Autonomous Vehicle Global Localization using Deep Learning Visual Odometry and a loosely coupled GNSS system

E. Ramona Stefanescu
Stanford University
Stanford, California 94305
ramona28@stanford.edu

Abstract

For autonomous driving, robust and precise self-localization of vehicles is one of the main challenges. This is typically performed using a combination of wheel odometry and inertial sensing (gyroscopes and accelerometers). This approach has two limitations: inertial sensors are prone to drift, and wheel odometry is unreliable in rough terrain (wheels tend to slip and sink). Hence, great effort is put into the improvement of Visual Odometry (VO) methods to obtain additional localization measurements for automotive applications. In this project, I will present a comparison of traditional computer vision VO approach using ORB tracked features with a Deep Learning based monocular VO algorithm by leveraging deep Recurrent Convolutional Neural Networks (RCNNs).

1. Introduction

Many major car manufacturers, technology companies such as Google and Apple, and universities are actively working on developing self-driving cars. Google's self-driving car project relies on a combination of lasers, radars, and cameras in the form of a roof-mounted sensor pod to navigate pre-mapped environments.

Visual Odometry (VO), as one of the most essential techniques for pose estimation and robot localization, has attracted significant interest in both the computer vision and robotics communities over the fast few decades. Due to its practical importance, plenty of research effort has been devoted to the topic over the years. One popular approach to the problem is based on detecting and matching local feature points and using the obtained correspondences to determine the relative poses (Fig 1). The performance of such system is highly dependent on the accuracy of the local feature matches, which are commonly determined using methods like FAST or ORB features (as seen in Fig. 2).

Recently, methods based on convolutional neural networks (CNNs) have clearly outperformed previous state-of-the-art results in many computer vision problems, such as image classification, object recognition, and image retrieval. In a paper by [8], local feature description based on convolutional neural networks enables more accurate feature matching and geometric verification (Fig. 3), however the application of this work was in the context of large-scale image retrieval and never in a feature tracking and camera trajectory estimation due to the real-time requirements.

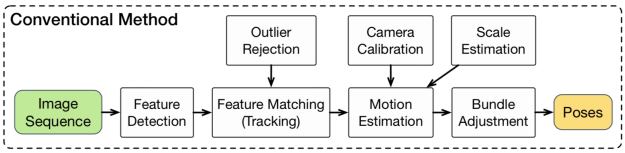

Figure 1. Traditional computer vision pose estimation using Visual Odometry

Making Convolutional Neural Networks (CNNs) successful in learning problems like image based ego motion estimation, highly depends on the ability of the network to extract the temporal information from videos. Therefore, the architecture of a network needs the capability to learn temporal features.

Usually, the position estimation or localization of autonomous vehicles is difficult and error-prone and therefore uses a combination of different input sensors. In this project, I will work on improving the limitations of using an IMU + GPS system on a localization framework by augmenting it with a deep neural network based visual odometry module. To overcome some of the hardware/software challenges (hardware synchronization, different coordinate frame etc) I will use ROS as a middleware ecosystem.

Although, some state-of-the-art algorithm based on camera calibration, feature detection, feature matching (or tracking), outlier rejection, motion estimation have shown excellent performance in terms of accuracy and robustness, they are usually hard-coded with significant engineering ef-

Figure 2. a) FAST features b) ORB features

fort. Deep Learning has recently been dominating many computer vision tasks with promising results. However, there is very limited work on Visual Odometry, which I presume is because most of the existing deep learning architectures and pre-trained models are essentially designed to tackle recognition and classification problems. A VO algorithm ideally should model motion dynamics by examining the changes and connections on a sequence of images.

1.1. Related work

In VO the goal is to recover the full trajectory of one camera or a camera system from images. This is incrementally done by estimating the relative transformation between the camera positions at two time steps and accumulating all transformations over time to recover the full trajectory. Given sequences of images recorded from a camera, one can estimate the ego-motion of the car from feature matches between the camera images.

During the past fifteen years, considerable research effort has already been devoted to visual odometry in computer vision and robotics. Most of previous visual odometry methods can be typically divided into three categories. The first group is based on the expensive LiDAR sensor ([13]), which registers scans between the different time stamps. The second one relies on local feature matching across video frames([1, 14]), while the third group minimized the photometric error between the current frame and reference keyframe [7]. The feature-based visual odometry approaches are currently very popular. The key is to first estimate the camera poses from the salient point correspondences through a robust estimator like RANSAC ([9, 2, 7]) where the local feature extraction algorithm plays a very important role. Then, bundle adjustment is employed to simultaneously refine the camera poses. Most direct visual odometry methods are generally based on the Lucas-Kanade framework, which is one of the most widely used techniques in computer vision. These approaches directly find the optimal geometric transformation by minimizing the photometric error between the input and the warped reference frame. The major limitation of these methods is that they tend to become stuck at a local optimum and hence require a good initialization.

Deep learning (DL) is quickly becoming the dominant approach in computer vision. The many layers of a pre-trained CNN form a hierarchical model with increasingly higher level representation of the input data as one moves up the layer. It has been shown that many computer vision relayed tasks benefit from using the output from these upper layers as feature representations of the input images. These features have the advantage of being low-level enough to provide representations for a large number of concepts, yet are abstract enough to allow these concepts to be recognized using simple linear classifiers. In [3], deep architectures were proposed to learn feature descriptors. The siamise architecture forms the basis for these approaches [6], with the neural networks learning to embed 64×64 patches in a feature space where matching patches are closer to each other than non-matching patches. However, they do not learn keypoints detection, and do not handle various scales natively.

2. Technical Approach

There are few approaches for localization, which has been traditionally solved using Monte Carlo methods. Here, the Markov assumption is used to maintain a particle-based posterior representation of an agent's pose, integrating laser or visual observations. Typically, these methods operate only in a small environment without providing any global (geographic) positioning information. Furthermore, and most significantly, they rely on more specific measurement sources (e.g., depth measurements, sonar) and are restricted to small-scale environments (e.g., parking lots) where accurate 2D plans are available.

The way the pose is modeled is application-specific, but for rigid mobile vehicles it usually has three degrees of freedom (two for position plus one for orientation) in the pla-

nar case and six degrees of freedom (three for position plus three for orientation) in the 3-dimensional case.

2.1. Traditional computer vision Visual Odometry

Most of the localization methods use a variation of the Kalman filter or a particle filter to estimate the vehicle pose. Following the work of [4] for real-time visual odometry and improved by [14] and [7], I will implement a visual odometry pipeline that coupled with a GPS/RTK and IMU system, will provide a stable and robust global localization system.

Input A stream of gray scale images coming from a camera (initially I will test the algorithm on a mono setting and if available I will extended to a stereo system), angular rates and longitudinal accelerations from IMU and lat, lon GPS coordinates.

Output For every pair of images, I need to find the rotation matrix R and the translation vector t and integrate them with the other sensors output.

Once we have point-correspondences (FAST or ORB), we have several techniques for the computation of an essential matrix. The essential matrix is defined as follows:

$$y_1^T R y_2 = 0 \tag{1}$$

Here, y_1, y_2 are homogenous normalized image coordinates. The Nister algorithm solves a number of non-linear equations, and requires the minimum number of points possible, since the Essential Matrix has only five degrees of freedom. If all of our point correspondences were perfect, then we would have need only five feature correspondences between two successive frames to estimate motion accurately. However, the feature tracking algorithms are not perfect, and therefore we have several erroneous correspondence. A standard technique of handling outliers when doing model estimation is RANSAC. It is an iterative algorithm. At every iteration, it randomly samples five points from the set of correspondences, estimates the Essential Matrix, and then checks if the other points are inliers when using this essential matrix. The algorithm terminates after a fixed number of iterations, and the Essential matrix with with the maximum number of points agree, is used.

The pose of the camera be denoted by R_{pos}, t_{pos}. We can then track the trajectory using the following equation:

$$\begin{aligned} R_{pos} &= R \cdot R_{pos} \\ t_{pos} &= t_{pos} + t \cdot R_{pos} \end{aligned} \tag{2}$$

This approach I have implemented for the CS231A class project.

2.2. Deep Learning based Methods

DL has achieved promising results on some localization related applications. The features of CNNs, for instance, have been utilized for appearance based place recognition

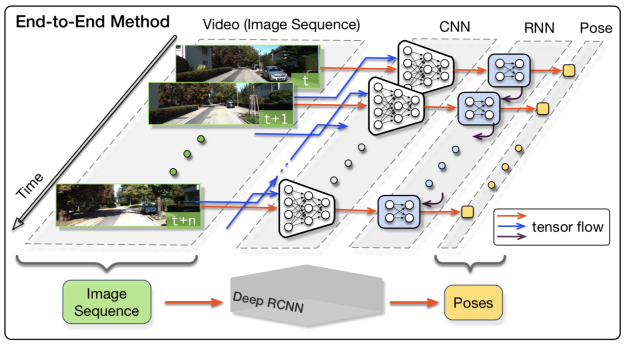

Figure 3. DeepVO - the network structure. Figure taken from [12]

[10]. Unfortunately, there is little work on VO or pose estimation. One of the seminal deep learning approaches for visual odometry was proposed by [5]. They propose a CNN architecture which infers odometry based on classification. A set of prior velocities and directions are classified through a softmax layer to infer the transformation between images and a stereo camera. A major drawback of this approach lies in modeling a regression problem as a classification one which reduces the representational capabilities of the learned model. [12] train a recurrent, convolutional neural net end-to-end for visual-inertial odometry. This approach concatenates IMU information from a recurrent network setup, and images processed using convolutional filters and a correlational map between successive frames. This model learns to predict pose - which is very similar to my goal of predicting pose. For this reason, I will use this architecture for the project.

A framework which can learn geometric feature representations is of importance to address the VO and derive connections among consecutive image frames. One major drawback of CNN architectures is the fact that they only analyses just-in-moment information, whereas VO is rather dependent on the correlative information across frames. Unlike traditional feed-forward artificial neural networks, RCNN can use its internal memory to process arbitrarily long sequences by its directed cycles between the hidden units. Therefore, I think that RCNN architectures are more suitable than CNN architectures for VO tasks. The advantage of the RCNN based architecture is to allow simultaneous feature extraction and sequential modeling of VO through a combination of CNN and RNN.

3. End-to-end visual odometry

It is mainly composed of CNN based feature extraction and RNN based sequential modeling. Most of computer vision DNN architecture are designed to tackle recognition, classification and detection problems. A framework which can learn geometric feature representations is of importance

to address the VO and derive connections among consecutive image frames. The architecture proposed takes a image sequence as input, passed through a CNN to produce an effective feature for the monocular VO and the fed into a RNN for sequential learning.

The advantage of the RCNN based architecture is to allow simultaneous feature extraction and sequential modeling of VO through a combination of CNN and RNN.

3.1. CNN based Feature Extraction

In order to automatically learn effective features that are suitable for the VO problem, a CNN is developed to perform feature extraction on the concatenation of two consecutive monocular RGB images.

The feature representation is ideally geometric instead of being associated with appearance or visual context because VO systems need to be generalized and deployed in unknown environments. The idea is to used an architecture which is less generic, but may perform better given the data and optimization techniques. A simple choice is to stack both input images together and feed them through a rather generic network, allowing the network to decide itself how to process the image pair to extract the motion information.

In principle, if the network is large enough, it could learn to predict the pose. However, we can never be sure that a local gradient optimization like stochastic gradient descent can get the network to this point.

The network has 9 convolutional layers and each layer is followed by a rectified linear unit (ReLU) activation except Conv6, i.e., 17 layers in total. The sizes of the receptive fields in the network gradually reduce from 7×7 to 5×5 and then 3×3 to capture small interesting features. Zero-paddings are introduced to either adapt to the configurations of the receptive fields or preserve the spatial dimension of the tensor after convolution. The number of the channels, i.e., the number of filters for feature detection, increases to learn various features.

The CNN takes raw RGB images as input because the network is trained to learn an efficient feature representation with reduced dimensionality for the VO. This learnt feature representation not only compresses the original high-dimensional RGB image into a compact description, but also boosts the successive sequential training procedure. Hence, the last convolutional feature Conv6 is passed to the RNN for sequential modeling.

3.2. RNN based Sequential Modeling

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. RNNs are very suitable for modeling the dependencies across image sequences and for creating a temporal motion model since it has a memory of hidden states over time and has directed cycles among

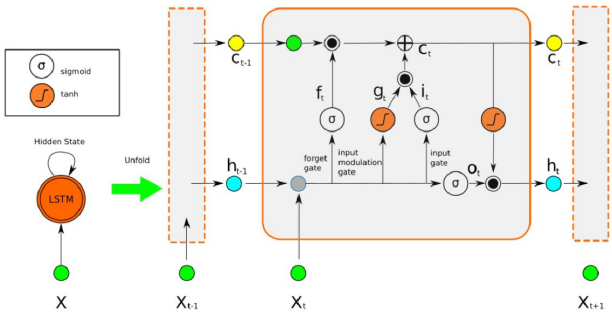

Figure 4. Information flow through the hidden units of the LSTM

hidden units, enabling the current hidden state to be a function of arbitrary sequences of inputs . Thus, using RNN, the pose estimation of the current frame benefits from information encapsulated in previous frames [11].

Given a set of features x_t at time t, RNN updates at time step t, W denote corresponding weight matrices of the hidden units, b the bias vector, and H an element-wise hyperbolic tangent based activation function. Long short-term memory (LSTM) is more suitable than RNN to exploit longer trajectories since it avoids the vanishing gradient problem of RNN resulting in a higher capacity of learning long-term relations among the sequences by introducing memory gates such as input, forget and output gates and hidden units of several blocks. The input gate controls the amount of new information flowing into the current state, the forget gate adjusts the amount of existing information that remains in the memory and the output gate decides which part of the information triggers the activations. The folded LSTM and its unfolded version over time are shown in Fig. 4 along with the internal structure of a LSTM memory cell. It can be seen that unfolded LSTMs correspond to timestamps. Given the input vector x_t at time t, the output vector and the cell state vector of the previous LSTM unit, the LSTM updates at time step t according to the following equations, where σ is sigmoid non-linearity, $\tanh$ is hyperbolic tangent non-linearity, W terms denote corresponding weight matrices, b terms denote bias vectors, i_t, f_t, g_t, c_t and o_t are input gate, forget gate, input modulation gate, the cell state and output gate at time t, respectively:

$$
\begin{aligned}
f_t &= \sigma(W_f \cdot [x_t, h_{t-1}] + b_f) \\
g_t &= \tanh(W_g \cdot [x_t, h_{t-1} + b_g) \\
o_t &= \sigma(W_o \cdot [x_t, h_{t-1}] + b_o) \\
i_t &= \sigma(W_i \cdot [x_t, h_{t-1}] + b_i) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
\tag{3}
$$

The deep RNN is constructed by stacking two LSTM layers with the hidden states of a LSTM being the input of

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

Table 1. RCNN layers

the other one. Each of the LSTM layers has 1000 hidden states.

3.3. Cost function

In a probabilistic formulation, the VO system computes the conditional probability of the poses $Y_y = (y_1, \ldots, y_t)$ given a sequence of monocular RGB images $X_t = (x_1, \ldots, x_t)$ up to time t such that:

$$p(Y_t|X_t) = p(y_1, \ldots, y_t|x_1, \ldots, x_t) \tag{4}$$

To find the optimal parameters θ^* for the VO, the DNN maximizes 4:

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\, p(Y_t|X_t; \theta) \tag{5}$$

To find the hyperparameter θ, the Euclidean distance between the ground truth pose (p_t, ϕ_t) at time t and its estimated one $(\hat{p_t}, \hat{\phi_t})$ is minimized.

The training loss is given as the mean squared error between network output and target pose, summed over batch and time dimensions.

$$L(\text{Batch}; \theta) = \frac{1}{B} \sum_{i=1}^{B} \sum_{t=1}^{T} \|\hat{p}_{i,t} - p_{i,t}\|^2 + \kappa \|\hat{\phi}_{i,t} - \phi_{i,t}\|^2 \tag{6}$$

where κ is introduced as a weighting factor since translational and rotational units are not directly comparable. [12] use the Adagrad optimizer with a learning rate of 0.001. Howevrer parameters such as the batch size, sequence length, dropout details, and dropout details are missing from the paper.

While the paper makes no mention of treating angle differences in any particular way in the loss function, we ensured the angular difference between estimate and target rotation is always in the range $[-\pi, \pi]$. Without special consideration, the difference between e.g. 1° and 359° would come out as 358° instead of -2°. We thus compute the difference in the rotational component as

$$\text{diff}(\hat{\phi}_i, \phi_i) = atan2\left(\sin \hat{\phi}_i - \phi_i, \cos \hat{\phi}_i - \phi_i\right) \tag{7}$$

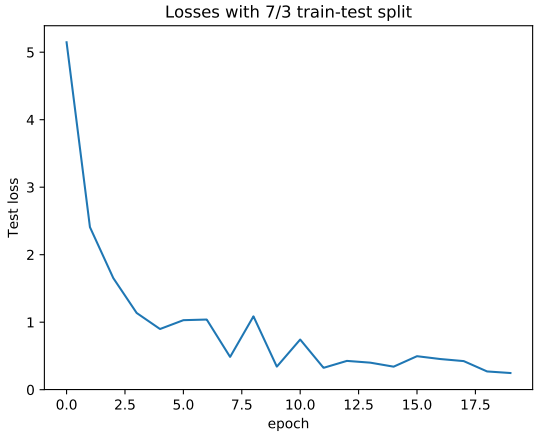

Figure 5. Losses over time on forward training with 100 memory size, 2 batch size, 6 sequence length and 0.001 learning rate

This ensures the difference will have the proper sign and be in the interval $[-\pi, \pi]$.

4. Bayesian filtering

The integration of a local frame VO with a global position estimator (GPS+IMU) is done using a Bayesian filter. The filtering problem can be expressed as estimating the state x of the dynamic discrete system, given:

- the analytical knowledge of the state transition function f_t and the statistical knowledge of the state noise w_t
- the analytical knowledge of the output function h_t and the statistical knowledge of the observation noise v_t
- a realization of the system output $z_{0:t}$ up to time t.

A probabilistic filter for a system is a mathematical tool whose goal is to estimate system state history given the measurements, that is estimating:

$$p(x_{0:t}|z_{0:t}) \tag{8}$$

Often it is interesting to evaluate only the marginal distribution of the current state:

$$p(x_t|z_{0:t}) \tag{9}$$

given the observations.

Many approaches for computing the state estimate have been proposed - many of them rely on the assumption that the process being observed is Markov. A process is Markov if the current measurement is independent from the past ones, given the current state: $p(z_t|z_{0:t-1}, x_t) = p(z_t|x_t)$.

Extended Kalman Fiter is the first and most direct application of the Kalman filter to non-linear systems. This filter model attempts to allow for system non-linearities by linearizing the system through the use of Jacobian matrices. This takes the for of a minimum mean-square-error estimator based on the first-order Taylor series expansion. The EKF filter equations are as follows:

- *Prediction Phase*:

$$\hat{x}_t^- = f(\hat{x}_{t-1}, u_t) \tag{10}$$

$$P_t^- = A_k P_{k-1} A_k^T + W_t Q_t W_t^T \tag{11}$$

- *Update Phase*:

$$\begin{aligned} K_t &= P_t^- H_t^T (H_t P_t^- H_t^T + V_t R_t V_t^T)^{-1} \\ \hat{x}_t &= \hat{x}_t^- + K_t(z_t - h(\hat{x}_t^-)) \\ P_t &= (I - K_t H_t) P_t^- \end{aligned} \tag{12}$$

Where h, f: the non-linear measurement functions, $\hat{x}$: estimated state, $\hat{x}_k^-$: current state prediction, u: control variables, P: state variance matrix (i.e., error of estimation), Q: Process variance matrix (i.e., error due to process), z: measurement variables, K: Kalman gain, R: measurement variance matrix (i.e., error from measurements), A: the Jacobian matrix of partial derivatives of f with respect to x, W: the Jacobian matrix of partial derivatives of f with respect to w, H: the Jacobian matrix of partial derivatives of h with respect to x, V: The Jacobian matrix of partial derivatives of h with respect to v. Subscripts are as follows: k current time period, $k-1$ previous time period.

5. Dataset and software overview

Collecting data for training a CNN to be able to estimate the ego motion is relatively easy. There is no need to manually label data as the labels can be recorded synchronizing the camera with a GPS or IMU. While [12] used the well known *KITTI* [1] dataset for training, I have decided to create my own dataset, having available a Point Grey Blackfly Mono, 1.3 Mpx, at a 10fps, a Xsense IMU and a Ublox GPS.

The feature extraction for visual motion estimation is very different to other autonomous vehicle related tasks, such as object detection or scene segmentation. In monocular visual ego motion estimation the relevant information is available only within the time domain. Viewing a single image does not provide any temporal information for estimating movements. Thus, camera images need to be synchronized with the GPS in order to obtain the scale information to be used in the training process.

The code is written for Python 2.7 against the 1.3 version of TensorFlow. The authors of the original DeepVO paper [12] didn't release their code, so the implementation I wrote was inspired by TensorFlow RNN tutorial [2] and FlowNet [3]. The training was performed on the Google Compute platform.

Training was performed on a longer driven route in the area of Menlo Park, while the testing was performed on a much shorter route. Images were collected using the same camera and GPS/IMU configuration. On the testing dataset, the network is likely to overfit, since many inputs are identical and similar convergence can be observed on the dataset. On more realistic data with a lot of fast rotations, I was unable to properly evaluate the performance of the network. However, I was able to learn simple forward-backward motions without orientation changes as seen in Figure 5.

6. Results

The available sensor data are: vehicle speed (v) in heading direction (ϕ) and a yaw rate ($\dot{\phi}$) which both have to be fused with the position (x and y) from a GPS sensor.

$$x_k = \begin{bmatrix} x \\ y \\ \phi \\ v \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} position\ X \\ position\ Y \\ heading \\ velocity \\ yaw\ rate \end{bmatrix} \tag{13}$$

The process matrix for a constant turn and constant velocity model which is used in this case has the following form:

$$g(x) = \begin{bmatrix} x + \frac{v}{\dot{\phi}}(-sin(\phi) + sin(dt * \dot{\phi} + \phi)) \\ y + \frac{v}{\dot{\phi}}(cos(\phi) - cos(dt * \dot{\phi} + \phi)) \\ dt * \dot{\phi} + \phi \\ v \\ \dot{\phi} \end{bmatrix} \tag{14}$$

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

[2] https://github.com/tensorflow/models/blob/master/tutorials/rnn/ptb/ptb_word_lm.py

[3] https://github.com/lmb-freiburg/flownet2

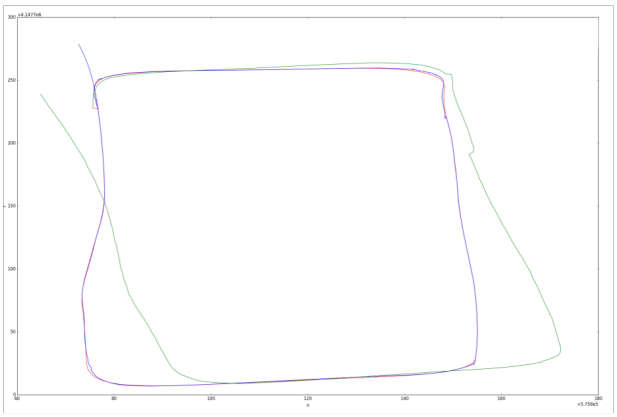

Figure 6. Resulted trajectory: GPS in red, ORB VO in blue, Deep VO in green

The Jacobian of the process matrix J_A with respect to the state vector is defined as:

$$J_A = \begin{bmatrix} 1.0 & 0.0 & h_{13} & h_{14} & h_{15} \\ 0.0 & 1.0 & h_{23} & h_{24} & h_{25} \\ 0.0 & 0.0 & 1.0 & 0.0 & dt \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{15}$$

where:

$$h_{13} = \frac{v}{\dot{\phi}}(-cos(\phi) + cos(dt * \dot{\phi} + \phi)) \tag{16}$$

$$h_{14} = \frac{1}{\dot{\phi}}(-sin(\phi) + sin(dt * \dot{\phi} + \phi)) \tag{17}$$

$$h_{15} = \frac{dt * v}{\dot{\phi}}(cos(dt * \dot{\phi} + \phi) - \frac{v}{\dot{\phi}^2}(sin(dt * \dot{\phi} + \phi)) \tag{18}$$

$$h_{23} = \frac{v}{\dot{\phi}}(sin(dt * \dot{\phi} + \phi) - sin(\phi)) \tag{19}$$

$$J_A = \begin{bmatrix} 1.0 & 0.0 & h_{13} & h_{14} & h_{15} \\ 0.0 & 1.0 & h_{23} & h_{24} & h_{25} \\ 0.0 & 0.0 & 1.0 & 0.0 & dt \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix} \tag{20}$$

$$h_{24} = \frac{1}{\dot{\phi}} * (-cos(dt * \dot{\phi} + \phi) + cos(\phi) \tag{21}$$

$$h_{25} = \frac{dt * v}{\dot{\phi}} * sin(dt * \dot{\phi} + \phi) - \frac{v}{\dot{\phi}^2}(-cos(dt * \dot{\phi} + \phi)) \tag{22}$$

If a GPS measurement is available, the following function maps the state to the measurement.

$$h = \begin{bmatrix} x \\ y \\ v \\ \dot{\phi} \end{bmatrix} \tag{23}$$

The matrix J_H is the Jacobian of the Measurement function h with respect to the state. Function h can be used to compute the predicted measurement from the predicted state. GPS measurements are available the J_H matrix is defined as:

$$J_H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{24}$$

Otherwise, if no GPS measurements are available, then the J_H matrix is equal to:

$$J_H = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{25}$$

In Figure 6 the global integrated trajectories are presented. As expected, when GPS data is available the resulted trajectory is interpolated trajectory is smooth and can be considered "ground truth". The traditional VO is noisy and at the corners, when new features are computed, the accuracy is low. Also, as the length of the trajectory increases, both the translational and rotational error of all the proposed models significantly decrease.

The DL VO has a lot of drift and as expected it underperforms the traditional VO. As evaluation metric, I will use Root Mean Squared Error (RMSE) as well the histogram of errors. RMSE is a quadratic scoring rule that measures the average magnitude of the error. Since the errors are squared before they are averaged, RMSE gives a relatively high weight to large errors. This means the RMSE is useful when large errors are particularly undesirable. For the tested route, the DL VO RMSE equals $0.333491m$ for the UTM coordinates and $0.083772deg$ for the yaw angle, compared the the traditional VO, the RMSE equals $0.087362m$ and $0.010872deg$ for the UTM coordinates and yaw angle, respectively. The histogram of all errors is presented in Figure 7.

One of the reasons for this performance is that even though the proposed DL VO is less generic, this type of deep learning architecture is essentially designed to tackle recognition and classification problems and not extract the motion information. It is however expected that novel architectures that combines both traditional VO and DL VO to emerge in the future.

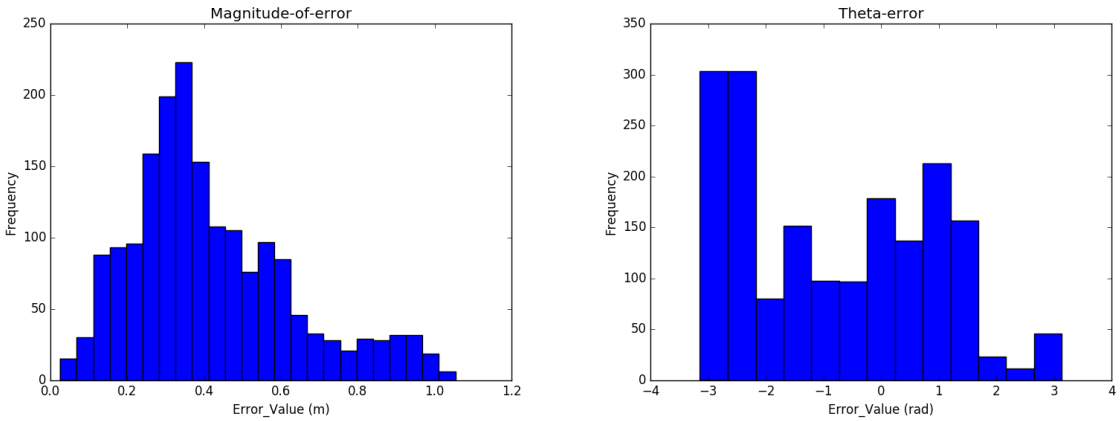

Figure 7. Histogram of errors: a) distance between GPS UTM coordinates and global DL VO b) yaw angle

7. Conclusions

In this project a comparison between traditional computer vision visual odometry and a end-to-end monocular VO based on Deep Learning is presented. The novelty of the project is the integration of the local frame odometry from the deep neural network with a global system of reference given by GPS and IMU and providing accuracy estimates in a global frame. The main benefit of using an end-to-end framework is that there is no need to carefully fine-tune the parameters of the system and with enough training scenarios, one can obtain a smooth and close to the ground truth trajectory.

Many issues faced by traditional VO techniques such as feature correspondence establishment in low textured areas, high reflections, motion blur and low image quality are hoped to be handled by an end-to-end DL VO. However, as of now, in the literature there is no DL VO proposed that can replace the classic geometry based approach.

As future work, the framework needs to be tested for various trajectories with different complexity levels of motions, including complicated paths with fast incremental translations and rotations. I also consider combining deep VO with some functionalities from the traditional VO pipelines such as RANSAC for outlier detection and bundle fusion for globally consistent pose estimation etc to avoid drifts.

The code is available at: https://github.com/ramonastef28/ProjectCS231n.

References

[1] H. Badino, A. Yamamoto, and T. Kanade. Visual odometry by multi-frame feature integration. In *Computer Vision Workshops (ICCVW), 2013 IEEE International Conference on*, pages 222–229. IEEE, 2013.

[2] 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.

[3] X. Han, T. Leung, Y. Jia, R. Sukthankar, and A. C. Berg. Matchnet: Unifying feature and metric learning for patch-based matching. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 3279–3286. IEEE, 2015.

[4] A. Howard. Real-time stereo visual odometry for autonomous ground vehicles. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 3946–3952. IEEE, 2008.

[5] K. R. Konda and R. Memisevic. Learning visual odometry with a convolutional network. In *VISAPP (1)*, pages 486–490, 2015.

[6] J. L. Long, N. Zhang, and T. Darrell. Do convnets learn correspondence? In *Advances in Neural Information Processing Systems*, pages 1601–1609, 2014.

[7] 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.

[8] H. Noh, A. Araujo, J. Sim, and B. Han. Image retrieval with deep local features and attention-based keypoints. *CoRR*, abs/1612.06321, 2016.

[9] D. Scaramuzza and F. Fraundorfer. Visual odometry [tutorial]. *IEEE robotics & automation magazine*, 18(4):80–92, 2011.

[10] N. Sünderhauf, S. Shirazi, A. Jacobson, F. Dayoub, E. Pepperell, B. Upcroft, and M. Milford. Place recognition with convnet landmarks: Viewpoint-robust, condition-robust, training-free. *Proceedings of Robotics: Science and Systems XII*, 2015.

[11] M. Turan, Y. Almalioglu, H. Araujo, E. Konukoglu, and M. Sitti. Deep endovo: A recurrent convolutional neural network (rcnn) based visual odometry approach for endoscopic capsule robots. *Neurocomputing*, 275:1861–1870, 2018.

[12] S. Wang, R. Clark, H. Wen, and N. Trigoni. Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. *CoRR*, abs/1709.08429, 2017.

[13] J. Zhang and S. Singh. Visual-lidar odometry and mapping: Low-drift, robust, and fast. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 2174–2181. IEEE, 2015.

[14] J. Ziegler, H. Lategahn, M. Schreiber, C. G. Keller, C. Knoppel, J. Hipp, M. Haueis, and C. Stiller. Video based localization for bertha. In *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*, pages 1231–1238. IEEE, 2014.