Transformer-Based Deep Monocular Visual Odometry for Edge Devices

Anton Klochkov
ITMO University, BrainGarden LLC
akl@braingarden.ai

Ivan Drokin
BrainGarden LLC
ivan@braingarden.ai

***Abstract*—A lot of recent works have shown that deep learning-based visual odometry methods outperform existing feature-based approaches in the monocular case. However, most of them cannot be used in mobile robotics because they require a sufficiently powerful computing device. In this paper, we propose a method with a significant reduction in computing resources and with a slight decrease in accuracy. To achieve that we replaced recurrent block by lightweight transformer-based module.**

We evaluate the proposed model on the KITTI dataset for calculating accuracy and test a computation cost on NVIDIA Jetson Nano and NVIDIA Jetson AGX Xavier. Our experiments show that the proposed model works faster than the considered original model.

The code for training and testing is available at https://github.com/toshiks/TBDVO.

I. Introduction

Visual Odometry (VO) is one of the most required techniques for pose estimation (ego-motion) from image sequences. It is crucial for a lot of applications in robotics (SLAM, autonomous driving, etc.), augmented/virtual reality, etc.

Over the past few decades, it has been proposed a lot of traditional (feature-based, direct photometric-based) methods for visual odometry. Although state-of-the-art algorithms show perfect robustness and accuracy, they are usually required significant engineering effort and careful integration into the system. Also, some of them perform well only within special environmental conditions, such as light, variety in textures, etc. In monocular camera case methods have low perfomance due to the scale drift [1], [2] and low robustness.

Recently, deep learning has been applied to a lot of computer vision tasks [3], [4], including visual odometry [5]–[10]. Most of the proposed models used Convolutional Neural Networks (CNNs) or a combination of them with Recurrent Neural Networks (RNNs) for calculation relative camera pose. CNN-only based approaches [5], [6], [10], [11] have large computation cost, because for robust performance, it requires combination of pose/depth/optical flow estimation networks [3], [4], [12]. RNN-based methods [7], [9], [13] have of lot of parameters because of LSTM block and also required significant computing resources.

In this paper, we propose a method for computational optimization of RNN-like architectures. We choose one of the well-known models for monocular visual odometry DeepVO [7] as a basis. We suggest using an encoder of transformer [14] instead of LSTM. Due to positional encoding, the sequence of frames is not disturbed, and the self-attention module helps to evaluate the context, similar to LSTM, but more efficiently: information is processed simultaneously, so information will not be lost with large sequences, while this happens in LSTM cells.

We evaluate the performance of the proposed model on the KITTI dataset [15]. Image sequence processing speed is evaluated on NVIDIA Jetson Nano and NVIDIA Jetson AGX Xavier. Our experiments show that on small sequences, our approach is faster than DeepVO [7], and with an increase in the number of frames, LSTM significantly loses in speed to the transformer. We also managed to reduce the number of model's trainable parameters by 10 times (which had a positive effect on the learning process) and the number of calculations.

II. Related works

A. Traditional methods for visual odometry

Traditional methods for ego-motion [16]–[19], which have been studied for the past few decades, include a several-stage pipeline for image processing. At the first stage feature detectors is used (SIFT [20], SURF [21], Shi-Tomasi [22], etc.) for group of keypoints extracting. After that, at the second stage, keypoints matching on consecutive frames is performed. The last stage estimates the global scale of detected keypoints by motion estimation (minimization of reprojection error).

However, these approaches have low-quality accuracy in an environment with low texture environments or poor illumination [23]. Other methods, which estimate the motion by pixel intensities [24], require extensive computational resources.

B. Deep learning for visual odometry

In the last decade, deep learning approaches tightly entered the toolkit for solving computer vision problems. In particular, to address issues with traditional feature-based methods for visual odometry, many works are adopting CNNs [5], [6], [25] for extracting and matching features to estimate visual odometry. In these works, authors concatenate two consecutive images, process them by CNN and transform visual features to pose by stacking of linear layers. Processing only two images has a serious disadvantage: neural network incapable of modeling sequential information [7], which affects the quality.

For extracting space-time motion information one can use 3D CNN, but it requires extensive computational resources,

ISSN 2305-7254

even for very small images. Moreover, such an approach does not allow work with different sequence sizes. Also, many methods use depth estimation network [12] to approximate a global scale: limitations of them we describe in next section.

Finally, for processing the sequence of images, RNNs [7], [13] can be used. In these approaches, the sequence is splitted into pairs of consecutive images, each of them is processed by CNN and fed into the LSTM block. The hidden state inside LSTM allows to model dynamics and relations among a sequence of CNN features. Authors use RNNs to solve the problem with extracting space-time motion information, but have limitations of sequence length: with increasing count of images information is lost and the number of computational operations increases. Also, RNNs have limited context: elements of sequence processes one by one, so recent values have the greatest influence on prediction.

Partially issue connected with losing information solved in the [9] by using external memory block, which can save information about moving among frames in long sequences. But memory block requires additional computation costs.

C. Deep learning for monocular depth estimation

Most of the modern state-of-the-art models for visual odometry used depth maps for minimizing the photometric error between the recovered image from it and the original image. So, the temporal information is included in the training of depth, which produces to more accurate pose estimation.

Most of these works solve the problem in an unsupervised way by learning depth map estimation using the photometric error [26], [27]. However, approaches in them are not able to recover the global scale, which needs for accurate depth map estimation. To avoid it in [8], [28] was proposed using stereo image pairs to estimate scaled visual odometry.

Also, in the recent work [29] was shown, that modern technics for optimization depth estimation networks, can give only 19 FPS (Frame Per Second) for input image shape 224×224 on NVIDIA TX2 edge platform (which is comparable in performance with NVIDIA Jetson Nano or NVIDIA Jetson AGX Xavier). Because, usually for solving visual odometry tasks need a pose estimation network (which has similarities with the depth estimation model architecture), the speed of the final approach can be less than 10 FPS.

For these reasons, we do not consider approaches with depth estimation models.

D. Deep learning for optical flow estimation

As well as methods from the previous section most of the modern state-of-the-art models for visual odometry used optical flow estimation network [3]. Some of them [11] used encoder and decoder. Another [7], [9] include an only encoder, assuming that this will help to converge faster. Out network contains encoder of optical flow model from [3].

E. Transformer

Recently success of transformers [14] in NLP (Natural Language Processing) promoted researchers to apply them in computer vision tasks. Transformer architectures are based on a self-attention mechanism that learns the relationships between elements of a sequence. Unlike RNN, Transformers can learn long dependencies between input sequence elements and support parallel processing of sequence [30].

Recent work [11] shows that transformer can be applied for visual odometry tasks. But authors use a combination of flow and depth estimation networks, what computationally inefficient.

In this paper, we suggest replacing LSTM with the encoder of transformer on the example of the model from DeepVO [7] to solve problems with long sequences.

III. Method

In this section we begin with an overview of proposed network(Sec. III-A). After, we describe proposed modules in the framework in (Sec. III-B, Sec. III-C, Sec. III-D). Finally, we presen the loss function in (Sec. III-E).

A. Overview

The architecture of the proposed method for visual odometry is shown in Fig 1. For camera pose estimation our model use sequence of consecutive image as input $S = \{I_0, \ldots, I_{N+1}\}$. After, we split the sequence into N pairs $P = \{(I_0, I_1), (I_1, I_2), \ldots, (I_N, I_{N+1})\}$. From these pairs independently we extract features by CNN. Then, we reduce the number of features using average pooling. The sequence of processed features we use as input to the encoder of the transformer (EoT), where the model learns sequential information. Finally, for each element of the sequence, we pass high-level features, which contain sequential information, over linear layers for transformation into N relative poses.

B. Feature Extractor

Traditional methods for visual odometry start with extracting and matching corresponding feature points. To address the same task, we use CNN especially designed for extracting geometric features.

Our feature extractor is based on FlowNet [3], which extract from pairs optical flow. But for our task is not necessary to have an approximated vector of moving each pixel. Because sequential information we extract by the encoder of the transformer, we do not need a decoder from FlowNet [3]. So, we use an only encoder.

FlowNet [3] encoder contains stack of convolution layers. First of them has kernel size 7×7, next to — 5×5 and remaining layers — 3×3. Each convolution is accompanied by BatchNorm2d, LeakyReLU, and Dropout layers. The number of channels (filters for feature extraction) increases from layer to layer. It allows collecting as many features as possible.

In our case for two concatenated RGB images with shape $[180, 600]$ transformed by CNN to tensor with shape $[1024, 3, 10]$. It means, that input to the next linear layer, which is located in EoT, should map 30720 number features to hidden size number. Because we need to learn as much as possible information from the sequence, we need a big enough hidden

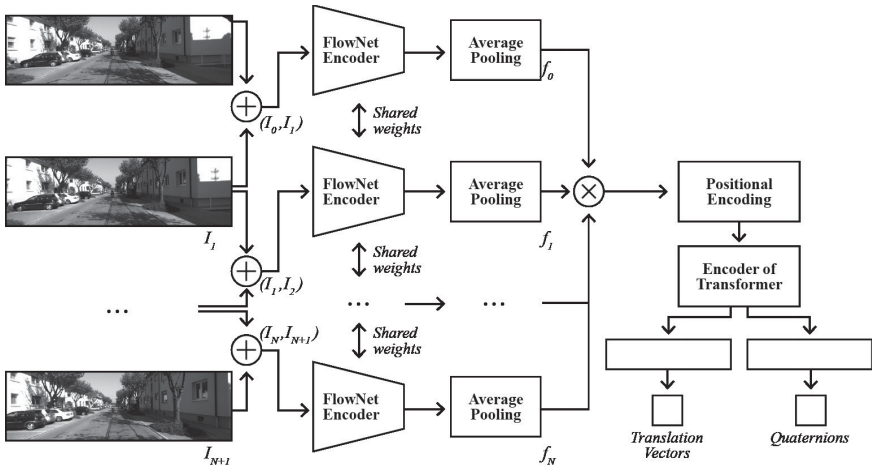

Fig. 1. The architecture of the proposed method for visual odometry. The dimensions of the tensors depend on input image size. Camera image credit: KITTI dataset [15]. $\bigoplus$ means concatenation over the channel, $\bigotimes$ — stack of tensors.

size, for example, 1000 [7]. It means, for our case we need linear layers with about $3 \cdot 10^7$ parameters. It is too much and presents this layer as a bottleneck.

To address this problem we use average pooling for each feature map. Finally, after the feature extractor, we have a tensor with dimension $[batch\_size, N, 1024]$. We denote features from CNN after average pooling as $\{f_0, \dots, f_N\}$:

$$f_N = AvgPool(CNN(< I_N, I_{N+1} >)) \tag{1}$$

where $<,>$ — concatenation images over channels.

C. Encoder of transformer

EoT [14] allows the model to extract sequential information from a stack of features for pose refining. We treat the pose estimation as a machine translation task, where context matters. So, EoT translates features from CNN ($\{f_0, \dots, f_N\}$) to high-level representation of camera pose in an one-to-one correspondence. Information about relative positions of frames we inject using position encoding to input:

$$\{f'_0, \dots, f'_N\} = PositionalEncoder\left(\{f_0, \dots, f_N\}\right) \tag{2}$$

$$\{t_0, \dots, t_N\} = EoT\left(\{f'_0, \dots, f'_N\}, \{f_0, \dots, f_N\}\right) \tag{3}$$

In this way, context embeddings allow the model not to forget about the original geometric features from CNN, input embeddings include position information and allow the network to refine relative poses in the right order by multi-head self-attention blocks. Also here we save on calculations because we do not need a decoder block from the transformer and any processing of features from CNN.

D. Pose regression

For transformation high-level features from EoT to relative poses we use fully connected layers. Our representation of relative pose contains two parts: translation of rotations and coordinate. We use quaternion $q_{i,i+1} = (q_w, q_x, q_y, q_z)$ for angles, because euler angles have gimble lock and rotation matrix have a lot degrees of freedom [31]. For coordinate translation we use vector with three values $T_{i,i+1} = (x, y, z)$.

Quaternions are one of the best choice [31] for deep learning because they are easily formulated in a continuous and differentiable way. But the main problem with them is that they have two unique values (from each hemisphere) that map to a single rotation. To avoid this we constrain all quaternions to one hemisphere.

E. Loss for training

The loss function for training includes several components, which we describe below.

1) Geodesic loss: To avoid computational errors with transformation ground truth to quaternions, we decide convert predicted quaternions to rotation matrices according to the following formula:

$$R(q) = 2 \begin{bmatrix} 0.5 - q_y^2 - q_z^2 & q_x q_y - q_w q_z & q_x q_z + q_w q_y \\ q_x q_y + q_w q_z & 0.5 - q_x^2 - q_z^2 & q_y q_z - q_w q_x \\ q_x q_z - q_w q_y & q_y q_z + q_w q_x & 0.5 - q_x^2 - q_y^2 \end{bmatrix} \tag{4}$$

During training, we want to reduce the error between the predicted rotation matrix and from the ground truth. To address that, we find the distance between rotation matrices, which is useful for pose estimation problems [32]:

$$L_r(R_1, R_2) = \arccos \frac{\mathrm{tr}(R_1 R_2^T) - 1}{2} \tag{5}$$

where R_1, R_2 — rotation matrices.

2) Odometry loss: For reducing error in the predicted relative pose we propose a simple loss function that minimizes the distance between relative rotations and coordinates. For angles we use Eq. 5, for coordinates, we combine Mean Absolute Error (MAE) and Mean Squared Error (MSE):

$$L_{r_{odom}} = \frac{1}{N} \sum_{i=0}^{N} L_r(R_{i,i+1}, \hat{R}_{i,i+1}) \tag{6}$$

$$L_{x_{odom}} = \frac{1}{N}\sum_{i=0}^{N}\left(\left|\left|T_{i,i+1} - \hat{T}_{i,i+1}\right|\right|_1 + \left|\left|T_{i,i+1} - \hat{T}_{i,i+1}\right|\right|_2^2\right) \tag{7}$$

$$L_{odom} = \alpha_1 L_{x_{odom}} + \alpha_2 L_{r_{odom}} \tag{8}$$

where $R_{i,i+1}, T_{i,i+1}$ relative rotation matrix and transformation from ground truth between I_i and I_{i+1}, $\hat{R}_{i,i+1}, \hat{T}_{i,i+1}$ — from prediction of model, α_1, α_2 — constants for scaling between rotations and coordinates.

Odometry loss (Eq. 8) minimizes the error of relative pose between two frames. We suppose this constraint allows the model to be more robust and not be adapted to the environment [5].

3) Geometric loss: Eq. 8 enforces the local consistency between the two frames. But the small error in relative pose can produce a large error for the global pose. To avoid this, we calculate geometrics loss [5], which find error between global poses.

To achieve that, firstly we present relative poses in the homogeneous coordinates:

$$P_{i,i+1} = \begin{bmatrix} R_{i,i+1} & T_{i,i+1} \\ 0 & 1 \end{bmatrix} \tag{9}$$

Secondly, we calculate global poses:

$$P_{i+1} = P_{zero} P_{0,1} P_{1,2} \dots P_{i,i+1} \tag{10}$$

where P_{i+1} — global pose at frame $i+1$, P_{zero} — pose in origin.

After that we calculate the loss function according to the following formulas:

$$L_{r_{geo}} = \frac{1}{N}\sum_{i=0}^{N} L_r(R_i, \hat{R}_i) \tag{11}$$

$$L_{x_{geo}} = \frac{1}{N}\sum_{i=0}^{N}\left(\left|\left|T_i - \hat{T}_i\right|\right|_1 + \left|\left|T_i - \hat{T}_i\right|\right|_2^2\right) \tag{12}$$

$$L_{geo} = \beta_1 L_{x_{geo}} + \beta_2 L_{r_{geo}} \tag{13}$$

where R_i, T_i global rotation matrix and transformation from ground truth at frame I_i, $\hat{R}_i, \hat{T}_i$ — from prediction of model, β_1, β_2 — constants for scaling between rotations and coordinates.

4) Overall loss: The overall loss function in our work can be expressed by the following:

$$L_{over} = \lambda_1 L_{geo} + \lambda_2 L_{odom} \tag{14}$$

where λ_1, λ_2 — constants for scaling between two loss functions.

IV. Experiments

Here we evaluate the performance and speed of our proposal. We mainly use KITTI [15] dataset for benchmarking performance. For speed calculation, we run our model on NVIDIA Jetson Nano, NVIDIA Jetson AGX Xavier, and Personal Computer. The code for training and testing is available at *https://github.com/toshiks/TBDVO*.

A. Implementation and training

The model is implemented in Pytorch [33] and Pytorch-lightning [34]. For all our experiments we set $\lambda_1 = \lambda_2 = 1$, $\alpha_1 = \beta_1 = 1$, $\alpha_2 = \beta_2 = 10$. It allows to achieve the balance between the rotation and the translation errors, because the rotation error much smaller than the translation. We use AdamW optimizer without changing default parameters. The similar approach for choosing the coefficients is described in works [5], [7]. The initial learning rate is set to 0.0004 and goes down by $\gamma = 0.8$ every 10 epoches.

We use pre-trained FlowNet [3] weights to initialize CNN backbone. For the rest of the parameters, we used Xavier initialization. The hidden size for the encoder of the transformer is 1024.

We train the model on two NVIDIA GeForce RTX 2070 with batch size set to 8 and sequence length set to 7 frames. The image size fed into the network is resized to 180×600.

To make the model more robust we use augmentations from Albumentations [35]. We randomly applied over each sequence RGBShift, RandomGamma, RandomBrightnessContrast, ColorJitter. Also, we flip sequence to balance the dataset by the value of the rotation angles and skip frames (with probability 0.3) to balance by the speed of a car.

B. KITTI Dataset

For a fair comparison with existing methods, we use KITTI Visual Odometry Benchmark [15]. The dataset includes 11 driving sequences with available ground truth. Dataset was recorded at a low frame rate (about 10 FPS) by driving in urban areas with the dynamic object. The approximate speed of the car was up to 60 km/h, it is challenging for monocular visual odometry algorithms.

For evaluating and testing we spilt the dataset into two parts. Sequence 00, 02, 08, and 09 we use for training and another to evaluate metrics.

C. Visual odometry

The performance of visual odometry is evaluated using KITTI Visual Odometry evaluation metrics [15], i.e., averaged Root Mean Square Errors (RMSEs) of the translation and rotation errors for all subsequences of length ranging from 100 to 800 meters.

We compare the performance of our model with deep learning approaches DeepVO [7] and its modification DeepVOM, and with monocular VISO2 [17] as a classic baseline. DeepVOM is our modification for reducing the number of parameters. To achieve that we use average pooling for each feature map from CNN.

TABLE I. RESULTS ON THE KITTI DATASET. DEEPVO [7], DEEPVOM AND OUR MODEL ARE SUPERVISED METHODS TRAINED ON SEQ 00, 02, 08 AND 09. VISO2-M [17] TRADITIONAL METHOD FOR MONOCULAR VISUAL ODOMETRY. THE BEST RESULTS ARE HIGHLIGHTED.

Method	Sequence													
	03		04		05		06		07		10		Avg	
	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}
VISO2-M [17]	8.47	8.82	4.69	4.49	19.22	17.58	7.30	6.14	23.61	29.11	41.56	32.99	17.48	16.52
DeepVO [7]	10.10	6.19	5.90	0.87	5.49	1.91	4.75	1.37	6.17	3.72	10.86	3.72	7.22	2.96
DeepVOM	**8.05**	4.51	6.14	**0.21**	**4.74**	**1.51**	4.48	**0.95**	**3.88**	**0.90**	7.75	2.37	5.84	**1.74**
Ours	8.10	**4.56**	**4.63**	1.44	5.13	1.59	**4.02**	1.25	5.26	2.29	**6.84**	**1.97**	**5.66**	2.18

t_{rel} : average translational RMSE drift (%) on length from 100, 200 to 800 m.
r_{rel} : average rotational RMSE drift (°/100m) on length from 100, 200 to 800 m.

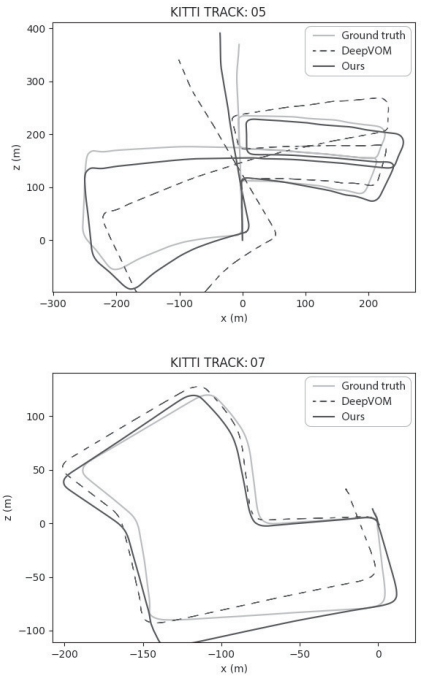

Fig. 2. Reconstructed trajectories of sequences 05 and 07 from the odometry benchmark of the KITTI dataset. [15]

As can be seen in Table I, the proposed approach is comparable in performance with DeepVO(M) [7] and outperforms VISO2-M [17]. Moreover, our solution has the best average score. We consider it is connected with the attention mechanism, which helps to produce better results due to the simultaneous processing of several frames. Examples of reconstructed trajectories we show on Fig. 2.

In Tables II, III, IV we show, that our solution significantly outperforms the LSTM-based model in terms of speed. Taking into account the results from the Table I, we can conclude that our solution has practically not lost quality, but gained speed.

D. Computational complexity analysis

Deep learning-based methods are generally considered to be computationally expensive. We make the architecture of our model faster than the classical RNN-based model due to parallelization and simultaneous processing of several frames by the encoder of the transformer. Moreover, some blocks inside the encoder can be cashed unlike LSTM, where processing is performed sequentially.

1) Devices: in this section we describe hardware environments in our experiments.

NVIDIA Jetson Nano a low-power, high capability, efficient GPU. This device comes with a 128-core integrated NVIDIA Maxwell GPU and a quad-core 64-bit ARM CPU. It has 4GB of LPDDR4 memory at 25.6GB/s and 5W/10W power modes. We flash an SD card with JetPack 4.6 and installed all required packages. We run all our tests on this device with 5W power mode.

NVIDIA Jetson AGX Xavier a low-power, high capability, efficient GPU. Much faster than the NVIDIA Jetson Nano. This device comes with a 512-core integrated NVIDIA Volta GPU and an 8-core 64-bit ARM CPU. It has 64GB of LPDDR4x memory at 136.5GB/s and 10W/15W/30W power modes. We install JetPack 4.6 onboard and all required packages. We run all our tests on this device with maximal power mode.

Personal computer contains modern 8-core 64-bit CPU AMD Ryzen 7 2700. It has 31GB of DDR4 memory at 17GB/s and NVIDIA GeForce RTX 2070. We install Ubuntu 20.04.2 LTS on the device with all required packages.

2) Benchmark implementation: in this section we describe how we implement benchmarks.

All the tests we perform on Pytorch [33]. We divide our tests into two parts.

In the first part of the experiments, we analyze the speed of the whole trained models. Because production usually uses caching, we apply this technique to cache previous results of CNN: we do not need to process by CNN all frames from sequence — the last two are sufficient. Such optimization significantly speeds up the solution and allows to estimate the impact of the sequential block of the model (all parts of models

TABLE II. RESULT OF BENCHMARKS ON NVIDIA JETSON NANO

	Jetson Nano	
	CPU [ms]	GPU [ms]
DeepVOM (Full)	2015.93 ± 19.03	180.37 ± 1.26
Ours (Full)	$\mathbf{1865.30 \pm 16.03}$	$\mathbf{151.76 \pm 0.92}$
DeepVOM (w/o CNN)	231.00 ± 5.04	37.69 ± 0.68
Ours (w/o CNN)	$\mathbf{73.83 \pm 5.25}$	$\mathbf{11.26 \pm 0.78}$

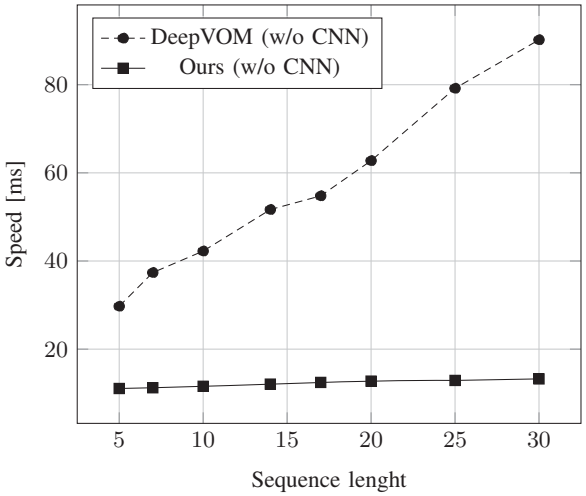

Fig. 3. Comparison of the speed of models at different sequence lengths running on the GPU of NVIDIA Jetson Nano.

after average poolings). We note this part of experiments as DeepVOM(Full) and Ours (Full).

In the second part, we analyze speed only sequential block of the model. It allows estimating the impact of the sequential block without CNN. We note this part of experiments as DeepVOM (w/o CNN) and Ours (w/o CNN).

Before each benchmark, we warm up a model, because the first iterations usually have low speed. During GPU tests we synchronize the interaction with the GPU, because otherwise some of the calculations may still be happening when we load the next elements.

Note here, the LSTM module has low-level optimization unlike the encoder of transformer or positional encoding, and even under these quite dishonest circumstances, our solution shows significant superiority.

3) Benchmars on NVIDIA Jetson Nano: presented on Table II. Our approach is about 3 times faster: as on GPU, as on CPU. Because CNN inference is quite slow on Jetson Nano, there is no significant increase in speed for whole models. Difference between them is about 1 FPS.

But if we need a more long sequence, DeepVOM significantly degrades. It happens because LSTM process sequence items one by one, but encoder of transformer simultaneously. We demonstrate this fact on Fig. 3. The speed of our solution was practically unchanged, at the same moment LSTM slowed down several times.

TABLE III. RESULT OF BENCHMARKS ON NVIDIA JETSON AGX XAVIER

	Jetson AGX Xavier	
	CPU [ms]	GPU [ms]
DeepVOM (Full)	482.73 ± 17.84	26.51 ± 0.13
Ours (Full)	$\mathbf{400.85 \pm 19.27}$	$\mathbf{23.01 \pm 0.10}$
DeepVOM (w/o CNN)	113.48 ± 0.78	5.27 ± 0.06
Ours (w/o CNN)	$\mathbf{47.48 \pm 0.48}$	$\mathbf{4.88 \pm 0.07}$

TABLE IV. RESULT OF BENCHMARKS ON PERSONAL COMPUTER

	Server	
	CPU (ms)	GPU (ms)
DeepVOM (Full)	55.59 ± 0.37	4.40 ± 0.30
Ours (Full)	$\mathbf{44.04 \pm 0.47}$	$\mathbf{3.67 \pm 0.34}$
DeepVOM (w/o CNN)	14.48 ± 0.06	1.25 ± 0.02
Ours (w/o CNN)	$\mathbf{3.47 \pm 0.02}$	$\mathbf{1.09 \pm 0.01}$

4) Benchmars on NVIDIA Jetson AGX Xavier: presented on Table III. This device is one of the most powerful among edge platforms. For mobile robots, this can be a great solution, because together with AI tasks, you can perform navigation and movement tasks, while the resources will remain. So benchmarks on that device are important to the industry.

Table III shows that our proposal works better (44 FPS vs 37 FPS for full models). Although the solution speed exceeds real-time (25 FPS) on the GPU, the DeepVOM model can degrade with increasing sequence length. In the case of the CPU, our solution is guaranteed to give 2 FPS, while the LSTM-based solution cannot provide such guarantees.

5) Benchmars on the Personal computer: presented on Table IV. The personal computer is the most powerful device of the considered. But even here our solution is much faster. On the CPU, our model almost reaches real-time (22 FPS) and the difference between the solutions is about 5 FPS. But as the sequence length increased, we also observed degradation of the DeepVOM speed.

V. CONCLUSION

We have proposed a novel approach for monocular visual odometry based on Transformer architecture. The proposed method does not require a large number of computing resources and sufficiently high metrics for real-world tasks. We have evaluated the approach on the KITTI dataset [15] and have compared it with an existed solution. We also have performed a computational complexity analysis on different devices. Our experiments show that the proposed method can be used on various autonomous devices with limited computing and energy resources. As the next step of our research, we plan to extend the proposed method for significantly longer sequences to reduce error accumulation over long distances.

REFERENCES

[1] H. Strasdat, J. Montiel, and A. J. Davison, "Scale drift-aware large scale monocular slam," *Robotics: Science and Systems VI*, vol. 2, no. 3, p. 7, 2010.

[2] N. Yang, R. Wang, and D. Cremers, "Feature-based or direct: An evaluation of monocular visual odometry," *arXiv preprint arXiv:1705.04300*, pp. 1–12, 2017.

[3] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2758–2766.

[4] A. Kendall, M. Grimes, and R. Cipolla, "Convolutional networks for real-time 6-dof camera relocalization," *CoRR*, vol. abs/1505.07427, 2015. [Online]. Available: http://arxiv.org/abs/1505.07427

[5] A. Valada, N. Radwan, and W. Burgard, "Deep auxiliary learning for visual localization and odometry," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 6939–6946.

[6] N. Radwan, A. Valada, and W. Burgard, "Vlocnet++: Deep multitask learning for semantic visual localization and odometry," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4407–4414, 2018.

[7] 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.

[8] R. Li, S. Wang, Z. Long, and D. Gu, "Undeepvo: Monocular visual odometry through unsupervised deep learning," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 7286–7291.

[9] F. Xue, X. Wang, S. Li, Q. Wang, J. Wang, and H. Zha, "Beyond tracking: Selecting memory and refining poses for deep visual odometry," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8575–8583.

[10] N. Yang, L. v. Stumberg, R. Wang, and D. Cremers, "D3vo: Deep depth, deep pose and deep uncertainty for monocular visual odometry," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1281–1292.

[11] X. Li, Y. Hou, P. Wang, Z. Gao, M. Xu, and W. Li, "Transformer guided geometry model for flow-based unsupervised visual odometry," *Neural Computing and Applications*, vol. 33, no. 13, pp. 8031–8042, 2021.

[12] C. Godard, O. Mac Aodha, M. Firman, and G. J. Brostow, "Digging into self-supervised monocular depth estimation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 3828–3838.

[13] N. Kaygusuz, O. Mendez, and R. Bowden, "Mdn-vo: Estimating visual odometry with confidence," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 3528–3533.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[15] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[16] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: an open-source library for real-time metric-semantic localization and mapping," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2020. [Online]. Available: https://github.com/MIT-SPARK/Kimera

[17] A. Geiger, J. Ziegler, and C. Stiller, "Stereoscan: Dense 3d reconstruction in real-time," in *IEEE Intelligent Vehicles Symposium*, Baden-Baden, Germany, June 2011.

[20] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004. [Online]. Available: https://doi.org/10.1023/B:VISI.0000029664.99615.94

[18] Z. Z. Nejad and A. H. Ahmadabadian, "Arm-vo: an efficient monocular visual odometry for ground vehicles on arm cpus," *Machine Vision and Applications*, vol. 30, no. 6, pp. 1061–1070, 2019.

[19] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Visual odometry and mapping for autonomous flight using an RGB-D camera," in *Robotics Research - The 15th International Symposium ISRR, 9-12 December 2011, Flagstaff, Arizona, USA*, ser. Springer Tracts in Advanced Robotics, H. I. Christensen and O. Khatib, Eds., vol. 100. Springer, 2011, pp. 235–252. [Online]. Available: https://doi.org/10.1007/978-3-319-29363-9_14

[21] H. Bay, T. Tuytelaars, and L. V. Gool, "Surf: Speeded up robust features," in *European conference on computer vision*. Springer, 2006, pp. 404–417.

[22] J. Shi *et al.*, "Good features to track," in *1994 Proceedings of IEEE conference on computer vision and pattern recognition*. IEEE, 1994, pp. 593–600.

[23] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE robotics & automation magazine*, vol. 18, no. 4, pp. 80–92, 2011.

[24] 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.

[25] X. Yin, X. Wang, X. Du, and Q. Chen, "Scale recovery for monocular visual odometry using depth estimated with deep convolutional neural fields," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 5870–5878.

[26] C. Godard, O. Mac Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 270–279.

[27] R. Mahjourian, M. Wicke, and A. Angelova, "Unsupervised learning of depth and ego-motion from monocular video using 3d geometric constraints," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 5667–5675.

[28] H. Zhan, R. Garg, C. S. Weerasekera, K. Li, H. Agarwal, and I. Reid, "Unsupervised learning of monocular depth estimation and visual odometry with deep feature reconstruction," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 340–349.

[29] X.-Z. Cui, Q. Feng, S.-Z. Wang, and J.-H. Zhang, "Monocular depth estimation with self-supervised learning for vineyard unmanned agricultural vehicle," *Sensors*, vol. 22, no. 3, p. 721, 2022.

[30] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *ACM Computing Surveys (CSUR)*, 2021.

[31] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5745–5753.

[32] D. Q. Huynh, "Metrics for 3d rotations: Comparison and analysis," *Journal of Mathematical Imaging and Vision*, vol. 35, no. 2, pp. 155–164, 2009.

[33] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[34] W. Falcon and The PyTorch Lightning team, "PyTorch Lightning," 3 2019. [Online]. Available: https://github.com/PyTorchLightning/pytorch-lightning

[35] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, "Albumentations: Fast and flexible image augmentations," *Information*, vol. 11, no. 2, 2020. [Online]. Available: https://www.mdpi.com/2078-2489/11/2/125