Estimating Metric Scale Visual Odometry from Videos using 3D Convolutional Networks

Alexander S. Koumis, James A. Preiss, and Gaurav S. Sukhatme

***Abstract*— We present an end-to-end deep learning approach for performing metric scale-sensitive regression tasks such visual odometry with a single camera and no additional sensors. We propose a novel 3D convolutional architecture, 3DC-VO, that can leverage temporal relationships over a short moving window of images to estimate linear and angular velocities. The network makes local predictions on stacks of images that can be integrated to form a full trajectory. We apply 3DC-VO to the KITTI visual odometry benchmark and the task of estimating a pilot's control inputs from a first-person video of a quadrotor flight. Our method exhibits increased accuracy relative to comparable learning-based algorithms trained on monocular images. We also show promising results for quadrotor control input prediction when trained on a new dataset collected with a UAV simulator.**

I. INTRODUCTION

Monocular visual odometry (VO) is a heavily studied topic in robotics as it enables robust 3D localization with a ubiquitous, cheap, lightweight sensor: a single camera. The goal of VO is to observe a sequence of images and estimate the motion of the camera that generated the sequence. Traditional geometric approaches to monocular VO suffer from scale ambiguity: it is not possible to tell whether an image sequence depicts small objects close to the camera, or large objects far away [1]. Scale accuracy can only be achieved with geometric methods in one of two ways: 1) by fusing information from a sensor that measures physical units, such as an inertial measurement unit (IMU) or global positioning system (GPS) receiver, or 2) by exploiting prior knowledge about objects in a scene, such as the the typical size of a detected vehicle.

Adding an IMU to the system is a straightforward solution that has delivered impressive results [2], [3]. However, one may also be interested in extracting camera trajectory data from pre-existing data sources, such as online videos, with no corresponding physical sensor data. In these scenarios, approach 2) is the only option. In this work, we are motivated by the eventual goal of performing imitation learning on acrobatic first-person view (FPV) quadrotor flight videos. This requires estimating the pilot's unknown control inputs to the quadrotor, a problem which can be reduced to estimating the trajectory of the quadrotor's onboard camera by the differential flatness property [4]. Since quadrotors move at high speed in the full six degrees of freedom (6DoF), FPV flight videos contain a considerably wider range of camera motions and positions than are found in typical videos. This motivates our design of a learning-based method that makes minimal assumptions about the objects or motion in the training data.

The authors are with the Department of Computer Science, University of Southern California, Los Angeles, USA. {koumis,japreiss,gaurav}@usc.edu

Fig. 1: The 3DC-VO network architecture can be trained to regress scale-accurate odometry information from video data. Shown above is a screen capture from a quadrotor simulator with superimposed predicted (red) and ground truth (blue) joystick positions. Joystick positions are equivalent to thrust (acceleration) and angular velocity of the quadrotor.

We assume access to a training dataset consisting of image sequences with corresponding camera trajectories. At test time, our method outputs an estimated camera trajectory from a single monocular image sequence. Our assumptions are most similar to those of [5]. We present a novel convolutional neural network architecture, 3DC-VO, adhering to the following design principles:

1) Support end-to-end training on monocular datasets with no assumptions about the environment, sensor properties, or system dynamics. This allows the work to be applicable towards a wider variety of robots and datasets.
2) Utilize a 3D convolutional architecture to consume time-series data without the slower and more complex training process of a recurrent neural network architecture. To our knowledge, this work represents the first application of 3D convolutions to monocular VO.
3) Target full 6DoF motion by avoiding any rotation representations susceptible to gimbal lock.

Our contributions include a full description of our proposed architecture along with experimental results showing it outperforming a comparable learning-based approach on the KITTI visual odometry benchmark on an automobile platform. We also exhibit accurate quadrotor control input predictions when trained on a dataset collected from a human pilot in a realistic flight simulator.

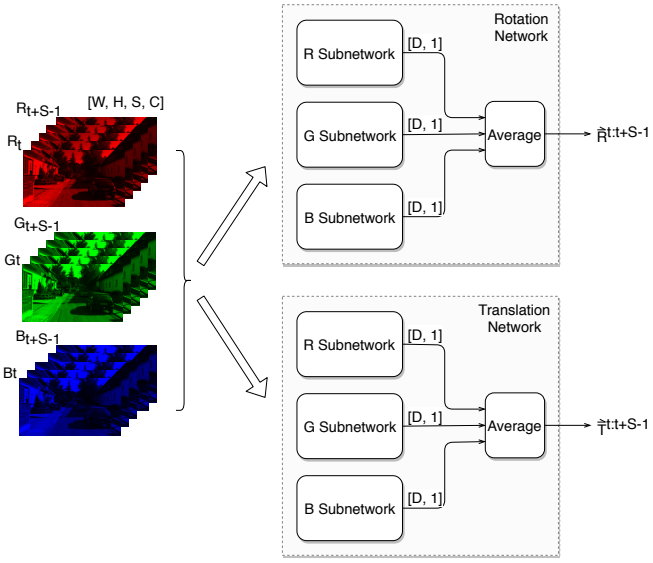


Fig. 2: High level 3DC-VO architecture. W, H, S, and C stand for width, height, stack size, and channels, respectively. D represents the number of dimensions of the output. For the KITTI task, each subnetwork outputs a single scalar, with the rotation network outputting the yaw offset between frame t and $t + S - 1$, and the translation network outputting the forward offset. For the quadrotor control estimation task, the rotation output is a three-dimensional angular velocity vector.

II. RELATED WORK

In this section we outline related work on VO, focusing only on methods that attempt to overcome the scale ambiguity problem. Existing monocular methods mostly fall into one of two categories; either inferring camera transformation by tracking geometric features explicitly, or by learning models to regress pose from pixel data. We describe related work from both categories. We note that VO is a vast field and we can only fit a small fraction of relevant work due to space limits.

A. Geometric methods

Sparse geometric methods operate by extracting features from images, matching them from frame to frame, and estimating the camera's relative pose offset based on their movement between frames. In this setting, metric scale is unobservable [1], [6]. Several solutions to this issue involve using information from extra sensors, such as IMUs or GPS receivers [2], [7], [3]. When adding hardware is not an option, methods must exploit *a priori* knowledge of metric properties of objects in the scene to infer scale. Perhaps due to the popularity of the KITTI benchmark, many such methods are car-centric: in [8], [9], the authors estimate scale from known sizes of detected vehicles, while in [10] scale is derived from known lane width, traffic sign sizes, and ceiling-to-floor height. Other methods use ground plane estimation along with an assumed distance from the camera to the ground [11], [12], [13]. [14] additionally incorporates car dynamics into their motion estimation. In contrast, our method targets general-purpose application and makes no assumptions specific to ground vehicles.

B. Learning-based methods

Learning-based approaches have the potential to estimate accurate metric scale by (implicitly or explicitly) learning information about the sizes of objects in the training data. Before the rise in popularity of deep learning in computer vision, work such as [15], [16], [17] explored methods such as Gaussian processes and support vector machines to compute frame to frame ego-motion. More recently, success has been found in using convolutional neural networks (CNNs) to estimate VO.

In [18], five-frame stereo subsequences are fed into a CNN with an input layer pretrained on a representation of depth and motion, which classifies local offset using discretized rotational and translational components. While this method uses a stack of sequential images larger than 2, it treats the images as channels in a standard 2D convolutional context, not explicitly taking advantage of temporal information. It also frames VO as a classification problem, introducing error due to discretization. Other works [19], [20] have exhibited excellent performance on the KITTI odometry benchmark [21] by learning to estimate a depth map from a single camera image, and incorporating the estimated depth map into a SLAM algorithm originally designed for sensors such as stereo cameras or LIDAR. These methods require an additional data source to train the depth estimator.

Several other approaches [22], [23] explore training CNNs on precomputed optical flow images to output local pose offsets, while [5], [24] feed learned optical flow representations into a recurrent Long Short Term Memory (LSTM) neural network [25] to predict the global offset. However, recent work has shown that models convolving over the time dimension can equal or exceed recurrent neural networks at learning tasks involving sequential input, while generally requiring less memory and training time [26]. Perhaps because the aerial robotics application has not been widely considered, some existing methods such as [5] output an orientation estimate as Euler angles in the inertial frame, creating the potential of gimbal lock when used with 6-DoF devices. In comparison, our system outputs short-horizon angular offset in the local coordinate system of the camera and integrates these estimates into a 3D rotation parameterization, free of gimbal lock issues.

For the task of regressing UAV control inputs, [27] trains a CNN to predict joystick inputs from human pilots in a drone racing simulation, with an observation of only one image. Their end-goal is slightly different than ours, as they aim to predict the pilot's next action instead of estimating the actions that produced the given image.

III. PRELIMINARIES

In this section, we formalize the metric-scale VO problem and our formulation thereof as a learning task suitable for a non-recurrent neural network. We use the following notation: $[n]$ denotes the set of integers $1, \ldots, n$. $\mathbb{U}^n$ denotes

the set of $n-$dimensional unit vectors under the Euclidean norm: $\mathbb{U}^n = \{x \in \mathbb{R}^n : \|x\|_2 = 1\}$. $\mathrm{Tr}(\cdot)$ denotes the matrix trace operator. $SO(3)$ denotes the group of three-dimensional rotations in Euclidean space under the operation of composition.

Consider the trajectory of a camera moving freely in three-dimensional space. At each time step $t \in [T]$, the camera's state can be described by a position $x_t \in \mathbb{R}^3$ and an orientation $R_t \in SO(3)$. We fix $x_0 = \mathbf{0}$ and $R_0 = I_{3\times 3}$; subsequent states are relative to this origin. At each time step t, the camera captures an image $I_t \in \mathbb{R}^{H\times W\times C}$, where (H, W) denotes the height and width of the input image in pixels and C denotes the number of color channels in the input (one for a grayscale image, three for an RGB image). The VO problem is the following: given the image sequence $I_1, \ldots, I_T$, estimate the values of (x_t, R_t) for all $t \in [T]$. We denote the estimated quantities as $(\hat{x}_t, \hat{R}_t)$. The *visual odometry* problem is distinguished from the *simultaneous localization and mapping* (SLAM) problem by the condition that the VO algorithm should behave like a Markovian process: the estimate $(\hat{x}_t, \hat{R}_t)$ should depend only on the observations $I_1, \ldots, I_t$, and the VO algorithm should maintain an internal state with bounded size independent of T. These imply that the VO algorithm should not attempt the *loop closure* process that is characteristic of SLAM algorithms. Furthermore, the VO algorithm is not tasked to build a map of the environment.

As discussed in Section I, the translational portion (estimating x_t) of the VO problem is ill-posed because any estimate $\gamma\hat{x}_1, \ldots, \gamma\hat{x}_T$ scaled by $\gamma \in \mathbb{R},\ \gamma \neq 0$ is an equally valid solution from a geometric perspective. (We are unconcerned with the possibility of a constant translation offset.) However, if one assumes that the environment contains recognizable objects of fixed size, only one γ is correct, because it will yield a trajectory that is consistent with the known sizes of objects in the scene. Our goal in this paper is to capture such prior knowledge about the sizes of objects by training an end-to-end learning-based method on a dataset containing realistic objects and a correct ground-truth camera trajectory. We emphasize that prior knowledge about object sizes is learned implicitly via the task of metric-scale VO; our method does not contain an explicit object recognition step.

Quadrotor background

We briefly recall some background information on quadrotors needed to describe the application of our method to the quadrotor control estimation task. A quadrotor is a flying vehicle actuated by four coplanar propellers positioned at the corners of a square. If one approximates the individual motor dynamics as instantaneous, one can treat the total thrust and body angular accelerations as direct control inputs, since they are invertible linear functions of the four motor thrusts [28]. However, human pilots do not control the angular acceleration directly because the rate at which the controls must be changed is beyond human ability. Instead, the pilot inputs angular *velocity* and the quadrotor's onboard flight controller executes a fast ($500+$ Hz) feedback control loop that drives the motors to achieve the desired angular velocity. It is also common for non-acrobatic pilots to use a higher-level control input such as body attitude or even translational velocity, however it is not possible to express a movement like a flip with such an input, since the control sticks are bounded. In this paper, we wish to handle the full range of quadrotor maneuvers, so we choose to estimate the angular velocity and total thrust.

IV. METHODOLOGY

This section presents our approach to the VO problem described in Section III. We first outline the structure of our approach. We then describe our deep convolutional architecture and training regime in detail. We separately adapt our method to two scenarios:

- KITTI visual odometry driving dataset, where the movement is essentially two-dimensional. KITTI is a widely used benchmark dataset that allows comparing our method to other published methods.
- Estimating control inputs from a first-person quadrotor flight video, which contains a much wider range of movements. As discussed in Section III, estimating quadrotor flight controls is essentially equivalent to VO.

A. Overall structure

Our approach is built on the idea of estimating short-horizon VO with a learned function, and integrating these estimates together into an estimate of the complete camera trajectory. We perform VO using a moving window of $S > 1$ frames, with a one-timestep increment such that consecutive windows are overlapping. For short-horizon VO, we learn a function

$$f_\theta : \mathbb{R}^{S\times H\times W\times C} \mapsto \mathbb{R}^3 \times SO(3), \tag{1}$$

parameterized by a real-valued vector θ. The function f_θ maps a fixed-length "stack" of $S \in \mathbb{N}$ images $I_t, I_{t+1}, \ldots, I_{t+S-1}$ to an estimated translation and rotation between the first and last image. We denote the estimated rotation between time s to time t, where $s < t$, by $\hat{R}_s^t \in SO(3)$. The estimated local translation $\hat{\delta}_s^t \in \mathbb{R}^3$ estimates the translation from x_s to x_t in the local coordinate frame of R_s. For a horizon $S = 1$, integrating these estimates can be trivial:

$$\hat{R}_t = \prod_{i=1}^{t-1} \hat{R}_i^{i+1}, \quad \hat{x}_t = \sum_{i=1}^{t-1} \hat{R}_i \delta_i^{i+1}, \tag{2}$$

where the matrix product is computed using left-multiplication of each term with the base case of an identity matrix. However, we wish to use a longer horizon $S > 1$ to give f_θ more information. When $S > 1$, the integration task is slightly less straightforward because the transition between images $I_t,\ I_{t+1}$ appears in each of the estimates from $\hat{R}_{t-S+2}^{t+1}$ through $\hat{R}_t^{t+S-1}$. Here we state a procedure to account for this.

A rotation matrix in $SO(3)$ can also be represented in axis-angle form. Within the axis-angle parameterization it is

natural to express a fraction or multiple of a rotation, whereas these quantities are not so easily computed in the matrix parameterization. Let $(v, \theta) \in \mathbb{U}^3 \times \mathbb{R}$ denote the rotation about the unit vector v by the angle θ, following the right-hand rule convention. For an arbitrary scalar $\alpha \in \mathbb{R}$, the rotation matrix corresponding to the rotation $(v, \alpha\theta)$ is given by the formula of Rodrigues:

$$R = I + \sin(\alpha\theta)V_\times + (1 - \cos(\alpha\theta))V_\times^2, \tag{3}$$

where $V_\times$ is the cross-product matrix of v,

$$V_\times = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix}. \tag{4}$$

A rotation matrix R is converted into axis-angle form as follows:

$$\theta = \cos^{-1}\left(\frac{\mathrm{Tr}(R) - 1}{2}\right), \quad v = \frac{1}{2\sin\theta}\left(R - R^T\right). \tag{5}$$

We let $R^{1/k}$ denote the rotation matrix, produced using the axis-angle formulae of (3)–(5), such that $(R^{1/k})^k = R$. We are now able to state our method for integrating the overlapping $\hat{R}_t^{t+S-1}$, $\hat{\delta}_t^{t+S-1}$ estimates:

$$\hat{R}_t \approx \prod_{i=1}^{t-S+1} (\hat{R}_i^{i+S-1})^{1/S}, \quad \hat{x}_t \approx \frac{1}{S}\sum_{i=1}^{t-1} \hat{R}_i \delta_i^{i+S-1}. \tag{6}$$

Some remarks:

- This formulation does not properly account for the edge cases at $t = 0$ and $t = T$. We consider these relatively unimportant because we use small stack sizes ($S \approx 5$) such that the lost accuracy is insignificant relative to the overall trajectory.
- This formulation relies on a small angle assumption. For very small θ, it can be derived from (3) that rotations are approximately commutative. With this assumption, spreading the contribution of R_i^{i+1} to all the rotation intervals that contain it is reasonable, even though the rotations from other timesteps are multiplied together "out of order". This implies a minimum sampling rate for the camera.
- We validated (6) by passing the ground truth camera trajectories from the KITTI dataset through this procedure and observing a minimal loss of accuracy.

It can be shown that, in the special case of two-dimensional rotations in a coordinate plane, (3) reduces to simple accumulation of an angle. For the KITTI dataset, the motion is approximately planar since there are no big hills in the driving location, and the car is never driven hard enough to induce significant body roll. For simplicity, we therefore estimate only the yaw angle rotation $\hat{\psi}$ between the first and last frames, and accumulate an absolute yaw estimate $\hat{\Psi}$ as $\hat{\Psi}_{t+1} = \hat{\Psi}_t + \frac{1}{S}\psi_{(t-S+1):t}$. For our quadrotor experiments, we regress on angular velocity directly rather than relative rotations between frames, since angular velocity corresponds directly with the control input used for acrobatic flight. An angular velocity estimate $\hat{\omega} \in \mathbb{R}^3$ can be converted into an axis-angle rotation using the formula

$$\theta = \|\omega\|, \quad v = \frac{\omega}{\|\omega\|}, \tag{7}$$

and proceeding with the integration as above.

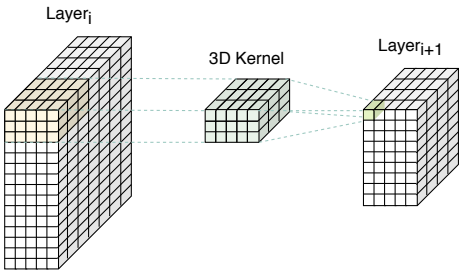


Fig. 3: A 3D convolution. In a 2D convolution, multiple 2D convolutional filters are stacked into a resulting 3D volume, whereas in a 3D convolution, 3D filters are stacked into a 4D volume, as show in Figure 4. The above graphic only represents one 3D filter.

B. General Network Architecture

In all experiments, we parameterize f_θ (1) as a (moderately) deep 3D-convolutional neural network. We begin by describing the generic properties of the network architecture that are shared across all experiments. In Sections IV-C and IV-D, we provide details of the architecture adjustments made for the KITTI and quadrotor control estimation experiments.

The general network architecture is composed of four hidden 3D convolutional layers, each followed by a batch normalization layer. The convolutional layers are followed by two fully connected (FC) layers. Each hidden layer uses the Rectified Linear Unit (ReLU) activation function, with the exception of the last hidden FC layer, which uses LeakyReLU [29], a modification of ReLU that outputs weighted negative values. The output layer uses a linear activation function. All layers employ L2 regularization ("weight decay") at a rate of 0.005 to help avoid overfitting. Our network uses a relatively modest number of layers and parameters compared with some recent deep architectures [30] – this is intended to avoid overfitting, since our training datasets are not particularly large.

A 3D convolution follows the same principle as a 2D convolution, but the kernel and stride are in three dimensions [31]. As shown in Figure 3, a 3D kernel slides over a volume, performing a convolution with pixels that are *spatially* nearby in the horizontal and vertical dimensions, and *temporally* nearby in the time dimension. Since each input pixel is vector-valued with a dimensionality depending on the number of convolution filters in the previous layer (or on C in the base case), the kernel of a 3D convolutional filter is actually a four-dimensional object, as illustrated in Figure 4.

Table I and Table II detail the hyperparameters of the subnetworks. The architecture is adapted differently for the

TABLE I: Neural network hyperparameters used to estimate rotation.

Layer	Filters	Kernel Size	Stride	Activation
3D Conv.	8	(3, 3, 5)	(3, 3, 1)	ReLU
Batch Norm.				
3D Conv.	16	(3, 3, 5)	(3, 3, 1)	ReLU
Batch Norm.				
3D Conv.	32	(3, 3, 5)	(3, 3, 1)	ReLU
Batch Norm.				
3D Conv.	4	(1, 1, 5)	(1, 1, 5)	ReLU
Batch Norm.				
Flatten				
Dense (64)				LeakyReLU
Dense				Linear

TABLE II: Neural network hyperparameters used to estimate translation.

Layer	Filters	Kernel Size	Stride	Activation
3D Conv.	8	(3, 3, 5)	(3, 3, 1)	ReLU
Batch Norm.				
Dropout (0.1)				
3D Conv.	8	(3, 3, 5)	(3, 3, 1)	ReLU
Batch Norm.				
Dropout (0.1)				
3D Conv.	16	(1, 1, 1)	(1, 1, 1)	ReLU
Batch Norm.				
3D Conv.	1	(1, 1, 5)	(1, 1, 5)	ReLU
Batch Norm.				
Flatten				
Dense (8)				LeakyReLU
Dropout (0.25)				
Dense				Linear

KITTI odometry and quadrotor control tasks, with varying input and output sizes resulting in different convolutional filter sizes for each.

The network is trained using the Adam optimizer [32] with a learning rate of 0.001.

C. KITTI VO Network

For the KITTI odometry task, separate rotation and translation CNN's are trained, with each network being broken up into three identical subnetworks, one for each color channel, shown in Figure 2. The input to the network is three image stacks (for each channel) of dimensions: $(W, H, S, C) = (160, 90, 5, 1)$. Each image is resized down from $(1241, 376)$ to $(300, 90)$ and cropped at the sides to $(160, 90)$. We also experimented with uncropped images, but the resulting increase in network parameters led to overfitting. Other alternative approaches explored included using a single subnetwork encompassing all color channels with input size $(160, 90, 5, 3)$, as well as using a single network to estimate both rotation and translation, but both of these approaches performed worse upon evaluation. The outputs of the color channel subnetworks are averaged to arrive at the final estimation. We are not aware of any theoretical reason why using separate networks for each color channel performs better than a single network, but we continued with this architecture because it performed best in our experiments.

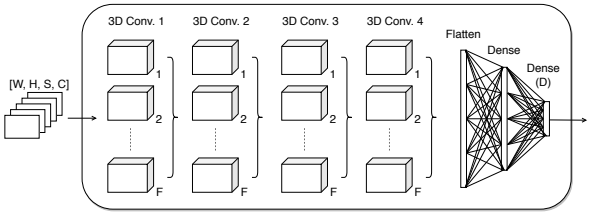


Fig. 4: Generic subnetwork structure. As with Figure 2, W, H, S, C, and F stand for width, height, stack size, channels, and number of filters, respectively, and D represents the network output dimension.

As car movement mostly exhibits only yaw and forward offsets (as discussed in Section IV-A) in the span of 5 frames (lasting roughly 0.5 seconds with KITTI images at 10Hz), the rotation and translation networks only output a single scalar each, yaw and forward offsets, respectively. The roll, pitch, lateral and elevation components are set to zero when aggregating the local offsets into the global trajectory.

In evaluation, we observed that estimation accuracy of very tight turns (ψ large) was not as good compared to straight sections. To account for this, we employ a weighted mean squared error (MSE) training loss for the rotation network, which penalizes angular offset errors above a certain threshold more than errors below it:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} w(\psi_i) \cdot (\hat{\psi}_i - \psi_i)^2, \tag{8}$$

where N is the minibatch size used. The function $w : \mathbb{R} \mapsto \mathbb{R}_{>0}$ is a nondecreasing weighting function. In our experiments, we used a simple piecewise constant weight

$$w(\psi_i) = \begin{cases} 2 & \psi_i > 0.1 \\ 1 & \text{otherwise.} \end{cases} \tag{9}$$

D. Quadrotor Control Network

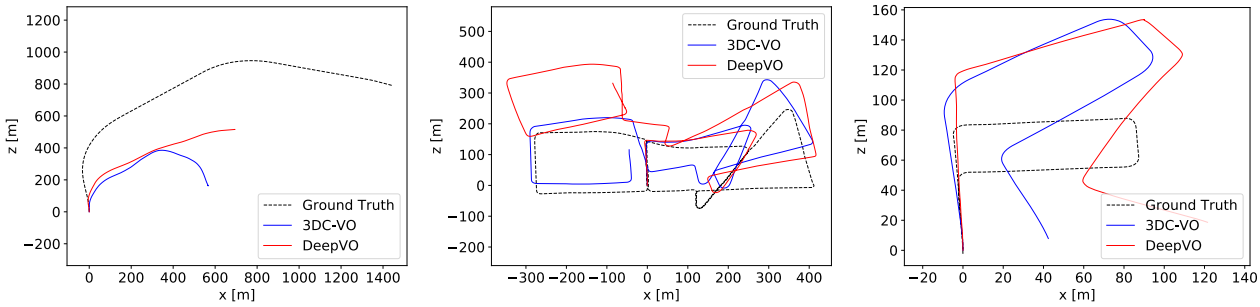
For the quadrotor control prediction task, two networks are created for predicting thrust and angular velocity, both using the same hyperparameters as the KITTI rotation network detailed in Section IV-C.The input images are of size $(256, 144)$, and the thrust network has 1 output and the angular network outputs rudder, elevator, and aileron. The joystick measurements from the first frame in the stack are used as the ground truth value.

Unweighted MSE was used as the loss function for optimizing this network, as our end goal was to predict local control inputs rather than aggregate local offsets into a global pose, so long term drift was not a concern.

V. EXPERIMENTAL RESULTS

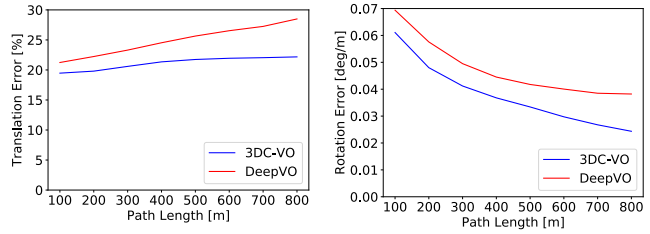
A. KITTI odometry task

1) Training: The KITTI dataset is provided as a set of 21 sequences of frames, with the first ten sequences containing ground truth poses and the rest without. We used sequences 00, 02, 08, and 09 for training, 03-07 and 10 for validation, and 11-21 as a test set.

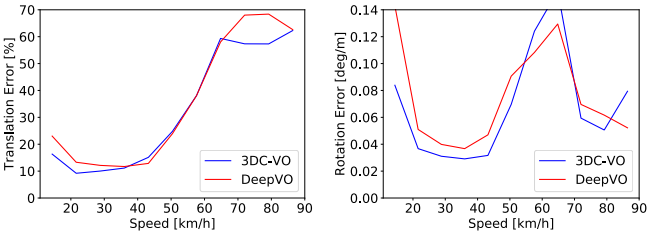


(a) Sequence 12 (b) Sequence 13 (c) Sequence 14

Fig. 5: Plots 12-14 of KITTI ground truth (in black), 3DC-VO predictions (in blue), and DeepVO predictions (in red). Although our method observes only a monocular image stream, making the odometry task susceptible to scale ambiguity, the scale of our estimated trajectories approximately match the ground truth.



(a) Trans. against path length (b) Rot. against path length



(c) Trans. against speed (d) Rot. against speed

Fig. 6: Translation and rotation errors for 3DC-VO and DeepVO against path length and speed for the KITTI evaluation sequences 10-21.

2) Testing: After evaluating our trained model on test sequences 11-21 and using the pose aggregation procedure detailed in Section IV-A to generate global homogeneous transformation matrices, we submit our results to the KITTI benchmark server to be judged against other submissions. The closest point of comparison for our method is DeepVO [5], a learning-based method that also trains exclusively on monocular data, but uses a recurrent neural network architecture to estimate global pose $(\hat{x}_t, \hat{R}_t)$ directly, instead of 3D convolutions to estimate local translation and rotation over a fixed window. Compared to DeepVO, our method achieves improved results of 21.0% translational error versus 24.55%, and $0.0394°$/m rotational error versus $0.0489°$/m. Figure 6 plots the average translational and rotational errors for each path length of size $(100, 200, \ldots, 800)$ meters, as well as errors against vehicle speed in 10 km/h increments (the 21.0% and $0.0394°$/m values on the KITTI leaderboard are the mean of these values, respectively). Figure 5 shows trajectories for sequences 12-14, as predicted by DeepVO and 3DC-VO, along with ground truth. Sequence 12 shows DeepVO predicting a slightly more accurate trajectory, with 3DC-VO demonstrating a clear advantage on sequences 13 and 14. We also note that our method is estimating metric scale accurately in these trajectories given only a monocular image stream. It is interesting to note that both methods' predictions are of similar scales, even when they are noticeably different from the ground truth, as seen in 5 a) and c). This suggests that even though the internal CNN architectures are significantly different in design, they could be learning similar features internally.

We emphasize that neither method is competitive on the KITTI leaderboard with methods that use additional sensor data such as IMU or LIDAR. In comparison, the key strength of our method (and DeepVO) is our ability to estimate metric scale using only monocular camera data. Our training and evaluation code for the KITTI odometry task is available at `https://www.github.com/alexanderkoumis/3dc_vo`.

B. Quadrotor control estimation task

1) Dataset generation: To the best of our knowledge, there is no public dataset containing image observations from a quadrotor alongside per-frame control inputs. We created such a dataset using Microsoft's AirSim [33], a quadrotor simulator based on the Unreal game engine. AirSim includes environment models with realistic graphics, such as a suburban neighborhood with trees, houses, power lines, and cars. We captured image sequences, along with ground truth pose and joystick positions, in 9 different environments, including a neighborhood, large city, forest, and plains scene. The pilot's control inputs were captured from a radio control transmitter of the same type used for real quadrotor flight, by capturing the controller's analog outputs through the PC's analog audio input. The resolution of the images from AirSim was $(254, 144)$. We used AirSim's software-in-the-loop control mode with the popular open-source flight control firmware PX4 [34], since PX4 implements the same type of

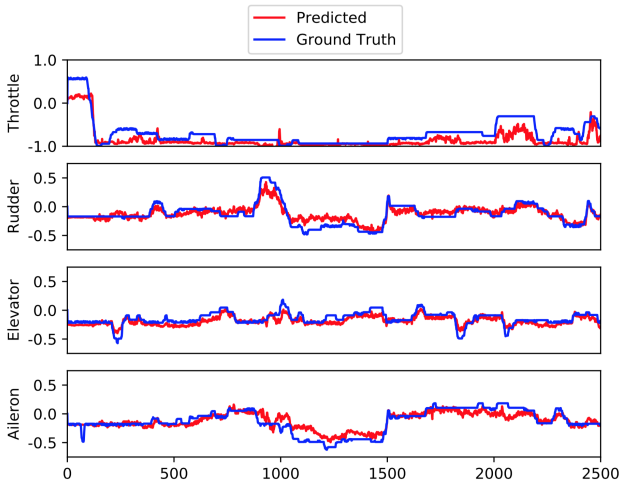


Fig. 7: Predicted (red) and ground truth (blue) joystick positions from a flight simulator sequence with a human pilot. Joystick axes have an affine relationship with thrust and angular velocity inputs. Offset from zero-mean is due to joystick calibration offset.

attitude rate controller commonly used onboard quadrotors for acrobatic flight control.

2) Training: One sequence was recorded per environment, 9 in total. All but one of these were used as the training set, with the city environment being left out for validation. We created two networks based on the rotation network described in Table I, one for predicting throttle and one for predicting the rotational control inputs. The networks predict the joystick positions at the beginning of an image stack, which is similar to the KITTI task of predicting velocity by computing the offset between frame t and $t + 4$, as joystick commands directly correspond to the desired thrust and angular velocity of the vehicle.

The throttle network was trained using images separated by four time steps, i.e. I_t, I_{t+5}, I_{t+10}, I_{t+15}, I_{t+20}, while the rotation network trained on consecutive images. The reason for this distinction is that the AirSim vehicle dynamics are that of a relatively heavy quadrotor, responding quickly to rotational input while reacting much slower to throttle input (resulting in little no change in image features in consecutive frames).

3) Testing: The effectiveness of the model was evaluated by examining plots of predicted versus ground truth joystick positions on the dataset sequences, as well as displaying real-time joystick predictions over the simulator as a pilot controlled a quadrotor. Figure 7 shows the joystick estimates compared to the ground truth values at each frame for a particular sequence in the dataset, with a 0.0895 mean absolute error and 0.3245 standard deviation. Figure 1 is a screen capture from a video (included with this work's supplementary material) showing a pilot controlling a quadrotor with overlaid real-time joystick predictions. This is a new dataset with no baselines for comparison, but the plots and real-time predictions show good performance making metric scale-accurate predictions on the joystick input.

VI. CONCLUSIONS

In this work we have shown that a 3D convolutional neural network is capable of performing end-to-end monocular VO with accurate scale. We avoid the recurrent architectures commonly used for sequence processing, allowing faster training and an improvement in accuracy on the KITTI driving dataset compared to a state-of-the-art RNN-based method. We also demonstrated an application of our architecture to quadrotor control estimation, a domain with significantly different camera motions and scene appearance.

While end-to-end learning-based architectures have room for improvement in terms of rotational accuracy compared to traditional geometric methods, they provide a way to estimate metric scale without any extra sensors or domain-specific assumptions. End-to-end learning can implicitly capture prior knowledge about the size and appearance of real-world objects, simply as a side effect of learning to estimate the rotation and translation between frames in physical units. In future work, we plan to leverage this property to build a large dataset of estimated trajectories and pilot controls from publicly available acrobatic flight videos, and use this dataset to explore imitation learning approaches for creating agents that fly like human acrobatic pilots. We also believe that improved accuracy is possible, potentially by experimenting with the following: more geometrically accurate integration of moving window velocities into pose, alternative loss function weighting for sharp turns, larger convolutional network architectures, and data augmentation to allow training large networks without overfitting.

REFERENCES

[1] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[2] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint kalman filter for vision-aided inertial navigation," in *ICRA*. IEEE, 2007, pp. 3565–3572.

[3] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Trans. Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[4] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *ICRA*. IEEE, 2011, pp. 2520–2525.

[5] S. Wang, R. Clark, H. Wen, and N. Trigoni, "Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 2043–2050.

[6] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE Robot. Automat. Mag.*, vol. 18, no. 4, pp. 80–92, 2011.

[7] M. W. Achtelik, M. Achtelik, S. Weiss, and R. Siegwart, "Onboard imu and monocular vision based control for mavs in unknown in-and outdoor environments," in *IEEE International Conference on Robotics and Automation (ICRA 2011)*. Eidgenössische Technische Hochschule Zürich, Autonomous Systems Lab, 2011.

[8] D. P. Frost, O. Khler, and D. W. Murray, "Object-aware bundle adjustment for correcting monocular scale drift," in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2012.

[9] E. Sucar and J.-B. Hayet, "Bayesian scale estimation for monocular slam based on generic object detection for correcting scale drift," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–7.

[10] A. Hanel, A. Mitschke, R. Boerner, D. Van Opdenbosch, L. Hoegner, D. Brodie, and U. Stilla, "Metric scale calculation for visual mapping algorithms." *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, vol. 42, no. 2, 2018.

[11] H. Mirabdollah and B. Mertsching, "Fast techniques for monocular visual odometry," in *Proceeding of 37th German Conference on Pattern Recognition (GCPR)*, 2015, pp. 297–307.

[12] S. Song and M. Chandraker, "Robust scale estimation in real-time monocular sfm for autonomous driving," in *CVPR*, Columbus, Ohio, USA, 2014.

[13] M. H. Mirabdollah and B. Mertsching, "On the second order statistics of essential matrix elements," in *Proceeding of 36th German Conference on Pattern Recognition*, 2014, pp. 547–557.

[14] N. Fanani, A. Stuerck, M. Ochs, H. Bradler, and R. Mester, "Predictive monocular odometry (pmo): What is possible without ransac and multiframe bundle adjustment?" *Image and Vision Computing*, 2017.

[15] V. Guizilini and F. Ramos, "Visual odometry learning for unmanned aerial vehicles," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 6213–6220.

[16] ——, "Semi-parametric models for visual odometry," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 3482–3489.

[17] T. A. Ciarfuglia, G. Costante, P. Valigi, and E. Ricci, "Evaluation of non-geometric methods for visual odometry," *Robotics and Autonomous Systems*, vol. 62, no. 12, pp. 1717–1730, 2014.

[18] K. R. Konda and R. Memisevic, "Learning visual odometry with a convolutional network." in *VISAPP (1)*, 2015, pp. 486–490.

[19] C. Zhao, L. Sun, P. Purkait, T. Duckett, and R. Stolkin, "Learning monocular visual odometry with dense 3d mapping from dense 3d flow," *arXiv preprint arXiv:1803.02286*, 2018.

[20] V. M. Babu, K. Das, A. Majumdar, and S. Kumar, "Undemon: Unsupervised deep network for depth and ego-motion estimation," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1082–1088.

[21] 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.

[22] G. Costante, M. Mancini, P. Valigi, and T. A. Ciarfuglia, "Exploring representation learning with cnns for frame-to-frame ego-motion estimation." *IEEE robotics and automation letters*, vol. 1, no. 1, pp. 18–25, 2016.

[23] P. Muller and A. Savakis, "Flowdometry: An optical flow and deep learning based approach to visual odometry," in *Applications of Computer Vision (WACV), 2017 IEEE Winter Conference on*. IEEE, 2017, pp. 624–631.

[24] J. Jiao, J. Jiao, Y. Mo, W. Liu, and Z. Deng, "Magicvo: End-to-end monocular visual odometry through deep bi-directional recurrent convolutional neural network," *arXiv preprint arXiv:1811.10964*, 2018.

[25] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[26] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, 2018.

[27] G. Li, M. Mueller, V. Casser, N. Smith, D. L. Michels, and B. Ghanem, "Teaching uavs to race with observational imitation learning," *arXiv preprint arXiv:1803.01129*, 2018.

[28] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor UAV on SE(3)," in *CDC*. IEEE, 2010, pp. 5420–5425.

[29] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, no. 1, 2013, p. 3.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[31] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 221–231, 2013.

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[33] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017. [Online]. Available: https://arxiv.org/abs/1705.05065

[34] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "PIXHAWK: A system for autonomous flight using onboard computer vision," in *ICRA*. IEEE, 2011, pp. 2992–2997.