How to boost a traditional indirect method for Monocular Visual Odometry via deep neural networks

Kirill Brodt
cyrill.brodt@gmail.com

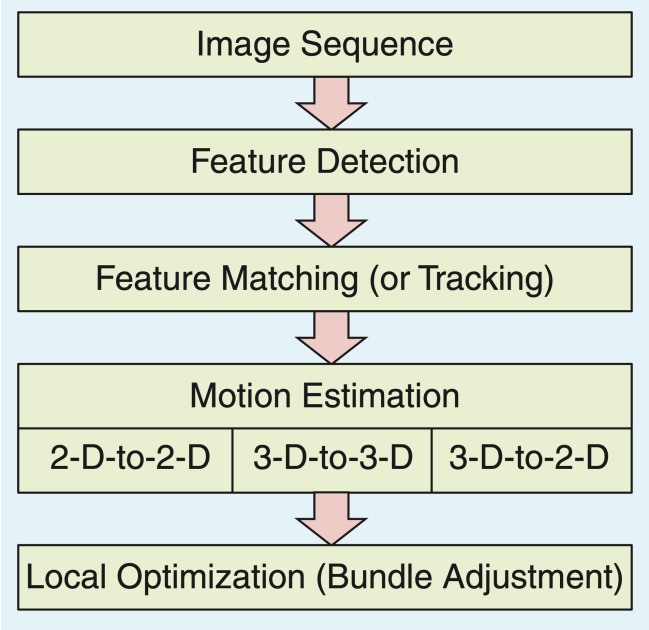

Figure 1: Visual Odometry pipeline [Scaramuzza and Fraundorfer, 2011]

Abstract

We present a simple improvement of traditional indirect method for Monocular Visual Odometry replacing classical feature detectors and feature matchers with deep neural network models. These models have good generalization performance without finetuning on specific task.

1 Introduction

Visual Odometry (VO) is the process by which the body egomotion is inferred by analyzing images captured while in motion. Scaramuzza and Fraundorfer [2011] present a general overview of the VO pipeline as in the Figure 1. We follow the similar pipeline except that we introduce deep neural networks for feature detection and feature matching.

2 Methodology

We observe that classical approaches based on ORB (or others) detectors and FLANN (or others) matchers fail on this task. So, we focus our analisys on feature detection and feature matching steps. We replace feature detection and matching models with state-of-the-art dense feature matching deep neural network model RoMa [Edstedt *et al.*, 2023b]. We further refine this matching with another deep neural network model DeDoDe descriptor [Edstedt *et al.*, 2023a]. We use publicity available pretrained weights for mentioned models. Our method doesn't require any retraining and has good generalization on new tasks. Though one may improve the performance by finetunning the models to the specific task.

3 Experiments

We first rotate two consecutive frames and crop the bottom part of the image with the static part of the car to remove the noisy feature detections. Then, we resize the images to $W \times H = 1344 \times 1712$ resolution and feed them to RoMa model with default parameters to detect and match features. Finally, we refine the matching using DeDoDe descriptor model. We show the example of RoMa model's matching for two consecutive frames in Figure 2.

After a set of features have been matched we use 2D-to-2D method to model the motion between frames. We find the essential matrix with RANSAC algorithm with confidence 0.99 and threshold 1 and recover the camera pose. To refine the pose we further use local bundle adjustment implemented in PyTorch [Edstedt, 2023]. Note, that we use only two consecutive frames and there is no way to determine the magnitude of translation between two cameras. We use constant magnitude equal to 7.571103106907284 calculated as a mean of magnitudes from train trajectories.

Evaluation metrics

The performance metric is the rotational relative pose error of the estimated camera poses:

$$E_{rot} = \frac{1}{\mathcal{F}} \sum_{(i,j)\in\mathcal{F}} \angle \left[(\hat{\mathbf{p}}_\mathbf{j} \ominus \hat{\mathbf{p}}_\mathbf{i}) \ominus (\mathbf{p}_\mathbf{j} \ominus \mathbf{p}_\mathbf{i}) \right], \qquad (1)$$

where $\mathcal{F}$ refers to a set of frames belonging to a single trajectory and (i, j) denotes two adjacent frames within the sequence. $\hat{\mathbf{p}}$ and $\mathbf{p}$ are the estimated and ground true camera

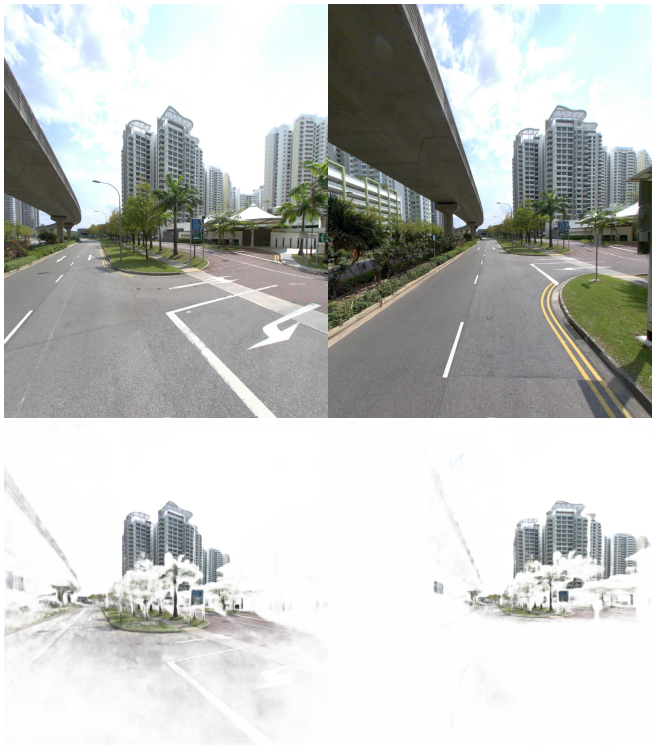

Figure 2: RoMa's dense matching (left `180426_013227300_Camera_0`, right `180426_013228811_Camera_0`)

Trajectory	Rotational Error		Runtime (min)
	Ours	Stationary	
1 (1398 frames)	0.0433	0.1972	35
2 (1301 frames)	0.0469	0.2026	33
3 (4914 frames)	0.0144	0.0954	125
4 (2390 frames)	0.0202	0.1037	61
5 (2218 frames)	0.0382	0.2028	56

Table 1: Rotational relative pose error over different trajectories.

pose coordinates, respectively, $\ominus$ denotes the inverse compositional operator and $\angle[\cdot]$ represents the rotation angle.

Results

We show the resulting metrics over five different trajectories in Table 1. We compare our method with the stationary camera model (constant predictions) and see the 5x improvement on average. We also tried a classical non-deep learning approach with ORB detector and FLANN matcher and didn't manage to get better performance than the stationary camera. Therefore, we don't show the metrics for the classical algorithms. We show the predicted sub-trajectories for trajectory 1 in Figure 3.

Performance

We have implemented our method in Python using OpenCV and PyTorch libraries. All the results presented in the report were computed with the default parameters presented in the

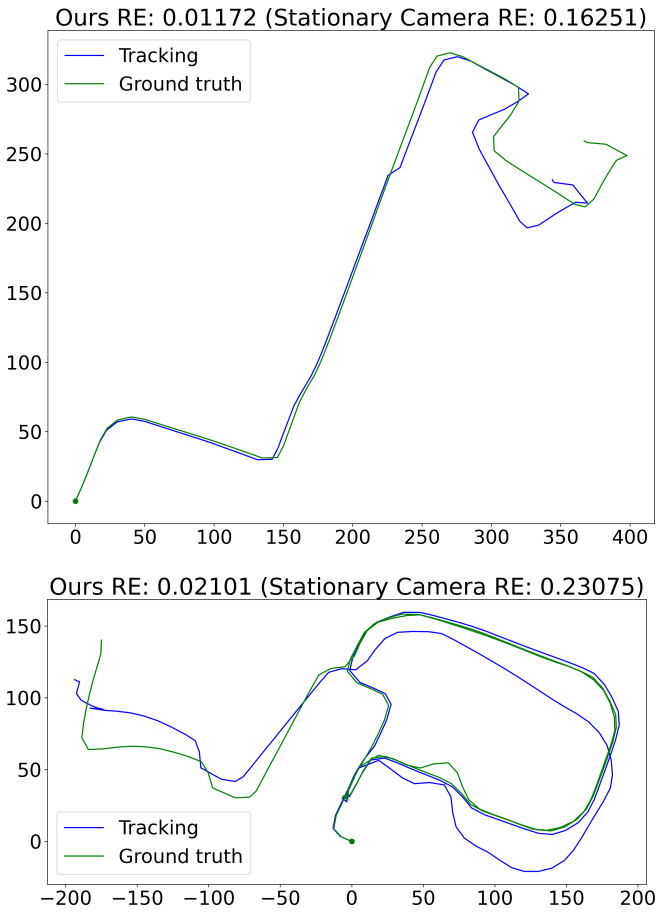

Figure 3: Examples of some predicted sub-trajectories of our method. Note, that here we use ground truth magnitudes of translation vectors between two cameras for improved presentation. The titles represent rotation error (RE).

text and on the machine with single core of Intel® Xeon® Gold 6148 CPU @ 2.40GHz, 4GM RAM and NVIDIA® Tesla® V100-SXM2 32GB. The actual GPU VRAM usage is 16GB. The full pipeline performs 0.65fps or 1.5s per frame.

References

[Edstedt *et al.*, 2023a] Johan Edstedt, Georg Bökman, Mårten Wadenbäck, and Michael Felsberg. DeDoDe: Detect, don't describe, describe, don't detect, for local feature matching. 2023.

[Edstedt *et al.*, 2023b] Johan Edstedt, Qiyu Sun, Georg Bökman, Mårten Wadenbäck, and Michael Felsberg. RoMa: Revisiting robust lossses for dense feature matching. *arXiv preprint arXiv:2305.15404*, 2023.

[Edstedt, 2023] Johan Edstedt. Micro bundle adjustment. 2023.

[Scaramuzza and Fraundorfer, 2011] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry part i: The first 30 years and fundamentals. 2011.