HW3
Camera Relocalization & Visual Odometry

Due: 2024/12/2 (一) 11:59 AM

3DCV 2023

Email: 3dcv@csie.ntu.edu.tw

Q1: Camera Relocalization

3DCV 2023

Email: 3dcv@csie.ntu.edu.tw

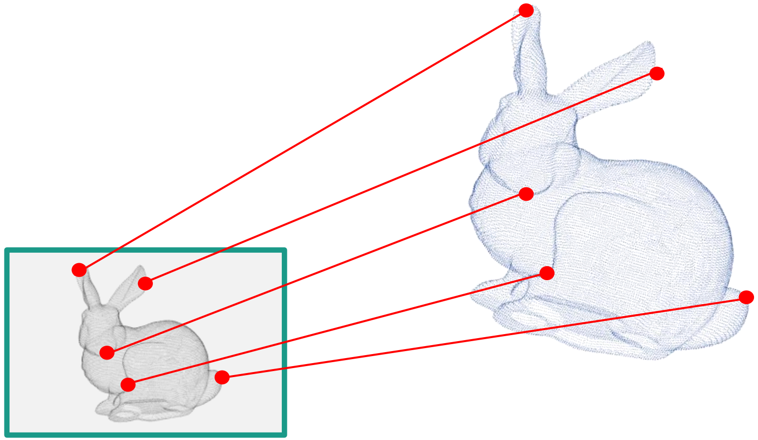
GitHub Classroom: https://classroom.github.com/a/XLxYYHh0

GitHub Registration: https://forms.gle/ucH5A2fsANX9MPzS7

Outline

The goal of this homework is to realize how a camera re-localization system works.

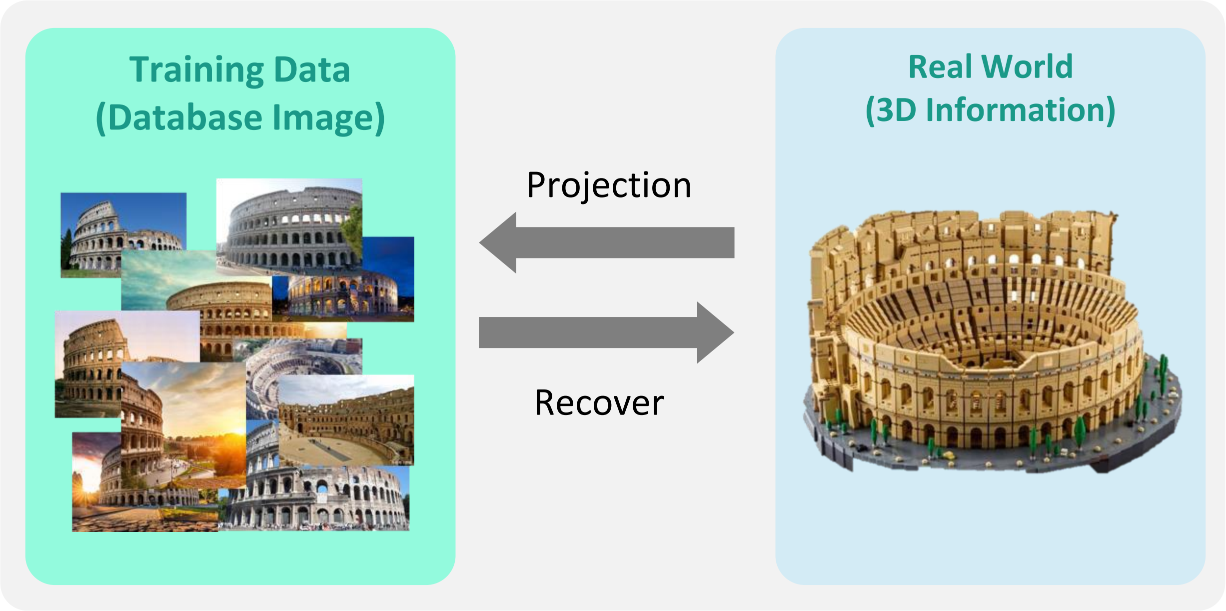
- Introduction
- Dataset
- Step 1: Camera Relocalization
- Step 2: Calculate Error
- Step 3: Results Visualization
- Bonus List
- Grading Policy



Introduction

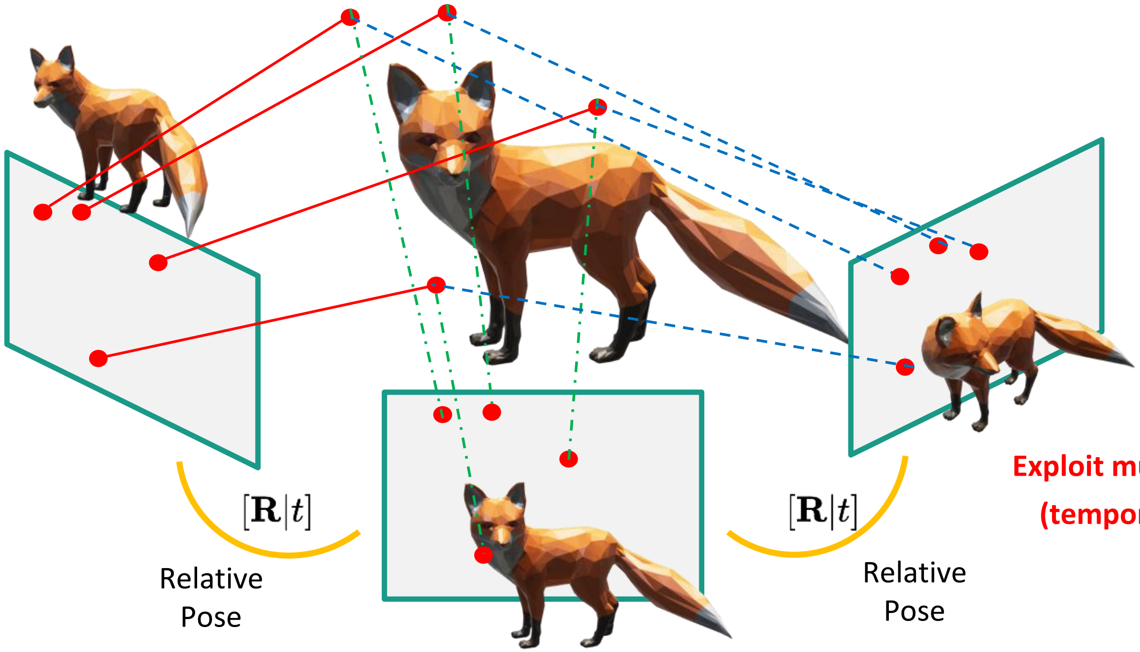
- **Camera Relocalization:** Determine the camera pose from the visual scene representation. In other words, the scene is **seen** (and modeled) **beforehand**. Now, given a query image that is taken is this environment, we are able to find out where this image is taken.

Query Image



Introduction

- **One-shot relocalization**: focus on a finding the pose of still image.
- **Temporal camera relocalization**: estimates the poses of every frame in the video sequence



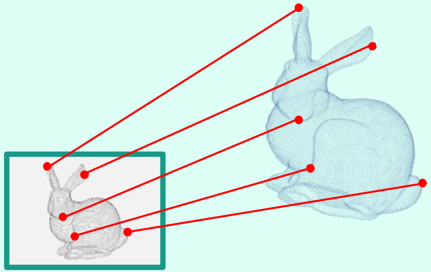
Exploit multi-view constraint (temporal information) !!

Methodology

- Common strategies for camera relocalization. Note that there are some approaches utilize hybrid models to increase the efficiency and robustness.
- Metric localization can only be achieved by machine (or deep) learning models.

2D-3D Matching Localization

3D scene models with local feature descriptors



Metric Localization (Absolute Pose Regression)

regresses the position and orientation of the camera

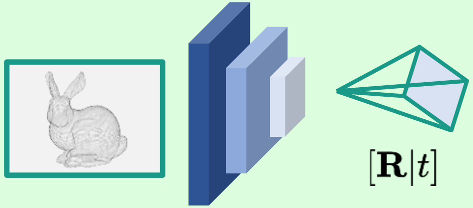
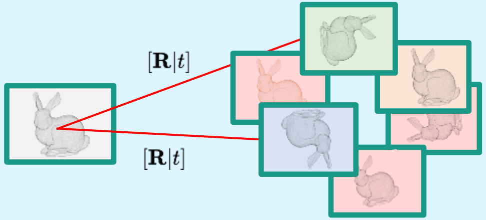


Image Retrieval Localization

pose approximation from top-k similar images



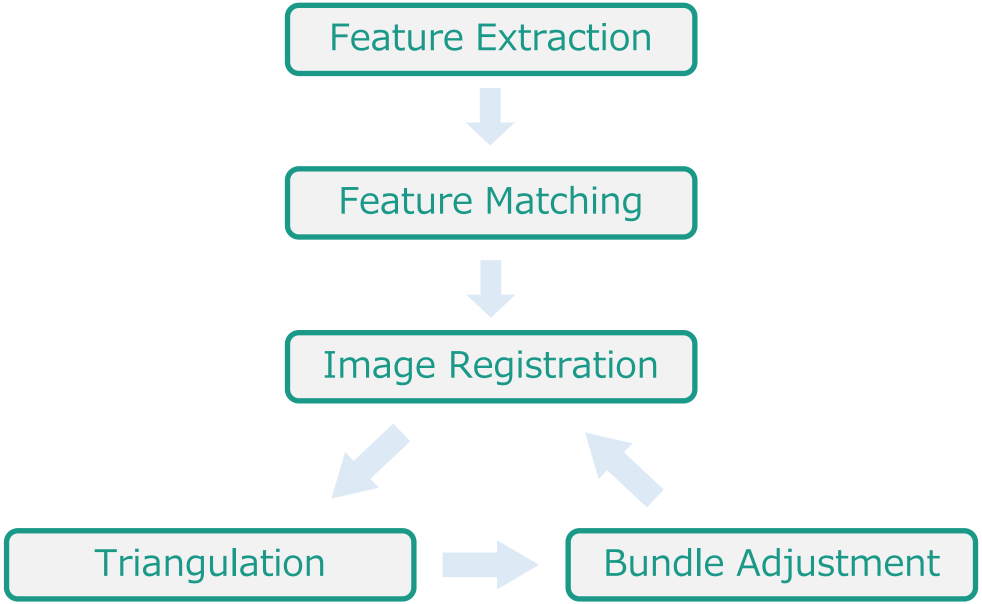
Welcome to the NTU Front Gate

- We collect multiple images of the NTU front gate, and reconstruct its 3D point cloud model via structure from motion.

About Dataset

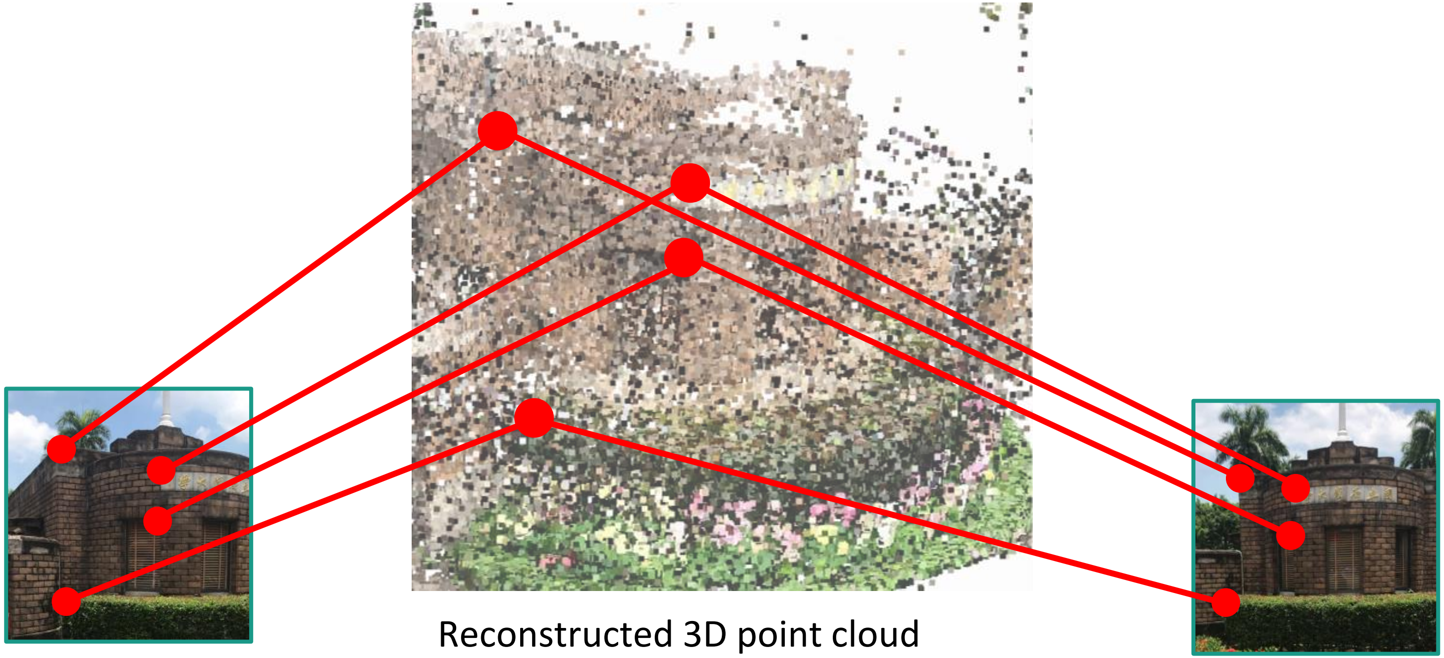
- 293 color images (1920x1080x3): 163 images for training, 130 images for testing
- 111,518 points (in world coordinate) with 682,467 local image descriptors

Dataset images



About Dataset

- 293 color images (1920x1080x3): 163 images for training, 130 images for testing
- 111,518 points (in world coordinate) with 682,467 local image descriptors



Reconstructed 3D point cloud

Data/image.pkl

⚠ The pose of an image is represented as the projection **from world to the camera** coordinate system. That is, p=K[R|T]X.

Camera Position(x,y,z) — TX, TY, TZ

Rotation (in quaternion) — QW, QX, QY, QZ

	IMAGE_ID	NAME	TX	TY	TZ	QW	QX	QY	QZ
0	1	train_img100.jpg	-3.12923	-0.273371	3.17218	0.969363	-0.003488	0.244797	0.019927
1	2	train_img104.jpg	-3.10598	-0.264036	3.12049	0.972423	-0.005048	0.232322	0.019880
2	3	train_img108.jpg	-3.06986	-0.270274	3.08285	0.975032	-0.004203	0.221007	0.021220
3	4	train_img112.jpg	-3.02027	-0.290710	3.07195	0.976940	-0.003627	0.212336	0.022091
4	5	train_img116.jpg	-2.98028	-0.307973	3.05439	0.979017	-0.002989	0.202524	0.022389
...	...	...	...	...	...	...	...	...	...
288	289	valid_img75.jpg	-2.86676	-0.366566	3.79563	0.931094	0.002295	0.363172	0.034118
289	290	valid_img80.jpg	-2.86618	-0.323873	3.72239	0.937160	-0.001973	0.347476	0.031431
290	291	valid_img85.jpg	-2.91426	-0.300918	3.59808	0.945271	-0.004261	0.325035	0.028210
291	292	valid_img90.jpg	-2.99320	-0.267023	3.46717	0.954254	-0.004443	0.298019	0.023733
292	293	valid_img95.jpg	-3.08001	-0.259334	3.30072	0.962891	-0.003862	0.269045	0.021006

293 rows × 9 columns

⚠ Note that the order is (QW, QX, QY, QZ)

Data/point_desc.pkl

- point_desc.pkl

Source Info

128D Descriptors

	POINT_ID	IMAGE_ID	XY	DESCRIPTORS
0	1	1	[94.94650268554688, 284.02899169921875]	[46, 43, 12, 11, 10, 5, 19, 37, 24, 16, 8, 9, ...
1	1	2	[99.05780029296875, 290.6889953613281]	[39, 42, 34, 14, 15, 12, 13, 31, 29, 11, 8, 7,...
2	1	3	[110.51899719238281, 291.7560119628906]	[47, 57, 39, 12, 12, 11, 9, 20, 43, 26, 13, 7,...
3	1	4	[131.70199584960938, 286.4880065917969]	[38, 58, 39, 12, 11, 11, 13, 16, 35, 20, 12, 8...
4	1	7	[156.52499389648438, 279.2149963378906]	[32, 38, 31, 19, 15, 6, 11, 32, 28, 14, 6, 10,...
...	...	...	...	...
1234453	129081	276	[816.5590209960938, 353.6910095214844]	[28, 20, 11, 16, 23, 18, 22, 25, 42, 11, 8, 24...
1234454	129081	278	[892.0490112304688, 384.6050109863281]	[30, 30, 15, 22, 28, 14, 15, 23, 47, 13, 10, 2...
1234455	129081	279	[965.5770263671875, 397.2950134277344]	[29, 22, 12, 18, 28, 16, 20, 30, 40, 12, 9, 27...
1234456	129081	280	[1039.56005859375, 405.864990234375]	[27, 24, 14, 15, 26, 16, 25, 33, 45, 12, 10, 2...
1234457	129081	280	[1045.989990234375, 404.6090087890625]	[23, 38, 24, 33, 28, 7, 3, 7, 52, 12, 12, 26, ...

⚠ If Point_ID is -1, then its 3D position is not available.

Data/train.pkl

- train.pkl

3D Point Position(x,y,z) | Source Info | 128D Descriptors

	POINT_ID	XYZ	RGB	IMAGE_ID	XY	DESCRIPTORS
0	1	[1.6093346, -1.1848674, 1.610395]	[87, 87, 77]	1	[94.94650268554688, 284.02899169921875]	[46, 43, 12, 11, 10, 5, 19, 37, 24, 16, 8, 9, ...
1	1	[1.6093346, -1.1848674, 1.610395]	[87, 87, 77]	2	[99.05780029296875, 290.6889953613281]	[39, 42, 34, 14, 15, 12, 13, 31, 29, 11, 8, 7,...
2	1	[1.6093346, -1.1848674, 1.610395]	[87, 87, 77]	3	[110.51899719238281, 291.7560119628906]	[47, 57, 39, 12, 12, 11, 9, 20, 43, 26, 13, 7,...
3	1	[1.6093346, -1.1848674, 1.610395]	[87, 87, 77]	4	[131.70199584960938, 286.4880065917969]	[38, 58, 39, 12, 11, 11, 13, 16, 35, 20, 12, 8...
4	1	[1.6093346, -1.1848674, 1.610395]	[87, 87, 77]	7	[156.52499389648438, 279.2149963378906]	[32, 38, 31, 19, 15, 6, 11, 32, 28, 14, 6, 10,...
...	...	...	...	...	...	...
682463	129081	[0.66382873, -1.3121917, 5.433149]	[33, 30, 28]	141	[834.9459838867188, 363.7510070800781]	[32, 26, 15, 19, 28, 14, 18, 30, 37, 12, 11, 2...
682464	129081	[0.66382873, -1.3121917, 5.433149]	[33, 30, 28]	142	[867.6019897460938, 366.8039855957031]	[33, 16, 6, 11, 25, 16, 18, 36, 41, 10, 7, 23,...
682465	129081	[0.66382873, -1.3121917, 5.433149]	[33, 30, 28]	144	[981.5599975585938, 398.8039855957031]	[25, 14, 7, 12, 27, 21, 24, 28, 50, 13, 8, 24,...
682466	129081	[0.66382873, -1.3121917, 5.433149]	[33, 30, 28]	145	[1039.56005859375, 405.864990234375]	[27, 24, 14, 15, 26, 16, 25, 33, 45, 12, 10, 2...
682467	129081	[0.66382873, -1.3121917, 5.433149]	[33, 30, 28]	145	[1045.989990234375, 404.6090087890625]	[23, 38, 24, 33, 28, 7, 3, 7, 52, 12, 12, 26, ...

682468 rows × 6 columns

About Dataset: Camera Parameters

- Review the Pinhole camera model:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \approx \begin{bmatrix} f_x & s & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- Intrinsic Parameters:

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1868.27 & 0 & 540 \\ 0 & 1869.18 & 960 \\ 0 & 0 & 1 \end{bmatrix}$$

- Distortion Parameters (Brown-Conrady Model):

$$D = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 \end{bmatrix} = [0.0847023, -0.192929, -0.000201144, -0.000725352]$$

Step 1: Camera Relocalization

For each validation image, compute its camera pose with respect to world coordinate. Find the 2D-3D correspondence by descriptor matching, and solve the camera pose.

Notes:

- You can choose what method you want on OpenCV website
  https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html
- If the method you choose is not the one we teach, you can briefly introduce the details, which will be the bonus part

Step 2: Calculate Error

For each camera pose you calculated, compute the median pose error (translation, rotation) with respect to ground truth camera pose. Provide some discussion.

Notes:

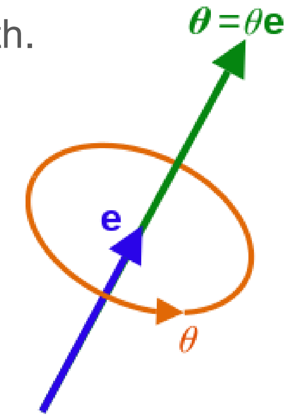
- Translation: median of all absolute pose differences (Euclidean Distance).

$$t_e = \|\mathbf{t} - \hat{\mathbf{t}}\|_2$$

- Rotation: median of relative rotation angle between estimation and ground-truth.

  (1. Find out the relative rotation and represent it as axis angle representation.

  2. Report the median of angles.)



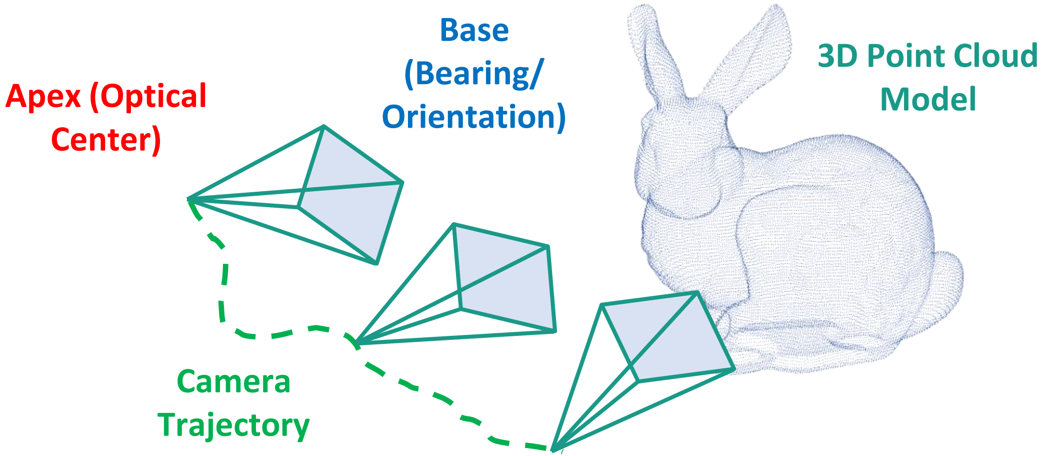
$$\mathcal{R} = R_e \widehat{\mathcal{R}}$$

Step 3: Results Visualization

step-3 For each camera pose you calculated, plot the trajectory and camera poses along with 3d point cloud model using Open3D. Explain how you draw and provide some discussion.

Notes:

- Draw the camera pose as a quadrangular pyramid, where the apex is the position of the optical center, and the normal of base is the bearing (orientation) of the camera.



Sample Code

- You should read the pickle files with **pandas**.

```
>>> import pandas as pd
>>> images_df = pd.read_pickle("dataframes/images.pkl")
```

- You may use **Scipy** to deal with 3D rotation representations.

```
>>> from scipy.spatial.transform import Rotation as R
>>> r = R.from_quat([0, 0, np.sin(np.pi/4), np.cos(np.pi/4)])
>>> r.as_rotvec()
array([0. , 0. , 1.57079633])
```

⚠ Be aware of the order.

Parameters: **quat : *array_like, shape (N, 4) or (4,)***

Each row is a (possibly non-unit norm) quaternion in scalar-last (x, y, z, w) format. Each quaternion will be normalized to unit norm.

Returns: **rotation : *`Rotation` instance***

Object containing the rotations represented by input quaternions.

Introduction to Open3D

- Install open3D

```
pip install open3d
```

- Basic manipulation in open3D (Example Drawing):

```
points = [[0, 0, 0], [1, 0, 0], [0, 1, 0], [1, 1, 0],
          [0, 0, 1], [1, 0, 1], [0, 1, 1], [1, 1, 1]]
lines  = [[0, 1], [0, 2], [1, 3], [2, 3], [4, 5], [4, 6],
          [5, 7], [6, 7], [0, 4], [1, 5], [2, 6], [3, 7]]

import open3d as o3d
line_set = o3d.geometry.LineSet()
line_set.points = o3d.utility.Vector3dVector(points)
line_set.lines = o3d.utility.Vector2iVector(lines)

vis = o3d.visualization.Visualizer()
vis.create_window()
vis.add_geometry(line_set) o3d.visualization.ViewControl.set_zoom(vis.get_view_control(), 0.8)
vis.run()
```

⚠ Please refer to the document to find the property you need.

http://www.open3d.org/docs/latest/tutorial/visualization/visualization.html

Bonus List

To get extra credits, you can try the following things:(including, but not limited to)

- **Introduce Others Method:** Briefly introduce the details of the method we did not teach.
- **Local Features:** Try different kinds of local features (including deep features)
- **Make it faster:** Come up with faster matching or image registration strategy. (prioritized matching, approximate nearest neighbor, coarse-to-fine strategy, image retrieval, …)
- **Make it more accurate**: Make the pose estimation more accurate. (Different PnP solving methods, outlier rejection strategies, …)
- **Absolute Pose Regression:** Train a deep neural network to regress the absolute camera pose. (For example, PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization, ICCV 2015)
- **Gaussian splatting based method**: Train a GS and Using GS for Relocalization. (Grounding Keypoint Descriptors into 3D Gaussian Splatting for Improved Visual Localization, MonoGS (CVPR2024))

Report

1. **The method you choose and explain how to use the function**
2. **Error**
3. **Result Visualization**
4. **Others detail or bonus**

YouTube link :

You should record your demonstration, including the start time and the GitHub clone action

- Example : https://youtu.be/-VnjVda7c8o?si=77nV7V1ngjZqoY5G

●Please tell us how to execute your codes, including the package used and the environment.

Grading (50%)

- We will evaluate both **the functionality of the code** and **the quality of the report**.
- **Functionality**: Can it run? How's the performance?
- **Quality**: theoretical/experimental analysis, observation, discussion, …
- Note that it **might be curved** based on overall performance of students.
- Grade
  - Meet the basic requirement (programming & report) → A
  - Basic requirement + advanced studies (programming & report) → A+

Q2: Visual Odometry

3DCV 2023

Email: 3dcv@csie.ntu.edu.tw

GitHub Classroom: https://classroom.github.com/a/LFInxJci

GitHub Registration: https://forms.gle/ucH5A2fsANX9MPzS7

Goal: Visual Odometry

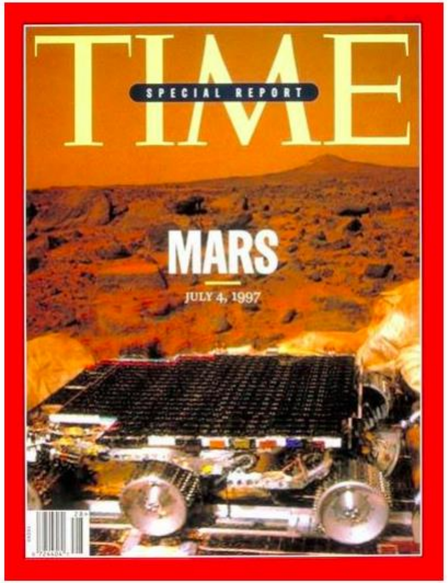
- Odometry

  Estimating change in position overtime

- Visual Odometry

  Estimating the motion of a camera in real time using sequential images (i.e., ego-motion)

- Difference from SLAM
  - VO mainly focuses on local consistency and aims to incrementally estimate the path of the camera pose after pose
  - SLAM aims to obtain a globally consistent estimate of the camera/robot trajectory and map



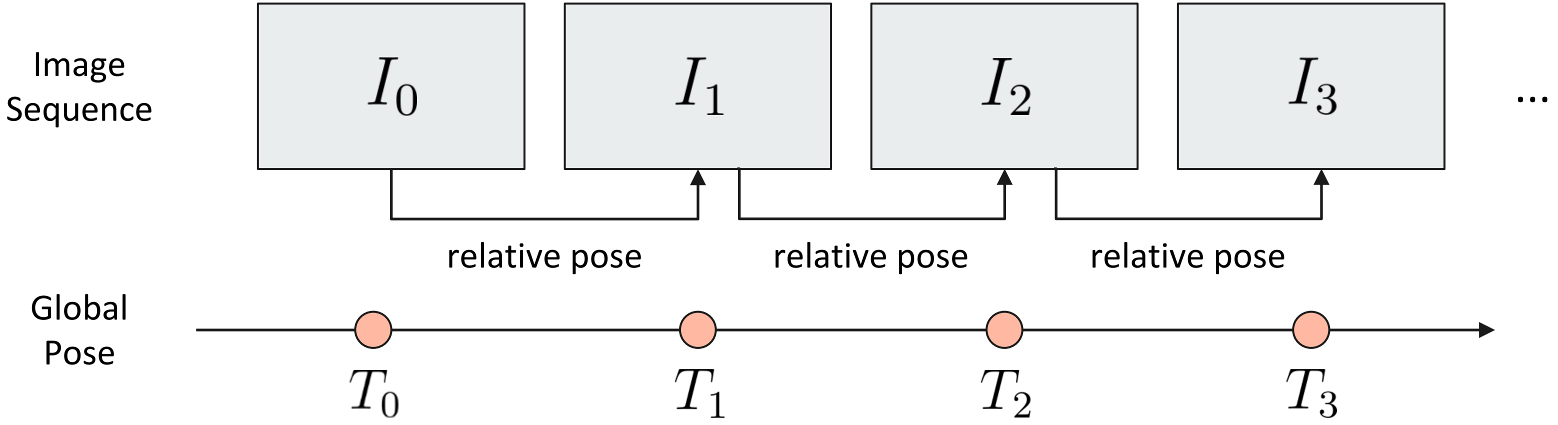
Pathfinder landing, 1997

ref: http://www.cs.toronto.edu/~urtasun/courses/CSC2541/03_odometry.pdf

Goal: Visual Odometry

Implement a VO based on two-view epipolar geometry

- Input: a provided image sequence and the camera intrinsic
- Output: a sequential global camera pose (w.r.t. the coordinate system of the 1st frame)
- You are **allowed to use any OpenCV API**



Step 1: Camera calibration

We have introduced camera calibration (slide)

Just calibrate the camera with the provided program to obtain camera intrinsic matrix and distortion coefficients

[How to use]

- $ python3 camera_calibration.py [CALIBRATE_VIDEO]

  Use "python3 camera_calibration.py --help" to check more argument information

  Enter SPACE key to add new frame to calibrate
- The program will save "camera_parameters.npy" by default

  Checkout "vo.py" template to know how to read the npy file

https://youtu.be/H1OzJPKYOFQ

Step 2: Feature Matching

We recommend to use ORB [Rublee 2011] as feature extractor

- faster than SIFT over 10x
- binary descriptor
- orientation and scale invariance
- Compute **Hamming distance** for binary feature matching
- Sample code: https://docs.opencv.org/4.5.1/dc/dc3/tutorial_py_matcher.html

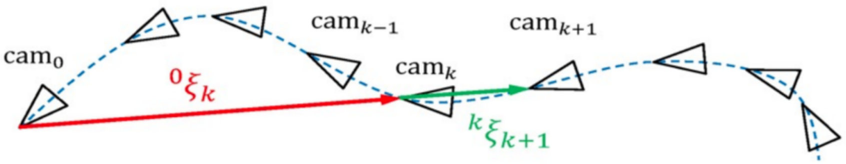
```
# create BFMatcher object
bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)

# Match descriptors.
matches = bf.match(des1,des2)
```

Rublee, et al. ORB: An efficient alternative to SIFT or SURF. ICCV, 2011.

Step 3: Pose from Epipolar Geometry

- Recap: page 13 in [Slide 21]



Visual odometry from 2D-correspondences

1. Capture new frame img_{k+1}
2. Extract and match features between img_{k+1} and img_k
3. Estimate the essential matrix $E_{k,k+1}$
4. Decompose the $E_{k,k+1}$ into ${}^kR_{k+1}$ and ${}^k\boldsymbol{t}_{k+1}$ to get the relative pose
$${}^k\xi_{k+1} = [{}^kR_{k+1} \quad {}^k\boldsymbol{t}_{k+1}]$$
5. Calculate the pose of camera $k+1$ relative to the first camera
$${}^0\xi_{k+1} = {}^0\xi_k \, {}^k\xi_{k+1}$$

Step 3: cv2.findEssentialMat

Step 4: cv2.recoverPose

Step 3: Pose from Epipolar Geometry

- Recap: Scale consistency in page 17 in [Slide 20]

  By default, the translation t from cv2.recoverPose is normalized to unit norm

  You have to rescale t according to previous triangulated points

A better visual odometry algorithm can look like this

- How to compute $\|{}^{k+1}\boldsymbol{t}_k\|$ from $\|{}^{k}\boldsymbol{t}_{k-1}\|$?
- Determine two scene points ${}^{k}\boldsymbol{X}_{k-1,k}$ and ${}^{k}\boldsymbol{X}'_{k-1,k}$ by triangulation of two 2D-correspondences ${}^{k-1}\boldsymbol{x} \leftrightarrow {}^{k}\boldsymbol{x}$ and ${}^{k-1}\boldsymbol{x}' \leftrightarrow {}^{k}\boldsymbol{x}'$
- Determine the same two scene points ${}^{k}\boldsymbol{X}_{k,k+1}$ and ${}^{k}\boldsymbol{X}'_{k,k+1}$ by triangulation of two 2D-correspondences ${}^{k}\boldsymbol{x} \leftrightarrow {}^{k+1}\boldsymbol{x}$ and ${}^{k}\boldsymbol{x}' \leftrightarrow {}^{k+1}\boldsymbol{x}'$
- Then

$$\frac{\left\|{}^{k-1}\boldsymbol{t}_k\right\|}{\left\|{}^{k}\boldsymbol{t}_{k+1}\right\|} = \frac{\left\|{}^{k}\boldsymbol{X}_{k-1,k} - {}^{k}\boldsymbol{X}'_{k-1,k}\right\|}{\left\|{}^{k}\boldsymbol{X}_{k,k+1} - {}^{k}\boldsymbol{X}'_{k,k+1}\right\|}$$

You can directly get triangulated points by cv2.recoverPose or further use cv2.triangulatePoints

Step 4: Results Visualization

- We provide template code (vo.py) of jointly showing current image and visualize camera trajectory in Open3D
- Draw the matched (tracked) point on current image
- Update the new camera pose in Open3D window
- Feel free to use any other 3D visualizer (e.g. pangolin) if you implement in C++

Template vo.py

We provide a template vo.py (tested in python3.6, 3.7, 3.8)

dependency: numpy, opencv-python==4.5.1.48, open3d==0.12.0

[How to run] python3 vo.py /path/to/frames/dir

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('input', help='directory of sequential frames')
    parser.add_argument('--camera_parameters', default='camera_parameters.npy', help='npy file of camera parameters')
    args = parser.parse_args()

    vo = SimpleVO(args)
    vo.run()
```

```
class SimpleVO:
    def __init__(self, args):
        camera_params = np.load(args.camera_parameters, allow_pickle=True)[()]
        self.K = camera_params['K']
        self.dist = camera_params['dist']

        self.frame_paths = sorted(list(glob.glob(os.path.join(args.input, '*.png'))))
```

We have already helped you read the camera parameters

Template vo.py

```python
def run(self):
    vis = o3d.visualization.Visualizer()
    vis.create_window()

    queue = mp.Queue()
    p = mp.Process(target=self.process_frames, args=(queue, ))
    p.start()

    keep_running = True
    while keep_running:
        try:
            R, t = queue.get(block=False)
            if R is not None:
                #TODO:
                # insert new camera pose here using vis.add_geometry()
                pass
        except: pass

        keep_running = keep_running and vis.poll_events()
    vis.destroy_window()
    p.join()
```

```python
def process_frames(self, queue):
    R, t = np.eye(3, dtype=np.float64), np.zeros((3, 1), dtype=np.float64)
    for frame_path in self.frame_paths[1:]:
        img = cv.imread(frame_path)
        #TODO: compute camera pose here

        queue.put((R, t))

        cv.imshow('frame', img)
        if cv.waitKey(30) == 27: break
```

Run two window (cv2.imshow and Open3D) in the same time

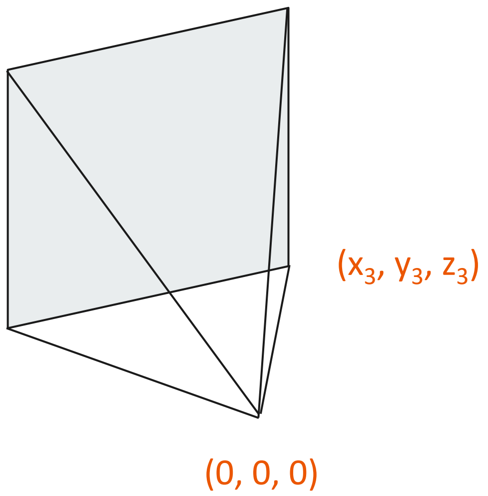
You can press ESC to kill each window

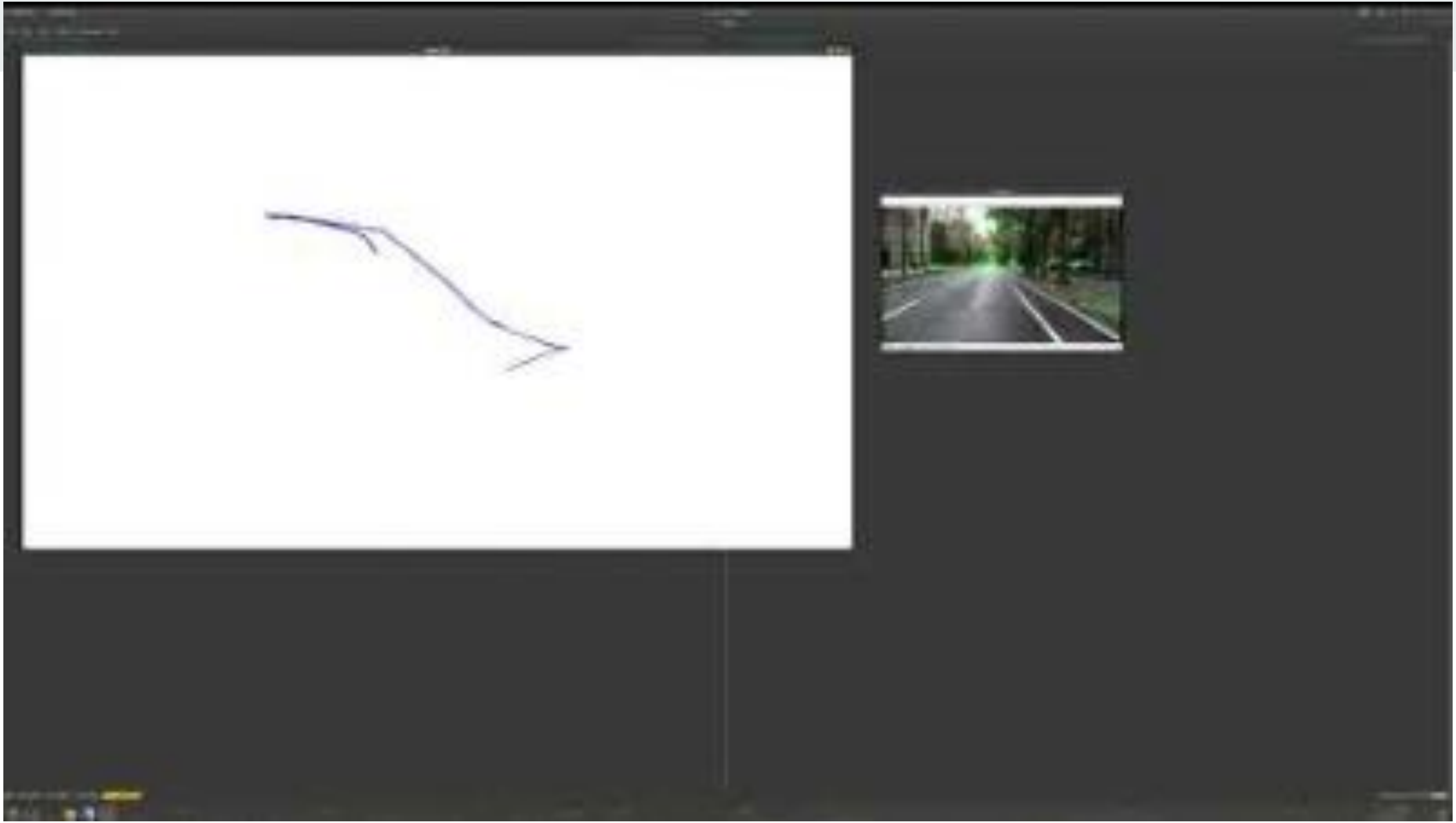
Add LinSet in Open3D

- Open3D LineSet tutorial
- The points of camera frame w.r.t. camera coord. sys., e.g.

$$\begin{bmatrix} x_3 \\ y_3 \\ z_3 \end{bmatrix} \sim K^{-1} \begin{bmatrix} w-1 \\ h-1 \\ 1 \end{bmatrix}$$

  z_3= 1 since the point is on the image plane (z=1)
- Finally, use R, t to transform (x_3, y_3, z_3) into global coord. sys.





https://youtu.be/nL9xXurote0

Report

- Briefly explain your method in each step
  - Camera calibration
  - Feature Matching
  - Pose from Epipolar Geometry (pseudo codes and comments)
  - Results Visualization
- Youtube link

① Your video should be like the video shown in **page 13**

② You should record your demonstration, including the start time and the GitHub clone action

  - Example : https://youtu.be/-VnjVda7c8o?si=77nV7V1ngjZqoY5G

- Please tell us how to execute your codes, including the package used and the environment.

Bonus List

- Loop detection
  - only detect, please describe your loop detection method
  - provide a demo video of loop detection with showing message when loop detected
- Point cloud and bundle adjustment (i.e. SLAM)
  - local bundle adjustment for local map
  - global bundle adjustment (loop closing) after loop detection
- Any performance improvement in the VO
  - runtime speedup (from the original speed to the improved speed on your device)
  - qualitative comparison of the original trajectory and the improved trajectory
- Implement both cv2.findEssentialMat and cv2.recoverPose by your self

Bonus List

Notice

- OpenCV from pip does not contain sfm module. You can still find another way to use it, e.g. implement in C++
- The time performance is important since VO is an online application
- Please also explain how to run your bonus code

Grading(50%)

- We will evaluate both **the functionality of the code** and **the quality of the report**.
- **Functionality**: Can it run? How’s the performance?
- **Quality**: theoretical/experimental analysis, observation, discussion, …
- Note that it **might be curved** based on overall performance of students.
- Grade
  - Meet the basic requirement (programming & report) → A
  - Basic requirement + advanced studies (programming & report) → A+

Grading Policy

- Push your code and report to the **correct** GitHub classroom.
- Programming Languages: Python (Python>=3.8)
- Report Format: PDF or Markdown

  (Warning for Markdown users: Latex equations cannot be rendered properly in GitHub)
- Late Submission: **-10% from your score** / day
- **Plagiarism: You have to write your own codes.**
- Discussion: We encourage you to discuss with your classmates, but remember to **mention their names and contributions in the report**.

Do not plagiarize !

- For Homework3, we are allowing you to submit references or admit to plagiarism from today until next 10 days.
  - **Deadline: 2024/12/2 (一) 11:59 AM**

- Plagiarism refers to **using a previous classmate's code from GitHub** as a template for modification or even submitting it directly.

Thanks

If you have any question, please email 3dcv@csie.ntu.edu.tw