EE 046746 - Technion - Computer Vision

Tutorial 10 - Structure From Motion

Elias Nehme

Agenda

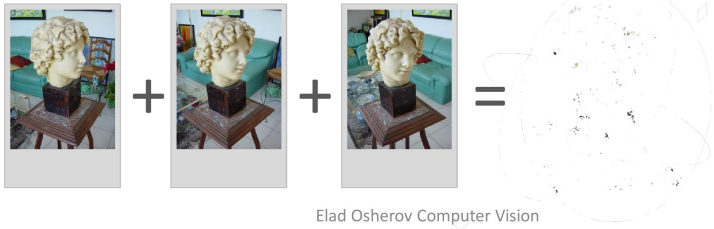
Steps

	Images → Points:	Structure from Motion
	Points → More points:	Multiple View Stereo
	Points → Meshes:	Model Fitting
+	Meshes → Models:	Texture Mapping
=	Images → Models:	Image-based Modeling

Elad Osherov Computer Vision

Steps

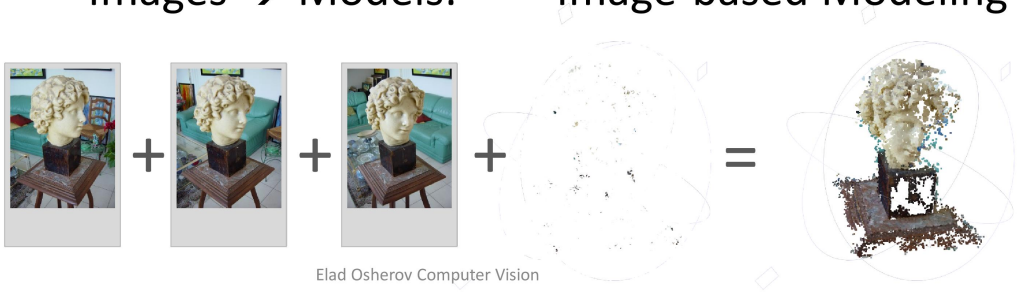
	Images → Points:	Structure from Motion
	Points → More points:	Multiple View Stereo
	Points → Meshes:	Model Fitting
+	Meshes → Models:	Texture Mapping
=	Images → Models:	Image-based Modeling



Elad Osherov Computer Vision

Steps

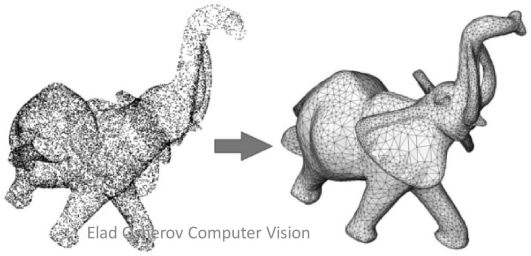
	Images → Points:	Structure from Motion
	Points → More points:	Multiple View Stereo
	Points → Meshes:	Model Fitting
+	Meshes → Models:	Texture Mapping
=	Images → Models:	Image-based Modeling



Elad Osherov Computer Vision

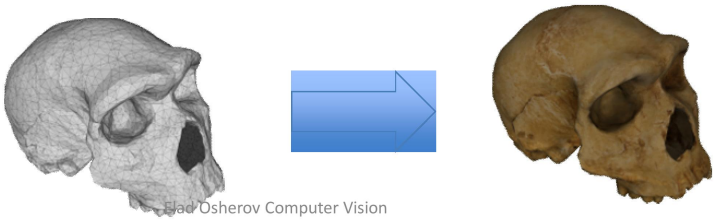
Steps

	Images → Points:	Structure from Motion
	Points → More points:	Multiple View Stereo
	Points → Meshes:	Model Fitting
+	Meshes → Models:	Texture Mapping
=	Images → Models:	Image-based Modeling



Steps

	Images → Points:	Structure from Motion
	Points → More points:	Multiple View Stereo
	Points → Meshes:	Model Fitting
+	Meshes → Models:	Texture Mapping
=	Images → Models:	Image-based Modeling



SfM Intro

- Reconstruction = Two-view Structure from Motion (SfM)

	Structure (scene geometry)	Motion (camera geometry)	Measurements
Pose Estimation	known	estimate	3D to 2D correspondences
Triangulation	estimate	known	2D to 2D coorespondences
Reconstruction	estimate	estimate	2D to 2D coorespondences

Pose Estimation

- In pose estimation, our goal is to estimate the pose of the camera - given by the camera matrix M.
  - We assume we have a set of matched points $\{P_i, p_i\}$ - that match points in the real world, in their homogeneous coordinates, to points in the image, in their homogeneous coordinates.
  - This means for each pair $\{P_i, p_i\}$, the perspective camera model holds:
$$p_i = MP$$
  - We've seen last tutorial that we can estimate M with SVD on N_p points.
    - How many N_p points?
  - We've seen we can rewrite M:
$$M = K\,[R|t] = K\,[R|-Rc] = [N|-Nc]$$
    - c can be found via SVD of M due to the relation: $Mc = 0$
    - Then N can be further decomposed into $N = KR$
    - This algorithm does not take into account noise, radial distortions, hard to impose prior knowledge (e.g. f), etc.
      - Solution? Minimize Reprojection errors = non-linear least squares

Pose Estimation

- Assuming the camera intrinsics are calibrated (K is known), how much DoF do we have?
- How much points N_p do we need in this case?
- This problem is usually referred to as Perspective-n-Point (PnP).
- Can be solved in openCV using `cv2.solvePnP()`

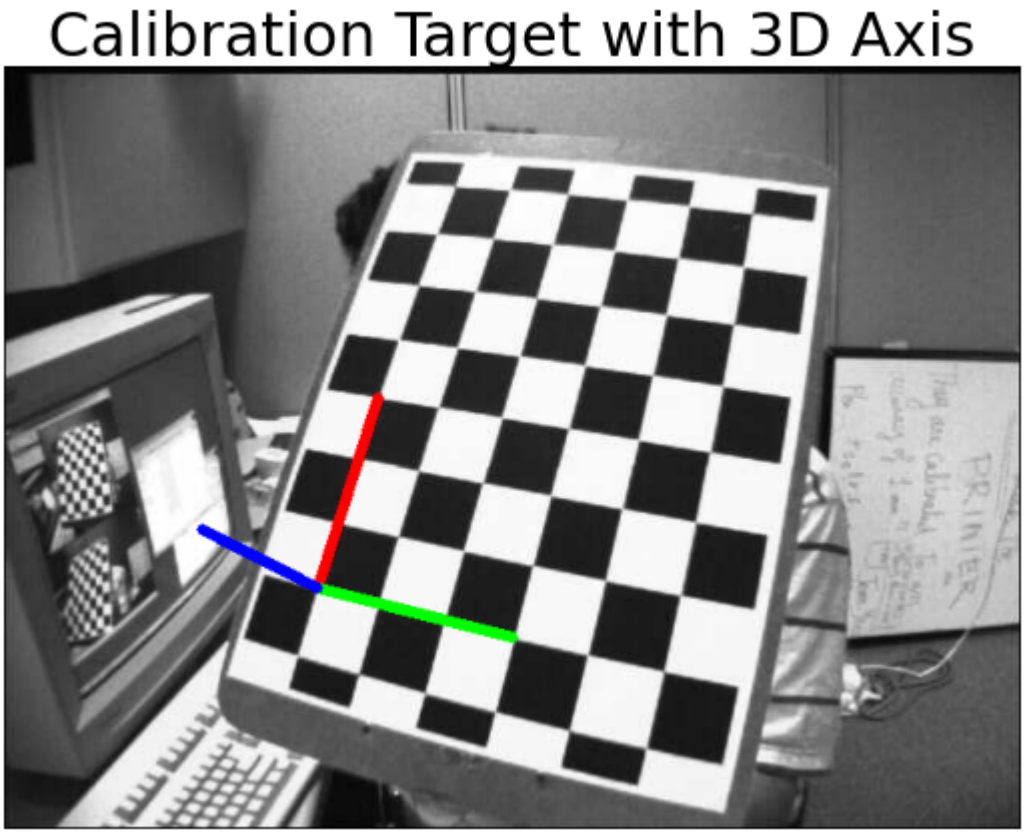
In [1]:
```
import numpy as np
import cv2
import glob

# load the camera calibration results from tut 9: K (mtx) and distortion coeff
s.
with np.load('./datasets/tut9_camera_calib.npz') as X:
    mtx, dist, _, _ = [X[i] for i in ('mtx','dist','rvecs','tvecs')]
```

In [2]:
```
# function that draws 3D axis at given image points starting from (0,0)
def draw(img, corners, imgpts):
    corner = tuple(corners[0].ravel())
    img = cv2.line(img, corner, tuple(imgpts[0].ravel()), (255,0,0), 5)
    img = cv2.line(img, corner, tuple(imgpts[1].ravel()), (0,255,0), 5)
    img = cv2.line(img, corner, tuple(imgpts[2].ravel()), (0,0,255), 5)
    return img
```

In [3]:
```
# termination criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
# prepare object pts
objp = np.zeros((6*7,3), np.float32)
objp[:,:2] = np.mgrid[0:7,0:6].T.reshape(-1,2)
# define points of 3D axis (length=3 & origin at first detected corner)
axis = np.float32([[3,0,0], [0,3,0], [0,0,-3]]).reshape(-1,3)
```

In [4]:
```
# plot 3D axis on each image (X = Blue, Y = Green, Z = Red)
k = 0
for fname in glob.glob('./assets/left*.jpg'):
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    ret, corners = cv2.findChessboardCorners(gray, (7,6),None)
    if ret == True:
        corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
        # Find the rotation and translation vectors.
        ret, rvecs, tvecs = cv2.solvePnP(objp, corners2, mtx, dist)
        # project 3D points to image plane
        imgpts, jac = cv2.projectPoints(axis, rvecs, tvecs, mtx, dist)
        img = draw(img,corners2,imgpts)
        k += 1
        if k == 8:
            imexample = img
        cv2.imshow('img',img)
        cv2.waitKey(500)
cv2.destroyAllWindows()
```

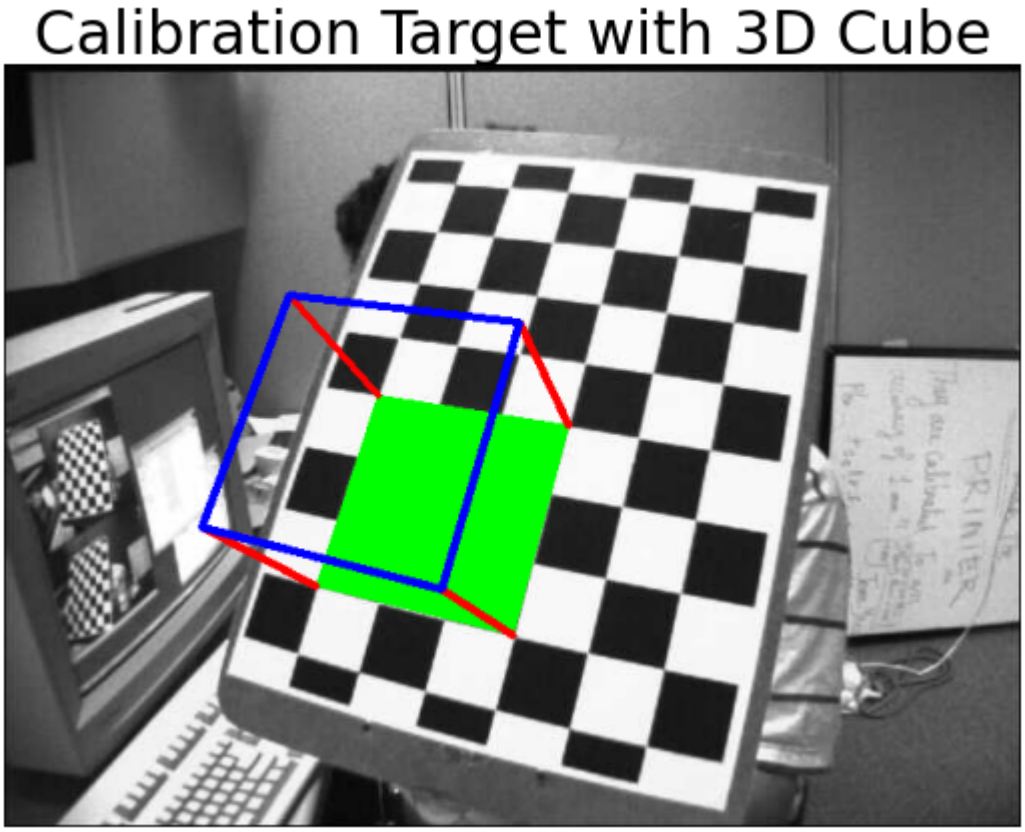
In [5]:
```
# show the last image and the projected axis
import matplotlib.pyplot as plt
plt.figure(figsize=(11,7))
plt.imshow(imexample)
plt.axis('off')
plt.title('Calibration Target with 3D Axis', fontsize=30)
plt.show()
```



In [6]:
```
# modified axis points to be corners of cube
axis = np.float32([[0,0,0], [0,3,0], [3,3,0], [3,0,0],
                   [0,0,-3],[0,3,-3],[3,3,-3],[3,0,-3] ])
# new drawing function
def draw(img, corners, imgpts):
    imgpts = np.int32(imgpts).reshape(-1,2)
    # draw ground floor in green
    img = cv2.drawContours(img, [imgpts[:4]],-1,(0,255,0),-3)
    # draw pillars in blue color
    for i,j in zip(range(4),range(4,8)):
        img = cv2.line(img, tuple(imgpts[i]), tuple(imgpts[j]),(255),3)
    # draw top layer in red color
    img = cv2.drawContours(img, [imgpts[4:]],-1,(0,0,255),3)
    return img
```

In [7]:
```
# plot 3D cube on each image (X = Blue, Y = Green, Z = Red)
k = 0
for fname in glob.glob('./assets/left*.jpg'):
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    ret, corners = cv2.findChessboardCorners(gray, (7,6),None)
    if ret == True:
        corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
        # Find the rotation and translation vectors.
        ret, rvecs, tvecs = cv2.solvePnP(objp, corners2, mtx, dist)
        # project 3D points to image plane
        imgpts, jac = cv2.projectPoints(axis, rvecs, tvecs, mtx, dist)
        img = draw(img,corners2,imgpts)
        k += 1
        if k == 8:
            imexample = img
        cv2.imshow('img',img)
        cv2.waitKey(500)
cv2.destroyAllWindows()
```

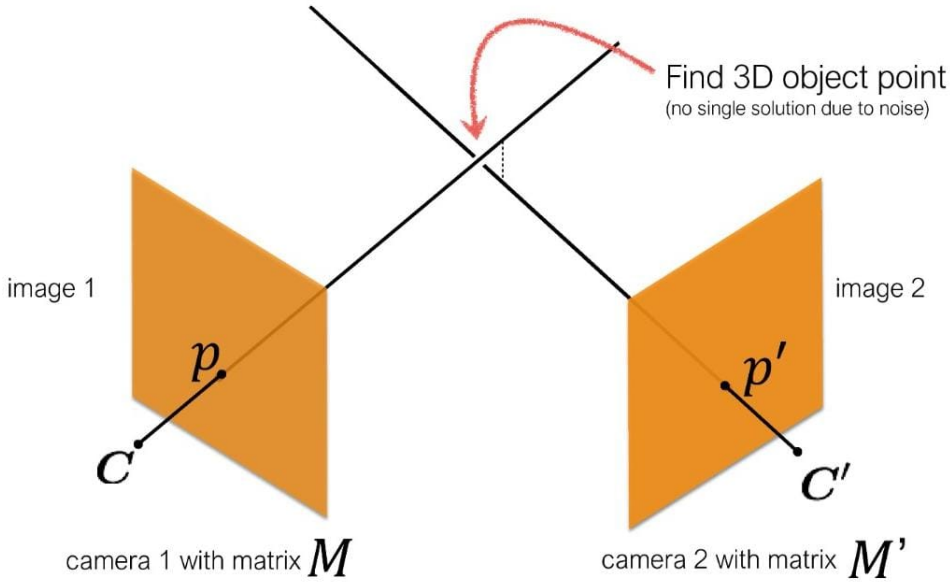
In [8]:
```
# show the last image and the projected cube
import matplotlib.pyplot as plt
plt.figure(figsize=(11,7))
plt.imshow(imexample)
plt.axis('off')
plt.title('Calibration Target with 3D Cube', fontsize=30)
plt.show()
```



Triangulation

- In triangulation, our goal is to estimate the structure of the scene (the location of a point P in the real world).
  - We assume we have a set of matched points $\{p_i, p_i'\}$ - that are matched homogeneous points in 2 different images.
  - We know the difference in geometry between the 2 cameras that took the images (their K, R and t, or respectively M and M').
  - So we know that the point in the real world, P, lies on both of the rays from the camera's centers to the points p, p':

$$p = MP, p' = M'P$$



Triangulation

- We remember $p = \alpha MP$ as well, which gives us:

$$p \times MP = 0, p' \times M'P = 0$$

- Switch to row-wise representation of the projection matrix:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \leftrightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} - & m_1^T & - \\ - & m_2^T & - \\ - & m_3^T & - \end{bmatrix} P$$

Triangulation

- Leads to two independent equations:

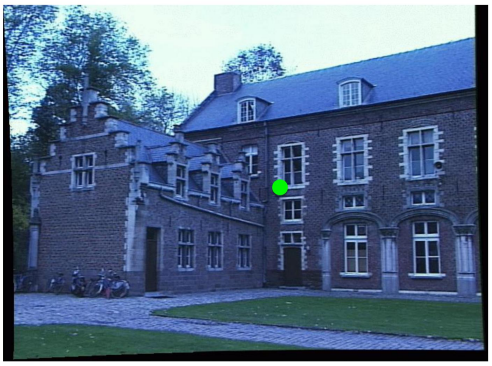
$$p \times MP = 0 \rightarrow \begin{bmatrix} ym_3^T P - m_2^T P \\ m_1^T P - xm_3^T P \end{bmatrix} = 0$$

- Using the matched point in image 2 as well $\rightarrow$ Solution by SVD:

$$\begin{bmatrix} ym_3^T - m_2^T \\ m_1^T - xm_3^T \\ \tilde{y}\tilde{m}_3^T - \tilde{m}_2^T \\ \tilde{m}_1^T - \tilde{x}\tilde{m}_3^T \end{bmatrix} P = 0 \leftrightarrow AP = 0$$

How to Find Matching Points?

How would you reconstruct 3D points?



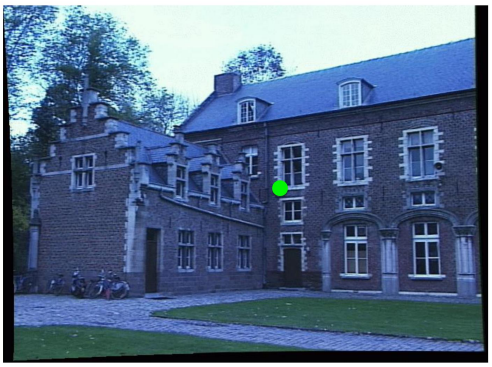
Left image

Right image

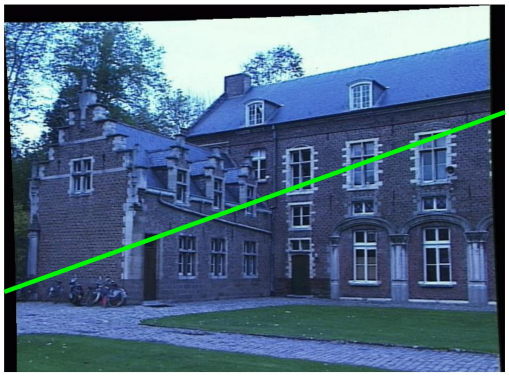
1. Select point in one image (how?)

How to Find Matching Points?

How would you reconstruct 3D points?



Left image

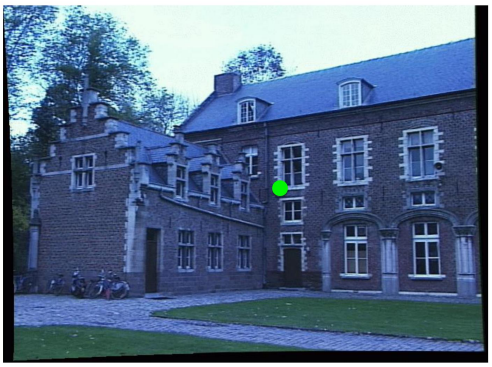


Right image

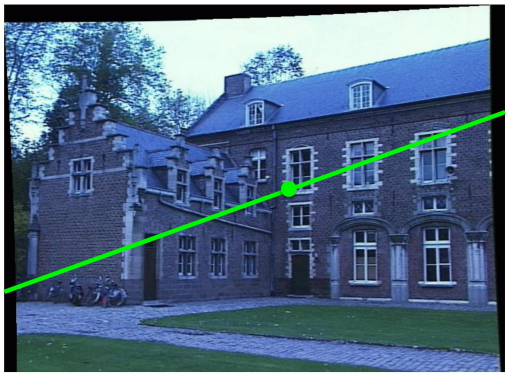
1. Select point in one image (how?)
2. Form epipolar line for that point in second image (how?)

How to Find Matching Points?

How would you reconstruct 3D points?



Left image



Right image

1. Select point in one image (how?)
2. Form epipolar line for that point in second image (how?)
3. Find matching point along line (how?)

Reconstruction

- In Reconstruction, or **Structure from Motion**, our goal is to estimate both the structure of the scene, and the pose (motion) of the camera.
  - We assume we only have a set of matched points $\{p_i, p_i'\}$ - that are matched homogeneous points in 2 different images.
  - We want to estimate the geometry, by estimating both M and M', and we also want to estimate the 3D points P_i, the give us the structure of the scene.

Reconstruction

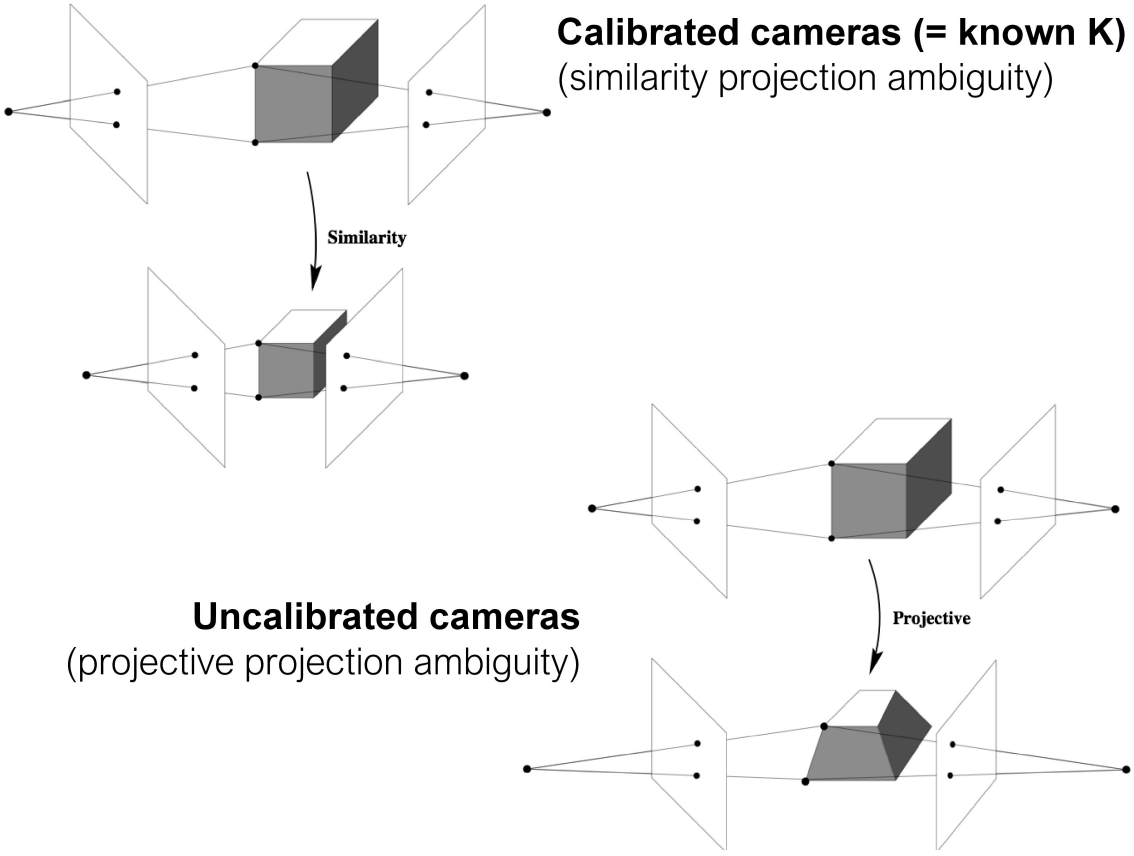
The SfM pipeline is thus:

1. Get $\{p_i, p_i'\}$: Feature detection, feature description, and matching.
2. Estimate F with the 8-point algorithm.
3. Estimate M and M':
   - If we know K, K' we can estimate E and from it get M.
   - Get M, M' from F.
4. Once we have $\{p_i, p_i'\}, M, M'$ we can use triangulation to get P_i.
   - This SVD is usually called The Direct Linear Transformation (DLT).

Reconstruction - Pose ambiguity

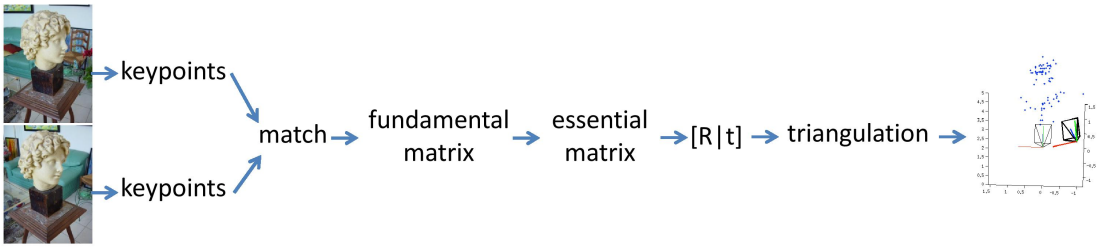
- Step 3 is ambiguous!
- Assuming calibrated camera intrinsics (K,K' known) we have 4 options for M,M'.
  - We supposedly can get from E both R and t (Using $E = [t_\times]R$ and $E = R[dC_\times]$), but since our p, p' are defined up to a scale, we get 2 possible R matrices, and 2 possible t vectors:
- This happens because a projection matrix models also $z < 0$, although not physically feasible.
- To decide between the different options we choose the one with 3D pts in front of the camera.

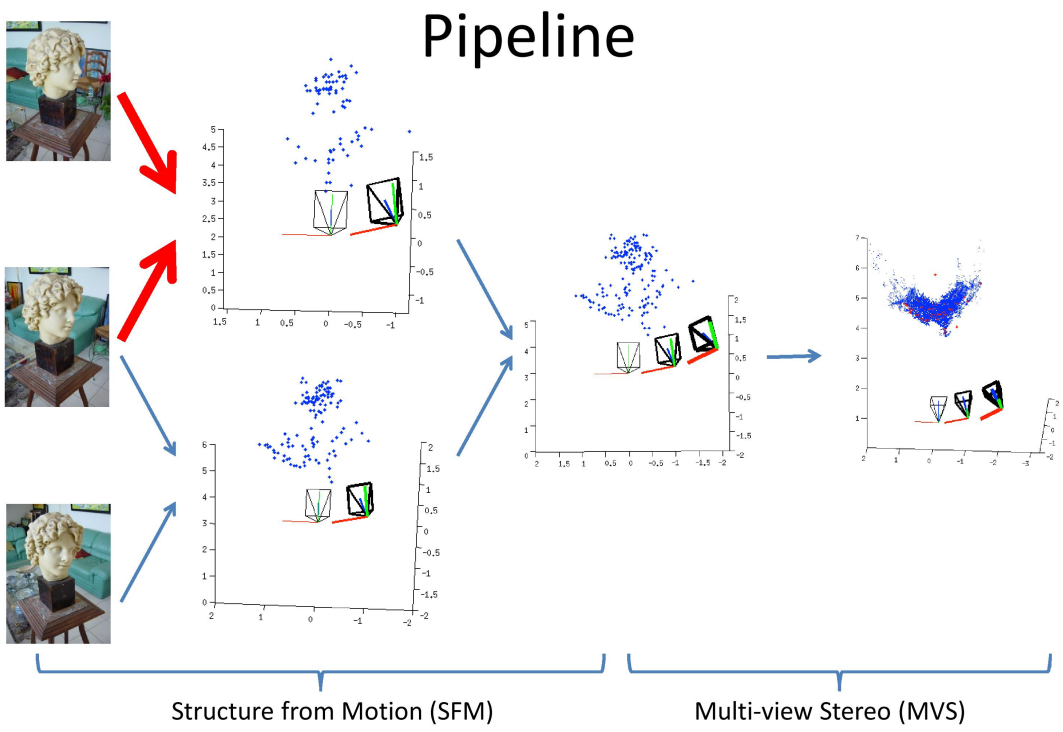
Reconstruction - Scene ambiguity

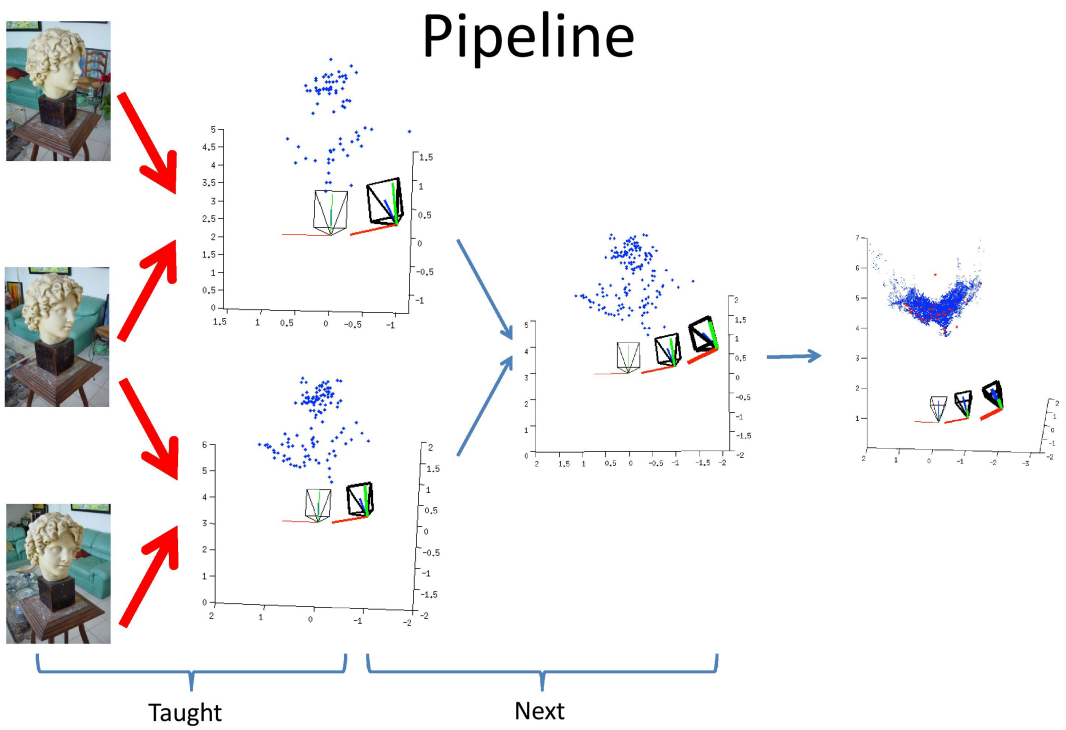


SfM Pipeline Summary

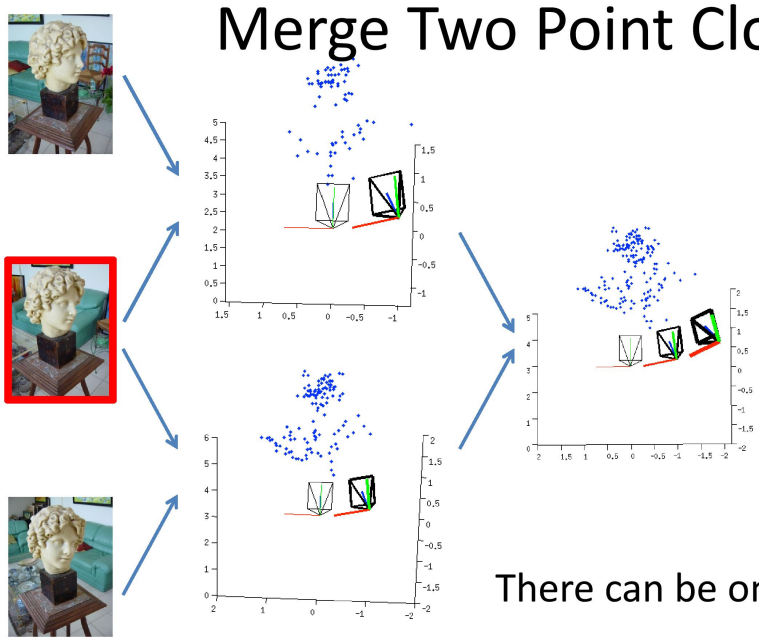
Two-view Reconstruction




Pipeline
Structure from Motion (SFM)
Multi-view Stereo (MVS)


Pipeline
Taught
Next

Merge Two Point Cloud



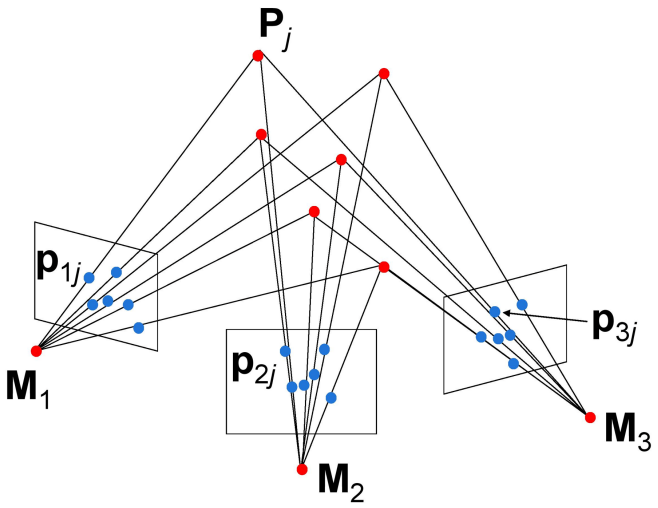
There can be only one $\left[\mathbf{R}_2|\mathbf{t}_2\right]$

Projective structure from motion

- Given: *m* images of *n* fixed 3D points

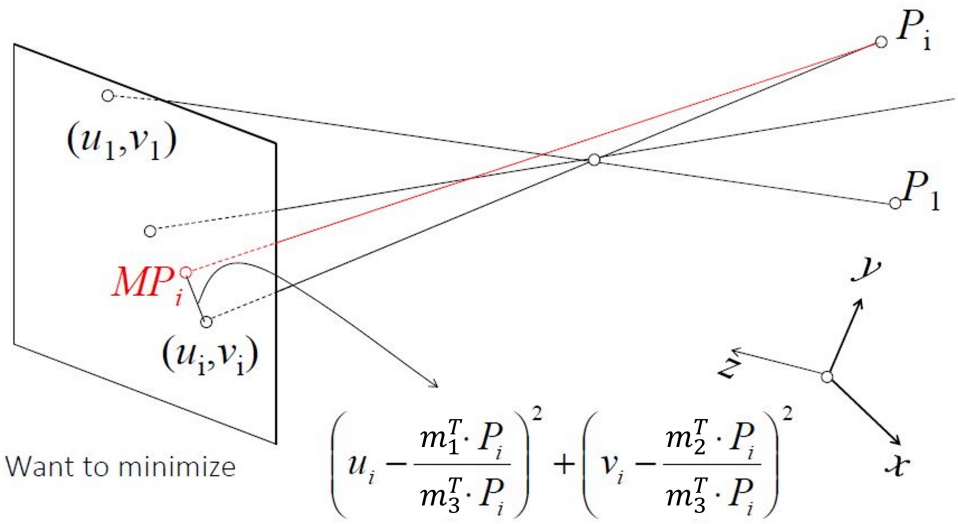
$$z_{ij}\,\mathbf{p}_{ij} = \boldsymbol{M}_i\,\mathbf{P}_j, \quad i = 1, \ldots, m, \quad j = 1, \ldots, n$$

- Problem: estimate *m* projection matrices $\mathbf{P}_i$ and *n* 3D points $\mathbf{P}_j$ from the *mn* correspondences $\mathbf{p}_{ij}$



Bundle Adjustment

- We assume we have m cameras observing a **stationary** 3D scene of n points.
- Not all points (landmarks) are observed in each image. Can happen due to occlusions or noise (e.g. missed detections).
- The objective function is given by minimizing the re-projection error for all observed projections of the n pts.



Bundle Adjustment

- Minimization problem, assuming a calibrated moving camera with intrinsics K:

$$\hat{R}_i, \hat{t}_i, \hat{P}_j = \arg\min_{R_i, t_i, P_j} \sum_{i=1}^{m} \sum_{j=1}^{n} \alpha_{ij} \| p_{ij} - K\,[R_i | t_i]\,P_j \|_2^2$$

where $\alpha_{ij} = 1$ if point j is present in camera i and zero otherwise.

Bundle Adjustment

- The resulting problem is non-linear in $M_1, \ldots, M_m$ and $P_1, \ldots, P_n$.
- Solution obtained through some greedy optimization algorithm (e.g. Levenberg-Marquardt).
- Initialization matters in this case!
  - Good initialization = Good local minima.

Bundle Adjustment

- The amount of unknowns is 11 per camera matrix M_i, and 3 per 3D point P_j.
- This is all true up to a projective transform Q with 15 DoF.
- Since every 2D-2D correpondence gives 2 constraints, we end up with the condition:

$$2mn \geq 11m + 3n - 15$$

Exercise - (Winter 2022 B, Q1)

1. A camera is rotating around its axis in a fixed angular velocity, and taking photos of a palace from within. Is it possible to reconstruct the depth of the objects in the scene based on the taken pictures? Write down the mathematical relationship between the points in the image taken at time t and the points in the image taken at $t + \Delta t$, when the camera has moved by an angle of θ.
2. Now the camera is mounted on a drone and the drone is moving around the palace in a circular orbit (the drone's position is unknown). Is it possible to reconstruct the 3D shape of the palace?
3. N camera matrices $M_1, \ldots, M_N \in \mathbb{R}^{3\times 4}$ are given. Write down the equations system for triangulating a 3D point P by using 2D point-correspondences $(p_1, p_2, \ldots, p_N)$ from N different images, and describe how you can solve it.
4. Can you use the drone-mounted camera for reconstructing the 3D shape of a bird flapping her wings, at each point of time while it's flying?

Solution

1. We can't recover the depth of the objects. When the two views are separated by a pure rotation, the mapping of the points in the images are given by a homography.

$$H = KR_\theta^{-1}K^{-1}$$

1. Now the camera is mounted on a drone and the drone is moving around the palace in a circular orbit (the drone's position is unknown). Is it possible to reconstruct the 3D shape of the palace?

Solution

1. Now we *can* recover the depth objects in the scene (specifically - the shape of the palace). This is the classic SfM setup we've just learned, where we can infer both the camera matrices and the 3D shape of the objects based on keypoint matching.

1. N camera matrices $M_1, \ldots, M_N \in \mathbb{R}^{3\times 4}$ are given. Write down the equations system for triangulating a 3D point P by using N 2D point-correspondences ($p_1, p_2, \ldots, p_N$) from N different images, and describe how you can solve it.

Solution

1. As denoted before, $p_i = (x_i, y_i)^T$, and $M_i = \begin{bmatrix} - & m_{i,1}^T & - \\ - & m_{i,2}^T & - \\ - & m_{i,3}^T & - \end{bmatrix}$.
   We can now write the triangulation equation with N cameras for a single point P by extending the equation from the lecture and this tutorial for more matches:
   $$\begin{bmatrix} y_1 m_{1,3}^T - m_{1,2}^T \\ m_{1,1}^T - x_1 m_{1,3}^T \\ \vdots \\ y_N m_{N,3}^T - m_{N,2}^T \\ m_{N,1}^T - x_N m_{N,3}^T \end{bmatrix} P = AP = 0$$
   The solution is given by SVD (homogeneous linear system).

1. Can you use the drone-mounted camera for reconstructing the 3D shape of a bird flapping her wings, at each point of time while it's flying?

Solution

1. Since we don't have a rigid body assumption on the bird's motion, we will not be able to reconstruct the shape of the bird using the proposed method. Our basic assumption in SfM is that the scene is static, and when the bird is flapping her wings the motion of the points will not be described by a simple matrix.

Recommended Videos

Warning!

- These videos do not replace the lectures and tutorials.
- Please use these to get a better understanding of the material, and not as an alternative to the written material.

Video By Subject

- Computer Vision First Principles - Shree Nayar (https://www.youtube.com/watch?v=JlOzyyhk1v0&ab_channel=FirstPrinciplesofComputerVision)
- Structure from Motion Pipeline - Calle Olsson (https://www.youtube.com/watch?v=i7ierVkXYa8&ab_channel=DrCalleOlsson)

Credits

- EE 046746 Spring 2022 - Hila Manor (https://github.com/HilaManor)
- EE 046746 Winter 2021 - Elias Nehme (https://eliasnehme.github.io/)
- GTSAM blog in Computer Vision Class (U. Maryland) (https://cmsc426.github.io/gtsam/) - Nitin J. Sanket (https://nitinjsanket.github.io/)
- Slides (Princeton) - Jianxiong Xiao (http://3dvision.princeton.edu/courses/SFMedu/slides.pdf)
- Slides (CS Technion) - Elad Osherov (https://scholar.google.com/citations?user=_STeqhAAAAAJ&hl=en)
- Slides (CMU) - Ioannis Gkioulekas (https://www.cs.cmu.edu/~igkioule/), Kris Kitani (http://www.cs.cmu.edu/~kkitani/), Srinivasa Narasimhan (http://www.cs.cmu.edu/~srinivas/)
- Multiple View Geometry in Computer Vision - Hartley and Zisserman - Chapter 9, 18
- Computer Vision: Algorithms and Applications (https://www.springer.com/gp/book/9781848829343) - Richard Szeliski - Chapter 7
- Icons from Icon8.com (https://icons8.com/) - https://icons8.com (https://icons8.com)