STM - Recruitment Assignment

Introduction

This assignment consists of different tasks to evaluate academic reading abilities, C++ coding skills and knowledge in the field. There are 5 different tasks in 4 parts. You are expected to submit for only 3 of them. More clearly, you are expected to analyze one of the papers provided in Part I, implement a solution for Part II and submit a deliverable either for Part III or Part IV.

Solving both of the tasks included in Part III and Part IV are optional since it requires background knowledge for two different approaches. However, if you have such knowledge, it is recommended to present it after completing the compulsory parts you have chosen.

Part I : Paper Analysis

One of the following papers should be analyzed and a report about its important points should be prepared. The length of the report should not exceed a few paragraphs as long as it contains the well explained key notes of the paper.

Optional: A discussion about the usage of these methodologies on a high-altitude aerial vehicle can be added to the report.

Paper I

Paper: Digging Into Self-Supervised Monocular Depth Estimation (https://arxiv.org/abs/1806.01260)

Abstract: "Per-pixel ground-truth depth data is challenging to acquire at scale. To overcome this limitation, self-supervised learning has emerged as a promising alternative for training models to perform monocular depth estimation. In this paper, we propose a set of improvements, which together result in both quantitatively and qualitatively improved depth maps compared to competing self-supervised methods.
Research on self-supervised monocular training usually explores increasingly complex architectures, loss functions, and image formation models, all of which have recently helped to close the gap with fully-supervised methods. We show that a surprisingly simple model, and associated design choices, lead to superior predictions. In particular, we propose (i) a minimum reprojection loss, designed to robustly handle occlusions, (ii) a full-resolution multi-scale sampling method that reduces visual artifacts, and (iii) an auto-masking loss to ignore training pixels that violate camera motion assumptions. We demonstrate the effectiveness of each component in isolation, and show high quality, state-of-the-art results on the KITTI benchmark."

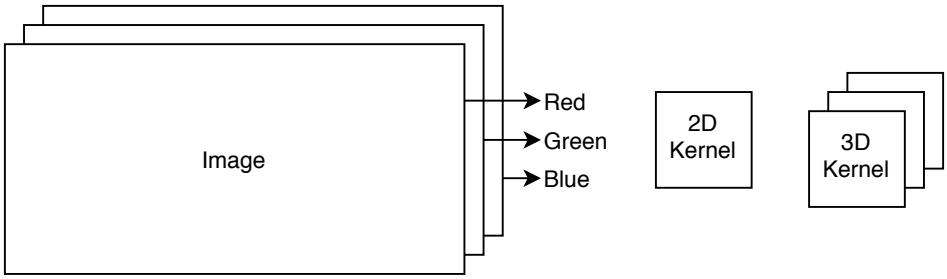

Figure 1: Multichannel image, 2D kernel and 3D kernel

Paper II

Paper: LSD-SLAM: Large-Scale Direct Monocular SLAM (https://link.springer.com/chapter/10.1007/978-3-319-10605-2_54)

Abstract: "We propose a direct (feature-less) monocular SLAM algorithm which, in contrast to current state-of-the-art regarding direct methods, allows to build large-scale, consistent maps of the environment. Along with highly accurate pose estimation based on direct image alignment, the 3D environment is reconstructed in real-time as pose-graph of keyframes with associated semi-dense depth maps. These are obtained by filtering over a large number of pixelwise small-baseline stereo comparisons. The explicitly scale-drift aware formulation allows the approach to operate on challenging sequences including large variations in scene scale. Major enablers are two key novelties: (1) a novel direct tracking method which operates on sim(3), thereby explicitly detecting scale-drift, and (2) an elegant probabilistic solution to include the effect of noisy depth values into tracking. The resulting direct monocular SLAM system runs in real-time on a CPU."

Part II: Image Filtering with Multi Channel Kernel

Image filtering is the most fundamental task for digital image processing systems. Almost all frameworks in this area support 2D convolution operation for image filtering. However, they limit the user with single-channel input image and convolution kernel. Although a color image is comprised of multiple channels (Red, Green, Blue channels for RGB color space), frameworks handle the single-channel image requirement when all channels are going to be filtered with the same single-channel 2D kernel. However, the image needs to be separated into single-channel images when channels are going to be filtered with different 2D kernels. The need for separation can be eliminated with a convolution with 3D (multi-channel) kernel. For multi-channel image, 2D Kernel and 3D kernel, refer to Figure 1. In this part, your duty is to implement convolution operation with a 3D kernel and convolve the provided image with the 3D kernel given in Figure 2 considering the following restrictions;

- Your implementation must be in C++ programming language. You can use image processing / computer vision libraries only for reading / writing images from / to files. It is allowed to use libraries for different purposes such as multi-threading.
- You can split the image data semantically; however, splitting it into several single-channel images is not allowed. The input and output image of the convolution operation must be multi-channel. For intermediate steps, you can separate kernel into as many parts as you want.

- You are required to implement the same operation using OpenCV without any restriction, and compare the results in terms of accuracy and time.
- There is no restriction in border type and anchor point as long as both implementations result in the same output.

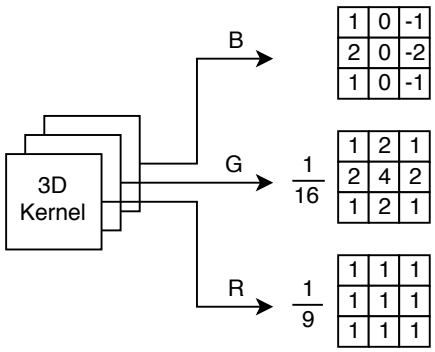

Figure 2: Input Kernel

Note that: This part is for testing the programming abilities; hence, try to present best of yourself in the important points which are readability, code structure, algorithm design, correctness and execution time.

Part III : Code Migration

In this part, you are expected to migrate the codebase of the **Paper I** from PyTorch to Tensorflow.

- **Link for the codebase;** GitHub - nianticlabs/monodepth2: Monocular depth estimation from a single image (https://github.com/nianticlabs/monodepth2)
- We are expecting the following modules in your implementation; monocular training, prediction (inference) and evaluation. Your code should successfully create models from scratch by training KITTI dataset and evaluate these results on same dataset.
- Since the examination will be done in a white-box manner, we will evaluate your code whether it works or not. Therefore, delivering a perfectly working project will not be the main criteria for evaluation. However, a working project will be very beneficial for your evaluation process.
- In general, there are no strict rules for this assignment so feel free to implement it your own way and think it is as a portfolio to display your talent.
- Since the training processes is only feasible with a decent NVIDIA GPU, we will send you a certain amount of Amazon EC2 credits to train your model on a EC2 cloud computer if you need a GPU. However, we are not recommending this solution because of the learning curve and possible deployment problems.
- Following github repos may be helpful in your project;
  - GitHub - mrharicot/monodepth: Unsupervised single image depth prediction with CNNs (https://github.com/mrharicot/monodepth)

- GitHub - tinghuiz/SfMLearner: An unsupervised learning framework for depth and ego-motion estimation from monocular videos (https://github.com/tinghuiz/SfMLearner)
- GitHub - ClubAI/MonoDepth-PyTorch: Unofficial implementation of Unsupervised Monocular Depth Estimation neural network MonoDepth in PyTorch (https://github.com/ClubAI/MonoDepth-PyTorch)

Part IV: Fundamental Matrix Estimation

In this part, you are expected to implement different algorithms for estimating the fundamental matrix between two images. Your implementation must be in C++ programming language. **The only restriction is that the usage of a library call for a function like OpenCV's *findFundamentalMat* is forbidden.**

You are expected to implement all of the following 5 functions.

Eight-Point Algorithm

Write a function implementing the eight-point algorithm to return the fundamental matrix from point correspondences between two images. Enforce the rank-two constraint for the fundamental matrix using singular value decomposition.

```
F_Matrix_Eight_Point()

Input Arguments:
        points1 (x,y) - pixel coordinates in the first image,
                corresponding to points2 in the second image
        points2 (x,y) - pixel coordinates in the second image,
                corresponding to points1 in the first image
Output:
        F - the fundamental matrix between the first image and the second image
```

Normalized Eight-Point Algorithm

Write a function implementing the normalized eight-point algorithm. In this case, for each image, the pixel coordinates are translated and scaled to ensure that the centroid of the image points is 0, and the standard deviation of the image points is 1. Again, enforce the rank-two constraint for the fundamental matrix. (Note that the normalized eight-point algorithm is expected to be more accurate than the eight-point algorithm due to better conditioning.)

```
F_Matrix_Normalized_Eight_Point()

Input Arguments:
        points1 - (x,y) pixel coordinates in the first image,
                corresponding to points2 in the second image
        points2 - (x,y) pixel coordinates in the second image,
                corresponding to points1 in the first image
Output:
        F - the fundamental matrix between the first image and the second image
```

Epipolar Lines

Write a function displaying the epipolar lines for selected point correspondences.

```
Plot_Epipolar_Lines()

Input Arguments:
        points1 - (x,y) pixel coordinates in the first image,
                corresponding to points2 in the second image
        points2 - (x,y) pixel coordinates in the second image,
                corresponding to points1 in the first image
        img1 - the first image
        img2 - the second image
        F - the fundamental matrix between the first image and the second image
Output:
        Display two images with matching points and corresponding epipolar lines
```

Image Rectification

Write a function generating homographies that transform the images such that the epipolar lines are parallel to the horizontal axis (i.e., image rectification), and displays the rectified images.

```
Rectify_Images()

Input Arguments:
        points1 - (x,y) pixel coordinates in the first image,
                corresponding to points2 in the second image
        points2 - (x,y) pixel coordinates in the second image,
                corresponding to points1 in the first image
        img1 - the first image
        img2 - the second image
        F - the fundamental matrix between the first image and the second image
Output:
        H1 - The homography applied to the first image
        H2 - The homography applied to the second image
        Display two images with matching points and corresponding epipolar lines
```

Using the image pairs (given in folder /Part4data/), test your functions. To do so, first write a script to manually mark and save the image correspondences. Then, for each image pair, calculate the fundamental matrix, generate images with epipolar lines drawn and the rectified images.

RANSAC

Now, you will automatically determine the correspondences and get rid of the outliers using the RANSAC algorithm. Use any computer vision library to detect and match the features (such as ORB and SIFT). Write a function implementing the normalized eight-point algorithm with RANSAC technique to eliminate the mismatches. For each image pair, calculate the fundamental matrix, generate images with epipolar lines drawn, and the rectified images.

(The RANSAC technique works as follows: You randomly select a minimum set of correspondences out of all correspondences, solve for the fundamental matrix using the selected correspondences, and count the

number of inliers that "agree" with the fundamental matrix. Repeat the process for a certain number of iterations, and choose the best case that has the most number of inliers. Remove the outliers and calculate the fundamental matrix one more time using only the inliers.)

F_Matrix_RANSAC()

```
Input Arguments:
        points1 - (x,y) pixel coordinates in the first image,
                corresponding to points2 in the second image
        points2 - (x,y) pixel coordinates in the second image,
                corresponding to points1 in the first image
        threshold - a value above which the RANSAC technique sets the correspondence as ou
Output:
        F - the fundamental matrix between the first image and the second image
```

Duration and Submission

- The deadline is 10 days after you received this assignment. At the end of this duration, you can submit your work even if it is incomplete.
- In case you have any questions, feel free to contact us anytime via the email address provided you by Human Resources Officer who contacted you.
- For C++ implementations, you are required to deliver a Makefile along with all dependencies.
- For Python implementation (in case you choose Part III), it is recommended to use conda environment to wrap your dependencies.
- You are also required to provide an installation and run guide for the evaluation of your code.
- If you successfully trained a model for Part III, you can deliver the model via Google Drive.
- You can deliver your code via a private repository link. You can use GitHub, Bitbucket or GitLab. Although we are strongly recommending to share this project on your personal git accounts (since it may help you in your future job applications), please keep them private during the evaluation process.
- After the evaluation of the project implementations, several questions about them will be asked to successful candidates during their job interview. So please be mind that every piece of code you deliver may be asked you on your job interview.