Final Project - MC907 A

LUIZ EDUARDO CARTOLANO,*

* *Computer Engineer - Undergraduate*
E-mail: 1183012@students.ic.unicamp.br

***Abstract* – 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 a wide variety of robotic applications, such as on the Mars Exploration Rovers. The classical approaches for solve the VO problems are usually based on the kinematics of the robot, so they can be easily induced to fail when either the motion of the robot or the environment are too challenging. In this way, novel VO methods use deep neural networks or attention models to improve or even replace the entire algorithm pipeline. In this project, we aim to brief introduce both algorithms and given sample results of the frameworks. Some obtained results are show in Figures 13, 14, 15 and 16. A video with the model results is available at https://biteable.com/watch/deepvo-2406592.**

***Palavras-chave* – Deep Learning - Visual Odometry - Robotics**

I. INTRODUCTION

Visual Odometry (VO) is one of the most essencial techniques for pose estimation and robot localization now a days. 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 optimisation, has been developed and broadly recognised as a golden rule to follow. Altough the state-of-the-art algorithms based on this pipeline have shown excellent performance, they are usually hard-coded, where each module of the pipeline is adapted to the hardware in order to achieve an amazing performance.

In parallel, in the last years, Deep Learning (DL) [1], 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 present some new approachs that uses DL for the VO problem. Most of them consist at a Deep Recurrent Convolutional Neural Networks to solve the problem. One of the first articles that addresses the solution is shown at [2] and a pipeline is shown at Figure 2.

Another approach that has been gaining attention, is the use of a visual saliency map in order to increase the performance of the state-of-the-art algorithms. The concept is quite simple, as we are going to futher explain, humans perform the task of mapping very differently from how it has been usually done at the classical approaches at robotics [3].

For this work, we aim to focus our efforts at the Deep Learning approach. This paper is organized as follows: Section II presents how the paper was organized in order to solve the problem. Section III describes the tests done and the methodologies covered. The results and analysis are presented in Section IV and finally, the conclusions at Section V.

II. PROPOSED WORK

The proposed work, which can be seen in more detail in 4, calls for the implementation of a project to solve a classical robotic problem. To explain the work in more detail, the next subsections will cover the problem and the tools used to solve it.

A. Problem

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. This work aims to implement a end-to-end Deep Learning technique to preview the robot position at the enviroment.

B. Tools

In order to carry out the proposed implementations it was used as programming language the 3.7 version of Python. In conjunction with it, the Jupyter Notebook platform was used, since with it's easier to track the results of the applied algorithms and also because it allows the execution of specific pieces of code, which ensures greater flexibility in using the various algorithms used.

III. MATERIALS AND METHODS

In this section we will explain in more detail the problem modeling, ie the machine learning techniques implemented in the project and how they were specified.

A. Related Work - Deep VO

The paper [2] presents a novel end-to-end framework for monocular VO by using deep Recurrent Convolutional Neural Networks (RCNNs) [5]. 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 [1], showing great results.

The model is mainly composed of CNN based feature extraction, as previous presented in [6], and RNN based

[1] http://www.cvlibs.net/datasets/kitti/eval_odometry.php

sequential modelling, as shown in [7]. The architecture of the proposed VO system is shown in Figure 3, it takes a monocular image sequence as input. 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.

The image feature extraction of is done by a CNN, which configuration is shown at Figure 4. The CNN takes raw RGB images instead of pre-processed counterparts, such as optical flow or depth images, as input because the network is trained to learn an efficient feature representation with reduced dimensionality for the VO.

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 is capable of modelling dependencies in a sequence, it is well suited to the VO problem which involves 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 :

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

In Figure 5, the losses results of the models are given for both training and validate datasets. As shown in the fig., the loss significantly reduces with time, and, once the training and validation losses shown a similar behavior we can also say that is free of overfiting, so, the model is well-fit.

The Figures 6 and 7 show us the obtained maps for the framework for both training and test datasets. As we can see in the figures, the obtained odometry from the system is very close to the ground truth position, showing how great the system performs. The system is analysed according to the KITTI VO/SLAM evaluation metrics, i.e., averaged Root Mean Square Errors (RMSEs) of the translational and rotational errors for all subsequences of lengths ranging. The average RMSEs of the estimated VO are given in Figure 8. Al- though the result of the DeepVO is worst than that of the stereo VISO2 (VISO2_S), it is consistently better than the monocular VISO2 (VISO2_M).

The paper 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. Based on the KITTI VO benchmark, it is verified that it can produce accurate VO results with precise scales and work well in completely new scenarios.

For this project, we aim to reproduce the paper approach using images and poses obtained at the AirSim, an open source simulator for autonomous vehicles built on Unreal Engine/Unity, from Microsoft AI&Research [2].

B. Project Dataset

1) Data Collection: In order to create our own *dataset* to perform the article model we had use the Microsoft AirSim Simulator to generate both images and poses.

Six sequences were generated under three different scenarios: Neighborhood(Figure 9), Africa (Figure 10) and LandscapeMountains(Figure 11).

The simulator were used at the *ComputerVision* mode and the cameras controlled via keyboard. Each sequence has around four hundred pictures, making a total of around two thousand images for the training set.

2) Images Data Prerocessing: The first stage of image preprocessing was eliminate all failed pictures given by the simulator, excluding them from both pictures and pose lists.

After that, as we read the images we convert them to grayscale and normalize their pixels values, in a range that goes from -0.5 to 0.5 in order to be able to use the pretrained weight of FlowNet, all of it were done using the OpenCV library.

At least, we concatenated the pictures two by two, once that the two images will going to work as input for our Neural Network.

3) Poses Data Preprocessing: The AirSim recording mode, as we further explain, give us a text file with the following informations for each image: timestamp, x_position, y_position, z_position, quaternions_x, quaternions_y, quaternions_z, quaternions_w and a filename of the image associated to that pose.

As explained in 8, a quaternion is a four-element vector that can be used to encode any rotation in a 3D coordinate system. Technically, a quaternion is composed of one real element and three complex elements, and it can be used for much more than rotations. For our work, we need the angles to be at an Euler representation. Euler angles provide a way to represent the 3D orientation of an object using a combination of three rotations about different axes.

In order to transform the given angles to an Euler representaion we need to use the following equations:

$$\phi = arctan(\frac{2(ab+cd)}{a^2-b^2-c^2+d^2}) \quad (2)$$

$$\theta = -arcsin(2(bd-ac)) \quad (3)$$

$$\Psi = arctan(\frac{2(ad+bc)}{a^2+b^2-c^2-d^2} \quad (4)$$

This can be easily done in *Python* using the *Scipy* [3] library.

Another thing we had to do in order to process the information was make the first pose our start point, so we subtract all the other poses from the first one value. In addition, in addition we subtract all the poses, with the except of the firts, from the previous pose value, so, the network has to predict only the dislocate between two images.

[2] https://github.com/microsoft/AirSim

[3] https://docs.scipy.org/doc/scipy-1.2.1/reference/generated/scipy.spatial.transform.Rotation.html

C. Deep VO Implementation

In order to simulate Wang's[2] paper for our dataset, we implemented the same RCNN model showed at Figure 2.

For implementing the model we used the PyTorch [4] library. All the convolutionals layers has 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*.

A 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 activation of the node or output for that input. And in the dropout stage, at each training stage, individual nodes are either dropped out of the net 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 CNN part we tested two different configurations, which results will be further commented. We first fitted the model without any previous information and, in a second stage, we loaded pretrained weight of FlowNet network [5].

For the RNN part of the network, we had two *LSTM* layers followed by a linear output, that receives 100 features as input and outputed 6 features (our 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.

To define the loss function we used the Mean Square Error, same as Wang's paper and which equations is defined at Equation 1. The used optimizer was the Adagrad, an algorithm for gradient-based optimization that does just this: It adapts the learning rate to the parameters, performing smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) for parameters associated with infrequent features.

IV. Results and Discussion

In this section we will first present all the results obtained during the implementation of the implemented solution, including tables and graphs generated. Once introduced, they will be discussed and compared.

A. Results

At Figures 12 and 13 we are able to see the training losses of the two different experiences made with the implemented neural network. The Figure 12 shows the training loss for 90 epochs when we didn't use the pretrained weigths of FlowNet, while the Figure 13, shows the results for 150 epochs using the pretrained weigths of FlowNet. These results, as we are going to further discuss were a lot worst than the obtained by [2].

After fit the model we work on validate the obtained weights on a different sequence that were designated as a test dataset. The first measure we did were measure the accuracy of the system. The accuracy were calculated by the follow equation:

$$acc = \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{t} \|\hat{p_k} - p_k\|_2^2 + \|\hat{\varphi_k} - \varphi_k\|_2^2 \qquad (5)$$

And we obtained an accuracy of *54.70%*. We also analyse the individual erros for position (x,y,z) and orientation, obtaining results of *9.81%* and *58.84%*.

Lastly, we can present as a result the odometry provided by our system ploted with the original position, the ground truth one. Figures 14, 15 and 16 shows that results. The blue line on the figures show the ground truth position given by the simulator and the green line the odometry given by our model.

B. Discussion

Firts we are going to compare the training losses obtained in our experiment for two different scenarios under the same dataset. Comparing Figures 12 and 13, it's possible to realize that the pretrained weigths of FlowNet weren't a big differential to fitting the model. Both of the experiments shows signals of over-fit, because the validation loss don't reduce as the same ratio of training loss. Actually, the validation still the same all over both experiments. The main reason of that results are, very likely, the number of images and poses used on training, that were far bellow the necessary.

Since we have an experiment that we were looking for to reproduce, we can also compare our results with them. If we look for Figure 5, we realized that Wang's model were a lot better fitted and different of ours didn't show any signal of over-fit. A main difference in both experiments is the number of images used for training, at [2] they used almost 7 thousand images from the Kitti dataset, while we used among 2 thousand images, which reinforce our previous idea that the main reason of over-fit were the number of images used for fit the model.

Analysing the accuracy results lead us to a very interesting perspective about the system performance, that is well better when predict angles deslocation than position. The most likely reason for that is on how images were collected in the simulator, the deslocation between to images is very abrupt.

Lastly is time to analyse the most visual of all results we obtained, the images comparing the ground truth and the odometry. If we look for Figure 14 it's possible to see that the ground truth has a lot of abrupt changes on both axis directions and this changes are not identified by the model. Figure 15, on the other hand, has a much smoother trajectory, what makes the odometry a lot more accurate, but still not able to manage a sharp turn. Figure 16 has the smoothest of all the trajectories, even on that the model were far away to show a good preview.

The same way compared the training losses of our project with Wang's, we can also compare our odometry with the

[4] https://pytorch.org/

[5] https://towardsdatascience.com/a-brief-review-of-flownet-dca6bd574de0

one obtained by Wang in [2] (Figure 6), even showing some big differences between the ground truth and the odometry the system is much more accurate than the one we obtained. There are few possible reasons for that. First, the fact that our model had over-fit. Second, the images fom the Kitti dataset our way more friendly than the ones from the simulator, since they are obtained by an autonomous car which made it smoother, with a constant velocity and also, the images are rectified (Image rectification is a transformation process used to project images onto a common image plane).

V. Conclusions

In general, the work did not show great results. But, it definitely was great for first experiment with the subject. Also, it was amazing to introduce with the application of machine learning techniques for the visual odometry problem.

Some positive points we can extract from this work is the fact that the using of Deep Learning techniques are plausible to the VO problem, and also, they showed that the simulator images are a good path to make a dataset for the problem.

Weaknesses of the work, which need to be improved in future iterations, are the number of images in the dataset, also some improvements on the neural network, in order to ensure a better fit to the model. Also, we could improve system's effectiveness is to add some kind of SLAM [9] or a salient model to preprocessing the images.

+————————-+

References

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015. 1

[2] 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. 1, 3, 4

[3] 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. 1

[4] E. Colombini. Project 4 specification. [Online]. Available: http://www.ic.unicamp.br/~esther/teaching/2019s2/mo651/P4.pdf 1

[5] 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. 1

[6] 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. 1

[7] 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. 2

[8] C. Robotics. Understanding quaternions. [Online]. Available: http://www.chrobotics.com/library/understanding-quaternions 2

[9] 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. 4

Attachments

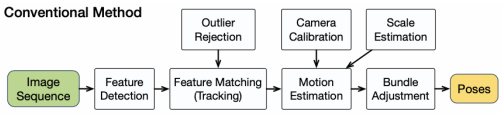

Figure 1. Classical pipeline for Visual Odometry.

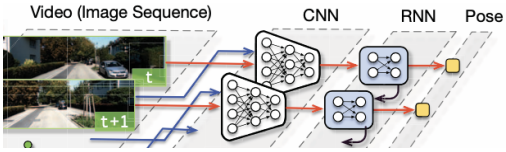

Figure 2. DeepVO pipeline for Visual Odometry.

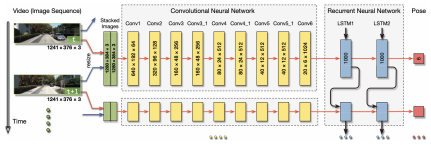

Figure 3. Architecture of the RCNN monocular VO system.

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 4. Configuration of the CNN for monocular VO system.

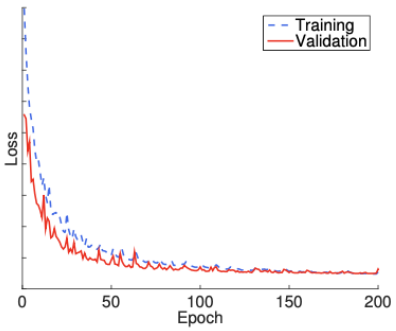

Figure 5. Training losses for the monocular VO system for the Wang paper.

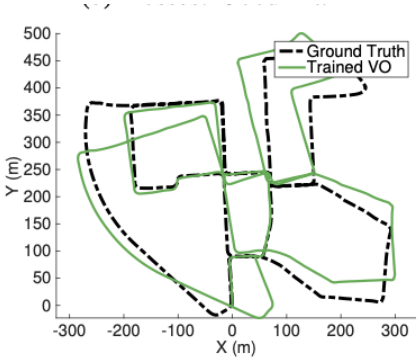

Figure 6. Obtained map for the monocular VO system under the training dataset for the Wang paper.

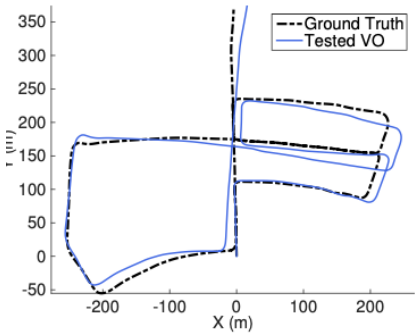

Figure 7. Obtained map for the monocular VO system under the test dataset for the Wang paper.

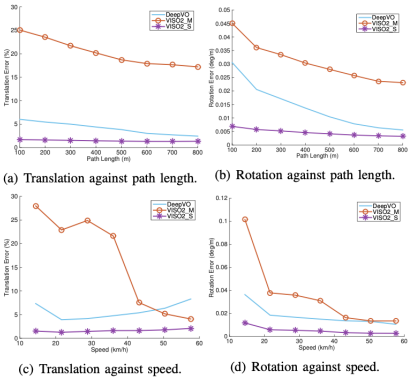

Figure 8. Average errors on translation and rotation against different path lengths and speeds for the Wang paper.

Figure 9. Example of image obtained at the Neighbourhood scene from AirSim.

Figure 10. Example of image obtained at the Africa scene from AirSim.

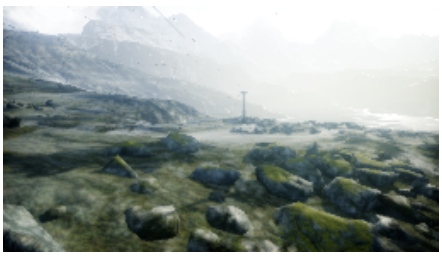

Figure 11. Example of image obtained at the LandscapeMountains scene from AirSim.

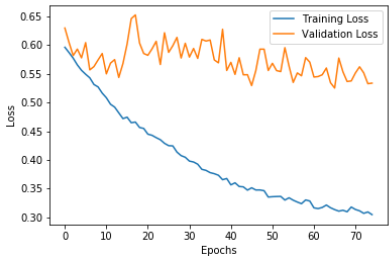

Figure 12. Training losses without the weights of FlowNet.

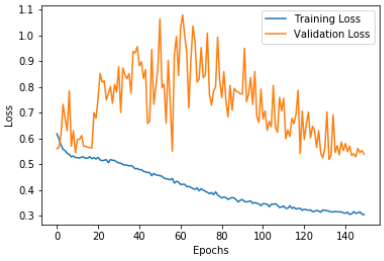

Figure 13. Training losses with the weights of FlowNet.

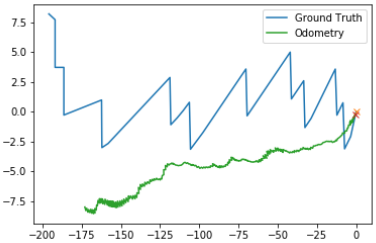

Figure 14. Comparison between the ground truth position and the odometry provided by the system for the first test sequence.

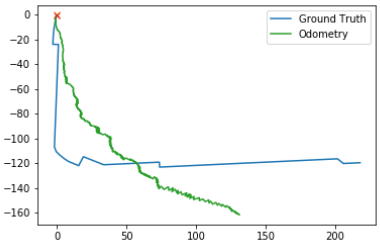

Figure 15. Comparison between the ground truth position and the odometry provided by the system for the second test sequence.

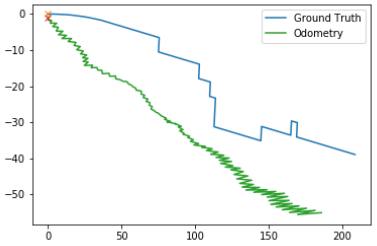

Figure 16. Comparison between the ground truth position and the odometry provided by the system for the third test sequence.