Handover Document

Data:

https://drive.google.com/drive/folders/1tI5g3iW628dM25E5n5vtMuqgVsXPv_zj?usp=drive_link
Include docker and data for augmentation.

Environment Setup:

HscNet:

Install conda environment.

CUDA:

On 20 series GPU, use CUDA 10 and "environment.yml".
On 30 series GPU, use CUDA 11 and "environment_3090.yml".

```
cd v4-hscnet
conda env create -f environment.yml
conda activate hscnet

cd ./pnpransac
python setup.py build_ext –inplace
```

If the GPU is 30 series or higher, change to use "environment_3090.yml" instead of "environment.yml".
"environment_3090.yml" is in the handover folder.

If you meet fatal error "numpy/arrayobject.h: No such file or directory" during building pnpransac, add "np.get_include()" in the "include_dirs" variable in setup.py.

setup.py:

```
import numpy as np
……
ext_modules = [include_dirs=[……, np.get_include()], ……]
```

Open3D Rendering Tool Setup:

Follow Open3D Docker ReadMe: https://hackmd.io/@VqRB_pbdS82ryqhKhyYZQQ/BJlNkVdqP.

Set the docker mount directory in "open3d-start/tools/run.sh":

```
......
Open3D_HOST=/home/<your home directory>/open3d_docker
```

If the docker can't mount to the folder (after activate the docker, there is empty), change the home directory "~/" to absolute directory "/home/XXX/" in "open3d-start/tools/run.sh".

Reference:

HscNet:
https://github.com/AaltoVision/hscnet

Open3D Docker:
https://hackmd.io/@VqRB_pbdS82ryqhKhyYZQQ/BJlNkVdqP

Augmentation rendering pipeline:
https://drive.google.com/drive/folders/1Km9n3yKPOG_O4hAA5MLQclBxUDsoIARy

Data Structure in Open3D Docker:

Put the provided data into the docker directory. Make sure the data for Open3D is following this structure.

```
.../open3d_docker/Open3D/examples/TestData/
    7scenes/
        [chess, fire, heads, office, pumpkin, redkitchen, stairs]/
            train_ori/          (original training data for augmentation)
                color/          (RGB image)
                depth/          (depth map)
                depth_cali/     (depth map after cali)
                pose/           (camera pose)
    12scenes/
        [5a, 5b , bed, gates362, gates381, kitchen1,kitchen2, living1, living2, lounge, luke, manolis]/
            train/              (original training data for augmentation)
                color/          (RGB image)
                depth/          (depth map)
                pose/           (camera pose)
            val/                (original testing data, non use)
                color/          (RGB image)
                depth/          (depth map)
                pose/           (camera pose)
```

The data is in the Scoly's HDD, too. Their filenames are arranged by Scoly. You just need to copy the data to your docker directory. "depth_cali" data for 7-Scenes can be generated by the code, but I have put it in the handover directory, so you can just copy it to your docker directory.

p.s. The data can be downloaded from the provided Google Drive Link.
p.s. If you want to copy from the HDD, the data is in the docker at Scoly's home directory.

Data Structure in Data Directory for HscNet:

Make sure the data for HscNet is following this structure.

```
…/data/
    7Scenes/
        sensorTrans.txt         (depth camera calibration parameters)
        train.txt               (data list for 7-Scenes 100% original training data)
        test.txt                (data list for 7-Scenes original testing data)
        train_few_shoot_*.txt   (data list for 7-Scenes few-shot original training data)
        [chess, fire, heads, office, pumpkin, redkitchen, stairs]/
            translation.txt     ([scene] average translation position)
            centers.npy         (cluster center coordinates)
            chess_aug_*.txt     (data list for the [scene] few-shot augmented training data)
            seq-*/              (original data)
                frame-*.color.png       (RGB image)
                frame-*.depth.png       (depth map, 1 channel gray scale)
                frame-*.label.png       (label map, 1 channel gray scale)
                frame-*.pose.txt        (camera pose)
            train_aug_*/        (augmented data)
                *.color.png     (augmented RGB image)
                *.color_[n3|n4|t3|t4].png     (augmented RGB image with RGB inpainting)
                *.color_noise.png             (augmented RGB image with RGB noise)
                *.depth.png     (augmented depth map, 1 channel gray scale)
                *.label.png     (new label map for augmented image, 1 channel gray scale)
                *.mask.png      (mask for invalid depth, 1 channel gray scale)
                *.pose.txt      (augmented camera pose)
                *.rotate.txt    (pose adjustment from original camera pose to augmented pose)
    12Scenes/
        [apt1, apt2, office1, office2]/
            [scene]/
                centers.npy             (cluster center coordinates)
                train.txt               (data list for the [scene] 100% original training data)
                test.txt                (data list for the [scene] original testing data)
```

train_few_shoot_*.txt (data list for the [scene] few-shot original training data)
aug_*_few_shoot_*.txt (data list for the [scene] few-shot augmented training data)
data/ (original data)
frame-*.color.png (RGB image)
frame-*.depth.png (depth map, 1 channel gray scale)
frame-*.label.png (label map, 1 channel gray scale)
frame-*.pose.txt (camera pose)
train_aug_*/ (augmented data)
*.color.png (augmented RGB image)
*.color_[n3|n4|t3|t4].png (augmented RGB image with RGB inpainting)
*.color_noise.png (augmented RGB image with RGB noise)
*.depth.png (augmented depth map, 1 channel gray scale)
*.label.png (new label map for augmented image, 1 channel gray scale)
*.mask.png (mask for invalid depth, 1 channel gray scale)
*.pose.txt (augmented camera pose)
*.rotate.txt (pose adjustment from original camera pose to augmented pose)

Data Augmentation Pipeline and Command:

Activate Open3D docker:

```
cd open3d-start/tools
sudo ./build.sh          (this just need to run at the first time to activate the docker)
sudo ./run.sh
sudo ./attach.sh

cd Open3D/build
pip3 install setuptools wheel
make install-pip-package
pip3 install nibabel scikit-learn

python3 [python file] ...... (run your python code)
```

Augmented image rendering:

1. Preprocessing
   A. Calibration the depth map for 7-Scenes data.
   There is difference between depth sensor and RGB camera on Kinect, so 7-Scenes depth map should be calibrated. The parameters are provided by DSAC.

The calibrated depth is saved in the name format of number without “frame-***”. It is not “seq-*/frame-*” format because it is for data augmentation. The new depth maps are saved in the “v4-calibration_depth” directory. I have prepared the calibrated depth map “depth_cali” data in handover directory. So you don’t need to calibrate it.

```
cd v4-calibration_depth

(setting the number of thread for data loader in "depth_inpainting.py")

python calibration_depth.py \
        --dataset 7S \                (dataset)
        --scene heads \               (scene name)
        --data_path ../data \         (data directory)
        --batch_size 8                (well, the main issue is number of threads for data loader,
                                      make sure you set the max available number of thread)

mkdir depth_cali
mv seq*/* depth_cali
mv depth_cali ~/open3d_docker/Open3D/examples/TestData/7scenes/heads/train_ori

p.s. It can calibrate the depth with depth inpainting, too. Just edit the depth map filename postfix from
".depth.png" to ".depth_inpaint.png".

mkdir depth_fst_inpaint_sec_cali
mv seq*/* depth_fst_inpaint_sec_cali
mv depth_fst_inpaint_sec_cali ~/open3d_docker/Open3D/examples/TestData/7scenes/heads/train_ori
```

B. Adding depth inpainting.

The new depth with inpainting is saved in the scene’s data directory with “depth_inpaint” posefix. In “cv2.inpaint function”, there are “cv2.INPAINT_NS” and “cv2.INPAINT_TELEA” 2 types of algorithm. And you can set the radius for the look up range to inpainting. My experience is setting “cv2.INPAINT_NS” and “radius = 3” for depth inpainting, as I write in the python file.

```
cd v4-depth-inpainting

(setting the number of thread for data loader in "depth_inpainting.py")

python depth_inpainting.py \
        --dataset 7S \
        --scene heads \
        --data_path ../data \
```

```
        --batch_size 8
```

2. Data augmentation

The python files are usually put in “open3d_docker/Open3D/examples/Python/Advanced”. Edit the python file setting and run it.

```
python3 v5_knn_*.py
```

How to adjust point size of 3D point cloud during rendering:

open3d_docker/Open3D/src/Open3D/Visualization/Visualizer/RenderOption.h

In the code at the 68 row, this variable “const double POINT_SIZE_DEFAULT = 2.0;”. You can change it, and remember to use the “make” command to compile Open3D again.

The augmented images are in “open3d_docker/Open3D/examples/TestData”.

Leave the docker environment, change the directory to the augmented images directory

Change the file owner and user group (the original permissions is root).

```
chown -R [username]:[username] <folder>
```

Move the augmented images folder to the data directory. (Ex: data/7Scenes/)

Edit the directory setting in “v4_rename_number2frame_aug.py”.

```
......
data_dir = 'data'                                   (the data directory)
dataset = '7Scenes'                                 (the dataset)
scene = 'fire'                                      (the scene)
aug = 'train_aug_0.5-64-k4_cali-set-point-size-2.0' (the augmented data folder)
......
```

This python file renames the augmented images and moves them to correct directory.

```
python v4_rename_number2frame_aug.py
```

3. Make the data list for augmented data.

100% data:

```
python v4_generate_list.py
```

few-shot data for 7-Scenes:

```
python v5_generate_list.py
```

few-shot data for 12-Scenes:

```
python v5_12S_generate_list.py
```

4. Generate label map for augmented data

```
cd v4-hscnet-label-aug

7-Scenes:
python train.py \
        --dataset 7S \
        --scene heads \
        --data_path ../data \
        --batch_size 8 \
        --scene_txt_postfix _few_shoot_1          (for few-shot data, the postfix of data list filename)

12-Scenes:
python train.py \
        --dataset 12S \
        --scene apt1/living \
        --data_path ../data \
        --batch_size 8 \
        --scene_txt_postfix _few_shoot_0.5        (for few-shot data, the postfix of data list filename)
```

5. Add RGB noise or RGB inpainting

Set the scene, folder, and data list in the python file and run it. In “cv2.inpaint function”, there are “cv2.INPAINT_NS” and “cv2.INPAINT_TELEA” 2 types of algorithm. And you can set the radius for the look up range to inpainting.

7-Scenes:

v4_add_RGB_noise.py

v4_rgb_inpainting.py

12-Scenes:

v4_12S_add_RGB_noise.py

v4_12S_rgb_inpainting.py

HscNet command:

v4-hscnet: For testing and training on original data.

v4-hscnet-aug: For training on augmented data.

They are different on data loader. Commands are the same.

Training:

```
python train.py \
        --model hscnet \
        --dataset 7S \                              (dataset)
        --scene chess \                             (scene)
        --n_iter 300000 \                           (total training iteration)
        --data_path ../data \                       (data directory)
        --batch_size 1 \                            (batch size, 1 is better)
        --fixed_save_freq False \        (if yes, save ckpt for fixed epoch; if no, save for each 20% epoch)
        --save_freq 1 \                             (how many fixed epoches to save checkpoint)
        --init_lr 1e-4 \                            (initial learning rate)
        --lr_decade True \                          (w/ or w/o learning rate decade)
        --lr_decade_reserve_iter 200000 \           (how many iterations leave for LR decade)
        --lr_decade_iter 50000 \                    (learning rate decade step)
        --scene_txt_postfix _aug_48_few_shoot_0.5   (this is for few-shot data, data list postfix)
```

Parameters:

100% Original data training iteration: 300,000

1% and 0.5% few-shot original data training iteration: 30,000

1% and 0.5% few-shot augmented data: 300,000

Initial learning rate: 1e-4

Iterations leave for learning rate decade: 4/6 * Total iteration

Learning rate decade step: 1/6 * Total iteration

Testing:

```
python eval.py --model hscnet --dataset [7S|12S] --scene <Scene Name> --checkpoint <Checkpoint
Directory> --data_path <Data Directory>
```

The evaluation result is shown on the screen, you can use “>” to redirect the output.

Example:

```
python eval.py --model hscnet --dataset 7S --scene chess
--checkpoint ../v4-hscnet-aug/checkpoints/7S-chess-lr0.0001-iters300000-bsize1-aug1-_aug_32_few_shoot_0.5
/model_469.pkl --data_path ../data > 469.txt
```

Or testing all the checkpoints:

```
for i in $(seq 1 5)
do python eval.py --model hscnet --dataset 7S --scene chess
--checkpoint ../v4-hscnet-aug/checkpoints/7S-chess-lr0.0001-iters300000-bsize1-aug1-_aug_32_few_shoot_0.5
/model_`expr $i \* 93`.pkl --data_path ../data > `expr $i \* 93`.txt
done
```

Collect the accuracy and median error:

```
for i in $(seq 1 5); do sed -n '2001p' `expr $i \* 93`.txt; done > acc.txt
for i in $(seq 1 5); do sed -n '2002p' `expr $i \* 93`.txt; done > err.txt
```

Confidence-based Sampling:

Directory:

ransac-v0, ransac-v1-lbl_1

Introduction:

ransac-v0: Analyze the original PnP-RASAC reference time

ransac-v1-lbl_1: Analyze PnP-RASAC with confidence-based sampling reference time

Command:

ransac-v0:

```
rm result.txt
python eval_ransac.py --model hscnet --dataset [7S|12S] --scene <Scene Name> --checkpoint <Checkpoint
Directory> --data_path <Data Directory> --num_hyp <Number of Hypotheses>
```

Example:

```
rm result.txt
python eval_ransac.py --model hscnet --dataset 7S --scene chess --checkpoint ~/1_thesis/v4-hscnet-aug/checkpoints/few-shot-7S-rgb-noise-dif-m/7S-chess-lr0.0001-iters300000-bsize1-aug1-_aug_64-k4_few_shoot_0.5-2.0-rgb-n4/model_235.pkl --data_path ../../data --num_hyp 256
```

ransac-v1-lbl_1:

```
rm result.txt
python eval_ransac.py --model hscnet --dataset [7S|12S] --scene <Scene Name> --checkpoint <Checkpoint Directory> --data_path <Data Directory> --num_hyp <Number of Hypotheses> --num_top <Number of Sample Points>
```

Example:

```
rm result.txt
python eval_ransac.py --model hscnet --dataset 12S --scene apt1/kitchen --checkpoint /data/checkpoints/final/12S/12S-apt1.kitchen-lr0.0001-iters300000-bsize1-aug1-_few_shoot_0.5-5.0-rgb-n4/model_2344.pkl --data_path ../../data --num_hyp 256 --num_top 2400
```

Parameters:

Number of hypotheses is 256, based on HscNet. You can also test for different number of hypotheses. The command test for number of hypotheses: 1, 2, 4, 8, 16, …, 256 is as follow:

```
for i in $(seq 0 8)
do
rm result.txt
python eval_ransac.py --model hscnet --dataset [7S|12S] --scene <Scene Name> --checkpoint <Checkpoint Directory> --data_path <Data Directory> --num_hyp $((2 ** $i))
done
```

Number of sample points is 2400 (total 4800 points, so it is 50%).