An Empirical Evaluation of Sequence-Based Deep Learning Architectures for Visual Odometry

Emilio Parisotto

Saturday 28th April, 2018

Abstract

Background. Many applications in 3D environments require the knowledge of an agent's position in that environment, including a wide range of important basic tasks such as environment navigation, mapping, and planning. One method to estimate position is visual odometry, where the egomotion of a camera is estimated by processing the relative movement between video frames recorded by the camera. Previous methods to do visual odometry relied on a mixture of feature matching and hand-coded heuristics to estimate egomotion, while some newer methods tried to learn it end-to-end using deep learning. Within the deep learning literature on egomotion, only several sequence-based models have been tried such as LSTMs or convolutional networks, but comparing between these models and other new ones (such as transformer networks) hasn't been done thoroughly.
Aim. The aim of this project is to evaluate the current effectiveness of several state-of-the-art deep learning architectures on the task of visual odometry within the KITTI odometry dataset.
Data. The KITTI odometry dataset is a set of videos collected using a self-driving car sensor platform for the purposes of estimating egomotion accurately from video. It consists of 21 sequences with sequence lengths of between 271 to 4981 RGB images, with around 41,000 images in total. 11/21 of the sequences have public ground-truth egomotion available, collected using a state-of-the-art localization system within the vehicle.
Methods. To prevent overfitting due to the low number of data samples (11 sequences total, 3 training sequences) we use pre-trained image motion featurization models which were trained on external data and are fixed during training of the sequence models. After featurization, we evaluate several sequenced-based models on the KITTI dataset: temporal convolutions, LSTM recurrent networks and attention-based transformer networks.
Results. We evaluate the models using relative-frame RMSE and global-frame absolute trajectory error (ATE). Relative Translational (Rotational) RMSE is proportional to the true training loss used, and measures L2 error between the predicted position (quaternion) and the ground-truth position (quaternion). ATE measures global error after aligning the estimated trajectory with the ground-truth using a scaling, rotation and translation transformation.
Conclusions. In conclusion, we evaluated a variety of learning-based sequential models on the task of visual odometry in KITTI and our results show that temporal residual networks obtain slightly lower ATE than the other models tested, suggesting that most of the models are only utilizing temporally local information and the benefits of long-temporal-horizon sequential models is not fully realized. The lack of a large visual odometry dataset thus presents a significant challenge for learning-based models to reach state-of-the-art performance.

1 Introduction

Determining one's position in an environment relative to other objects such as fixed landmarks is a universal component of many robotics applications, e.g. a roomba robotic vacuum cleaner needs to know where it has already cleaned to avoid wasting power and cleaning as fast as possible. There are several techniques to estimate an agent's position and orientation, usually under the name of odometry (and its variants) or the more complete Simultaneous Localization and Mapping (SLAM) approaches. Odometry is mainly the estimation of egomotion (the agent's relative motion between inputs) using some sensor modality, whereas SLAM aims to map the entire environment when starting in an unknown position within that environment. Visual odometry is odometry with images as its sensor modality and is used to determine an agent's position given camera data of its trajectory through time. By observing how the objects in frame move, it can be possible to recover the transformation the camera has underwent between two image frames.
Visual odometry is one method of estimating position, but it is prone to drift error where small errors in the relative transformation between frames compound over time, resulting in a growing absolute difference in position in the agent's true position and its estimated one as time progresses. This can be remedied by doing relocalization in a map of the environment. Unfortunately, this map is often unavailable and its construction depends on an agent that can already accurately localize. This problem of needing to simultaneously localize and map an environment is the problem of SLAM, and often relies on odometry methods as a subprocess of the technique.
In this paper, we focus on evaluating deep learning sequence models on the task of visual odometry in the KITTI benchmark. Our models are visual odometry techniques because we forego building any explicit map or doing relocalization techniques. In the sequel, we present a review of related work on SLAM and visual odometry. Then we describe the dataset, the KITTI visual odometry dataset. Afterwards, we detail the image featurization models we use to preprocess the dataset and describe the sequence models we evaluate on KITTI. Finally, we present results on the KITTI dataset for all the models we evaluate.

2 Background and Related Work

Despite the method we test not necessarily being a SLAM technique, we present overview of previous SLAM methods as they are closely related. As previously stated, SLAM is a process in which an agent needs to localize itself in an unknown environment and build a map of this environment at the same time, with uncertainties in both its motions and observations. SLAM has evolved from filter-based to graph-based (optimization-based) approaches. Some EKF-based systems have demonstrated state-of-the-art performance, such as the Multi-State Constraint Kalman Filter (Mourikis & Roumeliotis, 2007), the VIN (Kottas et al., 2013), and the system of Hesch et al. (Hesch et al., 2014). Those methods, even though efficient, heavily depend on linearization and Gaussian assumptions, and thus underperform their optimization-based counterparts, such as OK-VIS (Leutenegger et al., 2015), ORB-SLAM (Mur-Artal et al., 2015), and LSD-SLAM (Engel et al., 2014).
Graph-based SLAM typically includes two main components: the front-end and the back-end. The

front-end extracts relevant information (e.g. salient features) from the sensor data and associates each measurement to a specific map feature, while the back-end performs graph optimization on a graph of abstracted data produced by the front-end.
Graph-based SLAM can be categorized either as feature-based or direct methods depending on the type of front-end. Feature-based methods rely on local features (e.g. SIFT, SURF, FAST, ORB, etc.) for pose estimation. For example, ORB-SLAM (Mur-Artal et al., 2015) performs data association and camera relocalization with ORB features and DBoW2 (Gálvez-López & Tardos, 2012). RANSAC (Fischler & Bolles, 1987) is commonly used for geometric verification and outlier rejection, and there are also prioritized feature matching approaches (Sattler et al., 2016). However, hand-engineered feature detector and descriptors are not robust to motion blur, illumination changes, or strong viewpoint changes, any of which can cause localization to fail.
To avoid some of the aforementioned drawbacks of feature-based approaches, direct methods, such as LSD-SLAM (Engel et al., 2014), utilize extensive photometric information from the images to determine the pose, by minimizing the photometric error between corresponding pixels. This approach is in contrast to feature-based methods, which minimize the reprojection error. However, such methods are usually not applicable to wide baseline settings (Cadena et al., 2016) during large viewpoint changes. Recent work in (Forster et al., 2014) (Forster et al., 2017) combines feature and direct methods by minimizing the photometric error of features lying on intensity corners and edges. Some methods focus on dense recontruction of the scene, for instance (Whelan et al., 2016) builds dense globally consistent surfel-based maps of room scale environments explored using an RGB-D camera, without pose graph optimization, while KinectFusion (Newcombe et al., 2011) obtains depth measurements directly using active sensors and fuses them over time to recover high-quality surface maps. These approaches still suffer from strict calibration and synchronization requirements, and the data association modules require extensive parameter tuning in order to work correctly for a given scenario.
In light of the limitations of feature-based and direct approaches, deep networks are proposed to learn suitable feature representations that are robust against motion blur, occlusions, dynamic scenes, illumination, texture, and viewpoint changes. They have been successfully applied to several related multiview vision problems, including learning optical flow (Dosovitskiy et al., 2015), depth (Liu et al., 2015), homography between frame pairs (DeTone et al., 2016), and localization (Chaplot et al., 2018) and re-localization problems.
Recent work includes re-formulating the localization problem as a classification task (Weyand et al., 2016), a regression task (Kendall et al., 2015; Walch et al., 2016), end-to-end trainable filtering (Haarnoja et al., 2016), and differentiable RANSAC (Brachmann et al., 2017). More specifically, PlaNet (Weyand et al., 2016) formulates localization as a classification problem, predicting the corresponding tile from a set of tiles subdividing Earth surface for a given image, thus providing the approximate position from which a photo was taken. PoseNet (Kendall et al., 2015) formulates 6-DoF pose estimation as a regression problem. Similarly, (Melekhov et al., 2017) finetunes a pretrained classification network to estimate the relative pose between two cameras. To improve its performance, (Walch et al., 2016) added Long-Short Term Memory (LSTM) units to the fully-connected layers output, to perform structured dimensionality reduction, choosing the most useful feature correlations for the task of pose estimation. From a different angle, DSAC (Brachmann

et al., 2017) proposes a differentiable RANSAC so that a matching function that optimizes pose quality can be learned. These approaches are not robust to repeated structure or similar looking scenes, as they ignore the sequential and graphical nature of the problem. Addressing this limitation, work in (Clark et al., 2017) fused additional sequential inertial measurement with visual odometry. The Neural Graph Optimizer (Parisotto et al., 2018) used a transformer + temporal convolution hybrid model to do pose estimation in several simulated environments, drawing inspiration from graph SLAM methods which combine both with local estimation and global graph optimization. SemanticFusion (McCormac et al., 2017) combines convolutional neural networks (CNNs) and a dense ElasticFusion (Whelan et al., 2016). However, classic feature-based methods still outperform CNN-based methods published to date in terms of accuracies.
One key ingredient for the success of graph-based SLAM is the back-end optimization. The back-end builds the pose graph, in which two pose nodes share an edge if an odometry measurement is available between them, while a landmark and a robot-pose node share an edge if the landmark was observed from the corresponding robot pose. In pose graph optimization, the variables to be estimated are poses sampled along the trajectory of the robot, and each factor imposes a constraint on a pair of poses. Modern SLAM solvers exploit the sparse nature of the underlying factor graph and apply iterative linearization and optimization methods (e.g. nonlinear least squares via the Gauss-Newton or Levenberg-Marquardt algorithm). Several such solvers achieve excellent performance, for example, g2o (Kümmerle et al., 2011), TSAM (Dellaert, 2012), Ceres, iSAM (Kaess et al., 2012), SLAM++ (Salas-Moreno et al., 2013), and recently (Bowman et al., 2017) for optimization with semantic data association. The SLAM back-end offers a natural defense against data association and perceptual aliasing errors from the front-end, where similarly looking scenes, corresponding to distinct locations in the environment, would deceive place recognition. However, they depend heavily on linearization of the sensing and motion models, and require good initial guesses. Current systems can be easily induced to fail when either the motion of the robot or the environment are too challenging (e.g. fast robot dynamics or highly dynamic environments) (Cadena et al., 2016).

3 Data

The experiments in this project will use the KITTI odometry dataset. KITTI is a collection of datasets collected with a focus on applications that would be relevant to an autonomous driving platform, such as depth estimation, odometry, object tracking, semantic segmentation and more. There are several different datasets within KITTI for each of these applications, and in this project report we will focus on the odometry dataset.
The odometry dataset is a collection of 22 sequences collected from an autonomous driving recording platform with several advanced sensors to collect data streams such as RGB video, depth, LIDAR and accurately estimate egomotion. The particular sensor used for egomotion was a state-of-the-art OXTS RT 3003 localization system. This system combines GPS (Global Positioning System), GLONASS (Global Navigation Satellite System), an IMU (inertial measurement unit) and RTK (Real Time Kinematic) correction signals. The OXTS RT 3003 enables centimeter-level accuracy (open sky localization errors < 5 cm) and provides the ground truth for visual odometry methods

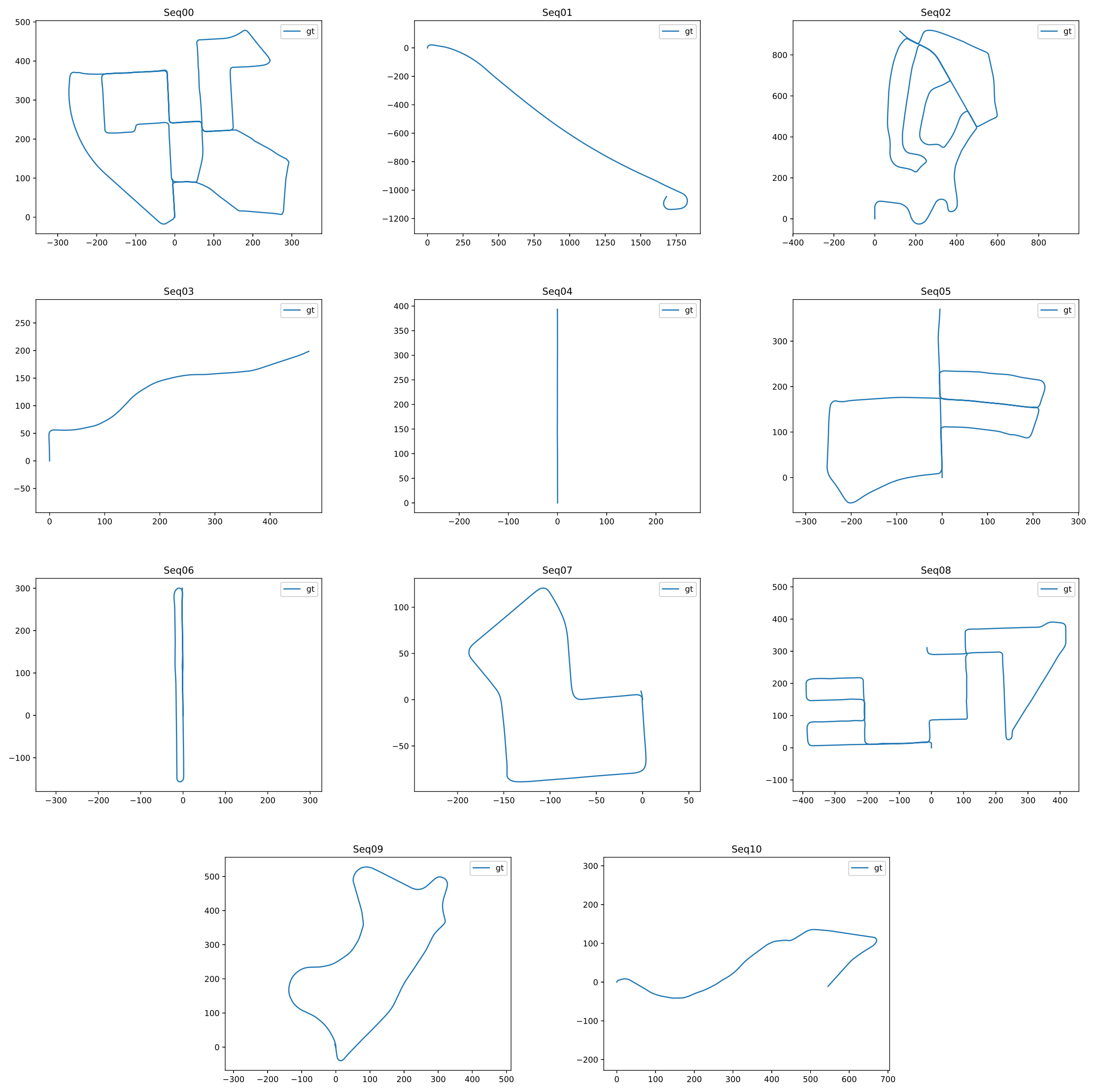

Figure 1: Ground-truth poses for all 11 labelled sequences from a bird's eye view.

to aim for. These sensors are prohibitively expensive for widespread consumer applications (costing on the same order of magnitude as a car itself) and this provides one reason why visual odometry is still an active area of research despite the impressive accuracy of this sensor. The images used in the dataset are collected from 4 PointGrey Flea2 video cameras (2 color cameras + 2 grayscale cameras) which have 10Hz sampling rate, 1392x512 pixel resolution, and $90° \times 35°$ field of view. The camera system is composed of stereo mounted cameras (1 RGB/Grayscale camera on each side) with 54 cm of distance between left/right camera, and 6 cm between RGB/Grayscale cameras on the same side. The ground-truth produced by the localization system is projected into the

coordinate system of the left camera.
The odometry dataset is comprised of 22 sequences of a car driving outdoors in a variety of scenarios, including traversing cities, roads and highways. There are around 41,000 images collected in total, measured over a distance travelled of around 39.2 km. There is only access to the ground-truth labels of 11 of the 22 sequences, as the rest are used as testing data for a benchmark leaderboard on this task. Top-down views of the ground-truths of the 11 sequences provided in the dataset are shown in Figure 1. In this paper, we only do the comparison between the 11 sequences on which ground-truth label data is publicly available. We do the train/test split of sequences [0, 2, 8, 9] for training and sequences [01, 03, 04, 05, 06, 07, 10] for testing.
For the purposes of experiments in this paper, the data is stored as a set of monocular RGB images (left camera) and ground-truth poses stored as the global transformation matrices that represent the position and orientation that the camera was in when the frame was captured. We preprocess every image into the resolution 1280×384 before featurization. The dataset presents several challenges for sequence-based deep learning models, namely the very low data count (11 sequences in total). Owing to this data sparsity we utilize external datasets to pretrain our image featurization models, the details of which are presented later in Section 5.

4 Methods

In this section, we will detail the high-level procedure we will use to estimate the camera poses. For the internal representation in the code, we choose to represent the transformations by splitting them up into a position+quaternion representation. The quaternion is a 4-dimensional structure that can represent 3D rotations very efficiently. We choose this representation since it can be composed with fewer issues than transformation matrices. It also provides an efficient parameterization of the output space since it is more restricted than the set of all transformation matrices.
Once the ground-truth transformations are converted to position+quaternion parameterization, we additionally transform them into relative frame. This means that each position+quaternion transformation in the sequence represents the delta transformation from the previous frame, i.e. the position value at time t is the movement since time $t-1$ and similarly for the rotation. Relative frame allows more manageable outputs for the regression models since the scales won't change much from position to position (whereas in global frame, the scale is changing rapidly as you proceed through the sequence). This aids the optimization especially for deep neural networks as they tend to only learn a limited output range.
Similarly to previous work (Wang et al., 2017) (Clark et al., 2017), we treat the visual odometry task as a regression task. That is, the network will take as input the image sequence and produce for each time step a transformation that will represent the camera's relative movement from the previous time step. This egomotion estimate can then be integrated over time to produce the final camera positions in global frame (with the initial pose set arbitrarily to the origin coordinates).
The performance of the deep networks was found to depend on the particular parameterization of the output space. We detail several different parameterizations below, and reference work that has previously used these parameterizations in localization and odometry applications.
Position+Quaternion: This parameterization is the one we previously described, where the

camera's transformation matrix is represented as a position vector plus a quaternion to represent orientation. This parameterization is often simple to deal with but comes with a disadvantage that the quaternion requires a normalization to reduce an extra degree of freedom. This normalization comes with the cost that it can often cause optimization issues when used for regression.
Position+Log Quaternion: This is an alternative parameterization of the basic position+quaternion where the quaternion is represented in a form where it does not have the normalization degree of freedom. This enables the optimization to avoid the difficult normalization step and regress estimated/ground-truth log quaternion values directly. In particular, a quaternion $q = (u, \mathbf{v})$ can be transformed to a log-quaternion using the following process:

$$\log q = \begin{cases} \frac{\mathbf{v}}{\|\mathbf{v}\|}\cos^{-1}(u), & if \|\mathbf{v}\| \neq 0 \\ \mathbf{0}, & else \end{cases}$$

The loq quaternion $\mathbf{w} = \log \mathbf{q}$ can be transformed back into quaternion form using:

$$\exp w = \left(\cos\|\mathbf{w}\|, \frac{\mathbf{w}}{\|\mathbf{w}\|}\sin\|\mathbf{w}\|\right)$$

The log-quaternion parameterization was found to perform better in a localization task (Brahmbhatt et al., 2017). In that paper, the quaternion was compared directly with the log-quaternion and some significant improvement was observed.
Position+Euler Angles: This parameterization represents rotation using euler angles, which are far simpler to visualize compared to quaternions since they are simply the angles of rotation from the elementary axes. This parameterization comes at the cost of singularity once the angles cross a certain point. But since we are only dealing with relative transformations and the angular velocies observed in this dataset are relatively small, we can directly use euler angles as the output of our model without needing extra code to handle the singularity. When we require the model's estimate in global frame, we simply convert the euler angles to quaternion form and then process it as we would with the other parameterizations.
Once we have the output of our model and the ground-truth in the same form, we use an L1-based regression loss to train the model. We chose L1 due to the fact that the differences between output and label are often far smaller than 1 in magnitude.

5 Featurization Models

5.1 Image Featurization

As explained in the first section, KITTI presents several challenges for deep learning algorithms beyond simply visual odometry, the most important of which is the data sparsity. In total, there is a total of 11 labeled sequences, a very low amount of training data for any learning method which trains on the entire visual sequence. To partially account for the data sparsity problem, we are required to utilize external data sources to pretrain portions of our model. The main use of external data will be within the featurization model, which will transform temporally-contiguous image pairs into a feature which presents high-level information on the relative motion between frames. These featurization models will be pretrained using external data other than the one we use for visual odometry. There are several models we experiment with to do this first-step featurization, and they are detailed in the subsections below.

5.2 FlowNet2-KITTI

The first featurization model we consider is called FlowNet2-KITTI, which is a pretrained FlowNet2 model that was fine-tuned on KITTI specific data. The FlowNet2 architecture was mainly designed to perform optical flow estimation, a highly related problem to visual odometry where movement of pixels in a pair of images is estimated (Ilg et al., 2017). Traditional methods to do optical flow relied on optimization procedures to solve the flow as data arrived, but this presented several shortcomings such as extensive computational requirements and a difficulty to obtain accurate results in real-time. The FlowNet architecture was one of the first to explore learning optical flow using a deep convolutional network. The network was trained on the ground-truth optical flow of several real and synthetic datasets.

FlowNet consists of a convolution phase and a deconvolution phase (Dosovitskiy et al., 2015). The convolution phase decreased the size of the image while increasing the feature channels. The deconvolution phase followed a refinement process. It started with the smallest feature map from the convolution phase and used that to predict a coarse estimate of the flow. The flow features of the highest layer were deconvolved so that the spatial dimensions grew, and stacked with an upsampled version of the coarse estimate of the flow and a skip connection from the previous convolution layer with the same size as the deconvolved features. Another coarse estimate of the flow was predicted using this as input. The process was repeated until the deconvolutions reached the same size as the original network image input, providing a now refined estimate of the flow for each pixel.

FlowNet itself had several architectural variants, with varying complexity of the internal layers. The simpler version, FlowNetSimple, was a more typical deep convolutional network similar to AlexNet, but the specific choices in layer kernel size and stride were chosen to be more suited for optical flow estimation. The more complicated network, FlowNetCorr, included a novel multiplicative layer which was inspired by the cross-correlation operator. This operation was similar to a normal convolutional operator applied to the second image in the image pair, except the filter weights of this operator were obtained using the first image in the pair. This multiplicative operator was shown in the original FlowNet paper to improve results on some datasets, but was later shown in the FlowNet2 paper to work significantly better than FlowNetSimple given a more thorough hyperparameter search.

The FlowNet2 paper extended the work of FlowNet, mainly in the architectural design of the network (Ilg et al., 2017). The main addition was the use of stacking, where several FlowNet variants were stacked on top of each other in a recurrent-network-like architecture, except the FlowNet variants did not share weights. The complete FlowNet2 archiecture had 4 subnetworks and 1 fusion network which produced the final result. It also included 2 streams, one for processing small displacements and one for large displacements.

5.2.1 FlowNet2-KITTI Training Data and Procedure

The training of FlowNet2-KITTI used 3 datasets, the Chairs dataset, the Things3D dataset and the KITTI2012+KITTI2015 optical flow dataset. The FlyingChairs (or Chairs) dataset contains 22,000 image pairs of chairs rendered on random realistic background images obtained from Flickr. These chairs are randomly transformed from image to image, providing an easy way to get the optical flow ground-truth. The Things3D dataset is similar but consists of 3D models from the ShapeNet dataset rendered on top of random static 3D backgrounds. The rendering in this dataset is more realistic and is therefore more challenging to learn. The KITTI2012 and KITTI2015 optical flow dataset was obtained with the same sensor platform as the KITTI visual odometry dataset. The ground-truth flow was obtained with a process of using the LIDAR sensor to detect background and objects, and then for every object finding a 3D model that closely resembles the object and then trying to fit the 3D rigid body motion, pose and scale of the 3D model to the object.
The FlowNet2-KITTI model weights were obtained from the pretrained models at `https://github.com/lmb-freiburg/flownet2`. The FlowNet2-KITTI model was trained with a complicated curriculum in order to maximize performance as shown in the original paper (Ilg et al., 2017). The subnetworks are trained one-by-one, using a Chairs→Things3D curriculum where the model is first trained on the simpler Chairs dataset and then fine-tuned on Things3D. Once the lower subnetworks are trained, higher subnetworks on the stack are added and the process is repeated. Lower subnetworks on the stack are held fixed while the higher subnetworks are optimized. The entire network is fine-tuned to work better on small displacements on the ChairsSDHom, a modification of Chairs where the displacements are much smaller. Finally, The entire network is fined-tuned on the ground-truths provided by the optical flow portion of the KITTI dataset.

5.2.2 FlowNet2-KITTI Feature Extraction Procedure for KITTI

FlowNet2-KITTI motion features were extracted in the following way. First, an 1280x384 RGB image pair was passed through the network to compute the optical flow. We then extracted the highest level flow features from the convolutional pass of each of the 4 subnetworks. This produced a $1024 \times 6 \times 20$ feature map for each subnetwork, which was concatenated channel-wise to produce the final $4096 \times 6 \times 20$ image pair flow features. This featurization was then passed into our model in a sequence to predict the relative motion between frames. To reduce the number of parameters in our model further, we choose to preprocess the final feature map by using pooling. We test two types of pooling, average pooling and max pooling, referred to as **FlowNet2-KITTI Average Pooling** and **FlowNet2-KITTI Max Pooling**, respectively.

5.3 PoseNet

The second featurization model we considered is called PoseNet. PoseNet was a component of the paper Zhou et al. (2017), where it was used as part of an unsupervised learning pipeline for depth and egomotion estimation. Therefore this model was itself trained to do visual odometry, but it did not require labels because of an unsupervised objective. The unsupervised objective is based on the following intuition: given knowledge about a camera's egomotion within a sequence of images and the depth of every pixel in those images, we can gain an unsupervised target by doing view synthesis using this data. In more detail, let the "source" view be the image at time step T and the "target" view be the image at time step $T+1$. Let p_t be the homogenous coordinate of a pixel in the target view, $\hat{D}_t(p_t)$ be the estimated depth of the pixel p_t, and K be the intrinsic matrix of the camera. Then we can obtain the projection of p_t onto the source view p_s using:

$$p_s \sim K\hat{T}_{t\to s}\hat{D}_t(p_t)K^{-1}p_t$$

As the pixels are continuous and don't align to the grid exactly, bilinear filtering is used to continuously interpolate between the discrete values (similarly to what was done in the Spatial Transformer Network). The depth and pose estimation networks are then trained using the objective:

$$L_{vs} = \sum_s \sum_p |I_t(p) - \hat{I}_s(p)|,$$

where $\hat{I}_s(p)$ is the source view projected onto the target view where the pixel-wise equation was given above. There was several additions on top of this base objective in order to improve performance, such as multi-scale depth prediction and a smoothness loss, and an "explainability mask" network which downweights the loss on portions of the images externally undergoing motion not explained by the camera's motion (e.g. a car moving in frame). Some regularization is done to prevent the explainability mask from downweighting the entire image.
The PoseNet architecture is essentially a temporal convolution over images which processes a stack of k images using a 2D deep convolutional network. The architecture we used used a temporal filter size of $k = 5$. The PoseNet predicts a relative transformation from the center image (the image at the center position of the temporal convolution) to every other image (so in our case, there are 4 output transformations). The PoseNet itself is a relatively standard deep convolutional network and is a much simpler architecture than FlowNet. It is made up of 7 stride-2 convolutions+ReLU followed by a 1x1 linear convolution with $6*(k-1)$ output channels, with 6 being the number of parameters to represent 3D position and euler angles for each transformation. This final feature map is then averaged spatially to produce the final estimate of the $6*(k-1)$ transformation values.

5.3.1 PoseNet Training Data and Procedure

The PoseNet was trained as part of the view synthesis pipeline along with the depth prediction network on the "raw" KITTI dataset using the pytorch reimplementation `https://github.com/ClementPinard/SfmLearner-Pytorch`. The raw KITTI dataset is obtained using the same sensor platform as the visual odometry KITTI, but did not include an extensive amount of annotations other than 3D bounding boxes and mostly contained the "raw" sensor data collected. Because the pipeline is trained using unsupervised objectives, the labels are not required and so only the images are used.

5.3.2 PoseNet Feature Extraction Procedure for KITTI

We extract the features from PoseNet by aggregating every 5 images in the sequence and passing it to the pretrained PoseNet model. For boundaries, we pad the images by wrapping and so we clone the first or last image for the number of positions outside the range of the sequence. For extraction from each KITTI sequence, we take the 5 images centered at time t, i.e. $(t-2, t-1, t, t+1, t+2)$ and pass it to the PoseNet to produce the features for time t. The features we extract are the $256 \times 3 \times 10$-dimensional features just before the 1x1 linear convolution output of PoseNet.

6 Sequence Models

The image featurization step converted the high-dimensional sequences of 2000-4000 1280x360 RGB images into a low dimensional sequence of image and motion features obtained from the higher layers of the featurization model. The next step after image featurization will be to learn the model that will transform this sequence of features into camera poses for each time step of the video. We explore a variety of sequence-based deep learning architectures, including a temporal convolution model (and a ResNet variant), a LSTM-based recurrent model, and finally transformer-based attention models. In the section below, we will give specific details on how the architectures are structured.

For all models, we flatten the spatial dimensions of the image features before input to the sequence model. After the sequence models process the image features, we predict from the feature at each timestep the relative transformation from the previous frame using a linear layer.

6.1 Temporal Convolution

The temporal convolution model takes as input the sequence of image features, and then does several passes of 1-dimensional convolutions on the temporal dimension of the input features. The architecture we chose was 3 layers of stride-3, pad-1, convolutions with ReLU activations. Each layer had 1024 dimensions. The effective temporal receptive field was 7 time steps, so the temporal convolution is limited to using short-range information only.

ResNet Variant: We also tried a ResNet variant of the temporal convolution which added the input to the output feature map. We first pass the features through a filter-size=1 linear temporal convolution layer to transform each time step's features into 1024 dimensions. We then do 3 layers of stride-3, pad-1 ResNet layers with ReLU activations. Each layer again had 1024 dimensions. The ResNet update for feature layer $l+1$ and temporal convolution $h(f)$ was the following: $f_{l+1} = max(0, h(f_l)) + f_l$. Similar to the standard temporal convolution model, the effective temporal receptive field was 7 time steps.

6.2 Recurrent Networks

The recurrent network uses an LSTM to process the input sequence of features into LSTM hidden representations. These LSTM representations could potentially have an unlimited temporal receptive field compared to the temporal convolutions, which have a hard limit on the number of time steps they take into consideration. The LSTM used here is a single layer of 1024 hidden units which processes inputs in a past-to-future direction. We also experimented with a bidirectional LSTM, which takes information from both the future and the past in predicting the features at each time step.

6.3 Transformer Networks

Transformer networks are relatively new deep models which rely almost exclusively on the soft-attention operator in order to transmit information between feature positions. While not being restricted to sequential inputs and being applicable even to general graph inputs, they have been used very successfully recently in machine translation tasks (Vaswani et al., 2017). The basic idea of the transformer network is that each feature position will do soft-attention over the other feature positions by (1) computing a query vector, (2) computing the dot-product of the query vector with the key vector of every other feature position, (3) normalizing the dot-products to produce a probability distribution over feature positions and finally (4) averaging the value features of each feature position using the dot-product probability distribution as the weights. To add more bias to the temporal aspect of the inputs, a “positional encoding” is used. This encoding was the same as in the machine translation paper (Vaswani et al., 2017), where 2 fixed feature vectors are appended to the feature of every time step. The appended positional encoding features are equal to:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

Here pos is the time step, i is the index of the feature dimension and d_{model} is the total dimension of the feature. In this work, we set $d_{model} = 1024$. The positional encoding allows an addressing based on relative positions of features, with each feature dimension providing a different scale of relative addressing.
We make a modification on the basic transformer network presented in Vaswani et al. (2017) where instead of having “multi-headed attention” where several attention steps are done in parallel, we do several attention passes in sequence and update the features of each time step using the output of the attention operator. To do the feature update, we use an LSTM update which takes as input the attention output + the feature from the previous time step + the previous LSTM cell state, and produces a new hidden feature and LSTM cell state.

7 Experimental Results

In this section, we present the experimental results for the models tested. In particular, we report Absolute Trajectory Error (ATE) (Sturm et al., 2012) as a test metric as well as RMSE of the global positions since our method takes into account full sequence information. We used the code at `https://github.com/raulmur/evaluate_ate_scale` to compute the ATE alignment. Here we give a brief description of each metric:
Relative Translation/Rotation RMSE: This metric takes the output of the model, reparameterizes it to position+quaternion format, and then takes the RMSE with the ground-truth relative-frame position+quaternion. Therefore this metric is the error in the estimated relative velocity/angular velocity between each image pair in the sequence. Since this metric only looks at relative transformations, it does not account for accumulating drift errors.
Global Translation/Rotation RMSE: This metric takes the output of the model, reparameterizes in position+quaternion format, but further integrates the relative transformations over time

to produce the estimated global transformations of each pose. The RMSE between the estimated global position (quaternion) and the ground-truth global-frame position (quaternion) is then taken to produce the global translation (rotation) metric.

Absolute Trajectory Error (ATE): This is similar to the global RMSE metric, but it includes a preprocessing step described in (Sturm et al., 2012) and `https://github.com/raulmur/evaluate_ate_scale`. This "aligning" preprocessing step seeks to align the estimated sequence with the ground-truth sequence by producing a scale, rotation and translation transformation that will be applied to every pose in the sequence. The aligning is done using the method of Horn (closed-form) which uses SVD to produce the transformation. After aligning, the RMSE is taken between the aligned estimated global position and the ground-truth global-frame position.

7.1 Training Procedure and Hyperparameter Details

We train all models using the Adam optimizer, using a small hyperparameter sweep over learning rates [0.0005, 0.001, 0.0025, 0.005] to coarsely determine a suitable learning rate. We found two tricks to help learning signficantly. The first is to downscale the translation and rotation output of the model by 100 (i.e. multiply the output vector by 0.01). This enabled the network to approximately have the same scale as the ground-truth labels. Without this, the optimization was often unstable and would oscillate around the ground-truth, never truly converging to a fixed value. The second trick is to put more weight on the rotation loss so that rotation errors are more heavily penalized. Even relatively small incorrect rotation estimates at corners could cause large global errors due to the fact that it will unalign the entire sequence after that corner. Therefore we upscale the rotation loss by 100 compared to the translation loss. On a preliminary test, we determined that using euler angles gave the best results compared to quaternion and log-quaternion parameterizations.

7.2 KITTI Results

Model	FlowNet2-KITTI Average Pooling Absolute Trajectory Error							
	Seq01	Seq03	Seq04	Seq05	Seq06	Seq07	Seq10	Avg
TempConv	103	4.68	1.93	17.6	37.5	4.76	5.87	25.1
TempRes	118	7.16	1.35	21.3	45.6	3.55	6.49	29.1
LSTM	46.7	12.8	2.51	43.0	25.9	15.1	7.30	21.9
Transformer (k=1)	85.9	6.62	2.21	19.7	43.8	6.76	8.78	24.8
Transformer (k=2)	32.5	11.0	2.71	28.4	47.5	12.2	16.0	**21.5**
Transformer (k=5)	78.7	9.55	4.46	56.8	48.1	11.4	19.6	32.6

Table 1: Absolute Trajectory Error (ATE) of the deep learning sequence models on the 6 test sequences of the KITTI dataset [01,03,04,05,06,07,10] preprocessed by FlowNet2-KITTI and spatially average pooled.

Model	FlowNet2-KITTI Max Pooling							
	Absolute Trajectory Error							
	Seq01	Seq03	Seq04	Seq05	Seq06	Seq07	Seq10	Avg
TempConv	137	4.26	2.69	17.3	15.0	6.66	8.37	**27.4**
TempRes	199	3.49	3.31	42.2	55.5	11.8	15.2	47.2
LSTM	113	6.39	3.37	38.5	50.3	12.6	15.6	34.3
Transformer (k=1)	68.3	9.44	2.23	38.1	64.0	21.8	21.1	32.1
Transformer (k=2)	65.7	16.4	5.07	78.8	77.0	39.4	46.2	46.9
Transformer (k=5)	31.6	10.4	2.62	82.2	57.7	14.1	22.5	31.6

Table 2: Absolute Trajectory Error (ATE) of the deep learning sequence models on the 6 test sequences of the KITTI dataset [01,03,04,05,06,07,10] preprocessed by FlowNet2-KITTI and spatially max pooled.

Model	PoseNet							
	Absolute Trajectory Error							
	Seq01	Seq03	Seq04	Seq05	Seq06	Seq07	Seq10	Avg
TempConv	45.9	6.18	1.23	46.2	32.1	19.9	11.6	23.3
TempRes	37.8	5.83	1.19	32.4	39.1	14.0	8.49	**19.8**
LSTM	45.8	6.36	2.30	60.8	27.4	25.2	17.2	26.4
Transformer (k=1)	21.4	5.81	1.87	31.4	55.0	20.9	22.9	22.7
Transformer (k=2)	52.6	4.64	1.24	55.3	54.2	12.8	13.4	27.7
Transformer (k=5)	125	4.75	1.14	60.5	55.5	12.1	27.0	40.8

Table 3: Absolute Trajectory Error (ATE) of the deep learning sequence models on the 6 test sequences of the KITTI dataset [01,03,04,05,06,07,10] preprocessed by PoseNet.

7.3 KITTI Discussion

We present the KITTI ATE results in Tables 1 2 and 3 for the 3 types of image featurization methods: FlowNet2-KITTI Average Pooling, FlowNet2-KITTI Max Pooling and PoseNet. From the average ATE on all test sequences, we can see that the FlowNet2-KITTI max pooling performs worse than either the FlowNet2-KITTI average pooling or PoseNet featurization methods. Interestingly on certain sequences with the model fixed, we can observe that the ATE with FlowNet2-KITTI features is sometimes much lower than with PoseNet features (e.g. Transformer (k=1) for Seq07+Seq10) while with the same model on a different sequence sometimes FlowNet2-KITTI features have much higher ATE than the model with PoseNet features (e.g. Transformer (k=1) for Seq01). This might be due to FlowNet2-KITTI and PoseNet features working better in different scales of egomotion and suggests that future results can potentially be improved by concatenating these different features.

Over all models and featurizations, the best average ATE result was obtained with the temporal residual network using the PoseNet featurization. In general, it seems that the temporal-convolution networks (TempConv+TempRes) worked best with the PoseNet, obtaining sometimes significantly

lower ATE (e.g. Seq01). The LSTM network often had results on par with the temporal convolution models. The transformer network seems to have results on par or better than the LSTM network, even though the transformer doesn't have as strong a sequential bias as the recurrent network. It also seemed like the transformer network with 1 or 2 iterations did better than having 5 iterations. In more detail, we can observe from the ATE results that the sequences 01, 05 and 06 are far more difficult than the other test sequences as all the models suffer from significantly higher ATE on those sequences. In particular, most models had difficulty with Seq01 which is the test sequence with the largest translational scale (as can be observed from Fig. 1). On Seq01, the transformer networks for at least one setting of k seem to beat the other models. Interestingly, by far the lowest ATE on this sequence was the transformer network (k=1) with PoseNet featurization.
We additionally report the Relative RMSE results in Tables 4 5and 6 for the 3 types of image featurization methods: FlowNet2-KITTI Average Pooling, FlowNet2-KITTI Max Pooling and PoseNet, respectively. We can see that the temporal convolution and residual networks consistently get lower average RMSE over all test sequences than other models despite often doing worse than other models at ATE. This suggests that the training loss we use isn't representative of the final metric we wish to improve. Preliminary experiments using global loss (ATE without aligning) showed significant difficulty at optimization due to the widely varying scales of the output and loss, and so only results involving relative-loss-trained networks are reported here.

8 Conclusion

In this report we tested and compared the effectiveness of several sequence-based deep learning methods. These models included temporal convolution and residual networks, LSTM recurrent networks, attention-based transformer networks. To deal with the very low amount of data we have available (3 training sequences total), we used image featurization models including FlowNet2-KITTI and PoseNet. Our results indicate that most models achieve similar average global ATE on the test sequences, with temporal residual networks doing slightly better. In conclusion, the results indicate that on KITTI deep learning models are mostly using local information, with models with hard upper limits on the temporal receptive field (TempConv/TempRes) doing just as well or better than models which can communicate over unlimited time horizons (given enough capacity). The current lack of a large dataset for visual odometry presents a significant challenge for learning-based visual odometry models, and better results could potentially be obtained with more data by enabling the end-to-end training of the featurization and sequence models, and providing more examples for the model to learn how to utilize temporally-distant information (e.g. to do loop-closure-like corrections).

References

Sean L Bowman, Nikolay Atanasov, Kostas Daniilidis, and George J Pappas. Probabilistic data association for semantic slam. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 1722–1729. IEEE, 2017.

Eric Brachmann, Alexander Krull, Sebastian Nowozin, Jamie Shotton, Frank Michel, Stefan Gumhold, and

Carsten Rother. Dsac-differentiable ransac for camera localization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 3, 2017.

Samarth Brahmbhatt, Jinwei Gu, Kihwan Kim, James Hays, and Jan Kautz. Mapnet: Geometry-aware learning of maps for camera localization. *arXiv preprint arXiv:1712.03342*, 2017.

Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

Devendra Singh Chaplot, Emilio Parisotto, and Ruslan Salakhutdinov. Active neural localization. In *International Conference on Learning Representations (ICLR)*, 2018.

Ronald Clark, Sen Wang, Hongkai Wen, Andrew Markham, and Niki Trigoni. Vinet: Visual-inertial odometry as a sequence-to-sequence learning problem. In *AAAI*, pp. 3995–4001, 2017.

Frank Dellaert. Factor graphs and gtsam: A hands-on introduction. Technical report, Georgia Institute of Technology, 2012.

Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Deep image homography estimation. *arXiv preprint arXiv:1606.03798*, 2016.

Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2758–2766, 2015.

Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pp. 834–849. Springer, 2014.

Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In *Readings in computer vision*, pp. 726–740. Elsevier, 1987.

Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 15–22. IEEE, 2014.

Christian Forster, Zichao Zhang, Michael Gassner, Manuel Werlberger, and Davide Scaramuzza. Svo: Semidirect visual odometry for monocular and multicamera systems. *IEEE Transactions on Robotics*, 33(2): 249–265, 2017.

Dorian Gálvez-López and Juan D Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, 2012.

Tuomas Haarnoja, Anurag Ajay, Sergey Levine, and Pieter Abbeel. Backprop kf: Learning discriminative deterministic state estimators. In *Advances in Neural Information Processing Systems*, pp. 4376–4384, 2016.

Joel A Hesch, Dimitrios G Kottas, Sean L Bowman, and Stergios I Roumeliotis. Camera-imu-based localization: Observability analysis and consistency improvement. *The International Journal of Robotics Research*, 33(1):182–201, 2014.

Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pp. 1647–1655. IEEE, 2017.

Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, 31(2):216–235, 2012.

Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pp. 2938–2946, 2015.

Dimitrios G Kottas, Joel A Hesch, Sean L Bowman, and Stergios I Roumeliotis. On the consistency of vision-aided inertial navigation. In *Experimental Robotics*, pp. 303–317. Springer, 2013.

Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g 2 o: A general framework for graph optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 3607–3613. IEEE, 2011.

Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34(3):314–334, 2015.

Fayao Liu, Chunhua Shen, and Guosheng Lin. Deep convolutional neural fields for depth estimation from a single image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5162–5170, 2015.

John McCormac, Ankur Handa, Andrew Davison, and Stefan Leutenegger. Semanticfusion: Dense 3d semantic mapping with convolutional neural networks. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 4628–4635. IEEE, 2017.

Iaroslav Melekhov, Juha Ylioinas, Juho Kannala, and Esa Rahtu. Relative camera pose estimation using convolutional neural networks. In *International Conference on Advanced Concepts for Intelligent Vision Systems*, pp. 675–687. Springer, 2017.

Anastasios I Mourikis and Stergios I Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. In *Robotics and automation, 2007 IEEE international conference on*, pp. 3565–3572. IEEE, 2007.

Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pp. 127–136. IEEE, 2011.

Emilio Parisotto, Devendra Singh Chaplot, Jian Zhang, and Ruslan Salakhutdinov. Global pose estimation with an attention-based recurrent network. *CoRR*, abs/1802.06857, 2018. URL `https://arxiv.org/abs/1802.06857`.

Renato F Salas-Moreno, Richard A Newcombe, Hauke Strasdat, Paul HJ Kelly, and Andrew J Davison. Slam++: Simultaneous localisation and mapping at the level of objects. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1352–1359, 2013.

Torsten Sattler, Bastian Leibe, and Leif Kobbelt. Efficient & effective prioritized matching for large-scale image-based localization. *IEEE transactions on pattern analysis and machine intelligence*, 2016.

Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 573–580. IEEE, 2012.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 6000–6010, 2017.

Florian Walch, Caner Hazirbas, Laura Leal-Taixé, Torsten Sattler, Sebastian Hilsenbeck, and Daniel Cremers. Image-based localization with spatial lstms. *CoRR*, abs/1611.07890, 2016. URL `http://arxiv.org/abs/1611.07890`.

Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 2043–2050. IEEE, 2017.

Tobias Weyand, Ilya Kostrikov, and James Philbin. Planet-photo geolocation with convolutional neural networks. In *European Conference on Computer Vision*, pp. 37–55. Springer, 2016.

Thomas Whelan, Renato F Salas-Moreno, Ben Glocker, Andrew J Davison, and Stefan Leutenegger. Elasticfusion: Real-time dense slam and light source estimation. *The International Journal of Robotics Research*, 35(14):1697–1716, 2016.

Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *CVPR*, volume 2, pp. 7, 2017.

9 Relative Error Results

Model	FlowNet2-KITTI Average Pooling Relative RMSE $\begin{pmatrix}\text{Translate}\\\text{Rotate}\end{pmatrix}$							
	Seq01	Seq03	Seq04	Seq05	Seq06	Seq07	Seq10	Avg
TempConv	7.06e-1	3.75e-2	1.26e-1	3.94e-2	5.39e-2	5.01e-2	6.96e-2	1.55e-1
	2.35e-3	2.50e-3	1.51e-3	2.26e-3	2.47e-3	2.39e-3	3.12e-3	**2.37e-3**
TempRes	5.45e-1	2.85e-2	6.23e-2	3.06e-2	3.60e-2	3.18e-2	5.15e-2	**1.12e-1**
	2.57e-3	2.60e-3	1.48e-3	2.27e-3	3.10e-3	2.71e-3	3.38e-3	2.59e-3
LSTM	6.09e-1	3.57e-2	1.07e-1	4.27e-2	6.67e-2	4.98e-2	7.82e-2	1.41e-1
	2.67e-3	2.94e-3	1.73e-3	2.48e-3	2.64e-3	2.81e-3	3.30e-3	2.65e-3
Transformer (k=1)	6.41e-1	6.60e-2	1.02e-1	6.65e-2	6.38e-2	1.09e-1	1.08e-1	1.65e-1
	3.16e-3	2.82e-3	1.79e-3	2.59e-3	2.62e-3	2.82e-3	3.50e-3	2.76e-3
Transformer (k=2)	8.29e-1	1.56e-1	3.06e-1	1.42e-1	1.87e-1	2.06e-1	1.91e-1	2.88e-1
	2.46e-3	2.65e-3	1.48e-3	2.27e-3	3.32e-3	2.42e-3	3.32e-3	2.56e-3
Transformer (k=5)	7.34e-1	9.52e-2	2.00e-1	1.32e-1	1.27e-1	1.34e-1	1.39e-1	2.23e-1
	3.24e-3	3.02e-3	2.26e-3	2.75e-3	3.89e-3	3.16e-3	3.67e-3	3.14e-3

Table 4: Evaluation of the deep learning sequence models on the 6 test sequences of the KITTI dataset [01,03,04,05,06,07,10].

Model	FlowNet2-KITTI Max Pooling Relative RMSE $\begin{pmatrix}\text{Translate}\\\text{Rotate}\end{pmatrix}$							
	Seq01	Seq03	Seq04	Seq05	Seq06	Seq07	Seq10	Avg
TempConv	6.39e-1	3.81e-2	1.24e-1	4.76e-2	5.34e-2	6.68e-2	7.24e-2	1.49e-1
	2.97e-3	2.71e-3	1.66e-3	2.54e-3	2.82e-3	2.82e-3	3.23e-3	**2.68e-3**
TempRes	6.08e-1	3.93e-2	1.10e-1	4.40e-2	5.25e-2	5.32e-2	7.58e-2	**1.40e-1**
	3.74e-3	2.75e-3	2.12e-3	2.69e-3	4.21e-3	3.35e-3	3.63e-3	3.21e-3
LSTM	7.25e-1	1.15e-1	1.76e-1	1.03e-1	1.04e-1	1.53e-1	1.47e-1	2.18e-1
	3.66e-3	3.27e-3	1.96e-3	3.20e-3	4.07e-3	4.24e-3	3.89e-3	3.47e-3
Transformer (k=1)	6.97e-1	1.95e-1	1.64e-1	1.57e-1	1.51e-1	2.20e-1	2.24e-1	2.58e-1
	3.27e-3	3.45e-3	1.51e-3	3.45e-3	5.39e-3	4.21e-3	4.11e-3	3.63e-3
Transformer (k=2)	8.61e-1	1.51e-1	3.65e-1	1.67e-1	2.35e-1	2.16e-1	1.95e-1	3.13e-1
	4.29e-3	4.75e-3	2.13e-3	6.11e-3	9.71e-3	8.67e-3	6.56e-3	6.03e-3
Transformer (k=5)	7.21e-1	2.40e-1	2.03e-1	1.62e-1	1.47e-1	2.35e-1	2.25e-1	2.76e-1
	3.15e-3	3.22e-3	1.64e-3	3.76e-3	4.99e-3	4.09e-3	4.02e-3	3.55e-3

Table 5: Evaluation of the deep learning sequence models on the 6 test sequences of the KITTI dataset [01,03,04,05,06,07,10].

Model	PoseNet Relative RMSE $\begin{pmatrix}\text{Translate}\\\text{Rotate}\end{pmatrix}$							
	Seq01	Seq03	Seq04	Seq05	Seq06	Seq07	Seq10	Avg
TempConv	8.17e-1	7.24e-2	1.93e-1	6.92e-2	1.10e-1	8.39e-2	9.58e-2	**2.06e-1**
	3.06e-3	2.49e-3	1.43e-3	2.22e-3	2.28e-3	2.79e-3	3.38e-3	**2.52e-3**
TempRes	8.19e-1	7.67e-2	2.01e-1	7.74e-2	1.09e-1	1.01e-1	1.11e-1	2.14e-1
	3.82e-3	2.83e-3	1.58e-3	2.65e-3	2.87e-3	3.28e-3	3.66e-3	2.96e-3
LSTM	8.17e-1	1.12e-1	1.69e-1	1.16e-1	1.05e-1	1.59e-1	1.59e-1	2.34e-1
	4.01e-3	2.67e-3	1.63e-3	2.68e-3	2.79e-3	3.17e-3	4.30e-3	3.04e-3
Transformer (k=1)	6.83e-1	1.94e-1	1.49e-1	1.50e-1	1.24e-1	1.91e-1	2.11e-1	2.43e-1
	3.57e-3	2.66e-3	1.53e-3	2.58e-3	3.54e-3	3.08e-3	4.03e-3	3.00e-3
Transformer (k=2)	8.14e-1	7.40e-2	2.00e-1	7.95e-2	1.20e-1	8.50e-2	9.91e-2	2.10e-1
	4.09e-3	2.78e-3	1.76e-3	2.67e-3	3.52e-3	3.08e-3	4.05e-3	3.14e-3
Transformer (k=5)	7.70e-1	9.82e-2	1.35e-1	8.92e-2	1.01e-1	1.31e-1	1.45e-1	2.10e-1
	4.39e-3	3.01e-3	1.81e-3	2.87e-3	3.78e-3	3.40e-3	4.10e-3	3.34e-3

Table 6: Evaluation of the deep learning sequence models on the 6 test sequences of the KITTI dataset [01,03,04,05,06,07,10].