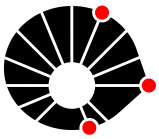
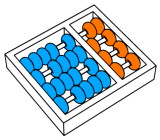




Deep Learning for Visual Odometry

L. Cartolano *E. Colombini*

Relatório Técnico - IC-PFG-20-16
Projeto Final de Graduação
2020 - Agosto

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Deep Learning for Visual Odometry

Luiz Cartolano* Esther Colombini†

Abstract

Odometry is the process of estimating change in position over time through data from motion sensors. However, the classical approaches for solving the odometry problem are usually based on the robot's kinematics, so they can be easily induced to fail when either the motion of the robot or the environment is too challenging. In robotics and computer vision, Visual Odometry is the process of determining the position and orientation of a robot by analyzing the associated camera images. It has been used in various robotic applications, such as on the Mars Exploration Rovers. Recently, novel VO methods have employed deep neural networks or attention models to improve or even replace the entire VO algorithm pipeline. In this project, we aim to evaluate these algorithms and study the fitness of these models under simulated images obtained from the Tartan Air '[1] dataset.

1 Introduction

Visual Odometry (VO) is one of the essential techniques for pose estimation and robot localization nowadays. The classical approach for VO systems is shown in Figure 1, which typically consists of camera calibration, feature detection, feature matching, outlier rejection, scale estimation, and local optimization, has been developed and broadly recognized as a golden rule to follow. Although the state-of-the-art algorithms based on this pipeline have

*Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP.
†Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP.

shown excellent performance, they are usually hard-coded. Each module of the pipeline is adapted to the hardware to achieve impressive performance.

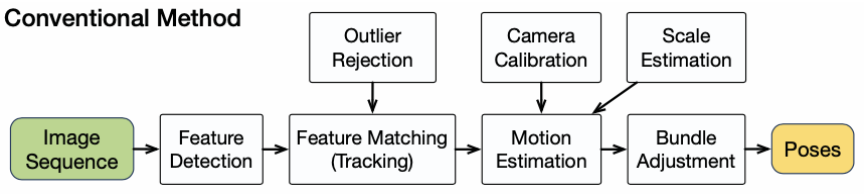


Figure 1: Classical pipeline for Visual Odometry.

In parallel, in the last years Deep Learning (DL) [2] has been dominating many computer vision tasks with spectacular results. Unfortunately, for the VO problem, this has not fully arrived yet, walking in short steps. In this work, we are going to analyze some approaches that use DL for the VO problem. Most of them employ Deep Recurrent Convolutional Neural Networks to solve the VO problem. One of the first articles that address the solution is DeepVO, [3], which pipeline is shown in Figure 2.

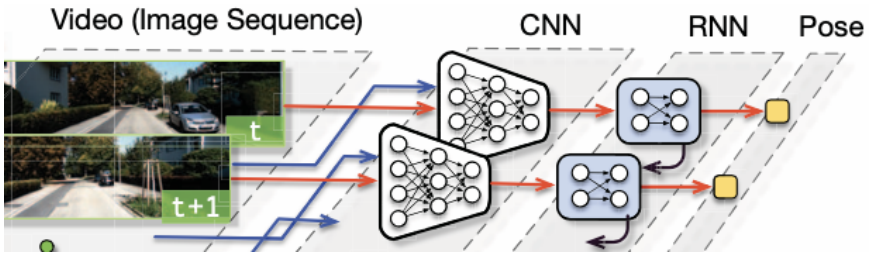


Figure 2: DeepVO pipeline for Visual Odometry. Extracted from [3].

Another approach that has been gaining attention is using a visual saliency map to increase the performance of state-of-the-art algorithms. The concept behind them is quite simple, as we will detail later. SalientDSO [4] is overviewed in Figure 3 and it follows this approach.

For this work, we aim to focus our efforts on analyzing both solutions, with and without saliency, under specific criteria. Then, we want to retrain the models using images obtained from the TartanAir dataset [1]. After obtaining the new models, we will redo the analysis,

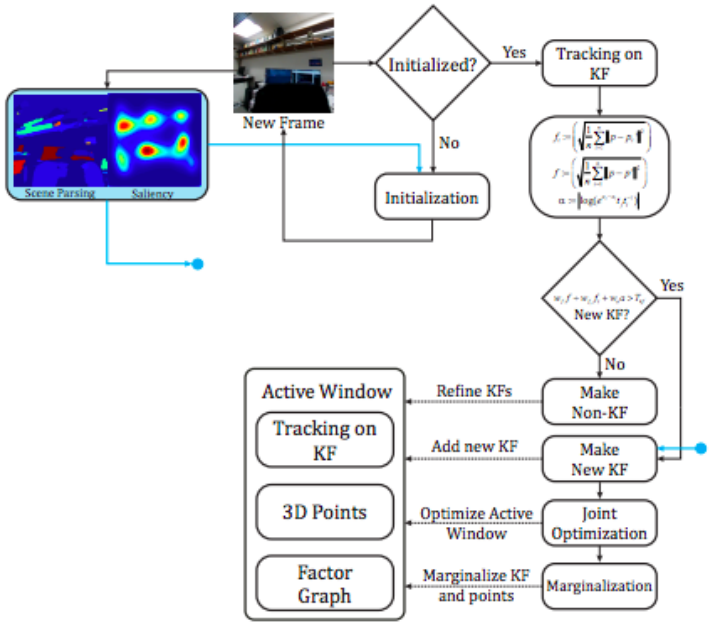


Figure 3: Algorithmic overview of SalientDSO. Extracted from [4].

and finally, we will fine-tune the evaluated models.

This project is organized as follows: Section 2 presents how the work was organized to solve the proposed problem. Section 3 shows previous papers that address the same problem. Section 4 describes the tests done and the methodologies covered, along with some basic concepts. Detailed experiments, along with quantitative and qualitative results, are given in Section 5. Finally, we present the conclusions at Section 6.

2 Proposed Work

In this work, we aim to evaluate how well Deep Learning-based models can predict the robots pose based on input images. To reach the final objective, we divided the work into three stages, represented by:

- **STEP 1: Literature review.** At this stage, we will study DeepVO [3] and SalientDSO [4], particularly how tha later implementats its attentional system for solving the prob-

lem.

- **STEP 2: Systems Performance Evaluation.** After the literature review, we will evaluate both systems under KITTI [5], EuRoC [6] and Tartan Air [1] datasets, using the original pre-trained weights of the networks.

- **STEP 3: Retrain the DL models.** After evaluating the systems with the original weights, we retrained the models under the Tartan Air [1] dataset and then reevaluated the system under the previously mentioned datasets to compare the obtained results.

3 Related Work

In this section, we aim to introduce the results obtained from the literature review. Presenting, with more details, the *DeepVO* and *SalientDSO* approaches, the main related works we are analyzing.

3.1 DeepVO

DeepVO [3] presents a novel end-to-end framework for monocular VO by using deep Recurrent Convolutional Neural Networks (RCNNs) [7]. Since it is trained and deployed in an end-to-end manner, it infers poses directly from a sequence of raw RGB images (videos) without adopting any module in the conventional VO pipeline. Based on the RCNNs, it not only automatically learns effective feature representation for the VO problem through Convolutional Neural Networks, but also implicitly models sequential dynamics and relations using deep Recurrent Neural Networks. The framework has been tested under the Kitti Dataset [5].

The model is mainly composed of a CNN based feature extraction, as previously presented in [8], and a RNN based sequential modeling, as shown in [9]. The architecture of the proposed VO system is shown in Figure 4. It takes a monocular image sequence as input where two consecutive images are stacked together to form a tensor for the deep RCNN to

learn how to extract motion information and estimate poses. The VO system develops over time and estimates new poses as images are captured.

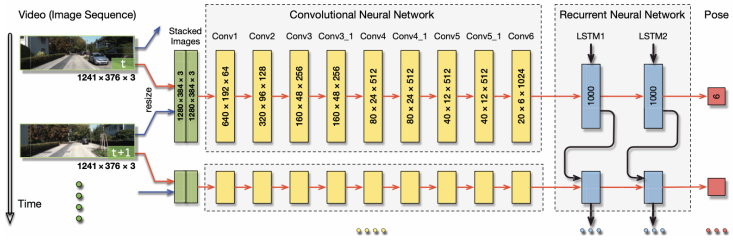


Figure 4: Algorithmic overview of SalientDSO.

The image feature extraction is done by a CNN, which configuration is shown in Figure 5. The CNN takes as input raw RGB images instead of pre-processed counterparts, such as optical flow or depth images, because the network is trained to learn an efficient feature representation with reduced dimensionality for the VO.

Layer	Receptive Field Size	Padding	Stride	Number of Channels
Conv1	7×7	3	2	64
Conv2	5×5	2	2	128
Conv3	5×5	2	2	256
Conv3_1	3×3	1	1	256
Conv4	3×3	1	2	512
Conv4_1	3×3	1	1	512
Conv5	3×3	1	2	512
Conv5_1	3×3	1	1	512
Conv6	3×3	1	2	1024

Figure 5: Algorithmic overview of SalientDSO.

Following the CNN, a deep RNN is designed to conduct sequential learning, i.e., to model dynamics and relations among a sequence of CNN features. Since the RNN can model dependencies in a sequence, it is well suited to the VO problem, which involves the temporal model (motion model) and sequential data (image sequence).

To learn the hyperparameters θ of the DNNs, the Euclidean distance between the ground

truth pose (p_k ,φ_k) at time k and its estimated one ($\hat{\mathrm{p}}_k$,$\hat{\varphi}_k$) is minimised. The loss function is composed of Mean Square Error (MSE) of all positions p and orientations φ:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{t} \|\hat{p_k} - p_k\|_2^2 + k \cdot \|\hat{\varphi_k} - \varphi_k\|_2^2 \tag{1}$$

This work presents a novel end-to-end monocular VO algorithm based on Deep Learning, that does not depend on any module in the conventional VO algorithms (even camera calibration) for pose estimation, and it is trained in an end-to-end manner, there is no need to carefully tune the parameters of the VO system.

3.2 SalientDSO

SalientDSO's [4] framework can be split into two major parts. A VO backbone, responsible for initializing and tracking camera pose and optimizing all model parameters. The other part includes the pre-processing step, which involves the saliency prediction and scene parsing using deep Convolutional Neural Networks (CNNs) and later using these outputs to select features and points. The algorithmic overview of SalientDSO is shown in Figure 2.

The work adopts DSO[10] as the backbone VO. DSO proposed a direct sparse model to optimize all parameters and perform windowed bundle adjustment jointly. However, the proposed work uses a specific point selection, that will be explained later.

The point selection proposed on SalientDSO [4] is based on visual saliency and scene parsing. Visual saliency is defined as the amount of attention a human would give to each pixel in an image. The paper adopted the SalGAN [11] for saliency prediction. SalGAN introduced the use of Generative Adversarial Network for saliency prediction. However, the saliency produced is not very robust to viewpoint and illumination changes. So, besides it, the framework utilize semantic information to filter the saliency, with the idea of weighting down the saliency of uninformative regions.

To extract the semantic information, the authors adopt the Pyramid Scene Parsing

[12]. In brief, the framework presents a deep neural network for pixel-level prediction tasks. PSPNet uses CNN layers to extract features. Then a pyramid parsing module is applied to harvest different sub-region representation, followed by up-sampling and concatenation layers to form the final feature representation. The final features are then fed into more CNN layers to obtain a pixel-level prediction.

Ultimately, the SalientDSO [4] select points based on saliency. An image is split into $K \times K$ patches. For any patch $M_i \in M$ the sampling weight sw_i is computed as

$$sw_i = median\{\hat{S}_j^{final}, \forall j \in M_i\} + s_{smooth}. \tag{2}$$

Once a patch M_i has been selected, it is further split into $d \times d$ blocks. For each block, the pixel with the highest gradient is only selected if it surpasses the region-adaptive threshold. With this strategy, points which are well distributed in the salient region are selected.

In short, the paper introduces the philosophy of attention and fixation to visual odometry, which brings the concept of attention and fixation based on visual saliency into Visual Odometry to achieve robust feature selection.

4 Materials and methods

In this section, we will explain the problem we are trying to solve in more detail, presenting the datasets and frameworks employed.

4.1 Datasets

For this work, we used three datasets to train and evaluate the frameworks studied. In this section, we aim to present the used datasets and give some overview of them.

4.1.1 KITTI Vision Benchmark

The KITTI dataset [5] has been recorded from a moving platform while driving in and around Karlsruhe, Germany. It includes camera images, laser scans, high-precision GPS measurements, and IMU accelerations from a combined GPS/IMU system. The dataset can be downloaded from http://www.cvlibs.net/datasets/kitti.

The raw data set is divided into the categories' Road',' City',' Residential',' Campus', and' Person'. Each sequence provides the raw data, object annotations in the form of 3D bounding box tracks, and a calibration file.

The calibration parameters are stored using the following notation:

- $s^{(i)} \in \mathbb{N}^2$ - original image size.
- $K^{(i)} \in \mathbb{R}^{3\times3}$ - calibration matrices.
- $s_{rect}^{(i)} \in \mathbb{N}^2$ - image size after rectification.

4.1.2 EuRoC MAV Datasets

The EuRoC datasets presented in [6] were recorded in the context of the European Robotics Challenge (EuRoC), to assess the contestant's visual-inertial SLAM and 3D reconstruction capabilities on MAVs. The datasets can be downloaded from http://projects.asl.ethz.ch/datasets/doku.php?id=kmavvisualinertialdatasets.

Each dataset contains one or more "sensor systems". Every sensor comes with a *sensor.yaml*, a file that specifies its calibration parameters, and a *data.csv* file containing the sensor measurements or points to data files.

Despite careful design and execution of the data collection experiments, few different issues are presented, like the fact that some of the datasets exhibit very dynamic motions, which are known to deteriorate the measurement accuracy of the laser tracking device.

4.1.3 Tartan Air

Collecting data in the real world often relies on an elaborate sensor setup and careful calibration. With recent advances in computer graphics, many synthetic datasets have been proposed. The Tartan Air dataset [1] collects a large dataset using photo-realistic simulation environments. They try to minimize the problems of converting simulation two real-life by increasing diversity on the collected images.

In this work, a special focus is on challenging environments with changing light conditions, low illumination, adverse weather, and dynamic objects. Getting a dataset with the following characteristics:

- a large size and high diversity
- realistic lighting
- multimodal data and ground truth labels
- diversity in motion patterns
- challenging scenarios

They adopt the Unreal Engine and collect the data using the AirSim plugin developed by Microsoft [13].

4.2 DeepVO Implementation

4.2.1 How to Use

In order to evaluate the DeepVO [3] framework we used an open source code distributed at `https://github.com/ChiWeiHsiao/DeepVO-pytorch`. In this implementation, the authors used *Python* along with the *PyTorch* [14] framework to reproduce the paper. We also use, in our first experiment, the original trained model provided by the authors at `https://drive.google.com/file/d/1l0s3rYWgN8bL0Fyofee8IhN-0knxJF22/view`.

The code's use has different requirements on whether you are only evaluating a sequence or training the neural network. For the first case, you need only the images; for the second, the target poses (ground truth) will also be needed.

In the pre-processing stage, the script converts the ground truth poses into six floats (Euler angle + translation) [15]. The code also uses the pre-trained weight of FlowNet [16](CNN part). Note that this pre-trained FlowNet model assumes that the RGB value range is $[-0.5, 0.5]$ so, some pre-processing of the images is also required.

The last detail to use the code is to specify the paths and changes hyperparameters in the file *params.py*. Adjusting the *batch_size*, the image size (img_w, img_h) and the *pin_mem* (which must be set to false in the absence of a GPU).

Lastly, *main.py* file is used for training the model, whereas *test.py* to predict poses of a sequence.

4.2.2 Implementation Details

In order to implement the same *RCNN* model showed in Figure 2, the authors used the *PyTorch* [14] library. All convolutionals layers have a *Conv2d* structure, followed by a *BatchNorm2d* (batch normalization), and then a *LeakyReLU* function (a Rectified Linear Activation Function) and in the end, a *Dropout*.

Convolution is the simple application of a filter to an input that results in an activation. The batch normalization reduces the amount by what the hidden unit values shift around (covariance shift). The activation function is responsible for transforming the summed weighted input from the node into the node's activation or output for that input. In the dropout stage, individual nodes are either dropped out with probability $1 - p$ or kept with probability p, so that a reduced network is left. Incoming and outgoing edges to a dropped-out node are also removed.

For the RNN part of the network, they used two *LSTM* layers followed by a linear output, that receives 100 features as input and outputs 6 features (the pose). The *Long*

Short-Term Memory (LSTM) networks are a modified version of recurrent neural networks, which makes it easier to remember past data in memory.

The loss function employed is the *Mean Square Error*, defined by Equation 1.

As the optimizer, they used *Adagrad*, an algorithm for gradient-based optimization. It works by adapting the learning rate to the parameters, performing smaller updates (i.e., low learning rates) for parameters associated with frequently occurring features, and more extensive updates (i.e., high learning rates) for parameters related to infrequent features.

4.3 SalientDSO Implementation

To evaluate the SalientDSO [4] framework we used an open-source code distributed at `https://github.com/prgumd/SalientDSO`. The code is implemented using *C++* and mix three different frameworks to reproduce the paper.

The first thing to use the code is to execute the setup, download the prerequisites, and compile it. At this step, some minor changes in the *OpenCV* library may be necessary. Also, if you are using a *MacOS* system, you will probably need to add some changes to the *CMake* file.

We need to execute both SalGAN and PSPNet in every image of all datasets and store results to use the given code. Once we had the original images, the saliency images, and the segmentation images, we were finally able to evaluate the system. In the next subsections, we aim to explain all the three major components that build SalientDSO and how to use them with a little bit more detail.

4.3.1 DSO

The DSO[10] is used as the backbone VO for the SalientDSO framework. Since the DSO is a direct odometry method, it demands some extra information to execute.

First, it demands a file specifying the geometric camera calibration file. Usually, for some datasets, the DSO also needs photometric calibration, but, for the ones we used at

this work, they do not exist, then, it's necessary to use the $mode = 1$ while executing the code.

Some adaptations on the DSO parameters will also be necessary for the code to execute properly.

4.3.2 SalGAN

The implemented code mix the DSO solution along with the SalGAN and the PSPNet, but, it does not merge all the codes at one pipeline. This means that we need to execute both separately on our dataset before executing the SalientDSO.

The SalGAN [11] code is available at https://github.com/imatge-upc/salgan. The SalGAN is a deep convolutional neural network for visual saliency prediction trained with adversarial examples. The first stage of the network consists of a generator model whose weights are learned by back-propagation computed from a binary cross-entropy (BCE) loss over downsampled versions of the saliency maps. The resulting prediction is processed by a discriminator network trained to solve a binary classification task between the saliency maps generated by the generative stage and the ground truth ones.

The pre-trained weights for the SalGAN are distributed at https://s3.amazonaws.com/lasagne/recipes/pretrained/imagenet/vgg16.pkl.

In this work, we did use the pre-trained weights of SalGAN to obtain the saliency images of our sequences and store them to be used ahead.

4.3.3 PSPNet

The PSPNet code, available at https://github.com/hellochick/PSPNet-tensorflow, is the python implementation to the Pyramid Scene Parsing [12].

We used the pre-trained weights available at https://drive.google.com/drive/folders/1S90PWzXEX_GNzulG1f2eTHvsruITgqsm?usp=sharing to execute the scene parsing on our datasets. It is important to highlight the existence of two different models, trained in differ-

ent datasets. The *Cityscapes* is trained using outside images, while the *ade20k* uses indoor ones. Since the major part of our datasets was recorded outdoor, we choose the model trained at the *Cityscapes* dataset.

4.4 Methodology

In this section, we will explain how the data was obtained and evaluated, showing, and explaining the used metrics with details.

The poses for all sequences, obtained under the application of both frameworks, were evaluated under three distinct criteria. The chosen criteria were trajectory, absolute trajectory error (ATE), and relative pose error (RPE).

4.4.1 Trajectory

A trajectory is a path that an object with mass in motion follows through space as a function of time. In classical mechanics, a trajectory is defined by Hamiltonian mechanics via canonical coordinates; hence, a complete trajectory is simultaneously defined by position and momentum.

To be possible to visually compare the estimated trajectory and the ground truth trajectory for the sequences we evaluate, we decide to plot both under the same map and scales, facilitating the visual comparison of both.

4.4.2 Absolute Trajectory Error - ATE

Global consistency can be evaluated by comparing the absolute distances between the estimated and the ground truth trajectory. As both trajectories can be specified in arbitrary coordinate frames, they first need to be aligned. This can be achieved in closed form using the Horn method.

Whereas the estimated trajectory is given by $P_{1:n}$, the ground truth trajectory is $Q_{1:n}$, and the Horn transformation is S. The absolute trajectory error at time step i can be

computed as

$$F_i = Q_i^{-1} S P_i \tag{3}$$

We propose to evaluate the root mean squared error over all time indices of the translational components due to our trajectory. The RMSE is given by

$$RMSE(F_{i:n}, \Delta) = (\frac{1}{n} \sum_{i=1}^{n} ||trans(F_i)||^2)^{1/2} \tag{4}$$

4.4.3 Relative Pose Error - RPE

The relative pose error measures the local accuracy of the trajectory over a fixed time interval. Considering that the estimated trajectory is given by $P_{1:n}$ and the ground truth trajectory is $Q_{1:n}$. We define the relative pose error at time step i as

$$E_i = (Q_i^{-1} Q_{i+\Delta})^{-1} (P_i^{-1} P_{i+\Delta}) \tag{5}$$

From a sequence of n camera poses, we obtain in this way $m = n - \Delta$ individual relative pose errors along the sequence. From these errors, we propose to compute the root mean squared error (RMSE) overall time indices of the translational component as

$$RMSE(E_{i:n}, \Delta) = (\frac{1}{m} \sum_{i=1}^{m} ||trans(E_i)||^2)^{1/2} \tag{6}$$

where $trans(E_i)$ refers to the translational components of the relative pose error E_i.

Furthermore, the time parameter Δ needs to be chosen. Since our work uses visual odometry systems that match consecutive frames, we choose $\Delta = 1$.

In this section, we are also present a short introduction explaining the process of retraining the DeepVO under the Tartan Air sequences. To be able to retrain the model was necessary

to create a special preprocessing script that had as function convert the Tartan Air poses format $[tq]$ to the KITTI ones $[Rt]$.

So it was only necessary to download the sequences and run the script to execute the code. The training was executed under the following sequences: abandonedfactory, abandonedfactory_night, office, amusement, ocean, and japanesealley. These were chosen to create a dataset that mixes indoor and outdoor environments, also with a lot of or little lighting. Besides that, the images were mostly composed of very different scenarios.

Finally, it is important to note that, since the training process requires high computational power, the structure provided by *Google Colab* [17] was used.

5 Results and Discussion

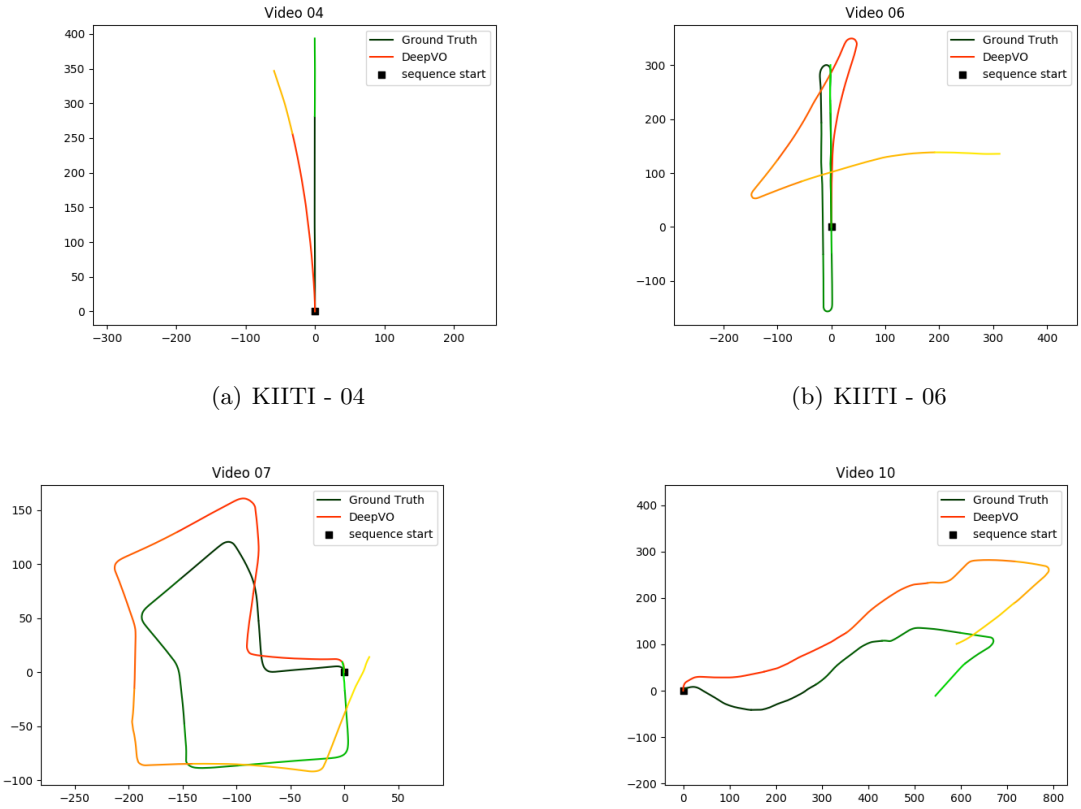
In this section, we present the results obtained by running the previously discusses models in the selected dataset. We also present the corresponding trajectories and metrics that resulted from these experiments.

5.1 Results

5.1.1 Results - Original Models

The first results obtained in this work came from the second step of Section 2. We were supposed to evaluate both frameworks under the KITTI, EuRoC, and TartanAir datasets.

At first, we evaluate the *DeepVO* framework [3]. For the DeepVO we were able to run the code and obtain results for all datasets, the obtained trajectories can be seen in Figures 6 and 7, the *ATEs* are available in Table 1 while the *RPEs* are in Table 2.



(a) KIITI - 04

(b) KIITI - 06

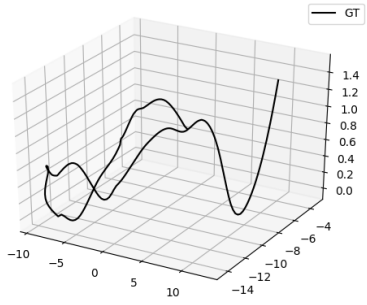
(c) KIITI - 07

(d) KIITI - 10

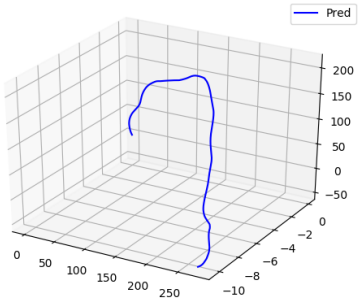
Figure 6: DeepVO trajectories for KITTI dataset.

Sequence	**ATE - X**	**ATE - Y**	**ATE - Z**
KITTI - Route 04	3.6389	0.2373	-
KITTI - Route 06	72.4715	5.3956	-
KITTI - Route 07	16.2760	3.9435	-
KITTI - Route 10	44.4991	3.1372	-
EuRoC - Route 01	44.4708	2.6125	40.6930
TartanAir - Seaside Town	19.7445	1.5691	76.1676
TartanAir - Hospital	91.6579	3.0211	83.0758

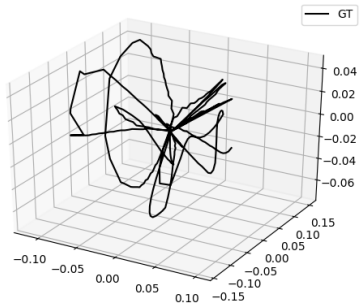
Table 1: DeepVO - Absolute Trajectory Error.



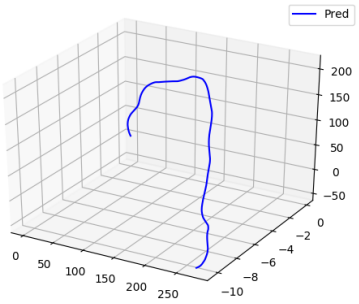
(a) TartanAir - Seaside - Ground Truth



(b) TartanAir - Seaside - Prediction



(c) TartanAir - Hospital - Ground Truth



(d) TartanAir - Hospital - Prediction

Figure 7: DeepVO trajectories for TartanAir dataset.

Sequence	**RPE - Translation**	**RPE - Rotation**
KITTI - Route 04	0.3878	0.0044
KITTI - Route 06	0.9912	0.0065
KITTI - Route 07	0.4367	0.0077
KITTI - Route 10	0.4714	0.0079
EuRoC - Route 01	0.7733	0.0604
TartanAir - Seaside Town	1.1242	0.0480
TartanAir - Hospital	1.1887	0.0299

Table 2: DeepVO - Relative Pose Error.

Following the analyzes performed in the second stage, we try to evaluate the same datasets under the SalientDSO framework [4]. Unfortunately, due to reasons we will discuss later, we could not execute the framework under the EuRoC and TartanAir datasets. The obtained trajectories under KITTI can be seen in Figure 8, the *ATEs* are available in Table 3 while the *RPEs* are in Table 4.

Sequence	**ATE - X**	**ATE - Y**	**ATE - Z**
KITTI - Route 04	3.4078	6.9245	-
KITTI - Route 06	40.0496	9.7292	-
KITTI - Route 07	51.2126	0.7585	-
KITTI - Route 10	163.4615	6.3018	-

Table 3: SalientDSO - Absolute Trajectory Error.

Sequence	**RPE - Translation**	**RPE - Rotation**
KITTI - Route 04	1.4146	0.0068
KITTI - Route 06	1.1308	0.0098
KITTI - Route 07	0.4944	0.0160
KITTI - Route 10	0.6448	0.0178

Table 4: SalientDSO - Relative Pose Error.

5.1.2 Results - DeepVO Training

In Figure 9 we can see the training losses of DeepVO using sequences obtained at the Tartan Air simulator. The figure shows the training loss for 50 epochs using the pre-trained weights of FlowNet [16].

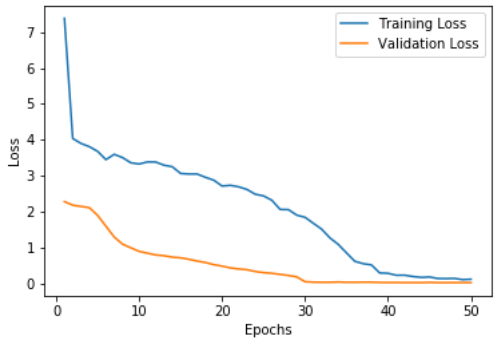


Figure 9: Training losses for the DeepVO system under the Tartan Air sequences.

5.1.3 Results - Tartan Air Model

After the process of retraining the DeepVO model, we evaluate the framework again. The obtained trajectories can be seen in Figures 10 and 11, the *ATEs* are available in Table 5 while the *RPEs* are in Table 6.

Sequence	**ATE - X**	**ATE - Y**	**ATE - Z**
KITTI - Route 04	4.7289	1.2373	-
KITTI - Route 06	27.4715	33.3956	-
KITTI - Route 07	11.2760	7.9435	-
KITTI - Route 10	44.4991	3.1372	-
TartanAir - Seaside Town	0.8346	3.0557	0.6134
TartanAir - Hospital	1.1106	4.7440	1.7134

Table 5: DeepVO - Absolute Trajectory Error.

Sequence	RPE - Translation	RPE - Rotation
KITTI - Route 04	2.0191	0.0029
KITTI - Route 06	1.1506	0.0844
KITTI - Route 07	0.7140	0.0576
KITTI - Route 10	0.8387	0.0971
TartanAir - Seaside Town	0.8346	3.0557
TartanAir - Hospital	0.0575	0.1074

Table 6: DeepVO - Relative Pose Error.

5.2 Discussion

The first thing we had to do before retraining the models under the Tartan Air dataset [1] was to analyze the obtained results with the original models given by DeepVO [3] and SalientDSO [4]. The results are available at Subsection 5.1.1.

Analyzing the DeepVO results for the model trained under the KITTI dataset [5] it is possible to conclude that the results are nothing more than average. If we pay attention to the trajectories shown in Figure 6, it is possible to realize that the results worsen as the trajectories show more direction changes. That observation is confirmed by the results shown in Figure 7, in which images simulate the ones obtained by drones. The Tartan Air sequences also show us another big weakness at DeepVO the projections for the *z-axis*. A fact we can observe in both trajectories and metrics, due to the very high *ATE* over the *z-axis*. A reasonable explanation for that lies in that the KITTI images are obtained from a car, so the original DeepVO model was entirely trained with modifications only on-axis x and y. Another possible justification for the lack of good results is the lack of diversity in the scenarios used for training since the sequences used are very similar. A problem we aim to solve with the Tartan Air [1] sequences.

Along with the DeepVO, we also execute our sequences on the SalientDSO [4] framework. However, unlike the first framework, as we explained in Subsection 3.2, the SalientDSO

adopts DSO [10] as the backbone VO. So, it depends on the intrinsic parameters of the camera to show results. That point is a problem the main justification for the bad results obtained if compared with the ones presents at paper [4]. To obtain all the poses for any sequence, we change some DSO parameters, which probably result in the high *ATE's* and *RPE's*. Another huge problem we face was the lack of conditions to run the framework at sequences obtained via drones. Unfortunately, contrary to what was planned, it was not possible to retrain the models for the framework using the Tartan Air [1] sequences, since that would mean retrain SalGAN [11] and PSPNet [12], and we consider it outside the scope of what was planned to study.

As shown before, we present in Figure 9 our training and validation loss for the DeepVO under new training scenarios. The model appears to have a good fit as a result, since both curves decrease similarly to the number of epochs passes. This assumption will be confirmed, or not, by the results obtained for the sequences that were not used during the training process.

After retraining the model under the Tartan Air [1] we redid the experiments done in Step 1. Analyzing the obtained results, it is evident that the results were much better for the Tartan Air sequences, but, on the other hand, a little bit worse for the KITTI[5] sequences. Looking only for the trajectories presented in Figure 10, we can see that for the sequences with less drift, the results are quite similar to the ones presented at the original paper [3]. For the sequences shown in Figure 11 it is evident that the results are a lot better than the ones shown in Figure 7 but far from expected. If we also look to the *ATE* and *RPE* results, it is possible to realize that the values decreased only for the Tartan Air sequences.

In short, the obtained results did not reach our expectations, which is not a problem for this project, as long as we can understand why and how to improve in future iterations. One first point we can note is related to the number of images used compared to the original paper. Even using all available sequences from Tartan Air [1] we could gather around four

thousand images, the original paper uses around seven thousand. So, for future works, we must collect more images for a larger dataset.

Another conclusion that we can reach came from the results obtained from the KITTI sequences. Then, a future protocol may include a hybrid between the datasets, blending simulated images (like the ones from Tartan Air [1]) with real images (like the ones from KITTI [5] and EuRoC [6], to bring more robustness to the system.

Lastly, another point we must study and evaluate in future works is to add image saliency models (like SalGAN [11]) to the DeepVO [3] pipeline. Since we are working with an odometry system that is based on images for location, there is a big chance of saliency to improve the pose predict process. A good indicator of this is the good results of SalientDSO [4].

6 Conclusions

In general, the work did not show impressive results. But, it was essential and fulfilled its purpose of evaluating the different models of deep learning for Visual Odometry. Making it possible to understand some of the state-of-the-art algorithms, knowing their strengths and weaknesses, and, more important, it was possible to figure out how to improve the frameworks for future iterations.

Some positive points we can extract from this work are the fact that the use of Deep Learning techniques is plausible to the VO problem, and also showed that the simulator images are a good path to make a dataset for the problem. The work also shows us, based on the SalientDSO [4] evaluation, that saliency techniques are, indeed, a good option for Visual Odometry.

Weaknesses of the work, which need to be improved in future iterations, are the number of images in the dataset. For future work, we must collect more images and mix simulated images with real-world images. Also, some improvements on the neural network, to ensure a better fit for the model. We could also improve the system's effectiveness is to add some

SLAM [18] or a salient model to preprocessing the images.

References

[1] W. Wang, D. Zhu, X. Wang, Y. Hu, Y. Qiu, C. Wang, Y. Hu, A. Kapoor, and S. Scherer, "Tartanair: A dataset to push the limits of visual slam," 2020. 1, 2, 4, 9, 20, 21, 22

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015. 2

[3] S. Wang, R. Clark, H. Wen, and N. Trigoni, "Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 2043–2050. 2, 3, 4, 9, 15, 20, 21, 22

[4] H.-J. Liang, N. J. Sanket, C. Fermüller, and Y. Aloimonos, "Salientdso: Bringing attention to direct sparse odometry," *IEEE Transactions on Automation Science and Engineering*, 2019. 2, 3, 6, 7, 11, 18, 20, 21, 22

[5] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013. 4, 8, 20, 21, 22

[6] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, 01 2016. 4, 8, 22

[7] M. Liang and X. Hu, "Recurrent convolutional neural network for object recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3367–3375. 4

[8] Y. Chen, H. Jiang, C. Li, X. Jia, and P. Ghamisi, "Deep feature extraction and classification of hyperspectral images based on convolutional neural networks," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 54, no. 10, pp. 6232–6251, 2016. 4

[9] J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio, "A recurrent latent variable model for sequential data," in *Advances in neural information processing systems*, 2015, pp. 2980–2988. 4

[10] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 3, pp. 611–625, 2017. 6, 11, 21

[11] J. Pan, C. C. Ferrer, K. McGuinness, N. E. O'Connor, J. Torres, E. Sayrol, and X. Giro-i Nieto, "Salgan: Visual saliency prediction with generative adversarial networks," *arXiv preprint arXiv:1701.01081*, 2017. 6, 12, 21, 22

[12] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2881–2890. 7, 12, 21

[13] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and service robotics*. Springer, 2018, pp. 621–635. 9

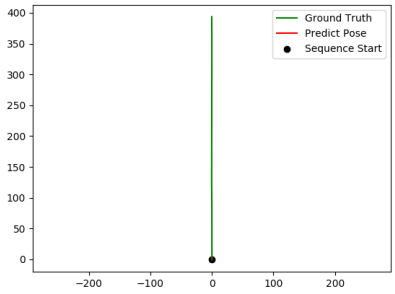
[14] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. Alche-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf 9, 10

[15] J. Diebel, "Representing attitude: Euler angles, unit quaternions, and rotation vectors," *Matrix*, vol. 58, no. 15-16, pp. 1–35, 2006. 10

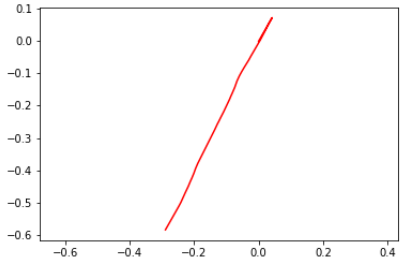
[16] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2462–2470. 10, 18

[17] E. Bisong, "Google colaboratory," in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. Springer, 2019, pp. 59–64. 15

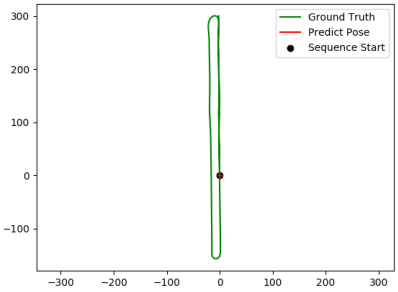
[18] H. Lim, J. Lim, and H. J. Kim, "Real-time 6-dof monocular visual slam in a large-scale environment," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 1532–1539. 23



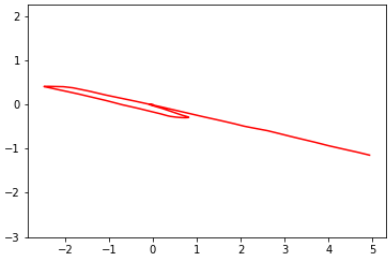
(a) KITTI - 04 - Ground Truth



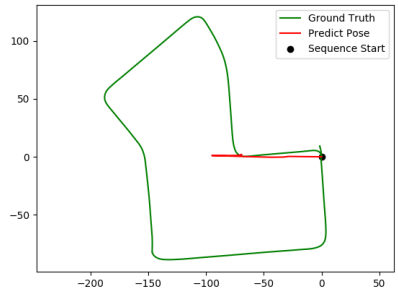
(b) KITTI - 04 - Prediction



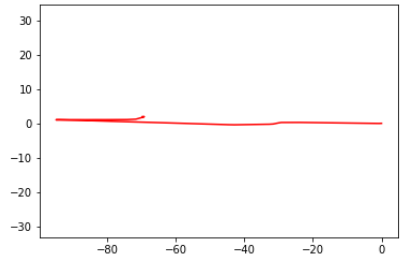
(c) KITTI - 06 - Ground Truth



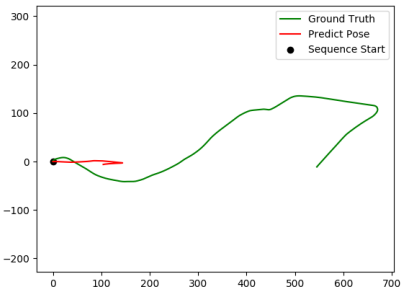
(d) KITTI - 06 - Prediction



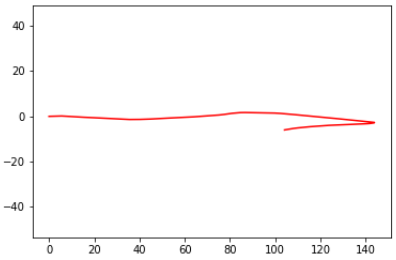
(e) KITTI - 07 - Ground Truth



(f) KITTI - 07 - Prediction

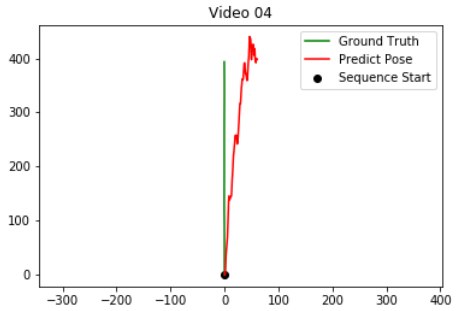


(g) KITTI - 10 - Ground Truth

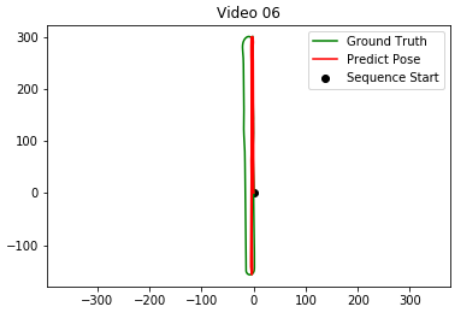


(h) KITTI - 10 - Prediction

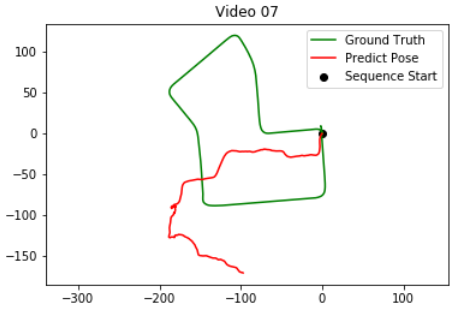
Figure 8: SalientDSO trajectories for KITTI dataset.



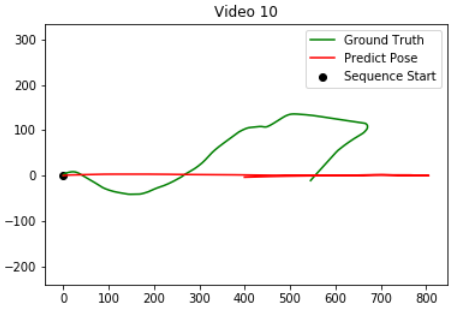
(a) KIITI - 04



(b) KIITI - 06

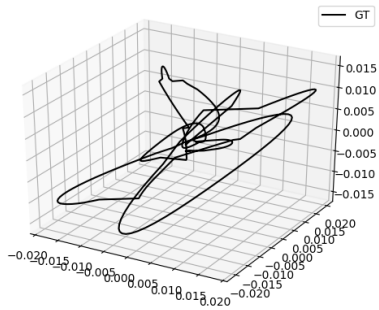


(c) KIITI - 07

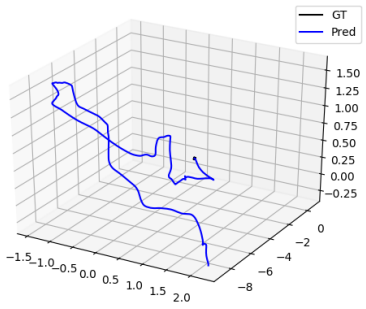


(d) KIITI - 10

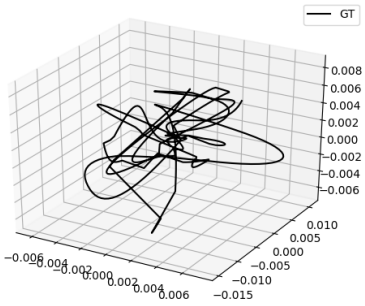
Figure 10: DeepVO trajectories for KITTI dataset.



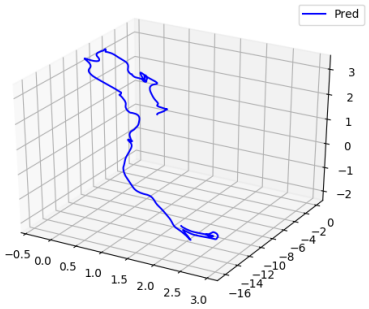
(a) TartanAir - Seaside - Ground Truth



(b) TartanAir - Seaside - Prediction



(c) TartanAir - Hospital - Ground Truth



(d) TartanAir - Hospital - Prediction

Figure 11: DeepVO trajectories for TartanAir dataset.