CSE252C: Homework 1

Computing Resources

Please read the README file of this repository for the instructions

Instructions

1. Attempt all questions.
2. Please comment all your code adequately.
3. Include all relevant information such as text answers, output images in notebook.
4. **Academic integrity:** The homework must be completed individually.
5. **Submission instructions:**
   (a) Submit the notebook and its PDF version on Gradescope.
   (b) Rename your submission files as Lastname_Firstname.ipynb and Lastname_Firstname.pdf.
   (c) Correctly select pages for each answer on Gradescope to allow proper grading.
6. **Due date:** Assignments are due Mon, May 4, by 4pm PST.

Question 1: Warm Up

We will first try SFM using the original implementation from `libviso2`[8],[9]. We will test on a dataset containing 300 images from one sequence of the KITTI dataset with ground-truth camera poses and camera calibration information.

Run the SFM algorithm using the following script. You are required to report two error metrics. The error metric for rotation is defined as the mean of Frobenius norm of the difference between the ground-truth rotation matrix and predicted rotation matrix. The error metric for translation is defined as mean of the L2 distance. Both errors will be printed on the screen as you run the code. **(5 points)**

In [2]:
```
import os
import sys
# change your base path
os.chdir('./pyviso/') # './'
print(os.getcwd())
```

```
/home/sgaba/sfm/pyviso
```

In [3]:

```
import os
import numpy as np
import pandas as pd
np.set_printoptions(precision=4)
np.set_printoptions(suppress=True)
import viso2
import matplotlib.pyplot as plt
from skimage.io import imread
import time

def errorMetric(RPred, RGt, TPred, TGt):
    diffRot = (RPred - RGt)
    diffTrans = (TPred - TGt)
    errorRot = np.sqrt(np.sum(np.multiply(diffRot.reshape(-1), diffRot.reshape
(-1))))
    errorTrans = np.sqrt(np.sum(np.multiply(diffTrans.reshape(-1), diffTrans.r
eshape(-1))))

    return errorRot, errorTrans

if_vis = False # set to True to do the visualization per frame; the images wil
l be saved at '.vis/'. Turn off if you just want the camera poses and errors
if_on_screen = False # if True the visualization per frame is going to be disp
layed realtime on screen; if False there will be no display, but in both optio
ns the images will be saved

# parameter settings (for an example, please download
# dataset_path = '../dataset_SfM/'
dataset_path = '/datasets/cse152-252-sp20-public/dataset_SfM' # On the ``ieng
6.ucsd.edu`` server
img_dir      = os.path.join(dataset_path, 'sequences/00/image_0')
gt_dir       = os.path.join(dataset_path, 'poses/00.txt')
calibFile    = os.path.join(dataset_path, 'sequences/00/calib.txt')
border       = 50;
gap          = 15;

# Load the camera calibration information
with open(calibFile) as fid:
    calibLines = fid.readlines()
    calibLines = [calibLine.strip() for calibLine in calibLines]

calibInfo = [float(calibStr) for calibStr in calibLines[0].split(' ')[1:]]
# param = {'f': calibInfo[0], 'cu': calibInfo[2], 'cv': calibInfo[6]}

# Load the ground-truth depth and rotation
with open(gt_dir) as fid:
    gtTr = [[float(TrStr) for TrStr in line.strip().split(' ')] for line in fi
d.readlines()]
gtTr = np.asarray(gtTr).reshape(-1, 3, 4)

# param['height'] = 1.6
# param['pitch']  = -0.08
# param['match'] = {'pre_step_size': 64}
first_frame  = 0
last_frame   = 300
epi = 1e-8
```

```
# init visual odometry
params = viso2.Mono_parameters()
params.calib.f = calibInfo[0]
params.calib.cu = calibInfo[2]
params.calib.cv = calibInfo[6]
params.height = 1.6
params.pitch = -0.08


first_frame  = 0
last_frame   = 300

# init transformation matrix array
Tr_total = []
Tr_total_np = []
Tr_total.append(viso2.Matrix_eye(4))
Tr_total_np.append(np.eye(4))

# init viso module
visoMono = viso2.VisualOdometryMono(params)

if if_vis:
    save_path = 'vis'
    os.makedirs(save_path, exist_ok=True)

    # create figure
    fig = plt.figure(figsize=(10, 15))
    ax1 = plt.subplot(211)
    ax1.axis('off')
    ax2 = plt.subplot(212)
    ax2.set_xticks(np.arange(-100, 100, step=10))
    ax2.set_yticks(np.arange(-500, 500, step=10))
    ax2.axis('equal')
    ax2.grid()
    if if_on_screen:
        plt.ion()
    else:
        plt.ioff()
```

In [ ]:

```
# for all frames do
if_replace = False
errorTransSum = 0
errorRotSum = 0
errorRot_list = []
errorTrans_list =[]

#initialize an empty df to report errors
errors_df = pd.DataFrame(columns = ["Mean Rotation Error","Mean Transition Err
or"])

for frame in range(first_frame, last_frame):
#     break
    # 1-based index
    k = frame-first_frame+1

    # read current images
    I = imread(os.path.join(img_dir, '%06d.png'%frame))
    assert(len(I.shape) == 2) # should be grayscale

    # compute egomotion
    process_result = visoMono.process_frame(I, if_replace)
    Tr = visoMono.getMotion()
    matrixer = viso2.Matrix(Tr)
    Tr_np = np.zeros((4, 4))
    Tr.toNumpy(Tr_np) # so awkward...

    # accumulate egomotion, starting with second frame
    if k > 1:
        if process_result is False:
            if_replace = True
            Tr_total.append(Tr_total[-1])
            Tr_total_np.append(Tr_total_np[-1])
        else:
            if_replace = False
            Tr_total.append(Tr_total[-1] * viso2.Matrix_inv(Tr))
            Tr_total_np.append(Tr_total_np[-1] @ np.linalg.inv(Tr_np)) # shoul
d be the same
            print(Tr_total_np[-1])

    # output statistics
    num_matches = visoMono.getNumberOfMatches()
    num_inliers = visoMono.getNumberOfInliers()
    matches = visoMono.getMatches()
    matches_np = np.empty([4, matches.size()])

    for i,m in enumerate(matches):
        matches_np[:, i] = (m.u1p, m.v1p, m.u1c, m.v1c)

    if if_vis:
        # update image
        ax1.clear()
        ax1.imshow(I, cmap='gray', vmin=0, vmax=255)
        if num_matches != 0:
            for n in range(num_matches):
                ax1.plot([matches_np[0, n], matches_np[2, n]], [matches_np[1,
```

```
n], matches_np[3, n]])
        ax1.set_title('Frame %d'%frame)

        # update trajectory
        if k > 1:
            ax2.plot([Tr_total_np[k-2][0, 3], Tr_total_np[k-1][0, 3]], \
                [Tr_total_np[k-2][2, 3], Tr_total_np[k-1][2, 3]], 'b.-', linew
idth=1)
            ax2.plot([gtTr[k-2][0, 3], gtTr[k-1][0, 3]], \
                [gtTr[k-2][2, 3], gtTr[k-1][2, 3]], 'r.-', linewidth=1)
        ax2.set_title('Blue: estimated trajectory; Red: ground truth trejector
y')

        plt.draw()

    # Compute rotation
    Rpred_p = Tr_total_np[k-2][0:3, 0:3]
    Rpred_c = Tr_total_np[k-1][0:3, 0:3]
    Rpred = Rpred_c.transpose() @ Rpred_p
    Rgt_p = np.squeeze(gtTr[k-2, 0:3, 0:3])
    Rgt_c = np.squeeze(gtTr[k-1, 0:3, 0:3])
    Rgt = Rgt_c.transpose() @ Rgt_p
    # Compute translation
    Tpred_p = Tr_total_np[k-2][0:3, 3:4]
    Tpred_c = Tr_total_np[k-1][0:3, 3:4]
    Tpred = Tpred_c - Tpred_p
    Tgt_p = gtTr[k-2, 0:3, 3:4]
    Tgt_c = gtTr[k-1, 0:3, 3:4]
    Tgt = Tgt_c - Tgt_p
    # Compute errors
    errorRot, errorTrans = errorMetric(Rpred, Rgt, Tpred, Tgt)
    errorRotSum = errorRotSum + errorRot
    errorTransSum = errorTransSum + errorTrans
    # errorRot_list.append(errorRot)
    # errorTrans_list.append(errorTrans)
    meanRotError = errorRotSum / (k-1+epi)
    meanTransError = errorTransSum / (k-1+epi)
    print('Mean Error Rotation: %.5f'%(meanRotError))
    print('Mean Error Translation: %.5f'%(meanTransError))

    errors_df.loc[len(errors_df)] = [meanRotError,meanTransError]

    print('== [Result] Frame: %d, Matches %d, Inliers: %.2f'%(frame, num_match
es, 100*num_inliers/(num_matches+1e-8)))

    if if_vis:
        # input('Paused; Press Enter to continue') # Option 1: Manually pause
 and resume
        if if_on_screen:
            plt.pause(0.1) # Or Option 2: enable to this to auto pause for a w
hile after daring to enable animation in case of a delay in drawing
        vis_path = os.path.join(save_path, 'frame%03d.jpg'%frame)
        fig.savefig(vis_path)
        print('Saved at %s'%vis_path)

        if frame % 50 == 0 or frame == last_frame-1:
            plt.figure(figsize=(10, 15))
```

```
            plt.imshow(plt.imread(vis_path))
            plt.axis('off')
            plt.show()


# input('Press Enter to exit')
```

1. Report the final rotation and translation error. **(2 points)**

- Final Frame Mean Rotation and Mean Translation Error are 0.00277 and 0.53721 respectively.

In [7]:
```
"""Not to be reported"""
# print("Rotation and Transition Error metric for all frames \n")
# errors_df.index = np.arange(1, len(errors_df) + 1)
# errors_df.index.name = "Frame Number"
# with pd.option_context('display.max_rows', None, 'display.max_columns', Non
e):
#     print(errors_df)
```

Out[7]:
```
'Not to be reported'
```

Then answer the questions below

1. In `libviso2`, the feature points are "bucketed" (`src/matcher. cpp : Line285 − 326`), which means in a certain area of region, the number of detected keypoint pairs should be within certain bounds. Why? **(3 points)**

- In bucketing, the image is divided into several nonoverlapping rectangles and in every bucket we keep a maximal number of feature points. First, the smaller number of features reduces the computational complexity of the algorithm which is an important prerequisite for real time applications. Second, this technique guarantees that the used image features are well distributed along the z-axis, i.e. the roll-axis of the vehicle. This turns out to be important for a good estimation of the linear and angluar velocities. The distribution of image features along the z-axis ensures that far as well as near features are used for the estimation process. This results in a precise estimation of the overall egomotion of the vehicle. Third, the used image features are uniformly distributed over the whole image. This benefits twice: In dynamic scenes where most of the detected features lie on independently moving objects, the technique guarantees that not all image features fall on independently moving objects but also on the static background. Second, the bucketing reduces the drift rates of the approach.

1. We have run SFM on a single camera, which means the scale of translation is unknown. However, as you may have observed, the predicted trajectory is still somehow similar to the ground-truth trajectory. How does `libviso2` handle this ambiguity (`viso_mono. cpp : Line245`)? **(5 points)**

- For SFM on a single camera or monocular vision, `libviso2` uses 8 point algorithm for fundamental matrix computation. For estimating the scale, it assumes that the camera is moving at a known and fixed height over ground. It computes the scaling factor by the ratio of estimated height and actual height (known). `libviso2` scales all the estimations with this computed scaling factor to handle monocular vision scale ambiguity.

1. Briefly explain the RANSAC algorithm used in `libviso2` (`viso_mono.cpp : Line113 – 129`). **(5 points)**

- For "param.ransac_iters" times:
  – Randomly pick 8 correspondences across two images
  – Compute Fundamental Matrix(F) using these 8 correspondences
  – Among all correspondences, count number of inliers with $x^T F x$ less than 'param.inlier_threshold'

  Pick the Fundamental Matrix with the largest number of inliers

Question 2: Using SIFT [4] for SFM

In the second task, you are required to use keypoints and feature descriptors from SIFT for SFM. The SIFT implementation can be found in directory `SIFT`.

(A) Go to `SIFT` directory and run `runSIFT.py` (e.g. `python runSIFT.py --input /datasets/cse152-252-sp20-public/dataset_SfM/sequences/00/image_0/`). You will save the detected keypoints and feature descriptors under the directory `SIFT`. For image `000abc.png`, the pre-computed features and keypoints should be saved in a `.npy` file named as `000abc_feature.npy`. The variable should be a $130 \times N$ matrix with `single` precision, where N is the number of feature points being detected. For each 130-dimensional feature vector, the first two dimensions are the location of the keypoints (column number first and then row number) on the image plane and the last 128 dimensions are the feature descriptor.

(B) Run the following script

In [ ]:
```
import runFeature
dataset_path = '/datasets/cse152-252-sp20-public/dataset_SfM'
feature_dir = 'SIFT'
errors_df = runFeature.runSFM(dataset_path, feature_dir )
```

1. Report the final rotation and translation error. **(2 points)**

- Final Frame Mean Rotation and Mean Translation Error are 0.00243 and 0.55841 respectively.

In [9]:
```
"""Not to be reported"""
# print("Rotation and Transition Error metric for all frames \n")
# errors_df.index = np.arange(1, len(errors_df) + 1)
# errors_df.index.name = "Frame Number"
# with pd.option_context('display.max_rows', None, 'display.max_columns', Non
e):
#     print(errors_df)
```

Out[9]: 'Not to be reported'

Next, answer the following questions:

1. Does SIFT yield higher accuracy than the original `libviso2`? Why or why not? If not, can you suggest one possible way to improve? **(5 points)**

- SIFT yields approximately same error/accuracy as `libviso2`. `libviso2` uses corners and blob-like keypoints similar to SIFT. In the dataset there is no significant scale and illumination variation thus extracting SIFT keypoints do not give any advantage. Also the motion of car is in a plane thus no advantange of rotation invariance by SIFT keypoints are not advantageous here.
- Few Literatures[1] have suggested idea to divide the SIFT features extracted from object image into several sub-collections before they are matched. The features are divided into several sub-collections considering the features arising from different octaves, that is from different frequency domains. This has shown significant improvement in accuracy and speed of the feature matching.
  Some literatures also suggest that Modifiable Area Harmony Dominating Rectification (MHDR) method[2] is useful to eliminate mismatched key-point couples automatically and protect matching couples. MHDR method has promising results in improving the accuracy and efficiency of the SIFT image matching. Further, SIFT features matching algorithm can also be improved by changing distance metric from euclidean to other problem specific metrics.

  [1] = https://dl.acm.org/doi/10.5555/2227536.2227552 (https://dl.acm.org/doi/10.5555/2227536.2227552)
  [2] = https://link.springer.com/chapter/10.1007/978-3-642-28308-6_31 (https://link.springer.com/chapter/10.1007/978-3-642-28308-6_31)

1. Explain how SIFT achieves invariance to

   a. illumination
   b. rotation
   c. scale

   (3 points)

- The SIFT algorithm has following major stages of computation used to generate the set of image features: Scale-space extrema detection, Keypoint localization, Orientation assignment and Keypoint descriptor. Invariance to each illumination, rotation and scale in relation to these stages are described below:

    1. Illumination: At final step of computation, the descriptors obtained are normalized to unit length. This make the descriptors invariant to affine changes in illumination. Robustness to non-linear illumination change is obtained by thresholding the values in the unit feature to each be no larger than 0.2 and then renormalizing to unit.
    2. Rotation: At orientation assignment step, each keypoint is assigned one or more orientations based on local image gradient directions. This is the key step in achieving invariance to rotation as the keypoint descriptor can be represented relative to this orientation and therefore achieve invariance to image rotation.
    3. Scale: At first step of algorithm, Local extrema detection(keypoints) are searched over all image location in difference of gaussian(DOG) at different scales make the algorithm invariant to scale.

Question 3: Using SuperPoint[1] for SFM

Now you are required to use keypoints and feature descriptors from SuperPoint for SFM. The code for the trained model of this method can be found from the `SuperPoint`.

(A) Go to `SuperPoint` directory and run `demo_superpoint.py`. `python demo_superpoint.py --input /datasets/cse152-252-sp20-public/dataset_SfM/sequences/00/image_0/ --cuda` The detected keypoints and feature descriptors are under the directory `SuperPoint`. The file format is similar to the SIFT case. For image `000abc.png`, the pre-computed features and keypoints should be saved in a `.npy` file named as `000abc_feature.npy`. The variable is a $258 \times N$ matrix with `single` precision, where N is the number of feature points being detected. For each 258-dimensional feature vector, the first two dimensions are the locations of the keypoint (column number first and then row number) on the image plane and the last 256 dimensions represent the feature descriptor.

(B) Run the following script

In [ ]:
```
import runFeature
dataset_path = '/datasets/cse152-252-sp20-public/dataset_SfM'
feature_dir = 'SuperPoint'
errors_df=runFeature.runSFM(dataset_path, feature_dir )
```

1. Report the final rotation and translation error. **(2 points)**

- Final Frame Mean Rotation and Mean Translation Error are 0.00369 and 0.52405 respectively.

In [13]:
```
"""Not to be reported"""
# print("Rotation and Transition Error metric for all frames \n")
# errors_df.index = np.arange(1, len(errors_df) + 1)
# errors_df.index.name = "Frame Number"
# with pd.option_context('display.max_rows', None, 'display.max_columns', Non
e):
#     print(errors_df)
```

Out[13]:
```
'Not to be reported'
```

Next, answer the following questions:

1. Does SuperPoint yield higher accuracy than the original `libviso2`? If so, why? If not, what steps can you take to improve? **(5 points)**

- SuperPoint yields approximately same error/accuracy as `libviso2`. No improvement is obtained because MagicPoint detected trained on the pseudo-ground truths on synthetic images, do not generalize well to complex images in this dataset.

  Improvement: One clear method to improvement is to iterate multiple times to increase impact of self-training

1. Explain briefly how the following issues are being handled in SuperPoint: **(3 points)**

```
a. Obtaining ground truth for keypoints.
b. Cheaply obtaining accurate ground truth matches, as compared to LIDAR in UCN
or SFM in LIFT.
c. Learning a correlated feature representation for keypoint detection and desc
ription?
```

1. In SuperPoint, we create a large dataset of pseudo-ground truth interest point locations in real images, supervised by the interest point detector itself, rather than a large-scale human annotation effort.
2. MagicPoint detector network is trained on synthetic dataset. we develop a multi-scale, multi-transform technique and Homographic Adaptation to improve its performance on real-world example. These techniques are computationally less expensive and have proven to be effective in improving networks performance. Thus cheaply obtaining accurate ground truths as compared to LIDAR in UCN or SFM in LIFT.
3. The network uses the features from same shared encoder for both detection and description.

Question 4: Using SPyNet [5] for SFM

Now we will compute camera motion from optical flow computed using SPyNet. We first uniformly sample points in an image, then consider the flow-displaced point in the other image as a match. A modified PyTorch implementation of SPyNet is provided in directory `Flow`.

(A) Go to `Flow` and run `demo_spynet.py`.

(B) Run the following script.

In [ ]:
```
import runMatch
dataset_path = '/datasets/cse152-252-sp20-public/dataset_SfM'
feature_dir = 'Flow'
errors_df=runMatch.runSFM(dataset_path, feature_dir )
```

1. Report the final rotation and translation error. **(2 points)**

- Final Frame Mean Rotation and Mean Translation Error are 0.00372 and 0.79390 respectively.

In [4]:
```
'''Not to be reported'''
# print("Rotation and Transition Error metric for all frames \n")
# errors_df.index = np.arange(1, len(errors_df) + 1)
# errors_df.index.name = "Frame Number"
# with pd.option_context('display.max_rows', None, 'display.max_columns', Non
e):
#     print(errors_df)
```

Out[4]:
```
'Not to be reported'
```

Next, answer the following questions:

1. Does SPyNet yield higher accuracy than the original `libviso2`? Why or why not? If not, what steps can you take to improve? **(5 points)**

- No. SPyNet yields lower accuracy than libviso2. SPyNet uniformly samples points in the image, then consider the flow-displaced point in the other image as a match. By Uniform sampling it tries to match features which are not interest points. This leds to mismatching and thus reduces the accuracy.
  One of the ways to improve the algorithm is to sample interest points rather than uniform sampling.

1. Explain how SPyNet achieves accurate flow with significantly lower computational cost. **(3 points)**

- SPyNet scales the image down and creates image pyramid of different resolution level. The Network computes difference between true and upsampled flows at each level called residual field. The residual fields across the image at a certain level are usually a small displacements and their computation is relatively simple. Thus, each level solves a simple problem of finding these residual fields, requiring a small CNN network with far less trainable parameters than networks like FlowNet. Further rather than having a network for warping, SPyNet bilinearly interpolates image flow as consecutive levels.

Question 5: Using Sfm-learner for SFM

Now we will use deep neural networks for SFM. Please follow the open-source Sfmlearner repository (https://github.com/ClementPinard/SfmLearner-Pytorch (https://github.com/ClementPinard/SfmLearner-Pytorch)) and the paper (https://people.eecs.berkeley.edu/~tinghuiz/projects/SfMLearner/cvpr17_sfm_final.pdf (https://people.eecs.berkeley.edu/~tinghuiz/projects/SfMLearner/cvpr17_sfm_final.pdf)) to answer the following questions.

1. Follow the steps to add sfm-learner as a submodule: `git submodule add https://github.com/ClementPinard/SfmLearner-Pytorch.git` .
2. you might have to do `git submodule update --init --recursive` for older git version or on other systems which already have previous commits locally.
3. For cloning the repository use `git clone --recursive <project url>` to download submodule also

1. Describe how photometric loss is implemented in Sfmlearner? Please refer to the paper and the code. **(5 points)**

- The Photometric loss is based on warping nearby views to the target using the computed depth and pose. The depth and explainability mask predictions from single-view depth network and pose/explainability network are used to compute the photometric loss. The source image is warped and L1 loss is computed with respected to target image. The loss is computed at different scale(factor of 2) of image and is accumulated over each scales.
  Mathematically loss can be written as:
  Photometric Loss $(L_1) = \sum_{<I_1,...,I_N>\in S} \sum_{p} \hat{E}_s(p) \mid I_t(p) - \hat{I}_s(p)|$
  where, $\hat{E}_s(p)$ is output(prediction) of explainability prediction network and is per-pixel(p) soft mask for each target-source pair.

  Further, to encourage non-trivial(non-zero) prediction of $\hat{E}_s$, the author introduce a regularization loss called explainability loss.

1. Train the model and show the training curve, validation curve, and visualization of predicted depth and warped images? Which loss is decreasing and which is increasing? Why? **(10 points)**

   Visualize the specific images: `val_Depth_Output_Normalized/0` , `val_Warped_Outputs/0` from tensorboard.

   Training data: `/datasets/cse152-252-sp20-public/sfmlearner_h128w416` .

   #### Install the environment `pip install -r requirements.txt`

   - fix scipy version problem:

   ```
   pip install scipy==1.1.0 --user
   ```

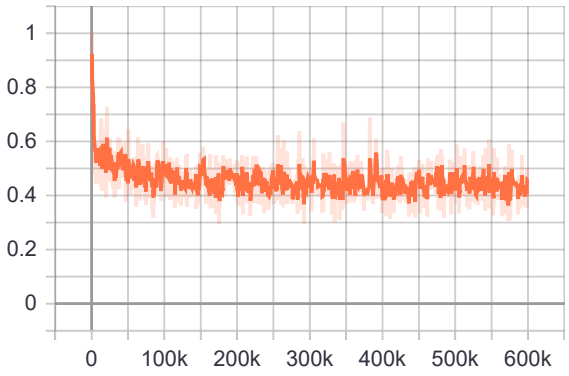
   #### Training script `cd SfmLearner-Pytorch`

   ```
   python3 train.py /datasets/cse152-252-sp20-public/sfmlearner_h128w416 -b4 -m0.2 -s0.1
   --epoch-size 3000 --sequence-length 3 --log-output
   ```

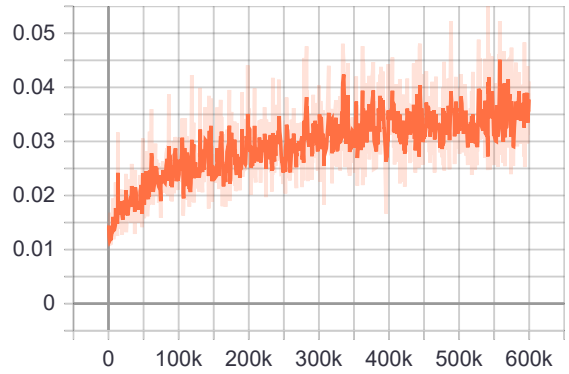
All the plots are with smoothness factor of 0.4

Loss vs epochs Plots for Training Dataset:

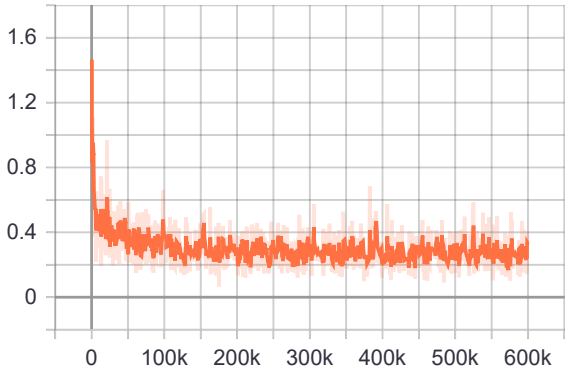
Total Loss



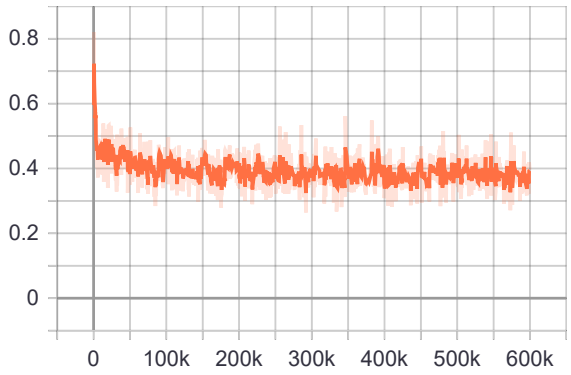
Disparity Smoothness Loss



Explanability Loss

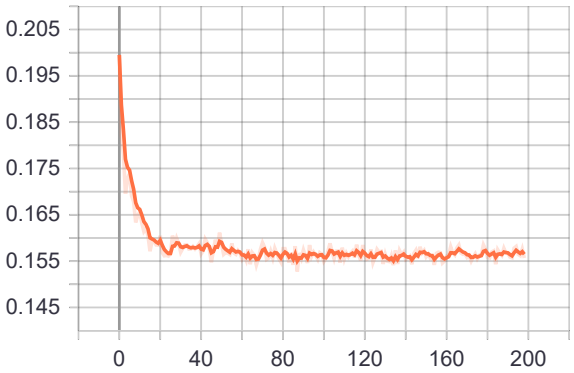


Photometric Error

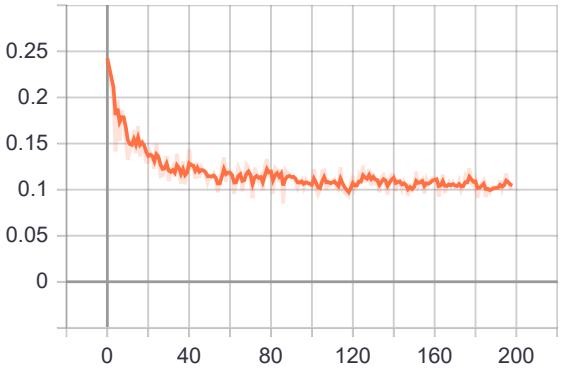


Loss vs epochs Plots for Validation Dataset:

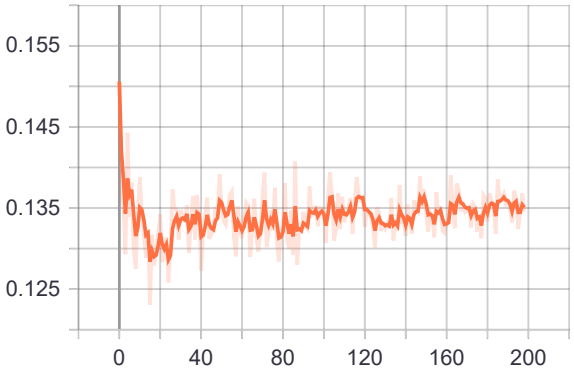
Total Validation Loss



Explanability Loss

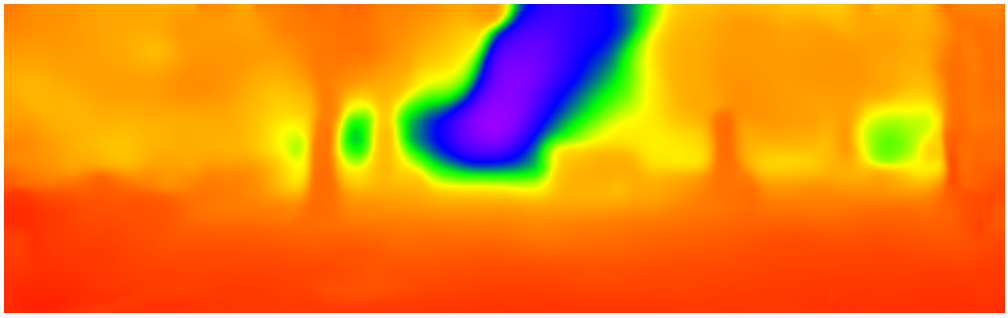


Photometric Error



Visualization of Results

val_Depth_Output_Normalized/0



val_Warped_Outputs/0

1. When training the model, we use 3 consecutive frames. Now, you will use the photometric consistency between the 1st and the 3rd frame. To be more specific, you can get the pose $T_{1,3} = T_{2,3} @ T_{1,2}$, where $T_{1,2}, T_{2,3}$ have already been computed in the original code. Add the constraint to the total loss and report the results in the same manner as in part 2 above. **(10 points)**

   #### Training script `cd SfmLearner-Pytorch`

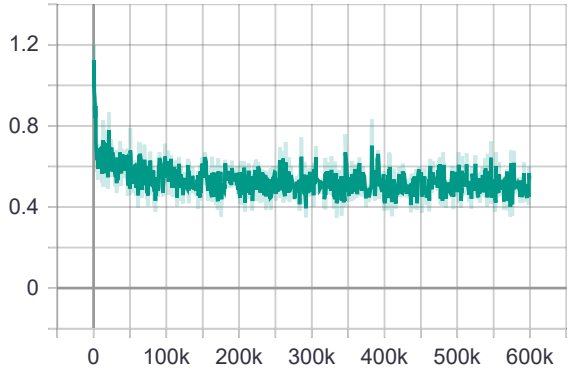
   ```
   python3 train.py /datasets/cse152-252-sp20-public/sfmlearner_h128w416 -b4 -m0.2 -s0.1
   --epoch-size 3000 --sequence-length 3 --log-output --with-photocon-loss
   ```

answer here

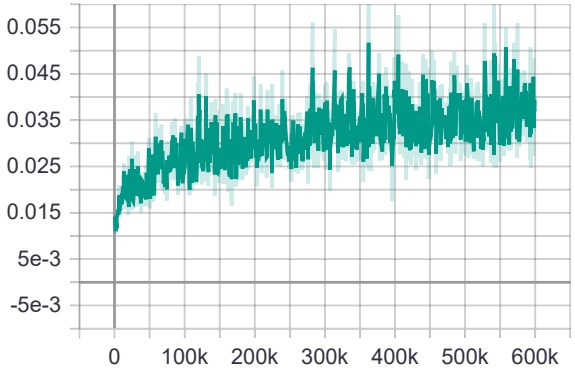
All the plots are with smoothness factor of 0.4

Loss vs epochs Plots for Training Dataset:

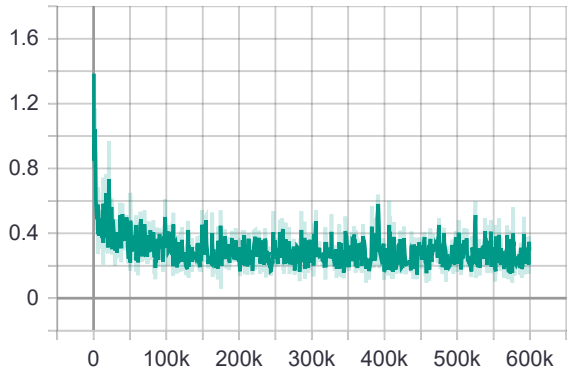
Total Loss



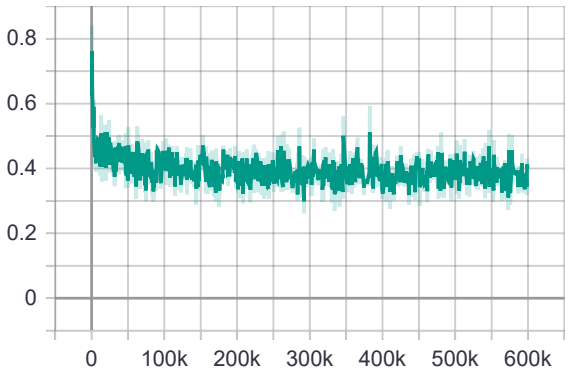
Disparity Smoothness Loss



Explanability Loss

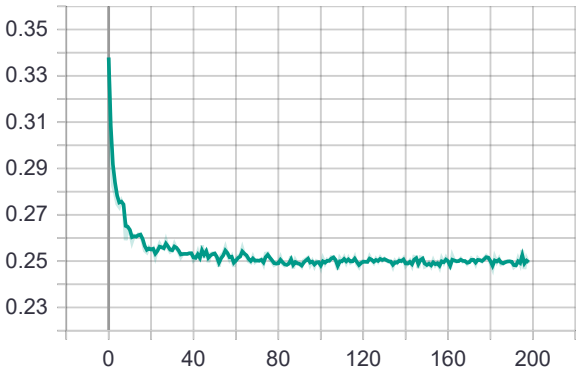


Photometric Error

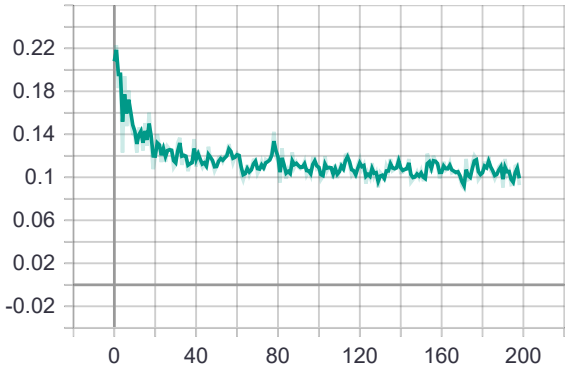


Loss vs epochs Plots for Validation Dataset:

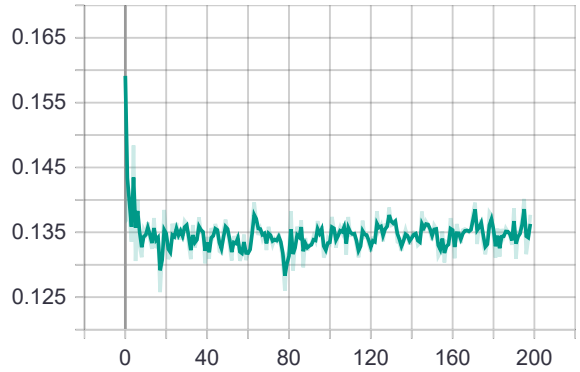
Total Validation Loss



Explanability Loss

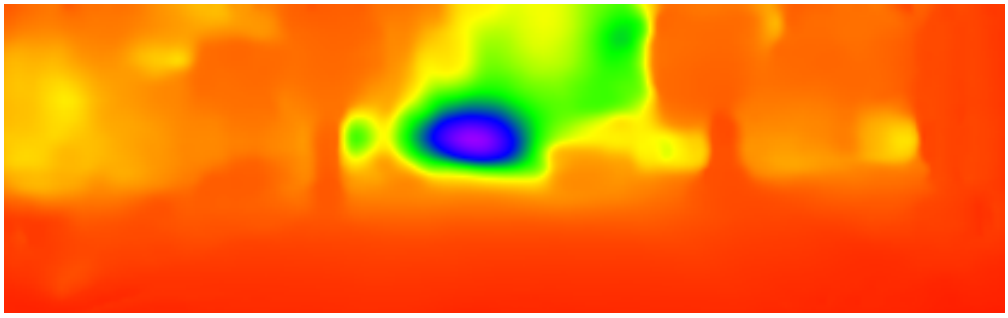


Photometric Error



Visualization of Results

val_Depth_Output_Normalized/0



val_Warped_Outputs/0

1. Now, you will evaluate your models from parts (2), (3) on the KITTI odometry dataset. What are the error metrics used in Sfm-learner? Please report the `ATE` and `RE` for sequence `09` and `10` . **(5 points)**

   Data: `/datasets/cse152-252-sp20-public/kitti` .

   #### Evaluation script `python3 test_pose.py /path/to/posenet --dataset-dir /datasets/cse152-252-sp20-public/kitti --sequences 09`

Error metrics used: Absolute Translation Error (ATE) and Rotational Error (RE)

Simple SfmLearner Model(3 losses) after training for 200 epochs

- Performance on sequence 09:

	ATE	RE
mean	0.0119	0.0023
std	0.0065	0.0012

- Performance on sequence 10:

	ATE	RE
mean	0.0108	0.0024
std	0.0084	0.0016

SfmLearner Model with Photoconsistency Loss(4 losses) after training for 200 epochs

- Performance on sequence 09:

	ATE	RE
mean	0.0128	0.0024
std	0.0074	0.0013

- Performance on sequence 10:

	ATE	RE
mean	0.0098	0.0025
std	0.0071	0.0017

Notes:

- scp data to local machines:

```
scp -r <USERNAME>@dsmlp-login.ucsd.edu:/datasets/cse152-252-sp20-public/sfmlearner_h128w416.zip
```

```
scp -r <USERNAME>@dsmlp-login.ucsd.edu:/datasets/cse152-252-sp20-public/kitti.zip ...
```

- tensorboard: open jupyter notebook from the link after `launch-scipy-ml-gpu.sh` . Click new `Tensorboard ..`

References

1. Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pages 224–236, 2018.
2. Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In Conference on Computer Vision and Pattern Recognition (CVPR), 2012.
3. Andreas Geiger, Julius Ziegler, and Christoph Stiller. Stereoscan: Dense 3d reconstruction in real-time. In Intelligent Vehicles Symposium (IV), 2011.
4. David G Lowe. Distinctive image features from scale-invariant keypoints. IJCV, 60(2):91–110, 2004.
5. Anurag Ranjan and Michael J Black. Optical flow estimation using a spatial pyramid network. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4161–4170, 2017.
6. A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. http://www.vlfeat.org/ (http://www.vlfeat.org/), 2008.
7. Lucas, Bruce D., and Takeo Kanade. "An iterative image registration technique with an application to stereo vision." (1981): 674.
8. B. Kitt, A. Geiger, and H. Lategahn, “Visual odometry based on stereo image sequences with RANSAC-based outlier rejection scheme,” in 2010 IEEE Intelligent Vehicles Symposium, La Jolla, CA, USA, Jun. 2010, pp. 486–492, doi: 10.1109/IVS.2010.5548123.
9. A. Geiger, J. Ziegler, and C. Stiller, “StereoScan: Dense 3d reconstruction in real-time,” in 2011 IEEE Intelligent Vehicles Symposium (IV), Baden-Baden, Germany, Jun. 2011, pp. 963–968, doi: 10.1109/IVS.2011.5940405.