





PRGFlow: Unified SWAP-aware deep global optical flow for aerial robot navigation

Nitin J. Sanket,  Chahat Deep Singh, 
Cornelia Fermüller,  and Yiannis Aloimonos 
Perception and Robotics Group, University of Maryland, College Park
✉ Email: nitin@umiacs.umd.edu

Global optical flow estimation is the foundation stone for obtaining odometry which is used to enable aerial robot navigation. However, such a method has to be of low latency and high robustness whilst also respecting the size, weight, area and power (SWAP) constraints of the robot. A combination of cameras coupled with inertial measurement units (IMUs) has proven to be the best combination in order to obtain such low latency odometry on resource-constrained aerial robots. Recently, deep learning approaches for visual inertial fusion have gained momentum due to their high accuracy and robustness. However, an equally noteworthy benefit for robotics of these techniques are their inherent scalability (adaptation to different sized aerial robots) and unification (same method works on different sized aerial robots). To this end, we present a deep learning approach called PRGFlow for obtaining global optical flow and then loosely fuse it with an IMU for full 6-DoF (Degrees of Freedom) relative pose estimation (which is then integrated to obtain odometry). The network is evaluated on the MSCOCO dataset and the dead-reckoned odometry on multiple real-flight trajectories without any fine-tuning or re-training. A detailed benchmark comparing different network architectures and loss functions to enable scalability is also presented. It is shown that the method outperforms classical feature matching methods by $2\times$ under noisy data. The supplementary material and code can be found at <http://prg.cs.umd.edu/PRGFlow>.

Introduction: A fundamental competence of aerial robots [1] is to estimate ego-motion or odometry before any control strategy is employed [2]. Different sensor combinations have been used previously to aid the odometry estimation with LIDAR based approaches topping the accuracy charts [3]. However such approaches cannot be used on smaller aerial robots due to their size, weight, area and power (SWAP) constraints (Figure 1). Such small aerial robots are generally preferred due to safety, agility and usability as swarms [4]. In the last decade, imaging sensors have struck the right balance considering accuracy and general sensor utility by utilising classical feature matching algorithms [5]. However, visual data is dense and requires a lot of computation, which creates challenges for low-latency applications. To this end, sensor fusion experts proposed to use IMUs along with imaging sensors, because IMUs are lightweight and are generally available on aerial robots [6, 7].

In the last five years, deep learning based approaches for visual inertial fusion have gained momentum [8]. convolutional neural networks (CNNs) that regress relative camera pose (called Pose networks) from a pair of images show that they can provide accuracy results comparable to those of classical VO algorithms but lack generalisation across datasets [9]. Such generalisation performance is however better observed in networks which produce outputs based on image pixel similarities such as optical flow [10]. Furthermore, a critical issue with deep networks for odometry/flow estimation is that to have the same accuracy as classical approaches they are generally computationally heavy leading to larger latency. However, leveraging hardware acceleration and better parallelisable architectures can mitigate this problem.

In this work, we present a method for global optical flow estimation (translational velocities) using deep learning called PRGFlow targeted towards a down/up-facing camera. We then couple this information with an altimeter source and an inertial measurement units (IMUs) to obtain the full 6-DoF (degrees of freedom) pose for aerial robot navigation.

A summary of our contributions are: (1) A deep learning approach to estimate odometry using visual, inertial and altimeter data. (2) A comprehensive benchmark of different network architectures, hardware architectures and loss functions. (3) Extensive real-flight experiments using PRGFlow without any fine-tuning or re-training to demonstrate the robustness of our method.



Fig. 1 Size comparison of various components used on quadrotors. (a) Snapdragon Flight, (b) PixFalcon, (c) 120 mm quadrotor platform with NanoPi Neo Core 2, (d) MYNT EYE stereo camera, (e) Google Coral USB accelerator, (f) Sipeed Maix Bit, (g) PX4Flow, (h) 210 mm quadrotor platform with Coral Dev board, (i) 360 mm quadrotor platform with Intel Up board, (j) 500 mm quadrotor platform with NVIDIA® Jetson™ TX2. Note that all components shown are to relative scale. All the images in this paper are best viewed in color on a computer screen at a zoom of 200%

PRGFlow framework: To compute global optical flow, we first model the relative pose transformation as a function of image matching as explained next. Let $\mathbf{x}_t, \mathbf{x}_{t+1}$ be the homogeneous point correspondences in image frames at t and $t+1$ respectively. Now, the transformation between the two image frames can be expressed as $\mathbf{x}_{t+1} = \mathbf{H}_t^{t+1} \mathbf{x}_t$, where \mathbf{H}_t^{t+1} represents the non-singular 3×3 transformation matrix between the two frames. In general, \mathbf{H}_t^{t+1} is a non-linear function of the 3D rotation matrix \mathbf{R}_t^{t+1} and 3D translation vector \mathbf{T}_t^{t+1} . However, for certain scene structures \mathbf{H}_t^{t+1} simplifies to a linear function. Such a scenario happens when: 1. Real-world area is planar or near-planar, 2. Focal length is large. This scenario is also called homography where \mathbf{H}_t^{t+1} is called the homography matrix. From \mathbf{H}_t^{t+1} , we can recover a finite number of $\{\mathbf{R}_t^{t+1}, \mathbf{T}_t^{t+1}\}$ solutions. However, in a practical scenario, this decomposition is noisy as the errors in \mathbf{R}_t^{t+1} and \mathbf{T}_t^{t+1} are coupled which is highly undesirable.

Complementary sensors help mitigate this problem. IMUs, which are readily available on aerial robots, can provide accurate angle measurements within a small interval. We compute the rotation estimates from a Magdwick filter [11] on IMU data to rotation compensate (stabilise in roll, pitch and yaw) the input images similar to [10, 12]. Once the rotation has been recovered (removed) using the IMU, the problem of estimating ego-motion reduces to finding \mathbf{T}_t^{t+1} (2D translation and zoom) which is global optical flow and scale. This is because, \mathbf{H}_t^{t+1} can be decomposed into simpler transformations such as in-plane rotation (yaw), zoom (scale), translation and out-of-plane rotations (pitch + roll) which renders it to be decomposed to \mathbf{R}_t^{t+1} and \mathbf{T}_t^{t+1} using multiple information sources. We call this rotation-compensated transformation as pseudo-similarity since it is one DoF less than the similarity transformation (2D translation, zoom and yaw). We present a deep CNN for estimating pseudo-similarity which we call PRGFlow (PRG stands for our lab's name). Mathematically, the pseudo-similarity transformation is given in Equation (1), where W, H, s, t_x and t_y depict the image width, image height, scale, X -translation and Y -translation respectively.

$$\mathbf{x}_{t+1} = \begin{bmatrix} \frac{W}{2} & 0 & \frac{W}{2} \\ 0 & \frac{H}{2} & \frac{H}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1+s & 0 & t_x \\ 0 & 1+s & t_y \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x}_t \quad (1)$$

To obtain better performance, we utilise the inverse compositional spatial transformer networks (IC-STN) [13]. IC-STN proposed the concept of utilising multiple blocks to predict the final transformation. Each block predicts an incremental transformation over the previous block. The final predicted transformation is obtained by combining (by matrix multiplication) the outputs of all blocks. Such a conceptualisation allows to break down the transformation into simpler ones, for e.g. one can break down pseudo-similarity (PS) into translation (T) and scale (S) transformations. In particular, we extend the work in [13] to support pseudo-similarity and different warp types.

Table 1. Performance metric variation with different hardware

	VanillaNet		ResNet		SqueezeNet		MobileNet		ShuffleNet	
	S	L	S	L	S	L	S	L	S	L
Frames Pper Ssecond (FPS) ↑										
Intel® Up Board	10.30	3.52	7.96	2.60	9.14	0.69	7.61	3.90	1.30	–
Google CoralDev	1000.00	874.00	1111.00	720.00	312.00	27.64	909.00	666.00	–	–
Google CoralUSB	714.28	870.00	625.00	847.00	285.03	24.17	900.90	769.23	–	–
NanoPi	40.87	16.39	30.85	10.78	28.31	2.43	38.70	15.96	29.79	4.77
BananaPi	31.67	12.46	24.00	8.67	22.26	2.34	29.87	12.79	24.62	4.40
NVIDIA® Jetson™ TX2	232.30	123.71	197.02	27.07	224.39	135.91	231.79	135.75	143.16	35.63
i9-PC	920.81	520.26	768.45	385.40	831.06	130.83	655.53	384.08	518.30	172.55
TitanXp-PC	556.71	645.18	936.61	436.94	978.04	238.60	940.94	411.83	803.69	289.44
CoralUSB+NanoPi	125.16	130.32	140.79	223.71	–	–	145.60	236.43	–	–
i7-Laptop	301.36	155.12	224.07	123.08	264.74	35.39	230.13	136.97	–	–
1070-Laptop	980.35	566.72	798.57	528.83	932.25	406.36	823.65	494.38	–	–
GOPS ↓	0.12	0.37	0.15	0.51	0.08	4.22	0.11	0.30	0.07	1.09
MParams ↓	0.20	2.07	0.20	2.11	0.14	3.14	0.21	2.06	0.19	2.08
Acc. (%) ↑	70.64	81.70	84.40	88.10	77.10	79.80	45.00	63.30	40.40	40.40
Acc./(MParams×10 ³) ↑	0.35	0.04	0.42	0.04	0.57	0.03	0.22	0.03	0.21	0.02
Acc./(GOP×10 ³) ↑	0.59	0.22	0.56	0.17	0.96	0.02	0.42	0.21	0.61	0.04
Accuracy×FPS/10 ³ ↑										
Intel® Up Board	0.73	0.29	0.67	0.23	0.70	0.05	0.34	0.25	0.05	–
Google CoralDev	69.70	70.53	92.77	63.43	23.74	2.18	38.36	40.96	–	–
Google CoralUSB	49.79	70.21	52.19	74.62	21.69	1.91	38.02	47.31	–	–
NanoPi	2.85	1.32	2.58	0.95	2.15	0.19	1.63	0.98	–	–
BananaPi	2.21	1.01	2.00	0.76	1.69	0.18	1.26	0.79	–	–
NVIDIA® Jetson™ TX2	16.41	10.11	16.63	2.38	17.30	10.85	10.43	8.59	5.78	1.44
i9-PC	65.05	42.51	64.86	33.95	64.07	10.44	29.50	24.31	20.94	6.97
TitanXp-PC	39.33	52.71	79.05	38.49	75.41	19.04	42.34	26.07	32.47	11.69
CoralUSB+NanoPi	8.72	10.52	11.76	19.71	–	–	6.14	14.54	–	–
i7-Laptop	21.29	12.67	18.91	10.84	20.41	2.82	10.36	8.67	–	–
1070-Laptop	68.33	45.73	66.68	46.59	70.94	32.06	34.76	30.40	–	–

First, second and third best result. See supplementary material for a detailed description of the hardware modules.

Dataset, training and testing details: We train and test all our networks on the MS-COCO dataset [14] using the `train2014` and `test2014` splits for training and testing respectively to predict the warp parameters: $\tilde{\mathbf{h}} = [\tilde{s} \tilde{t}_x \tilde{t}_y]^T$. During training, we obtain a random crop of size 300×300 px. which is then warped using pseudo-similarity to generate synthetic data using a random warp parameter in the range $\gamma_1 = \pm [0.25 \ 0.20 \ 0.20]$ (unless specified otherwise). Then the center 128×128 px. patches are extracted (to avoid boundary effects) and are stacked to obtain an input shape of $128 \times 128 \times 2N_c$ (N_c is the number of channels in each patch). Our networks were trained in Python 2.7 using TensorFlow1.14 on a desktop computer (See supplementary for details) running Ubuntu 18.04. We trained all our networks with a mini-batch-size of 32 using the ADAM optimiser for 100 epochs with early termination if we detect over-fitting on the validation set.

We tested our trained networks (trained only using in-domain range of data) using two different configurations: (1) In-domain and (2) Out-of-domain. In the in-domain testing (not to be confused with in-training dataset), we warped images from the `test2014` split of the MS-COCO dataset using a random warp parameter in range $\gamma_1 = \pm [0.25 \ 0.20 \ 0.20]$ (same warp range as training). This was used for evaluation as described in evaluation metrics section. To comment on the generalisation of the approach, we also tested it on out-of-domain warps, i.e. twice the warp range it was originally trained on, denoted by $\gamma_2 = \pm [0.50 \ 0.40 \ 0.40]$. This test was constructed to highlight the generalisability vs. speciality of the network configuration.

Network architectures: We use five network architectures inspired by previous works: VanillaNet [15], ResNet [16], ShuffleNet [17], MobileNet [18], and SqueezeNet [19]. Each network is composed of various blocks for predicting incremental transformation as explained previously. The output of each block is used as an incremental warp as proposed in [13]. For each of these blocks, we use the same name as the network, for e.g. VanillaNet has VanillaNet blocks. We use the following shorthand to denote the architecture. VanillaNet_a denotes VanillaNet with *a* number of VanillaNet blocks. We modify the networks from their original papers in the following manner: we remove max-pooling blocks and replace them with stride in the previous convolutional block. Instead of varying sub-sampling rates (rates at which strides change with respect to depth of the network), we keep it fixed to the same value

at every layer. For ease of understanding, more details can be found in the supplementary material. We also test two different sizes of networks, i.e. large (model size ≈ 8.3 MB) and small (model size ≈ 0.83 MB). Here, the model sizes are computed for storing float32 values for each neuron weight.

Loss functions: The trivial way to learn the warp parameters is to use a supervised loss (since the labels are “free” as the data is synthetically generated on the fly). The loss function \mathcal{L}_s used in the supervised case is an l_2 distance between predicted parameters $\tilde{\mathbf{h}}$ and ideal parameters \mathbf{h} [10].

We also study the unsupervised losses which are generally complicated since they are under-constrained compared to the ones in supervised approaches. Here, we study different unsupervised loss functions whose form is given by:

$$\mathcal{L}_{us} = \underset{\tilde{\mathbf{h}}}{\operatorname{argmin}} \mathbb{E}[\mathcal{D}\{\mathcal{W}(\mathcal{P}_1, \tilde{\mathbf{h}}), \mathcal{P}_2\} + \lambda_i \mathcal{R}_i] \quad (2)$$

where \mathcal{W} is a generic differentiable warp function which can take on different mathematical formulations based on its second argument (model parameters), and \mathcal{D} represents a distance measuring image similarity between the image frames. Finally, λ_i (chosen by cross-validation) and \mathcal{R}_i represent the *i*th Lagrange multiplier and its corresponding regularisation function respectively. We experiment with different \mathcal{D} and \mathcal{R} functions described next. \mathcal{D}_{L1} represents the generic l_1 photometric loss [20] commonly used for traditional images, \mathcal{D}_{Chab} represents the Chabonier loss [21] commonly used for optical flow estimation, \mathcal{D}_{SSIM} represents the loss based on Structure Similarity [22] commonly used for learning ego-motion and depth from traditional image sequences and \mathcal{D}_{Robust} represents the robust loss function presented in [23]. These functions can also be used as metric functions or regularisers and can take any generic input such as the raw image or a function of the image. We experiment with different inputs such as the raw RGB image, grayscale image, high-pass filtered image and image cornerness score (denoted by \mathcal{I} , \mathcal{G} , $\mathcal{Z}(\mathcal{I})$ and $\mathcal{C}(\mathcal{I})$ respectively). We use the following shorthand for representing loss functions: $\mathcal{D}_{SSIM}(\mathcal{I}) + 5.0\mathcal{D}_{L1}(\mathcal{Z}(\mathcal{I}))$ represents a loss function with SSIM on raw RGB images as the metric function and photometric l_1 on high-pass filtered images as the regularisation function with a Lagrange multiplier of 5.0.

Evaluation metrics: The scale and translation error in px. are given as:

$$\mathcal{E}_{\text{scale}} = \mathbb{M}\left(\sqrt{0.5(W^2 + H^2)}|\tilde{s} - \hat{s}|\right);$$

$$\mathcal{E}_{\text{trans}} = \mathbb{M}\left(0.5\left(\sqrt{(W(\tilde{t}_x - \hat{t}_x))^2 + (H(\tilde{t}_y - \hat{t}_y))^2}\right)\right) \quad (3)$$

Here, \mathbb{M} denotes the median value (we choose the median value over the mean to reject outlier samples with low texture). We also convert errors to percentage accuracy as follows:

$$\mathcal{A} = (1 - ((\mathcal{E}_{\text{scale}} + \mathcal{E}_{\text{trans}})/(\mathbb{I}_{\text{scale}} + \mathbb{I}_{\text{trans}}))) \times 100\% \quad (4)$$

Here, $\mathbb{I}_{\text{scale}}$ and $\mathbb{I}_{\text{trans}}$ denote the identity errors (error when the prediction values are zero) for scale and translation respectively. For a detailed description of the computing platforms used, refer to the supplementary material.

Experimental results and discussion: Table 1 provides a benchmark tabulation of all the networks on various performance metrics and computing platforms. All the networks include both Small (denoted as S and has 0.83 MB model size) and Large (denoted as L and has 8.3 MB model size) configurations trained using supervised l_2 loss function. Up board, TX2, PC and Laptop use the original Float32 model; NanoPi and BananaPi use the Int8-TFLite post-quantisation optimised model and finally CoralDev, CoralUSB and CoralUSB+NanoPi use the Int8-EdgeTPU post-quantisation optimised model since they gave us the best performance. An important factor to realise is that using Int8-EdgeTPU post-quantisation optimisation to run on either the Coral Dev board or the Coral USB accelerator can provide significant speed-ups of up to 52 \times compared to the original Float32 model without significant loss in accuracy.

Table 2(a) shows the results for different warp combinations where we use the following shorthand: PS \times 1 means that we have one pseudo-similarity block, T and S denote translation and scale blocks respectively. We can observe that as the number of warping blocks increases, the performance reaches a maximum and then deteriorates. This is because the number of neurons per warp block directly affects the performance. Increasing the number of warp blocks without increasing model size hurts accuracy as the number of neurons per warp block become small. Next, we study the performance of different network architectures. The results for large and small models can be found in Table 2(b, c) respectively. One can clearly observe that ResNet gives the best performance for both small and large networks and should be the choice when designing for maximum accuracy without any regard to the number of parameters or amount of OPS (operations). However, if one has to prioritise maximising accuracy whilst minimising number of parameters, then SqueezeNet and ResNet would be the choice for smaller and larger networks respectively. Another trend to observe is that we need 10 \times more parameters for a 19% increase in accuracy. Lastly, if one has to prioritise in maximising accuracy whilst minimising the number of OPS, then SqueezeNet and MobileNet would be the choice for smaller and larger networks respectively. Clearly, the most optimal architecture in-terms of accuracy, number of parameters and OPS is SqueezeNet for smaller networks and ResNet for larger networks. To gather more insight of what data input is well suited for pseudo-similarity estimation, we explore training and testing on the following inputs: (a) RGB image, (b) grayscale image, (c) high pass filtered image (Table 2(d)). Surprisingly, training on RGB images and evaluating on RGB images gives worse performance in the testing both for in-domain (γ_1) and out-of-domain (γ_2) ranges as compared to testing on grayscale data. We speculate that this is due to conflicting information in multiple channels. Another surprising observation is that training and/or testing on high-pass filtered images results in large errors, which is contrary to the classical approaches. We speculate that this is because CNNs rely on “staticity” of the image pixels (image pixels change slowly and are generally smooth).

From Table 2(e), we observe that the supervised network performs better than most unsupervised networks on out-of-domain tests. This hints that we need better loss functions for unsupervised methods and better network architectures to take advantage of these unsupervised losses.

Table 2. Pseudo-similarity estimation: A quantitative evaluation

Network (warping)	$\mathcal{E}_{\text{scale}}$ (px.)		$\mathcal{E}_{\text{trans}}$ (px.)		FLOPS (G) ↓	MParams ↓
	γ_1	γ_2	γ_1	γ_2		
(a) Different warp combinations for VanillaNet						
Identity	11.4	22.8	10.3	20.4	—	—
VanillaNet ₁ (PS×1)	2.4	15.0	1.3	12.5	0.37	2.07
VanillaNet ₁ (PS×1) DA*	4.1	17.7	2.3	14.2	0.37	2.07
VanillaNet ₂ (PS×2)	2.2	9.9	1.4	12.4	0.42	2.17
VanillaNet ₂ (S×1, T×1)	2.5	15.2	1.5	12.2	0.46	2.10
VanillaNet ₂ (T×1, S×1)	2.5	15.1	1.5	12.5	0.42	2.15
VanillaNet ₄ (PS×4)	2.3	11.9	1.5	14.9	0.42	2.15
VanillaNet ₄ (S×2, T×2)	2.6	15.4	1.6	12.6	0.46	2.08
VanillaNet ₄ (T×2, S×2)	2.0	8.5	1.5	12.5	0.46	2.08
VanillaNet ₄ (T×2, S×2) γ_2^\dagger	2.7	2.8	4.6	7.2	0.46	2.08
(b) Different network architecture using T×2, S×2 warping block for large model						
VanillaNet ₄	1.9	6.4	1.5	12.4	0.46	2.08
ResNet ₄	1.7	15.1	0.9	10.1	0.59	2.12
SqueezeNet ₄	2.1	5.7	2.2	13.8	2.20	2.12
MobileNet ₄	4.0	14.2	1.6	12.0	0.41	2.04
ShuffleNet ₄	6.4	17.4	3.0	13.9	1.20	2.10
(c) Different network architecture using T×2, S×2 warping block for small model						
VanillaNet ₄	3.3	8.9	3.1	14.0	0.18	0.21
ResNet ₄	4.4	12.5	2.4	12.1	0.20	0.20
SqueezeNet ₄	2.4	5.6	4.0	14.9	0.19	0.20
MobileNet ₄	8.3	18.7	3.7	13.4	0.16	0.20
ShuffleNet ₄	8.3	17.6	4.6	15.7	0.13	0.21
(d) Different network inputs using T×2, S×2 warping block for large VanillaNet ₄ model						
Identity	11.4	22.8	10.3	20.4	—	—
\mathcal{I} (\mathcal{I})	1.9	6.4	1.5	12.4	0.46	2.08
\mathcal{G} (\mathcal{I})	1.8	6.3	1.5	12.3	0.46	2.08
\mathcal{G} (\mathcal{G})	2.7	14.1	1.6	12.7	0.46	2.08
$\mathcal{Z}(\mathcal{I})$ ($\mathcal{Z}(\mathcal{I})$)	13.1	9.4	10.4	16.0	0.46	2.08
\mathcal{G} ($\mathcal{Z}(\mathcal{I})$)	11.8	20.7	9.8	19.8	0.46	2.08
\mathcal{I} ($\mathcal{Z}(\mathcal{I})$)	13.1	22.5	10.5	20.1	0.46	2.08
$\mathcal{Z}(\mathcal{I})$ (\mathcal{G})	8.5	19.8	4.1	17.6	0.46	2.08
$\mathcal{Z}(\mathcal{I})$ (\mathcal{I})	17.2	20.1	4.2	17.4	0.46	2.08
(e) Different loss functions using PS×1 warping block for large VanillaNet ₁ model						
Identity	11.4	22.8	10.3	20.4	—	—
Supervised \mathcal{L}_s (VanillaNet ₁)	2.4	15.0	1.3	12.5	0.37	2.07
$\mathcal{D}_{\text{Robust}}(\mathcal{I}, \mathcal{C}(\mathcal{I}))$ (ResSqueezeNet ₁)	12.9	25.2	7.2	11.7	1.01	2.18
$\mathcal{D}_{\text{SSIM}}(\mathcal{I})$ (ResSqueezeNet ₁)	3.4	21.2	6.0	13.8	1.01	2.18
$\mathcal{D}_{\text{SSIM}}(\mathcal{I}) + 0.1\mathcal{D}_{\text{L1}}(\mathcal{C}(\mathcal{I}))$ (ResSqueezeNet ₁)	2.0	16.1	6.2	14.6	1.01	2.18
$\mathcal{D}_{\text{SSIM}}(\mathcal{I}) + 0.1\mathcal{D}_{\text{L1}}(\mathcal{Z}(\mathcal{I}))$ (ResSqueezeNet ₁)	2.7	16.6	6.4	13.6	1.01	2.18
$\mathcal{D}_{\text{L1}}(\text{DB}(E))$ [10]	5.4	17.7	3.7	16.5	4.92	3.6
$\mathcal{D}_{\text{Chab}}(\text{DB}(E))$ [10]	5.1	17.1	3.4	16.7	4.92	3.6
Supervised DB(E) [10]	4.1	16.2	3.3	15.1	4.92	3.6

*Trained and tested with Gaussian noise, hue+saturation shifts, brightness, contrast and gamma changes.

\dagger Trained with shifts of γ_2 .

We also observed that under noise (changes to brightness, contrast, hue, saturation, gamma and additive Gaussian noise), PRGFlow works better than classical feature matching algorithms with about 2 \times accuracy. These results can be found in the supplementary material.

In the real-flight tests (see supplementary for details on data collection), we use the predictions \tilde{h} (ResNet₄ T \times 2, S \times 2 large model trained using supervised l_2 loss) which are obtained every four frames and integrated using dead-reckoning to obtain the final trajectory. The average RMSE_{ate} [24] for various trajectories shown in Figure 2 is less than 2% of trajectory length (even with severe noise) and is about 8 \times better than PX4Flow [12] estimates. Avg. RMSE_{ate} in PRGFlow trajectory estimates on noisy data is lower than that of PX4Flow estimates even on noiseless data by 10%. An important point to note is that, in serve noise, the scale estimates from classical approaches are generally close to identity error.

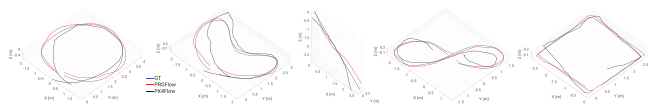


Fig. 2 Comparison of trajectory obtained by dead-reckoning (red) our estimates with respect to the ground truth (blue) and PX4Flow (black) estimates for quadrotor flight in various trajectory shapes (left to right): Circle, Moon, Line, Figure8 and Square

Conclusion: We presented a method to estimate global optical flow on an aerial robot using deep learning. This information is then combined with an IMU and an altimeter to obtain full 6-DoF pose. We provided comprehensive analysis of warping combinations, network architectures and loss functions. We also benchmarked all our approaches on commonly used hardware with different SWAP constraints for speed and accuracy. We hope this benchmark will serve as a reference manual for researchers and practitioners alike for designing neural networks for their specific applications. We also show extensive real-flight odometry (obtained by integrating the pose obtained) results, highlighting the robustness of PRGFlow without any fine-tuning or re-training. Finally, as a parting thought, utilising deep networks for estimating global optical flow on a noisy data would most likely lead to a more robust system.

Acknowledgments: The support of the Brin Family Foundation, the Northrop Grumman Mission Systems University Research Program, ONR under grant award N00014-17-1-2622 and National Science Foundation under grant BCS 1824198 are gratefully acknowledged.

© 2021 The Authors. *Electronics Letters* published by John Wiley & Sons Ltd on behalf of The Institution of Engineering and Technology

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.

Received: 16 March 2021 doi: 10.1049/ell2.12274

References

- Nitin, J. S., et al.: GapFlyt: Active vision based minimalist structure-less gap detection for quadrotor flight. *IEEE Robot. Automat. Lett.* **3**(4), 2799–2806 (2018)
- Nathan, M., et al.: Collaborative mapping of an earthquake-damaged building via ground and aerial robots. *J. Field Robot.* **29**(5), 832–841 (2012)
- Ji, Z., et al.: LOAM: Lidar odometry and mapping in real-time. *In Robot.: Sci. Syst.* **2**, 1–9 (2014)
- Weinstein, A., et al.: VIO-Swarm: A swarm of 250g quadrotors. *IEEE RA-L Robot. Automat. Lett.* **3**(3), 1801–1807 (2018)
- Morgan, Q., et al.: The open vision computer: An integrated sensing and compute system for mobile robots. In: 2019 Int. Conf. Robot. Automat. (ICRA), pp. 1834–1840. IEEE, Piscataway, NJ (2019)
- Anastasios, I. M., et al.: A multi-state constraint kalman filter for vision-aided inertial navigation. In: Proc. 2007 IEEE Int. Conf. on Robotics and Automation, pp. 3565–3572. IEEE, Piscataway, NJ (2007)
- Michael, B., et al.: Robust visual inertial odometry using a direct EKF-based approach. In: 2015 IEEE/RSJ int. conf. intell. robots syst. (IROS), pp. 298–304. IEEE, Piscataway, NJ (2015)
- Ronald, C., et al.: VINet: Visual-inertial odometry as a sequence-to-sequence learning problem. arXiv:1701.08376 (2017)
- Alex, K., et al.: Posenet: A convolutional network for real-time 6-dof camera relocation. In: Proc. IEEE Int. Conf. on Computer Vision, pp. 2938–2946. IEEE, Piscataway, NJ (2015)
- Nitin, J. S., et al.: Evdodgenet: Deep dynamic obstacle dodging with event cameras. In: 2020 IEEE Int. Conf. Robot. Automat. (ICRA), pp. 10651–10657. IEEE, Piscataway, NJ (2020)
- Sebastian, O. H. M., et al.: Estimation of imu and marg orientation using a gradient descent algorithm. In: 2011 IEEE Int. Conf. on Rehabilitation Robotics, pp. 1–7. IEEE, Piscataway, NJ (2011)
- Dominik, H., et al.: An open source and open hardware embedded metric optical flow CMOS camera for indoor and outdoor applications. In: 2013 IEEE Int. Conf. Robot. Automation, pp. 1736–1741. IEEE, Piscataway, NJ (2013)
- Chen-Hsuan, L., et al.: Inverse compositional spatial transformer networks. In: IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), pp. 2568–2576. IEEE, Piscataway, NJ (2017)
- Tsung-Yi, L., et al.: Microsoft COCO: Common objects in context. In: European Conf. on Computer Vision, pp. 740–755. Springer, Berlin, Heidelberg (2014)
- Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556 (2015)
- Kaiming, H., et al.: Deep residual learning for image recognition. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition, pp. 770–778. IEEE, Piscataway, NJ (2016)
- Ningning, M., et al.: Shufflenet v2: Practical guidelines for efficient cnn architecture design. In: Proc. European Conf. on Computer Vision (ECCV), pp. 116–131. Springer, Berlin, Heidelberg (2018)
- Andrew, G. H., et al.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv:1704.04861 (2017)
- Forrest, N. I., et al.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5 mb model size. arXiv:1602.07360 (2016)
- Hang, Z., et al.: Loss functions for image restoration with neural networks. *IEEE Trans. Comput. Imag.*, **3**(1), 47–57, 2016.
- Sun, D., et al.: A quantitative analysis of current practices in optical flow estimation and the principles behind them. *Int. J. Comput. Vis.* **106**(2), 115–137 (2014)
- Zhou, W., et al.: Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Process.* **13**(4), 600–612 (2004)
- Barron, J.: A general and adaptive robust loss function. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4326–4334. IEEE, Piscataway, NJ (2019)
- Zichao, Z., et al.: A tutorial on quantitative trajectory evaluation for visual (-inertial) odometry. In: 2018 IEEE/RSJ Int. Conf. Intell. Robots and Syst. (IROS), pp. 7244–7251. IEEE, Piscataway, NJ (2018)