Reimplementing BA-Net: Dense Bundle Adjustment Networks

Adrian Brandemuehl
adrianbr@student.ethz.ch

Lukas Hoyer
lhoyer@student.ethz.ch

Tommaso Macrì
macrit@student.ethz.ch

Jose Vazquez
vjose@student.ethz.ch

Abstract

This project focuses on developing a neural network architecture, which has embedded a non-linear least squares (NL-LS) optimization problem in its core. This means that gradients are being back-propagated from the output of this NL-LS problem to its input. The architecture is based on the work from Tang, et al [19] and it is extended using ideas from Lv et al. [16]. The main focus of this work is to provide a working, trainable implementation of a differentiable bundle adjustment layer. Extensions to [19] are suggested to improve pose estimation accuracy via modifications in the CNN based feature network, the damping factor estimation layer, and an additional subnetwork for the camera pose initialization.

1. Introduction

Computer Vision is increasingly gaining attention and importance both in the research community and in industry, both for providing sensing and localization functionalities to robotic systems. In particular, Structure from Motion (SfM) is the problem of estimating the 3D structure of the environment, given a 2D image sequence. Structure from Motion can be tackled by employing conventional methods that jointly optimize the environment structure, and the camera motion, using Bundle-Adjustment (BA) algorithms. Bundle Adjustment ([21], [1]) is a method to find 3D point positions and camera motion by minimizing the reprojection error. The optimization is a non-linear least squares problem that can be tackled by the use of the Levenberg-Marquardt (LM) algorithm [17].

Lately, new efforts have been made in the research community to tackle the problem of estimating the depth and the camera motion from images using Convolution Neural Networks (CNNs) [7, 6, 15, 22]. However, in contrast to BA, these approaches do not enforce geometric constraints on the structure of the scene and the camera motion.

Recent works combine depth and camera pose estimation from CNNs with BA [3]. The primal work BA-Net [19] formulates BA as a differentiable layer to enable end-to-end training of a neural network that predicts the camera motion and the per pixel depth of each frame using BA.

Even though, BA-Net is an important step towards combining CNNs with traditional optimization based techniques, there is no working source code available. In our project, we want to address this issue and reimplement BA-Net in pytorch to provide a baseline implementation. We originally intended to exactly reproduce the original results of BA-Net. However, this turned out to be more challenging than expected. First, the BA-Net paper does not provide all relevant details for reproducing the results. For example, the description of the data sampling lacks important information such as the threshold for the photo-consistency filtering, which prevented us from even reproducing the validation set, or training details such as batch size and the number of training iterations. Second, the source code provided by the authors of BA-Net is incomplete (the training setup and the data sampling is missing) and it contradicts the description in the paper (see Section 4.3). To address these issues, we designed our implementation in a configurable way to cover the different variants and unknown parameters. Also, we unit-tested the helper functions and the bundle adjustment code to avoid bugs. However, we encountered a very unstable training and the sheer amount of possible configurations as well as the long training time and the downtime of the Leonhard cluster prevented us from finding the reason in the complete setup. So, we narrowed down the scope of this project and focused on the training of the feature network and the differentiable bundle adjustment for the camera pose estimation.

The contributions of our project summarize as follows:

- We provide working pytorch implementations of all neural network components of BA-Net associated with camera motion estimation, most importantly including the differentiable BA layer.
- We introduce ideas from the paper "Taking a Deeper Look at the Inverse Compositional Algorithm" [16]

into our implementation that improve convergence and result in a better camera pose estimation. Specifically, we have implemented an improved damping factor estimation layer (see Section 4.4) and have demonstrated the effectiveness of a simplified multi-resolution feature net (see Section 4.5).

- We propose a Pose Initialization Network that learns to provide a good initialization for the bundle adjustment layer to further improve camera pose estimation (see Section 4.6).

2. Related Work

Recent works have studied estimating the 3D structure of the environment given images from the same scene using CNNs. In particular, Hand et al. [12] estimated the camera motion with a network from 2 images with known ground truth depth maps. Zhou et al. [27] used two CNNs, one for depth and camera motion estimation. Both networks were trained together with a photometric loss minimization in an unsupervised manner. More recently, the work from Clark et al. [3] solved the nonlinear least square problem in a binocular SfM setting, using an LSTM-RNN ([13]) as the optimization method.

BA-Net [19] bridges the classic white-box methods in computer vision with recent deep learning approaches by implementing BA as a differentiable optimization layer. A Multi Layer Perceptron (MLP) predicts the the damping factor for the LM algorithm, and a minimization of the feature-metric error between aligned CNN feature maps is performed. With an end-to-end trainable model, the feature extraction networks are able to learn appropriate features via backpropagation through the BA layer.

Lv et al. [16] proposed a combination of optimization and learning-based approaches as a variant of the original Lucas-Kanade image registration method, by training a general parametrized feed-forward model end-to-end. This algorithm incorporates knowledge about the geometry of the problem and enables a robust iterative estimation.

DeepSFM from Wei et al. [24] also aims to incorporate multi-view geometric constraints in a deep learning framework, but their method does not include restrictions on the number of LM iterations, and also their architecture is fully physical driven and suffers less from over-fitting issues. Shi et al. [18] also formulates the differentiable BA as a feature-metric layer, but with CNN input features from multiple images, optimizing structure and motion using a feature-metric error in a self-supervised fashion. DeepV2D from Tedd et al. [20] shares similarities with BA-Net, by also combining deep learning and multi-view geometry, but instead of performing only one nonlinear optimization considering all variables, decomposes the joint optimization into multiple smaller sub-problems. This decomposition of the problem makes the model more expressive since it is possible to optimize over per-pixel depth, and there is no constraints related to the use of depth basis maps (as in [19]) that could limit the accuracy of the depth estimate.

3. Methods

Our method is a re-implementation of BA-Net [19], a feature metric bundle adjustment network that implements a differentiable Levenberg-Marquardt (LM) [17] layer for nonlinear least squares optimization. This differentiable LM layer enables end-to-end training of the network to learn features and depth maps in a way that is beneficial for solving the SfM problem.

3.1. Bundle Adjustment (BA)

BA is a technique for jointly refining pose and depth estimations over a series of camera views. The optimization variables are defined as camera poses $\mathbb{T} = \{T_i | i = 1, \dots, N_i\}$, and scene point coordinates $\mathbb{P} = \{p_j | j = 1, \dots, N_j\}$, which are joinly represented as $\mathcal{X} = [T_1, T_2, \dots, T_{N_i}, p_1, \dots, p_{N_j}]$. The optimization problem seeks to minimize a cost function of the form

$$\mathcal{X}^* = \text{argmin} \sum_{i=1}^{N_i} \sum_{j=1}^{N_j} \| e_{i,j}(\mathcal{X}) \| \tag{1}$$

where $e_{i,j}(\mathcal{X})$ is the euclidean reprojection error in the case of sparse BA, or the photometric error in the case of dense or direct BA methods. In order to minimize this problem, the LM algorithm is typically used. At each iteration the optimal step is chosen by minimizing following equation with a regularization factor λ:

$$\Delta\mathcal{X}^* = \text{argmin} \| J(\mathcal{X})\Delta\mathcal{X} + E(\mathcal{X}) \| + \lambda \| D(\mathcal{X})\Delta\mathcal{X} \| \tag{2}$$

Here, $J(X)$ is the jacobian of the residual $E(X) = [e_{1,1}(\mathcal{X}), e_{1,2}(\mathcal{X}), \dots, e_{N_i,N_j}(\mathcal{X})]$ with respect to $\mathcal{X}$ and D is the element-wise square root of the diagonal entries of the approximated Hessian $J(\mathcal{X})^T J(\mathcal{X})$. Note that the definition of $E(\mathcal{X})$ is problem dependent. Our derivation of both can be found in Appendix A. Once the residuals and Jacobian have been computed, the optimization step from equation 2 can be solved in closed form:

$$\Delta\mathcal{X} = (J(\mathcal{X})^T J(\mathcal{X}) + \lambda D(\mathcal{X}))^{-1} J(\mathcal{X})^T E(\mathcal{X}) \tag{3}$$

After the optimal updates are determined, the solution of the k-th iteration can be found as:

$$\mathcal{X}_k = \Delta\mathcal{X} \circ \mathcal{X}_{k-1} \tag{4}$$

Dense methods [8, 5, 9] have yielded promising results while largely solving the challenges of sparse methods by

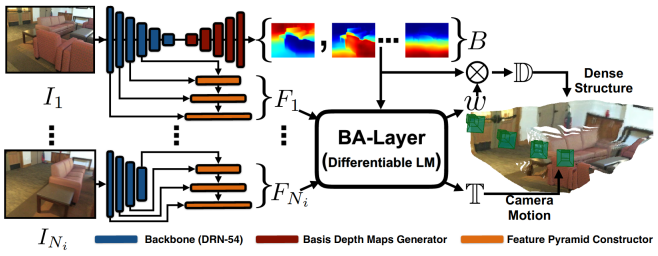


Figure 1: BA-Net Architecture diagram from original paper [19]

introducing a photometric loss that removes the need for feature matching. The error terms of the photometric loss are defined as:

$$e^f_{i,j}(\mathcal{X}) = I_i(\pi(T_i, d_j \cdot q_j)) - I_1(q_{i,j}) \tag{5}$$

where $d_j \in \mathbb{D} = \{d_j | j = 1, \ldots, N_j\}$ is the depth of a pixel q_j in the image I_1 and $d_j \cdot q_j$ converts it to its 3D coordinate. Thus the photometric BA optimization parameter vector is $\mathcal{X} = [T_1, T_2, \ldots, T_{N_i}, d_1, \ldots, d_{N_j}]$.

Direct methods have their own disadvantages. Most importantly, proper initialization becomes crucial as the photometric loss is highly non-convex, thus, not suitable for larger image baselines.

3.2. BA-Net Architecture

The BA-Net architecture consists of two major steps: feature map and depth map generation, then bundle adjustment. The feature maps and depth map generators both share a pretrained DRN [26] backbone encoder with a few modifications described in [19]. The original BA-Net architecture can be seen in Figure 1.

In BA-Net, feature maps are generated by a multi-scale hierarchy of convolutional network layers that extract features corresponding to different scale levels. This multi-scale approach is similar to direct methods and enables optimization of the camera poses at coarse and fine scales. Coarser layers increase the convergence radius of the BA, while finer layers increase the final accuracy of the converged solution. Each pyramid level is obtained from the corresponding DRN layer and the upsampled previous pyramid level. A convolutional filter is applied to reduce the dimension to the final feature maps to 128 channels.

Depth values are given special treatment in the BA-Net architecture. Per-pixel depth would be computationally intractable and hugely increase the number of parameters that must be learned. Instead, Tang et. al. implement a basis depth map method where the BA optimizes a linear combination of a set of basis depth images. These basis depth maps are generated from a convolutional network for monocular depth estimation. As is typical in monocular depth estimation, an encoder-decoder network is used to predict depth maps. BA-Net uses DRN as the encoder, in order to share the feature pyramid backbone with the depth prediction network, and a modified version of FCRN [15] with 128 output features (basis depth maps) as depth decoder. The final depth map is a linear combination of these depth maps:

$$\mathbb{D} = \text{ReLU}(w^T B) \tag{6}$$

Where $\mathcal{D}$ is the final depth map for the pixels of the first camera, B is a 128 channel image where each channel represents a basis depth map, and w is the coordinate vector optimized by the BA layer.

The novel contribution of Tang et. al. is the differentiable Bundle Adjustment layer that implements a differentiable nonlinear least squares optimizer. The goal of BA-Net is to use features that are trained via backpropagation rather than features that are pretrained for other problems and re-used. In order to backpropagate the gradients from the final output loss to all of the other layers of the network, a differentiable nonlinear least squares optimizer is needed. Due to its prevalence in conventional BA methods, the Levenberg Marquardt (LM) algorithm is chosen. LM is not inherently differentiable due to (1) an if-else termination strategy on a convergence criteria and (2) modification of the damping factor based on an if-else decision. These two problems are solved as follows: (1) a fixed iteration limit is implemented, removing the need for a convergence check, and (2) an MLP is trained to predict a suitable damping layer based on the current residuals of the problem, making the damping factor selection fully differentiable. LM iterations are done first on the coarse feature maps then worked down to the finest levels with 5 steps at each scale.

4. Experiments

4.1. Datasets

In this project, we use two datasets. We started with the KITTI dataset [10], which provides image sequences of a car-mounted camera as well as lidar point clouds that can be used as (sparse) ground truth depth information. For the training / test split as well as the crop of the image region with meaningful depth ground truth, we follow the work of Eigen et al. [7]. As there is no ground truth camera pose provided by KITTI itself, we generate ground truth poses using LibVISO2 [11]. During the project, we later switched to the ScanNet dataset [4] as it includes a higher variety of camera motion. ScanNet is a large indoor dataset with 1,513 sequences in 706 different scenes compromising of 2.5 million RGB-D images including camera motion. To reduce the the required disk space, we downsample the images and depth to 320x240 pixels. From the image sequences, we sample 250,000 training and 200 validation pairs, each with an offset of 15 frames.

Table 1: Depth Estimation on KITTI

Method	Abs Rel	Squ Rel	RMSE	RMSE log
Eigen [7]	0.203	1.548	6.307	0.282
Zhou [27]	0.208	0.1768	6.856	0.283
Wang [23]	0.151	1.257	5.583	0.228
Ours	0.106	0.717	4.266	0.176
BA-Net [19]	**0.083**	**0.025**	**3.640**	**0.134**

4.2. Depth Estimation

As a first step, we implemented and trained the DRN-22 [26] backbone and the basis depth map generator based on a FCRN decoder [15]. As described in the BA-Net paper, we replace the dilations of DRN with convolutions with strides to reduce the memory footprint. To test both implementations independent of the BA layer, we use a 1x1 convolution to learn the weight vector that combines the basis depth maps. We initialize the DRN-22 backbone with pretrained weights [26] and train the network using adam [14] with a learning rate of 0.001 and a batch size of 4 on the KITTI dataset. As proposed in [15] we use the berHu loss.

For evaluation, we provide the common metrics for depth estimation, specifically the RMSE of the raw and logarithmically-scaled depth as well as the absolute and the squared relative difference, which are the mean of the ratios of the absolute / squared error and the ground truth depth. As shown in Table 1, our implementation without bundle adjustment outperforms the baselines, which BA-Net compared with. The performance gap of our implementation and BA-Net is probably due to the missing BA layer. Qualitative results including a selection of basis depth maps can be seen in Figure B.1. Note that depth estimations in the top part of the image are incorrect due to the missing ground truth in that area as the lidar only covers the scene up to a certain height. During quantitative evaluation those regions are ignored by cropping the image as suggested in [7].

Joint training of the entire BA-Net architecture proved to be very unstable. The gradients caused by the $SE(3)$ pose error conflicted with the one caused by the depth prediction error, hurting depth predictions. For this reason and the long down-time of the Leonhard cluster, we have decided to simplify our setup and to focus on the camera pose estimation first. Also, we transitioned from the KITTI dataset to ScanNet as KITTI mostly contains degenerate forward motion and ScanNet can provide us with a wide variety of motions.

4.3. Camera Pose Estimation with BA-Net

In order to focus on the camera pose estimation, we provide the bundle adjustment with the ground truth depth and only optimize for the camera pose. During re-implementing the relevant network layers in pytorch, we found several contradictions between the description in the BA-Net paper and their Github repository. First, in the source code of the feature pyramid constructor, the feature channels are projected to 128 before concatenating them with the next level, while in the paper, all feature channels are concatenated with the next level and only the output of each level is projected to 128 channels. The latter results in a much higher number of feature channels, which noticeably increases runtime. Second, the damping layer in the paper contains ReLU activations until the end, while the implementation uses SELU activations for the intermediate layers and tanh for the last layer. Moreover, the output of the damping layer in the source code is the norm of the residuals to the power of (2 + the last activation). We have decided to generally follow the Github implementation as we assume that it has a higher chance to be correct. For faster training, we use the smaller DRN-22 backbone instead of the DRN-54 backbone. For the training, we use ADAM [14] with a learning rate of 0.001, a batch size of 1, 30k training iterations to fit the slowest model into the Leonhard 24 h queue, and the euclidean distance and the relative angle error as loss function. As baselines, we use the camera pose of the first camera assuming no camera motion and a photometric BA with five scales and five iterations for each scale.

Table 2 shows the translation error (euclidean distance in cm) and the rotation error of the BA-Net implementation (row 3) compared to both baselines (row 1 and 2). It can be seen that BA-Net outperforms both baselines by a noticeable margin. Especially, the translation error is reduced compared to the photometric BA, which performs even worse than using the identity transform. This shows the effectiveness of using a learned feature metric error instead of the photometric error. The latter results in a highly non-convex objective function that is difficult to optimize.

Note that, we are not able to directly compare with the results from the BA-Net paper as they only provide results on joint BA for pose and depth estimation and their validation set is not reproducible. The value that is used for filtering validation pairs is based on the photometric error using the ground truth depth and pose is unspecified and highly influences the difficulty of the validation set. If a low value is used, the validation set does not contain occlusions or lightning changes anymore.

4.4. Improved Damping Factor Estimation

In this section, we further study the influence of the damping factor layer. First, we implement the damping layer according to the paper description instead of the official code. With the modified damping activation, we had to reduce the learning rate to 3×10^{-5}, which also becomes our default learning rate for all of the following experiments. The paper version with ReLU activations (row 4) outperforms the Github version (row 3) for both camera translation and rotation estimation.

Table 2: Camera Pose Estimation Performance

	variant	rot. (°)	trans. (cm)
1	No Camera Motion	9.051	13.54
2	Photometric BA	5.855	14.88
3	**BA-Net**	4.109	10.57
4	- damping ReLU	3.197	9.319
5	- damping ELU	3.188	9.288
6	**- modified damping**	**2.834**	**8.592**
7	- mod. d. w/o pretrain	3.104	9.434
8	**DIC**	2.246	7.035
9	- w/o level 5	3.281	10.14
10	- both frames	2.435	7.506
11	- loss all levels	2.473	7.141
12	- w/o min. valid depth	4.82	13.76
13	**- Pose Init Net**	**2.159**	**5.922**

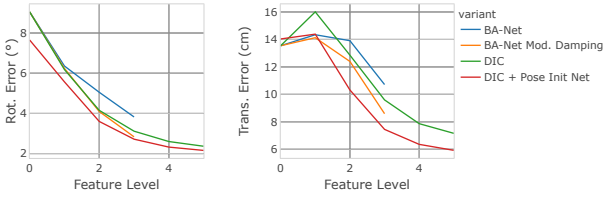


Figure 2: Camera pose error after bundle adjustment for the corresponding feature level.

We further propose a damping factor network with two modifications. First, we replace the ReLU by an ELU activation (with an offset of 1 for the last layer to prevent negative values) to have continuous gradients for low damping values. Second, we use independent damping factor estimation networks for each feature level. This is important as the residuals that are used as input to the damping layer depend not only on the quality of the camera pose estimation but also heavily on the feature level. This can be observed in Figure B.2a, where the residuals and lambda increase at iteration 5 when switching to the next feature pyramid level. Table 2 shows that the proposed changes further improve the camera pose estimation performance of BA-Net (compare row 6 and 4). The better utilization of the feature levels with the independent damping factor estimation, can also be seen in Figure 2 (compare orange with blue), where the rotation and translation error are plotted after the last iteration of each bundle adjustment level. In Figure 3, we also visualize the distribution of the errors over the samples of the validation set. It shows that the modified damping layer (orange) mostly improves the performance for medium error cases, while the outliers are mostly unaffected.

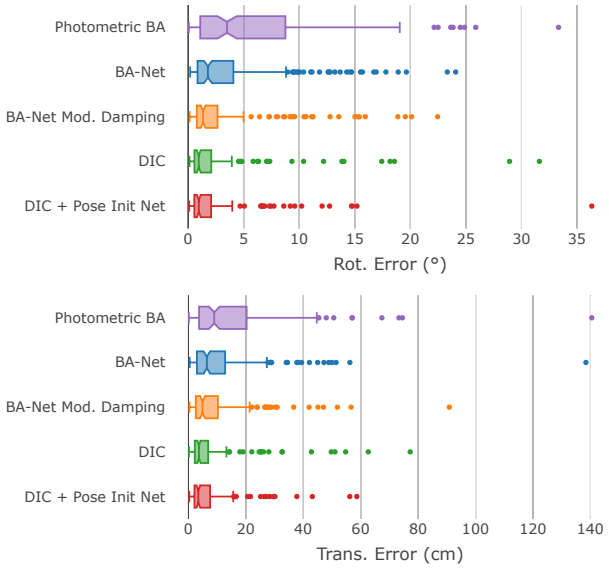


Figure 3: Camera pose error distribution over the validation set for different architectures.

4.5. Improved Feature Net

Next, we further analyse the feature network. First, we ablate the influence of the pretraining of the DRN encoder. The comparison of row 6 and 7 in Table 2 shows that initializing the backbone with pretrained weights improves the performance slightly. Second, we compare the U-Net-style feature network that BA-Net uses with an alternative feature network (referred to as DIC) using dilated convolutions [25] but no skip connections. This is inspired by the network design of the paper "Taking a Deeper Look at the Inverse Compositional Algorithm" [16]. The network consists of 5 convolutional blocks, where each block contains three 3x3 convolutions with BatchNorm and ELU activation. The second and the third convolution have a dilation rate of 2 and the second convolution increases the number of channels. The output channels of the blocks are 32, 64, 96, 128, and 160 respectively. Importantly, we use reflection padding for the convolution to avoid border artifacts. The output of a block serves as a feature level for the BA before it is downsampled to half the size using average pooling and passed to the next block. In contrast to [16] we use 5 blocks instead of 4 and do not sum the channels of a feature level for the BA. Note that this network is not pretrained. As we have 5 levels for the BA, we reduce the number of iterations to 3. For the damping layer, we use the modified version from the previous section. One important aspect due to the downtime of Leonhard is the reduced training time of the DIC feature net, which is lower by a factor of two compared to BA-Net.

Table 2 shows that the alternative feature network (DIC, row 8) noticeably improves the performance compared to

the BA-Net variants from the previous sections. Figure 3 shows that the improved performance is consistent across the entire error distribution except some outliers.

For our implementation of the feature net, we have taken design decisions that differ from the proposed architecture in [16]. First, we use 5 feature levels instead of 4. Table 2 (compare row 8 and 9) shows that the fifth block provides a significant performance improvement. Second, in [16] the feature net is provided with both images. Even though, in their setting this worked better, in ours it slightly decreases performance (compare row 10 and 8). The same is the case for applying the loss to the intermediate camera pose estimations of the BA after each feature level (row 11). Third, we want to mention an implementation detail that was crucial for our setting. In order to train the network also for camera translation, we had to filter points that were closer than 50 cm to the camera. Table 2 row 11 shows the results without that filtering. The camera translation is barely optimized in this case. This happens because most of the terms in the projection jacobian are inversely proportional to the depth of the 3d points in the target camera frame, causing the points close to the camera to overpower the contribution of the points that are further away. Note that in the ScanNet dataset there are many points that are classified as invalid and have zero depth, additionally there are some noisy depth estimates which contain points that also hurt the optimization process.

4.6. Pose Initialization Network

The initial pose estimate is crucial for the success of the BA. So far, we followed BA-Net and used the identity as initialization. To further improve the performance of the camera pose estimation, we propose to learn the pose initialization using a small network consisting of three convolutional layers with 3x3 kernels on top of the concatenated last feature levels of both images. Table 2 shows that the pose initialization network further improves the translation error by 1 cm. Surprisingly, Figure 2 shows that the improved translation estimation is not due to a better first estimate but a better convergence in later BA levels. We assume that the pose initialization network learns to predict a pose that is not necessarily very accurate but rather results in a better BA optimization. This hypothesis is also qualitatively supported by Figure B.2c.

5. Conclusions

The re-implementation of [19] proved to be a challenging task due to lack of a complete open-source implementation and the discrepancies within the article and the available code. Additionally, the training of neural network that back-propagates gradients through a Levenberg-Marquardt optimization module was found to be really unstable and it requires careful exploration of the architecture configuration, hyperparameters and training data to be able to achieve reasonable results in the optimized predictions. On a simplified setup, in which we use the ground truth depth and only optimize the camera pose, we show that our reimplementation of BA-Net, including a differentiable bundle adjustment layer, significantly outperforms photometric BA. We show that BA-Net can further be improved using a simplified feature network, a layer-wise damping factor network, and a pose initialization network.

6. Work Distribution and External Resources

Work Distribution

- Adrian: Scannet data loader + Depth and camera jacobian calculations for LM + BA layer implementation + CI and testing
- Lukas: Kitti data loader + Implementation DRN, FCRN, FPN + Setup training framework + Depth estimation on Kitti + Training of feature pyramid for camera pose estimation + Ablations on damping layer, feature net, loss function, pose init net + Visualizations
- Tommaso: Ground truth estimation of the Kitti camera poses using libviso2 [11] + Leonhard training setup
- Josè: Implementation of learned damping factor layer + Depth and camera jacobian calculations for LM + BA layer implementation + Setup training framework with pytorch-lightning + Extensive unittesting of jacobians and forward BA implementation

External Source Code For our implementation, we have used source code from several sources. Specifically, for the data loading of Kitti, we used the data loader of monodepth2[1] as template and for the generation of the ground truth poses, we used libviso2[2]. To convert the implementation of the network components of BA-Net from tensorflow to pytorch, we adapted following implementations of DRN[3] and FCRN[4]. In particular, we matched those implementations with the described changes in the BA-Net paper and the tensorflow code fragments[5][6]. For the DIC feature net, we have used[7] as starting point. We also utilized some geometry related helper functions provided by our supervisor Paul.

[1] https://github.com/nianticlabs/monodepth2/tree/master/datasets

[2] http://www.cvlibs.net/software/libviso/

[3] https://github.com/fyu/drn/blob/master/drn.py

[4] https://github.com/dontLoveBugs/FCRN_pytorch/blob/master/network/FCRN.py

[5] https://github.com/frobelbest/BANet/blob/master/enc.py

[6] https://github.com/frobelbest/BANet/blob/master/dec.py

[7] https://github.com/lvzhaoyang/DeeperInverseCompositionalAlgorithm/blob/master/code/models/algorithms.py#L119

References

[1] S. Agarwal, N. Snavely, S. M. Seitz, and R. Szeliski. Bundle adjustment in the large. In *European conference on computer vision*, pages 29–42. Springer, 2010. 1

[2] J. L. Blanco. A tutorial on se(3) transformation parameterizations and on-manifold optimization. 09 2010. 8

[3] R. Clark, M. Bloesch, J. Czarnowski, S. Leutenegger, and A. J. Davison. Learning to solve nonlinear least squares for monocular stereo. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 284–299, 2018. 1, 2

[4] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5828–5839, 2017. 3

[5] A. Delaunoy and M. Pollefeys. Photometric bundle adjustment for dense multi-view 3d modeling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1486–1493, 2014. 2

[6] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE international conference on computer vision*, pages 2650–2658, 2015. 1

[7] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in neural information processing systems*, pages 2366–2374, 2014. 1, 3, 4

[8] J. Engel, T. Schöps, and D. Cremers. Lsd-slam: Large-scale direct monocular slam. In *European conference on computer vision*, pages 834–849. Springer, 2014. 2

[9] J. Engel, V. Usenko, and D. Cremers. A photometrically calibrated benchmark for monocular visual odometry. *arXiv preprint arXiv:1607.02555*, 2016. 2

[10] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012. 3

[11] A. Geiger, J. Ziegler, and C. Stiller. Stereoscan: Dense 3d reconstruction in real-time. In *2011 IEEE intelligent vehicles symposium (IV)*, pages 963–968. Ieee, 2011. 3, 6

[12] A. Handa, M. Bloesch, V. Pătrăucean, S. Stent, J. McCormac, and A. Davison. gvnn: Neural network library for geometric computer vision. In *European Conference on Computer Vision*, pages 67–82. Springer, 2016. 2

[13] S. Hochreiter, A. S. Younger, and P. R. Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pages 87–94. Springer, 2001. 2

[14] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 4

[15] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab. Deeper depth prediction with fully convolutional residual networks. In *2016 Fourth international conference on 3D vision (3DV)*, pages 239–248. IEEE, 2016. 1, 3, 4

[16] Z. Lv, F. Dellaert, J. M. Rehg, and A. Geiger. Taking a deeper look at the inverse compositional algorithm. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4581–4590, 2019. 1, 2, 5, 6

[17] J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006. 1, 2

[18] Y. Shi, J. Zhu, Y. Fang, K. Lien, and J. Gu. Self-supervised learning of depth and ego-motion with differentiable bundle adjustment. *arXiv preprint arXiv:1909.13163*, 2019. 2

[19] C. Tang and P. Tan. Ba-net: Dense bundle adjustment network. *arXiv preprint arXiv:1806.04807*, 2018. 1, 2, 3, 4, 6

[20] Z. Teed and J. Deng. Deepv2d: Video to depth with differentiable structure from motion. *arXiv preprint arXiv:1812.04605*, 2018. 2

[21] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon. Bundle adjustment—a modern synthesis. In *International workshop on vision algorithms*, pages 298–372. Springer, 1999. 1

[22] B. Ummenhofer, H. Zhou, J. Uhrig, N. Mayer, E. Ilg, A. Dosovitskiy, and T. Brox. Demon: Depth and motion network for learning monocular stereo. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5038–5047, 2017. 1

[23] C. Wang, J. Miguel Buenaposada, R. Zhu, and S. Lucey. Learning depth from monocular videos using direct methods. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2022–2030, 2018. 4

[24] X. Wei, Y. Zhang, Z. Li, Y. Fu, and X. Xue. Deepsfm: Structure from motion via deep bundle adjustment. *arXiv preprint arXiv:1912.09697*, 2019. 2

[25] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015. 5

[26] F. Yu, V. Koltun, and T. Funkhouser. Dilated residual networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 472–480, 2017. 3, 4

[27] 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*, pages 1851–1858, 2017. 2, 4

A. Residual and Jacobian Derivations

For feature-metric BA, the residual matrix can be defined component-wise like in Equation 5:

$$e^f_{i,j}(\mathcal{X}) = F_i(\pi(T_i, \mathrm{ReLU}(w^T B_j) \cdot q_j)) - F_1(q_{i,j}) \tag{7}$$

where B_j is the j-th column of B.

The jacobian of the residual $E(\mathcal{X})$ with respect to the optimization variable camera transformations can be derived using the chain rule as follows:

$$J_{\mathrm{T}} = \frac{\partial E}{\partial T} = \begin{bmatrix} \frac{\partial F}{\partial u} & \frac{\partial F}{\partial v} \end{bmatrix} J_{\mathrm{camera}} \tag{8}$$

where $\frac{\partial F}{\partial u}$ and $\frac{\partial F}{\partial v}$ are the feature partials with respect to the pixel coordinates u and v, and J_{camera} is the jacobian of the projection from 3D scene coordinates. For our purposes, we use the definition that $T_i = [t_i \omega_i]$ where t_i is the translation of the transformation and ω_i is the rotation represented in angle axis form. The feature partials are approximated using discrete finite differences on the height and width dimensions of the feature maps.

The camera jacobian J_{camera} is the jacobian of the point projection function π with respect to the camera pose parameters T. The derivation can be found in [2] and the result is:

$$J_{\mathrm{camera}} = \frac{\partial \pi}{\partial T} = \begin{bmatrix} \frac{f_x}{p_z} & 0 & -\frac{f_x g_x}{g_z^2} & -f_x \frac{g_x g_y}{g_z^2} & f_x(1 + \frac{g_x^2}{g_z^2}) & -f_x \frac{g_y}{g_z} \\ 0 & \frac{f_y}{p_z} & -\frac{f_y g_y}{g_z^2} & -f_y(1 + \frac{g_y^2}{g_z^2}) & f_y \frac{g_x g_y}{g_z^2} & f_y \frac{g_x}{g_z} \end{bmatrix} \tag{9}$$

The point depth jacobians can be derived in a similar fashion:

$$J_{\mathrm{w}} = \frac{\partial E}{\partial w} = \begin{bmatrix} \frac{\partial F}{\partial u} & \frac{\partial F}{\partial v} \end{bmatrix} J_{\mathrm{depth}} \tag{10}$$

where J_{depth} is the jacobian of the projection function with respect to the depth basis coordinate vector w. The main derivation is also in [2] with a minor modification to account for our definition of the point depth as a matrix vector multiplication

$$J_{\mathrm{depth}} = \frac{\partial \pi}{\partial w} = \begin{bmatrix} \frac{f_x}{g_z} & 0 & -f_x \frac{g_x}{g_z^2} \\ 0 & \frac{f_y}{g_z} & -f_y \frac{g_y}{g_z^2} \end{bmatrix} R B_j \tag{11}$$

Assuming that the overall parameter vector is defined as $\mathcal{X} = [T_1, \ldots, T_{N_i}, w]$, the final jacobian can be written as:

$$J(\mathcal{X}) = \left[\begin{array}{ccc|c} J_{T_1} & & & \\ & \ddots & & J_w \\ & & J_{T_{N_i}} & \end{array}\right] \tag{12}$$

B. Qualitative Examples

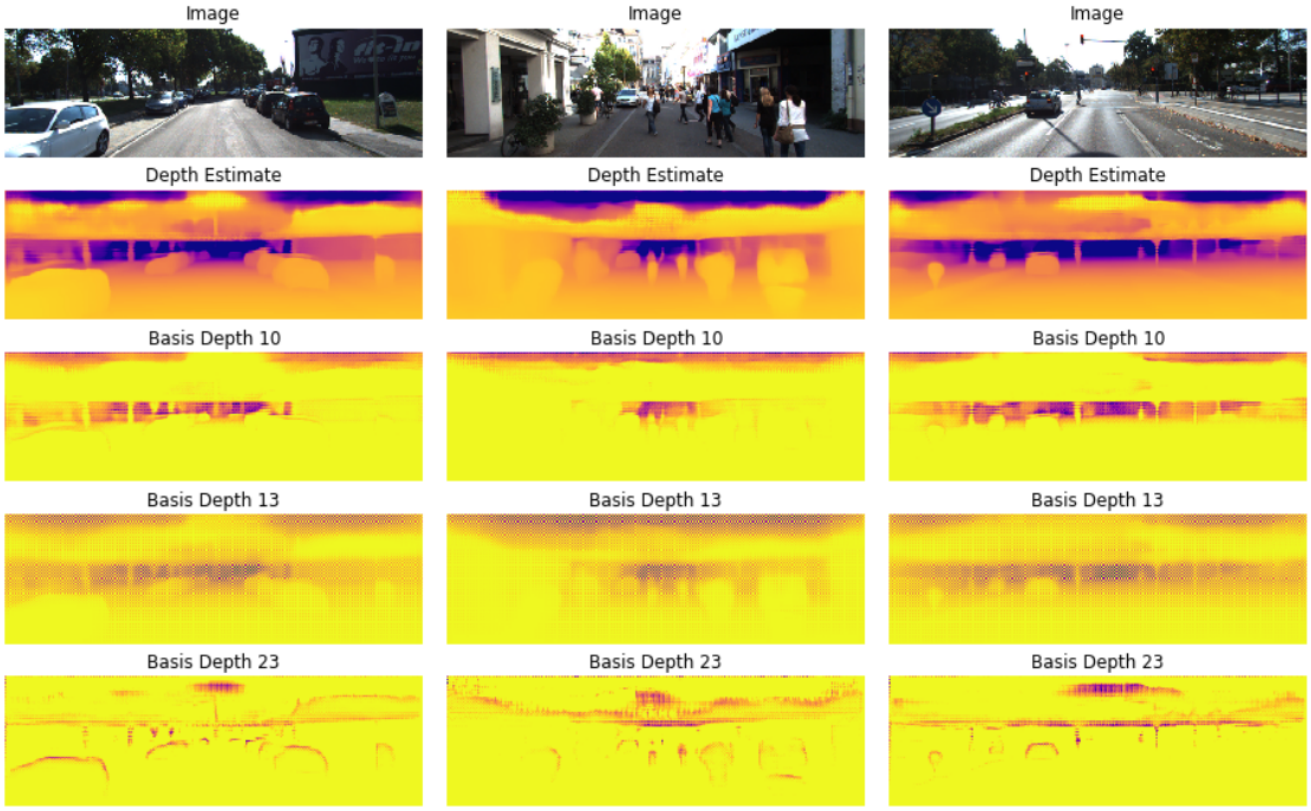
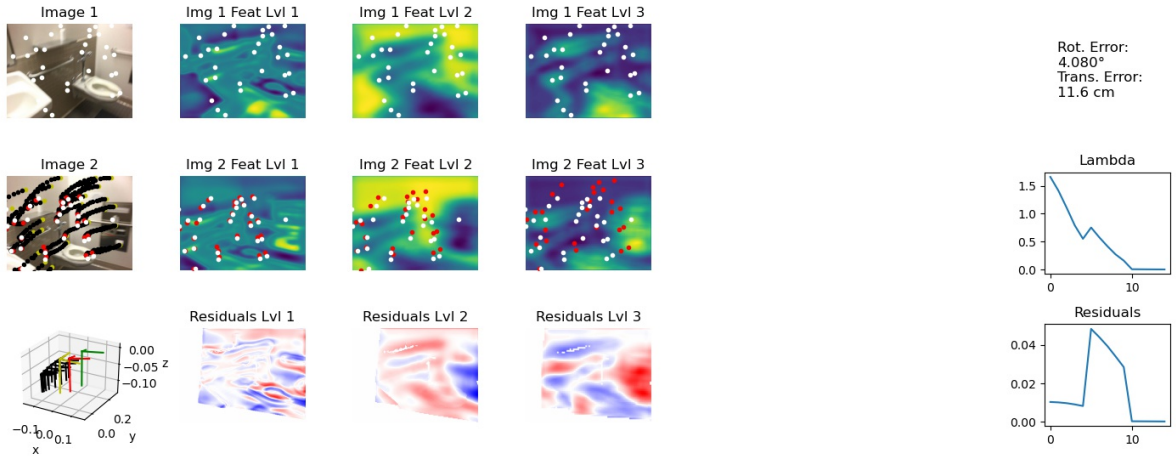
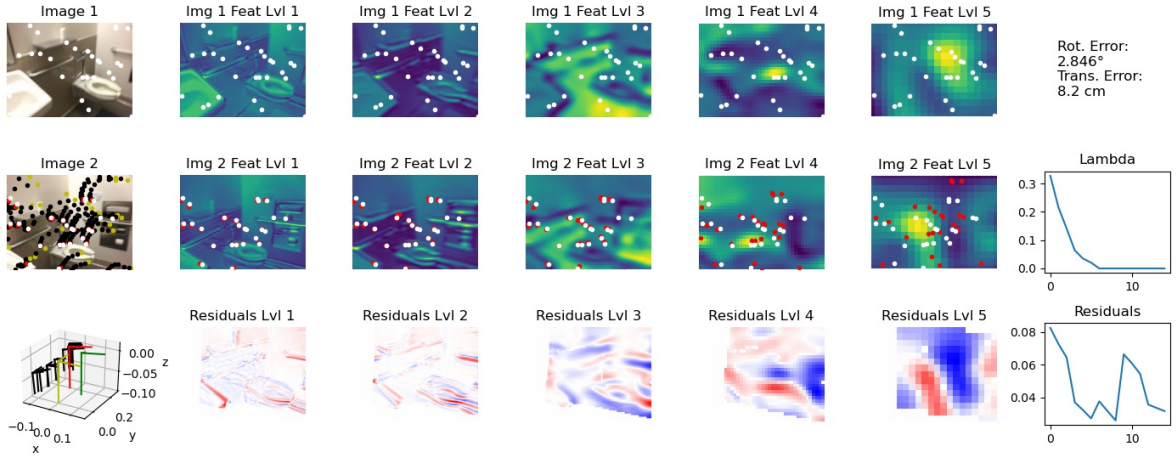


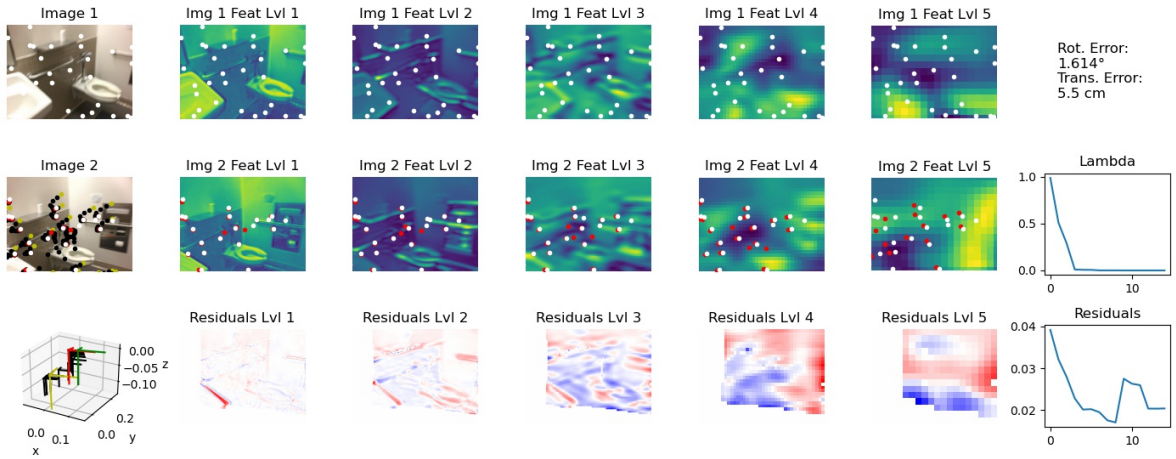
Figure B.1: Qualitative examples of the depth estimation network without bundle adjustment as well as their basis depth maps.



(a) BA-Net

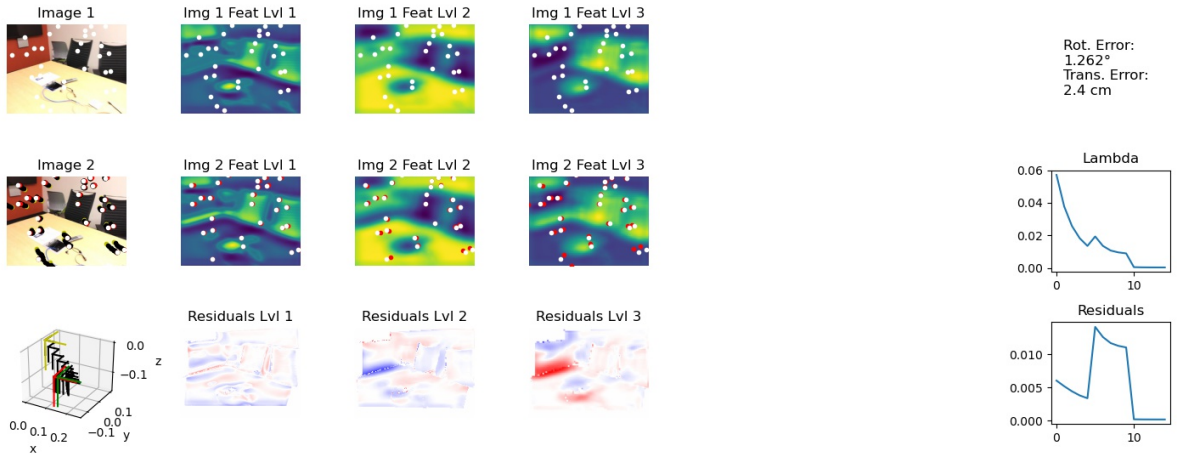


(b) DIC

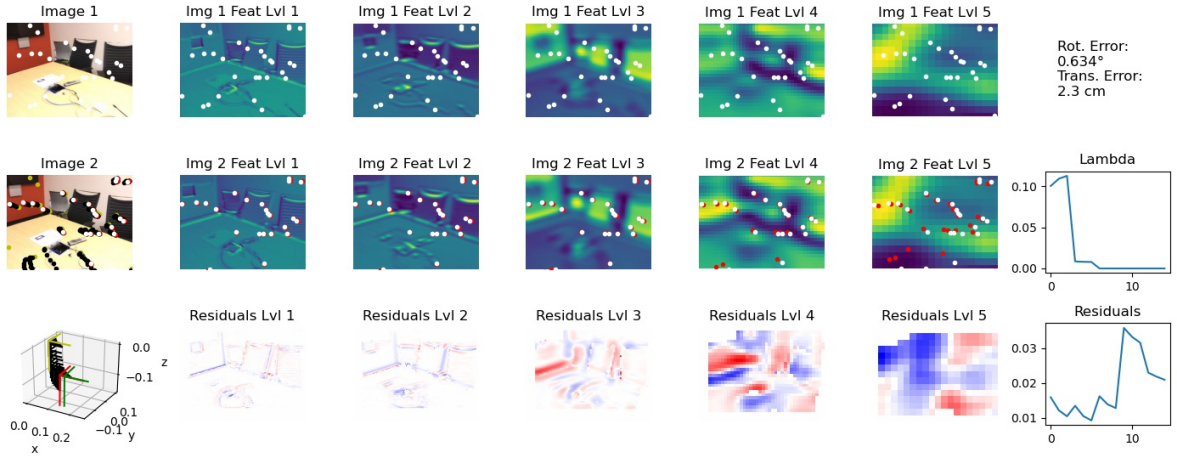


(c) DIC + Pose Initialization Network

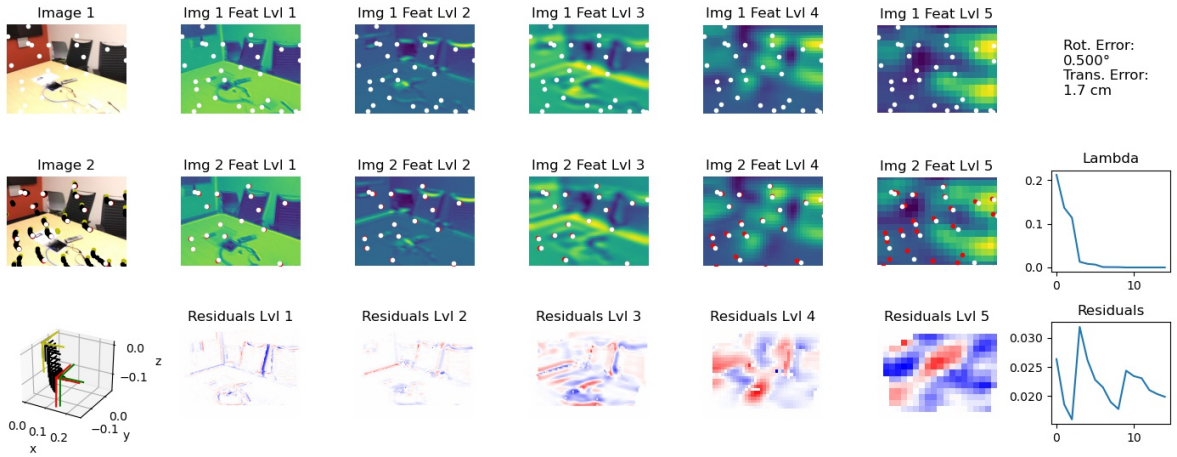
Figure B.2: Camera pose estimation on example 1 for different architectures. The image pair as well as the corresponding feature levels (average over all channels) as well as the residuals are shown. We also plot the camera poses during the optimization (bottom left). The yellow pose is the initialization, the black one the poses during the BA optimization, the red one the final prediction, and the green one the ground truth. In the images, a random point cloud is projected into the images. Here the ground truth is visualized in white instead of green for better visibility. The red points in the feature levels correspond to the predicted pose after the BA for the corresponding feature level.



(a) BA-Net

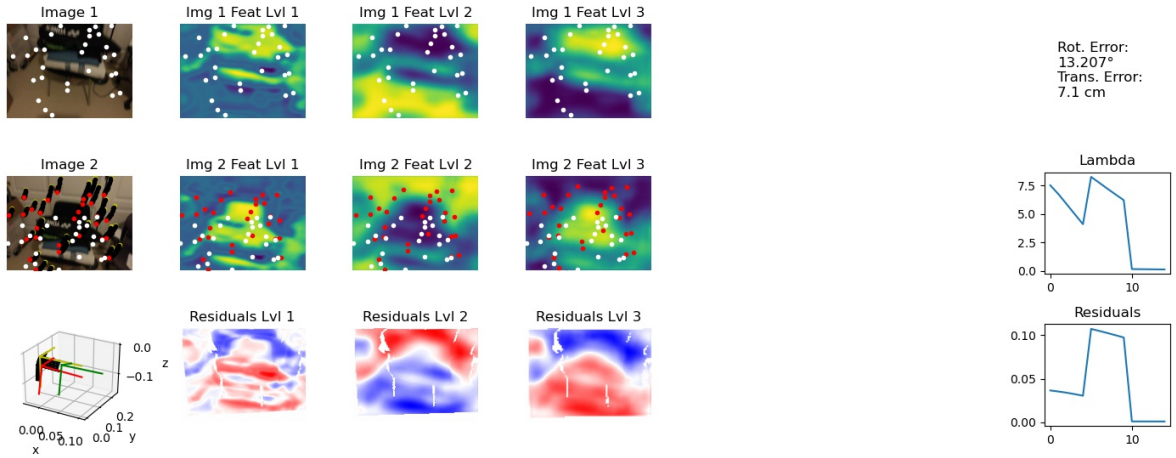


(b) DIC

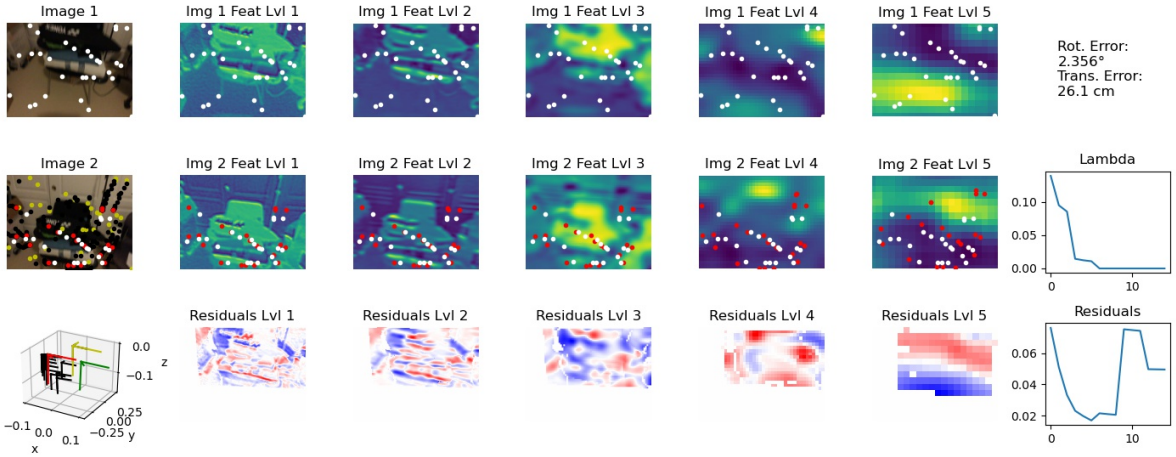


(c) DIC + Pose Initialization Network

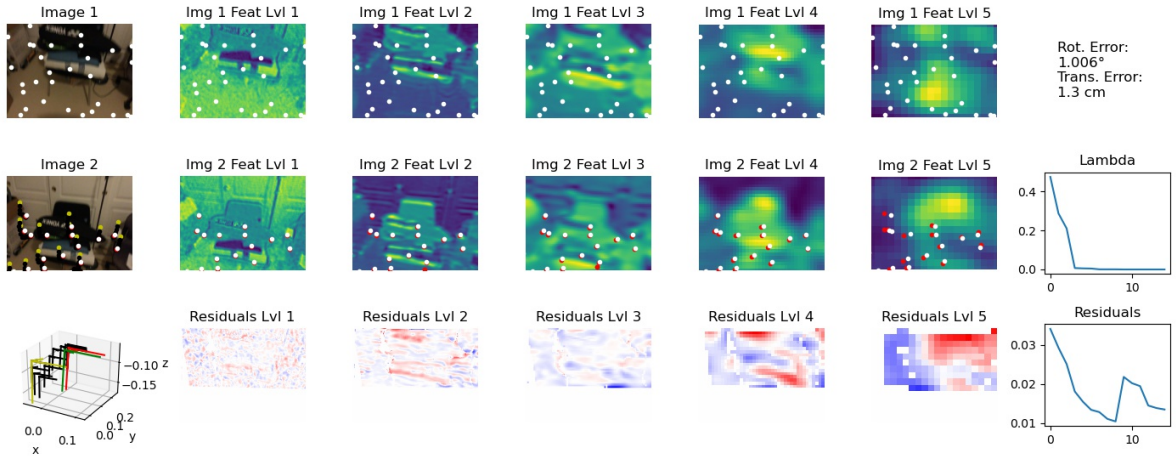
Figure B.3: Camera pose estimation on example 2 for different architectures. See Figure B.2 for further explanations.



(a) BA-Net



(b) DIC



(c) DIC + Pose Initialization Network

Figure B.4: Camera pose estimation on example 3 for different architectures. See Figure B.2 for further explanations.