Visual Navigation for Flying Robots

Bundle Adjustment and Stereo Correspondence

Dr. Jürgen Sturm

Project Proposal Presentations

- 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

$$M = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} \in \mathrm{SE}(3) \subset \mathbb{R}^{4\times 4}$$

Pro: easy to concatenate and invert
Con: not minimal

- Representation as a twist coordinates

$$\boldsymbol{\xi} = \begin{pmatrix} v_x & v_y & v_z & \omega_x & \omega_y & \omega_z \end{pmatrix}^\top \in \mathbf{R}^6$$

Pro: minimal
Con: need to convert to matrix for concatenation and inversion

Remember: 3D Transformations

- From twist coordinates to twist

$$\hat{\boldsymbol{\xi}} = \begin{pmatrix} 0 & -\omega_z & \omega_y & v_x \\ \omega_z & 0 & -\omega_x & v_y \\ -\omega_y & \omega_x & 0 & v_z \\ 0 & 0 & 0 & 0 \end{pmatrix} \in \mathrm{se}(3)$$

- Exponential map between se(3) and SE(3)

$$M = \exp \hat{\boldsymbol{\xi}} \qquad \hat{\boldsymbol{\xi}} = \log M$$

alternative notation: $M = \exp[\boldsymbol{\xi}]^\wedge \qquad \boldsymbol{\xi} = [\log M]^\vee$

Remember: Rodrigues' formula

- **Given:** Twist coordinates

$$\boldsymbol{\xi} = (\boldsymbol{\omega}^\top, \boldsymbol{v}^\top)^\top = (\omega_x, \omega_y, \omega_z, v_x, v_y, v_z)^\top$$

$$= (t\bar{\boldsymbol{\omega}}^\top, \boldsymbol{v}^\top)^\top \text{ with } \|\bar{\boldsymbol{\omega}}\| = 1, t = \|\boldsymbol{\omega}\|$$

- **Return:** Homogeneous transformation

$$R = I + [\bar{\boldsymbol{\omega}}]_\times \sin(t) + [\bar{\boldsymbol{\omega}}]_\times^2 (1 - \cos t)$$

$$\mathbf{t} = (I - R)[\bar{\boldsymbol{\omega}}]_\times \boldsymbol{v} + \bar{\boldsymbol{\omega}}\bar{\boldsymbol{\omega}}^\top \boldsymbol{v} t$$

$$M = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix}$$

Notation

- Camera poses in a minimal representation (e.g., twists)

$$\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_n$$

- … as transformation matrices

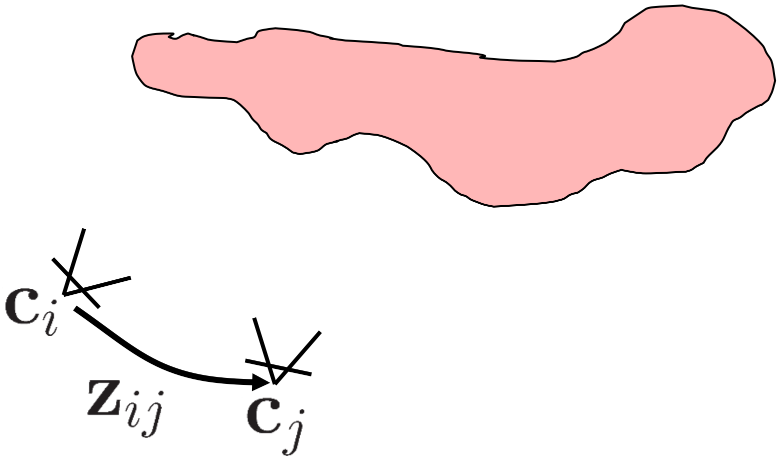
$$M_1, M_2, \ldots, M_n$$

- … as rotation matrices and translation vectors

$$(R_1, \mathbf{t}_1), (R_2, \mathbf{t}_2), \ldots, (R_n, \mathbf{t}_n)$$

Incremental Motion Estimation

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

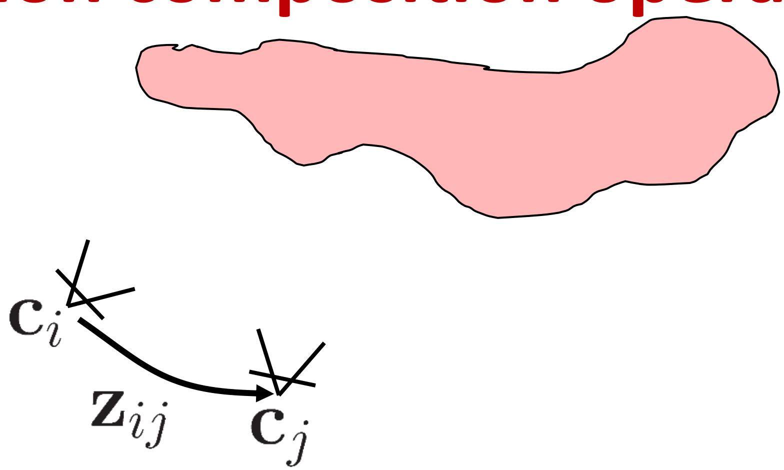


Incremental Motion Estimation

- **Idea:** Estimate camera motion from frame to frame
- **Motion concatenation (for twists)**

$$\hat{\mathbf{c}}_j = \log\left(\exp \hat{\mathbf{c}}_i \exp \hat{\mathbf{z}}_{ij}\right)$$

- **Motion composition operator (in general)**

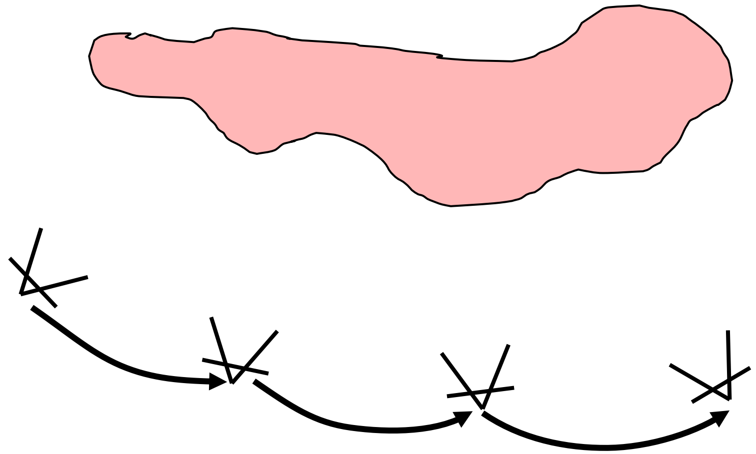


$$\mathbf{c}_j = \mathbf{c}_i \oplus \mathbf{z}_{ij}$$

$$\mathbf{z}_{ij} = \mathbf{c}_j \ominus \mathbf{c}_i$$

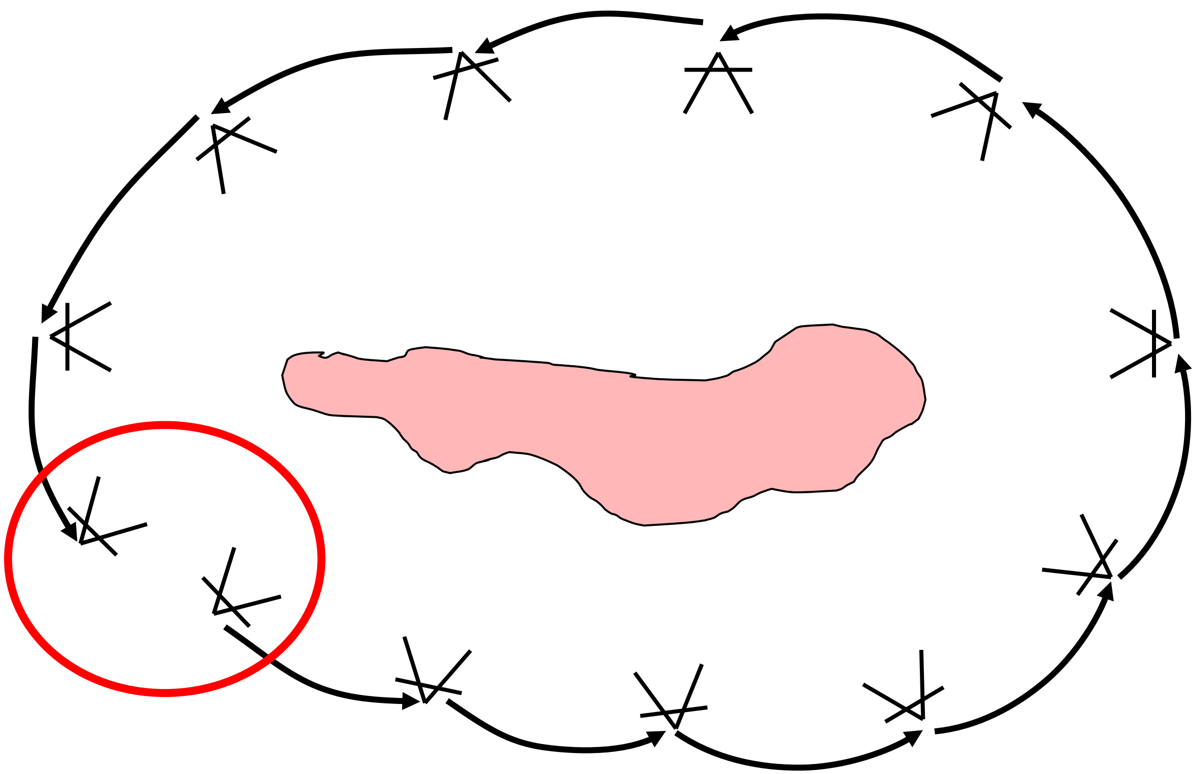
Incremental Motion Estimation

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



Incremental Motion Estimation

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

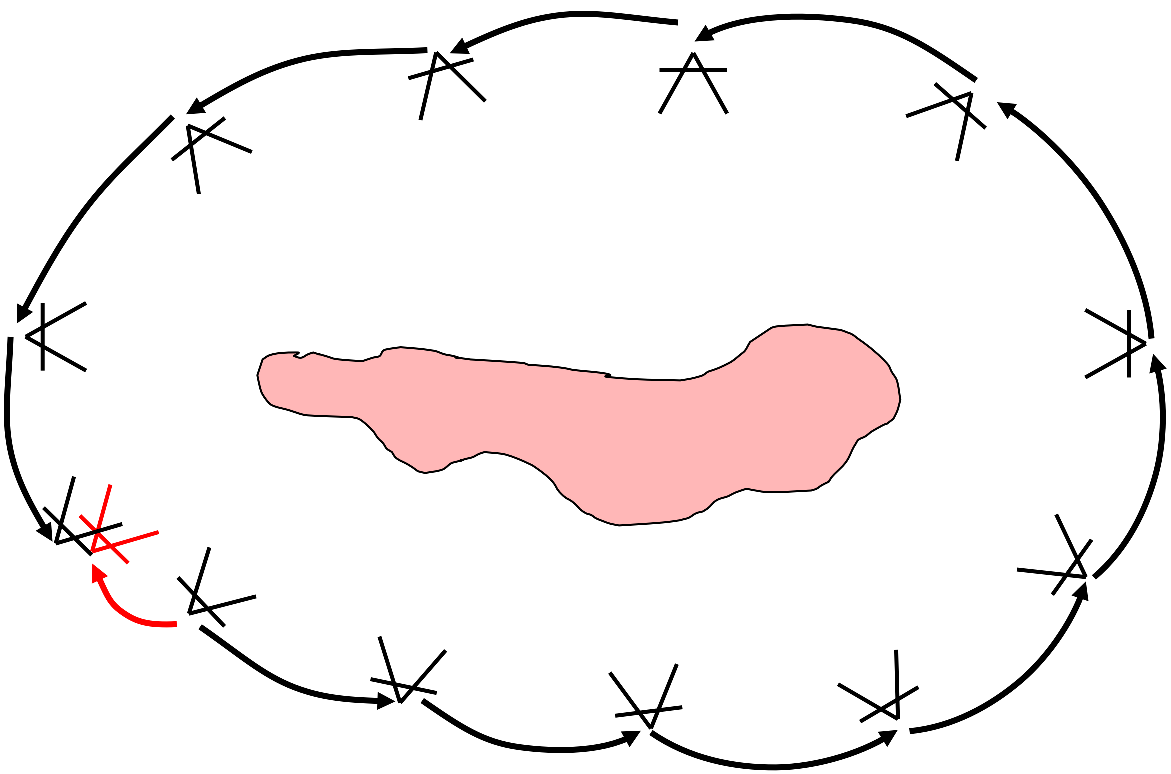


Loop Closures

- **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

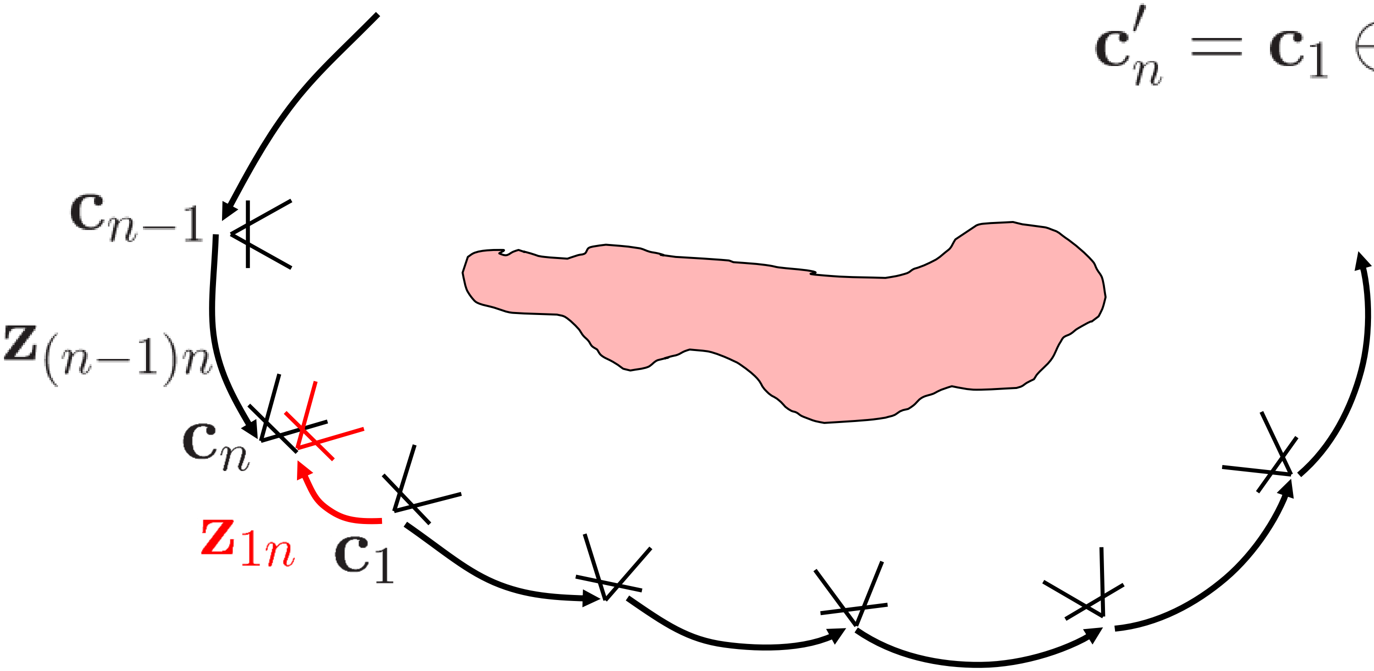


Incremental Motion Estimation

- **Idea:** Estimate camera motion from frame to frame
- Two ways to compute $\mathbf{c}_n$:

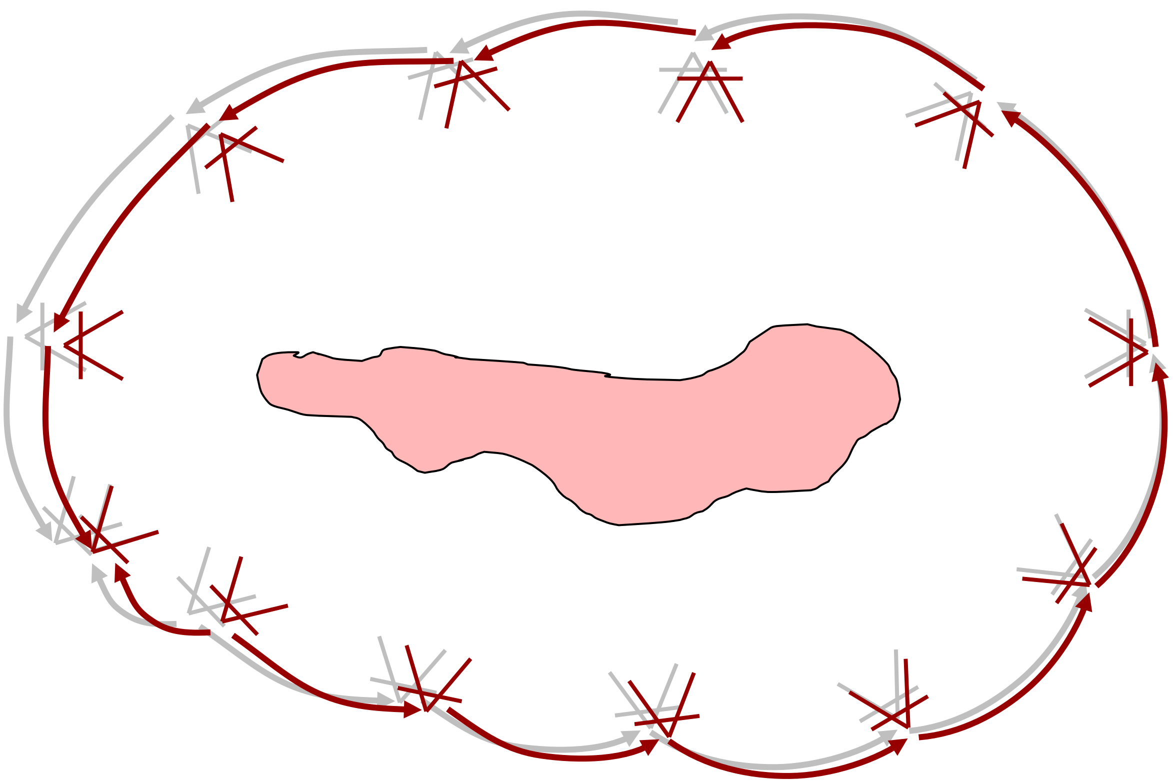
$$\mathbf{c}_n = \mathbf{c}_{n-1} \oplus \mathbf{z}_{(n-1)n}$$

$$\mathbf{c}'_n = \mathbf{c}_1 \oplus \mathbf{z}_{1n}$$



Loop Closures

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

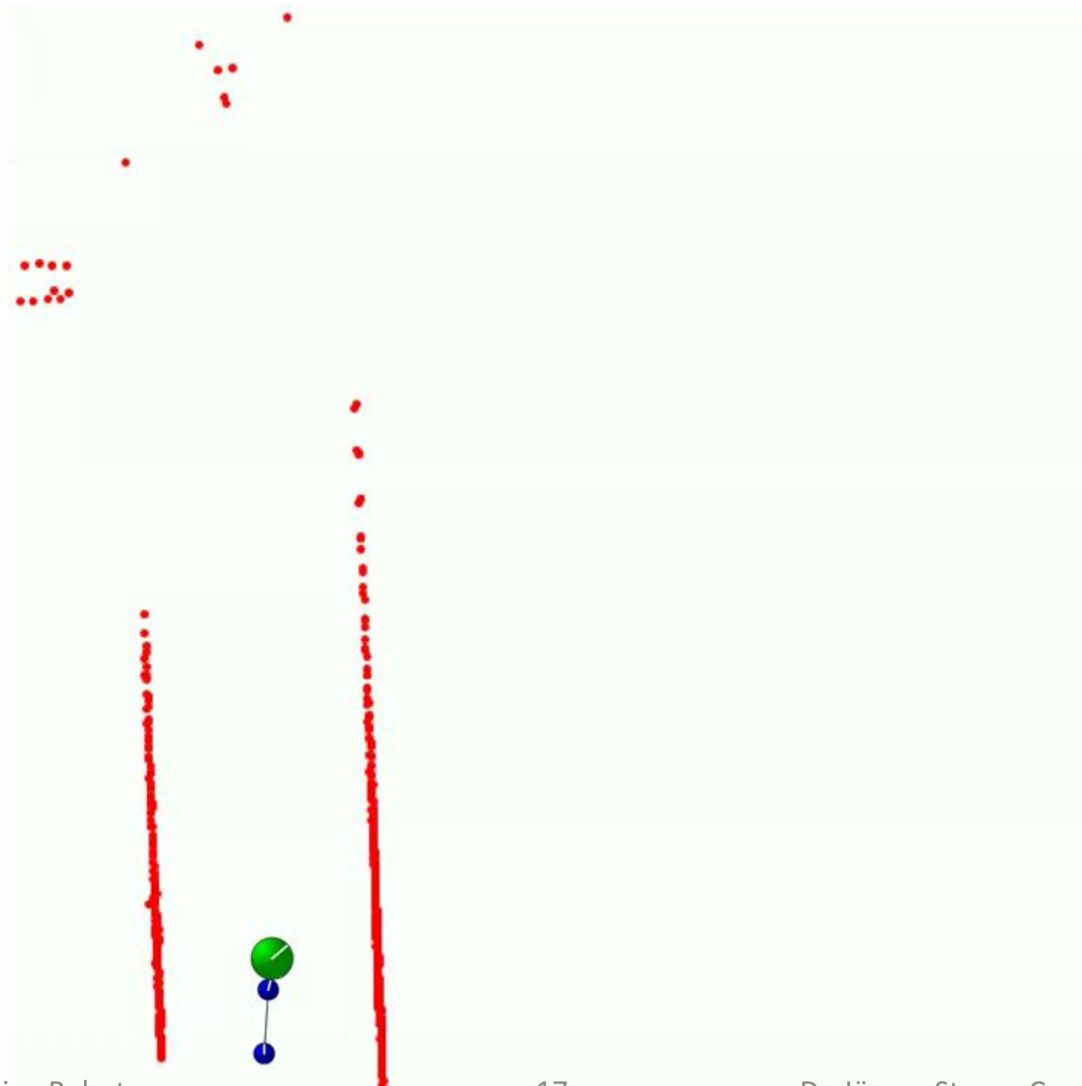


Graph SLAM

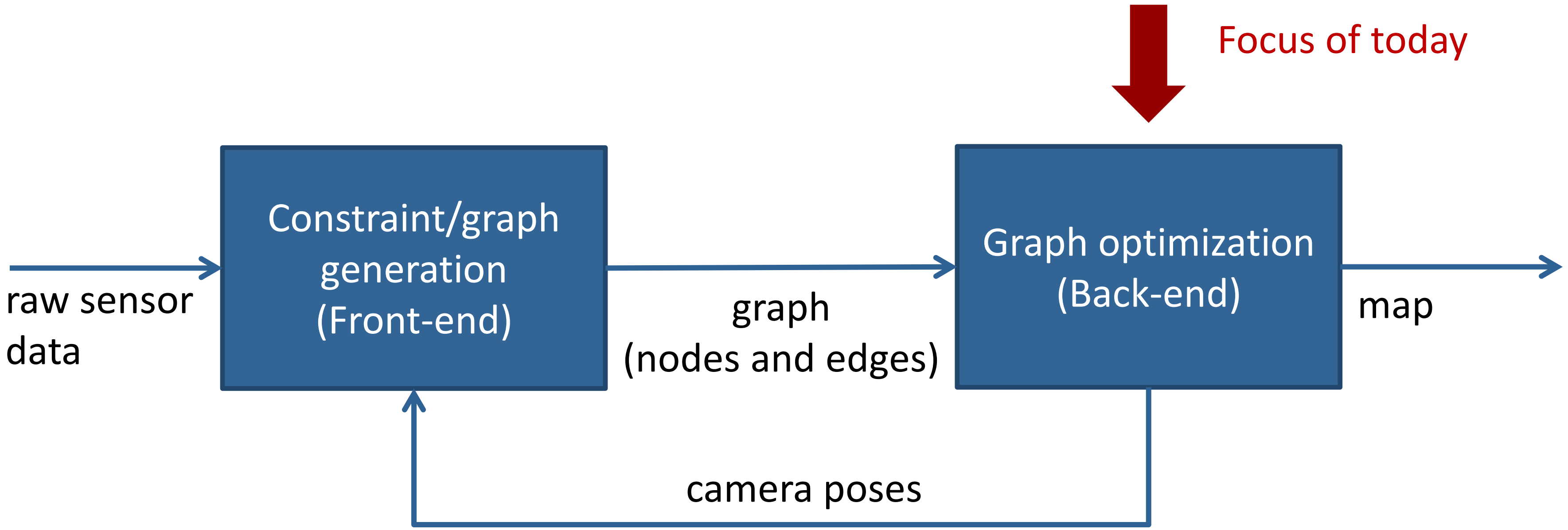
[Thrun and Montemerlo, 2006; Olson et al., 2006]

- 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

Example: Graph SLAM on Intel Dataset



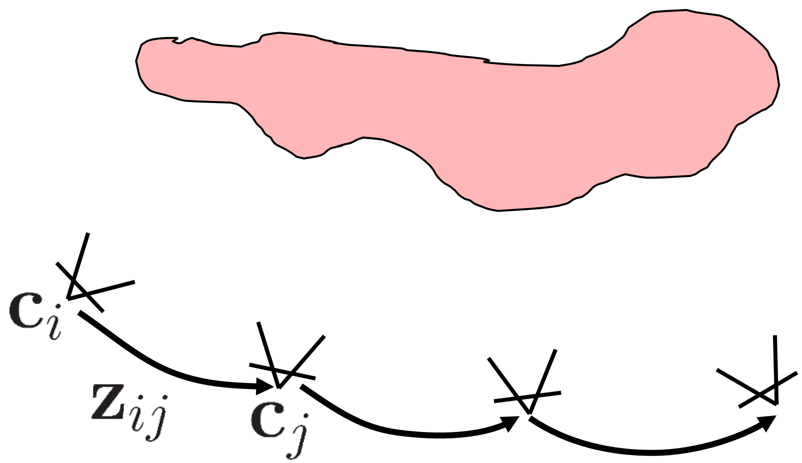
Graph SLAM Architecture



- 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 $\mathbf{z}_{ij} \in \mathbb{R}^6$
- **Wanted:** Set of camera poses $\mathbf{c}_1, \ldots, \mathbf{c}_n \in \mathbb{R}^6$
  ➔ State vector $\mathbf{x} = (\mathbf{c}_1^\top, \ldots, \mathbf{c}_n^\top)^\top \in \mathbb{R}^{6n}$



Map Error

- Real observation $\mathbf{z}_{ij}$
- Expected observation $\bar{\mathbf{z}}_{ij} = \mathbf{c}_j \ominus \mathbf{c}_i$

- Difference between observation and expectation

$$\mathbf{e}_{ij} = \mathbf{z}_{ij} \ominus \bar{\mathbf{z}}_{ij}$$

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

Error Function

- **Assumption:** Sensor noise is normally distributed

$$\mathbf{e}_{ij} \sim \mathcal{N}(\mathbf{0}, \Sigma_{ij})$$

- Error term for one observation (proportional to negative loglikelihood)

$$f_{ij}(\mathbf{x}) = \mathbf{e}_{ij}(\mathbf{x})^\top \Sigma_{ij}^{-1} \mathbf{e}_{ij}(\mathbf{x})$$

- Note: error is a scalar $f_{ij}(\mathbf{x}) \in \mathbb{R}$

Error Function

- Map error (over all observations)

$$f(\mathbf{x}) = \sum_{ij} f_{ij}(\mathbf{x}) = \sum_{ij} \mathbf{e}_{ij}(\mathbf{x})^\top \Sigma_{ij}^{-1} \mathbf{e}_{ij}(\mathbf{x})$$

- **Minimize this error** by optimizing the camera poses

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \sum_{ij} \mathbf{e}_{ij}(\mathbf{x})^\top \Sigma_{ij}^{-1} \mathbf{e}_{ij}(\mathbf{x})$$

- How can we solve this optimization problem?

Non-Linear Optimization Techniques

- Gradient descend
- Gauss-Newton
- Levenberg-Marquardt

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

$$f(\mathbf{x}) = \sum_{ij} f_{ij}(\mathbf{x}) = \sum_{ij} \mathbf{e}_{ij}(\mathbf{x})^\top \Sigma_{ij}^{-1} \mathbf{e}_{ij}(\mathbf{x})$$

- Evaluate the error function around the initial guess

$$f(\mathbf{x} + \Delta\mathbf{x}) = \sum_{ij} \mathbf{e}_{ij}(\mathbf{x} + \Delta\mathbf{x})^\top \Sigma_{ij}^{-1} \underline{\mathbf{e}_{ij}(\mathbf{x} + \Delta\mathbf{x})}$$

Let's derive this term first…

Linearize the Error Function

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

$$\mathbf{e}_{ij}(\mathbf{x} + \Delta\mathbf{x}) \simeq \mathbf{e}_{ij}(\mathbf{x}) + J_{ij}\Delta\mathbf{x}$$

with

$$J_{ij}(\mathbf{x}) = \begin{pmatrix} \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{c}_1} & \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{c}_2} & \cdots & \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{c}_n} \end{pmatrix}$$

Derivatives of the Error Terms

- Does one error function $\mathbf{e}_{ij}(\mathbf{x})$ depend on all state variables in $\mathbf{x}$?

Derivatives of the Error Terms

- Does one error function $\mathbf{e}_{ij}(\mathbf{x})$ depend on all state variables in $\mathbf{x}$?
  - No, $\mathbf{e}_{ij}(\mathbf{x})$ depends only on $\mathbf{c}_i$ and $\mathbf{c}_j$

Derivatives of the Error Terms

- Does one error function $\mathbf{e}_{ij}(\mathbf{x})$ depend on all state variables in $\mathbf{x}$?
  - No, $\mathbf{e}_{ij}(\mathbf{x})$ depends only on $\mathbf{c}_i$ and $\mathbf{c}_j$
- Is there any consequence on the **structure** of the Jacobian?

Derivatives of the Error Terms

- Does one error function $\mathbf{e}_{ij}(\mathbf{x})$ depend on all state variables in $\mathbf{x}$?
  - No, $\mathbf{e}_{ij}(\mathbf{x})$ depends only on $\mathbf{c}_i$ and $\mathbf{c}_j$
- Is there any consequence on the **structure** of the Jacobian?
  - Yes, it will be non-zero only in the columns corresponding to $\mathbf{c}_i$ and $\mathbf{c}_j$
  - Jacobian is **sparse**

$$J_{ij}(\mathbf{x}) = \begin{pmatrix} \mathbf{0} & \cdots & \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{c}_i} & \cdots & \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{c}_j} & \cdots & \mathbf{0} \end{pmatrix}$$

Linearizing the Error Function

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

$$\simeq \mathbf{c} + 2\mathbf{b}^\top \Delta\mathbf{x} + \Delta\mathbf{x}^\top H \Delta\mathbf{x}$$

with $\mathbf{b}^\top = \sum_{ij} \mathbf{e}_{ij}^\top \Sigma_{ij}^{-1} J_{ij}$

$$H = \sum_{ij} J_{ij}^\top \Sigma_{ij}^{-1} J_{ij}$$

- What is the structure of $\mathbf{b}^\top$ and H? (Remember: all J_{ij}'s are sparse)

Illustration of the Structure

$$\mathbf{b}_{ij}^\top = \mathbf{e}_{ij}^\top \Sigma_{ij}^{-1} J_{ij}$$

Non-zero only at $\mathbf{c}_i$ and $\mathbf{c}_j$

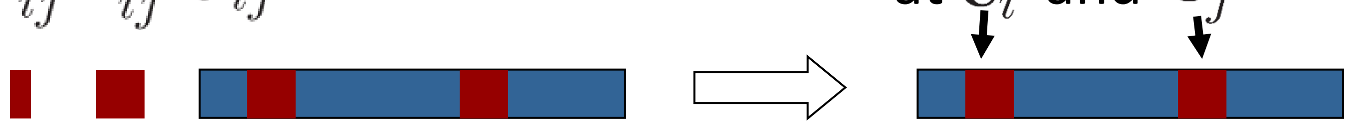
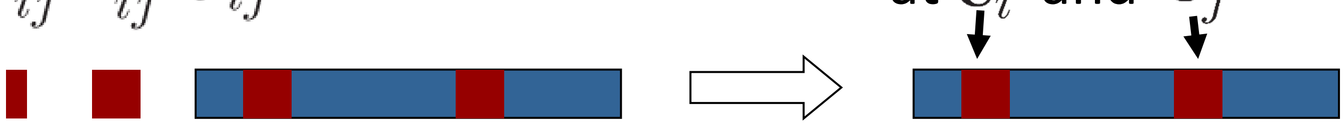


Illustration of the Structure

$$\mathbf{b}_{ij}^\top = \mathbf{e}_{ij}^\top \Sigma_{ij}^{-1} J_{ij}$$

Non-zero only at $\mathbf{c}_i$ and $\mathbf{c}_j$



$$H_{ij} = J_{ij}^\top \Sigma_{ij}^{-1} J_{ij}$$

Non-zero on the main diagonal at $\mathbf{c}_i$ and $\mathbf{c}_j$

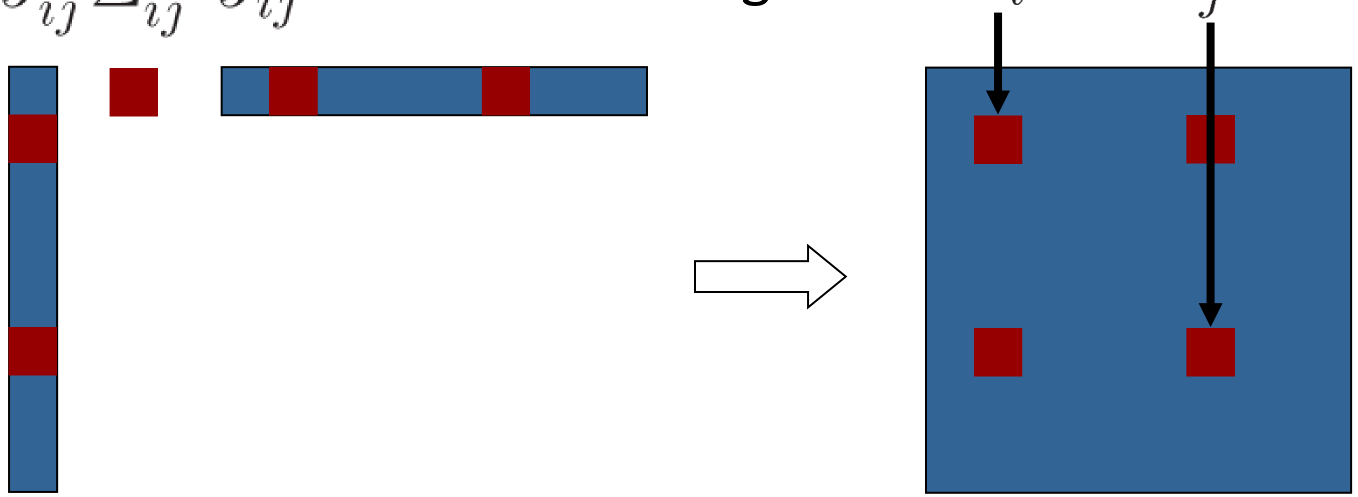
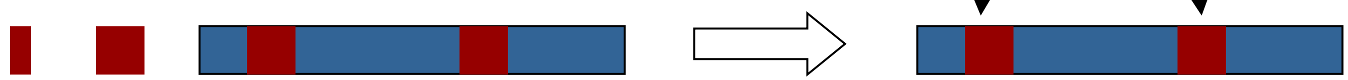


Illustration of the Structure

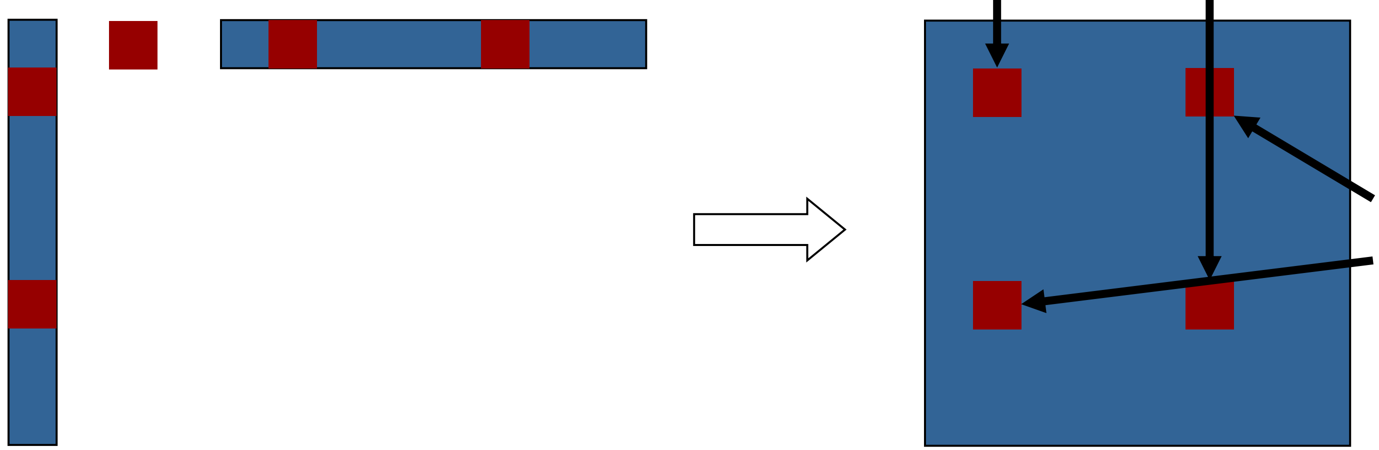
$$\mathbf{b}_{ij}^\top = \mathbf{e}_{ij}^\top \Sigma_{ij}^{-1} J_{ij}$$

Non-zero only at $\mathbf{c}_i$ and $\mathbf{c}_j$



$$H_{ij} = J_{ij}^\top \Sigma_{ij}^{-1} J_{ij}$$

Non-zero on the main diagonal at $\mathbf{c}_i$ and $\mathbf{c}_j$

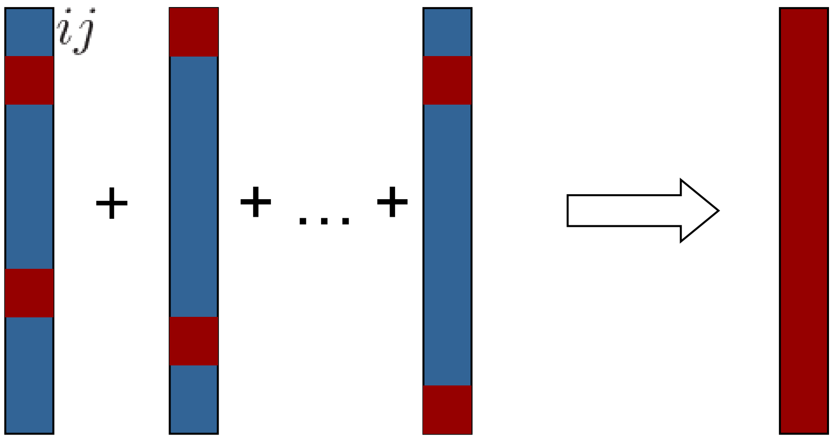


... and at the blocks *ij,ji*

Illustration of the Structure

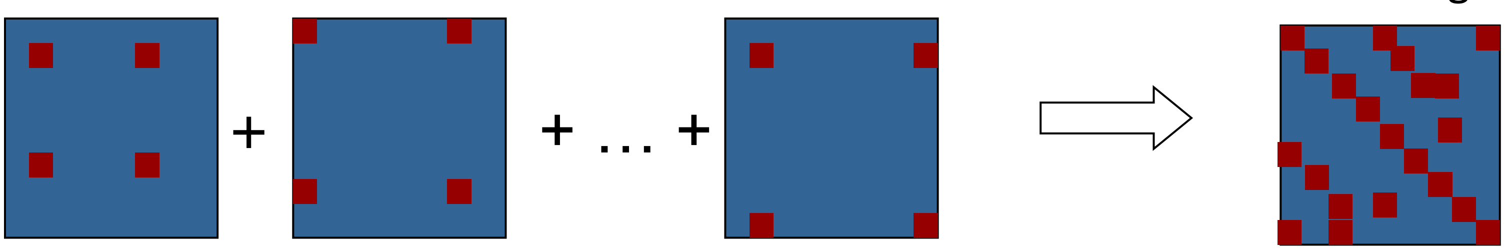
$$\mathbf{b} = \sum_{ij} \mathbf{b}_{ij}$$

b: dense vector



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

H: sparse block structure with main diagonal



(Linear) Least Squares Minimization

1. Linearize error function

$$f(\mathbf{x} + \Delta\mathbf{x}) \simeq \mathbf{c} + 2\mathbf{b}^\top \Delta\mathbf{x} + \Delta\mathbf{x}^\top H \Delta\mathbf{x}$$

2. Compute the derivative

$$\frac{\mathrm{d}f(\mathbf{x} + \Delta\mathbf{x})}{\mathrm{d}\Delta\mathbf{x}} = 2\mathbf{b} + 2H\Delta\mathbf{x}$$

3. Set derivative to zero

$$H\Delta\mathbf{x} = -\mathbf{b}$$

4. Solve this linear system of equations, e.g.,

$$\Delta\mathbf{x} = -H^{-1}\mathbf{b}$$

Gauss-Newton Method

Problem: $f(\mathbf{x})$ is non-linear!

Algorithm: Repeat until convergence

1. Compute the terms of the linear system

$$\mathbf{b}^\top = \sum_{ij} \mathbf{e}_{ij}^T \Sigma_{ij}^{-1} J_{ij} \qquad H = \sum_{ij} J_{ij}^\top \Sigma_{ij}^{-1} J_{ij}$$

2. Solve the linear system to get new increment

$$H \Delta \mathbf{x} = -\mathbf{b}$$

3. Update previous estimate

$$\mathbf{x} \leftarrow \mathbf{x} + \Delta \mathbf{x}$$

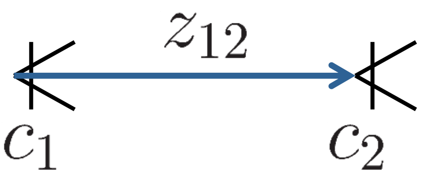
Sparsity of the Hessian

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

Example in 1D

- Two camera poses $c_1, c_2 \in \mathbb{R}$
- State vector $\mathbf{x} = (c_1, c_2)^\top \in \mathbb{R}^2$
- One (distance) observation $z_{12} \in \mathbb{R}$

- Initial guess $c_1 = c_2 = 0$
- Observation $z_{12} = 1$
- Sensor noise $\Sigma_{12} = 0.5$



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 $$b^\top = e_{12}^\top \Sigma^{-1} e_{12} = \begin{pmatrix} 2 & -2 \end{pmatrix}$$ $$H = J_{12}^\top \Sigma^{-1} J_{12} = \begin{pmatrix} 2 & -2 \\ -2 & 2 \end{pmatrix}$$
- Solve the system $$\Delta x = -H^{-1} b$$ **but** $\det H = 0$ **???**

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 $H, \mathbf{b}$ 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)**

$$H = \begin{pmatrix} 2 & -2 \\ -2 & 2 \end{pmatrix} + \boxed{\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}}$$

additional constraint that sets $\Delta c_1 = 0$

$$\Delta x = -H^{-1} b$$

$$\Delta x = \begin{pmatrix} 0 & 1 \end{pmatrix}^\top$$

Levenberg-Marquardt Algorithm

- **Idea:** Add a damping factor

$$(H + \lambda I)\Delta\mathbf{x} = -\mathbf{b}$$

$$(J^\top J + \lambda I)\Delta\mathbf{x} = -J^\top\mathbf{e}$$

- What is the effect of this damping factor?
  - Small λ?
  - Large λ?

Levenberg-Marquardt Algorithm

- **Idea:** Add a damping factor

$$(H + \lambda I)\Delta\mathbf{x} = -\mathbf{b}$$

$$(J^\top J + \lambda I)\Delta\mathbf{x} = -J^\top \mathbf{e}$$

- What is the effect of this damping factor?
  - Small λ → same as least squares
  - Large λ → steepest descent (with small step size)
- **Algorithm**
  - If error decreases, accept $\Delta\mathbf{x}$ and reduce λ
  - If error increases, reject $\Delta\mathbf{x}$ and increase λ

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

$$\mathbf{x} = (\mathbf{c}_1^\top, \ldots, \mathbf{c}_n^\top)^\top \in \mathbb{R}^{6n}$$

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

$$\mathbf{x} = (\underbrace{\mathbf{c}_1^\top, \ldots, \mathbf{c}_n^\top}_{\mathbf{c} \in \mathbb{R}^{6n}}, \underbrace{\mathbf{p}_1^\top, \ldots, \mathbf{p}_m^\top}_{\mathbf{p} \in \mathbb{R}^{3m}})^\top \in \mathbb{R}^{6n+3m}$$

- Typically $m \gg n$ (why?)

Error Function

- Camera pose $\mathbf{c}_i \in \mathbb{R}^6$
- Feature point $\mathbf{p}_j \in \mathbb{R}^3$
- Observed feature location $\mathbf{z}_{ij} \in \mathbb{R}^2$
- Expected feature location

$$g(\mathbf{c}_i, \mathbf{p}_j) = R_i^\top (\mathbf{t}_i - \mathbf{p}_j)$$

$$h(\mathbf{c}_i, \mathbf{p}_j) = g_{x,y}(\mathbf{c}_i, \mathbf{p}_j) / g_z(\mathbf{c}_i, \mathbf{p}_j)$$

Error Function

- Difference between observation and expectation

$$\mathbf{e}_{ij} = \mathbf{z}_{ij} - h(\mathbf{c}_i, \mathbf{p}_j)$$

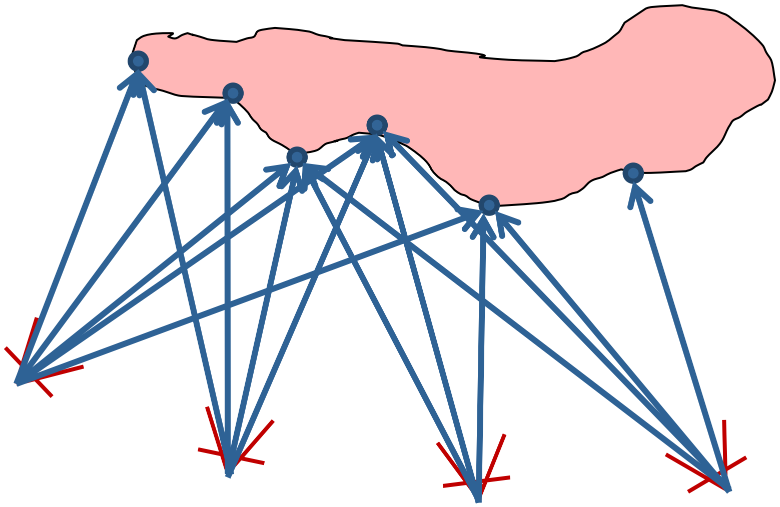
- Error function

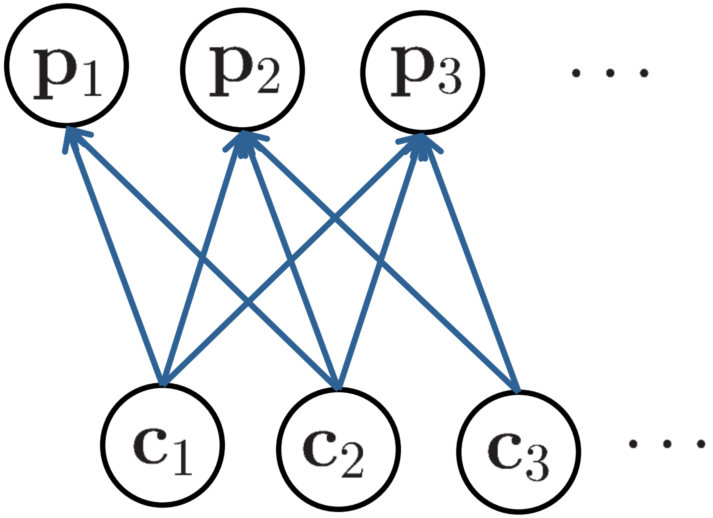
$$f(\mathbf{c}, \mathbf{p}) = \sum_{ij} \mathbf{e}_{ij}^\top \Sigma^{-1} \mathbf{e}_{ij}$$

- Covariance Σ is often chosen isotropic and on the order of one pixel

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

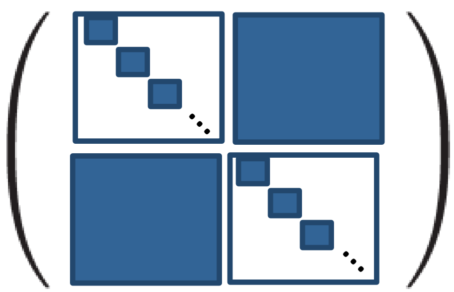
- Characteristic structure

$$\begin{pmatrix} J_{\mathbf{c}}^\top J_{\mathbf{c}} & J_{\mathbf{c}}^\top J_{\mathbf{p}} \\ J_{\mathbf{p}}^\top J_{\mathbf{c}} & J_{\mathbf{p}}^\top J_{\mathbf{p}} \end{pmatrix} \begin{pmatrix} \Delta \mathbf{c} \\ \Delta \mathbf{p} \end{pmatrix} = \begin{pmatrix} -J_{\mathbf{c}}^\top \mathbf{e}_{\mathbf{c}} \\ -J_{\mathbf{p}}^\top \mathbf{e}_{\mathbf{p}} \end{pmatrix}$$

$$\begin{pmatrix} H_{\mathbf{cc}} & H_{\mathbf{cp}} \\ H_{\mathbf{pc}} & H_{\mathbf{pp}} \end{pmatrix} \begin{pmatrix} \Delta \mathbf{c} \\ \Delta \mathbf{p} \end{pmatrix} = \begin{pmatrix} -J_{\mathbf{c}}^\top \mathbf{e}_{\mathbf{c}} \\ -J_{\mathbf{p}}^\top \mathbf{e}_{\mathbf{p}} \end{pmatrix}$$

Primary Structure

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



$$\begin{pmatrix}\Delta \mathbf{c}\\ \Delta \mathbf{p}\end{pmatrix} = \begin{pmatrix} -J_{\mathbf{c}}^\top \mathbf{e}_{\mathbf{c}} \\ -J_{\mathbf{p}}^\top \mathbf{e}_{\mathbf{p}}\end{pmatrix}$$

- This can be efficiently solved using the Schur Complement

Schur Complement

- Given: Linear system

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

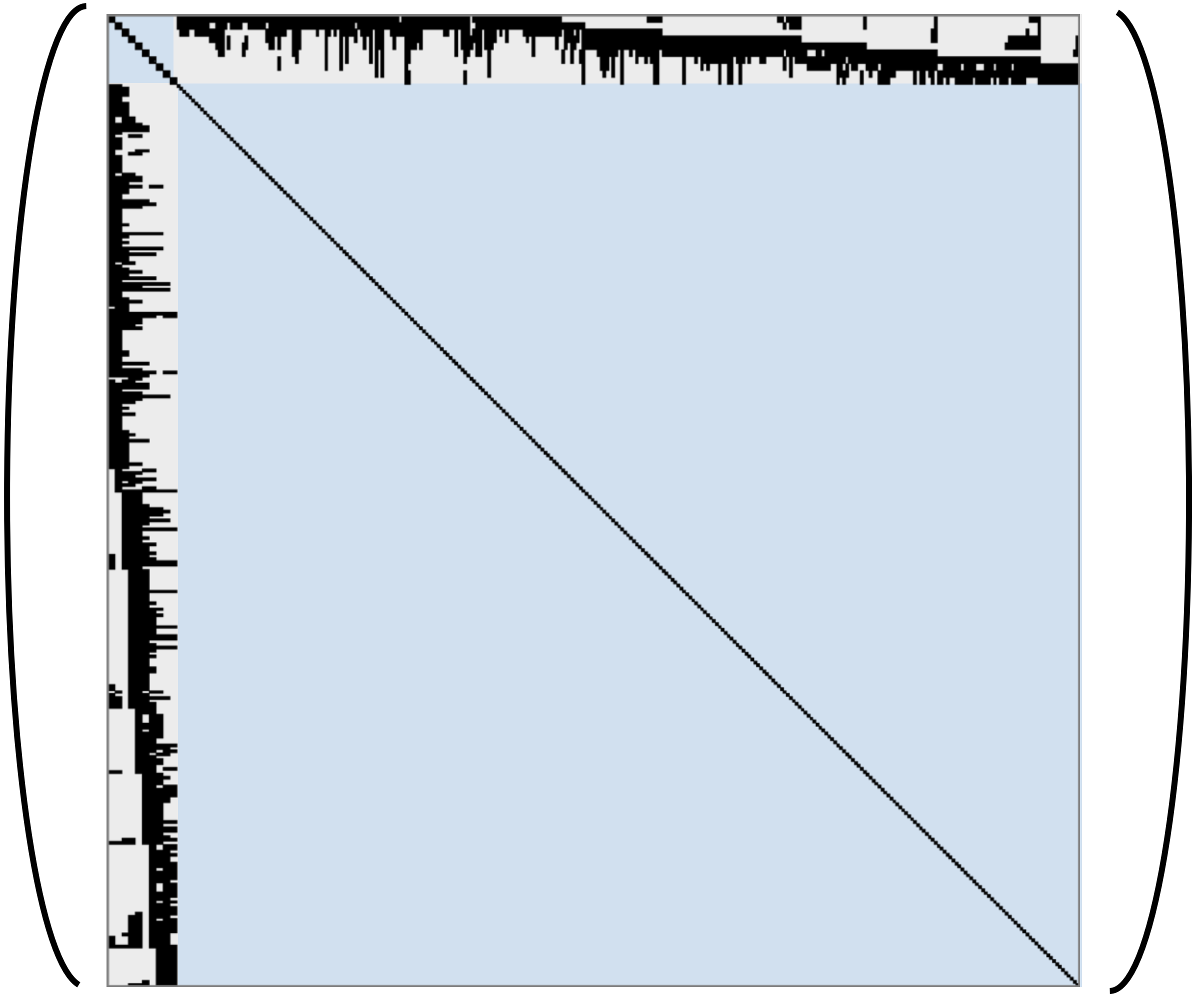
- If D is invertible, then (using Gauss elimination)

$$\begin{aligned} (A - BD^{-1}C)\mathbf{x} &= \mathbf{a} - BD^{-1}\mathbf{b} \\ \mathbf{y} &= D^{-1}(\mathbf{b} - C\mathbf{x}) \end{aligned}$$

- **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)

$$H =$$



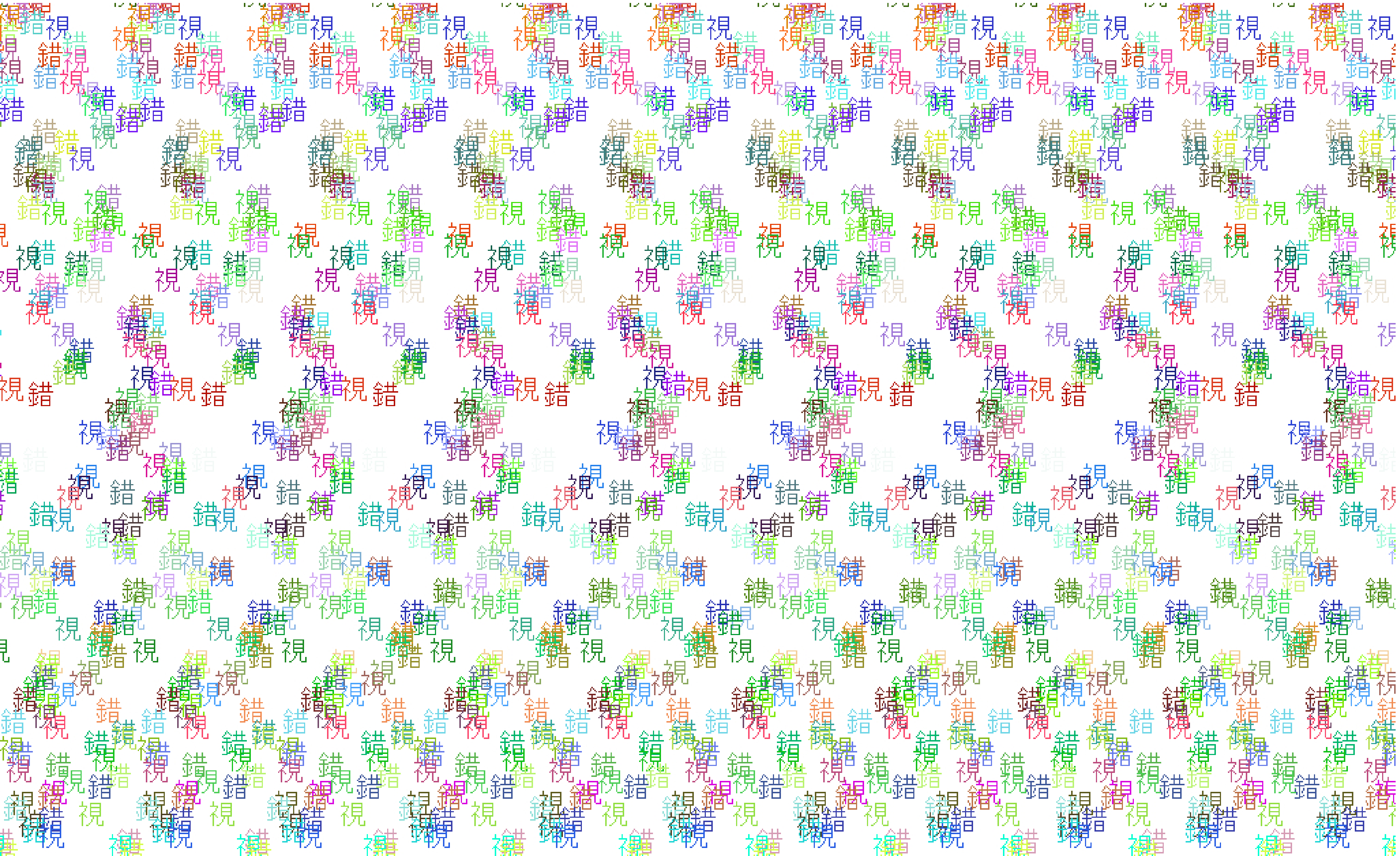
From Sparse Maps to Dense Maps

- 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?

From Sparse Maps to Dense Maps

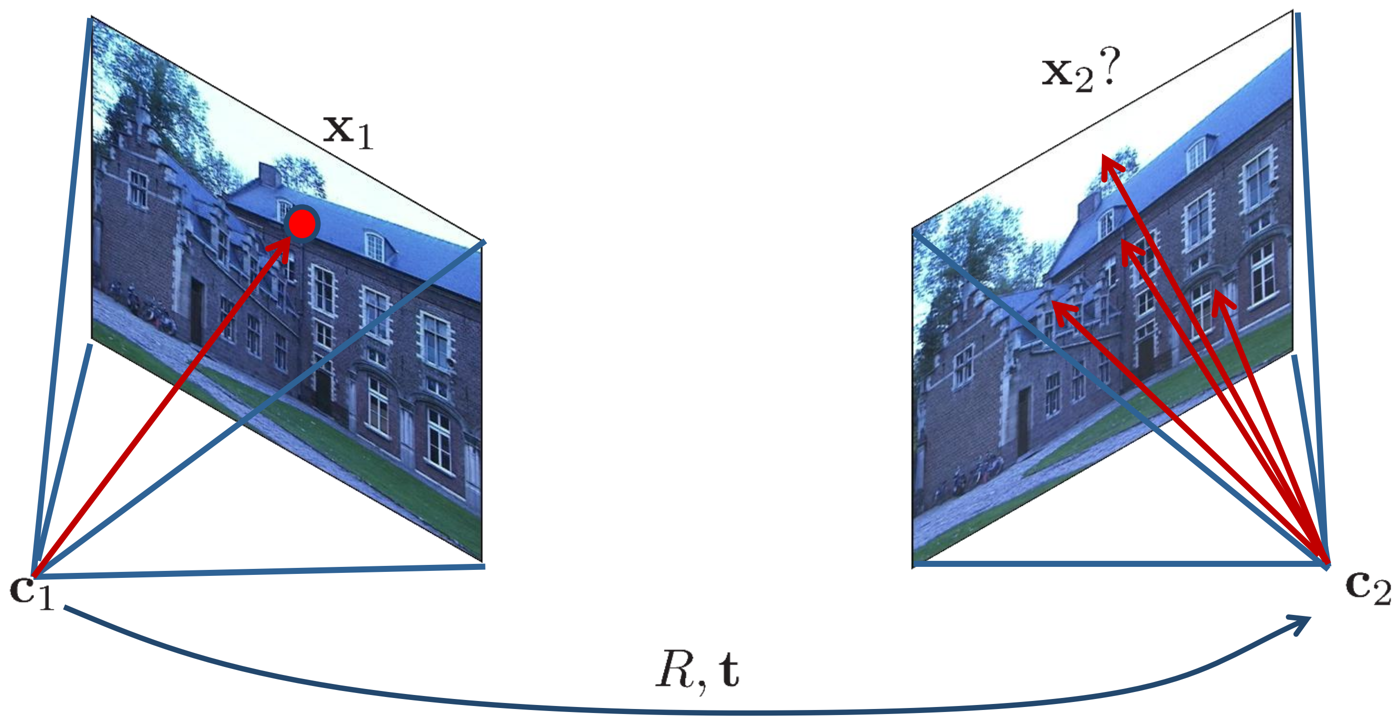
- **Today:** Estimation of depth dense images (stereo cameras, laser triangulation, structured light/Kinect)
- **Next week:** Dense map representations and data fusion

Human Stereo Vision



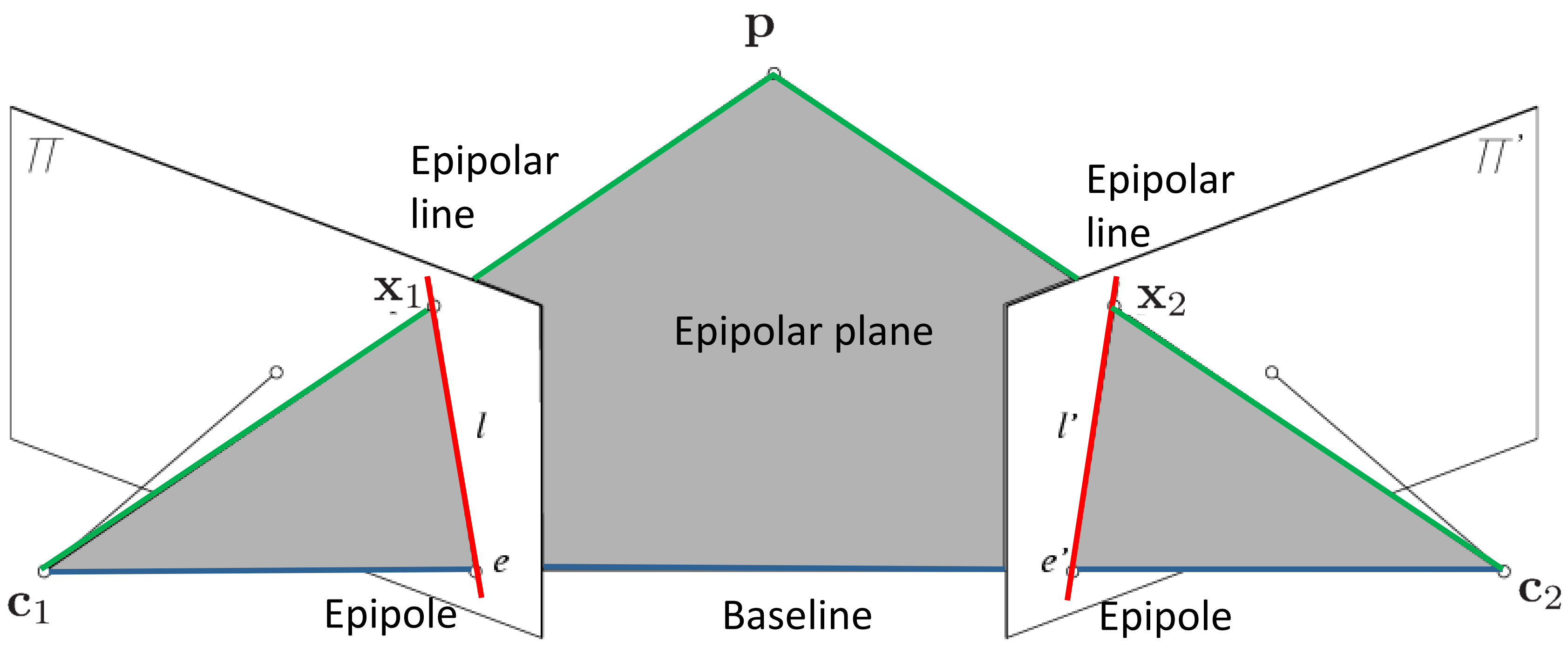
Stereo Correspondence Constraints

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



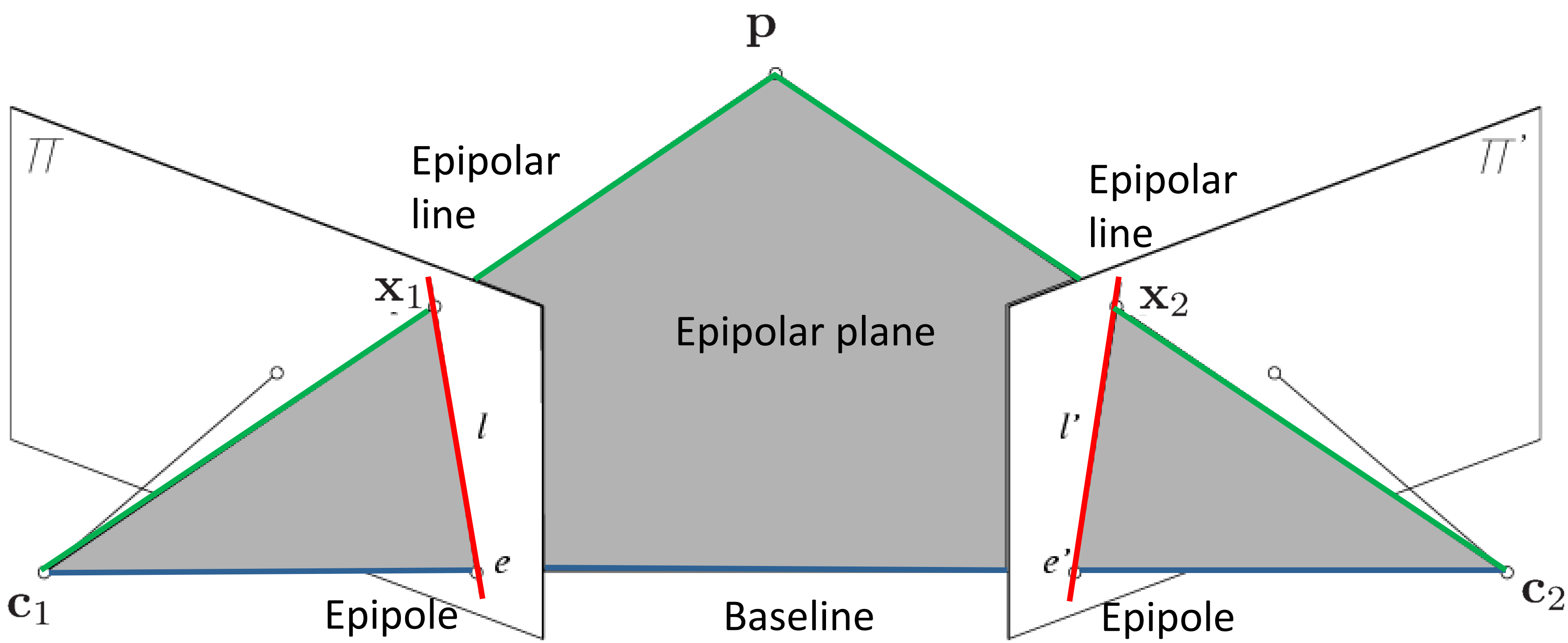
Reminder: Epipolar Geometry

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



Epipolar Plane

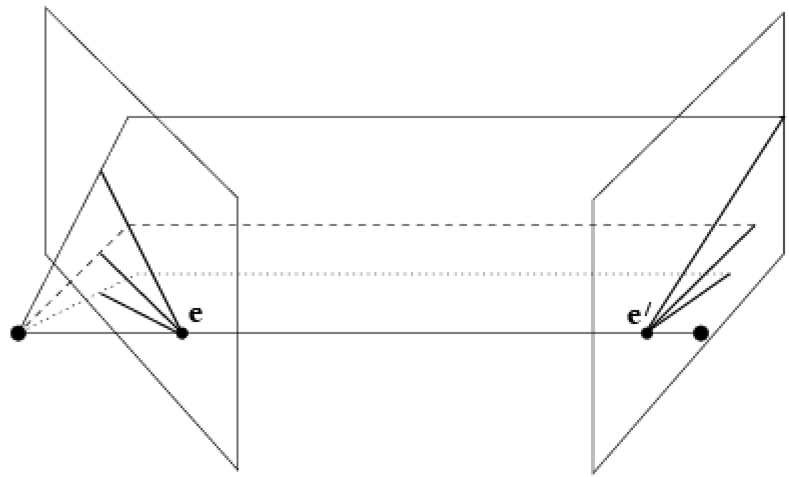
- All epipolar lines intersect at the epipoles
- An epipolar plane intersects the left and right image planes in epipolar lines

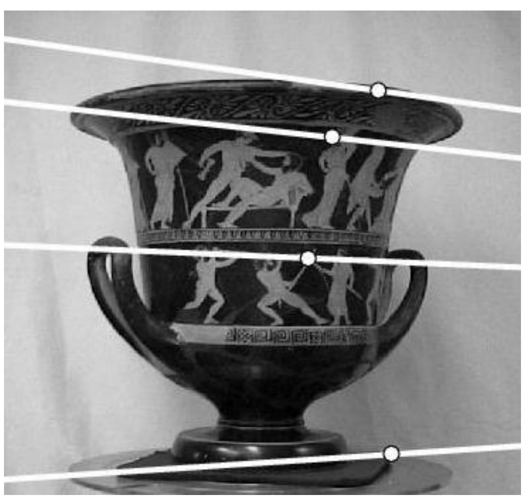


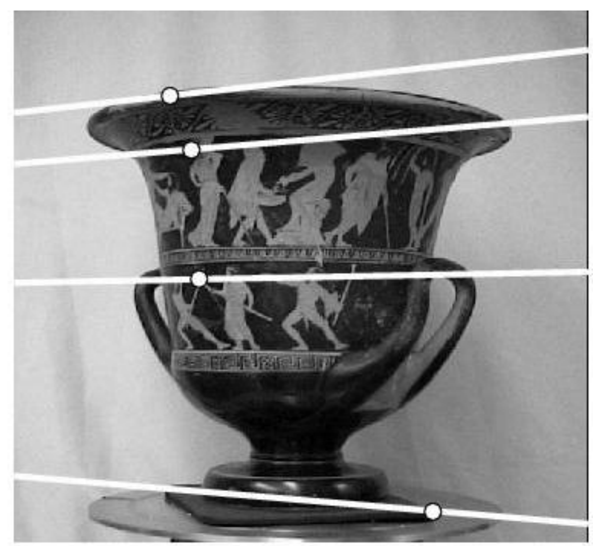
Epipolar Constraint

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

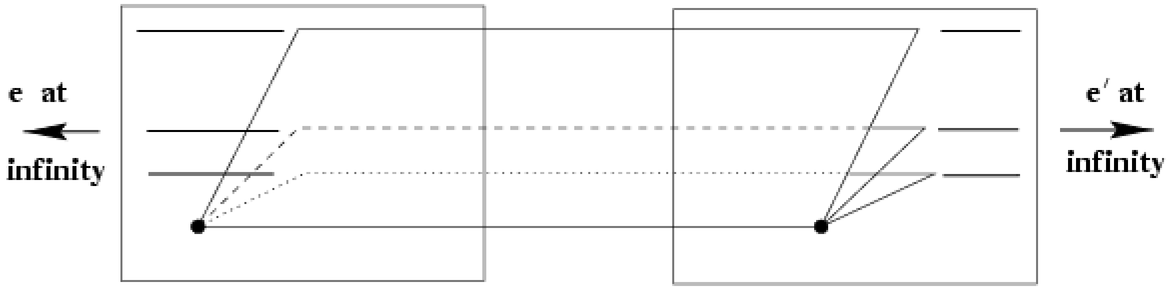
Example: Converging Cameras

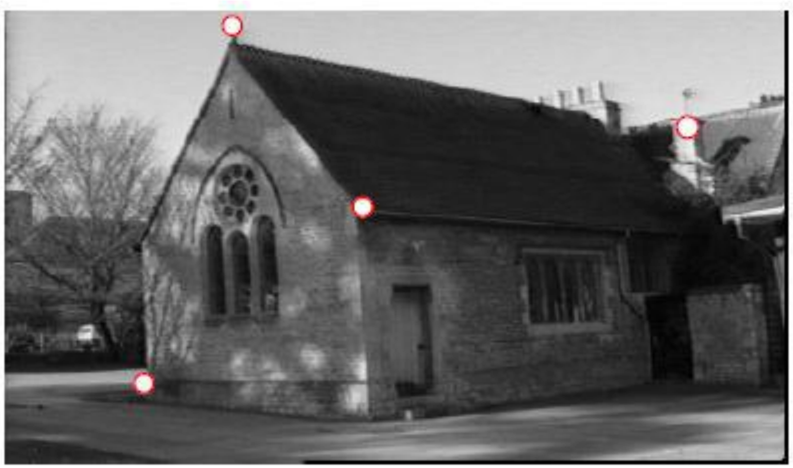






Example: Parallel Cameras



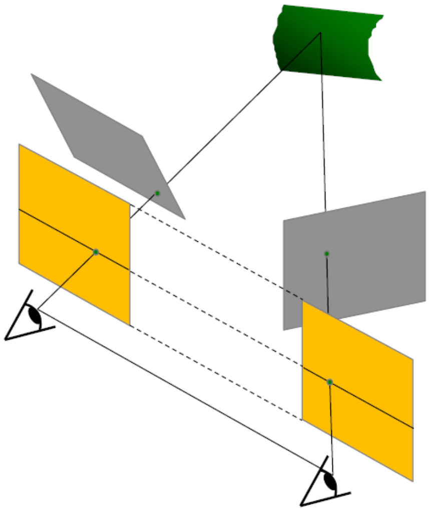


Rectification

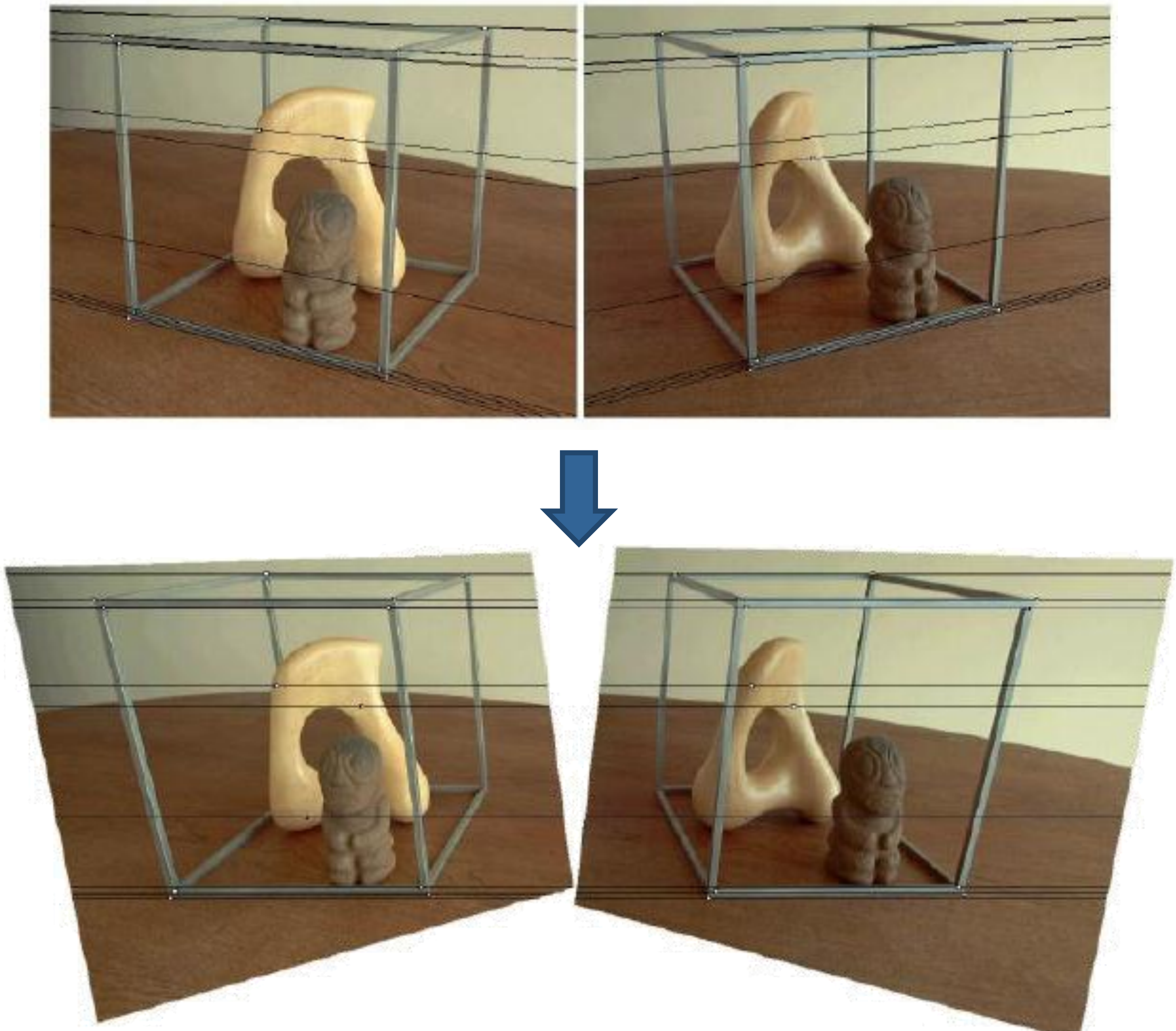
- In practice, it is convenient if the image scanlines (rows) are the epipolar lines

→ Reproject image planes onto a common plane parallel to the baseline (two 3x3 homographies)

- Afterwards pixel motion is horizontal

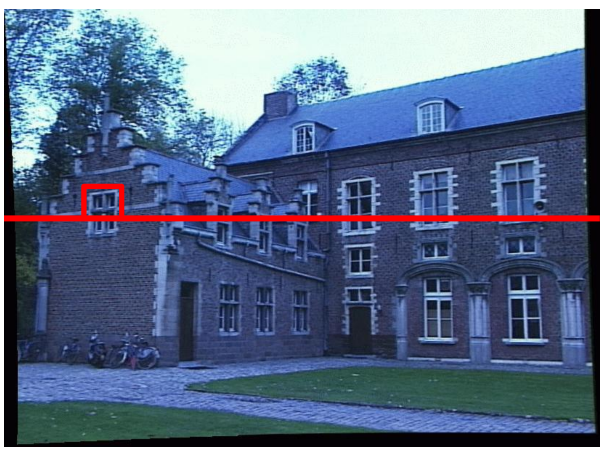


Example: Rectification



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)



left image

right image

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

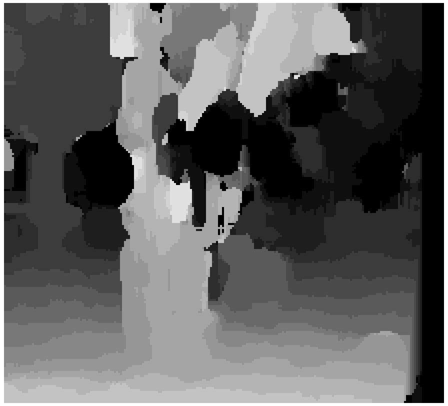
- Input

- 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)

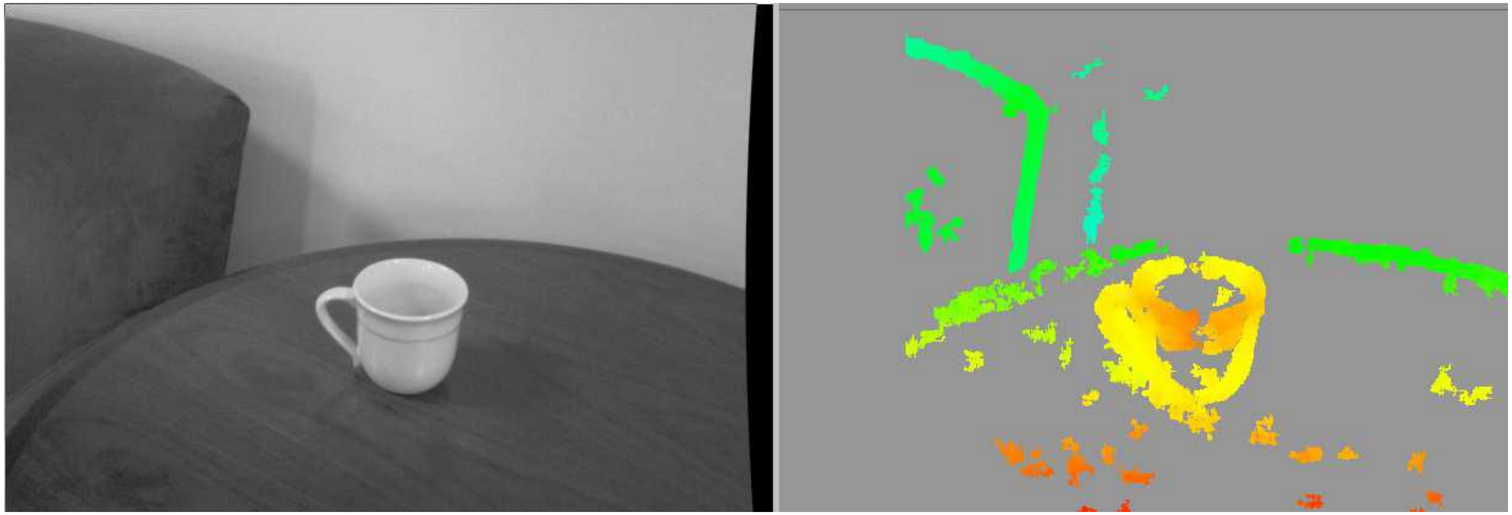
3x3



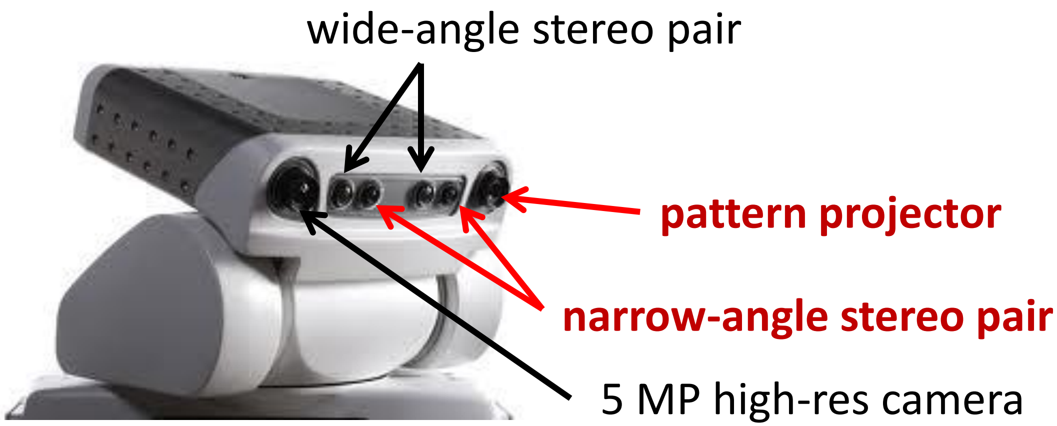
20x20

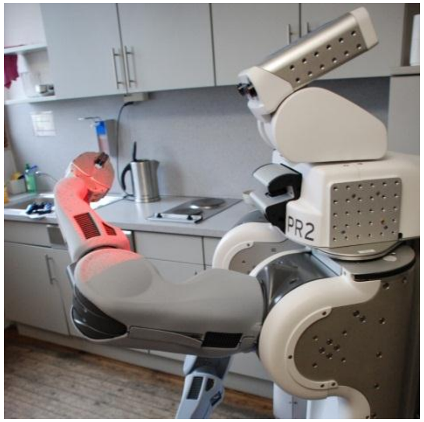
Problems with Stereo

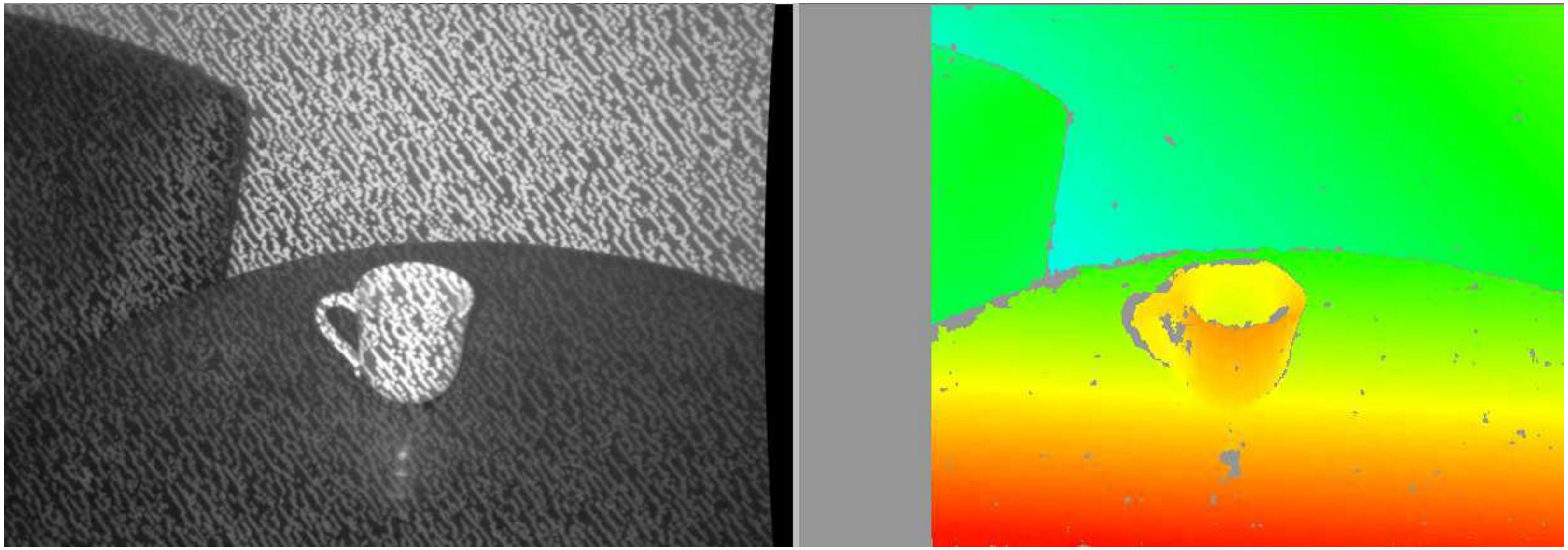
- Block matching typically fails in regions with low texture
  - Global optimization/regularization (speciality of our research group)
  - Additional texture projection



Example: PR2 Robot with Projected Texture Stereo







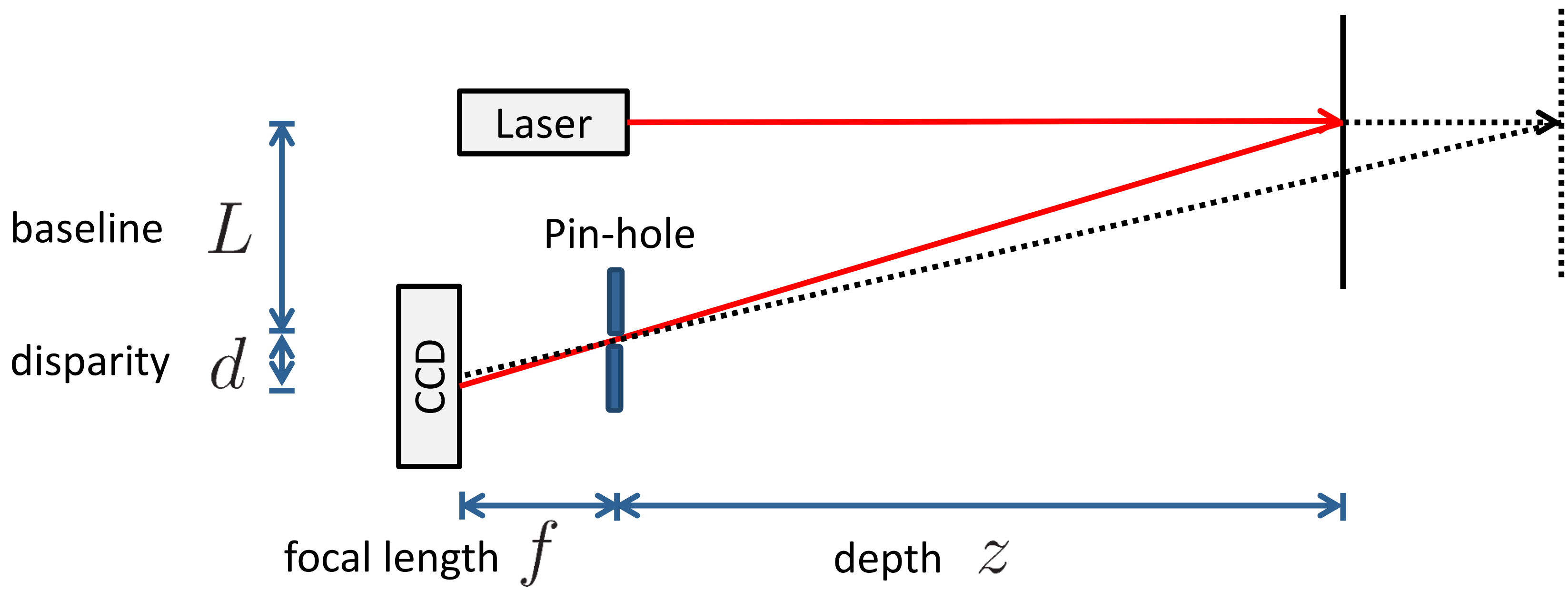
Laser Triangulation

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

Laser Triangulation

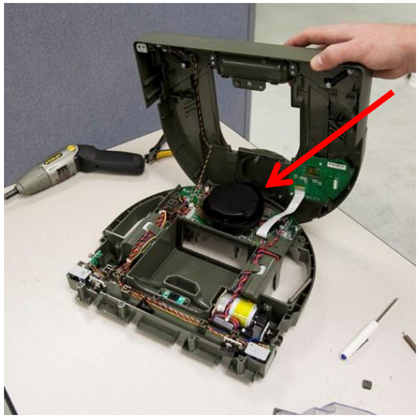
- Function principle

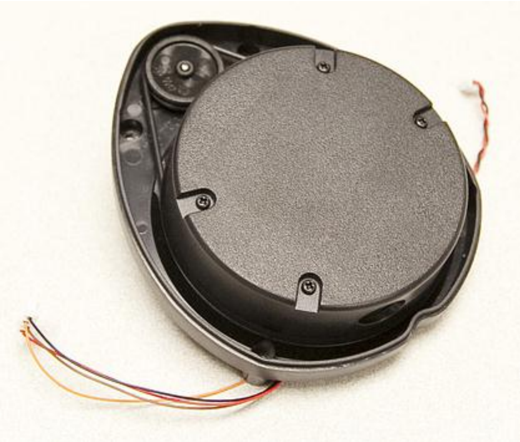


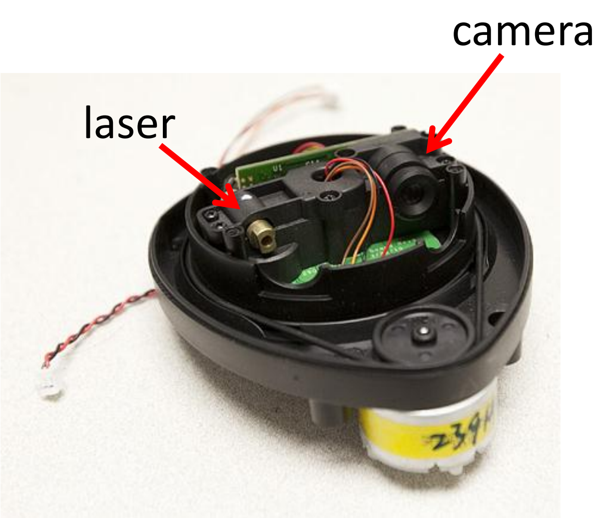
- Depth triangulation $z = f\frac{L}{d}$
(note: same for stereo disparities)

Example: Neato XV-11

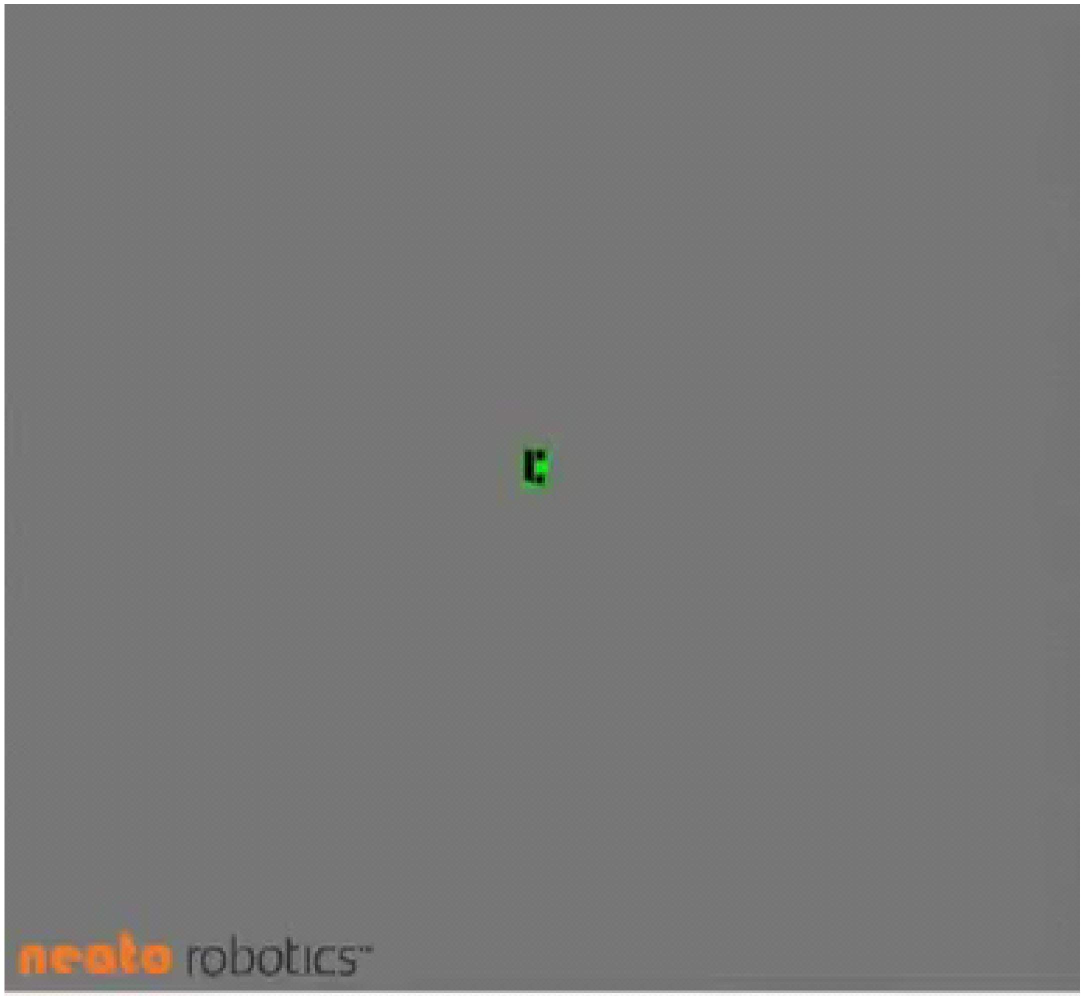
- K. Konolige, “A low-cost laser distance sensor”, ICRA 2008
- Specs: 360deg, 10Hz, 30 USD





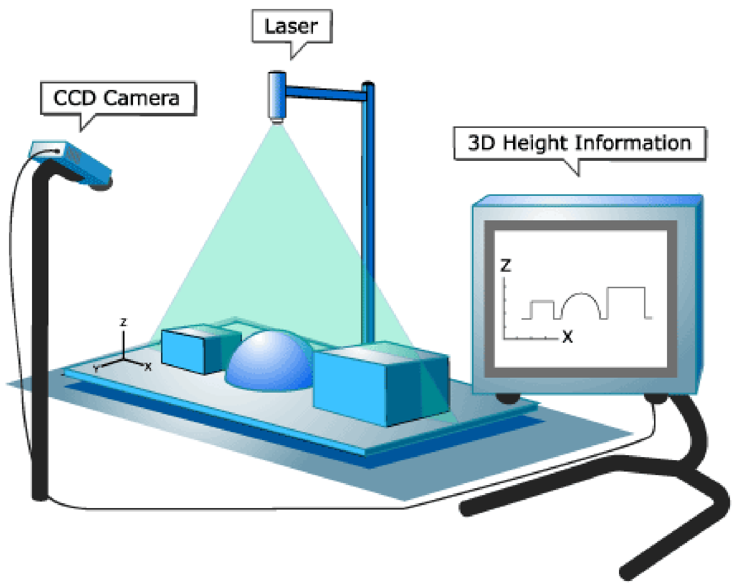


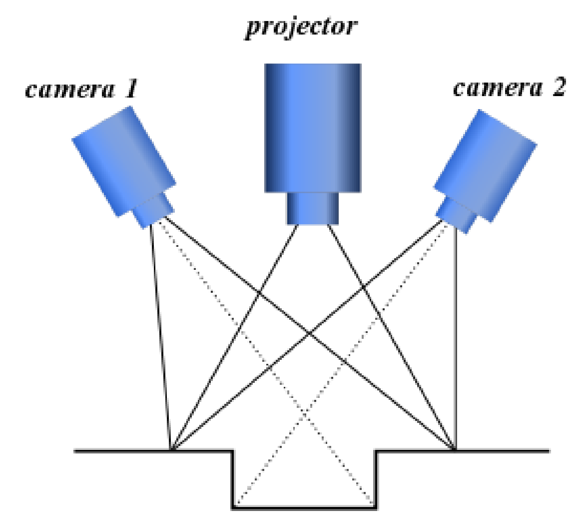
How Does the Data Look Like?



Laser Triangulation

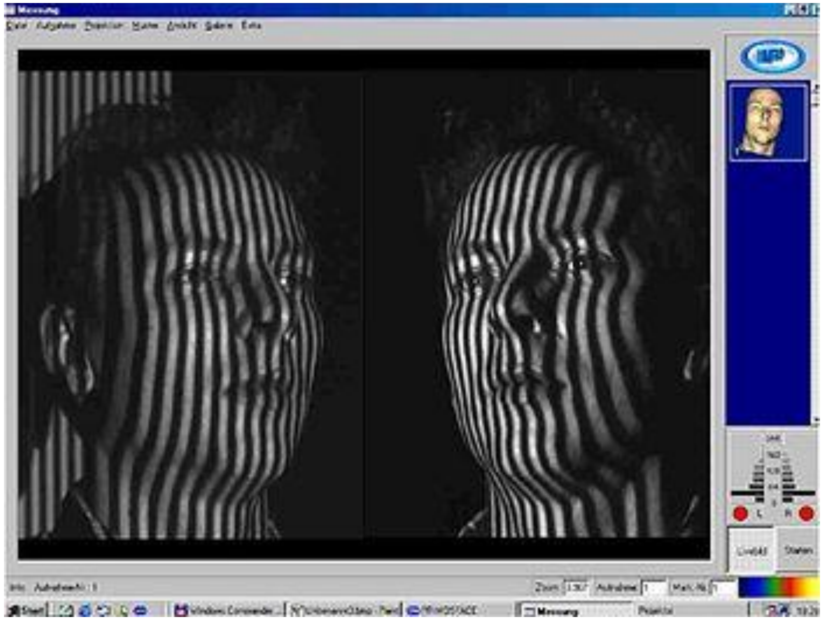
- Stripe laser + 2D camera
- Often used on conveyer belts (volume sensing)
- Large baseline gives better depth resolution but more occlusions → use two cameras





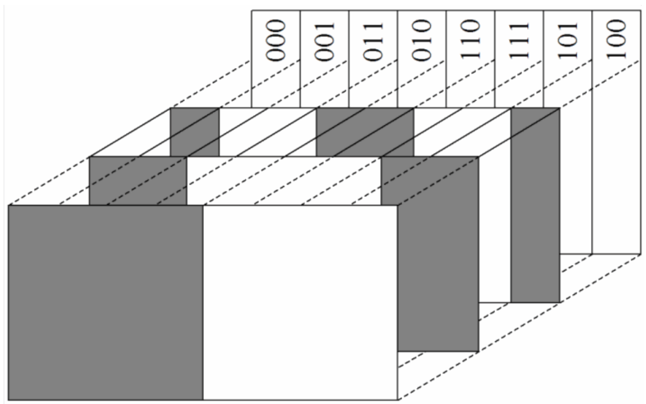
Structured Light

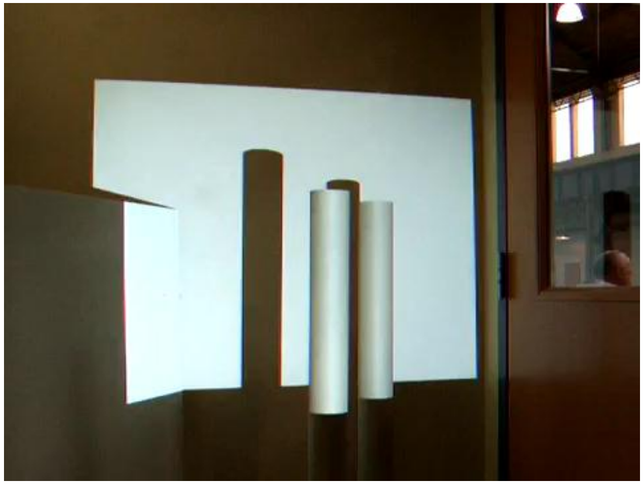
- Multiple stripes / 2D pattern
- Data association more difficult



Structured Light

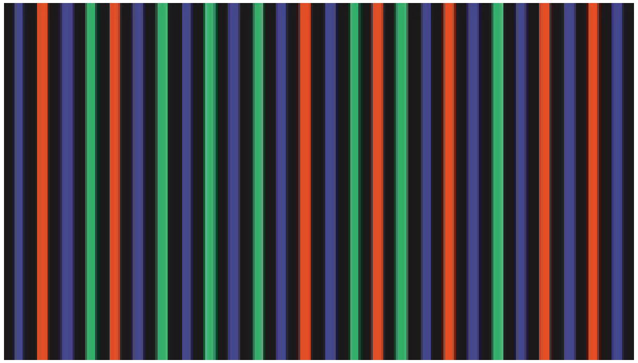
- Multiple stripes / 2D pattern
- Data association more difficult
- Coding schemes
  - Temporal: Coded light





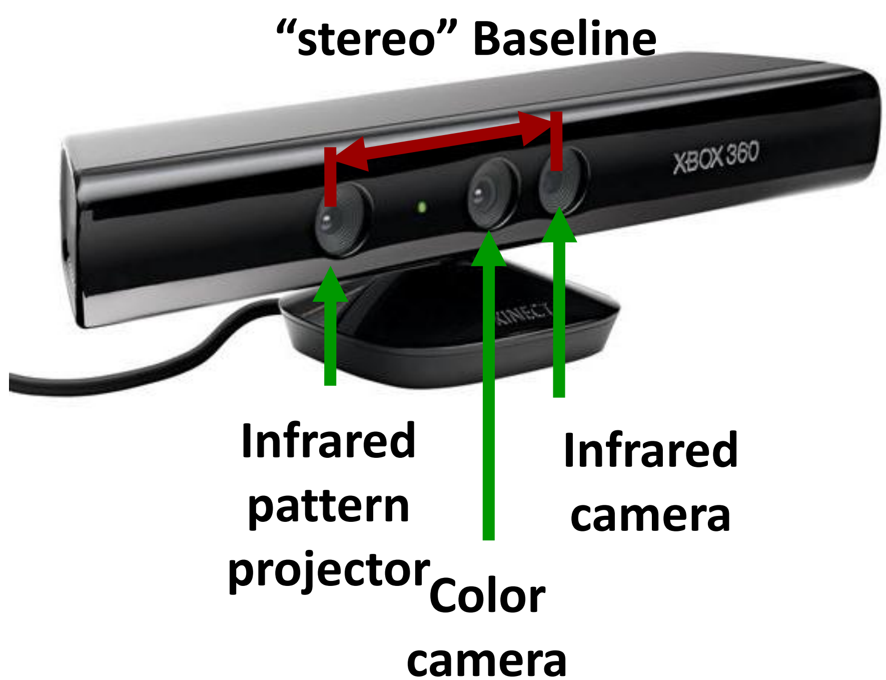
Structured Light

- Multiple stripes / 2D pattern
- Data association more difficult
- Coding schemes
  - Temporal: Coded light
  - Wavelength: Color
  - Spatial: Pattern (e.g., diffraction patterns)



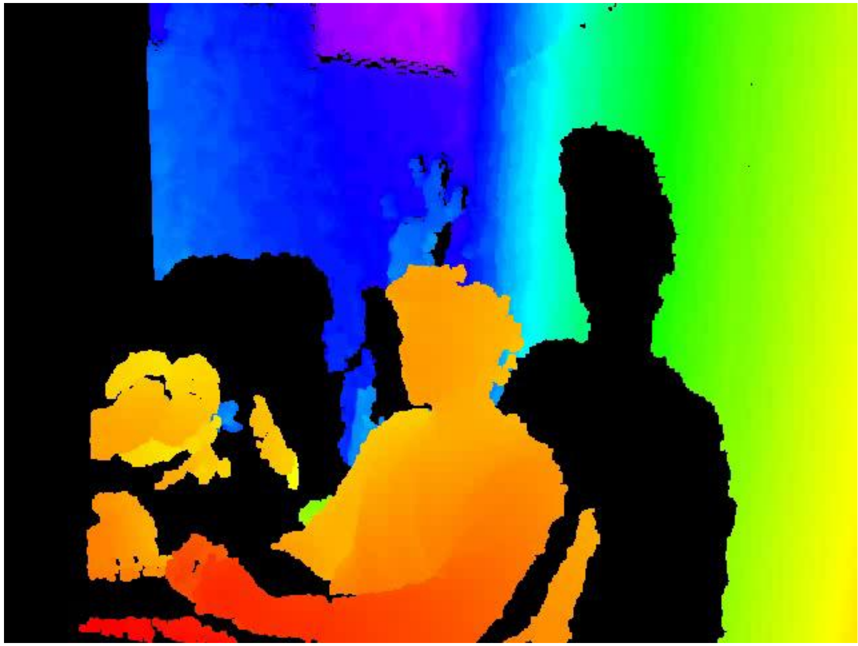
Sensor Principle of Kinect

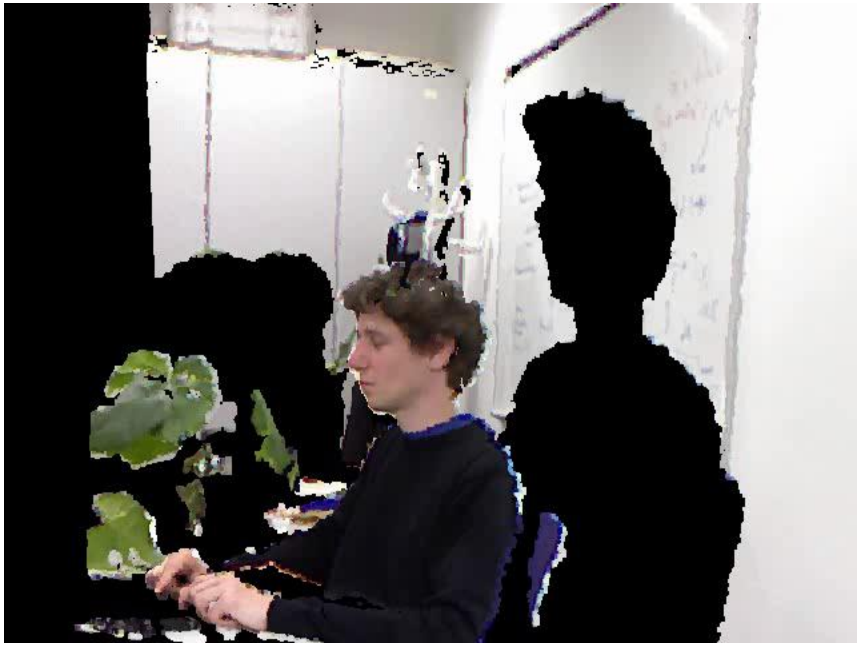
- Kinect projects a diffraction pattern (speckles) in near-infrared light
- CMOS IR camera observes the scene



Example Data

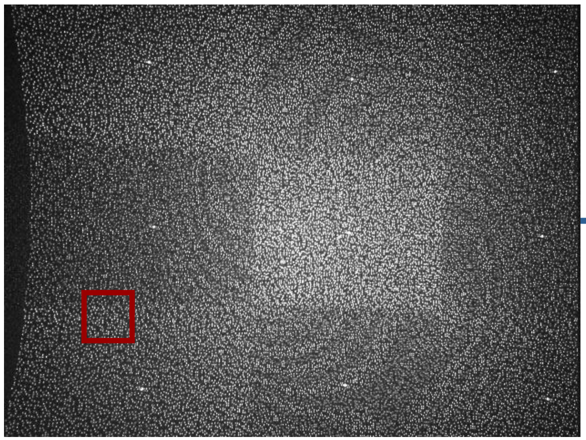
- Kinect provides color (RGB) and depth (D) video
- This allows for novel approaches for (robot) perception



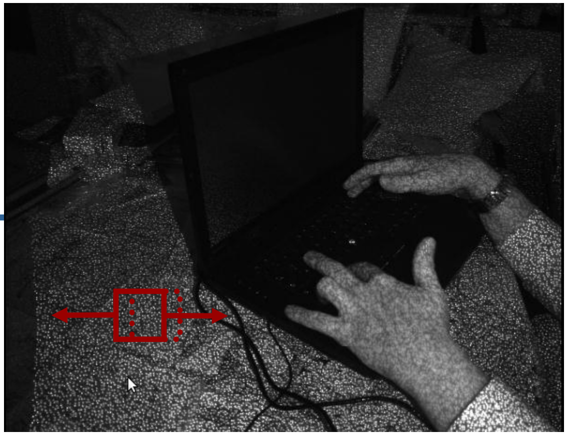


Sensor Principle of Kinect

Infrared pattern
(known)

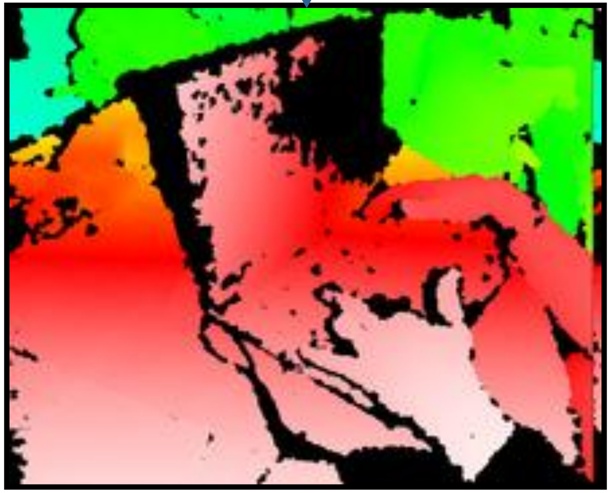


Infrared image
(with distorted pattern)



Standard block matcher (9x9)

Disparity image



Depth image
(color encodes distance from camera)

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

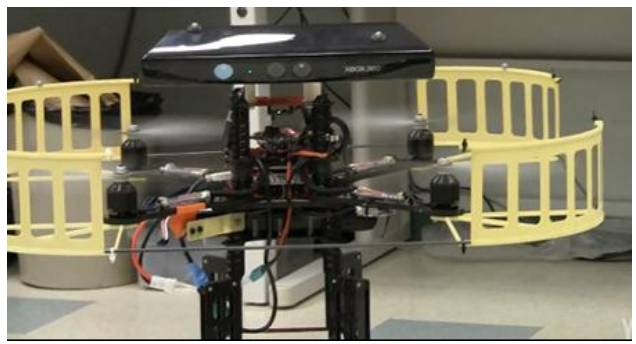
- Infrared camera has 640x480 @ 30 Hz
  - Depth correlation runs on FPGA
  - 11-bit depth image
  - 0.8m – 5m range
  - Depth sensing does not work in direct sunlight (why?)
- RGB camera has 640x480 @ 30 Hz
  - Bayer color filter
- Four 16-bit microphones with DSP for beam forming @ 16kHz
- Requires 12V (for motor), weighs 500 grams
- Human pose recognition runs on Xbox CPU and uses only 10-15% processing power @30 Hz
  (Paper: http://research.microsoft.com/apps/pubs/default.aspx?id=145347)

History

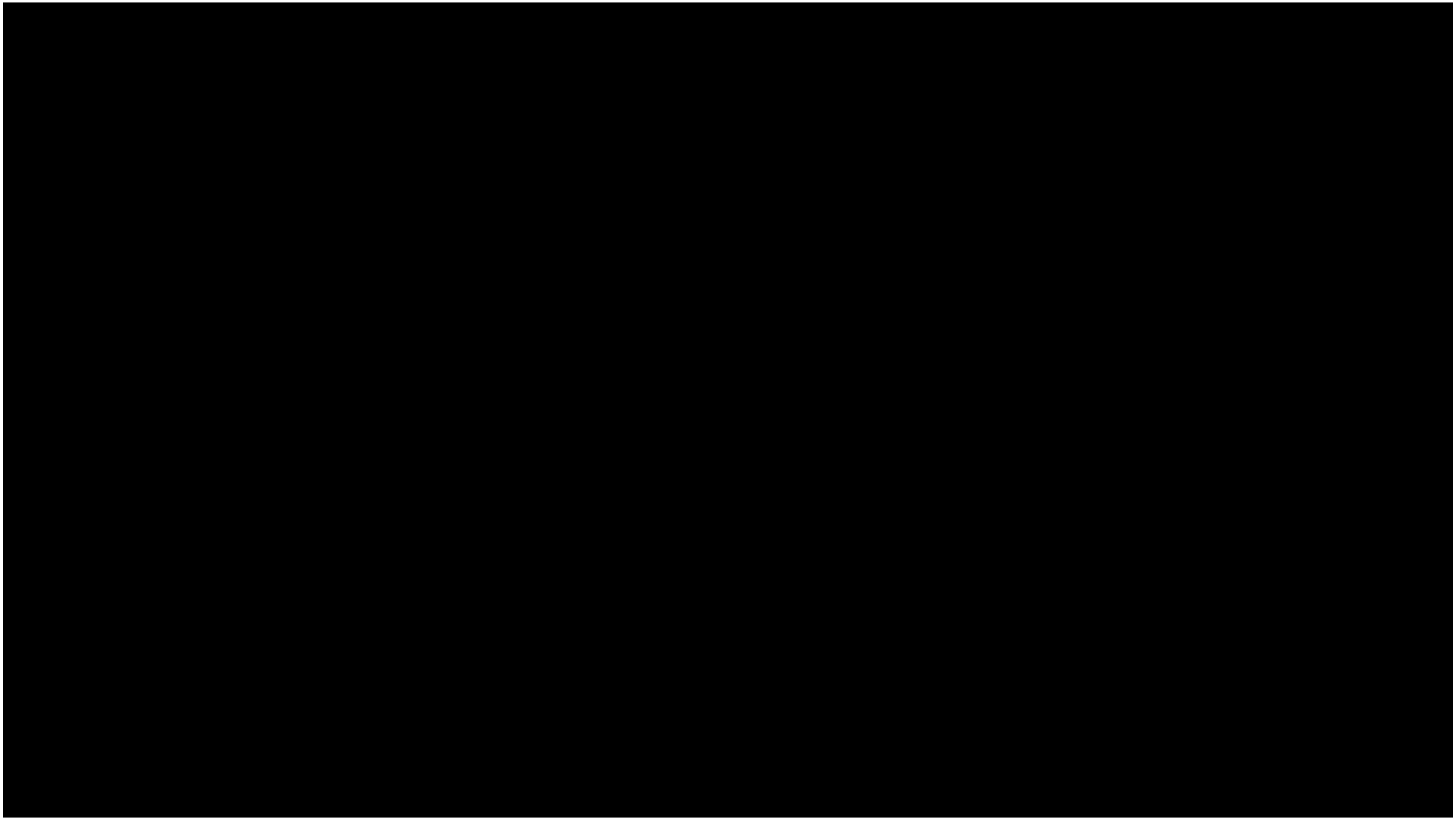
- 2005: Developed by PrimeSense (Israel)
- 2006: Offer to Nintendo and Microsoft, both companies declined
- 2007: Alex Kidman becomes new incubation director at Microsoft, decides to explore PrimeSense device. Johnny Lee assembles a team to investigate technology and develop game concepts
- 2008: The 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 (JVCI, T-SMC)

Impact of the Kinect Sensor

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


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?