3D Scene Point Cloud Reconstruction

3D Computer Vision – Term Project

(CTU FEE subjects B4M33TDV, BE4M33TDV, XP33VID)

Martin Matoušek and Jaroslav Moravec

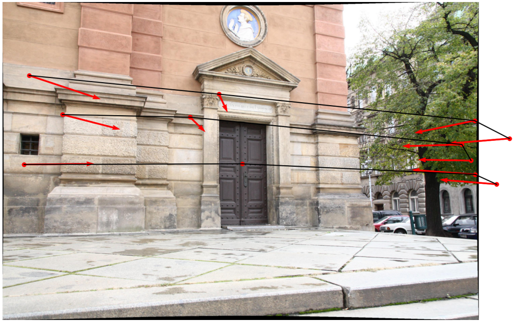
November 2024

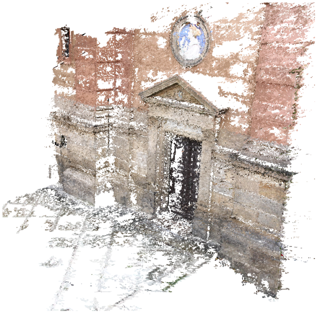
CENTER FOR MACHINE PERCEPTION

Term Project

Phases:

1. Calibrated epipolar geometry (slides: 3–14)
2. Cameras and scene structure (slides: 15–28)
3. Stereo reconstruction (slides: 29–35)

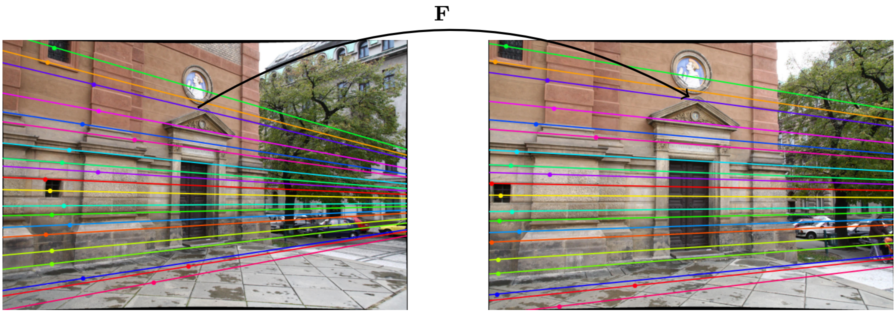




Calibrated Epipolar Geometry

Term project phases:

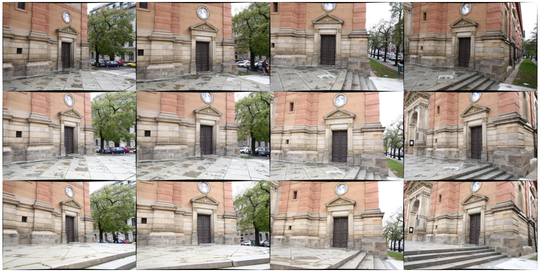
1. **Calibrated epipolar geometry**
2. Cameras and scene structure
3. Stereo reconstruction



Task Outline

Task: **Estimate the calibrated epipolar geometry.**

1. Select one pair of images from the set below and download them from here.
2. Download keypoints and *tentative* correspondences (matches) between the selected pair of views from here.
3. Use robust estimation (RANSAC/MLESAC) to find the essential matrix.
4. Show outliers and inliers of the epipolar geometry (as a needle map).
5. Select a reasonable sub-set of inliers (e.g., every n-th) and show **corresponding** points and **corresponding** epipolar lines in both images (use colors to show correspondence).



Sparse Correspondences

The sparse correspondences for provided images has been precomputed and they are available. Note, that the correspondences are tentative, so they may contain mismatches.

The correspondences are stored in several files:

- detected image keypoints (here `u_<id>.txt`)
- 0-based indices of corresponding keypoints (here `m_<i1>_<i2>.txt`)

Example: Working with correspondences

Keypoints in Image 1

```
5.3 1613.4
7.0  364.8
9.5 1522.3
9.9  585.1
10.9  571.7  <---
11.2  578.6
11.3  666.1
    ...
```

Keypoints in Image 2

```
6.3 1749.0
8.4 1753.3
8.9  497.9
10.4  540.9
11.0  683.2
11.0  687.8
11.1  589.8
11.3  583.4  <---
12.1 1212.6
12.2  949.3
     ...
```

Correspondences

```
 4 7 <---
 5 18285
11 27631
  ...
```

Calibrated Epipolar Geometry

- Two views of the scene are related by a so-called essential matrix $\mathbf{E}$:

$$\mathbf{E} = [-\mathbf{t}]_\times \mathbf{R},$$

  where $\mathbf{R}$ and $\mathbf{t}$ are the rotation matrix and the translation vector from the second view to the first view, respectively.
- All images were taken by the same perspective camera with known calibration matrix $\mathbf{K}$.
- Given two corresponding homogeneous image coordinates $\underline{\mathbf{x}} \in I_1$ and $\underline{\mathbf{y}} \in I_2$, we get their **normalized coordinates** $\underline{\mathbf{x}}'$ and $\underline{\mathbf{y}}'$ as follows:

$$\underline{\mathbf{x}}' = \mathbf{K}^{-1}\underline{\mathbf{x}} \quad \text{and} \quad \underline{\mathbf{y}}' = \mathbf{K}^{-1}\underline{\mathbf{y}}.$$

  Then it must hold that:

$$\underline{\mathbf{y}}'^{\top}\mathbf{E}\,\underline{\mathbf{x}}' = 0.$$

- Acquired correspondences are tentative – can contain outliers and noise; use robust estimation (RANSAC/MLESAC).

Algorithm Overview

- Transform all keypoints from both images by (the common) $\mathbf{K}^{-1}$ to get the normalized coordinates

$$\mathbf{K} = \begin{pmatrix} 2080 & 0 & 1421 \\ 0 & 2080 & 957 \\ 0 & 0 & 1 \end{pmatrix}.$$

- RANSAC iteration:
  1. Randomly sample five correspondences
  2. Estimate potential essential matrices (multiple solutions)
  3. Decompose each $\mathbf{E}$ into $(\mathbf{R}, \mathbf{t})$ (four combinations)
  4. Chirality: select the pair of rotation $\mathbf{R}$ and translation $\mathbf{t}$, such that all five reconstructed 3D points are in front of both cameras (at most one solution $(\mathbf{R}, \mathbf{t})$ for each essential matrix).
  5. For a given $\mathbf{E}$ and known $\mathbf{K}$ compute support using Sampson error with original points

Note: From one set of five corresponding points, we get multiple hypothesis $(\mathbf{E}, \mathbf{R}, \mathbf{t})$, each needs to be verified separately

Hypothesis Generation – Essential Matrix Estimation

- The matrix $\mathbf{E}$ (up to scale) has five degrees of freedom and can be estimated from five normalized correspondences using a so-called 5-point algorithm. Install the p5 package using PIP.

Hint: Install the external C++ library P5 with PIP

- Use PIP package manager to instal p5:

```
python3 -m pip install http://cmp.felk.cvut.cz/cmp/courses/TDV/code/python-p5-1.0.tar.gz
```

- To use the function `p5.p5gb(...)` in your code, simply:

```
import p5
```

- Randomly sample five normalized correspondences $\mathbf{x}'^{(i)} \sim \mathbf{y}'^{(i)}$ and use the 5-point algorithm to estimate the essential matrix.
  Multiple solutions may be returned.

Hint: Python

```
import p5
...
# Sample five normalized correspondences xs∼ys
Es=p5.p5gb(e2p(xs), e2p(ys))
```

Hypothesis Generation – Essential Matrix Decomposition

- Each of the returned estential matrices yields one of four possible combinations of rotation and translation. Here, we will refresh the steps of the decomposition method from the lecture, see the slides for more details.
- Given an essential matrix $\mathbf{E}$:

  1. Compute SVD of $\mathbf{E} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$ and verify that $\mathbf{D} = \lambda \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \lambda \neq 0.$
  2. Ensure that $\mathbf{U}$ and $\mathbf{V}$ are rotation matrices, set:
  $$\mathbf{U} = \det(\mathbf{U}) \cdot \mathbf{U} \quad \text{and} \quad \mathbf{V} = \det(\mathbf{V}) \cdot \mathbf{V}$$
  3. Compute the possible rotation matrix and translation vector as:
  $$\mathbf{R}(\alpha) = \mathbf{U} \begin{pmatrix} 0 & \alpha & 0 \\ -\alpha & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{V}^\top, \quad \mathbf{t}(\beta) = -\beta \mathbf{U}_3,$$
  where $|\alpha| = 1$, $\beta \neq 0$ and $\mathbf{U}_3$ is the third column of $\mathbf{U}$.
- This decomposition yields four possible combinations of $\mathbf{R}$ and $\mathbf{t}$ for combinations of $\alpha = \pm 1$ and $\beta = \pm 1$.

Hypothesis Generation – Chirality Constraint

- Set the first camera as canonical, i.e., $\mathbf{P}_1 = \mathbf{K}\begin{bmatrix}\mathbf{I} & \mathbf{0}\end{bmatrix}$, then the second one is defined as $\mathbf{P}_2 = \mathbf{K}\begin{bmatrix}\mathbf{R}(\alpha) & \mathbf{t}(\beta)\end{bmatrix}$. Given the (same) five sampled correspondences $\mathbf{x}^{(i)} \sim \mathbf{y}^{(i)}$ $(i = 1, ..., 5)$, we perform triangulation with numerical conditioning:
  1. For the correspondence $\mathbf{x}^{(i)} \sim \mathbf{y}^{(i)}$ construct the system of linear equations:

$$\mathbf{D} = \begin{bmatrix} x_1^{(i)}(\mathbf{p}_3^1)^\top - (\mathbf{p}_1^1)^\top \\ x_2^{(i)}(\mathbf{p}_3^1)^\top - (\mathbf{p}_2^1)^\top \\ y_1^{(i)}(\mathbf{p}_3^2)^\top - (\mathbf{p}_1^2)^\top \\ y_2^{(i)}(\mathbf{p}_3^2)^\top - (\mathbf{p}_2^2)^\top \end{bmatrix}, \text{ where } (\mathbf{p}_i^j)^\top \text{ is the } i\text{-th row of } \mathbf{P}_j.$$

  2. Re-scale the problem by a regular diagonal conditioning matrix $\mathbf{S} \in \mathbb{R}^{4\times 4}$, then:

$$\mathbf{0} = \mathbf{D}\underline{\mathbf{X}} = \mathbf{D}\mathbf{S}\mathbf{S}^{-1}\underline{\mathbf{X}}.$$

  3. Solve $\mathbf{D}\mathbf{S}\underline{\mathbf{X}}' = 0$ for $\underline{\mathbf{X}}'$.
  4. Compute the 3D point $\underline{\mathbf{X}} = \mathbf{S}\underline{\mathbf{X}}'$.

  (see slides from the lecture for more details)

Hint: Python

```
S=np.diag(1/np.max(np.abs(D), axis=0))
u,_,_=np.linalg.svd((D@S).T@(D@S))
X = (S@u[:, -1]) / (S[-1, :]@u[:, -1])
```

- We then select such a combination of $\mathbf{R}$ and $\mathbf{t}$, so that all points are in front of both cameras. I.e.: $(\mathbf{P}_1\underline{\mathbf{X}}^{(i)})_3 > 0 \ \wedge \ (\mathbf{P}_2\underline{\mathbf{X}}^{(i)})_3 > 0, \forall i = 1, ..., 5.$

Consensus Verification

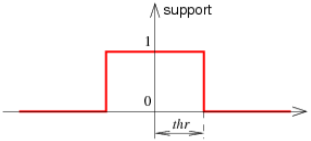
- Compute the fundamental matrix $\mathbf{F}$ from the found essential matrix $\mathbf{E}$ (to estimate errors in pixels):

$$\mathbf{F} = (\mathbf{K}^\top)^{-1}\,\mathbf{E}\,\mathbf{K}^{-1}.$$

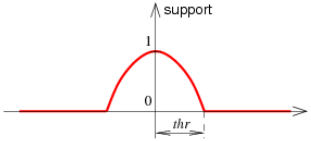
- To verify the quality of each hypothesis, we will use the Sampson error. For a 2D correspondence $\mathbf{x} \sim \mathbf{y}$, its squared Sampson error is defined as:

$$\varepsilon_{\mathbf{F}}(\mathbf{x},\mathbf{y})^2 = \frac{(\underline{\mathbf{y}}^\top \mathbf{F}\underline{\mathbf{x}})^2}{||\mathbf{S}\mathbf{F}\underline{\mathbf{x}}||^2 + ||\mathbf{S}\mathbf{F}^\top\underline{\mathbf{y}}||^2}, \text{ where } \quad \mathbf{S} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}. \tag{1}$$

- Use RANSAC (left) or MLESAC (right) support function.



$$s_i = \begin{cases} 1 & \text{if } \varepsilon_{\mathbf{F}}(x^{(i)}, y^{(i)})^2 \leq \theta^2 \\ 0 & \text{otherwise} \end{cases}$$



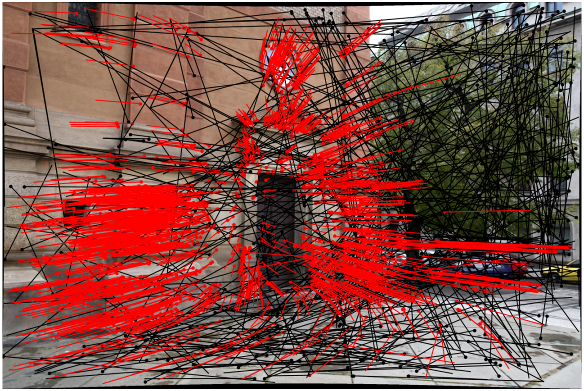
$$s_i = \begin{cases} 1 - \frac{\varepsilon_{\mathbf{F}}(x^{(i)}, y^{(i)})^2}{\theta^2} & \text{if } \varepsilon_{\mathbf{F}}(x^{(i)}, y^{(i)})^2 \leq \theta^2 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{support} = \sum_i s_i$$

Expected Results: Inlier Correspondences

Visualize the inliers (red) and outliers (black) of the estimated essential matrix $\mathbf{E}^*$ as a needle map. Given one selected correspondence $\mathbf{x} \sim \mathbf{y}$, its needle visualization on the first image can be done as follows:

```
plt.scatter(x[0], x[1], 20, 'r')
plt.plot([x[0], y[0]], [x[1], y[1]], 'r-', linewidth=2)
```

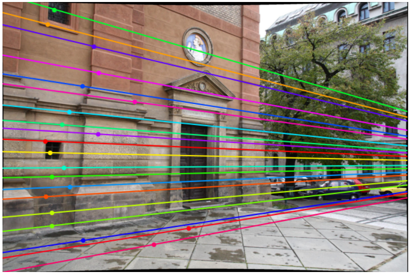


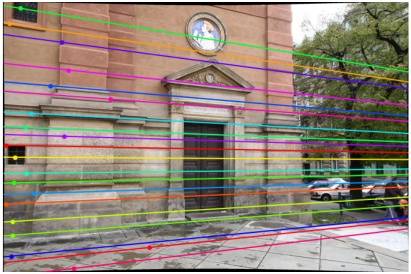
Expected Results: Epipolar Lines

Visualize a subset of inliers as corresponding points and epipolar lines of the estimated fundamental matrix $\mathbf{F}^* = (\mathbf{K}^\top)^{-1}\,\mathbf{E}^*\,\mathbf{K}^{-1}$ (use the same color for corresponding points and lines). Given one selected inlier 2D correspondence $\mathbf{x} \sim \mathbf{y}$, we get:

$$\mathbf{l}_1 = \mathbf{F^*}^\top \mathbf{y} \quad \text{and} \quad \mathbf{l}_2 = \mathbf{F}^* \mathbf{x},$$

where $\mathbf{l}_1$ is the epipolar line in the first image and $\mathbf{l}_2$ is the corresponding epipolar line in the second image.





Toolbox Functions

- For this assignment, you should implement three toolbox functions, which will be needed throughout the term project:
  1. Linear triangulation with numerical conditioning
  2. Essential matrix decomposition with chirality constraint
  3. Sampson error estimation
- You can check your implementations of these toolbox functions through automatic evaluation in BRUTE. In case of a mistake, you will be prompted with the anticipated result for a given input.

Toolbox

`X = Pu2X(P1, P2, u1, u2)`
Reconstructs n 3D points `X` from n corresponding 2D points `u1` and `u2` observed by two cameras `P1` and `P2`, respectively. See slide 10. Inputs: `P1, P2` $\in \mathbb{R}^{3\times 4}$; `u1, u2` $\in \mathbb{R}^{3\times n}$. Output: `X` $\in \mathbb{R}^{4\times n}$.

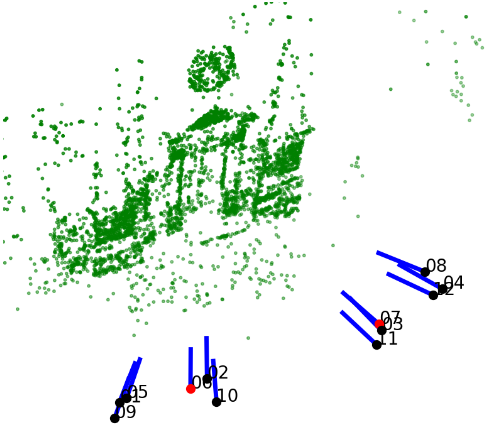
`[R, t] = EutoRt(E, u1, u2)`
Decomposes the given essential matrix `E` into rotation `R` and translation `t` (see slide 9). It returns the combination, which fulfills the chirality constraint for n points `u1` and `u2` (see slide 10). If the chirality fails, it returns `R = []`.
Inputs: `E` $\in \mathbb{R}^{3\times 3}$; `u1, u2` $\in \mathbb{R}^{3\times n}$. Outputs: `R` $\in \mathbb{R}^{3\times 3}$; `t` $\in \mathbb{R}^{3\times 1}$.

`err = err_F_sampson(F, u1, u2)`
Returns the **squared** (less computationally intensive) Sampson error for n homogeneous corresponding coordinates `u1`$\sim$`u2`. See (1). Inputs: `F` $\in \mathbb{R}^{3\times 3}$; `u1, u2` $\in \mathbb{R}^{3\times n}$. Output: `err` $\in \mathbb{R}^{1\times n}$.

Cameras and Scene Structure

Term project phases:

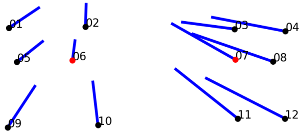
1. Calibrated epipolar geometry
2. **Cameras and scene structure**
3. Stereo reconstruction



Task Outline

Task: **Estimate projection matrices and reconstruct a sparse point cloud.**

1. Optimize the estimated rotation and translation between the first pair of images w.r.t. Sampson error on inliers.
2. Use the Sampson correction on 2D inlier correspondences and then reconstruct a sparse point cloud from them.
3. Download the rest of images, keypoints and correspondences (archive).
4. Iteratively append new cameras using the greedy stepwise gluing.
5. Show the estimated location and optical axis of each camera and (small subset of) 3D points.
6. Create the reconstructed sparse point cloud as `ply/vrml`.



Extrinsics Optimization on the First Pair

- Given estimated $\mathbf{R}^*$ and $\mathbf{t}^*$ from the previous assignment, find rotation $\mathbf{R}_{opt}$ and translation $\mathbf{t}_{opt}$ so that the Sampson error on **inliers** is minimized.
- To ensure faster and stable optimization, look for a refinement $\mathbf{R}_r$ for $\mathbf{R}^*$ and increment $\mathbf{t}_i$ for $\mathbf{t}^*$. These should be applied in the error function as:

$$\mathbf{R}_{opt} = \mathbf{R}^* \, \mathbf{R}_r \text{ and } \mathbf{t}_{opt} = \mathbf{t}^* + \mathbf{t}_i .$$

- $\mathbf{R}_r$: Use Rodrigues' rotation formula (parameters $\mathbf{\Phi} \in \mathbb{R}^3$)[1]:

$$\mathbf{R}_r(\mathbf{\Phi}) = \mathbf{I} + \frac{\sin(\varphi)}{\varphi}\,[\mathbf{\Phi}]_\times + \frac{1-\cos(\varphi)}{\varphi^2}\,[\mathbf{\Phi}]_\times^2 \text{, where } \varphi = ||\mathbf{\Phi}||.$$

- $\mathbf{t}_i$: Use linear combination of two orthogonal vectors (parameters $a, b \in \mathbb{R}$):

$$\mathbf{t}_i = a\mathbf{p} + b\mathbf{q}, \text{ where } \mathbf{p}, \mathbf{q} \in N(\mathbf{t}^*).$$

- Normalize $\mathbf{t}_{opt}$, so that the length of base is unity.

Hint: PYTHON

```
N = scipy.linalg.null_space(t_rans.reshape((1, 3)))
p, q = N[:, 0], N[:, 1]
```

[1]Note: $\lim_{\varphi\to 0} \frac{\sin(\varphi)}{\varphi} = 1$ and $\lim_{\varphi\to 0} \frac{1-\cos(\varphi)}{\varphi^2} = \frac{1}{2}$

Triangulation Once Again: The Golden Standard Method

- Triangulate a 2D correspondence $\mathbf{x} \sim \mathbf{y}$ observed by cameras $\mathbf{P}_1$ and $\mathbf{P}_2$:
  1. Given $\mathbf{P}_1 = \begin{bmatrix}\mathbf{Q}_1 & \mathbf{q}_1\end{bmatrix}$ and $\mathbf{P}_2 = \begin{bmatrix}\mathbf{Q}_2 & \mathbf{q}_2\end{bmatrix}$, estimate the fundamental matrix:
  $$\mathbf{F} = (\mathbf{Q}_1\mathbf{Q}_2^{-1})^\top \left[\mathbf{q}_1 - (\mathbf{Q}_1\mathbf{Q}_2^{-1})\mathbf{q}_2\right]_\times .$$
  2. Use Sampson correction:
  $$\begin{bmatrix}\mathbf{x}_c \\ \mathbf{y}_c\end{bmatrix} = \begin{bmatrix}\mathbf{x} \\ \mathbf{y}\end{bmatrix} - \frac{\underline{\mathbf{y}}^\top \mathbf{F}\underline{\mathbf{x}}}{||\mathbf{S}\mathbf{F}\underline{\mathbf{x}}||^2 + ||\mathbf{S}\mathbf{F}^\top\underline{\mathbf{y}}||^2} \begin{bmatrix}(\mathbf{F}_1)^\top \underline{\mathbf{y}} \\ (\mathbf{F}_2)^\top \underline{\mathbf{y}} \\ (\mathbf{F}^1)^\top \underline{\mathbf{x}} \\ (\mathbf{F}^2)^\top \underline{\mathbf{x}}\end{bmatrix},$$
  where $\mathbf{F}_i$ is i-th column of $\mathbf{F}$ and $\left(\mathbf{F}^i\right)^\top$ is i-th row of $\mathbf{F}$.
  3. Use SVD triangulation algorithm with numerical conditioning from slide 10 on $\mathbf{x}_c \sim \mathbf{y}_c$ with cameras $\mathbf{P}_1$ and $\mathbf{P}_2$.

Toolbox

`[F] = PP2F(P1, P2)`
Estimates the fundamental matrix F from two projection matrices P1 and P2.
Inputs: P1, P2 $\in \mathbb{R}^{3\times 4}$.
Outputs: F $\in \mathbb{R}^{3\times 3}$.

`[S] = sqc(x)`
Estimates the skew-symmetric matrix for cross-product.
Inputs: x $\in \mathbb{R}^{3\times 1}$. Outputs: S $\in \mathbb{R}^{3\times 3}$.

Toolbox

`[nu1, nu2] = u_correct_sampson(F, u1, u2)`
Estimates sampson-corrected coordinates based on the given fundamental matrix F from image points in homogeneous corrdinates u1∼u2.
Inputs: F $\in \mathbb{R}^{3\times 3}$; u1, u2 $\in \mathbb{R}^{3\times n}$. Outputs: nu1, nu2 $\in \mathbb{R}^{3\times n}$.

Initial Sparse Point Cloud Reconstruction

- Use the triangulation method from previous slide on inlier 2D correspondences between the first pair of cameras with projection matrices $\mathbf{P}_1 = \mathbf{K}\begin{bmatrix}\mathbf{I} & \mathbf{0}\end{bmatrix}$ and $\mathbf{P}_2 = \mathbf{K}\begin{bmatrix}\mathbf{R}_{opt} & \mathbf{t}_{opt}\end{bmatrix}$.
- Some of these points might still be false matches and not the actual inliers. Let us thus keep only points that fulfill chirality (are in front of both cameras) and apical angle constraints.
- Given a reconstructed 3D point $\mathbf{X}$, we want it to have a sufficient apical angle, i.e., larger than some threshold θ_α:

$$\alpha = \mathrm{acos}\left(\frac{(\mathbf{C}_1 - \mathbf{X})^\top (\mathbf{C}_2 - \mathbf{X})}{||\mathbf{C}_1 - \mathbf{X}||\,||\mathbf{C}_2 - \mathbf{X}||}\right) \geq \theta_\alpha,$$

where $\mathbf{C}_i$ is the location of the camera i in the world coordinate system.

Toolbox

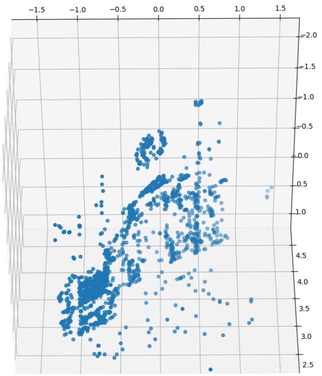
`[mask] = X_chirality_apical_angle(X, R_w1, t_w1, R_w2, t_w2, thr_a)`

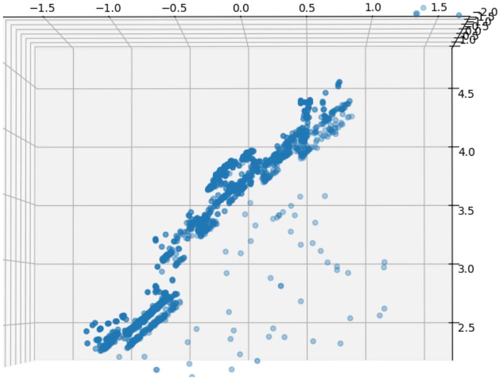
Returns a boolean array. It indicates 3D points from `X` that lie in front of cameras (`R_w1`, `t_w1`) and (`R_w2`, `t_w2`). The angle between two vectors from `X` to each camera location must also be greater or equal to `thr_a`.

Inputs: `X` $\in \mathbb{R}^{4\times n}$; `R_w1, R_w2` $\in \mathbb{R}^{3\times 3}$; `t_w1, t_w2` $\in \mathbb{R}^{3\times 1}$. Outputs: `mask` $\in \mathbb{B}^{1\times n}$.

Initial Sparse Point Cloud Reconstruction

- Use the triangulation method from previous slide on inlier 2D correspondences between the first pair of cameras with projection matrices $\mathbf{P}_1 = \mathbf{K}\begin{bmatrix}\mathbf{I} & \mathbf{0}\end{bmatrix}$ and $\mathbf{P}_2 = \mathbf{K}\begin{bmatrix}\mathbf{R}_{opt} & \mathbf{t}_{opt}\end{bmatrix}$.





Working With Correspondences

- Download the rest of images, keypoints and correspondences (archive).
- This assignment requires to do some extensive work with tentative correspondences, as these need to propagate from 2D-2D (image-to-image) into 3D-2D (scene-to-image). We have prepared a corresp.py package, which can help you with that.
  - You can look at the API of the package and some more thorough description and visualization in the documentation here.
- Start by loading all the tentative image-to-image correspondences into the object.

Hint: Python

```
import corresp
...
NUM_OF_IMAGES = 12
corr = corresp.Corresp(NUM_OF_IMAGES)
for i in range(0, NUM_OF_IMAGES):
    for j in range(i+1, NUM_OF_IMAGES):
        corr.add_pair(i, j, np.loadtxt(
                              'scene_1/corresp/m_{:02d}_{:02d}.txt'.format(i+1, j+1)
                                      ).astype(dtype=np.int32)
                    )
```

Initializing First Pair and Sparse Point Cloud

- Initialize the `Corresp` object with the filtered inliers from the first pair. It automatically removes non-consistent image-to-image tentative correspondences and establishes potential scene-to-image correspondences.

Hint: Python

```
i1, i2 = 0, 1 # indices of the first pair of cameras

P = np.zeros((NUM_OF_IMAGES, 3, 4))
P[i1,:,:] = K @ np.hstack([np.eye(3), np.zeros((3, 1))])
P[i2,:,:] = K @ np.hstack([R_opt, t_opt[:, None]])

F = toolbox.PP2F(P[i1], P[i2])
[nu1_h, nu2_h] = toolbox.u_correct_sampson(F, u1_h[:, inl_rans], u2_h[:, inl_rans])
X_pcl = toolbox.Pu2X(P[i1], P[i2], nu1_h, nu2_h)
inl_chir_api = toolbox.X_chirality_apical_angle(X_pcl, np.eye(3), np.zeros((3,)),
                                                 R_opt, t_opt, thr_a)
X_pcl = X_pcl[:, inl_chir_api] # initial filtered point cloud

idx = np.where(inl_rans)[0][inl_chir_api]
corr.start(i1, i2, idx)
```

Stepwise Camera Gluing

- There are several different approaches to recover cameras extrinsics (rotations and translations). We will use a simple greedy algorithm. The method has been described in lectures (see *Reconstructing Camera System by Gluing Camera Triples*).
- We have already initialized the set of cameras with the first pair and reconstructed the initial point cloud. Now, we need to repeatedely append cameras one-by-one, as follows:
  1. Select new camera, which is suitable for appending based on some (manually chosen) heuristic.
  2. Use robust estimation to find the best (w.r.t. reprojection error) parameters $(\mathbf{R}, \mathbf{t})$ from tentative 3D-2D correspondences (P3P algorithm).
  3. Optimize the estimated extrinsics on inlier 3D-2D correspondences w.r.t. reprojection error.
  4. Reconstruct 3D points between the new camera and all other cameras with already known projection matrices. Add inlier 3D points (fulfilling chirality and apical angle constraints) into the sparse point cloud.
  5. Verify the newly reconstructed points in other cameras.

Estimation of a New Camera

- Select a new camera to append:
  - e.g., the one with the most tentative scene-to-image correspondences

Hint: Python

```
ig, _ = corr.get_green_cameras()
Xu_count, _ = corr.get_Xucount(ig)
iNew = ig[np.argmax(Xu_count)]
```

- Get tentative 3D-2D correspondences between the sparse point cloud and the new camera image.

Hint: Python

```
[X_idx, u_idx, _] = corr.get_Xu(iNew)
```

- Estimate the global pose and orientation of this new camera using the P3P algorithm (our implementation).

Hint: Python

```
import p3p
...
# Let us have tentative correspondences between X_pcl[:, X_idx]∼u[iNew][:, u_idx]
# Sample three random correspondences Xw∼uc_h in homogeneous coordinates
X_cam_sols = p3p.p3p_grunert(Xw, K_inv @ uc_h) # multiple solutions to P3P problem
for X_cam in X_cam_sols:
    R_new, t_new = p3p.XX2Rt_simple(Xw, X_cam) # get (R_new, t_new) for each solution
```

- Use RANSAC scheme with reprojection error on all tentative scene-to-image correspondences to find the best $(\mathbf{R}, \mathbf{t})$.

New Camera Refinement

- Perform a similar optimization of the rotation $\mathbf{R}$ and translation $\mathbf{t}$ as on Slide 17. Use the reprojection error on **inlier** 3D-2D correspondences between the pointcloud and image keypoints. Here, the scale is known (from the first pair), hence the translation increment is simply $\mathbf{t}_i \in \mathbb{R}^3$.

Hint: Python

```
def err_reproj(params, Xs, us, R_rans, t_rans, K):
    R = R_rans @ rodrigues(params[0:3])
    t = t_rans + params[3:6]
    X_proj = toolbox.p2e(K @ np.hstack([R, t[:, None]]) @ Xs)
    return np.sum(np.linalg.norm(X_proj - us, axis=0))
```

- Save the new camera into the array:

Hint: Python

```
params = fmin(err_reproj, np.zeros((6,)), args=(X_pcl[:, X_idx][:, inl_rans],
                              u[iNew][:, u_idx][:, inl_rans], R_rans, t_rans, K))
R_opt = R_rans @ rodrigues(params[0:3])
t_opt = t_rans + params[3:6]
P[iNew, :, :] = K @ np.hstack([R_opt, t_opt[:, None]])
corr.join_camera(iNew, np.nonzero(inl_rans)[0])
```

New 3D Points Reconstruction

- Iterate over already known cameras and get tentative image-to-image correspondences, which were not used yet:

Hint: Python

```
ilist = corr.get_cneighbours(iNew)
for ic in ilist:
    [miNew, mic] = corr.get_m(iNew, ic)
```

- Reconstruct new 3D points:
  1. Estimate the fundamental matrix from the two projection matrices (see Slide 18).
  2. Use Sampson error to find actual inlier 2D-2D correspondences.
  3. Use projection matrices to triangulate the inliers using the golden standard method from Slide 18.
  4. Choose only points that fulfill the chirality and apical angle constraints (see Slide 19).
- Add new 3D points into the point cloud and update the correspondences.

Hint: Python

```
X_pcl = np.concatenate((X_pcl, X_new_h[:, inl_chir_api]), axis=1)
corr.new_x(iNew, ic, np.where(inl_samps)[0][inl_chir_api])
```

Verify Consistency of New Points with Other Cameras

- Newly reconstructed points from previous slide might yeild a potential new 3D-2D correspondence with some other (already known) camera. We need to verify them, before continuing the gluing.[2]
- Iterate over all cameras and verify new tentative 3D-2D correspondences:

Hint: Python

```
ilist = corr.get_selected_cameras()
for ic in ilist:
    [X_idx, u_idx, Xu_verified] = corr.get_Xu(ic)
    # Find inliers based on the reprojection error between
    # X_pcl[:, X_idx][:, ~Xu_verified]∼u[ic][:, u_idx][:, ~Xu_verified]
    ...
    corr.verify_x(ic, np.where(~Xu_verified)[0][inl])
```

- Finalize the appending of a new camera:

Hint: Python

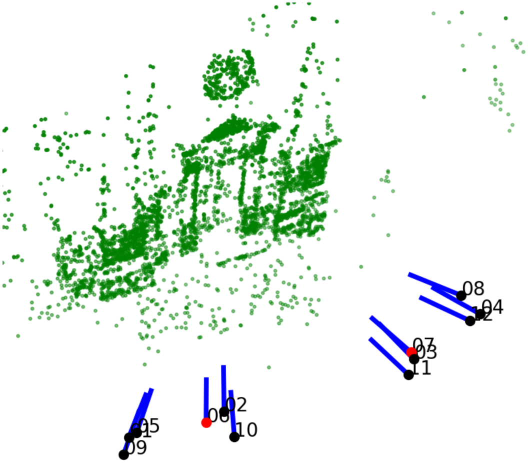
```
corr.finalize_camera()
```

- Repeat from Slide 23, if there is still some unknown camera.

[2]Note: This verification is mainly needed, if you want to check the tentative 3D-2D correspondences, e.g., before bundle adjustment. In this term project, you can skip this step by calling `corr.verify_x(ic, [])` on all known cameras.

Expected Results: Cameras Locations and Optical Axes

- Show the location of each camera $\mathbf{C}_i = -\mathbf{R}_i^\top \mathbf{t}_i$ with its optical axis $(\mathbf{R}_i)_3$. Show the initial pair of cameras in different color. Show (small subset of) the sparse point cloud as well.



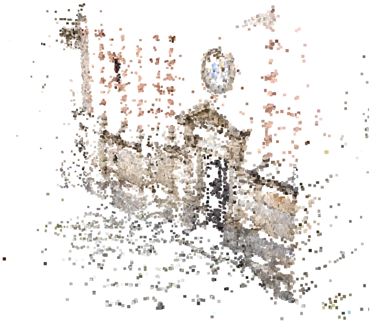
Expected Results: Sparse Point Cloud

- Save the final sparse point cloud in a `.ply` file, using geom_export library.

Hint: Python

```
import ge
g = ge.GePly('out_sparse.ply')
g.points(toolbox.p2e(X_pcl)) # You can add color as a second argument 3xn
g.close()
```

- Use e.g., MeshLab to view the saved point cloud and show it to us.
- Please, upload the sparse point cloud in BRUTE as well (task `E_sparse`).



Stereo Reconstruction

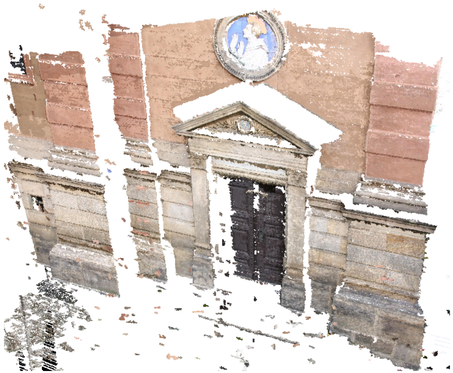
Term project phases:

1. Calibrated epipolar geometry
2. Cameras and scene structure
3. **Stereo reconstruction**

Task Outline

Task: **Reconstruct a dense point cloud using stereo matching.**

1. Select image pairs.
2. Rectify the pairs to align their epipolar lines.
3. Use a stereo matching method to compute disparity maps.
4. Reconstruct a dense 3D point cloud from the maps.
5. Save the reconstructed point cloud as `ply/vrml`.



Stereo Rectification

- Let us have two images I_1 and I_2 with their projection matrices:

$$\mathbf{P}_1 = \mathbf{K}\begin{bmatrix}\mathbf{R}_1 & \mathbf{t}_1\end{bmatrix} \quad \text{and} \quad \mathbf{P}_2 = \mathbf{K}\begin{bmatrix}\mathbf{R}_2 & \mathbf{t}_2\end{bmatrix}.$$

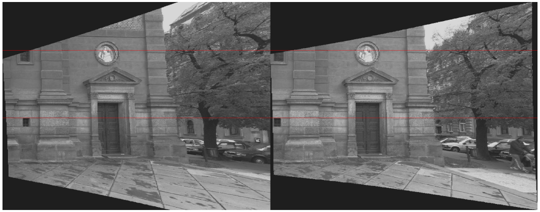
- We can estimate the relative rotation and translation of the pair:

$$\mathbf{R}_{1\rightarrow 2} = \mathbf{R}_2\mathbf{R}_1^\top \quad \text{and} \quad \mathbf{t}_{1\rightarrow 2} = -\mathbf{R}_2\mathbf{R}_1^\top\mathbf{t}_1 + \mathbf{t}_2.$$

- Use our code to receive rectification homographies $\mathbf{H}_1$ and $\mathbf{H}_2$ and rectified images:

```
[H1, H2, img1Rect, img2Rect] = rectify.rectify(img1, img2, K,  R12, t12)
```

- Verify that the rectification aligned rows of images:



Stereo Matching

- Crop images to have the same size:

```
h = min(img1Rect.shape[0], img2Rect.shape[0])
w = min(img1Rect.shape[1], img2Rect.shape[1])
img1Rect, img2Rect = img1Rect[:h, :w], img2Rect[:h, :w]
```

- Select apropriate disparity range (for each pair):

```
min_disp, max_disp = -1000, 500
```

- Initialize SGBM stereo matching algorithm from OpenCV:

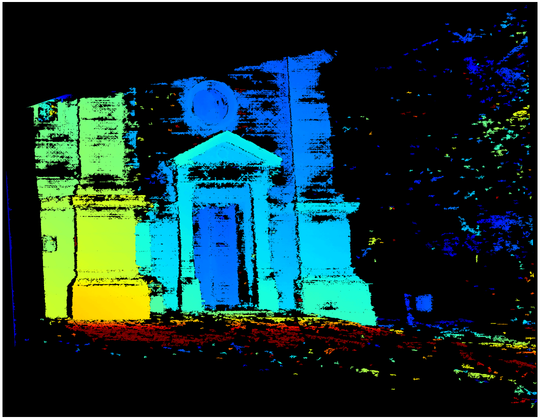
```
num_disp = int(2**np.ceil(np.log2(max_disp - min_disp)))
stereo = cv2.StereoSGBM.create(numDisparities=num_disp, minDisparity=min_disp,
                               blockSize=3, uniquenessRatio=30,
                               speckleWindowSize=100, speckleRange=5,
                               disp12MaxDiff=1, P1=8*3*3**2, P2=32*3*3**2)
```

- Add zero-padding and compute the stereo disparity map:

```
pad1 = np.zeros((h, max(0, num_disp+min_disp)), dtype=img1Rect.dtype)
pad2 = np.zeros((h, max(0, -min_disp)), dtype=img1Rect.dtype)
img1Rect = np.hstack((pad1, img1Rect, pad2))
img2Rect = np.hstack((pad1, img2Rect, pad2))
disparity = stereo.compute(img1Rect,img2Rect)[:, pad1.shape[1] : -pad2.shape[1]]
```

Stereo Matching

- Visualize the disparity map (remove invalid disparities):



- SGBM returns disparities in `int16` with 4-bit fixed decimal point, hence we need to divide the disparity by 16:

```
disparity = disparity.astype(float) / 16.0
```

Dense Point Cloud Reconstruction

- Given a 2D point $\begin{bmatrix} u & v & 1 \end{bmatrix}^\top$ in the rectified image with a corresponding disparity d, we obtain a correspondence:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \begin{bmatrix} u - d \\ v \\ 1 \end{bmatrix}$$

  between rectified images.
- Use inverse homographies $\mathbf{H}_1^{-1}$ and $\mathbf{H}_2^{-1}$ to transform these corresponding points into the original images.
- Use the original image projection matrices $\mathbf{P}_1$ and $\mathbf{P}_2$ to triangulate the 3D point (see 10).
- Add the triangulated point into the cloud, if it fulfills the chirality and apical angle constraints (see 19).
- (Optional) Collect image color associated with each 3D point (project the point onto all images, collect colors and use element-wise median).

Expected Result: Dense Point Cloud

- Select all horizontal and vertical pairs of images and reconstruct the point cloud.
- Save the final dense point cloud in a `.ply` file, using geom_export library.

```
import ge
g = ge.GePly('out_dense.ply')
g.points(toolbox.p2e(X_pcl)) # You can add color as a second argument 3xn
g.close()
```

- Use e.g., MeshLab to view the saved point cloud and show it to us.
- Please, upload the dense point cloud in BRUTE (task `E_dense`).
- Upload all your term project code in BRUTE as well (task `E_codes`).