A Fused Convolutional and Recurrent Neural Network for Visual-Inertial Odometry

Justin Gorgen
A53096607
jgorgen@ucsd.edu

Haifeng Huang
A53208823
hah086@ucsd.edu

Hao-en Sung
A53204772
h3sung@ucsd.edu

Yen-ting Chen
A53214417
yec068@ucsd.edu

Abstract

The results of a recent paper [1] on a neural network architecture for Visual-Inertial Odometry (VIO) are replicated to verify the performance of the algorithm. The algorithm achieves an error rate of 1% on KITTI 500m dataset, and is able to tolerate offset in rotation, translation, and time between the inertial measurement unit (IMU) and camera. In our project, we re-implement the model in Python with Keras and Tensorflow and perform experiments on KITTI dataset, which is available on GitHub (link). We are going to compare to reference implementation and state of the art by plotting model losses and trajectory.

1. Introduction

The problem of navigation is ancient. Ancient humans have developed absolute positioning methodologies such as celestial navigation or trade-wind-based localization [2] to navigate between Pacific Islands thousands of miles apart. There is a distinction between the kind of absolute navigation performed when finding correct island in the middle of the pacific, and relative navigation, or odometry, which is performed by robots exploring a room [3]. Systems that combine cameras and inertial-measurement-units (IMU) can be used to estimate absolute positioning or relative positioning [4, 5]. This paper will use an IMU and a monocular camera system to calculate relative positioning, integrating small changes in movement to estimate the total change in position over time.

2. A Background on Odometry

The most concise definition of odometry comes from [3], "Odometry is the use of data from motion sensors to estimate change in position over time." For ground vehicles, the most common method of calculating odometry is a wheel-encoder, which calculates turns of a vehicles wheels to estimate distance traveled. For air and sea vehicles, wheel encoders cannot be used, and instead these types of vehicles often use a combination of IMUs and pitot tubes that measure the local speed relative to the medium. An IMU is an electromechanical device that can measure rotations and accelerations in three-dimensions.

Visual-Inertial Odometry is the process of calculating odometry using an IMU and a camera. While not strictly a motion sensor, a single camera can be used to estimate change in position using aspects from multiple view geometry [6], a process referred to as "structure-from-motion." Both IMUs and cameras alone have several weaknesses as odometry sensors, but as demonstrated by more than a decade of research, fused measurements from IMUs and cameras can provide estimates of odometry more precise than wheel encoders or pitot-tubes.

In the following sections, we are going to respectively introduce inertial measurement units, visual odometry, and the joint usage of them.

2.1. Inertial Measurement Units

Inertial Measurement Units come in many shapes and sizes, with sensing technologies ranging from spinning-wheel gyroscopes to million-dollar arrays of ring-laser gyroscopes to hundred-dollar micro-electromechanical systems (MEMS) devices [7]. The basic principle of using IMUs for odometry is to carefully sum changes in attitude and velocity and attitude over time to estimate the current velocity and attitude, and then integrate the current velocity to estimate the current change in position. A common method of calculating IMU odometry, from Paul Groves's *Principles of GNSS, Inertial and Multisensor Integrated Navigation Systems* [7], is to use a 15 state Kalman filter, with 3 states each for position, velocity, attitude, accelerometer bias errors, and gyroscope bias errors.

The problem with integrating IMUs alone for odometry is that real accelerometers and gyroscopes have time-varying, non-zero-mean errors, called biases. Small bias errors in accelerometer and gyroscope measurements are integrated twice to calculate change in position, and thus small constant errors in acceleration become quadratic errors in position, as shown in Figure 1.

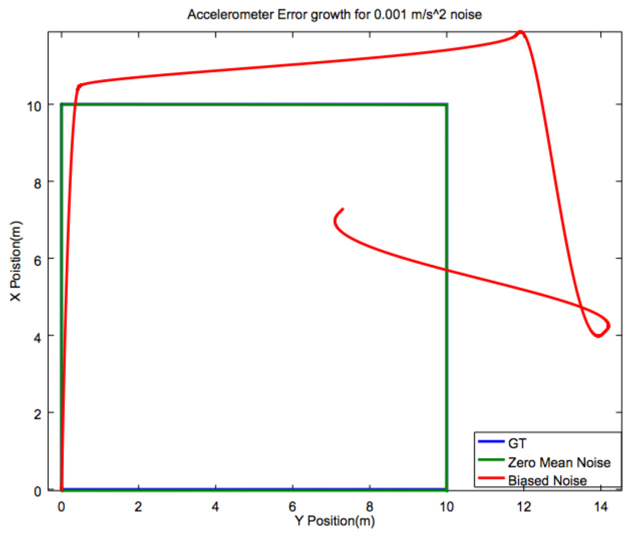


Figure 1. A simple simulation showing the effect of 0.001 m/s^2 Gaussian errors on acceleration for a 120-second long simulation. Zero mean errors have little effect on the overall shape of the trajectory, but biased accelerometer errors even as small as 0.001 m/s^2 grow quickly.

2.2. Visual Odometry

Visual odometry (VO) is the art of calculating changes in position using only optical sensors, typically using cameras in the infrared to ultra-violet range. While the celestial navigation used by ancient mariners is a form of visual odometry, modern interpretations involve using monocular or stereo vision sensors to estimate the motion of the camera or the vehicle to which the camera is attached. This is distinct from the tracking problem, where a moving object's motion is estimated from a static, off-platform camera. Research into monocular VO is limited, due to the inherent ambiguity of scale with a single camera. Meanwhile, stereo VO is much more mature, and NASA/JPL use stereo VO as a "critical vehicle safety system" on the Spirit and Opportunity Mars rovers [8]. Although critical safety system, during their missions Spirit and Opportunity did not use VO as a primary navigation aid, using it for only 14% of their trajectory. Due to issues with inclement weather and camera-recalibration, and the high reliability of other sensors, VO was mainly used to detect wheel slip. This shows there is room for other sensors to aid and improve visual odometry.

Visual Odometry is very closely related to the computer-vision problem called structure from motion (SFM) and the control/estimation problem called Simultaneous Localization and Mapping (SLAM). However, both SFM and SLAM create and update maps of the environment, while VO has no such requirement of building a map of the environment. Thus, some approaches to solve the VO problem can be solved by a good SLAM or SFM algorithm, but at a higher computational expense. Recent research into using deep neural networks to solve the VO problem are an attempt to create a real-time, causal VO solution that does not have the extra burden of creating an explicit map of the environment.

2.3. Visual-Inertial Odometry

Visual-inertial odometry is traditionally accomplished using a sensor fusion algorithm, such as an Extended Kalman Filter (EKF) [9]. The best algorithm such as [9], perform on the order of 0.2% error per distance traveled, even using only a monocular camera. The traditional approach to visual-inertial odometry involves:

- Identifying salient features in each frame of a video sequence using a hand-tuned algorithm such as SIFT
- Matching features between each frame (ostensibly requires N^2 comparisons for matching N features, but various heuristics have been used to greatly speed this process)
- Storing and managing a database of features
- Calculating a camera pose from the visible features
- Calculating a correction for the feature locations to improve future position estimates

Each of these processes require hand-tuning based on the expected motion profile of the camera system. Additionally, filtering-based algorithms such as [9], require a calibration maneuver in order to observe and calibrate the external calibration parameters of the camera, i.e. the lever arm and rotation between the IMU and the camera. Nevertheless, the performance of the hand-tuned approach is very good, the only downside is the requirement of at least laptop-level CPU and GPU hardware to produce a real-time measurement as well as the labor needed to tune the model.

A major breakthrough in machine-learning based VIO was published in January of 2017 [1]. This recurrent-neural-network based approach presented errors of 1% on the KITTI dataset, which is on the same order of magnitude of error performance as hand-tuned VIO algorithms. The approach in [1] can calibrate for offsets in time and space between the camera and IMU data streams, and this approach is the method replicated in this paper.

2.4. Long short-Term Memory Unit

RNN has the disadvantage that using standard training techniques, they are unable to learn to store and operate on long-term trends in the input and thus do not provide much benefit over standard feed-forward networks. For this reason, the Long Short-Term Memory (LSTM) architecture was introduced to allow RNN to learn longer-term trends (Hochreiter and Schmidhuber 1997) [10].

The LSTM unit has three gates: input gate, output gate and forget gate, all of which are logistic functions whose values are ranging from 0 to 1. Input gate will determine the extent to which the input information will be taken into the hidden units, output gate will determine the extent to which the state of the hidden units will be outputs, and the forget gate will determine the extent to which the hidden units will remember about former information. The values of these three gates are updated while training by the LSTM unit itself. This capability of selectively storing and forgetting memories make the LSTM units suitable for problems closely related to time issues, like the images and IMU data we have gotten since these information are continuous in time rather than independent with each other.

2.5. Convolutional Neural Network

Convolutional neural network is a type of feed-forward artificial neural network, consisting of multiple layers of receptive fields, which are small neuron collections processing part of the image. To be more specific, a set of neurons constructs a kernel and convolution operation of layer inputs and kernels will be done within layers.

While the architectures of CNN and its derivation manifest, the simplest and the key structure is to apply convolution in a "sliding window" fashion, computing a single prediction for each input image patch. This method works well to keep spatial information and features of images.

Convolutional neural network had made a huge success on large-scale image classification by Krizhevsky *et al.* [11]. Following CNN applications include feature extracting [12], segmentation [13, 14] and depth prediction [15].

The previous work also demonstrates that CNN performs well at learning input-output relations between images and labels with enough data. With these properties, Fischer argue that CNN is a good model to predict optical flow, and the result turns out satisfying. [16]

3. Methodology

In this section, we first give a big picture of the whole network structure. Later, we not only introduce each component in the model, including FlowNetCorr, LSTM for inertial unit, core LSTM, and concatenation layer, in different sub-sections respectively, but also our implementation details of them. We also slightly change the optimization procedure, as mentioned in the last sub-section.

3.1. Network Framework

The network the authors used to implement sequence-to-sequence learning approach to visual-inertial odometry, VINet, is shown in Figure 2. The input to the network is monocular RGB images and IMU data which is a 6 dimensional vector containing the x, y, z components of acceleration and angular velocity measured using a gyroscope. The output of the network is a 7 dimensional vector - a 3 dimensional translation and 4 dimensional orientation quaternion - representing the change in pose of the robot from the start of the sequence.

For time step t, they fed in images at time step t and $t+1$ into two parallel convolutional neural networks at one end of the whole network and concatenate the feature maps after three convolutional layers by applying convolutional layers to the concatenated feature maps. At last, the output is flattened as a feature vector of visual information. At the other end of the network, the authors fed in 6 dimensional vector of IMU data into IMU LSTM network and concatenated the output inertial feature vector of the IMU LSTM network with visual vector generated from convolutional neural network to form the core LSTM. The core LSTM not only took visual and inertial vectors and the state of the core LSTM of the last time step as input, but also was fed in the output of the core LSTM of the last time step, which is the prediction of the translation and orientation quaternion. In the case of odometry estimation the availability of the previous output state is particularly important as the output is essentially an accumulation of incremental displacements at each step.

3.2. FlowNetCorr for Optical Flow Initialization

The authors initialized the convolutional neural network using Imagenet at first, but they found that the convergence was slow and the performance is not very well, so they used the FlowNet [16] which is trained to predict optical flow from RGB images to initialize the network up to the *Conv6* layer and removed the layers which produce the high-resolution optical flow output and fed in only a $1024 \times 6 \times 20$ vector which they flattened and concatenated with the feature vector produced by the IMU-LSTM before being fed to the Core LSTM.

3.2.1 Implementation Details

There are two versions of FlowNet: FlowNetSimple and FlowNetCorr, proposed in [16], as shown in Figure 3. FlowNetSimple applies convolution layers in a 'sliding window' fashion, hence it computes a single prediction (e.g. class label) or feature vector for each input image patch. However, in order to get depth information from images of adjacent time stamp, we need to apply convolutional layers to two consecutive images at the same time, in which case FlowNetSimple is not suitable anymore. Thus, the CNN we are going to use is FlowNetCorr. In FlowNetCorr, the model will 1) take two consecutive images as inputs once a time, 2) apply convolution respectively for *Cov1*, *Cov2* and *Cov3*, 3) merge them with a newly-defined correlation layer, and 4) run through *Cov4*, *Cov5* and *Cov6* to get final output for vision information. The correlation layer, is implemented as a set of 441 images produced by pixel-wise

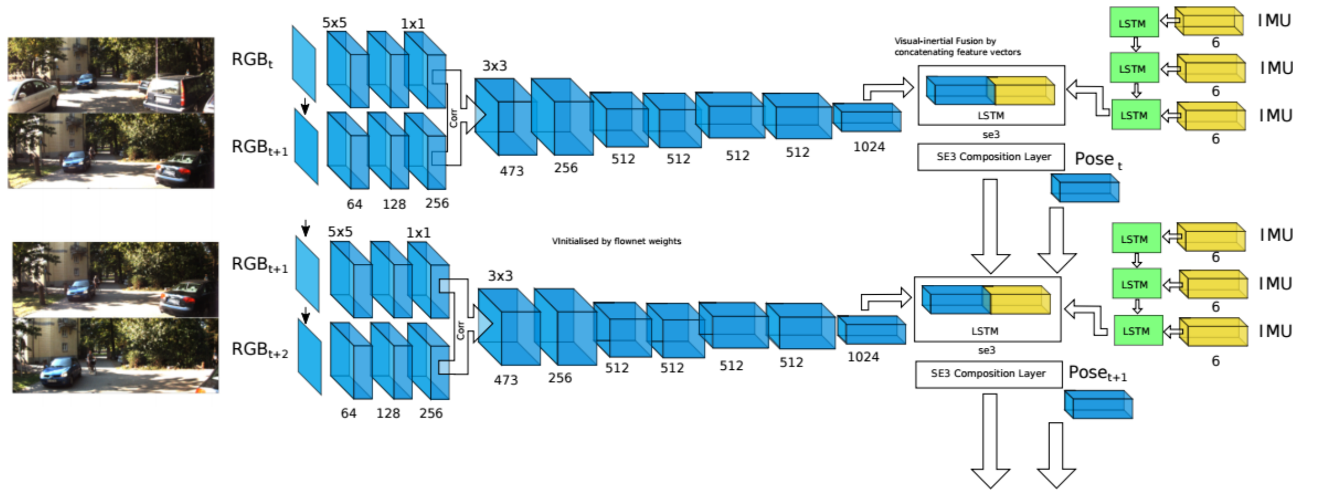


Figure 2. The proposed VINet architecture for visual-inertial odometry. The network consists of a core LSTM processing the pose output at camera-rate and an IMU LSTM processing data at the IMU rate.

dot products between two feature maps of parallel convolutional network. The implementation of correlation was taken from FlowNetCorr, with a maximum displacement of 20 pixels in each of the x and y directions and a stride of 2 [16]. This layer does not have parameters, but takes a lot of time and memory to compute due to the large number of per-pixel multiplications. The purpose of the correlation layer is to extract depth information from adjacent images pairs. Also, the output of the correlation layer concatenated it with another convolutional layer which is convoluted on the feature maps of the first convolutional neural network and called $conv\_redir$ to preserve more information from the image.

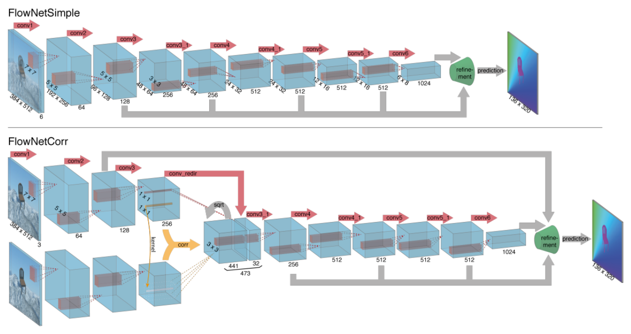


Figure 3. Two Types of FlowNet: FlowNetSimple and FlowNetCorr

3.3. LSTM for Inertial Measurement

Because the IMU data (100 Hz) arrives 10 times faster than the visual data (10 Hz), the authors processed the IMU data using a small LSTM at the IMU rate, and fed that result to the Core-LSTM. Multi-rate LSTM makes the fusion of two images and several IMU data in each iteration of the training and testing possible.

3.3.1 Implementation Details

The way we implemented it is to take two images and 10 IMU data within the time span of the two images, each with six features: angular velocity and acceleration in x, y, z direction at the same time, then fed the 10 IMU data into three layers of LSTM. The original paper did not mention how many layers of IMU-LSTM they used, so originally we just used one layer of IMU-LSTM with 1000 units, but we found out that the model was hard to train due to the excessive computation needed, so we decided to change it to three layers with 15 units each, because more layers with less units is more representative than fewer layers with a lot of units. Also we set *stateful* to true, because we need to make the IMU-LSTM remember the information of past in each sequence separately.

3.4. Core LSTM

Core-LSTM takes flattened convoluted images and output from IMU-LSTM as input, which has $1024 \times \frac{H}{64} \times \frac{W}{64} + D_{IMU}$ input dimension, where H is the image height, W is the image width, and D_{IMU} is the dimension of IMU-LSTM output.

3.4.1 Implementation Details

After flattening the feature vectors of convolutional neural network, we concatenated it with the output of IMU-LSTM and fed that vector into Core-LSTM. Originally we only had one layer of Core-LSTM and we got bad results for losses, later we changed the number of LSTM layers to be two and we found much better performance than before. As we have said before, more layers of LSTM will store more information than just one layer and are more representative in terms of information conveyed by images and IMU data.

3.5. SE(3) Concatenation of Transformations

The pose of a camera relative to an initial starting point is conventionally represented as an element of the special Euclidean group SE(3) of transformations.

$$\mathbf{T} = \left\{ \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{pmatrix} | \mathbf{R} \in SO(3), \mathbf{t} \in R^3 \right\} \tag{1}$$

However, the Lie Algebra se(3) of SE(3), representing the instantaneous transformation, can be described by components which are not subject to orthogonality constraints.

$$\frac{\xi}{dt} = \left\{ \begin{pmatrix} [\omega]_\times & v \\ 0 & 0 \end{pmatrix} | \omega \in so(3), v \in R^3 \right\} \tag{2}$$

Conversion between se(3) and SE(3) is then easily accomplished using the exponential map

$$exp : se(3) \to SE(3) \tag{3}$$

The CNN-RNN thus performs the mapping from the input data to the lie algebra se(3) of the current time step. An exponential map is used to convert these to the special euclidean group SE(3) and right multiply it to the cumulated SE(3) of the last time step to get the cumulated sum of current time step.

Figure 4 is the illustration of the SE(3) composition layer - a parameter-free layer which concatenates transformations between frames on SE(3).

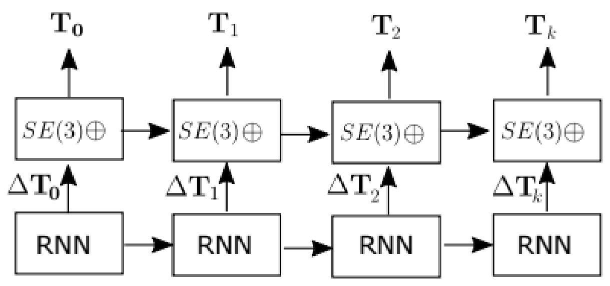


Figure 4. Illustration of the SE(3) composition layer - a parameter-free layer which concatenates transformations between frames on SE(3).

3.6. Optimization Formula

3.6.1 Formula Introduction

According to [1], they use two kinds of loss functions:

- Frame-to-frame pose, i.e. se(3)

$$\mathcal{L}_{se(3)-VINET} = \alpha \sum \|\omega - \hat{\omega}\| + \beta \|v - \hat{v}\| \tag{4}$$

- Full pose, i.e. SE(3)

$$\mathcal{L}_{SE(3)-VINET} = \alpha \sum \|\boldsymbol{q} - \hat{\boldsymbol{q}}\| + \beta \|\mathbf{t} - \hat{\mathbf{t}}\| \tag{5}$$

and they plan to iteratively optimize these two formulas until model convergence. In their experiment, the ratio between λ_2 and λ_1 will vary from 100 to 0.1 for better model performance.

There are multiplicative interactions between α and β in the loss formulas in (4) and (5). Furthermore, there are ambiguities in the definition of quaternion errors $\|\boldsymbol{q} - \hat{\boldsymbol{q}}\|$, and ambiguities in camera position $\hat{\mathbf{t}}$. This ambiguity in $\hat{\mathbf{t}}$ exists because any camera center at distance $\|\hat{\mathbf{t}}\|$ from the origin has an orientation $\hat{R}$ such that $\hat{\mathbf{t}} = \mathbf{t}$. Therefore, we define our own losses $\mathcal{L}_{se(3)}$ and $\mathcal{L}_{SE(3)}$ to constrain the volume of points with identical losses.

- Frame-to-frame pose, i.e. se(3)

$$\mathcal{L}_{se(3)} = \sum \alpha \|\omega - \hat{\omega}\|^2 + \beta \|v - \hat{v}\|^2 \tag{6}$$

- Full pose, i.e. SE(3)

$$\mathcal{L}_{SE(3)} = \sum \alpha \|\Delta\omega\|^2 + \beta \|\Delta\mathbf{t}^w\|^2 \tag{7}$$

The main improvement in (6) is that the effects of α and β on scaling the $\mathcal{L}_{se(3)}$ are now orthogonal. In (7), the attitude error $\Delta\omega$ is defined in terms of the SO3 matrix $\mathbf{R}_w^{w'}$, which is the rotation from the true world frame to the incorrect world frame estimated by the network. Additionally, the position loss in 7 is defined as the difference between the true and estimated coordinates of the camera center in the world coordinate frame. The losses for SE(3) are derived from standard attitude error estimations used in the navigation community [7].

$$\Delta\omega = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{8}$$

$$\begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} = logm\left(\mathbf{R}_w^{w'}\right) \tag{9}$$

$$\mathbf{R}_w^{w'} = \left(\mathbf{R}_{w'}^c\right)^T \mathbf{R}_w^c \tag{10}$$

Similarly, the definition of position error in $\mathcal{L}_{SE(3)}$ is defined as:

$$\Delta \mathbf{t} = \mathbf{t}^{w'} - \mathbf{t}^{w} \tag{11}$$

In (11), $\mathbf{t}^w$, the location of the camera center in world coordinates, are calculated from the inverse of the SE(3) position in (1) using the equation:

$$\mathbf{t}^w = \mathbf{T}^{-1}(1:3,4) \tag{12}$$

$$\mathbf{t}^w = -\mathbf{R}^T\mathbf{t} \tag{13}$$

The SE(3) position error defined in (11) can only be minimized at one location in the world frame where $\mathbf{t}^{w'} = \mathbf{t}^w$.

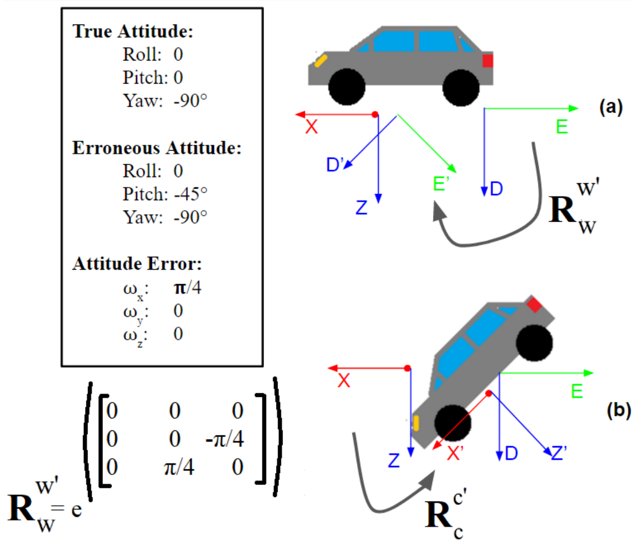


Figure 5. Illustration of the definition of attitude error. (a) A representation of attitude error as a rotation $\mathbf{R}_w^{w'}$ from the true world frame w to an incorrect world frame w'. (b) The same error, depicted as a rotation $R_c^{c'}$, is not used because the true frame c changes with time.

3.6.2 Implementation Details

In our project, we found it difficult to alter loss functions during training with Keras. Suppose λ_1 is the learning rate of SE(3) losses and λ_2 is the learning rate of se(3) losses. We decided to jointly train on se(3) and SE(3) at the same time fixing the ratio between λ_2 and λ_1 at 100 at the first two epochs and ratio to be 1 for the remaining 25 epochs. Additionally, due to memory constraints on the AWS computing platform, the image size for the KITTI data set had to be reduced from 1392×512 to 540×200, and the batch size reduced to 3 which was unspecified in the VINet paper. The important limitation produced by a batch size of 3 is that SE(3) poses are not able propagate further than 3 timesteps (about 0.3 seconds) in a graph. This limits the ability of the implementation to learn correct past SE(3) pose estimates. Furthermore, there is a large disparity between estimates of computation time. VINet claims that forward propagation for a pair of image through the convolutional layers takes 160 milliseconds, while our implementation in Keras and TensorFlow takes 5.2 seconds for a single pair of images. Thus, while the VINet paper claimed to complete 200 epochs of training in 6 hours of wall time, our implementation only trained the model for 27 epochs due to time limits. Also for the first two epochs, we used the weights between angle and position of 100 to more concentrate on the training on the angle and after that we trained on the model with weight ratio to be 1.

4. Experiment Design

4.1. Model Performance

To demonstrate the model performance, we are going to plot the loss Figures for both training and testing data in terms of angular se(3), position se(3), angular SE(3), position SE(3), and overall losses.

Beside that, we also plot the trace Figure for ground truth and our prediction on testing dataset in 3D Figure. Different from what we do in training phase, in prediction phase, we need to keep looping over our prediction at last time step. Thus, the error might accumulate in a short time.

5. Experiment Result

5.1. Introduction of KITTI Dataset

The odometry dataset provided by KITTI consists of 22 sequences, of which the first 11 sequences are labeled with ground truth. Each sequence contains frames ranging from 81 to 2000 frames. For every frame, a 3×4 ground truth transformation matrix which describes the transformation from the beginning to current time stamp is provided. In addition to frames, the information of IMU is available for every frame, but only in several sequences. Since our model is designed for data with IMU information, we will discard the sequences without IMU data in our experiment.

5.2. Experiment Details

In the experiment of KITTI odometry dataset, we take *sequence 01* as our test sequence and the other 9 sequences as training set. Before the deadline, we finally finish training our model for 27 epochs. In each epoch, we will choose three sequences to train our model, with 2700 iterations for each training epoch and 1100 iterations for each testing epoch. Because we need to avoid training on the same sequences in each epoch, we choose different three sequences in each epoch. In our current setting, each epoch takes roughly 2.5 hours, which results in about 67.5 total training hours.

5.2.1 Model Performance

The result of the experiments are shown in the Figure 6 to 10. Figure 6 shows the attitude losses derived from SE(3) and se(3) of both training set and test set. Figure 7 gives the position losses. By summing the weighted attitude losses and the weighted position losses we gain the total losses, which are shown in Figure 8. Based on the model we train, we predict the trajectory of both train set and test set. The prediction is demonstrated in Figure 9(a) and 9(b).

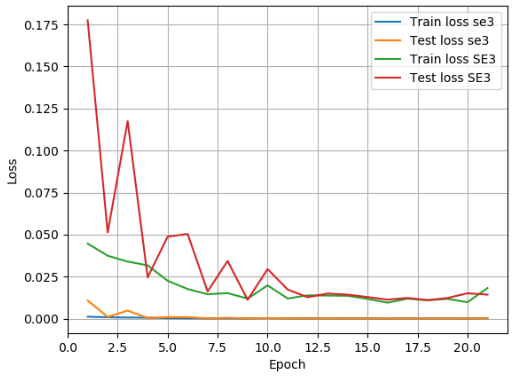


Figure 6. KITTI Attitude Measurement

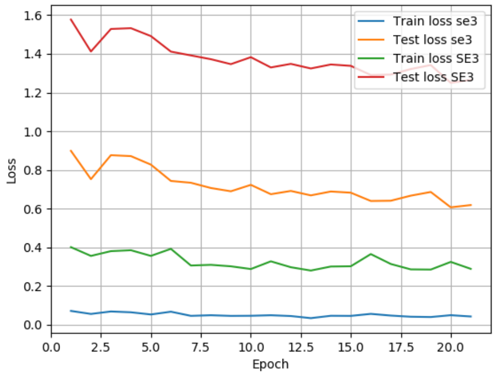


Figure 7. KITTI Position Measurement

5.3. Further Experiment on Frame-to-frame Prediction

We can see that the losses of rotation decrease over epochs, while the ones of translation do not drop as we expected. This is due to the reason that we have not found an appropriate ratio between the weights for position loss and the ones for rotation loss. Since the ratio is not mentioned in the VINet paper, we have to try to train the model with

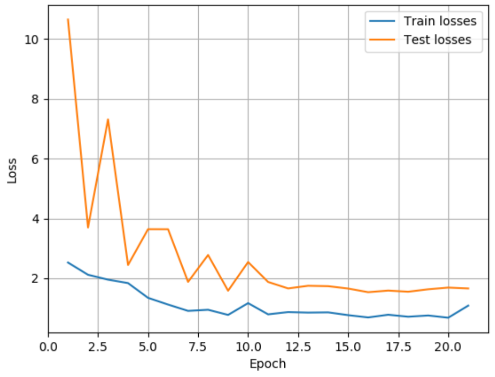
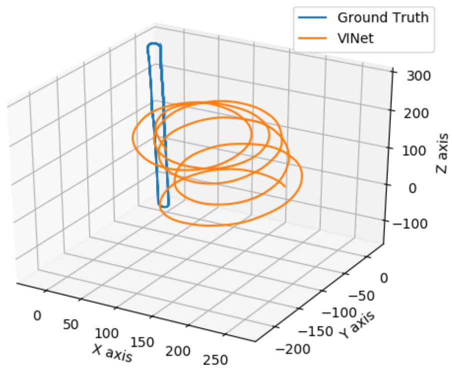
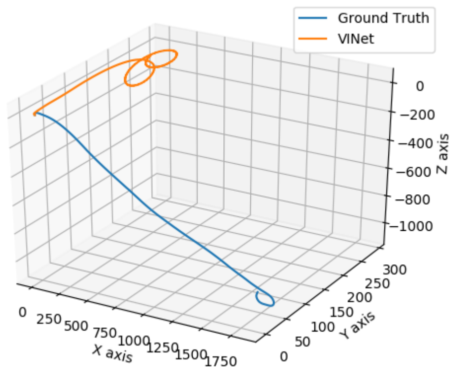


Figure 8. KITTI Total losses



(a) Trace Prediction on Training Data



(b) Trace Prediction on Testing Data

Figure 9. KITTI Trace Prediction

different ratios to find a proper one, but we do not manage to find the one which is able to have the translation losses drop more rapidly. In the view that the trajectory performance is not good, we are curious about the reason and design an additional experiment for it. Instead of using the previous predicted attitude and position as the starting point to predict next time step, we decide to always "peep" at the ground truth. To be more specific, for the prediction of each time step $t + 1$, we regard the ground truth for time step t as initial state.

From Figure 10, we can tell that the model performs perfectly well on predicting displacements. In comparison to Figure 9(a) and Figure 9(b), whose results are caused by accumulated errors, Figure 10 verifies the correctness of model performance in terms of se(3).

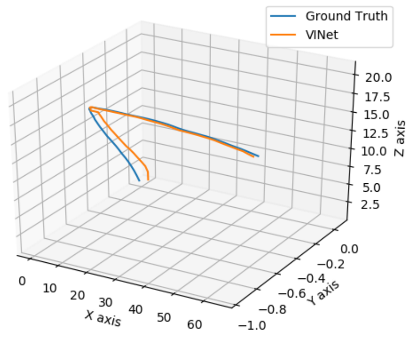


Figure 10. KITTI Trace Predicted by Displacement

6. Conclusion

Because the paper did not provide the implementation details of the model, we implemented the whole model in Keras from scratch with plenty of assumptions. First we implemented the correlation layer by doing correlation between image patches of parallel feature maps to extract depth information from both images. Then we concatenated that with another convolutional layer called *conv_redir* which will preserve some information from the first image to improve the performance. We used three layers of IMU-LSTM of 15 units each and two layers of Core-LSTM. For the SE(3) composition layer, we create a customized layer and slightly different loss functions, angular se(3), position se(3), angular SE(3), position SE(3), and overall losses. Furthermore, a batch size of 3 severely limited the ability of the SE(3) loss to propagate position errors backwards through time, the as shown in Figure 7.

In the original paper, the authors trained the model for 200 epochs and we only trained for 27 epochs, so we didn't get the results as good as the original paper. But we can see from the Figures that the se(3) and SE(3) losses for rotation are decreasing as expected, while the losses for translation remain almost the same. This indicates that the network is learning rotations well from the data, but is having difficulty estimating the scale of translations from the monocular camera data. Figures 6 and 7 also demonstrates the the network is learning to minimize se(3) losses, but this implementation has difficulty minimizing SE(3) losses without the ability to perform long sequences of backpropagation through time.

References

[1] Ronald Clark, Sen Wang, Hongkai Wen, Andrew Markham, and Niki Trigoni. Vinet: Visual-inertial odometry as a sequence-to-sequence learning problem. *arXiv preprint arXiv:1701.08376*, 2017. 1, 2, 5

[2] M Walker. Navigating oceans and cultures: Polynesian and european navigation systems in the late eighteenth century. *Journal of the Royal Society of New Zealand*, 42(2):93–98, 2012. 1

[3] Carol Fairchild and Thomas Harman. *ROS Robotics By Example*. Packt Publishing, 2016. 1

[4] Lee Lemay, Chen-Chi Chu, Demoz Gebre-Egziabher, and Rohan Ramlall. Precise input and output error characterization for loosely integrated ins/gps/camera navigation system. In *Proceedings of the 2011 International Technical Meeting of The Institute of Navigation*, pages 880–894, 2011. 1

[5] Lee Lemay, Chen-Chi Chu, Demoz Gebre-Egziabher, and Justin Gorgen. Compensating for colored measurement noise in vision-aided ins. In *Proceedings of 2012 Joint Navigation Conference (JNC 2012)*, 2012. 1

[6] Richard Hartley and A. Zisserman. Multiple view geometry in computer vision, 2000. 1

[7] Paul D Groves. Principles of gnss, inertial, and multisensor integrated navigation systems, 2008. 1, 5

[8] Mark Maimone, Yang Cheng, and Larry Matthies. Two years of visual odometry on the mars exploration rovers. *Journal of Field Robotics*, 24(3):169–186, 2007. 2

[9] Eagle Sunrise Jones. *Large scale visual navigation and community map building*. PhD thesis, University of California Los Angeles, 2009. 2

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 2

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. 3

[12] Philipp Fischer, Alexey Dosovitskiy, and Thomas Brox. Descriptor matching with convolutional neural networks: a comparison to sift. *arXiv preprint arXiv:1405.5769*, 2014. 3

[13] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1915–1929, 2013. 3

[14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014. 3

[15] David Eigen, Christian Puhrsch, and Rob 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. 3

[16] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazırbaş, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. *arXiv preprint arXiv:1504.06852*, 2015. 3, 4