COMP 423/523:
Computer Vision for Autonomous Driving
Homework 3

Due: June, 05 2022

Introduction

In this assignment, you will first answer some questions based on the concepts and the algorithms that we learned in the class. Then, you will implement parts of stereo estimation, optical flow, and visual odometry algorithms with OpenCV on KITTI by completing the template code attached.

1. (**20 points**) Pen-and-paper part
2. (**20 points**) Stereo Estimation
3. (**20 points**) `pykitti`
4. (**20 points**) Bullet Time Effect
5. (**20 points**) Monocular VO

Important Note: This is an individual assignment, not a group work. You can discuss your solutions with each other but you are supposed to work on the code on your own.

0.1 Setup: 3D

In this homework, you will create mesh files (the ones with 'ply' extension) for storing 3D data. You can visualize 3D data using a program such as MeshLab or Open3D. We recommend you to visualize 'ply' files in your local machine as it is challenging in a notebook or on the cluster.

1. MeshLab:
    - Install MeshLab: https://www.meshlab.net/download

- Visualize ply files: dragging 'ply' file into an open MeshLab window should work.

2. Open3D:
   - Install Open3D: http://www.open3d.org/docs/release/getting_started.html
   - Visualize ply file: http://www.open3d.org/docs/latest/tutorial/Basic/file_io.html#Mesh

0.2 Setup: `ARFlow`

To install `ARFlow` you can follow these steps either on Colab or on your own machine to make it work:

- Make sure you have python 3.6 or 3.7 installed.
- Download the repository:
  `wget https://github.com/lliuz/ARFlow/archive/master.zip`
- Unzip the repository:
  `unzip master.zip`
- Navigate to the repository:
  `cd ARFlow-master/`
- From the menu on the left, open `models/pwclite.py` file, comment the 6th line and uncomment the 7th line. Then, if you are using Colab you should not run the next cell but if you are on your local machine, install the requirements:
  `pip install -r requirements.txt`
  `pip install Pillow==6.2.2`
- Now you should be able to run the following in your notebook:
  `%run inference.py -m checkpoints/KITTI15/pwclite_ar.tar -s 384 640 -i examples/img1.png examples/img2.png`

1 Pen-and-Paper Questions

1. (**2 points**) What is the difference between depth and disparity? How are they related to each other mathematically?
2. (**2 points**) What do we assume to know in calibrated two-view geometry?
3. (**5 points**) Define the following terms related to epipolar geometry in your own sentences. Feel free to use the figure from the class, somewhere else, or draw a figure and upload its photo.

- Epipolar Line:
- Epipolar Plane:
- Epipole:
- Projection and Backprojection:
- Baseline:

4. (**2 points**) What is rectification and why do we do it? Find out an example of a rectification process from the literature, e.g. how is it performed on KITTI?

5. (**4 points**) Derive the matrices $M \in SE(3) \subset R^{4\times4}$ representing the following transformations:

   - Translation by the vector $T \in R^3$
   - Rotation by the rotation matrix $R \in R^{3\times3}$
   - Rotation by R followed by the translation T
   - Translation by T followed by the rotation R

   Hint: Remember that we can write the transformation matrix M for a given rotation matrix $R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$ and a translation vector $T = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$ as follows:
   $M = \begin{pmatrix} R & T \\ 0 & 1 \end{pmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$

6. (**5 points**) A classic ambiguity of the perspective projection is that one cannot tell an object from another object that is exactly twice as big but twice as far. Explain why this is true.

   Hint: Let $P = (X, Y, Z)$ be a point on the smaller object and $P' = (X', Y', Z')$ a point on the larger object. Define $X' = 2X, Y' = 2Y, Z' = 2Z$ and perspective projection as a function $p = \pi(P)$. How does π transform the world coordinate P to image coordinate p according to perspective projection? Repeat the same for P' and p'.

2 Practical Part

2.1 Stereo Estimation

In `stereo.ipynb`, we provide the code for computing stereo matching on OpenCV, run it and visualize the output disparity map. Your task is to implement the 'triangulate()' function to to convert the given disparity map to a point cloud using known parameters on KITTI.

Visualize the 3D point clouds using MeshLab or Open3D. Note that you might have to rotate the point cloud a little bit until you can make sense of it, especially with the estimated disparity map where there are outliers.

2.2 `pykitti`

There is a nice repository which serves as a development kit for KITTI in python called `pykitti`: https://github.com/utiasSTARS/pykitti.

Install it and repeat the steps in `mvs.ipynb` with the provided sequence to see what kind of properties of the dataset is available with pykitti. After that, you will compute stereo as you did in the first part but this time by using pykitti.

2.3 Bullet Time Effect

In this question, you will estimate optical flow using `ARFlow` (or you can use some other flow method if you cannot make it work) and then use it to interpolate between two images.

First, calculate the optical flow between two images, e.g. provided KITTI images in the example folder, using the `ARFlow` as illustrated by the example in `inference.py`. Then, synthesize 10 novel frames between the two images using linear interpolation.

Hint: First, compute the flow and then divide it by the number of frames. Then, at each step, warp the image incrementally to obtain images in between. You can use OpenCV's `remap()` function for warping or have a look at the `flow_warp` in `utils/warp_utils.py` file of the `ARFlow`.

- Where does the interpolation work well and where does it fail?
- Please also hand-in your interpolated images or create a small movie.

(Bonus) Multi-frame Bullet Time Sequence

If you had fun with the previous question, you might want to extend this effect to several frames as in the original Matrix movie!

Use consecutive images from a KITTI sequence or capture several images along a smooth camera trajectory and smoothly vary the pose of the objects in between. Between each two adjacent frames, interpolate an additional 10 to 100 frames and concatenate all images to one long slow motion video sequence.

2.4 Monocular VO

For each consecutive frame pair in the sequence, you will compute the relative pose between the frames and visualize it. You will use:

- pykitti code similar to what you wrote in par 2 to load the sequence with ground-truth info. (Check out the demo code for odometry dataset: github.com/utiasSTARS/pykitti/blob/master/demos/demo_odometry.py)
- OpenCV functions to compute and visualize the features and the essential matrix.

Please follow these steps to complete the assignment:

1. You can use the ORB Feature to do the feature matching: `orb = cv2.ORB_create()` to create the ORB object and then `orb.detectAndCompute()` to find the keypoints and descriptors on both frames.
2. You can use brute-force matcher to match ORB descriptors:
   `bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)`
3. After matching the descriptors, sort the matched keypoints.
4. Draw matches on the two images using the `cv2.drawMatches()` function.
5. Compute the essential matrix using the `cv2.findEssentialMat()` function. Note that you need the matching points and the instrinsics for this function.
6. Extract the rotation and translation from the essential matrix using the `cv2.recoverPose()` function.
7. Multiply the estimated rotation and translation with the previous rotation and translation. Initialize rotation to identity and translation to zeros on the first frame.
8. Display the current image with the keypoints on it using the `cv2.drawKeypoints()` function.
9. Update the previous rotation and translation as the current rotation and translation.
10. Draw the estimated trajectory as blue and ground-truth trajectory as green. You can use the `cv2.circle()` function.

You can create a video of your visualization of images and poses for the provided sequence.

Bonus: Compute the absolute trajectory error between the estimated trajectory and the ground-truth trajectory.

Some examples repositories that might be useful:

- bitbucket.org/castacks/visual_odometry_tutorial/src/master/visual-odometry
- github.com/uoip/monoVO-python