CMSC733: Final Project - Structure From Motion

Chayan Kumar Patodi
UID : 116327428
Email: ckp1804@terpmail.umd.edu

Saket Seshadri Gudimetla Hanumath
UID : 116332293
Email: saketsgh@terpmail.umd.edu

***Abstract*—The purpose of this project is to solve the problem of Structure from Motion i.e. reconstruct a 3D scene and simultaneously obtain the camera poses of a monocular camera with respect to the given scene. As the name suggests, an entire rigid structure is created from a set of images with different view points. The problem is solved using traditional computer vision and deep learning methods.**

I. Phase 1 : Traditional Approach

In this section we present the flow of how the traditional approach works. Following are the steps involved in the estimation of Structure from Motion -

1) Feature Matching and Outlier rejection using RANSAC
2) Estimating Fundamental Matrix
3) Estimating Essential Matrix from Fundamental Matrix
4) Estimate Camera Pose from Essential Matrix
5) Check for Cheirality Condition using Triangulation
6) Non-Linear Triangulation for refining 3D points
7) Perspective-n-Point for registering new images
8) Bundle Adjustment to refine obtained pose and 3-D points

A. Feature Matching and Outlier rejection

The initial step in the pipeline of SFM is to find 2-D image correspondences across various views. Normally one would find features using an interest point detector like SIFT, but in our case we were provided with text files containing feature correspondences between images. Six pictures depicting six different views of a building were provided. Our task is to find out the pose of the camera in each of the views (assuming the first view as the origin of the world coordinate system) and reconstruct the 3-D scene shown in the images. All the features are not useful for performing the aforementioned tasks. Hence, filtering out such outliers will be crucial for obtaining an accurate output.

For outlier rejection we use a Fundamental Matrix based Random Sample Consensus (RANSAC) method. The maximum number of correspondences that satisfy the Fundamental matrix equation (1) are stored as inliers along with the Fundamental matrix computed in that iteration.

$$x^{'T} F x = 0 \tag{1}$$

Using this method we estimate the maximum number of inliers for the pair of images - $(1,2),(2,3),(3,4),(4,5),(5,6)$. The output of RANSAC that shows the inliers and outliers can be seen in the figure (1).

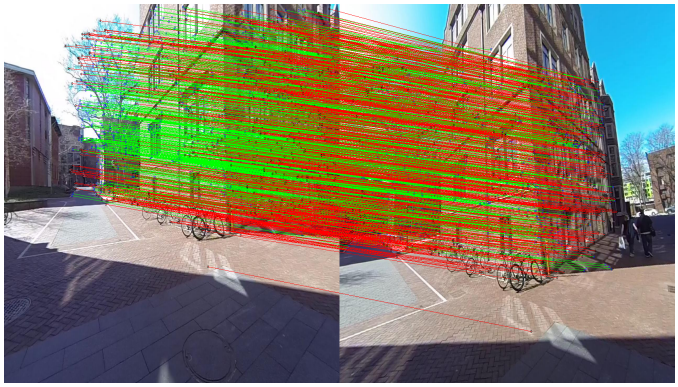


Fig. 1. plot of inliers(green) and outliers(red) after RANSAC on image pairs 1 and 2

B. Estimating Fundamental Matrix

The Fundametal Matrix relates the point correspondences in two images using the epipolar constraint equation (1) where $x^{'}$ is the pixel coordinate in the right image and x is the same for the left image in a stereo setup. The Fundamental matrix helps us identify the epipole which is the intersection of all the epipolar lines in the image. The line joining the two epipoles in the two stereo images gives us the relative translation between the two cameras (which we obtain using the Essential Matrix). The Fundamental Matrix can be estimated using the 8-point algorithm described as follows.

The equation (1) can be expanded in the following way -

$$\begin{bmatrix} x_i' & y_i' & 1 \end{bmatrix} \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = 0$$

From this we get the following equation -

$$x_i x_i' f_{11} + x_i y_i' f_{21} + x_i f_{31} + y_i x_i' f_{12} + y_i y_i' f_{22} + y_i f_{32} + x_i' f_{13} + y_i' f_{23} + f_{33} = 0$$

In F matrix estimation, each point only contributes one constraint as the epipolar constraint is a scalar equation. Thus, we require at least 8 points to solve the above homogenous system. Thus, this is called the 8-point algorithm. Solving this requires stacking the equation for 8 points into the following form -

$$\begin{bmatrix} x_1x'_1 & x_1y'_1 & x_1 & y_1x'_1 & y_1y'_1 & y_1 & x'_1 & y'_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_mx'_m & x_my'_m & x_m & y_mx'_m & y_my'_m & y_m & x'_m & y'_m & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{21} \\ f_{31} \\ f_{12} \\ f_{22} \\ f_{32} \\ f_{13} \\ f_{23} \\ f_{33} \end{bmatrix} = 0$$

This system of equation can be answered by solving the linear least squares using Singular Value Decomposition (SVD). The solution is obtained using the last column of V in USV^T decomposition of the above equation. Rank-2 constraint is then applied in order to compensate for any noise in the correspondences that can accidentally make the F matrix a full rank (rank-3) matrix which we do not want. The F matrix estimated for the first image pair $(1, 2)$ is as follows -

$$F = \begin{bmatrix} -3.2749e-07 & -1.0256e-05 & 2.6871e-03 \\ 1.2735e-05 & -7.4194e-07 & -4.1189e-03 \\ -4.3912e-03 & 1.9287e-03 & 9.9997e-01 \end{bmatrix}$$

C. Estimating Essential Matrix from Fundamental Matrix

The Essential Matrix gives us the relative pose of the camera. It is a $3X3$ matrix like F with some additional properties. The Essential Matrix is obtained from the Fundamental Matrix using the following equation -

$$E = K^T F K$$

where K is the intrinsic camera matrix and F is the Fundamental Matrix computed from previous step. Like before we need to enforce the rank-2 constraint to offset the effect caused by noise in the correspondences. This is done by taking SVD of the initial E estimate and then the sigma matrix is corrected as follows -

$$\mathbf{E} = U \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T$$

The Essential Matrix computed using the above steps is as follows -

$$E = \begin{bmatrix} -0.0184 & -0.7134 & -0.2942 \\ 0.8759 & -0.0552 & 0.4477 \\ 0.1830 & -0.6149 & -0.1469 \end{bmatrix}$$

D. Estimate Camera Pose from Essential Matrix

The camera-1 (image-1) is considered to coincide with origin of the world coordinate system and all the remaining camera poses (for each image) are estimated with respect to the first camera (or image).

The camera pose consists of 6 degrees-of-freedom (DOF) Rotation (Roll, Pitch, Yaw) and Translation (X, Y, Z) of the camera with respect to the world. Four camera poses for camera-2 are obtained which we then disambiguate in the next step using cheirality checks. The four initial pose estimates are obtained using the following set of equations -

Let

$$E = UDV^T$$

and

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Then the four poses are-

$$C_1 = U(:, 3), R1 = UWV^T$$
$$C_2 = -U(:, 3), R2 = UWV^T$$
$$C_3 = U(:, 3), R3 = UW^TV^T$$
$$C_4 = -U(:, 3), R4 = UW^TV^T$$

The camera pose is further corrected by checking determinant of the rotation matrix R, if it is less than zero then the pose is made negative of its original value before sending out.

E. Check for Cheirality Condition using Triangulation

Linear Triangulation is performed using the 2D-2D correspondences of image-1 and image-2. The following equation is used for Triangulation -

$$A = \begin{bmatrix} xm_3^T - m_1^T \\ ym_3^T - m_2^T \\ x'm_3^{'T} - m_1^{'T} \\ y'm_3^{'T} - m_2^{'T} \end{bmatrix}$$
$$AX = 0$$

where m and m' are the rows of the projection matrix of the two images. The above equation is solved using linear least squares method (as discussed previously). The Cheirality condition helps us disambiguate the camera pose to obtain the best pose for the given camera. This is done by checking the simple inequality -

$$(r3, X - C) > 0$$

where $r3$ is the third row of the rotation matrix and C is one of the four estimated camera centers. The camera pose that has the most 3-D points in front of it will be considered the optimal pose. In this way we determine the optimal pose. The plot of linear triangulation and pose disambiguity can be seen in figures 2 and 3.

F. Non-Linear Triangulation

The non-linear triangulation is performed in order to optimise the 3-D points obtained from previous step by minimising the re-projection error defined by the following equation -

$$\sum_{j=1,2} \left(u^j - \frac{P_1^{jT}\widetilde{X}}{P_3^{jT}X} \right)^2 + \left(v^j - \frac{P_2^{jT}\widetilde{X}}{P_3^{jT}X} \right)^2$$

We use the function $scipy.optimize.least\_squares$ to minimize the re-projection error. The output of linear v/s non-linear triangulation can be seen in the figure 4

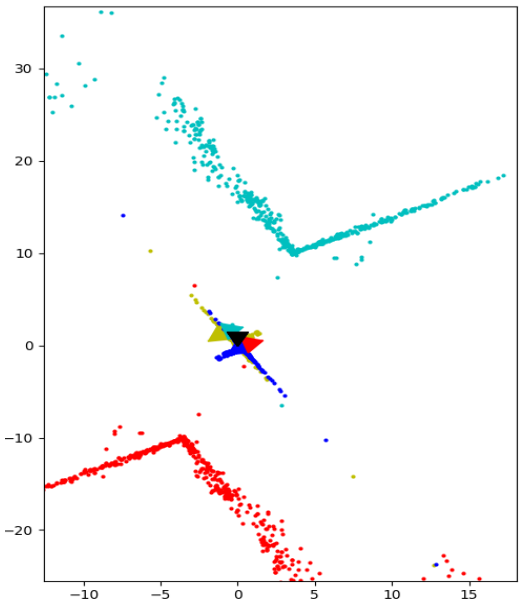


Fig. 2. linear triangulation for image-1,2

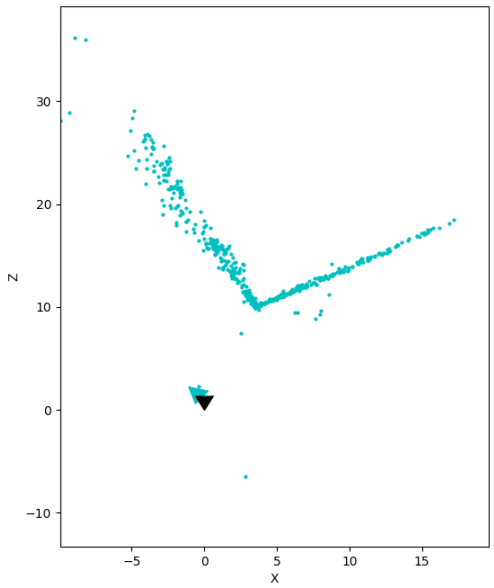


Fig. 3. output of cheirality check

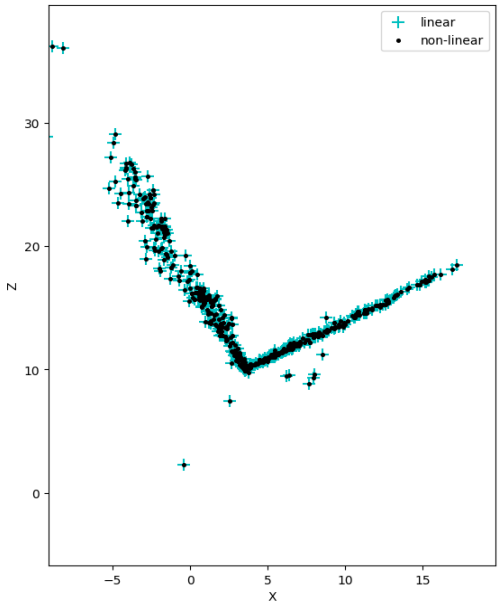


Fig. 4. linear v/s non-linear triangulation

G. *Perspective-n-Points*

1) Linear-PnP: We use the 2D-3D correspondences obtained from image-1 and image-2 in order to register the remaining 4 images. At present we do not have pose estimates of the remaining images but we can obtain the 2D-3D correspondences for each image using previously obtained images by using the provided text files and comparing them with correspondences obtained in the previous steps. We select the following pair of images to work with - $(2,3),(3,4),(4,5),(5,6)$. These pair of images have maximum correspondences between them which are then used to establish the 2D-3D correspondences for the newer images.

The PnP problem consists of using these established correspondences to obtain an initial pose estimate of each image(3-6). Given 2D-3D correspondences, and the intrinsic paramter K, we estimate the camera pose using linear least squares. 2D points are normalized by the intrinsic parameter to isolate camera parameters, (C,R), i.e. $K^{-1}x$. A linear least squares system that relates the 3D and 2D points can be solved for (t,R) where $t = -R^T C$.

We then use RANSAC to minimize the re-projection error. The initial estimates are then refined further using non-linear PnP.

2) Non-Linear-PnP: The objective in this stage is to optimise the pose estimates for given 2D and 3D points and minimize reprojection error given by the following equation -

$$\min_{C,q} \sum_{i=1,J} \left(u^j - \frac{P_1^{jT}\widetilde{X_j}}{P_3^{jT}\widetilde{X_j}} \right)^2 + \left(v^j - \frac{P_2^{jT}\widetilde{X_j}}{P_3^{jT}X_j} \right)^2$$

We convert Rotation matrix to Quaternion form in order to enforce orthogonality of the rotation matrix.

When we implemented the PnP it was for a subset of all the image points as we did not have 2D-3D correspondence for each image point. Now, since we have efficient pose estimates for the given images we can use Linear and Non-linear Triangulation as before to obtain the best estimates of 3D points. The plot showing the refined camera poses along with their refined 3D correspondences can be seen in figure 5.

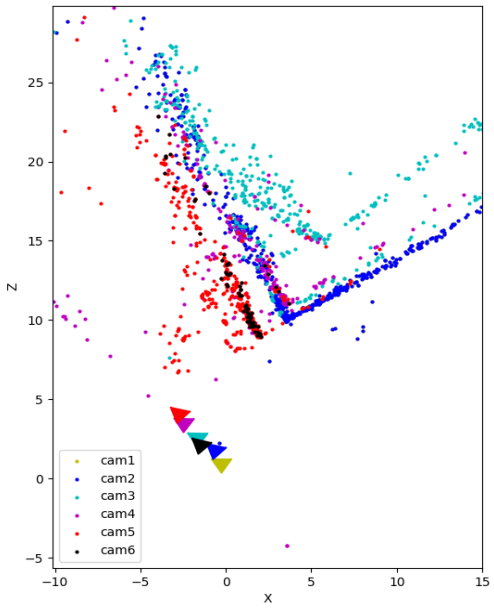


Fig. 5. plot of all camera poses and their respective 3D correspondences

H. Bundle Adjustment

Bundle Adjustment is done in order to simultaneously optimise the pose as well as the 3D world points whilst minimizing the error given by the following equation -

$$\min_{\{C_i,q_i\}_{i=1}^{I},\{X\}_{j=1}^{J}} \sum_{i=1}^{I}\sum_{j=1}^{J} V_{ij}\left(\left(u^j - \frac{P_1^{jT}\tilde{X}}{P_3^{jT}\tilde{X}}\right)^2 + \left(v^j - \frac{P_2^{jT}\tilde{X}}{P_3^{jT}\tilde{X}}\right)^2\right)$$

For implementing this we used the approach given in [5].
The parameters to be optimized (7 in total for each point including 4 from camera pose and 3 from the 3D points) are given as a single flattened 1D array. A 2D-3D map consisting of 2D images stored as tuples with the corresponding 3D location indices. A sparse visibility matrix is computed using the method given in [5]. The only difference in our approach is that we transform the rotation matrix into quaternion form hence our total parameters for pose are 7 instead of 9 (as given in their approach). In each iteration bundle adjustment is performed to optimise all the camera poses and 3D points which are then used in the next iteration for registering new camera.
The output of bundle adjustment can be seen in the figure 6
The mean projection errors are summarised in the table I-H

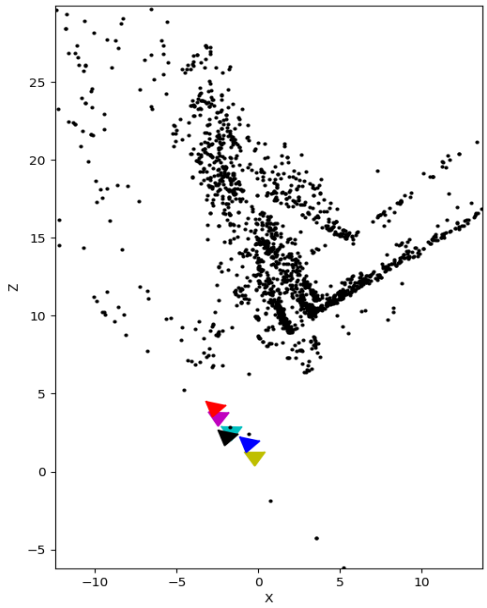


Fig. 6. bundle adjustment output for all cameras/images

Camera pair	LinTri	Non-LinTri	LinPnP	Non-LinPnP	B.A
1-2	11.562	10.362	-	-	-
2-3	140.345	138.273	1210.386	621.504	6.46
3-4	28.697	27.755	3917.499	163.109	5.52
4-5	10.241	9.894	11530.67	89.6922	7.201
5-6	5585.278	5485.465	50835.45	29336.059	182.99

I. Conclusion and Final Thoughts

It was difficult to get enough point correspondences from 2D-3D for each image (especially 6). Due to this the pose estimates in such images generated in the PnP step do not generalise very well for all the points. Few outliers cause the mean projection error to shoot up. This problem can be solved by fine tuning the RANSAC which takes a lot of trial-error attempts. We were able to achieve decent reprojections for all the images except the 6th one. The re-projected images-3,4,5 can be seen in figures 7, 8 and 9.

Fig. 7. reprojection for image 3

Fig. 8. reprojection for image 4

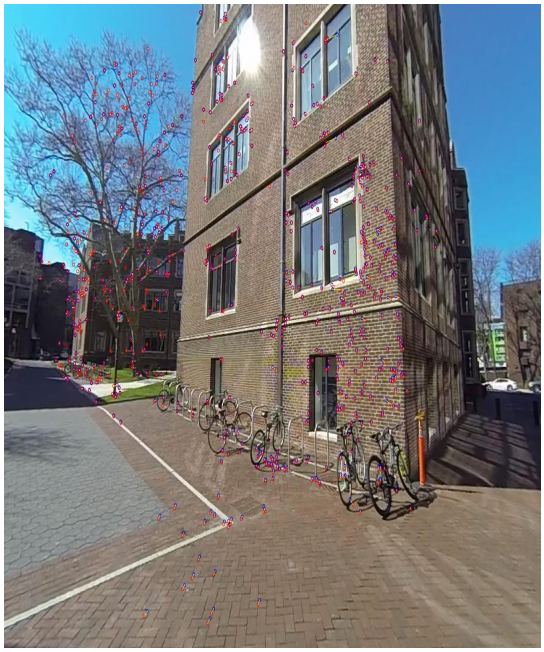


Fig. 9. reprojection for image 5

II. Phase 2 : Unsupervised Deep Learning

A. *Introduction*

In this project we explore an unsupervised deep learning approach to retrieve depth and Ego-Motion from motion. We used [1] to understand the whole pipeline and then modified their code to improve the results.

B. *SFM Learner*

The approach that [1] uses is jointly training a single view depth CNN and a camera pose estimation CNN from unlabeled video sequences. The unique thing about this is,even though the models are trained in a joint manner, they can be used independently during test-time.The architecture that they have used is show in the figure 10.

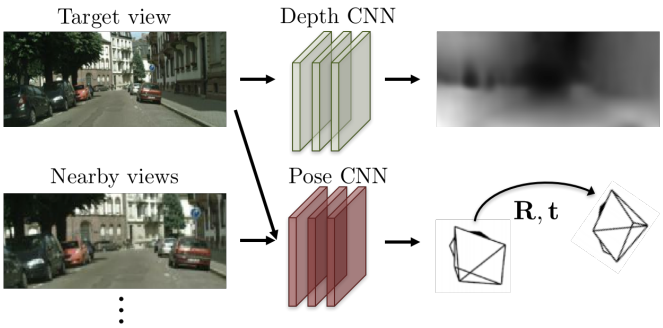
C. *Preparation*

The dataset we were given for training and validation was a subset of KITTI Dataset, with only 12k training images, which were vertically stacked in a sequence. The size of the dataset really affects the model and the results. We will show the difference in their original results and their model trained on our dataset in the next section.

1) *Training:*

We had to make some changes to the train.py and SFMLearner.py files, in order to train our model. The inconsistencies were in the way how the directory structure was formatted and how the filenames were stored in the textfiles.

(a) Training: unlabeled video clips.



(b) Testing: single-view depth and multi-view pose estimation.

Fig. 10. Original Architecture of the paper

• First we train their model, with our dataset. We keep everything same, 200k iterations, 64 epochs, and batch size to be 4. It took us 20 hours to do so, on GTX 1050Ti.
• We also train multiple models with our dataset.

*2) **Testing**:*

For testing, we prepare the testing data for the provided testing scripts.Since we were provided with Raw dataset **(2011_09_26_drive_0015_sync)**, we had to do two steps :
• I created Data_to_text.py script ,to read the test files name and writes them to a text file testfiles.txt. This testfiles.txt would be called in the testing scripts.
• For pose estimation, the directory structure is completely different. So we created a folder and modified the raw_data accordingly, to read the files in a sequence.

*3) **Evaluation**:*

For depth estimation, we need the predictions generated by the test script and then use that to run the evaluation script. We get the errors and accuracy, which we will discuss in the results section. We don't need to change a lot in the depth evaluation script. This was the most challenging part of the project. Gathering all the resources for pose evaluation was a tricky task. We had OXTS data from IMU. We used the function from pykitti library [2]. Then function reads OXTS data and uses that to estimate the pose. We use **Mercator Projection** for estimating the pose. We convert the Rotation into the quaternion form. Since we selected the sequence length to be 3, we select the first three frames, calculate the pose for those and then make first frame of that sequence to be the origin. We do this to obtain the ground truth for all the frames, which will be used for evaluation purposes.

We create two scripts to achieve the above:

• **OXTS_to_text.py** : We read the OXTS data file path and names and save them to a text file , oxtsfiles.txt

• **ExtractGT.py** : This script is important for pose evaluation. This script contains function to read the timestamps , read the OXTS data, and convert that data into the desired form, and then save it to the folder Ground_Truth. We have ground truth in a sequence of three, for all the frames for the downloaded dataset.

Note: Since, we have generated the ground truth on our own, we might have missed some calibration factor, which results in high translation values in the x and y directions. This messes with the error and accuracy for all the models.

D. Updates Made to improve the results:

• **Attempt 1:**

The following changes were made, taking inspiration from Geonet [3].
(1) **Data Augmentation** : So we implemented three different things in this one function.
Gamma Shift : This was implemented in such a way that every channel in the frame, was multipled by a random value between 0.8 and 1.2
Brightness Shift : This was done in similar fashion. The brightness part of the image was multiplied by a random value between 0.5 and 2.0
Color Shift : The color channel shift was different, and this was done for the sole purpose of training the new loss components.
(2) **Loss Function** : The loss function in the original paper, includes three losses, a pixel loss, a smoothness loss and an explainability loss. For our first approach, we use all of those three losses.We also introduced a Structural Similarity Loss , as mentioned in [3].Photometric loss is not invariant to brightness, luminosity and illumination. So it works with no error, when there is constant brightness and illumination. SSIM is used, as structure is independent of ilumination. SSIM provides a metric for measuring perpetual differences.

$$L_{ssim} = \sum_{s} \frac{1 - SSIM(I_t, I_s)}{2}$$

(3) **Adaptive Learning Rate** : SFMLearner paper used a constant learning rate of 0.0002.We tried experimenting with the learning rate, by reducing it every 14000 iterations.Even though the total loss improved, smoothness loss overshoots in such scenario.
(4) **Modified Architecture** : We are still using dispNet for depth estimation as the original paper, but for Pose Estimation, we made some changes taking inspiration from the paper [4]. This had an negative impact on the model, increased the overall loss and threw many errors.

• **Attempt 2:**

(1) **Data Augmentation** : As stated above the data augmentation was done in a similar manner. We also, included random gaussian noise in the images, with a mean of 0 and standard deviation 0.01. We thought this would make our model more robust to noise and train better.
(2) **Loss Function** : As stated above, original SFMLearner uses three loss. In our first attempt, we used all those three, alongside SSIM. When we looked at the tensorboard plots, we realised that the explainability loss wasn't doing much towards the performance. Even [1] disabled it, in later updates to the code. In this approach, we exclude the explainability loss, by making the weight to this zero.
(3) **Learning Rate** : As stated above, adaptive learning rate resulted in an overshoot in the smoothness loss and thus we went with the author's original learning rate of 0.002. (4) **Architecture** : We stick with the original architecture for this approach.

E. Results

We conducted wide range of experiments, which included tweaking the hyper-parameters,augmenting the data and introducing the loss functions. We present the the results for each model and dataset, in the following tables and figures.The plot functions are from tensorboard summary.

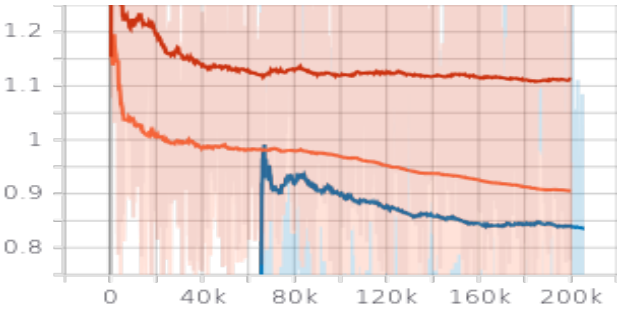


Fig. 11. **Pixel Loss**. Orange is their model trained on our dataset, blue is our first approach and Red is our model trained on our dataset.

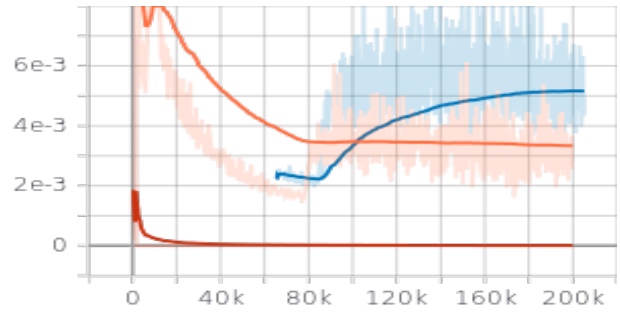


Fig. 12. Smoothness Loss. Orange is their model trained on our dataset, blue is our first approach and Red is our model trained on our dataset.

The comparison values for depth estimation can be found in the table I.The comparison values for pose estimation can be found in the table II.The loss value and comparison (Pixel loss , Smoothness Loss and the total training loss) can be seen in figure 11 , 12 and 14 respectively.

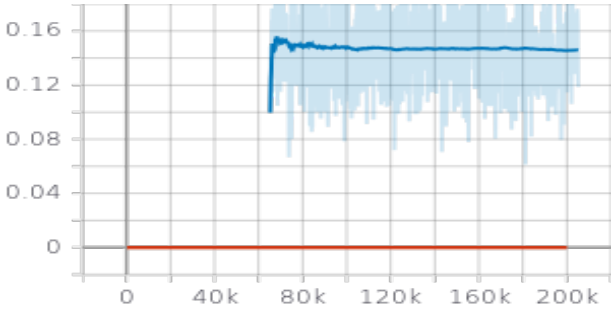


Fig. 13. Explainablity Loss. Orange is their model trained on our dataset, blue is our first approach and Red is our model trained on our dataset.

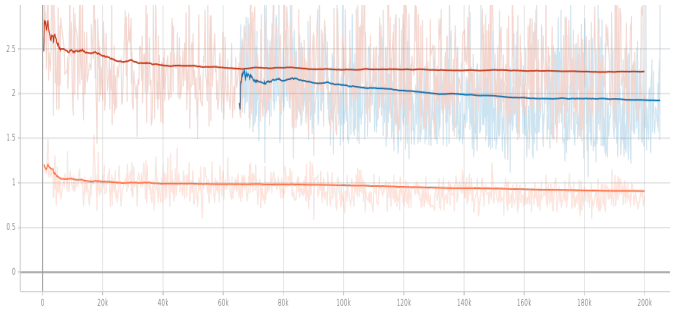


Fig. 14. Training Loss. Orange is their model trained on our dataset, blue is our first approach and Red is our model trained on our dataset.

Fig. 15. Original Image

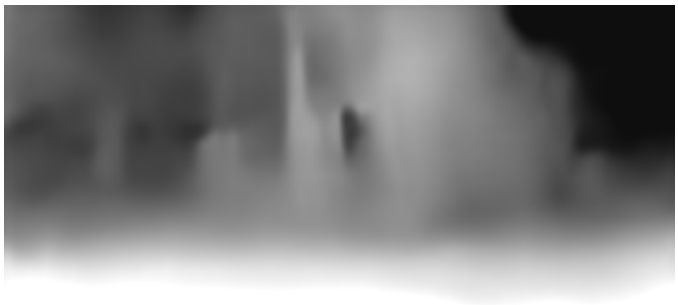


Fig. 16. Depth Image from original model

Model	abs_rel	sq_rel	rms	log_rms	d1_all	a1	a2	a3
SFM Learner Model (cited)	0.221	2.226	7.527	0.294	0.00	0.676	0.885	0.954
SFM Learner (Downloaded)	0.2645	5.3273	10.5205	0.3532	0.00	0.6813	0.8590	0.9283
SFM Learner on our Dataset	0.5250	21.7272	15.4223	0.5162	0.00	0.6100	0.8104	0.8774
1st Attempt (explainability)	0.6244	31.095	16.907	0.5541	0.00	0.2446	0.4752	0.6996
2nd Attempt(w/o exp)	0.4874	19.8391	14.6565	0.4959	0.00	0.6493	0.8328	0.8857

TABLE I
RESULTS GENERATED ON VARIOUS MODELS FOR DEPTH ESTIMATION

Model	ATE Mean	ATE Std.
SFM Learner (Downloaded)	0.5257	0.1577
SFM Learner on our Dataset	0.5257	0.1577
2nd Attempt(w/o exp)	0.4806	0.1608

TABLE II
RESULTS GENERATED ON VARIOUS MODELS FOR POSE ESTIMATION

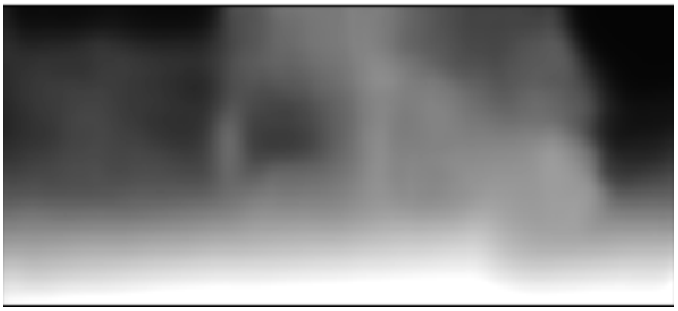


Fig. 17. Depth Image from our model

F. Analysis Of Outputs and Problems faced

Looking at the tables and plots above, we can see that our upgraded model does perform better to some extent, compared to their model on our dataset. We were able to decrease the loss further more and increase the accuracy to some extent. Training on more images could have given us better results. We could also do something about the regularization. Data augmentation, helped the model learn distinguishing image features in low-contrast images. Adaptive learning rate caused in the overshoot of smoothness loss, as visible in figure 12. Explainibility mask was of no good to the performance, either as visible in figure 13.The total loss is seen in figure 14 and could have been improved by increasing the iterations to some extent.
The depth image output for an image 15 from the given data for phase1 from both the models can be found in 16 and 17. Talking about the pose estimation, since we are not using the odometry set and working on the raw dataset, we can see the results are very different from what was cited in the paper. We compare the results for pose estimation in II. As visible, their initial downloaded model and their model trained on our dataset, performs similarly. Augmentation does help with the pose estimation, and improves the trajectory error, but the reason for such high values would be our generated ground truth.
The **problems** faced in this was preparing the raw dataset and generating the ground truth for evaluation purposes. That was resolved by writing multiple scripts, according to our needs to get the desired results. In our first approach, our training stopped at 66k iterations. We did continue it again from that checkpoint, but it ended up messing the tensorboard plots as visible.

G. Future Work

1. We can do multi-view inputs for depth estimation.
2. We can use ResNet50 as architecture for depth estimation.
3.For data augmentation, we can try more augmetations, such as rotation and flips.

REFERENCES

[1] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1851–1858, 2017.
[2] https://github.com/utiasSTARS/pykitti/tree/master/pykitti
[3] GeoNet: Unsupervised Learning of Dense Depth, Optical Flow and Camera Pose, Zhichao Yin and Jianping Shi
[4] Mayer, Nikolaus Ilg, Eddy Hausser, Philip Fischer, Philipp Cremers, Daniel Dosovitskiy, Alexey Brox, Thomas. (2016). A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation. 4040-4048. 10.1109/CVPR.2016.438.
[5] https://scipy-cookbook.readthedocs.io/items/bundle_adjustment.html