



# HW3 Visual Odometry

Due: 2022/11/22 (二) 11:59 AM

3DCV 2022

Email: [3dcv@csie.ntu.edu.tw](mailto:3dcv@csie.ntu.edu.tw)

GitHub Classroom: <https://classroom.github.com/a/e3Dk6xcT>

GitHub Registration: <https://forms.gle/ikNyfxXJUw7krcG88>

# 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

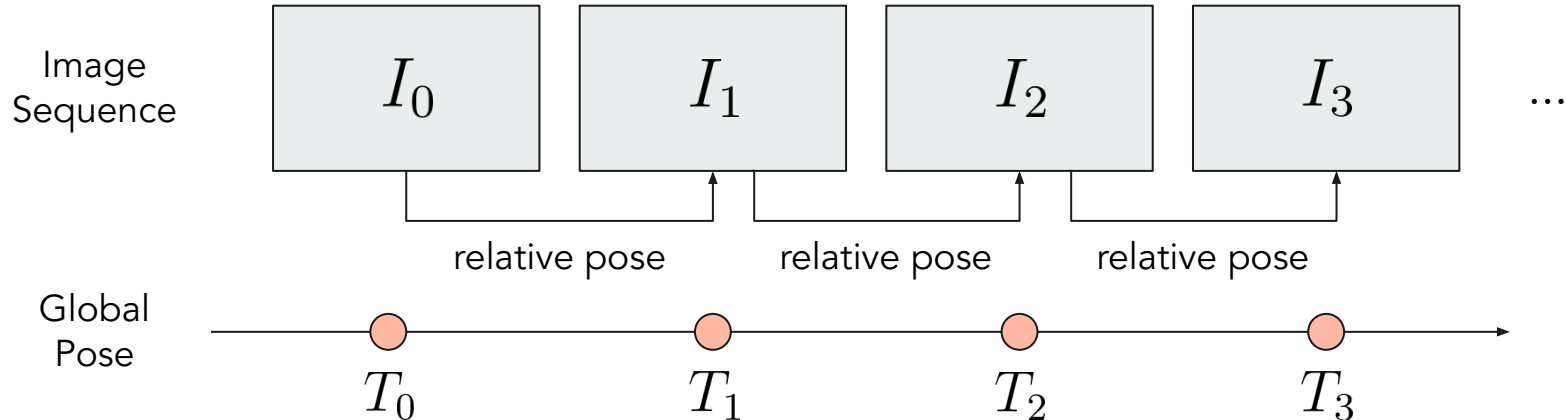


Pathfinder landing, 1997

# 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 1<sup>st</sup> frame)
- You are allowed to use any OpenCV API



# Step 1: Camera calibration

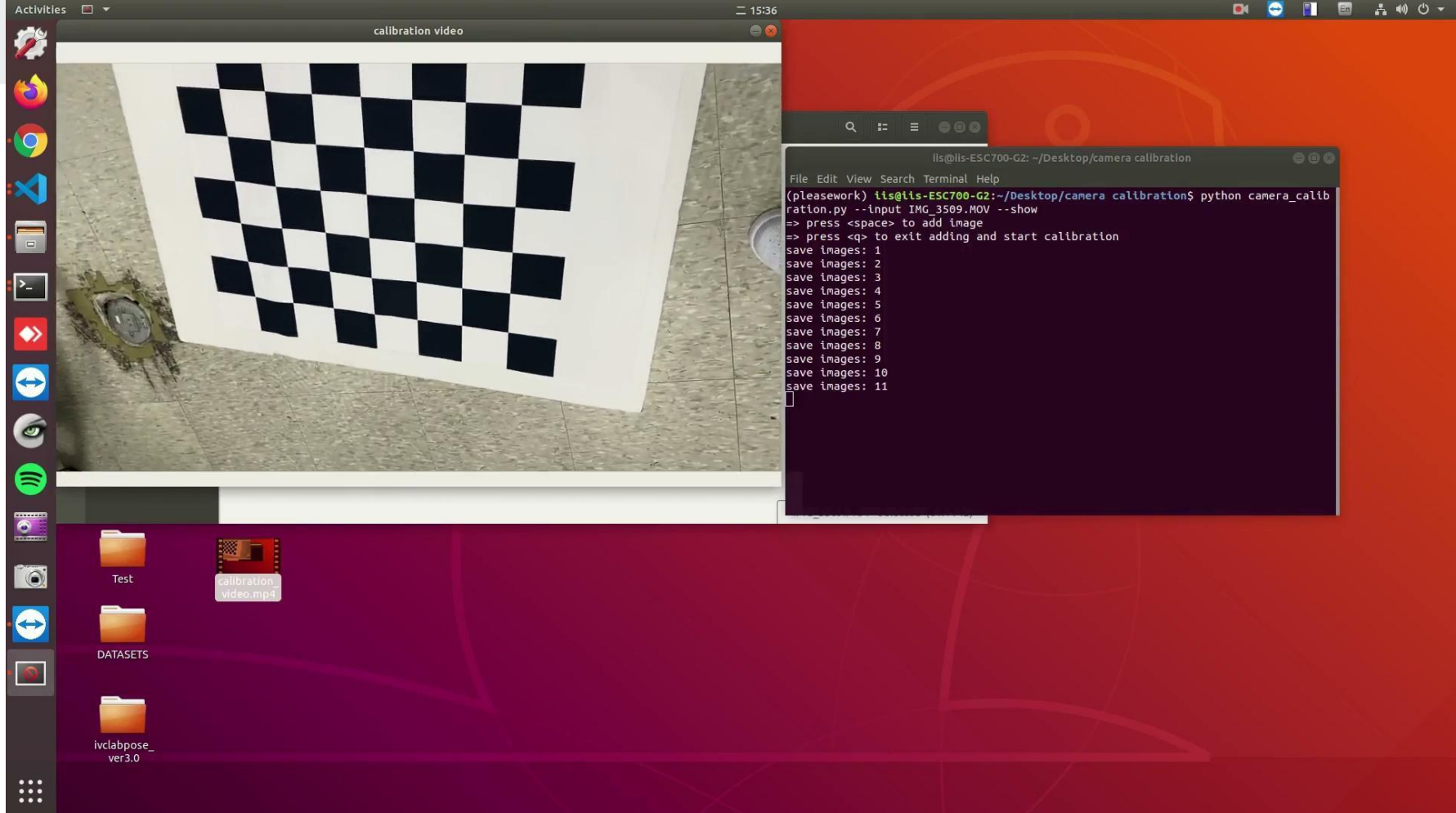


We have introduced camera calibration ([slide](#))

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



[\[Video Link\]](#) If you cannot watch here

## 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:

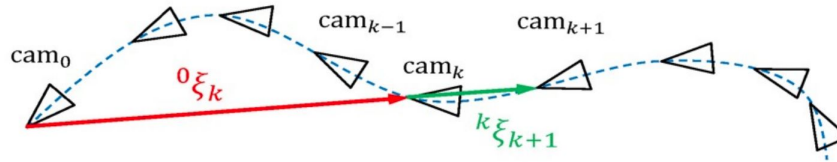
[https://docs.opencv.org/4.5.1/dc/dc3/tutorial\\_py\\_matcher.html](https://docs.opencv.org/4.5.1/dc/dc3/tutorial_py_matcher.html)

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

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

## 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  ${}^kt_{k+1}$  to get the relative pose

$${}^k\xi_{k+1} = [{}^kR_{k+1} \quad {}^kt_{k+1}]$$

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  $\|^{k+1}t_k\|$  from  $\|{}^k t_{k-1}\|$  ?
- Determine two scene points  ${}^k X_{k-1,k}$  and  ${}^k X'_{k-1,k}$  by triangulation of two 2D-correspondences  ${}^{k-1}x \leftrightarrow {}^k x$  and  ${}^{k-1}x' \leftrightarrow {}^k x'$
- Determine the same two scene points  ${}^k X_{k,k+1}$  and  ${}^k X'_{k,k+1}$  by triangulation of two 2D-correspondences  ${}^k x \leftrightarrow {}^{k+1}x$  and  ${}^k x' \leftrightarrow {}^{k+1}x'$
- Then

$$\frac{\|{}^{k-1}t_k\|}{\|{}^k t_{k+1}\|} = \frac{\|{}^k X_{k-1,k} - {}^k X'_{k-1,k}\|}{\|{}^k X_{k,k+1} - {}^k X'_{k,k+1}\|}$$

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

```
48 if __name__ == '__main__':
49     parser = argparse.ArgumentParser()
50     parser.add_argument('input', help='directory of sequential frames')
51     parser.add_argument('--camera_parameters', default='camera_parameters.npy', help='npy file of camera parameters')
52     args = parser.parse_args()
53
54     vo = SimpleVO(args)
55     vo.run()

7 class SimpleVO:
8     def __init__(self, args):
9         camera_params = np.load(args.camera_parameters, allow_pickle=True)[()]
10         self.K = camera_params['K']
11         self.dist = camera_params['dist']
12
13         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

```
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()
```

```
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

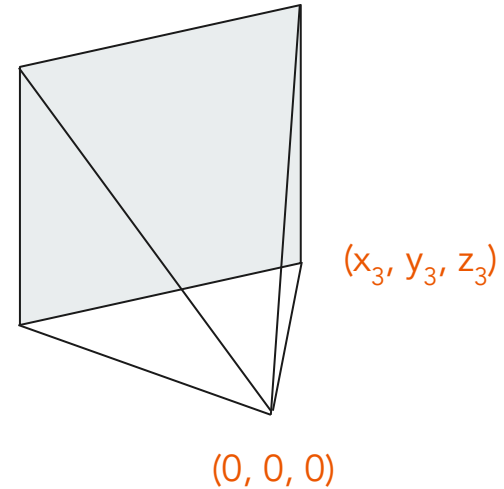
# Add LinSet in Open3D

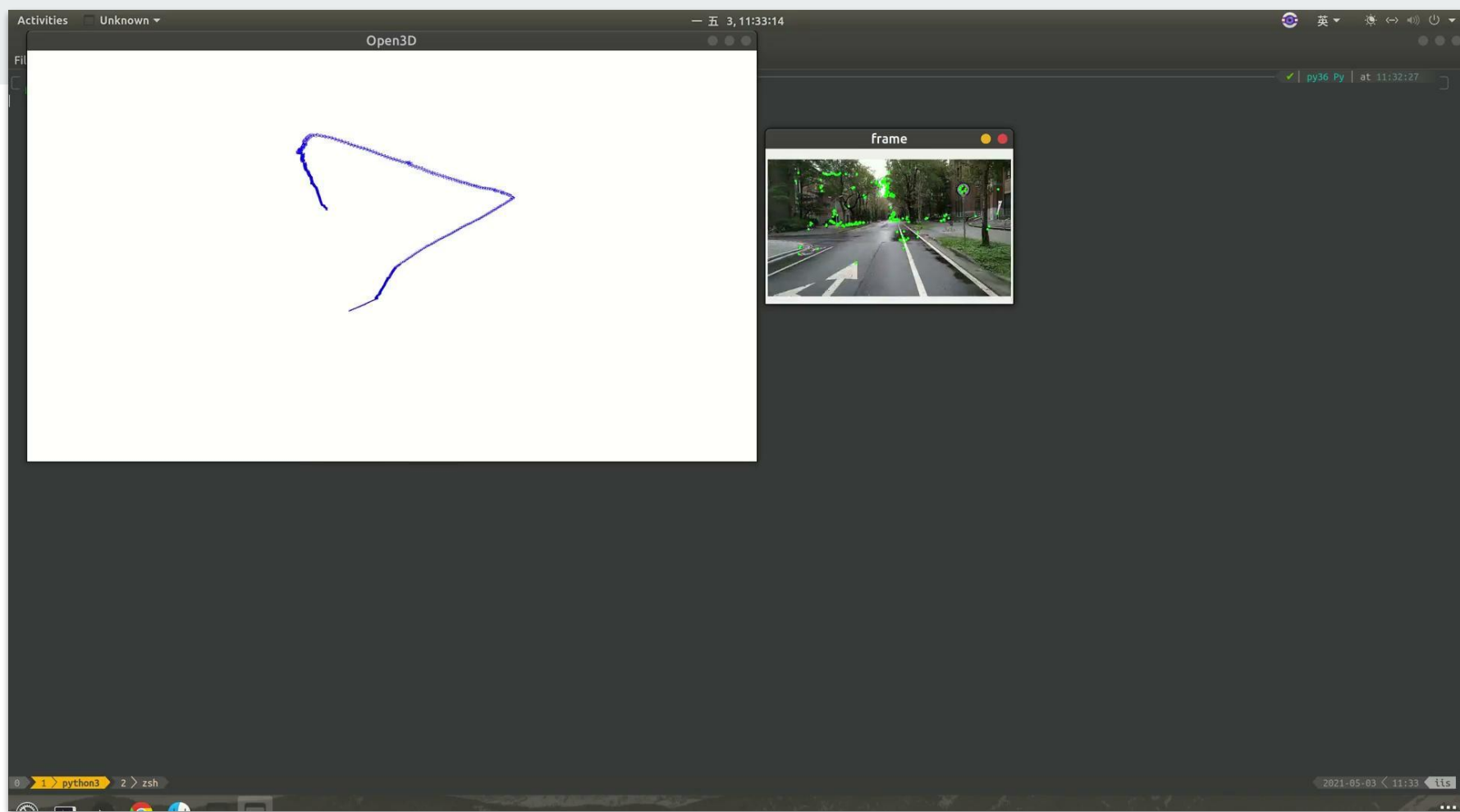
- Open3D LineSet [tutorial](#)
- The points of camera frame w.r.t. camera coord. sys., e.g.

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

$z_3 = 1$  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.





[\[Video Link\]](#) If you cannot watch here

# Report



- Briefly explain your method in each step
  - Camera calibration
  - Feature Matching
  - Pose from Epipolar Geometry (pseudo codes and comments)
  - Results Visualization
- Youtube link
  - Your video should be like the video shown in page 13
- 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



- 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 GitHub classroom.
- Programming Languages: Python (Python $\geq$ 3.8), (C++)
- 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.



# Thanks

If you have any question, please email [3dcv@csie.ntu.edu.tw](mailto:3dcv@csie.ntu.edu.tw)