Learning Pose Estimation for UAV Autonomous Navigation and Landing Using Visual-Inertial Sensor Data

Francesca Baldini[1], Animashree Anandkumar[1], and Richard M. Murray[1]

***Abstract*— In this work, we propose a new learning approach for autonomous navigation and landing of an Unmanned-Aerial-Vehicle (UAV). We develop a multimodal fusion of deep neural architectures for visual-inertial odometry. We train the model in an end-to-end fashion to estimate the current vehicle pose from streams of visual and inertial measurements. We first evaluate the accuracy of our estimation by comparing the prediction of the model to traditional algorithms on the publicly available EuRoC MAV dataset. The results illustrate a 25% improvement in estimation accuracy over the baseline. Finally, we integrate the architecture in the closed-loop flight control system of Airsim - a plugin simulator for Unreal Engine - and we provide simulation results for autonomous navigation and landing.**

I. INTRODUCTION

Unmanned-Aerial-Vehicles (UAVs) can provide significant support for several applications, e.g., rescue operations, environmental monitoring, package delivery, and surveillance, to name a few. To guarantee a high safety level in autonomous UAV operations it is crucial to have an accurate estimate of the vehicle's state at any given time.

Standard techniques deployed for pose estimation are Visual-Inertial Odometry (VIO) and Simultaneous Localization and Mapping (SLAM) [1–5]. These methods fuse information from both visual and inertial sensors to localize the vehicle. Visual-only or inertial-only odometry estimators suffer from drifts and scale ambiguity. In visual odometry, loop closure can reduce the drift problem, but we still need to integrate external information to solve for the scale ambiguity. Therefore, by fusing inertial and visual data, not only do we resolve the scale ambiguity, but we also increase the accuracy of the odometry itself.

The pipeline for VIO and SLAM typically consists of camera calibration, followed by feature detection and tracking, outlier rejection, motion and scale estimation, optimization back-end, and local optimization (Bundle Adjustment). However, these techniques lack robustness when deployed in challenging conditions, such as low-texture or low-light environments, or in the presence of noises, blurs, camera occlusions, dynamic objects in the scene, and camera calibration errors [1, 6, 7]. Additionally, different scenarios may require different types of features for tracking and matching, as well as more adaptive algorithms. Nevertheless, VIO and SLAM methods only track pre-designed hand-engineered descriptors in all the contexts.

The advent of deep learning technology has made neural networks more appealing for dealing with visual-inertial odometry problems [8]. By learning features from data rather than using hand-designed descriptors, deep neural networks can adapt to different contexts [9, 10].

While neural networks can extract visual information about vehicle position and orientation, data gathered from inertial measurement units (IMUs) provide complementary information about the pose. By combining neural networks that learn visual features with data gathered from inertial measurement units, we propose a multimodal fusion learning approach to increase odometer estimation accuracy

A. Contributions

We propose a new end-to-end approach for online pose estimation that leverages multimodal fusion learning. This consists of a convolutional neural network for image regression and two long short-term memories (LSTMs) of different sizes to account for both sequential and temporal relationships of the input data streams. A small LSTM architecture integrates arrays of acceleration and angular velocity from the inertial measurements unit sensor. A bigger core LSTM processes visual and inertial feature representations along with the previous vehicle's pose and returns position and orientation estimates at any given time.

We assess the performance of our model and compare it to a baseline algorithms for visual inertia odometry on the publicly available EuRoC MAV dataset. The results show that our method significantly outperforms the state-of-the-art for odometry estimation, improving the accuracy up to 25% over the baseline.

We then integrate our data-driven odometry module in a closed-loop flight control system, providing a new method for real-time autonomous navigation and landing. To this end, we generate a simulated *Downtown* environment using Airsim, a flight simulator available as a plugin for Unreal Engine [11]. We collect images and inertial measurements flying in the simulated environment and we train the model on the new synthetic dataset. The network outputs are now the input to the

F.Baldini is supported in part by Darpa PAI grant HR0011-18-9-0035. A. Anandkumar is supported in part by Darpa PAI grant HR0011-18-9-0035, Bren Endowed Chair, Microsoft Faculty Fellowship, Google Faculty Award, Adobe Grant,

[1]California Institute of Technology, Pasadena CA 91125, USA

Github: https://github.com/Baldins/LearningUAVposeEstimation

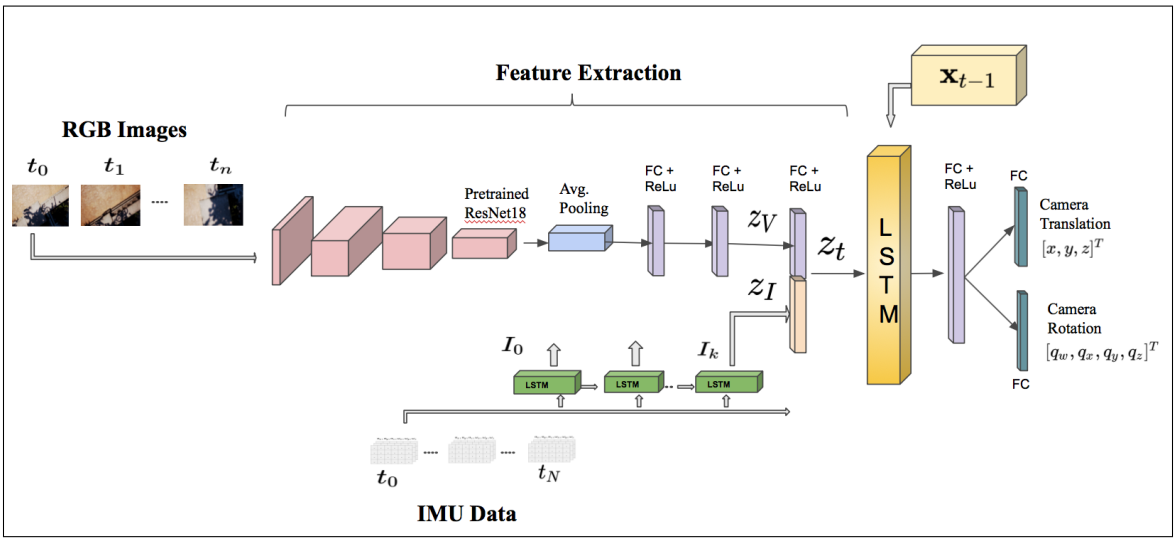

Fig. 1. Architecture for the data-driven VIO module. It consists of visual and inertial encoders, feature concatenations, temporal modeling, and pose regression. The CNN module determines the most discriminative visual features z_V. A small LSTM module transforms windows batch of inertial measurements collected between two consecutive frames into a single inertial feature vector z_I. The visual and inertial feature vectors are then concatenated in a single representation z_t. The core LSTM uses the feature vector z_t along with the previous estimate x_{t-1} and makes a prediction about the robot pose (translation and rotation).

flight control system that generates velocity commands for the UAV system. We show through real-time simulations that our closed-loop data-driven control system can successfully navigate and land the UAV on the designed target with less than 10 cm of error.

Algorithm 1: Data-driven pose estimation

Input: Input Image I_t,
IMU: $\omega_{t-T:t}, a_{t-T:t}$,
Previous Pose: x_{t-1}
Output: Estimated pose: $x_t = [x, y, z, q_w, q_x, q_y, q_z]^T$
while *Flying:* **do**
  for $t = 1, \ldots, T$ **do**
    Collect data (Images and IMU);
    if *Camera Corrupted* **then**
      Use IMU only
    else
      Use Image and IMU
    end
    1. y_V = CNN(I_t);
    2. y_I = LSTM($\omega_{t-T:t}, a_{t-T:t}$);
    3. z_t = concat(y_V, y_I);
    4. $\hat{x}_t$ = CoreLSTM(z_t, x_{t-1})
  end
end

B. Previous Work

Traditional localization approaches for autonomous UAV navigation rely on computer vision algorithms supplemented by sensors, including Global Positioning Systems (GPS) and Inertial Measurement Units (IMUs) [2, 12–14]. Visual Servoing methods use the tracked features as inputs to a control law that directs the robot into a desired pose [6, 15, 16]. However, environmental noise and the presence of moving objects can negatively affect the tracking process and reduce the estimation accuracy. Additionally, VIO systems demand heavy computation due to image processing and sensor fusion. The advent of deep learning techniques has created new benchmarks in almost all areas of computer vision. In the last few years, CNN architectures for pose estimation have captured the interests of the robotics community [9, 17–19]. PoseNet has been the first approach to use CNNs to address the metric localization problem [10]. Additionally, other architectures have been deployed to estimate the incremental motion of the camera using only sequential camera images or a combination of visual and inertial data [9, 17, 20–22].

II. PRELIMINARIES

A. Localization Problem

Given the actual state $x_t = [x, y, z, q_w, q_x, q_y, q_z]^T \in \mathbb{R}^7$, we train the neural network to estimate the vehicle pose $\hat{x}_t = [\hat{x}, \hat{y}, \hat{z}, \hat{q}_w, \hat{q}_x, \hat{q}_y, \hat{q}_z]^T \in \mathbb{R}^7$ from a continuous stream of images and inertial measurements. The inputs for our model are observation tuples $y_t = \{y_I, y_V\}$ of RGB images (y_V) and IMU data (y_I), where $y_I = [\tau_t, a_x, a_y, a_z, \omega_x, \omega_y, \omega_z]^T \in \mathbb{R}^{N \times 7}$, τ_t is the timestamp of the inertial measurement, y_t is the linear acceleration, y_t is the angular velocity, and N is the number of inertial observation between two consecutive camera frames t and $t+1$. The online localization task aims to estimate the pose of the vehicle x_t at any given time given the current observations y_t and previous pose state x_{t-1}. In the learning framework, we aim to model the mapping f between raw data and the current pose as follows:

$x_t = f(x_{t-1}, y_t)$, $f : \mathbb{R}^6, \mathbb{R}^{p \times q} \rightarrow \mathbb{R}^7$, where p, q are the image dimensions.

B. Control Problem

We make the assumption that the target position $x_d^p = [x_d, y_d, z_d]^T$ is know. After the controller receives the reference position of the target x_d^p, the desired velocities $x_d^v = [x_d, y_d, z_d]^T$ are computed based on the rate of change of position set points, i.e., $x_d^v = \dot{x}_d^p$. We use the velocity reference x_d^v along with the position reference x_d^p to compute the final throttle and attitude angle commands $u_{com} = [F_d, \phi_d, \theta_d, \psi_d]^T$ that are then fed back into the low-level controller. Given the target pose coordinate, we simulate a control law for u_{com} depending only on x_d^p and $\dot{x}_d^p$ such that $x_t \rightarrow x_d^p$ and $(v_t, \omega_t) \rightarrow 0$. Fig. 2 illustrates the control system architecture.

III. Architecture

Fig. 1 depicts the architecture of our model. The inputs to the network are synchronized data from visual and inertial sensors. We estimate the UAV's absolute pose by minimizing the geometry consistency loss function described in III-A.

We train the network in an end-to-end fashion to regress the vehicle pose from sequences of images and windows of inertial measurements collected between two consecutive image frames.

a) Image feature extractor: To encode image features, we use ResNet18, pre-trained on the ImageNet dataset, truncated before the last average pooling layer. Each of the convolutions is followed by batch normalization and the Rectified Linear Unit (ReLU). We replace the average pooling with global average pooling and subsequently add two inner-product layers. The output is a visual feature vector representation z_V.

b) Inertial feature extraction: IMU measurements are generally available at a rate of an order of magnitude higher (e.g., $100-200Hz$) than visual data (e.g., $10-20Hz$). A Long Short-Term Memory (LSTM) processes batches of IMU data (acceleration and angular velocity) between two consecutive image frames and outputs an inertial feature vector y_I. LSTM exploits the temporal dependencies of the input data by maintaining hidden states throughout the window.

c) Intermediate fully-connected layer: The inertial feature vector y_I is concatenated with the visual feature representation y_V into a single feature z_t representing the motion dynamics of the robot: $x_t = \texttt{concat}(y_V, y_V)$. This vector is then carried over to the core LSTM for sequential modeling.

d) Core LSTM: The core LSTM takes as input the motion feature z_t along with its previous hidden states h_{t-1} and models the dynamics and the connections between sequences of features, where $z_t = f(z_t, h_{t-1})$. The use of the LSTM module allows for the rapid deployment of visual-inertial pose tracking. These models can maintain the memory of the hidden states over time and have feedback loops among them. In this way, they enable their hidden state to be related to the previous one, allowing them to learn the connection between the last input and pose state in the sequence. Finally, the output of the LSTM is carried into a fully-connected layer, which serves as an odometry estimation. The first inner-product layer is of dimension 1024, and the following two are of dimensions 3 and 4 for regressing the translation x and rotation q as quaternions. Overall, the fully connected layer maps the features vector representation z_t into a pose vector as follows: $x_t = LSTM(z_t, h_{t-1})$.

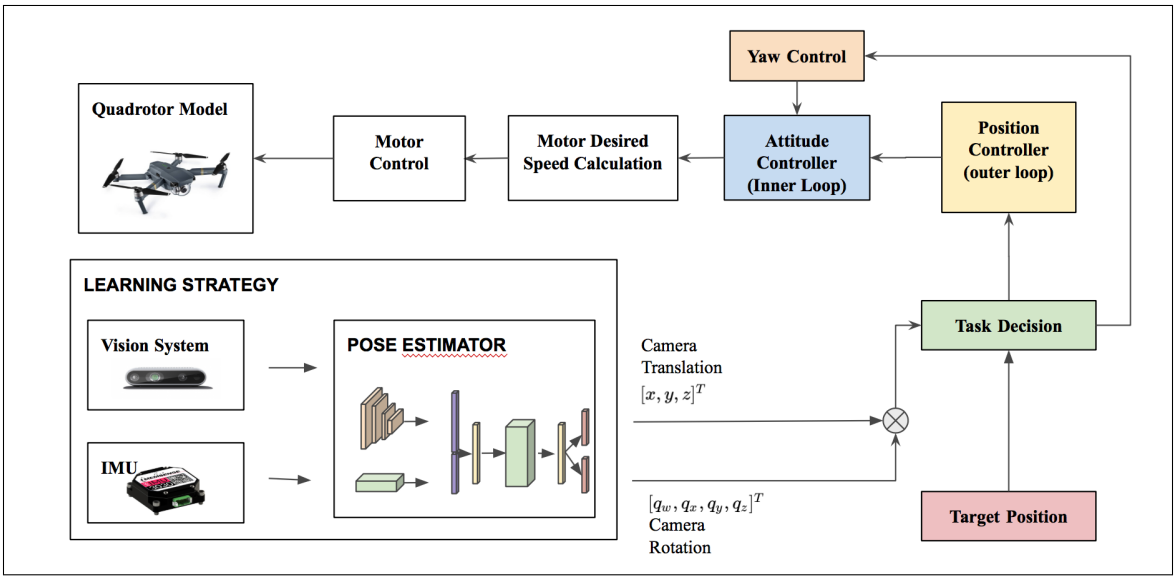

Fig. 2. High-level illustration of the data-driven GNC model. The neural network makes a prediction about the robot position and orientation. The low level controller uses the predicted pose along with the desired one to generate velocity commands to drive the robot to its destination.

A. Learning and Inference

To regress the pose of the vehicle, we compute the Euclidean loss between the estimated pose and the ground truth. We adopt Adam optimizer to minimize this loss function, starting with an initial rate of 10^{-4} [23].

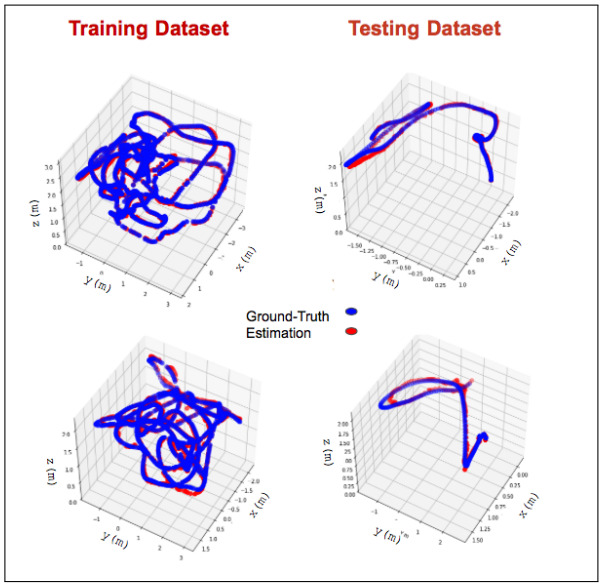

Fig. 3. Sampled 3D trajectories of results on the training and testing sequences

a) Loss Function: We predict the position and orientation of the robot following the work of Kendall et al., with the following modification [24]. In our loss function, we introduce an additional constraint that penalizes both the L_1 and L_2 Euclidean norm. Let $x_t = [x, y, z]^T \in \mathbb{R}^3, q_t = [q_w, q_x, q_y, q_z]^T \in \mathbb{R}^4$ be the ground-truth translation and rotation vector, respectively, and $\hat{x}_t, \hat{q}_t$ their estimates.

Our loss function is as follows:

$$\mathscr{L}_\beta(I) = \mathscr{L}_x(I) + \beta \mathscr{L}_q(I) \tag{1}$$

where

$$\mathscr{L}_x(I) = \|\hat{x}_t - x_t\|_{L_2} + \gamma \|\hat{x}_t - x_t\|_{L_1}$$

and

$$\mathscr{L}_q(I) = \left\| \hat{q}_t - \frac{q_t}{\|q_t\|} \right\|_{L_2} + \gamma \left\| \hat{q}_t - \frac{q_t}{\|q_t\|} \right\|_{L_1}$$

represents the translation and the rotation loss. β is a scale factor that balances the weights of position and orientation, which are expressed in different units, and γ is a coefficient introduced to balance the two Euclidean norms. However, β requires significant tuning to get consistent results, as shown in [24]. To avoid this issue, we replace β by introducing learnable parameters. The final loss function is as follows:

$$\mathscr{L}_\sigma(I) = \mathscr{L}_x(I) \exp(-\hat{s}_x) + \hat{s}_x + \mathscr{L}_q(I) \exp\left(-\hat{s}_q\right) + \hat{s}_q \tag{2}$$

where $\hat{s} := \log \hat{\sigma}^2$ is the learnable variable and each variable acts as a weight for the respective component in the loss function.

TABLE I
MEAN ERROR ON EUROC MAV DATASET

Algorithm	V1-01	V1-02	V1-03	V2-01	V2-02	V2-03
This Paper	**0.03**	**0.05**	**0.10**	**0.05**	**0.05**	**0.08**
svomsf [26]	0.39	0.63	x	0.17	0.37	x
msckf [2]	0.29	0.20	0.67	0.11	0.16	1.13
okvis [27]	0.09	0.18	0.2	0.12	0.14	0.14
rovio [1]	0.1	0.10	0,14	0.12	0.14	0.14
vinsmono [7]	0.07	0.10	0.13	0.08	0.08	0.21
vinsmonolc [28]	0.04	0.05	0.12	0.06	0.05	0.09
svogtsam [29]	0.12	0.16	x	0.08	xx	x

IV. BENCHMARKING

All the experiments are carried out on an Intel Xeon CPU E5-2650 @ 2 GHz processor 192 GB RAM, and NVIDIA TITAN RTX GPU. We first evaluate the performance of the estimator by comparing our results to the state-of-the-art for VIO on the EuRoC MAV dataset provided by [25]. This dataset consists of eleven visual-inertial challenge sequences recorded onboard a micro-aerial-vehicle (MAV) flying in an indoor environment. It provides stereo monochrome images at 20 Hz, temporally synchronized IMU data at 200 Hz, and ground-truth positioning measurements from the Vicon motion capture system.

For quantitative pose evaluation, we compute the average root mean square (RMS) translation and rotation error and compare our results to traditional methods reported in [1, 2, 7, 25–29] . The translation errors are computed across the entire length of the trajectory and ground truth. The RMSE for regression represents the sample standard deviation of the differences between predicted values and real values

$$RMSE = \sqrt{\frac{\sum_{i=1}^{I} (\hat{y}_i - y_i)^2}{n}} \tag{3}$$

where y is the real and $\hat{y}$ is the predicted one.

Table I shows a comparative analysis of average translation RMSE obtained with our method and other traditional odometry estimators on the EuRoC MAV dataset. This dataset contains sequences of videos with different characteristics, some of which are more challenging than others. In all the experiments, our method outperforms the baseline.

V. INTEGRATION WITH THE FLIGHT CONTROL SYSTEM

A. The Dataset

To generate the testing scenario, we use Airsim [11], an open-source simulator that aims to close the gap between simulation and reality. As a plugin, we can use Airsim in any environment developed for Unreal Engine (UE). We collect training data in the virtual *Downtown* environment retrieved from the Unreal Engine marketplace. Fig. 5 shows an example scenario. We split each

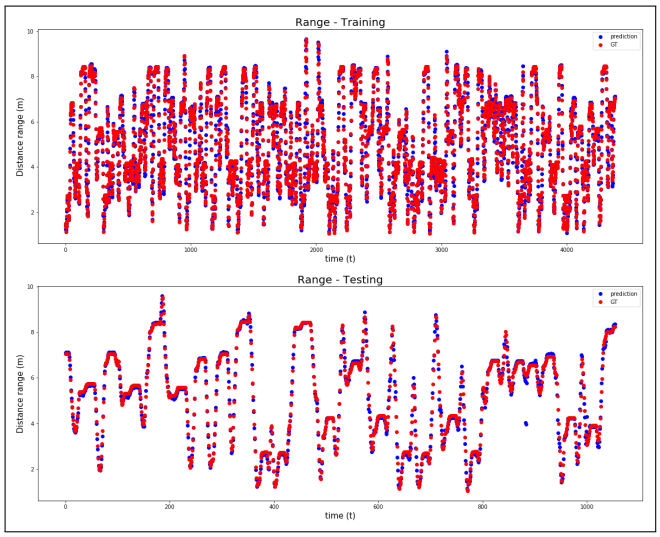

Fig. 4. Range measurements for training and testing dataset

dataset into two sub-datasets, one for training and the other for testing. Each flight within the dataset contains timestamped values for the ground-truth 6 DOF pose of the UAV at $100Hz$, IMU measurements at $100Hz$, and camera streams (downward-facing) at $10Hz$. Ground-truth and sensor data are then pre-processed and synchronized.

B. Experiment Implementation

In the simulation, the UAV collects training data flying at a constant velocity over a grass field, landing on top of randomly chosen pillars. The UAV is equipped with a proportional-integral-derivative (PID) flight controller (FC) that maintains fixed altitude and takes as input the current pose of the vehicle. We label the recorded frames of the simulator with the corresponding pose measurements from the dataset.

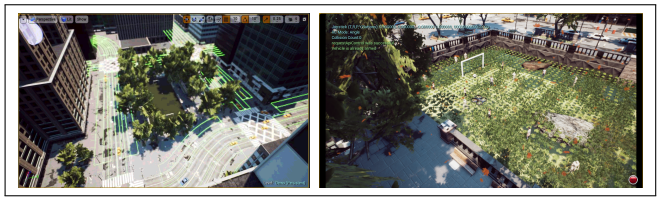

Fig. 5. Images sampled from the *Downtown* simulated environments used in the paper.

The total simulation time is $5h$, and we collect images with a step-size of 0.1 seconds. We then split the dataset into two sub-datasets with a ratio of 0.8 and 0.2 as a training and testing set, respectively. We finally down-sample the resolution of images to $512 \times 288 \times 3$ (RGB) to reduce the computational cost. Each image is normalized to the range $[0,1]$. Both datasets present corrupted data images. To deal with this issue, we trained the network to use only IMU data as a new corrupted imaged shows up, as shown in Algorithm 1.

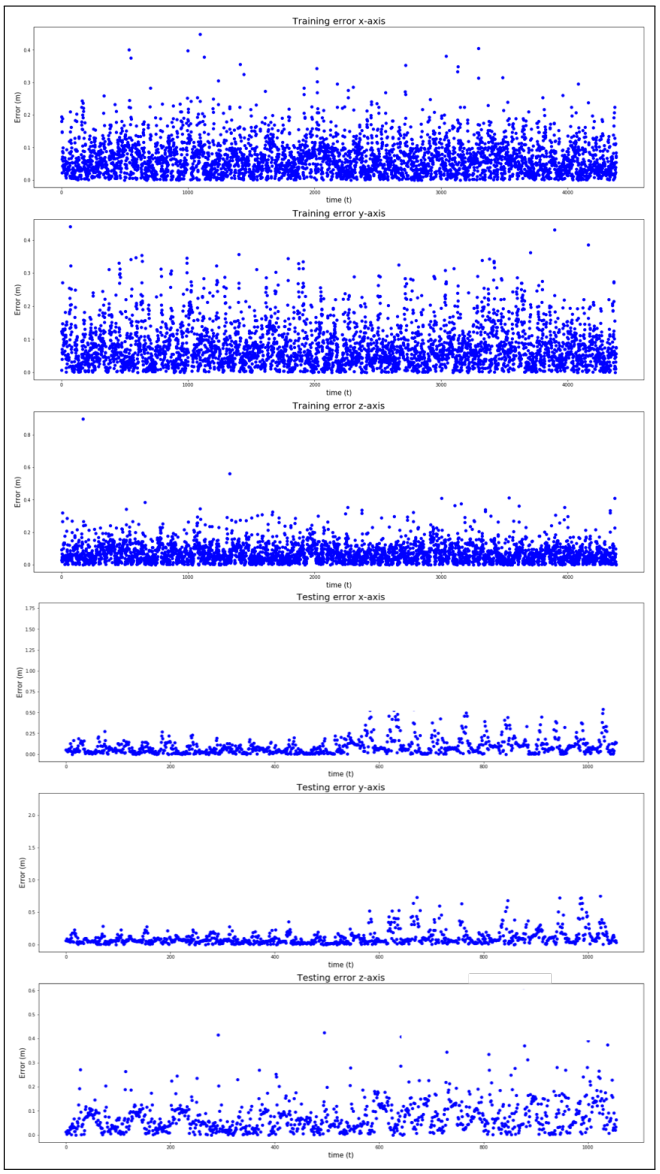

Fig. 6. Localization RMSE for training and test dataset

a) Lower bounds from steady-state Kalman filter: Kalman filters (KF) are the optimal state estimator for systems with a linear process and measurement models. To quantify the accuracy of our estimation, we use a worst-case estimation error derived from the steady-state covariance of the KF. This error is used to define a lower bound on the neural network performance.

We consider the following discrete-time linear Gaussian state-space model consider the following discrete-time linear Gaussian state-space model:

$$x_{t+1} = Ax_t + w_t, \quad t \in \mathbb{N} \tag{4}$$

$$y_t = Hx_t + v_t, \tag{5}$$

where $\boldsymbol{x_t}$ is the state of the system and $\boldsymbol{y}_t$ is the measurements vector. A and H are matrices of appropriate dimensions. The process noise and the measurement noise are distributed according to $w_t \approx \mathcal{N}(0, \boldsymbol{Q})$ and

$v_t \approx \mathcal{N}(0,R)$ respectively, with Q,R covariance matrices. This model can be used in a target-tracking context to describe a linear target motion and measurement model. We assume that the sensor can measure only the position of the target:

$$d_t = \sqrt{(x_t - x_t)^2 + (y_t - y_t)^2 + (z_t - z_t)^2} + \tilde{d}_t \qquad (6)$$

where $\tilde{d}_t$ is the error in the measurement (+/- 1 pixel).

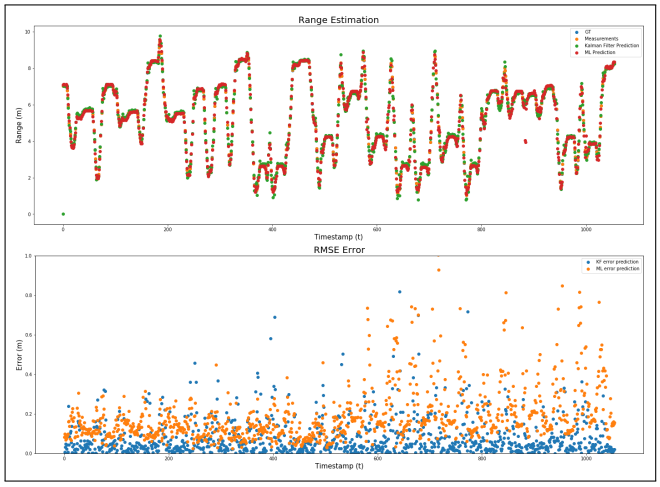

Fig. 7. Kalman Filter (blue) and ML (orange) tracking prediction and RMSE.

The one step ahead prediction for the performance of a Kalman Filter is given by $P_{t|t} = (I - K_t H)P_{t|t-1}$. We can now formulate Kalman-like recursions for an overall system as follows:

$$\begin{aligned} P_{t+1|t} &= AP_{t|t}A^T + Q \\ K_t &= P_{t|t-1}H^T \left(HP_{t|t-1}H^T + R\right)^{-1} \\ P_{t|t} &= (I - K_t H)P_{t|t-1} \end{aligned} \qquad (7)$$

By performing the proper substitutions, we obtain

$$\begin{aligned} P_{t+1|t} = AP_{t|t-1}A^T - AP_{t|t-1}H^T \left(HP_{t|t-1}H^T + R\right)^{-1} \\ \times HP_{t|t-1}A^T + Q \end{aligned} \qquad (8)$$

which represents the standard Riccati difference equation associated with the Kalman filter. For $k \to \infty$, it has a steady-state algebraic equivalence given by

$$P = APA^T - APH^T \left(HPH^T + R\right)^{-1} HPA^T + Q \qquad (9)$$

In general, it is difficult to compare Kalman filters with neural networks. The Kalman filter utilizes a linear system for the localization estimates, whereas neural networks localize the vehicle by learning the mapping between sensor data and 6-DOF poses. However, it is possible to use the steady-state covariance from a Kalman filter as a quality measure for the learning-based estimation. Based on the experimental results, we found an empirical bound for the ML estimation equal to $10cm$. Fig. 7 shows the RMSE error comparison between the KF, our estimate, and the ground-truth.

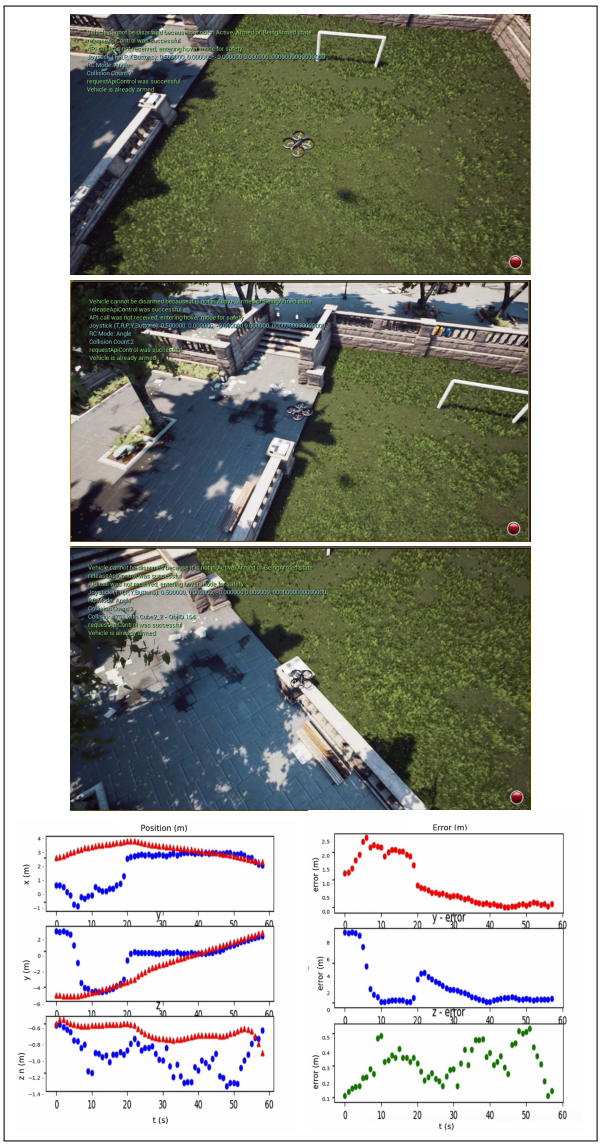

Fig. 8. UAV autonomous navigation and landing in AirSim.

C. UAV Landing

We finally integrate the neural network for pose estimation in a closed-loop flight control system, and we use Airsim as a default simulator. Fig. 9 exploits the general framework of the integration. Airsim has a built-in Python API. A non-custom Python script receives state estimated by our model and sends these inputs to the Unreal environment through the Airsim API. We capture asynchronous images and inertial measurements with associated labels, i.e., real state information. The guidance logic takes as input the current estimate along with the desired target pose and outputs reference velocities to the low-level controller. This input command is then used to control the motors of the vehicle. Fig. 8 illustrates an example of the simulated UAV landing. During the take-off phase, the model is not able to localize the robot accurately because of a lack of training data in that specific area. However, by using information from the

inertial measurements, we are finally able to converge to accurate positioning of the vehicle and to perform the landing on the desired target. Fig. 9 illustrates the system architecture for autonomous landing in Airsim and Unreal Engine.

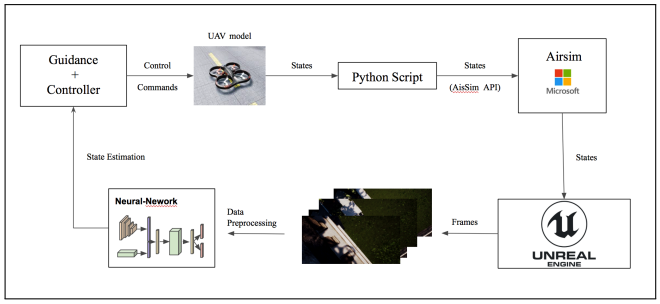

Fig. 9. Neural Feedback Control Framework

D. Comparison to feature-based methods

Traditional approaches for VIO use feature descriptors such as SIFT, SURF, and ORB [30] to detect distinguishable points in the image, such as corners and edges. These features, however, are chosen according to an engineer's judgment and a long trial and error process. To localize the vehicle, feature-based VIO systems are required to track a significant number of features through consecutive images, failing when this requirement is not met. However, given the low-frequency operation rate of the camera, we might not be able to receive temporally-close sequences of images, leading to an unsuccessful initialization or loose of the track. In contrast, learned

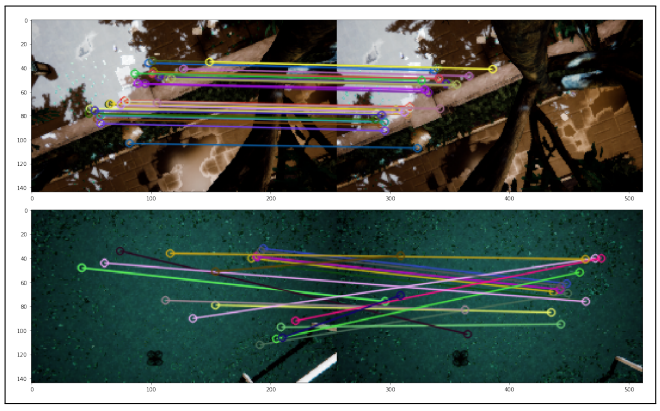

Fig. 10. a) Good vs bad matches. In the first case, the features are easily recognizable, and the algorithm performs a good match between two consecutive frames. In the second case, the wind and the light changes make the tracked points not recognizable in the following frame yielding to a features mismatch.

features are more effective than hand-engineered ones. CNNs are trained to learn features rather than programmed. Hence, CNNs can learn more descriptive and salient features compared to traditional methods, developing better representations for the image data. For this reason, deep learning methods outperform traditional feature-based algorithms.

VI. Discussion and Future Work

The advantage of using deep neural networks over traditional methods relies on its capability to produce a state estimate in just a single step. However, one drawback of supervised deep learning networks is their requirement of abundant training data that must be sampled across all expected operating conditions. If there is not enough training data to present all the expected UAV operating conditions, then the network may not be able to perform accurate localization, as shown in Fig. 8.

a) Learning to Generalize to Dynamic Environments with Meta-Learning: A key challenge for supervised navigation algorithms is the generalizability to handle changing, dynamic environments. Given that it is impossible to generate training data that cover all possible situations the UAV can encounter, we aim to build a meta-learning/adaptive algorithm that allows the model to predict the pose of the vehicle in unseen conditions. Fig. 11 illustrates samples of perturbed *Downtown* environment. Each environment consists of the same central structure, i.e., a grass field surrounded by pillars, with different weather conditions, materials, and lights.

b) Loss function for ego-motion consistency: From the results, we observe that in some cases, the estimated pose is not consistent with the previous estimate. One possible solution is to use prior information about system dynamics at the previous time step to constrain the prediction of the next estimated pose. Hence, future work may be focused on deriving a loss function that incorporates previous motion information to guarantee consistency between continuous streams of data.

VII. Conclusion

We propose a new end-to-end learning method for pose estimation, and we assest the performances of our model with state-of-the-art methods for visual-inertial estimation on the EuRoC MAV benchmark dataset. The results show that our deep learning approach outperforms the baseline compared to its feature-based counterparts. We finally integrate our estimator in the Airsim closed-loop control system, and we demonstrate in a simulation that our data-driven policy can navigate and land a UAV autonomously on its target with less than 10cm of error.

References

[1] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, "Robust visual inertial odometry using a direct ekf-based approach," in *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2015, pp. 298–304.

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

[3] R. Mur-Artal and J. D. Tardós, "Visual-inertial monocular slam with map reuse," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 796–803, 2017.

[4] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *European conference on computer vision*. Springer, 2014, pp. 834–849.

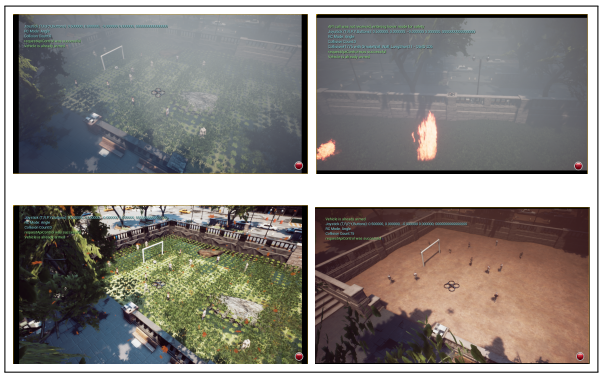

Fig. 11. Images sampled from the modified *Downtown* simulated environments for fine-tuning and meta-learning

[5] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*. IEEE Computer Society, 2007, pp. 1–10.

[6] S. Han, A. Censi, A. D. Straw, and R. M. Murray, "A bio-plausible design for visual pose stabilization," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 5679–5686.

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

[8] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[9] R. Clark, S. Wang, H. Wen, A. Markham, and N. Trigoni, "Vinet: Visual-inertial odometry as a sequence-to-sequence learning problem," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[10] A. Kendall, M. Grimes, and R. Cipolla, "Posenet: A convolutional network for real-time 6-dof camera relocalization," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2938–2946.

[11] 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

[12] C. Forster, M. Pizzoli, and D. Scaramuzza, "Svo: Fast semi-direct monocular visual odometry," in *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2014, pp. 15–22.

[13] L. R. G. Carrillo, A. E. D. López, R. Lozano, and C. Pégard, "Combining stereo vision and inertial navigation system for a quad-rotor uav," *Journal of intelligent & robotic systems*, vol. 65, no. 1-4, pp. 373–387, 2012.

[14] Y. Liu, Z. Chen, W. Zheng, H. Wang, and J. Liu, "Monocular visual-inertial slam: Continuous preintegration and reliable initialization," *Sensors*, vol. 17, no. 11, p. 2613, 2017.

[15] D. Fontanelli, A. Danesi, F. A. Belo, P. Salaris, and A. Bicchi, "Visual servoing in the large," *The International Journal of Robotics Research*, vol. 28, no. 6, pp. 802–814, 2009.

[16] S. Hutchinson, G. D. Hager, and P. I. Corke, "A tutorial on visual servo control," *IEEE transactions on robotics and automation*, vol. 12, no. 5, pp. 651–670, 1996.

[17] 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.

[18] A. Byravan, F. Leeb, F. Meier, and D. Fox, "Se3-pose-nets: Structured deep dynamics models for visuomotor planning and control," *arXiv preprint arXiv:1710.00489*, 2017.

[19] Q. Bateux, E. Marchand, J. Leitner, F. Chaumette, and P. Corke, "Visual servoing from deep neural networks," *arXiv preprint arXiv:1705.08940*, 2017.

[20] O. Shakernia, Y. Ma, T. J. Koo, and S. Sastry, "Landing an unmanned air vehicle: Vision based motion estimation and nonlinear control," *Asian journal of control*, vol. 1, no. 3, pp. 128–145, 1999.

[21] B. Herissé, T. Hamel, R. Mahony, and F.-X. Russotto, "Landing a vtol unmanned aerial vehicle on a moving platform using optical flow," *IEEE Transactions on robotics*, vol. 28, no. 1, pp. 77–89, 2011.

[22] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," in *Advances in neural information processing systems*, 2014, pp. 2366–2374.

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

[24] A. Kendall and R. Cipolla, "Geometric loss functions for camera pose regression with deep learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5974–5983.

[25] J. Delmerico and D. Scaramuzza, "A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2502–2509.

[26] J. Kaiser, A. Martinelli, F. Fontana, and D. Scaramuzza, "Simultaneous state initialization and gyroscope bias calibration in visual inertial aided navigation," *IEEE Robotics and Automation Letters*, vol. 2, no. 1, pp. 18–25, 2016.

[27] S. Leutenegger, P. Furgale, V. Rabaud, M. Chli, K. Konolige, and R. Siegwart, "Keyframe-based visual-inertial slam using nonlinear optimization," *Proceedings of Robotis Science and Systems (RSS) 2013*, 2013.

[28] S. Lynen, M. W. Achtelik, S. Weiss, M. Chli, and R. Siegwart, "A robust and modular multi-sensor fusion approach applied to mav navigation," in *2013 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2013, pp. 3923–3929.

[29] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "On-manifold preintegration for real-time visual–inertial odometry," *IEEE Transactions on Robotics*, vol. 33, no. 1, pp. 1–21, 2016.

[30] E. Karami, S. Prasad, and M. Shehata, "Image matching using sift, surf, brief and orb: performance comparison for distorted images," *arXiv preprint arXiv:1710.02726*, 2017.