

Global Pose Estimation with an Attention-based Recurrent Network

Emilio Parisotto^{*,1,2}, Devendra Singh Chaplot^{*,1,2}, Jian Zhang¹, Ruslan Salakhutdinov^{1,2}
*Equal Contribution.

¹Apple Inc., 1 Infinite Loop, Cupertino, CA 95014, USA.

²Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA 15213, USA.

{eparisot, dchaplot}@cs.cmu.edu

{jianz, rsalakhutdinov}@apple.com

Abstract

The ability for an agent to localize itself within an environment is crucial for many real-world applications. For unknown environments, Simultaneous Localization and Mapping (SLAM) enables incremental and concurrent building of and localizing within a map. We present a new, differentiable architecture, Neural Graph Optimizer, progressing towards a complete neural network solution for SLAM by designing a system composed of a local pose estimation model, a novel pose selection module, and a novel graph optimization process. The entire architecture is trained in an end-to-end fashion, enabling the network to automatically learn domain-specific features relevant to the visual odometry and avoid the involved process of feature engineering. We demonstrate the effectiveness of our system on a simulated 2D maze and the 3D ViZ-Doom environment.

1. Introduction

The ability for an agent to localize itself within an environment is a crucial prerequisite for many real-world applications, such as household robots [42], autonomous drones [14], augmented and virtual reality applications, and video game AI [36]. In most cases, the main challenge for an agent localizing itself is that, the agent is not provided with a map of the environment and therefore the agent must simultaneously map the environment and localize itself within the incomplete map it has produced. A wide variety of algorithms to solve this Simultaneous Localization and Mapping (SLAM) task have been developed over a long history [42, 4], with modern methods achieving impressive accuracy and real-time performance [33, 25, 34, 12]. These methods still have several shortcomings, owing mainly to the hand-engineered features, dense matching, and heuristics used in the design of these algorithms. For example, most methods are brittle in certain scenarios, such as varying lighting conditions (e.g. changing time of day), differ-

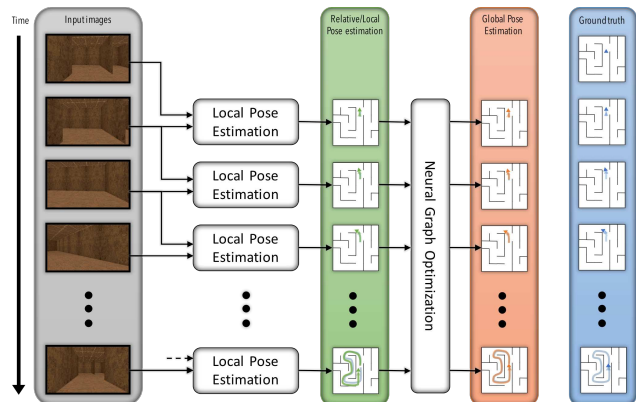


Figure 1. Components of the proposed model along with sample input, output and ground truth. The Local Pose Estimation model predicts the relative pose change between consecutive observations and Neural Graph Optimization model jointly optimizes the predictions of the Local Pose Estimation model to predict global pose changes. The local pose estimates, global pose estimates, and ground truth trajectory are shown in green, orange and blue, respectively.

ent weather conditions or seasons [40], repetitive structures, textureless objects, extremely large viewpoint changes, dynamic elements within the environment, and faulty sensor calibration [4]. Because these situations are common in real-world scenarios, robust applications of those systems are difficult.

In this paper, we develop a method which can be made more robust to the common situations where previous SLAM algorithms typically degrade. To do this, we formulate a novel neural network architecture called “Neural Graph Optimizer”. Neural Graph Optimizer consists of differentiable analogues of the common types of subsystems used in modern SLAM algorithms, such as a local pose estimation model, a pose selection module (key frame selection, essential graph), and a graph optimization process. Because each component in the system is differentiable, the entire architecture can be trained in an end-to-end fashion,

enabling the network to learn invariances to the types of scenarios observed during training.

To demonstrate the ability of our method to learn pose estimation, we use trajectories sampled from several simulated environments. The first environment is a 2D maze where the agent has a single-pixel row-scan as input. We then scale the model up to 3D mazes based on the ViZDoom environment [21], where the agent receives an image of the first-person view of the world as input.

2. Related Work

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 [33], the VIN [23], and the system of Hesch et al. [19]. Those methods, even though efficient, heavily depend on linearization and Gaussian assumptions, and thus under-perform their optimization-based counterparts, such as OK-VIS [27], ORB-SLAM [34], and LSD-SLAM [12].

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 [34] performs data association and camera relocalization with ORB features and DBoW2 [16]. RANSAC [13] is commonly used for geometric verification and outlier rejection, and there are also prioritized feature matching approaches [39]. 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 [12], 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 [4] during large viewpoint changes. Recent work in [14] [15] combines feature and direct methods by minimizing the photometric error of features lying on intensity corners and edges. Some meth-

ods focus on dense reconstruction of the scene, for instance [47] builds dense globally consistent surfel-based maps of room scale environments explored using an RGB-D camera, without pose graph optimisation, while KinectFusion [35] obtains depth measurements directly using active sensors and fuse 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 [11], depth [28], homography between frame pairs [9], and localization [5] and re-localization problems.

Recent work includes re-formulating the localization problem as a classification task [46], a regression task [22, 44], end-to-end trainable filtering [18], and differentiable RANSAC [3]. More specifically, PlaNet [46] 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 [22] formulates 6-DoF pose estimation as a regression problem. One drawback of the PoseNet approach is its relative inaccuracy, compared to state-of-the-art SIFT methods. Similarly, [30] fine-tunes a pretrained classification network to estimate the relative pose between two cameras. To improve its performance, [44] 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 [3] 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 [6] fused additional sequential inertial measurement with visual odometry. SemanticFusion [29] combines convolutional neural networks (CNNs) and a dense ElasticFusion [47]. However, classic feature-based methods still outperform CNN-based methods published to date in terms of accuracies.

Recently, there has been an increasing interest in combining navigation and planning in an end-to-end deep reinforcement learning (DRL) framework. The efforts to date can be divided into two categories depending on the presence of external memory in the architecture or not. Target-driven visual navigation takes a visual observation and an image of the target [50] or range findings [41] as input, and

plans goal seeking actions in a 3D indoor simulated environment as the output.

In simulated environments, [31] uses stacked LSTM in a goal-driven RL problem with auxiliary tasks of depth prediction and loop-closure classification, while [48] added successor features to ease transfer from previously mastered navigation tasks to new ones. Work in [1] augmented DRL with Faster-RCNN for object detection and SLAM (ORB-SLAM2) for pose estimation; observing images and depth from VizDoom, they built semantic maps with 3D reconstruction and bounding boxes as input to a RL policy.

To deal with the limited memory of standard recurrent architectures (such as LSTM) more structured external memories have been developed to take the spatial relations of memories into account. [17] assumes known ego-motion and constructs a metric egocentric multi-scale belief map (top-down-view latent representation of free space) of the world with a 2D spatial memory, upon which RL plans a sequence of actions towards goals in the environment with a value iteration network. Neural Map in [36] is a writable structured 2D external memory map for an agent to learn to navigate within 2D and 3D maze environments. These works all assume precise egomotion and thus perfect localization, a prerequisite that can rarely be met in real-world scenarios. Relaxing this assumption and resembling traditional occupancy grid SLAM, Neural SLAM [49] uses an occupancy-grid-like memory map, assuming only an initial pose is provided, and updates the pose beliefs and grid map using end-to-end DRL.

One of 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 [25], TSAM [8], Ceres, iSAM [20], SLAM++ [38], and recently [2] 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 environ-

ment are too challenging (e.g. fast robot dynamics or highly dynamic environments) [4].

In this work we formulate a complete end-to-end trainable solution to the graph-based SLAM problem. We present a novel architecture that combines a CNN-based local front-end and an attention-based differentiable back-end. We learn effective features automatically and perform implicit loop closure by designing an additional differentiable Neural Graph Optimizer to perform global optimization over entire pose trajectories and correct errors accumulated by the local estimation model.

3. Method

The Neural Graph Optimizer architecture is split into distinct differentiable components. Similar to many of the previous methods, we split the process into local adjustments between temporally adjacent frames combined with a global optimization procedure which distributes error over the entire observed trajectory. As will be shown in the experiments, the global graph optimization procedure is critical to removing drift (the accumulation of small errors over long trajectories). The graph optimization procedure does this by learning to do loop closures, recognizing when the agent has revisited the same location, and enforcing a constraint that those poses should be nearly equal. The local model is crucial for providing a good starting point for the global optimization. It does this by estimating relative transformations between two temporally adjacent frames. By accumulating transformations from the start of the trajectory to the end, we can use this model to get the initial pose estimate within the global frame.

The complete model architecture is shown in Fig. 2. We will describe relative poses as $\Delta P = (\Delta p_1, \dots, \Delta p_T)$ with the first pose set as the origin, i.e. Δp_1 is the transformation from origin to pose 1, Δp_2 is the transformation from pose 1 to pose 2, and so on. These relative poses can be transformed into a global frame of reference by accumulating the relative pose changes along the trajectory, i.e. $p_1 = \Delta p_1 \mathbf{I}$, $p_2 = \Delta p_2 \Delta p_1 \mathbf{I}$, and so on. These global poses will be referred to as $P = (p_1, \dots, p_T)$. There exists a differential function $r2g = g2r^{-1}$ such that $P = r2g(\Delta P)$ and $\Delta P = g2r(P)$. Each component is described in more detail in the next sections.

3.1. Local Pose Estimation Network

The Local Pose Estimation network learns to predict the relative pose change between two consecutive frames. From two consecutive observations, where each observation is, for example, an RGB frame, this component predicts the x-coordinate, y-coordinate, and orientation (Δx , Δy and $\Delta \theta$) of the second frame with respect to the first frame. It can also optionally take in side information, such as the action taken by the agent between the two frames. The architecture

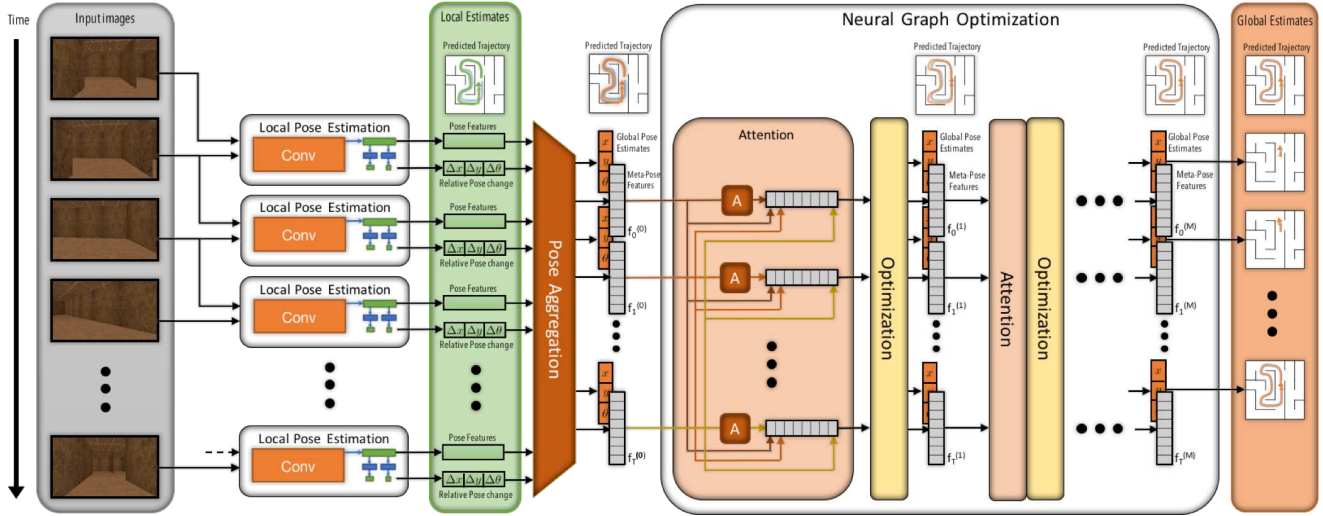


Figure 2. The architecture of the proposed model, showing the Local Pose Estimation, the Pose Aggregation, and the Neural Graph Optimization modules.

of the Local Pose Estimation network is shown in Fig. 3. Both frames are stacked and passed through a series of convolutional layers. The output of the convolutional layers is flattened and passed to two fully-connected layers that predict the translational and rotational pose change respectively.

Some of the recent work showed that optical flow is useful in predicting frame-to-frame ego-motion [7]. The architecture of the Local Pose Estimation network is inspired by the architecture of Flownet [11] which predicts the optical flow between two frames. The convolutional layers in the Local Pose Estimation network are identical to the convolutional layers in Flownet. Prior work on visual odometry and visual inertial odometry has also used the convolutional layer architecture of Flownet [6, 45].

3.2. Pose Aggregation

The next step of the architecture is a Pose Aggregation network which takes in a large number of low-level poses and pose features (up to 2000 for 2D, 1000 for 3D VizDoom environment) and reduces them into a smaller number of more temporally distant “meta-poses” and “meta-pose features” (around 250 for 2D, 125 for 3D VizDoom). These resulting meta-poses and meta-pose features are then passed to the Neural Graph Optimization procedure.

For pose feature aggregation, we utilize a deep temporal convolutional network with several alternating layers of (kernel size 3, stride 1, padding 1) dimension-preserving convolutions and (kernel size 2, stride 2, padding 0) dimension-reducing max pooling (where each max pooling operation halves the sequence size). The number of times we halve the sequence length is a hyperparameter. Instead of temporal convolutions, we could have utilized recurrent networks, but we decided to focus on convolutions for computational and memory-efficiency reasons.

In addition to the pose features being aggregated into meta-pose features by the temporal convolution, we also compose all the local pose transformations that were predicted by the Local Pose Estimation model. This composition gives us an initial global pose estimate for each of the meta-poses. The combined meta-features and meta-poses are then passed onto the Neural Graph Optimization layer for the final global pose adjustments, as shown in Fig. 2.

3.3. Neural Graph Optimization

The final component of our system is the “Neural Graph Optimizer”. This submodule aggregates information over the entire pose trajectory with the goal of redistributing error to minimize drift. The Neural Graph Optimizer model is a neural analogue of the global optimization procedures commonly used in traditional state-of-the-art SLAM packages, such as the g2o framework [25]. We define the Neural Graph Optimizer as a recurrent network submodule which takes as input sequential pose features and outputs a refined estimate of these poses.

In more detail, the Neural Graph Optimizer takes as input some initial T relative pose estimates (i.e. the aggregated output of the local pose estimation network) $\Delta \mathbf{P}^{(0)} = (\Delta p_1^{(0)}, \dots, \Delta p_T^{(0)})$ and produces two outputs for each pose:

$$\nabla \mathbf{P}^{(1)} = (\nabla p_1^{(1)}, \dots, \nabla p_T^{(1)}), \text{ and}$$

$$\beta^{(1)} = (\beta_1^{(1)}, \dots, \beta_T^{(1)}).$$

New pose estimates are then constructed by performing an iterative update:

$$\Delta p_i^{(1)} = \Delta p_i^{(0)} + \beta_i^{(1)} \nabla p_i^{(1)}.$$

The Neural Graph Optimizer procedure can then be rerun on the new pose estimates $\Delta \mathbf{P}^{(1)} = (\Delta p_1^{(1)}, \dots, \Delta p_T^{(1)})$

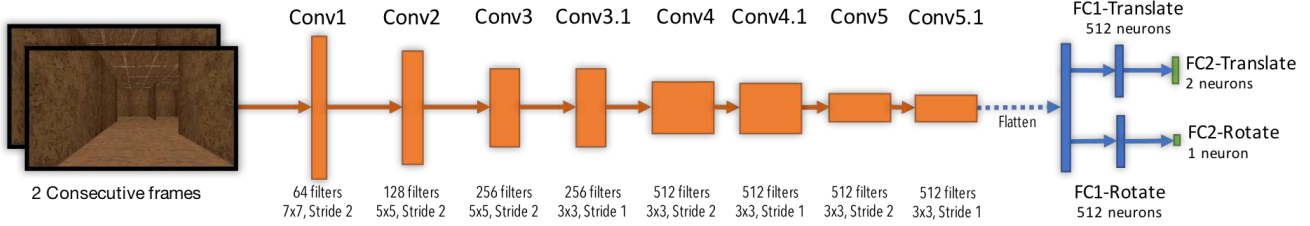


Figure 3. The architecture of the Local Pose Estimation network. The architecture of the convolutional layers is adapted from the architecture of the Flownet [11].

to produce $\Delta \mathbf{P}^{(2)} = (\Delta p_1^{(2)}, \dots, \Delta p_T^{(2)})$, and so on. The process is repeated until some pre-specified number of iterations M has taken place. We then transform the refined relative pose estimates into the final global output: $\mathbf{P}^{(M)} = r2g(\Delta \mathbf{P}^{(M)})$.

The specific architecture of the Neural Graph Optimizer is based on two priors that are intuitively useful for pose optimization. The first prior is the notion that poses that are temporally adjacent should have similar outputs, while the second prior is that visually similar but temporally disparate poses should also have similar outputs since this provides a hint that a place has been revisited, thereby potentially enabling a loop closure-like correction of drift. We express these priors by using two architectural systems in the Neural Graph Optimizer. The first is a Transformer-like [43] attention phase where information is propagated over the entire sequence, and the second is a convolutional phase where local temporal information is aggregated.

3.3.1 Attention Phase

Suppose there is a meta-pose sequence of T steps, processed by the pose aggregation network into an initial set of features at each time step: $\mathbf{F}^{(0)} = (f_1^{(0)}, \dots, f_T^{(0)})$. The attention phase computes, for each pose, a soft-attention operation over the entire trajectory. This attention operation allows each pose to query information over long time spans. The attention phase takes as input the pose feature sequence $(f_1^{(i-1)}, \dots, f_T^{(i-1)})$ and produces for each time step a query vector: $(q_1^{(i-1)}, \dots, q_T^{(i-1)})$ using a fully-connected layer. Then, for each query vector $q_t^{(i-1)}$, a soft-attention operation is carried out to produce an attention vector $a_t^{(i-1)}$ as follows:

$$\begin{aligned}
 C_{tu} &= \langle q_t, f_u \rangle, \\
 \alpha_{tu} &= \frac{C_{tu}}{\sum_{v=1}^T C_{tv}}, \\
 a_t &= \sum_{v=1}^T \alpha_{tv} \odot f_v,
 \end{aligned}$$

where the superscripts $(i-1)$ were omitted for clarity of notation. This produces a sequence of attention vectors $(a_1^{(i-1)}, \dots, a_T^{(i-1)})$, which are passed along with $(f_1^{(i-1)}, \dots, f_T^{(i-1)})$ to the next ‘‘Optimization’’ phase.

3.3.2 Optimization

The optimization phase aggregates local temporal information by passing the pose features through several temporal convolutions and is responsible for producing the iterative adjustments: $\{\nabla p_1^{(i)}, \dots, \nabla p_T^{(i)}\}$ and $\{\beta_1^{(i)}, \dots, \beta_T^{(i)}\}$. The optimization phase proceeds as follows: First, the attention and feature vectors are concatenated into a new sequence of features:

$$\left(\begin{bmatrix} f_1^{(i-1)} \\ a_1^{(i-1)} \end{bmatrix}, \dots, \begin{bmatrix} f_T^{(i-1)} \\ a_T^{(i-1)} \end{bmatrix} \right).$$

These features are then passed through several layers of 1D convolutions h_l and activations σ_l :

$$\begin{bmatrix} \mathbf{F}^{(i)} \\ \nabla \mathbf{P}^{(i)} \\ \beta^{(i)} \end{bmatrix} = \sigma_L \left(h_L \left(\dots h_1 \left(\begin{bmatrix} f_1^{(i-1)} \\ a_1^{(i-1)} \end{bmatrix} \dots \begin{bmatrix} f_T^{(i-1)} \\ a_T^{(i-1)} \end{bmatrix} \right) \dots \right) \right)$$

to produce the current iteration’s adjustments $(\nabla \mathbf{P}^{(i)})$ and $\beta^{(i)}$ as well as the feature layer for the next iteration of the process $(\mathbf{F}^{(i)})$.

For our experiments, we use 9 layers of convolutions with filter size 3 and ReLU activations. While temporal convolutions have a limited receptive field which provides a hard upper limit on how far they can transmit information across time, we found that in practice they worked better than using a bidirectional LSTMs.

3.3.3 Induced Attention Graph

We now provide some intuition on why the attention phase enables higher performance than only using the optimization phase, or running all pose features through bidirectional LSTMs. We can see that during the attention phase, some similarity graph C is constructed such that each element C_{tu} is the inner product between the query vector q_t

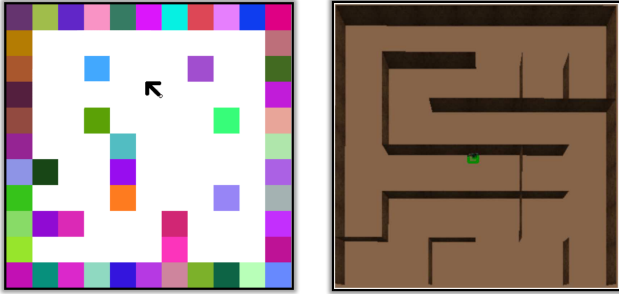


Figure 4. **Left:** A screenshot of the 2D environment based on Box2D. **Right:** A bird's eye view of the 3D environment based on the Doom game engine.

Results on the 2D Environment	
Model	RMSE
Only Local Estimation	17.80
Global Estimation - 1 Attend-Opt iteration	10.21
Global Estimation - 5 Attend-Opt iterations	3.16

Table 1. Results for different Neural Graph Optimizer architectures and hyperparameters, in terms of test set Global RMSE. We can see that the addition of the global optimization procedure reduces the loss by more than 80% as compared to solely using the local pose model.

and the pose feature vector f_u . Therefore C represents a similarity matrix between the queries and pose features, and those with very similar features will thus have high information bandwidth through the attention operator because the attention weight α_{tu} will be near 1 for highly similar query and pose features, and near 0 otherwise. The attention operation is thus inducing a connectivity graph between poses with highly similar features. This therefore resembles a soft, differentiable analogue of the pose graph constructed in SLAM algorithms such as ORB-SLAM [34].

4. Experiments

We use two simulation environments for our experiments, a 2D environment based on Box2D and a 3D environment based on the Doom game engine. To train the system, we pretrained the local pose estimation model and then trained the global optimizer with the local pose model held fixed. This was mainly due to the large sequence lengths we were required to process (on the order of 1000 time steps). This limited the amount of sequences we could process due to the large memory requirements. Training the system in stages enabled us to preprocess the sequence images into a far more memory-efficient compressed representation.

4.1. 2D Environment

For the 2D Environment, random maze designs are generated using Prim's algorithm [37], and the environment is created using Box2D (box2d.org). The agent projects 241

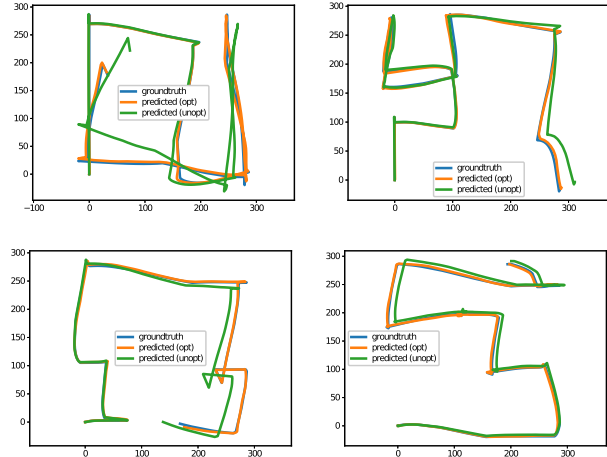


Figure 5. Images visually demonstrating the effect on pose estimates of adding the Neural Graph Optimizer module on top of the local pose estimation model in the 2D environment. We can see that the global optimization procedure greatly reduces drift. These figures were generated with the 5 iteration Neural Graph Optimization model.

rays uniformly in front of itself with an effective field of view of 300° . The observation of the agent includes the RGB values as well as the depth of the points where these rays hit a wall. An example of the 2D environment is shown in Fig. 4. Each cell in the maze has a random color. The agent can take one of three discrete actions at every time step: move-forward, turn-left, or turn-right. These actions result in translational acceleration if the action is move-forward or angular acceleration if the action is turn-left or turn-right. Data is collected by visiting four different corners on the maze using Dijkstra's algorithm [10].

For this environment, the training data is generated by worker threads in parallel with the model training and each training datapoint is used only once. A test set is fixed and common for all experiments. Each epoch of training consists of 200,000 datapoints. The error metric is Root Mean Squared Error (RMSE) in pose estimation.

To improve upon the results produced by the local pose estimation model, we train a Neural Graph Optimizer on the pose outputs of a pretrained Local Pose Estimation model. For the 2D environment, as shown in Table 1, we observed over 80% improvement in the correction of drift compared to using only the local pose estimation model, as measured by the root mean squared error loss. We can see that increasing the number of iterations (applying the attention operator and then the temporal aggregation operator) improved results from 1 to 5 iterations. We show some sample trajectories in Fig. 5 before and after the Neural Graph Optimizer procedure.

Results on the 3D Doom Environment				
Model	Seen		Unseen	
	% Err. Trans.	% Err. Rot.	% Err. Trans.	% Err. Rot.
Only Local Estimation	1.65	0.117	1.62	0.122
Global Estimation - 1 Attend-Opt iteration	1.42	0.071	1.16	0.071
Global Estimation - 5 Attend-Opt iterations	1.25	0.057	1.04	0.056
DeepVO [45]	1.78	0.079	2.39	0.091

Table 2. Results for different Neural Graph Optimizer architectures and hyperparameters, in terms of % translation and rotation error on maps either seen or unseen during training time. We can see that the addition of the global optimization procedure reduces error significantly compared to using only the local pose model. In addition, increasing the number of attention iterations provides an increase in performance.

4.2. 3D Environment

For the 3D Environment, random maze designs are generated using the Kruskal’s algorithm [24], and the environment is created using the ViZDoom API [21]. The agent observes the environment in a first-person view with a field-of-view of 108° . An example of the 3D environment design is shown in Fig. 4. Similar to the 2D environment, the pose predictions are 3-dimensional tuples (x, y, angle) and the agent can take one of three discrete actions at every time step: move-forward, turn-left, or turn-right, which results in translational or angular acceleration. For collecting data in this environment, a navigation network [26] is trained to maximize the distance travelled by the agent using the Asynchronous Advantage Actor-Critic algorithm [32]. The data is collected by using the policy learned by the navigation network.

Like the 2D environment, the training data is generated by worker threads in parallel with the model training, and each training datapoint is used only once. We additionally sample two test sets, one containing 39 trajectories sampled from maze geometries that were seen during training and one containing 39 trajectories sampled from novel maze geometries that the agent had not encountered during training.

4.2.1 Results

Results are shown in Table 2. Here we report % Error in Translation and Rotation for seen/unseen mazes, where the accumulated drift error is divided by the entire distance traveled in each trajectory. Observe that the local model is significantly improved by using global optimization and performance of the global model improves as we increase the number of Attend-Opt iterations from 1 to 5. The global model outperforms the DeepVO [45] baseline on both the test sets. Additionally, we can clearly see that the model itself does not overfit to the training environments it experienced, and gets similar or even lower error on unseen test mazes. Learning curves are shown in Fig 6. We can see that performance plateaus decrease significantly early on and then progress is much slower after around 2000 updates.

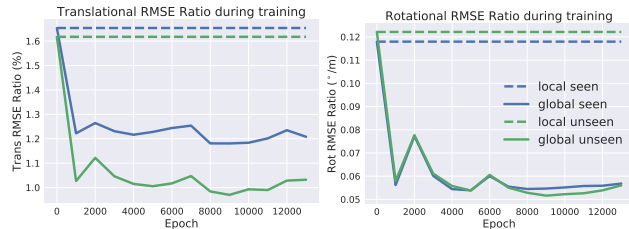


Figure 6. Training curves for Doom over 13,000 updates for the 5 iteration Attend-Opt model. We show the performance on both seen and unseen test sets as training progress. The dotted line represents the estimate provided by using only the local model. We can see there is a large reduction in error when making use of the global optimizer.

The baseline DeepVO [45] is one of the state-of-the-art methods using deep neural nets for monocular visual odometry. It stacks 2 consecutive frames and passes them through 9 convolutional layers followed by 2 LSTM layers to estimate the pose changes. As compared to the proposed Local Pose Estimation model which observes only the last 2 frames at the time, the DeepVO model can potentially utilize information from all the prior frames using the LSTM layer. However, the DeepVO model does not correct its previous predictions as it observes new information. The Neural Graph Optimizer has the ability to correct its predictions using the Attention operation and consequently leads to improved performance.

4.2.2 Analysis

We next plot the total rotational and translational errors as a function of number of steps in the trajectory in Figures 7 (for unseen mazes) and 8 (for seen mazes). The global model reduces the slope of the rate of increase of both translational and rotation errors as compared to the local estimates. Figures 9 and 10 display the ratio of the translational (left) and rotational (right) drift error over distance traveled. We can see from these plots that the trend is negative, meaning that drift accumulates much slower than the distance being traveled. This indicates that the model is likely to generalize well to arbitrarily long trajectories. Additionally, in all plots, we can see a clear ordering of the performance of

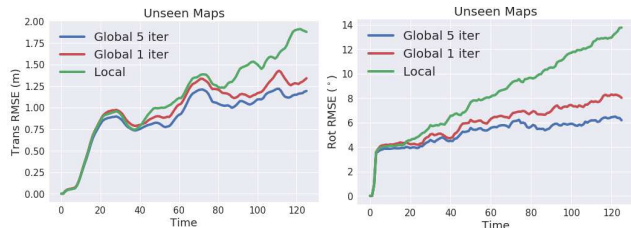


Figure 7. Translational (Left) and Rotational (Right) RMSE as a function of number of images in the trajectory in **unseen mazes**.

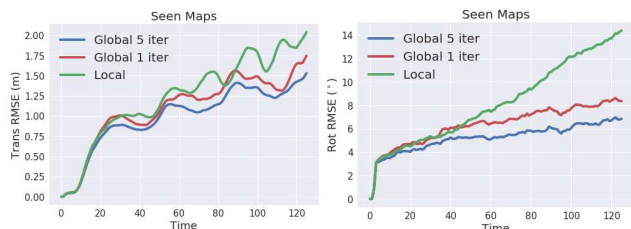


Figure 8. Translational (Left) and Rotational (Right) RMSE as a function of number of images in the trajectory in **seen mazes**.

the models, where the local model performs worst, one iteration of Attend-Opt increases model performance significantly, and increasing the number of Attend-Opt iterations to 5 further increases model performance.

The plots in Figures 7 and 8 as well as the numbers in Table 2 show that the improvement in rotational errors due to the neural optimization is higher than the improvement in translation errors. Fig 11 shows sample trajectories with estimates of both global and local pose estimates. As seen in the figure, the neural graph optimizer considerably improves the rotation estimates, consequently leading to significant improvements in the drift reduction.

5. Conclusion

In this paper, we designed a novel attention-based architecture to perform an end-to-end trainable global pose estimation. Compared to the previous work on using deep networks to do pose estimation, our method uses an attention operation to re-estimate its trajectory at each time step and therefore enables iterative refinement of the quality of its estimates as more data is available. We demonstrate the benefit of the model on two simulators, the first is a top-down 2D maze world and the second is a 3D random maze environment running the Doom engine. Our results show that our method has an increased performance compared to models which used only temporally local information.

The proposed method can be further extended to a complete end-to-end graph-based SLAM system by adding a re-localization module which uses pose features to relocalize in a known map [5]. It can also be extended to an Active SLAM system where the agent also decides the actions, in order to map the environment as fast as possible.

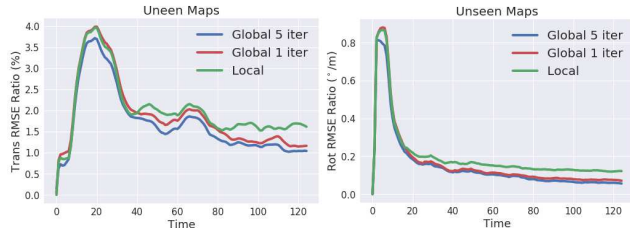


Figure 9. Ratio of the Translational (Left) and Rotational (Right) RMSE to the distance travelled as a function of number of images in the trajectory in **unseen mazes**.

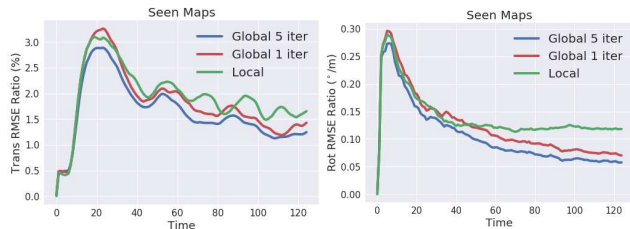


Figure 10. Ratio of the Translational (Left) and Rotational (Right) RMSE to the distance travelled as a function of number of images in the trajectory in **seen mazes**.

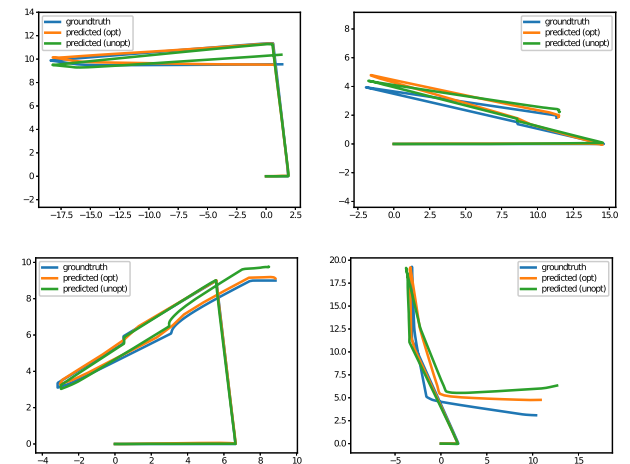


Figure 11. Images visually demonstrating the effect on pose estimates of adding the Neural Graph Optimizer module on top of the local pose estimation model in the 3D environment. We can see that the global optimization procedure greatly reduces drift. These figures were generated with the 5 iteration Neural Graph Optimization model. The agent always starts at the origin (0, 0).

Acknowledgments

We thank Tim Barfoot and Russ Webb for helpful comments and discussions. We would also like to thank Barry Theobald and Megan Maher for helpful feedback on the manuscript.

References

- [1] S. Bhatti, A. Desmaison, O. Miksik, N. Nardelli, N. Sidharth, and P. H. Torr. Playing doom with slam-augmented deep reinforcement learning. *arXiv preprint arXiv:1612.00380*, 2016.
- [2] S. L. Bowman, N. Atanov, K. Daniilidis, and G. J. Pappas. Probabilistic data association for semantic slam. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 1722–1729. IEEE, 2017.
- [3] E. Brachmann, A. Krull, S. Nowozin, J. Shotton, F. Michel, S. Gumhold, and C. Rother. Dsac-differentiable ransac for camera localization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 3, 2017.
- [4] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. 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.
- [5] D. S. Chaplot, E. Parisotto, and R. Salakhutdinov. Active neural localization. In *International Conference on Learning Representations (ICLR)*, 2018.
- [6] R. Clark, S. Wang, H. Wen, A. Markham, and N. Trigoni. Vinet: Visual-inertial odometry as a sequence-to-sequence learning problem. In *AAAI*, pages 3995–4001, 2017.
- [7] G. Costante, M. Mancini, P. Valigi, and T. A. Ciarfuglia. Exploring representation learning with cnns for frame-to-frame ego-motion estimation. *IEEE robotics and automation letters*, 1(1):18–25, 2016.
- [8] F. Dellaert. Factor graphs and gtsam: A hands-on introduction. Technical report, Georgia Institute of Technology, 2012.
- [9] D. DeTone, T. Malisiewicz, and A. Rabinovich. Deep image homography estimation. *arXiv preprint arXiv:1606.03798*, 2016.
- [10] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.
- [11] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. FlowNet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2758–2766, 2015.
- [12] 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.
- [13] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In *Readings in computer vision*, pages 726–740. Elsevier, 1987.
- [14] C. Forster, M. Pizzoli, and D. Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 15–22. IEEE, 2014.
- [15] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza. Svo: Semidirect visual odometry for monocular and multicamera systems. *IEEE Transactions on Robotics*, 33(2):249–265, 2017.
- [16] D. Gálvez-López and J. D. Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, 2012.
- [17] S. Gupta, J. Davidson, S. Levine, R. Sukthankar, and J. Malik. Cognitive mapping and planning for visual navigation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2616–2625, 2017.
- [18] T. Haarnoja, A. Ajay, S. Levine, and P. Abbeel. Backprop kf: Learning discriminative deterministic state estimators. In *Advances in Neural Information Processing Systems*, pages 4376–4384, 2016.
- [19] J. A. Hesch, D. G. Kottas, S. L. Bowman, and S. I. Roumeliotis. Camera-imu-based localization: Observability analysis and consistency improvement. *The International Journal of Robotics Research*, 33(1):182–201, 2014.
- [20] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert. isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, 31(2):216–235, 2012.
- [21] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, pages 1–8. IEEE, 2016.
- [22] A. Kendall, M. Grimes, and R. Cipolla. PoseNet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, 2015.
- [23] D. G. Kottas, J. A. Hesch, S. L. Bowman, and S. I. Roumeliotis. On the consistency of vision-aided inertial navigation. In *Experimental Robotics*, pages 303–317. Springer, 2013.
- [24] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956.
- [25] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3607–3613. IEEE, 2011.
- [26] G. Lample and D. S. Chaplot. Playing fps games with deep reinforcement learning. In *AAAI*, pages 2140–2146, 2017.
- [27] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale. Keyframe-based visual-inertial odometry using non-linear optimization. *The International Journal of Robotics Research*, 34(3):314–334, 2015.
- [28] F. Liu, C. Shen, and G. Lin. Deep convolutional neural fields for depth estimation from a single image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5162–5170, 2015.
- [29] J. McCormac, A. Handa, A. Davison, and S. Leutenegger. Semanticfusion: Dense 3d semantic mapping with convolutional neural networks. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 4628–4635. IEEE, 2017.
- [30] I. Melekhov, J. Ylioinas, J. Kannala, and E. Rahtu. Relative camera pose estimation using convolutional neural networks. In *International Conference on Advanced Concepts for Intelligent Vision Systems*, pages 675–687. Springer, 2017.

- [31] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.
- [32] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.
- [33] A. I. Mourikis and S. I. Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. In *Robotics and automation, 2007 IEEE international conference on*, pages 3565–3572. IEEE, 2007.
- [34] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.
- [35] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.
- [36] E. Parisotto and R. Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. *arXiv preprint arXiv:1702.08360*, 2017.
- [37] R. C. Prim. Shortest connection networks and some generalizations. *Bell Labs Technical Journal*, 36(6):1389–1401, 1957.
- [38] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. Kelly, and A. J. Davison. Slam++: Simultaneous localisation and mapping at the level of objects. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1352–1359, 2013.
- [39] T. Sattler, B. Leibe, and L. Kobbelt. Efficient & effective prioritized matching for large-scale image-based localization. *IEEE transactions on pattern analysis and machine intelligence*, 2016.
- [40] T. Sattler, W. Maddern, A. Torii, J. Sivic, T. Pajdla, M. Pollefeys, and M. Okutomi. Benchmarking 6dof urban visual localization in changing conditions. *arXiv preprint arXiv:1707.09092*, 2017.
- [41] L. Tai, G. Paolo, and M. Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 31–36. IEEE, 2017.
- [42] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT press, 2005.
- [43] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010, 2017.
- [44] F. Walch, C. Hazirbas, L. Leal-Taixé, T. Sattler, S. Hilsenbeck, and D. Cremers. Image-based localization with spatial lstms. *CoRR*, abs/1611.07890, 2016.
- [45] S. Wang, R. Clark, H. Wen, and N. Trigoni. Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 2043–2050. IEEE, 2017.
- [46] T. Weyand, I. Kostrikov, and J. Philbin. Planet-photo geolocation with convolutional neural networks. In *European Conference on Computer Vision*, pages 37–55. Springer, 2016.
- [47] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger. Elasticfusion: Real-time dense slam and light source estimation. *The International Journal of Robotics Research*, 35(14):1697–1716, 2016.
- [48] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard. Deep reinforcement learning with successor features for navigation across similar environments. *arXiv preprint arXiv:1612.05533*, 2016.
- [49] J. Zhang, L. Tai, J. Boedecker, W. Burgard, and M. Liu. Neural SLAM. *CoRR*, abs/1706.09520, 2017.
- [50] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3357–3364. IEEE, 2017.