begin{bmatrix}
\int_{t_0}^{t_1} a(t) dt \\
\int_{t_0}^{t_2} a(t) dt
\end{bmatrix}
\]

Martinelli, Closed-form solution of visual-inertial structure from motion, International Journal of Computer Vision, 2014
Closed-form Solution (general case)

- Considers N feature observations and 6DOF case
- Can be used to initialize filters and smoothers (which always need an initialization point)
- More complex to derive than the 1D case. But it also reaches a linear system of equations that can be solved using the pseudoinverse:

\[ AX = S \]

\( X \) is the vector of unknowns:
- 3D Point distances (wrt the first camera)
- Absolute scale
- Initial velocity
- Gravity vector
- Biases

\( A \) and \( S \) contain 2D feature coordinates, acceleration, and angular velocity measurements.

- Martinelli, Vision and IMU data fusion: Closed-form solutions for attitude, speed, absolute scale, and bias determination, TRO’12
- Martinelli, Closed-form solution of visual-inertial structure from motion, Int. Journal of Comp. Vision, JCV’14
- Kaiser, Martinelli, Fontana, Scaramuzza, Simultaneous state initialization and gyroscope bias calibration in visual inertial aided navigation, IEEE RAL’17
Outline

- Introduction
- IMU model and Camera-IMU system
- Different paradigms
  - Closed-form solution
  - Filtering approaches
    - Maximum a posteriori estimation (non linear optimizers)
      - Fixed-lag Smoothing (aka sliding window estimators)
      - Full smoothing methods
- Camera-IMU extrinsic calibration and Synchronization
Different paradigms

<table>
<thead>
<tr>
<th>Filtering</th>
<th>Fixed-lag Smoothing</th>
<th>Full smoothing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only updates the most recent states • (e.g., extended Kalman filter)</td>
<td>Optimizes window of states • Marginalization • Nonlinear least squares optimization</td>
<td>Optimize all states • Nonlinear Least squares optimization</td>
</tr>
<tr>
<td>✗1 Linearization</td>
<td>✗Re-Linearize</td>
<td>✗Re-Linearize</td>
</tr>
<tr>
<td>✗Accumulation of linearization errors</td>
<td>✗Accumulation of linearization errors</td>
<td>✗Sparse Matrices</td>
</tr>
<tr>
<td>✗Gaussian approximation of marginalized states</td>
<td>✗Gaussian approximation of marginalized states</td>
<td>✗Highest Accuracy</td>
</tr>
<tr>
<td>✓Fastest</td>
<td>✓Fast</td>
<td>✗Slow (but fast with GTSAM)</td>
</tr>
</tbody>
</table>

- Fixed-lag Smoothing:
  - Only updates the most recent states.
  - (e.g., extended Kalman filter)

- Full smoothing:
  - Optimizes all states.
  - Nonlinear Least squares optimization.
Filtering: Kalman Filter in a Nutshell

- Assumptions: **linear system, Gaussian noise**

**System dynamics**

\[
x(k) = A(k-1)x(k) + u(k-1) + v(k-1)
\]

\[
z(k) = H(k)x(k) + w(k)
\]

- \( x(k) \): state
- \( u(k) \): control input, can be 0
- \( z(k) \): measurement

**Kalman Filter**

\[
x_m(0) = x_0, P_m(0) = P_0
\]

**Prediction**

\[
\hat{x}_p(k) = A(k-1)\hat{x}_m(k-1) + u(k-1)
\]

\[
P_p(k) = A(k-1)P_m(k-1)A^T(k-1) + Q(k-1)
\]

**Measurement update**

\[
P_m(k) = \left( P_p(k) + H^T(k)R^{-1}(k)H(k) \right)^{-1}
\]

\[
\hat{x}_m(k) = \hat{x}_p(k) + P_m(k)H^T(k)R^{-1}(k)(z(k) - H(k)\hat{x}_p(k))
\]

Weight between the model prediction and measurement
Filtering: Kalman Filter in a Nutshell

- Nonlinear system: linearization

**System dynamics**

\[
x(k) = q_{k-1}(x(k-1), u(k-1), v(k-1))
\]

\[
z(k) = h_k(x(k), w(k))
\]

Process and measurement noise and initial state are Gaussian.

**Extended Kalman Filter**

**Prediction**

\[
\hat{x}_p(k) = q_{k-1}(\hat{x}_m(k-1), u(k-1), 0)
\]

\[
P_p(k) = A(k-1)P_m(k-1)A^T(k-1) + L(k-1)Q(k-1)L^T(k-1)
\]

**Measurement update**

\[
K(k) = P_p(k)H^T(k)(H(k)P_p(k)H^T(k) + M(k)R(k)M^T(k))^{-1}
\]

\[
\hat{x}_m(k) = \hat{x}_p(k) + K(k)(z(k) - h_k(\hat{x}_p, 0))
\]

\[
P_m(k) = (I - K(k)H(k))P_p(k)
\]

Key idea:

- Linearize around the estimated states
- \(A(k), L(k), H(k), M(k)\) are partial derivatives with respect to states and noise
Filtering: Visual Inertial Formulation

System states:

Tightly coupled: $X = \begin{bmatrix} w p(t); q_{WB}(t); w v(t); b^a(t); b^g(t); w L_1; w L_2; \ldots; w L_K \end{bmatrix}$

Loosely coupled: $X = \begin{bmatrix} w p(t); q_{WB}(t); w v(t); b^a(t); b^g(t) \end{bmatrix}$

Process Model: from IMU

- Integration of IMU states (rotation, position, velocity)
- Propagation of IMU noise
  - needed for calculating the Kalman Filter gain
Filtering: Visual Inertial Formulation

Measurement Model: from camera

Transform point to camera frame

\[
\begin{bmatrix}
    c_x \\
    c_y \\
    c_z
\end{bmatrix}
= \mathbf{R}_{CB} \left( \mathbf{R}_{BW} \left( \mathbf{wL} - \mathbf{wp} \right) - \mathbf{p}_{CB} \right)
\]

\[
\mathbf{H}_{\text{Landmark}} = \mathbf{R}_{CB} \mathbf{R}_{BW} = \mathbf{R}_{CW}
\]

\[
\mathbf{H}_{\text{pose}} = \mathbf{R}_{CB} \left[ -\mathbf{R}_{BW} \begin{bmatrix} \mathbf{B} \mathbf{L} \end{bmatrix}_x \right]
\]

Pinhole projection (without distortion)

\[
\begin{bmatrix}
    u \\
    v
\end{bmatrix}
= \begin{bmatrix}
    f_x \frac{c_x}{c_z} + c_x \\
    f_y \frac{c_y}{c_z} + c_y
\end{bmatrix}
\]

\[
\mathbf{H}_{\text{proj}} = \begin{bmatrix}
    f_x \frac{1}{z} & 0 & -f_x \frac{x}{z^2} \\
    0 & f_y \frac{1}{z} & -f_y \frac{y}{z^2}
\end{bmatrix}
\]

Drop C for clarity

\[
\mathbf{H}_X = \mathbf{H}_{\text{proj}} \mathbf{H}_{\text{pose}}
\]

\[
\mathbf{H}_L = \mathbf{H}_{\text{proj}} \mathbf{H}_{\text{Landmark}}
\]
Filtering: ROVIO

- EKF state: \( X = \left[ w_p(t); w_{\text{WB}}(t); w_v(t); b^a(t); b^g(t); w_{L_1}; w_{L_2}; \ldots; w_{L_K} \right] \)
- Minimizes the photometric error instead of the reprojection error

ROVIO: Robust Visual Inertial Odometry Using a Direct EKF-Based Approach

http://github.com/ethz-asl/rovio

Michael Bloesch, Sammy Omari, Marco Hutter, Roland Siegwart

Bloesch, Michael, et al. "Iterated extended Kalman filter based visual-inertial odometry using direct photometric feedback“, IJRR’17
Filtering: Potential Problems

- Wrong linearization point
  - Linearization depends on the current estimates of states, which may be erroneous
  - Linearization around different values of the same variable leads to estimator inconsistency (wrong observability/covariance estimation)

- Wrong covariance/initial states
  - Intuitively, wrong weights for measurements and prediction
  - May be overconfident/underconfident

- Explosion of number of states
  - Roughly cubic in the number of the states
  - Each 3D point: 3 variables
  - → a few landmarks (~20) are typically tracked to allow real-time operation
Filtering: Problems

Alternative: MSCKF [Mourikis & Roumeliotis, ICRA’07]: used in Google Tango

- Keeps a window of recent states and updates them using EKF
- Incorporate visual observations without including point positions into the states

Mourikis & Roumeliotis, A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation, TRO’16
Li, Mingyang, and Anastasios I. Mourikis, High-precision, consistent EKF-based visual–inertial odometry, IJRR’13
Filtering: Google Tango

Mourikis & Roumeliotis, A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation, TRO’16
Li, Mingyang, and Anastasios I. Mourikis, High-precision, consistent EKF-based visual–inertial odometry, IJRR’13
Outline

- Introduction
- IMU model and Camera-IMU system
- Different paradigms
  - Closed-form solution
  - Filtering approaches
    - Maximum a posteriori estimation (non linear optimizers)
      - Fixed-lag Smoothing (aka sliding window estimators)
      - Full smoothing methods
- Camera-IMU extrinsic calibration and Synchronization
Maximum A Posteriori (MAP) Estimation

- Fusion solved as a non-linear optimization problem
- Increased accuracy over filtering methods

\[ x_k = f(x_{k-1}) \]
\[ z_i = h(x_{i_k}, l_{ij}) \]

\[ X = \{x_1, \ldots, x_N\}: \text{ robot states} \]
\[ L = \{l_1, \ldots\}: \text{ 3D points} \]
\[ Z = \{z_1, \ldots, z_M\}: \text{ features & IMU measurements} \]

\[ \{X^*, L^*\} = \arg\max_{\{X, L\}} P(X, L | Z) \]
\[ = \arg\min_{\{X, L\}} \left\{ \sum_{k=1}^{N} \left\| f(x_{k-1}) - x_k \right\|_{A_k}^2 + \sum_{i=1}^{M} \left\| h(x_{i_k}, l_{ij}) - z_i \right\|_{\Sigma_i}^2 \right\} \]

IMU residuals \quad Reprojection residuals

[Jung, CVPR’01] [Sterlow’04] [Bryson, ICRA’09] [Indelman, RAS’13] [Patron-Perez, IJCV’15][Leutenegger, RSS’13-IJRR’15] [Forster, RSS’15, TRO’16]
MAP: a nonlinear least squares problem

- Bayesian Theorem

\[ P(A|B) = \frac{P(B|A)P(A)}{P(B)} \]

Applied to state estimation problem:
- X: states (position, attitude, velocity, and 3D point position)
- Z: measurements (feature positions, IMU readings)

Max a Posteriori: given the observation, what is the optimal estimation of the states?

- Gaussian Property: for iid variables

\[
f(x_1,\ldots,x_k \mid \mu,\sigma^2) = \frac{1}{\sqrt{2\sigma^2\pi}} e^{-\frac{\Sigma(x_i-\mu)^2}{2\sigma^2}}\]

Maximizing the probability is equivalent to minimizing the square root sum
 MAP: a nonlinear least squares problem

SLAM as a MAP problem

\[ x_k = f(x_{k-1}) \]
\[ z_i = h(x_{i_k}, l_{i_j}) \]

\[ X = \{x_1, \ldots, x_N\}: \text{robot states} \]
\[ L = \{l_1, \ldots\}: \text{3D points} \]
\[ Z = \{z_i, \ldots, z_M\}: \text{feature positions} \]

\[
P(X, L | Z) \propto P(Z | X, L) P(X, L)
\]
\[
\propto \left( \prod_{i=1}^{M} P(z_i | X, L) \right) P(X)
\]
\[
\propto P(x_0) \left( \prod_{i=1}^{M} P(z_i | x_{i_k}, l_{i_j}) \right) \left( \prod_{k=2}^{N} P(x_k | x_{k-1}) \right)
\]

\- \( X, L \) are independent, and no prior information about \( L \)
\- Measurements are independent
\- Markov process model
MAP: a nonlinear least squares problem

- SLAM as a least squares problem

\[
P(X,L \mid Z) \propto P(x_0) \left( \prod_{i=1}^{M} P(z_i \mid x_{i_k}, l_{i_j}) \right) \left( \prod_{k=2}^{N} P(x_k \mid x_{k-1}) \right)
\]

Without the prior, applying the property of Gaussian distribution:

\[
\{X^*, L^*\} = \arg \max_{\{X,L\}} P(X,L \mid Z)
\]

\[
= \arg \min_{\{X,L\}} \left\{ \sum_{k=1}^{N} \left\| f(x_{k-1}) - x_k \right\|_{\Lambda_k}^2 + \sum_{i=1}^{M} \left\| h(x_{i_k}, l_{i_j}) - z_i \right\|_{\Sigma_i}^2 \right\}
\]

Notes:

- Normalize the residuals with the variance of process noise and measurement noise (so-called **Mahalanobis distance**).
MAP: Nonlinear optimization

- Gauss-Newton method

\[ \theta^* = \arg \min_{\theta} \sum_{i=1}^{M} \| f_i(\theta) - z_i \|^2 \]

Solve it iteratively

\[ \varepsilon^* = \arg \min_{\varepsilon} \sum_{i=1}^{M} \| f_i(\theta^s + \varepsilon) - z_i \|^2 \]

\[ \theta^{s+1} = \theta^s + \varepsilon \]

Applying first-order approximation:

\[ \varepsilon^* = \arg \min_{\varepsilon} \sum_{i=1}^{M} \| f_i(\theta^s) - z_i + J_i \varepsilon \|^2 \]

\[ = \arg \min_{\varepsilon} \sum_{i=1}^{M} \| r_i(\theta^s) + J_i \varepsilon \|^2 \]

\[ \varepsilon^* = -(J^TJ)^{-1} J^T r(\theta) \]
MAP: visual inertial formulation

- **States**
  \[
  X_R(k) = [p_{WB}[k], q_{WB}[k], v_{WB}[k], b^a[k], b^g[k]]
  \]
  \[
  X_L = [L_{W1}, L_{W2}, ..., L_{WL}]
  \]
  Combined: \[
  X = [X_R[1], X_R[2], ..., X_R[k], X_L]
  \]

- **Dynamics Jacobians**
  - IMU integration w.r.t \( x_{k-1} \)
  - Residual w.r.t. \( x_k \)

- **Measurements Jacobians** (same as filtering method)
  - Feature position w.r.t. pose
  - Feature position w.r.t. 3D coordinates
Fixed-lag smoothing: Basic Idea

- Recall MAP estimation
  \[ \varepsilon^* = -(J^T J)^{-1} J^T r(\theta) \]

  \( J^T J \) is also called the Hessian matrix.

- Hessian for full bundle adjustment: \( n \times n \), \( n \) number of all the states
  pose, velocity | landmarks

If only part of the states are of interest, can we think of a way for simplification?
Fixed-lag smoothing: Marginalization

- Schur complement

\[
M = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \quad \bar{D} = D - CA^{-1}B \quad \text{Schur complement of } A \text{ in } M \\
\bar{A} = A - BD^{-1}C \quad \text{Schur complement of } D \text{ in } M
\]

- Reduced linear system

\[
\begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}
\]

\[
\begin{pmatrix} 1 & 0 \\ -CA^{-1} & 1 \end{pmatrix} \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}
\]

\[
\begin{pmatrix} A & B \\ 0 & \bar{D} \end{pmatrix} = \begin{pmatrix} b_1 \\ \bar{b}_2 \end{pmatrix} \quad \bar{b}_2 = b_2 - CA^{-1}b_1
\]

We can then just solve for \( x_2 \), and (optionally) solve for \( x_1 \) by back substitution.
Fixed-lag smoothing: Marginalization

- Generalized Schur complement
  - Any principal submatrix: selecting \( n \) rows and \( n \) columns of the same index (i.e., select any states to marginalize)
  - Nonsingular submatrix: use generalized inverse (e.g., Moore–Penrose pseudoinverse)

- Special structure of SLAM

Marginalization causes fill-in, no longer maintaining the sparse structure.

Inverse of diagonal matrix is very efficient to calculate.
Fixed-lag smoothing: Implementation

- States and formulations are similar to MAP estimation.
- Which states to marginalize?
  - Old states: keep a window of recent frames
  - Landmarks: structureless
- Marginalizing states vs. dropping the states
  - Dropping the states: loss of information, not optimal
  - Marginalization: optimal if there is no linearization error, but introduces fill-in, causing performance penalty

Therefore, dropping states is also used to trade accuracy for speed.

Fixed-lag smoothing: OKVIS

OKVIS: Open Keyfram-based Visual-Inertial SLAM

A reference implementation of:

MAP: why it is slow

- Re-linearization
  - Need to recalculate the Jacobian for each iteration
  - But it is also an important reason why MAP is accurate

- The number of states is large
  - Will see next: fix-lag smoothing and marginalization

- Re-integration of IMU measurements
  - The integration from $k$ to $k+1$ is related to the state estimation at time $k$
  - Preintegration

Lupton, Todd, and Salah Sukkarieh. "Visual-inertial-aided navigation for high-dynamic motion in built environments without initial conditions."
MAP: IMU Preintegration

Standard:
Evaluate error in global frame:

\[
\begin{align*}
\mathbf{e}_R &= \hat{\mathbf{R}}(\bar{\omega}, \bar{R}_{k-1})^T \mathbf{R}_k \\
\mathbf{e}_V &= \hat{\mathbf{v}}(\bar{\omega}, \bar{a}, \mathbf{v}_{k-1}) - \mathbf{v}_k \\
\mathbf{e}_p &= \hat{\mathbf{p}}(\bar{\omega}, \bar{a}, \mathbf{p}_{k-1}) - \mathbf{p}_k
\end{align*}
\]

Preintegration:
Evaluate relative errors:

\[
\begin{align*}
\mathbf{e}_R &= \Delta \hat{\mathbf{R}}^T \Delta \mathbf{R} \\
\mathbf{e}_V &= \Delta \hat{\mathbf{v}} - \Delta \mathbf{v} \\
\mathbf{e}_p &= \Delta \hat{\mathbf{p}} - \Delta \mathbf{p}
\end{align*}
\]

Preintegration of IMU deltas possible with no initial condition required.

Repeat integration when previous state changes!
Solves the same optimization problem but:

- Keeps all the frames (from the start of the trajectory)
- To make the optimization efficient
  - it makes the graph sparser using keyframes
  - pre-integrates the IMU data between keyframes
- Optimization salved using factor graphs (GTSAM)
  - Very fast because it only optimizes the poses which are affected by a new observation

\[
\{ X^*, L^* \} = \arg \max_{\{ X, L \}} P( X, L \mid Z) \\
= \arg \min_{\{ X, L \}} \left\{ \sum_{k=1}^{N} \| f(x_{k-1}) - x_k \|_{\Lambda_k}^2 + \sum_{i=1}^{M} \| h(x_{i_k}, l_{ij}) - z_i \|_{\Sigma_i}^2 \right\}
\]

IMU residuals \quad Reprojection residuals

SVO + IMU Preintegration

IMU Preintegration on Manifold for Efficient Visual-Inertial Maximum-a-Posteriori Estimation

Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza

SVO + IMU Preintegration

Accuracy: 0.1% of the travel distance

Visual-Inertial Fusion: further reading and code

- **Closed form solution:**
  - for 6DOF motion both $s$ and $v_0$ can be determined **1 feature observation and at least 3 views** [Martinelli, TRO’12, IJCV’14, RAL’16]
  - Can be used to **initialize filters and smoothers**

- **Filters:** *update only last state → fast if number of features is low (~20)*
  - [Mourikis, ICRA’07, CVPR’08], [Jones, IJRR’11] [Kottas, ISER’12][Bloesch, IROS’15] [Wu et al., RSS’15], [Hesch, IJRR’14], [Weiss, JFR’13]
  - **Open source:** ROVIO [Bloesch, IROS’15, IJRR’17], MSCKF [Mourikis, ICRA’07] (i.e., Google Tango)

- **Fixed-lag smoothers:** *update a window of states → slower but more accurate*
  - [Mourikis, CVPR’08] [Sibley, IJRR’10], [Dong, ICRA’11], [Leutenegger, RSS’13-IJRR’15]
  - **Open source:** OKVIS [Leutenegger, RSS’13-IJRR’15]

- **Full-smoothing methods:** *update entire history of states → slower but more accurate*
  - [Jung, CVPR’01] [Sterlow’04] [Bryson, ICRA’09] [Indelman, RAS’13] [Patron-Perez, IJCV’15] [Forster, RSS’15, TRO’16]
  - **Open source:** SVO+IMU [Forster, TRO’17]
Open Problem: consistency

- Filters
  - Linearization around different values of the same variable may lead to error

- Smoothing methods
  - May get stuck in local minima
Outline

- Introduction
- IMU model and Camera-IMU system
- Different paradigms
  - Closed-form solution
  - Filtering approaches
    - Maximum a posteriori estimation (non linear optimizers)
      - Fixed-lag Smoothing (aka sliding window estimators)
      - Full smoothing methods
- Camera-IMU extrinsic calibration and Synchronization
Camera-IMU calibration

- **Goal**: estimate the rigid-body transformation $T_{BC}$ and delay $t_d$ between a camera and an IMU rigidly attached. Assume that the camera has already been intrinsically calibrated.

- **Data**:
  - Image points of detected calibration pattern (checkerboard).
  - IMU measurements: accelerometer $\{a_k\}$ and gyroscope $\{\omega_k\}$.

---

Camera-IMU calibration - Example

- **Data acquisition**: Move the sensor in front of a static calibration pattern, exciting all degrees of freedom, and trying to make smooth motions.

Images. Extract points from calibration pattern.
Camera-IMU calibration

- **Approach:** Minimize a cost function (Furgale’13):

\[
J(\theta) := J_{\text{feat}} + J_{\text{acc}} + J_{\text{gyro}} + J_{\text{bias}_{\text{acc}}} + J_{\text{bias}_{\text{gyro}}}
\]

(Feature reprojection Error)

\[
\sum_k (a_{\text{IMU}}(t_k - t_d) - a_{\text{Cam}}(t_k))^2 \sum_k (\omega_{\text{IMU}}(t_k - t_d) - \omega_{\text{Cam}}(t_k))^2 \int \left\| \frac{db_{\text{acc}}}{dt}(u) \right\|^2 du \int \left\| \frac{db_{\text{gyro}}}{dt}(u) \right\|^2 du
\]

- **Unknowns:** \( T_{BC}, t_d, g_w, T_{WB}(t), b_{\text{acc}}(t), b_{\text{gyro}}(t) \)
  - Gravity \( g_w \), 6-DOF trajectory of the IMU \( T_{WB}(t) \), 3-DOF biases of the IMU \( b_{\text{acc}}(t), b_{\text{gyro}}(t) \)

- **Continuous-time modelling using splines** for \( T_{WB}(t), b_{\text{acc}}(t), ... \)

- **Numerical solver:** Levenberg-Marquardt (i.e., Gauss-Newton).

Camera-IMU calibration - Example

- **Software solution**: *Kalibr* (Furgale’13).
  - Generates a **report** after optimizing the cost function.

**Residuals:**
- Reprojection error [px]: 0.0976 ± 0.051
- Gyroscope error [rad/s]: 0.0167 ± 0.009
- Accelerometer error [m/s^2]: 0.0595 ± 0.031

**Transformation** $T_{ci}$ (imu to cam):

\[
\begin{bmatrix}
0.99995526 & -0.00934911 & -0.00143776 & 0.00008436 \\
0.00936458 & 0.99989388 & 0.01115983 & 0.00197427 \\
0.00133327 & -0.0111728 & 0.99993669 & -0.05054946 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

**Time shift** (delay $d$)
- cam0 to imu0: [s] ($t_{imu} = t_{cam} + shift$)
  
  0.00270636270255

**Gravity vector** in target coords: [m/s^2]

\[
\begin{bmatrix}
0.04170719 & -0.01000423 & -9.80645621
\end{bmatrix}
\]