Title: Visual Navigation (19.06.2012)
Date: Tue Jun 19 10:16:39 CEST 2012
Duration: 86:43 min
Pages: 85

Visual Navigation for Flying Robots

Bundle Adjustment and Stereo Correspondence

Dr. Jürgen Sturm

- This Thursday
- Don’t forget to put title, team name, team members on first slide
- Pitch has to fit in 5 minutes (+5 minutes discussion)
- 9 x (5+5) = 90 minutes
- Recommendation: use 3-5 slides
Agenda for Today

- Map optimization
  - Graph SLAM
  - Bundle adjustment
- Depth reconstruction
  - Laser triangulation
  - Structured light (Kinect)
  - Stereo cameras

Remember: 3D Transformations

- Representation as a homogeneous matrix
  \[
  M = \begin{pmatrix} R & t \\ 0^T & 1 \end{pmatrix} \in \text{SE}(3) \subset \mathbb{R}^{4 \times 4}
  \]
  - Pro: easy to concatenate and invert
  - Con: not minimal

- Representation as a twist coordinates
  \[
  \xi = (v_x, v_y, v_z, \omega_x, \omega_y, \omega_z)^\top \in \mathbb{R}^6
  \]
  - Pro: minimal
  - Con: need to convert to matrix for concatenation and inversion

Remember: Rodrigues’ formula

- Given: Twist coordinates
  \[
  \xi = (\omega^\top, v^\top)^\top = (\omega_x, \omega_y, \omega_z, v_x, v_y, v_z)^\top = (t\bar{\omega}^\top, v^\top)^\top
  \]
  with \( \|\bar{\omega}\| = 1, t = \|\omega\| \)

- Return: Homogeneous transformation
  \[
  R = I + [\bar{\omega}]_x \sin(t) + [\bar{\omega}]_x^2 (1 - \cos t)
  \]
  \[
  t = (I - R)[\bar{\omega}]_x v + \bar{\omega}^\top vt
  \]
  \[
  M = \begin{pmatrix} R & t \\ 0^\top & 1 \end{pmatrix}
  \]
  \[
  \xi = \log M
  \]
  \[
  M = \exp \xi
  \]
  \[
  \exp[\xi]^\wedge
  \]
  alternative notation:
Notation

- Camera poses in a minimal representation (e.g., twists)
  \[ c_1, c_2, \ldots, c_n \]
- ... as transformation matrices
  \[ M_1, M_2, \ldots, M_n \]
- ... as rotation matrices and translation vectors
  \[ (R_1, t_1), (R_2, t_2), \ldots, (R_n, t_n) \]

Incremental Motion Estimation

- Idea: Estimate camera motion from frame to frame

Loop Closures

- Idea: Estimate camera motion from frame to frame
- Problem:
  - Estimates are inherently noisy
  - Error accumulates over time → drift
Incremental Motion Estimation

- **Idea:** Estimate camera motion from frame to frame

Graph SLAM

[Olson et al., 2006]

- Use a graph to represent the model
- Every **node** in the graph corresponds to a pose of the robot during mapping
- Every **edge** between two nodes corresponds to a spatial constraint between them
- **Graph-based SLAM:** Build the graph and find the robot poses that **minimize the error** introduced by the constraints

Loop Closures

- **Solution:** Use loop-closures to minimize the drift / minimize the error over all constraints

Example: Graph SLAM on Intel Dataset
Example: Graph SLAM on Intel Dataset

Graph SLAM Architecture

Focus of today

- Interleaving process of front-end and back-end
- A consistent map helps to determine new constraints by reducing the search space

Problem Definition

- **Given:** Set of observations \( z_{ij} \in \mathbb{R}^6 \)

- **Wanted:** Set of camera poses \( c_1, \ldots, c_n \in \mathbb{R}^6 \)
  
  \( \Rightarrow \) State vector \( x = (c_1^T, \ldots, c_n^T)^T \in \mathbb{R}^{6n} \)

Map Error

- Real observation \( z_{ij} \)
- Expected observation \( \bar{z}_{ij} = c_j \otimes c_i \)

- Difference between observation and expectation \( e_{ij} = z_{ij} \otimes \bar{z}_{ij} \)

- Given the correct map, this difference is the result of sensor noise...
Map Error

- Real observation \( z_{ij} \)
- Expected observation \( \bar{z}_{ij} = c_j \oplus c_i \)

- Difference between observation and expectation
  \[ e_{ij} = z_{ij} \ominus \bar{z}_{ij} \]

- Given the correct map, this difference is the result of sensor noise...

Incremental Motion Estimation

- **Idea:** Estimate camera motion from frame to frame

Error Function

- Map error (over all observations)
  \[ f(x) = \sum_{ij} f_{ij}(x) = \sum_{ij} e_{ij}(x)^T \Sigma_{ij}^{-1} e_{ij}(x) \]

- **Minimize this error** by optimizing the camera poses
  \[ x^* = \arg \min_x \sum_{ij} e_{ij}(x)^T \Sigma_{ij}^{-1} e_{ij}(x) \]

- How can we solve this optimization problem?
### Non-Linear Minimization

**Gauss-Newton Method:**
1. Linearize the error function
2. Compute its derivative
3. Set the derivative to zero
4. Solve the linear system
5. Iterate this procedure until convergence

---

### Step 1: Linearize the Error Function

- **Error function**
  
  \[ f(x) = \sum_{ij} f_{ij}(x) = \sum_{ij} e_{ij}(x)^T \Sigma^{-1}_{ij} e_{ij}(x) \]

- **Evaluate the error function around the initial guess**
  
  \[ f(x + \Delta x) = \sum_{ij} e_{ij}(x + \Delta x)^T \Sigma^{-1}_{ij} e_{ij}(x + \Delta x) \]

  Let's derive this term first...
Linearize the Error Function

- Approximate the error function around an initial guess $\mathbf{x}$ using Taylor expansion

\[ e_{ij}(\mathbf{x} + \Delta \mathbf{x}) \simeq e_{ij}(\mathbf{x}) + J_{ij} \Delta \mathbf{x} \]

with

\[ J_{ij}(\mathbf{x}) = \begin{pmatrix} \frac{\partial e_{ij}(\mathbf{x})}{\partial e_1}(\mathbf{x}) & \frac{\partial e_{ij}(\mathbf{x})}{\partial e_2}(\mathbf{x}) & \cdots & \frac{\partial e_{ij}(\mathbf{x})}{\partial e_n}(\mathbf{x}) \end{pmatrix} \]

Linearize $f(\mathbf{x}) = \sum_{ij} e_{ij}(\mathbf{x}) \Sigma_{ij}^{-1} e_{ij}(\mathbf{x})$

\[ \simeq \mathbf{c} + 2 \mathbf{b}^T \Delta \mathbf{x} + \Delta \mathbf{x}^T \mathbf{H} \Delta \mathbf{x} \]

with $\mathbf{b}^T = \sum_{ij} e_{ij}^T \Sigma_{ij}^{-1} J_{ij}$

\[ \mathbf{H} = \sum_{ij} J_{ij}^T \Sigma_{ij}^{-1} J_{ij} \]

- What is the structure of $\mathbf{b}^T$ and $\mathbf{H}$?
  (Remember: all $J_{ij}$'s are sparse)

Illustration of the Structure

$\mathbf{b}_{ij}^T = e_{ij}^T \Sigma_{ij}^{-1} J_{ij}$

Non-zero only at $c_i$ and $c_j$

$\mathbf{b}= \sum_{ij} \mathbf{b}_{ij}$

$\mathbf{b}$: dense vector

$\mathbf{H}= \sum_{ij} H_{ij}$

$\mathbf{H}$: sparse block structure with main diagonal
(Linear) Least Squares Minimization

1. Linearize error function
   \[ f(x + \Delta x) \approx c + 2b^T \Delta x + \Delta x^T H \Delta x \]
2. Compute the derivative
   \[ \frac{df(x + \Delta x)}{d\Delta x} = 2b + 2H\Delta x \]
3. Set derivative to zero
   \[ H\Delta x = -b \]
4. Solve this linear system of equations, e.g.,
   \[ \Delta x = -H^{-1}b \]

Gauss-Newton Method

**Problem:** \( f(x) \) is non-linear!

**Algorithm:** Repeat until convergence

1. Compute the terms of the linear system
   \[ b^T = \sum_{ij} e_{ij}^T \Sigma_{ij}^{-1} J_{ij} \]
   \[ H = \sum_{ij} J_{ij}^T \Sigma_{ij}^{-1} J_{ij} \]
2. Solve the linear system to get new increment
   \[ H\Delta x = -b \]
3. Update previous estimate
   \[ x \leftarrow x + \Delta x \]

Sparsity of the Hessian

- The Hessian is
  - positive semi-definite
  - symmetric
  - sparse
- This allows the use of efficient solvers
  - Sparse Cholesky decomposition (~100M matrix elements)
  - Preconditioned conjugate gradients (~1.000 matrix elements)
  - ... many others
Sparsity of the Hessian

- The Hessian is
  - positive semi-definit
  - symmetric
  - sparse

- This allows the use of efficient solvers
  - Sparse Cholesky decomposition (~100M matrix elements)
  - Preconditioned conjugate gradients (~1.000 matrix elements)
  - ... many others

Example in 1D

- Error
  \[ e_{12} = z_{12} - \bar{z}_{12} = z_{12} - (c_2 - c_1) = 1 - (0 - 0) = 1 \]

- Jacobian
  \[ J_{12} = \begin{pmatrix} \frac{\partial e_{12}}{\partial c_1} \\ \frac{\partial e_{12}}{\partial c_2} \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix} \]

- Build linear system of equations
  \[ b^T = e_{12}^T \Sigma^{-1} e_{12} = \begin{pmatrix} 2 \\ -2 \end{pmatrix} \]
  \[ H = J_{12}^T \Sigma^{-1} J_{12} = \begin{pmatrix} 2 & -2 \\ -2 & 2 \end{pmatrix} \]

- Solve the system
  \[ \Delta x = -H^{-1} b \quad \text{but} \quad \det H = 0 \ ??? \]

What Went Wrong?

- The constraint only specifies a relative constraint between two nodes
- Any poses for the nodes would be fine as long as their relative coordinates fit
- One node needs to be fixed
  - Option 1: Remove one row/column corresponding to the fixed pose
  - Option 2: Add to \( H \), \( b \) a linear constraint \( 1 \cdot \Delta c_1 = 0 \)
  - Option 3: Add the identity matrix to \( H \) (Levenberg-Marquardt)
Fixing One Node

- The constraint only specifies a relative constraint between two nodes.
- Any poses for the nodes would be fine as long as their relative coordinates fit.
- **One node needs to be fixed (here: Option 2)**

\[ H = \begin{pmatrix} 2 & -2 \\ -2 & 2 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \]

additional constraint that sets \( \Delta c_1 = 0 \)

\[ \Delta x = -H^{-1}b \]

\[ \Delta x = (0 \ 1) \top \]

Levenberg-Marquardt Algorithm

- **Idea:** Add a damping factor

\[ (H + \lambda I)\Delta x = -b \]

\[ (J^\top J + \lambda I)\Delta x = -J^\top e \]

- What is the effect of this damping factor?
  - Small \( \lambda \)?
  - Large \( \lambda \)?

Non-Linear Minimization

- One of the state-of-the-art solution to compute the maximum likelihood estimate.
- Various open-source implementations available:
  - g2o [Kuemmerle et al., 2011]
  - sba [Lourakis and Argyros, 2009]
  - iSAM [Kaess et al., 2008]
- Other extensions:
  - Robust error functions
  - Alternative parameterizations

Bundle Adjustment

- **Graph SLAM:** Optimize (only) the camera poses

\[ x = (c_1^\top, \ldots, c_n^\top)^\top \in \mathbb{R}^{6n} \]

- **Bundle Adjustment:** Optimize both 6DOF camera poses and 3D (feature) points

\[ x = \left( c_1^\top, \ldots, c_n^\top, p_1^\top, \ldots, p_m^\top \right)^\top \in \mathbb{R}^{6n+3m} \]

- Typically \( m \gg n \) (why?)
**Error Function**

- Camera pose \( c_i \in \mathbb{R}^6 \)
- Feature point \( p_j \in \mathbb{R}^3 \)
- Observed feature location \( z_{ij} \in \mathbb{R}^2 \)
- Expected feature location
  \[
g(c_i, p_j) = R_i^T (t_i - p_j)
\]
  \[
h(c_i, p_j) = g_{x,y}(c_i, p_j) / g_z(c_i, p_j)
\]

**Illustration of the Structure**

- Each camera sees several points
- Each point is seen by several cameras
- Cameras are independent of each other (given the points), same for the points

**Primary Structure**

- Characteristic structure
  \[
  \begin{pmatrix}
  J_c^T & J_c^T & J_p^T & J_p^T \\
  J_p^T & J_c^T & J_p^T & J_p^T \\
  
  \end{pmatrix}
  \begin{pmatrix}
  \Delta c \\
  \Delta p \\
  
  \end{pmatrix}
  =
  \begin{pmatrix}
  -J_c^T e_c \\
  -J_p^T e_p \\
  
  \end{pmatrix}
  \]

  \[
  \begin{pmatrix}
  H_{cc} & H_{cp} \\
  H_{pc} & H_{pp} \\
  
  \end{pmatrix}
  \begin{pmatrix}
  \Delta c \\
  \Delta p \\
  
  \end{pmatrix}
  =
  \begin{pmatrix}
  -J_c^T e_c \\
  -J_p^T e_p \\
  
  \end{pmatrix}
  \]
Primary Structure

- **Insight:** $H_{cc}$ and $H_{pp}$ are block-diagonal (because each constraint depends only on one camera and one point)

  $$
  \begin{pmatrix}
  \Delta c \\
  \Delta p 
  \end{pmatrix}
  =
  \begin{pmatrix}
  -J_{c}^T e_c \\
  -J_{p}^T e_p 
  \end{pmatrix}
  $$

- This can be efficiently solved using the Schur Complement

Schur Complement

- **Given:** Linear system

  $$
  \begin{pmatrix}
  A & B \\
  C & D
  \end{pmatrix}
  \begin{pmatrix}
  x \\
  y
  \end{pmatrix}
  =
  \begin{pmatrix}
  a \\
  b
  \end{pmatrix}
  $$

- If $D$ is invertible, then (using Gauss elimination)

  $$(A - BD^{-1}C)x = a - BD^{-1}b$$

  $$y = D^{-1}(b - Cx)$$

- **Reduced complexity,** i.e., invert one $p \times p$ and $p \times p$ matrix instead of one $(p + q) \times (p + q)$ matrix

Example Hessian (Lourakis and Argyros, 2009)

$$
H = \begin{pmatrix}
\vdots & \vdots & \vdots \\
\vdots & \ddots & \vdots \\
\vdots & \vdots & \ddots
\end{pmatrix}
$$
From Sparse Maps to Dense Maps

- So far, we only looked at sparse 3D maps
  - We know where the (sparse) cameras are
  - We know where the (sparse) 3D feature points are
- How can we turn these models into volumetric 3D models?

Human Stereo Vision

- Given a point in the left image, where can the corresponding point be in the right image?

Stereo Correspondence Constraints

- Today: Estimation of depth dense images (stereo cameras, laser triangulation, structured light/Kinect)
- Next week: Dense map representations and data fusion
Reminder: Epipolar Geometry

- A point in one image “generates” a line in another image (called the **epipolar line**)
- Epipolar constraint $\hat{x}_2^T E \hat{x}_1 = 0$

**Example: Converging Cameras**

**Example: Parallel Cameras**

- This is useful because it reduces the correspondence problem to a 1D search along an epipolar line
**Rectification**

- In practice, it is convenient if the image scanlines (rows) are the epipolar lines
- Reproject image planes onto a common plane parallel to the baseline (two 3x3 homographies)
- Afterwards pixel motion is horizontal

**Basic Stereo Algorithm**

- For each pixel in the left image
  - Compare with every pixel on the same epipolar line in the right image
  - Pick pixel with minimum matching cost (noisy)
  - Better: match small blocks/patches (SSD, SAD, NCC)

**Block Matching Algorithm**

**Input:** Two images and camera calibrations

**Output:** Disparity (or depth) image

**Algorithm:**

1. Geometry correction (undistortion and rectification)
2. Matching cost computation along search window
3. Extrema extraction (at sub-pixel accuracy)
4. Post-filtering (clean up noise)
Example

- Input

- Output

What is the Influence of the Block Size?

- Common choices are 5x5 .. 11x11
- Smaller neighborhood: more details
- Larger neighborhood: less noise
- Suppress pixels with low confidence (e.g., check ratio best match vs. 2nd best match)

Example: PR2 Robot with Projected Texture Stereo

- Block matching typically fails in regions with low texture
  - Global optimization/regularization (speciality of our research group)
  - Additional texture projection

Wide-angle stereo pair

Pattern projector

Narrow-angle stereo pair

5 MP high-res camera
Laser Triangulation

Idea:
- Well-defined light pattern (e.g., point or line) projected on scene
- Observed by a line/matrix camera or a position-sensitive device (PSD)
- Simple triangulation to compute distance

- Function principle
  - Depth triangulation \( z = \frac{fL}{d} \) (note: same for stereo disparities)

Example: Neato XV-11
- Specs: 360deg, 10Hz, 30 USD

How Does the Data Look Like?
Laser Triangulation

- Stripe laser + 2D camera
- Often used on conveyer belts (volume sensing)
- Large baseline gives better depth resolution but more occlusions → use two cameras

Structured Light

- Multiple stripes / 2D pattern
- Data association more difficult

- Coding schemes
  - Temporal: Coded light

- Coding schemes
  - Temporal: Coded light
  - Wavelength: Color
  - Spatial: Pattern (e.g., diffraction patterns)
Sensor Principle of Kinect

- Kinect projects a diffraction pattern (speckles) in near-infrared light
- CMOS IR camera observes the scene

Example Data

- Kinect provides color (RGB) and depth (D) video
- This allows for novel approaches for (robot) perception

Sensor Principle of Kinect

- Pattern is memorized at a known depth
- For each pixel in the IR image
  - Extract 9x9 template from memorized pattern
  - Correlate with current IR image over 64 pixels and search for the maximum
  - Interpolate maximum to obtain sub-pixel accuracy (1/8 pixel)
- Calculate depth by triangulation
Technical Specs

- Infrared camera has 640x480 @ 30 Hz
  - Depth correlation runs on FPGA
  - 11-bit depth image
  - 0.8m – 5m range
  - Depth sensing does not work in direct sunlight (why?)
- RGB camera has 640x480 @ 30 Hz
  - Bayer color filter
- Four 16-bit microphones with DSP for beam forming @ 16kHz
- Requires 12V (for motor), weighs 500 grams
- Human pose recognition runs on Xbox CPU and uses only 10-15% processing power @30 Hz

Impact of the Kinect Sensor

- Sold >18M units, >8M in first 60 days (Guinness: “fastest selling consumer electronics device”)
- Has become a “standard” sensor in robotics

History

- 2005: Developed by PrimeSense (Israel)
- 2006: Offer to Nintendo and Microsoft, both companies declined
- 2007: Alex Kidman becomes new incubation director at Microsoft, decides to explore PrimeSense device. Johnny Lee assembles a team to investigate technology and develop game concepts
- 2008: The group around Prof. Andrew Blake and Jamie Shotton (Microsoft Research) develops pose recognition
- 2009: The group around Prof. Dieter Fox (Intel Labs / Univ. of Washington) works on RGB-D mapping and RGB-D object recognition
- Nov 4, 2010: Official market launch
- Nov 10, 2010: First open-source driver available
- 2011: First programming competitions (ROS 3D, PrimeSense), First workshops (RSS, Euron)
- 2012: First special Issues (JVIC, T-SMC)

Kinect: Applications
Open Research Questions

- How can RGB-D sensing facilitate in solving hard perception problems in robotics?
  - Interest points and feature descriptors?
  - Simultaneous localization and mapping?
  - Collision avoidance and visual navigation?
  - Object recognition and localization?
  - Human-robot interaction?
  - Semantic scene interpretation?