Urban MAV

A visual odometry dataset

Q. Booster

for neural network purposes

TU Delft

Urban MAV
A visual odometry dataset

by

Q. Booster

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday April 14, 2021 at 9:30 AM.

Student number:	4358023	
Project duration:	June 17, 2020 – April 14, 2021	
Thesis committee:	Prof. Dr. G. C. H. E. de Croon,	TU Delft, supervisor
	Ass. Prof. Dr. A. Sharpanskykh,	TU Delft
	Ir. C. de Wagter,	TU Delft

This thesis is confidential and cannot be made public until April 14, 2021.

TU Delft

Preface

Within this document the reader will find a report on the master thesis that serves as the conclusion of my years of studying at the Aerospace Engineering faculty of Delft University of Technology. During the years spend on the bachelor and master an interest in programming was born. The subject of this thesis, *the implementation of neural networks in visual odometry*, served to me as an instrument to broaden this interest with knowledge towards machine learning and artificial intelligence.

Never during the entire studies did I expect to perform under such unlikely circumstances as a world-wide pandemic, which affected both my internship as well as this thesis. It was the cause of many online meetings and much more hours of working from home.

Because of this, I found great support with my roommates, with whom the many secluded hours became more enjoyable. The mayor burden of support however laid on the shoulders of my girlfriend, without whom this thesis would have been considerably more cumbersome. I wish for them all a prosperous time ahead, may it be during their studies or the job-life that awaits them.

Support of a more technical form came from my supervisor Guido de Croon. Even though the restricted contact limited the ability of social interaction, the benefit of asking questions was still at hand. This role was also fulfilled by Yingfu Xu, to whom I want to extent these words of thanks.

Finally, I would like to wish the reader an enjoyable and knowledge amending time. Thank you for your interest in this work.

Q. Booster
Delft, March 2021

Contents

List of Figures vii

List of Tables ix

Nomenclature xi

1 Introduction 1

2 Thesis Paper 3

A Literature Review 23

A.1 Introduction . . . 23
A.2 Existing VO Methods . . . 24
A.2.1 Monocular and Binocular . . . 24
A.2.2 Geometric VO . . . 24
A.2.3 Visual Inertial Odometry . . . 25
A.2.4 SLAM . . . 25
A.2.5 Neural Networks . . . 26
A.3 Extant Training Datasets . . . 29
A.4 Datasets Production . . . 31
A.4.1 Physical Data Formation . . . 31
A.4.2 UAV Simulators . . . 31
A.4.3 Jumping the Reality-gap . . . 33
A.5 Deep Neural Networks Potential . . . 33
A.6 Preliminary Tests and Results . . . 34
A.6.1 Selection of the Baseline model . . . 34
A.6.2 Validation results . . . 34
A.6.3 Possible Future Improvements . . . 35
A.7 Conclusion . . . 36

B Discussion 37

Bibliography 39

C Code Overview 45

List of Figures

A.1 Over- and under-fitting demonstration. 27
A.2 DeepVIO pipeline. 28
A.3 Unsupervised depth and pose estimation pipeline. 28
A.4 Recording platform of the KITTI database. 30
A.5 An optical flow comparison of conventional datasets conducted for UZH-FPV. 31
A.6 Translational output of concatenated input. 35
A.7 Rotational output of concatenated input. 35

List of Tables

A.1 Overview of distinct Neural Networks and their characteristics. 29
A.2 Overview of select datasets. 31
A.3 Overview of state-of-the-art drone simulators 32

Nomenclature

ADR Autonomous drone races.

CNN Convolutional neural network.

IMU Inertial measurement units.

LSTM Long short-term memory.

MAV Micro air vehicles.

NN Neural networks.

RNN Recurrent neural networks.

SLAM Simultaneous Localisation and Mapping.

VIO Visual inertial odometry.

VO Visual odometry.

1 Introduction

The most common used classification of drones is Micro air vehicles (MAV), with quadrotors being the most conventional MAV [1]. Its relatively small size and rotary wings provide high maneuverability, including vertical take-off and hovering, making them useful in confined and hard to reach spaces. Another benefit of the MAV in comparison to other drone classifications is its relatively smaller production costs [1]. Because of these features they play an increasing role within our society by aiding in human tasks, by applying them in for example site inspection, agriculture and rescue missions [2–4]. However, its agility comes at a cost which takes the form of high power consumption [5]. Making improving MAV designs a challenge.

To put these challenges to the test Autonomous Drone Races take place. During these events MAVs must traverse a parkour of waypoints. As the name of these types of races indicates the drones must do this autonomously [6]. All calculations and decisions are done using on-board computers. In combination with the above mentioned power consumption constraint this results in outstanding efforts in improving the on-board technology and software by the participating teams [7].

A component of these on-board calculations is the relative position, also referred to as pose estimation [8]. When done using information extracted from cameras this method is called Visual odometry (VO) [9]. These VO algorithms may extract geometric features of image pairs and compare these in order to determine the relative position changes. However, these methods are highly dependent on thresholds set by the user. Additionally, the VO algorithm can be trained using Neural networks (NN), a computing system created to learn various tasks [10]. The system is based on the biological neural network and as such consists out of neurons. These neurons can form a layer. Each connection between a neuron and the input of such a layer has a weight. The weights are adjusted by the loss function in order to learn the task at hand. In general, multiple layers are adopted in sequel, creating a deep neural network [11].

An example of such a deep neural network based visual odometry method is the SfmLearner [12]. Whose architecture consists out of seven convolutional layers and an additional output layer, which results in three rotational and three translational values. Due to its simplicity its architecture provides a solid foundation for this experimental research. However, where the SfmLearner couples the pose estimation to a depth estimating neural network to create a unsupervised method, this research will leave out the depth estimations and thus enhance the architecture for supervised network purposes [12]. The rationale behind this springs from the intention of the research, being to further fuel the advancements in neural network based monocular visual odometry methods.

Provided by chapter 2 is the article that contains this thesis research. After testing the above described neural network on a variety of high-end datasets an additional challenge is brought to light, as overfitting becomes a regular occurring phenomenon. Resulting in the need for and creation of a new state-of-the-art dataset. As these results are described in a paper format, it can be read as a stand-alone report. However, an initial literature study was conducted that laid the foundation of this research.

As this is not described within the paper, Appendix A specifies the required background information on the available literature as well as the initial interpretations of this research. After which Appendix B concludes with a final discussion. Additionally, some relevant components of the codes used to train the neural networks and create the dataset can be found in Appendix C.

2 Thesis Paper

The following chapter presents the master thesis results on neural network based visual odometry. As to conform to the required format, it has been written in article style. It can therefore be read as a self-contained document.

First off, it will provide some introduction on the topic. After which the equipment that is used to train the neural network is established. As images are often distorted, the pinhole camera calibration method is described, which enables a image to seem more similar to real life scenarios. Hereafter, a description on the importance of overfitting is provided and the used calculations of accuracy are specified. From this point on some state-of-the-art datasets are introduced. Results of whom on a neural network are then elaborated upon. It is then that the necessity for a modernized dataset with decoupled motions is established. The method of creating such a dataset is described after which the article is concluded.

The UrbanMAV Dataset for Learning-based Visual Ego-Motion Estimation

Q. Booster
Delft University of Technology

Abstract **- Visual odometry is an important component of most vision-based autonomous flight systems. Current visual odometry algorithms which are based on feature detection and tracking suffer when the robot performs quick rotations or in general moves fast through its environment. Neural-network-based visual odometry (VO) forms a promise to better cope with such fast motions. However, current datasets seem unsuited for training the neural networks for such tasks. Mostly, the variation in different types of motions in these datasets is quite low. Moreover, different motion directions always occur in an entangled manner. Hence, a new dataset is created specifically for training neural-network-based VO. Furthermore, a method of training a neural network with various outputs is created with promising results in balancing the learning process, even if variations in motion are limited.**

keywords - Visual Odometry, Micro Air Vehicles, Neural Networks, Dataset

1 Introduction

The modest size and high maneuverability of micro air vehicles (MAVs), makes them suitable for helping humans in tasks such as site inspection, agriculture and rescue missions [1–3]. However, limited on-board space, weight and power impose a challenge on the advancements MAV designs [4]. Outstanding efforts towards solving these complications have been shown by teams participating in autonomous drone races [5].

An area of advancements is the estimation of the flight path of the drone. This can be done by means of estimations of relative poses between image frames of on-board cameras. This methodology is referred to as visual odometry (VO). Feature based methods unfortunately are restricted by the required threshold values [6–8]. Neural networks can be trained in order to function as such VO methods [9–11].

An automobile based neural network for visual odometry is SfmLearner [12]. Its simple architecture lends itself well for further research. The original KITTI dataset as well as two drone-based datasets UZH-FPV and EuRoC are used to train a supervised version of the SfmLearner [13–15]. Upon examination of the resulting learning curves, great discrepancies are found between the various outputs of the neural network. The neural network tends to focus on learning the outputs that tend to reach higher values, as intrinsically these have higher error margins. Which takes away from the learning process of the other outputs. To balance this a contemporary method is invented, which provides a smoother learning curve during training and enables multiple outputs to be trained in conjunction in a more stable manner.

On top of the out of balance training, overfitting seems to be a regular occurring phenomenon. It is concluded that the existing drone focused datasets are not large enough to support neural network based visual odometry. In order to fill this gap, this paper introduces a dataset, which holds a larger amount of image pairs. It is hypothesised that improved results can be attained if the movements in the six degrees of freedom of a drone are decoupled when recorded for the dataset. It is believed that this decoupled motion will provide a stronger basis for the neural network to learn to the recognise the separate motions. This data is created using a simulated environment from UnrealCV [16]. Using Python packages enabling communication with the simulation, the creation of carefully selected decoupled data was enabled.

2 Methodology

The following section provides some background information on the training process. First-off, required camera calibration for real-life images is provided. After which several technical details on the computer setup is given. Lastly, over-fitting and general ways to overcome this are introduced, as-well-as a method of calculating the accuracy of the neural network.

2.1 Pinhole camera calibration

Although modern cameras are easily capable of capturing moments in the everyday life, they do so with a distortion due to lens parameters. Since every camera

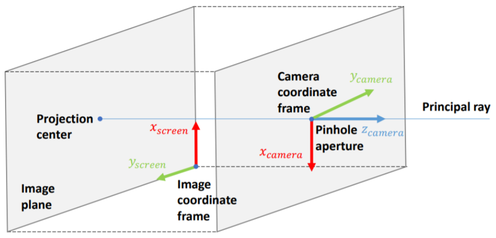

Figure 1: Axis systems required for pinhole camera model [17].

is different, intrinsically their distortions differ as well. These discrepancies in comparison with how the world is represented can be described by the pinhole camera model. In order to map the distortions two axis systems are required

1. Camera axis system
2. Image axis system

as depicted in Figure 1. The relation between these two axis systems is described by the camera matrix in Equation 1. Where f is the distance between a point on the xy-plane in the image axis system and the camera axis system and c_x and c_y describe the difference of origin between the two systems [17].

$$\begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{1}$$

Radial distortion due to the lens is not yet described by the above method. In order to encompass these distortions a polynomial based method can be used, whose coefficients are referred to as distortion coefficients [18].

The OpenCV python library makes use of both these methods in order to describe the distortions on images [19]. The effect of undistorting images using this method is depicted in Figure 2. Although curved element in the image have been rendered out, making the image more comparable with the real world conditions. However, a striking disadvantage is the loss of the outer pixels of the image.

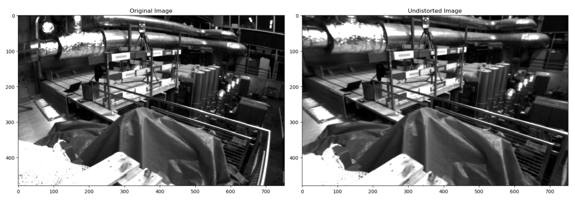

Figure 2: Effect of the distortion modelling by OpenCV.

2.2 Technical specifications

Training and testing of the neural networks can be done using a variety of soft- and hardware. For the latter, a Hewlett-Packard ZBook Studio G5 was available, equipped with a CUDA enabled NVIDIA Quadro P1000, a mid-tier graphics card [20]. By leveraging the available GPU with CUDA improved training speeds can be achieved, due to parallel computations, in comparison to the available CPU (Intel Core i7-8750H). By employing the Pytorch library development time can be kept to a minimum due to its ease of use. Additionally Pytorch can leverage the improved speed of graphics card by enabling CUDA [21].

2.3 Over-fitting and accuracy calculation

When working with neural networks, a pitfall can be overfitting. At first sight the results of a neural network may seem to correspond very well to the ground truth. However, the neural network may only be capable of showing such results on the data it has trained on. As a result of the lack of generalization, the network then can show bad performance when working with data that is new to it, which is very common in practical usage. One such scenario of overfitting is the result of training on parameters with low extensibility to other scenarios, such as different locations, routes or speeds. As a result the neural network is unable to learn useful algorithmic parameters and instead, almost like mnemonic, learns to couple certain image pairs with the corresponding location.

In order to overcome overfitting complications it is essential to get a hold of enough data [22]. Too small quantities can result in a network that is unprepared for data external of the training dataset. It is therefore fundamental to be aware of different methods that can provide improved training datasets.

First off, an indisputable method would be to create more data in a similar fashion as the methods described above [22]. Advantage being the ability to focus on the qualities that other datasets are lacking gives it much potential to improve the resultant neural network. However, this would require time and the essential equipment and is therefore best kept for scenarios where results can not be improved in other ways.

Instead the existing data can be adjusted in order to enlarge the dataset. This can be done by deforming the images [22]. In doing so caution must be taken into processing the data in a sensible manner. For example, the available KITTI dataset would not make sense if the input images were to be recycled up-side-down. Horizontally mirroring the data however can double the available data. It must be taken into account that the labels are adjusted accordingly, to correspond to the deformed image pair. Another example of such data adjustments with a lower risk of serving the neural network improper data would be to randomly adjust brightness of the images so that the network becomes less prone to situational factors.

One way a neural network may improperly learn the required patterns is by over-feeding it with the same type of data. An example for the neural network trained on KITTI-only data would be a long straight road with constant speeds which may lead to the neural network overfitting on the specific translational output. For this reason the Pytorch library possesses a shuffle function. This function is applied for all the networks trained in this work and will shuffle the whole dataset like a deck of cards to overcome some of the possible training errors. The shuffle entirely random and will take place before every epoch, so that no recurrence of patterns in the dataset takes place. However, this only holds for the training dataset. Those who are used for the accuracy computations, introduced in the following section, do not undergo this shuffle in order to depict a more real-life situation.

In order to value this data and keep track of the performance of the neural network it is important to test the accuracy of the neural network throughout training. By splitting of a part of the database and not using this for training purposes the opportunity to validate the network is created. By classifying the output for both validation and training data the performance can be estimated. This can be valued using an accuracy measure or by looking at the loss values.

3 Accuracy calculation

A conceivable way of describing the accuracy of an estimation of a neural network is dividing the estimation error by the true value, leading to Equation 2. Where η is the methods accuracy, α is the true value and β is the estimated value. With the error being equal to the true value minus the estimated value.

$$\eta = \begin{cases} 100\% \cdot (1 - \frac{\|(\|\alpha\| - \|\beta\|)\|}{\|\alpha\|}) & \|\alpha\| > \|\beta\| \\ 100\% \cdot (1 - \frac{\|(\|\beta\| - \|\alpha\|)\|}{\|\alpha\|}) & \|\alpha\| < \|\beta\| \\ 100\% & \|\alpha\| = \|\beta\| \\ 0\% & \|\eta\| > 100\% \end{cases} \tag{2}$$

However, problems with this equation arise once the sign of true value and estimated value differ, as this causes high accuracy values to drop below zero which may be less intuitive. To overcome this problem all accuracies belonging to estimations with incorrect signs are set to zero, no matter their correctness if signs were truthful.

In order for these plots to be made every epoch two tests are done. One serves as a analysis of the training, a single sequence which has also been used for training is analysed by the neural network without shuffle and with a batch size of one, similar to a real life situation. From the output the relevant training accuracy is calculated via the method described above. However, overfitting is invisible in this data and so a distorted image is created of the result. In order to bring the result into perspective the second test makes use of a sequence which has been left out of the training data. Again without a shuffle and with a batch size of one, the test accuracy is calculated. These are the recurrent values that are found in most of the figures.

4 Datasets

The following section introduces the various datasets that are used throughout this research.

4.1 KITTI

KITTI is frequently used when developing odometry methods for robotics. It consists of various databases, recordings from cameras attached to a Volkswagen station wagon, each with its own purposes. It can for example be used for object or road detection and tracking. A database for visual odometry has been made available, where IMU and GPS data provide accurate positional information throughout eleven available sequences [13].

As the objective of this report is the relative pose estimation between images, all unrelated information can be discarded. What is left are the images captured by the camera rig, mounted on top of the roof of the car, alongside the Velodyne Laserscanner, whose translational information is also utilized. Finally, IMU provides the rotational information [13].

Figure 3 shows a depiction of the locations and axis systems of the above mentioned data capturing instrumentation. The Velodyne Laserscanner and GPS/IMU appliances share their axis directional conventions but do not share their positioning. The left and right video cameras have their own axis conventions, spawning from the camera models discussed in subsection 2.1. All translational and rotational information during each sequence is provided with respect to the left video cameras axis system. Considering this alignment and the interest in monocular vision, it goes without saying that the images of the left camera are preferred over the, therefore excluded, images of the right side camera.

All information is conveyed with respect to the position of the first frame in the sequence. However, the camera axis system moves along with the car, so all relative pose information is desirably with respect to the camera at the frame in question. To solve this Equation 3 gives the translational matrix between a global and body frame. This world frame is portrayed by the

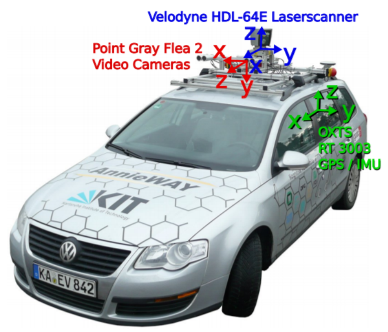
Velodyne HDL-64E Laserscanner
Point Gray Flea 2 Video Cameras
OXTS RT 3003 GPS / IMU

Figure 3: Recording platform of the KITTI database [13].

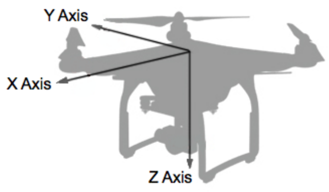

Figure 4: Conventional axis system [1].

first camera axis stance and the body frame is the camera position of the first image in the evaluated image pair. However, the camera axis system is first converted to the conventions indicated by Figure 4, with the roll angle ϕ around the x-axis pointing forward, pitch angle s θ around the y-axis pointing right and ψ yaw angle around the z-axis pointing to the lower side of the recording platform, following the right hand rule.

4.2 UZH-FPV

When training a visual odometry for micro air vehicles the movements of automobiles are incomparable due to their smaller use of pitch and roll motions. Furthermore, drones are capable of making more agile manoeuvres. Therefore, if a neural network based visual odometry method is supposed to deliver results on a drone, it is best practice to train this network using data that is captured using a drone rather than cars. One such database is the UZH FPV. More specifically it is labeled as a racing drone database [14].

The images for this dataset are recorded with a snapdragon stereo camera and the mDAVIS event camera. Considering the desired practicality on a wide variety of drones the latter data is omitted from this research. Furthermore, the snapdragon has both a horizontal and downward angled setting. As the former configuration can likewise be adopted for obstacle recognition, this is preferred over the downward facing camera [10].

For odometry method evaluation and supervised neu-

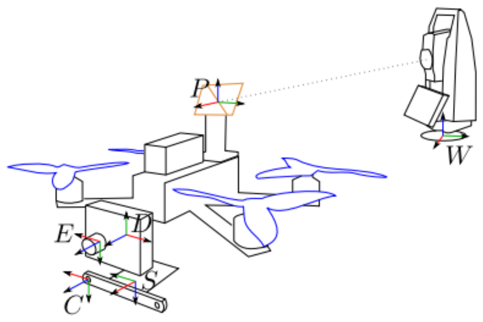

Figure 5: UZH-FPV database equipment axis systems [14]. The x, y and z axis are indicated in red, green and blue respectively.

ral network training it is essential to have labeled data. The ground truth values for this data are obtained with the help of a laser tracker, locking on a prism on top of the drone. The corresponding rotational values are recovered using an on-board IMU. The equipment axis systems are illustrated in Figure 5, where it can be seen that not all coordinate frames correspond in their directional labeling. In order to convert the IMU data into the left camera axis system Equation 4 is applied.

4.3 EuRoC

Another drone based dataset is EuRoC, created purposefully for visual odometry methods and three dimensional reconstruction of terrain [15], the latter functionality not being of use for this research. As with UZH-FPV, images were recorded with a stereo camera, ground-truth locations are determined via a laser-prism combination and IMU measures the angular rate.

The EuRoC ground truth data is provided in a similar fashion as the UZH-FPV dataset in subsection 4.2. For this reason the same methodology for data operation can be applied, with corresponding adjusted matrix values, as found in Equation 5.

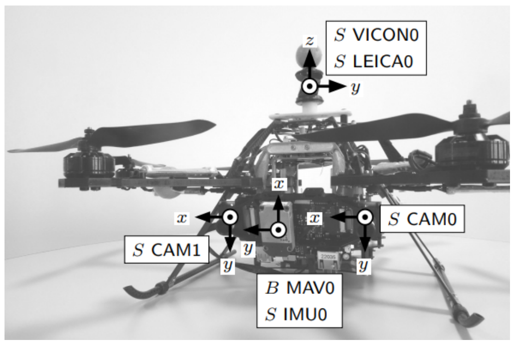

Figure 6: Axis system indication for the EuRoC database recording platform [15].

$$\mathbf{R}_G^b = \begin{bmatrix} \cos(\psi)\cos(\theta) & \sin(\psi)\cos(\theta) & -\sin(\theta) \\ \cos(\psi)\sin(\phi)\sin(\theta) - \cos(\phi)\sin(\psi) & \sin(\phi)\sin(\psi)\sin(\theta) + \cos(\phi)\cos(\psi) & \cos(\theta)\sin(\phi) \\ \cos(\phi)\cos(\psi)\sin(\theta) + \sin(\phi)\sin(\psi) & \cos(\phi)\sin(\psi)\sin(\theta) - \cos(\psi)\sin(\phi) & \cos(\phi)\cos(\theta) \end{bmatrix} \tag{3}$$

5 Testing results

5.1 Results on KITTI

Due to the simple architecture of the SfmLearner neural network, see Table 1, it is forms a starting point for this research. Although the goal of this research is to construct a visual odometry method for micro air vehicles, the SfmLearner originally was trained on the KITTI database. For this reason it was decide that initial tests should be conducted using the KITTI database as well. However, in order to do so the networks parameters required adjustment, due to its unsupervised nature. A summary of the parameters of this enhanced version are found in Table 2.

With these parameters initial tests were conducted. Unfortunately these were unsuccessful in reaching similar results as the unsupervised SfmLearner. Although translational estimations on the trained data were able to reach accuracies of over 90%, these same calculations were only 75% accurate on validation data. For the sake of consistency two tests were conducted.

1. Training on all sequences except for sequence 00, testing on sequence 00, training accuracy given by sequence 03
2. Training on all sequences except for sequence 05, testing on sequence 05, training accuracy given by sequence 00

Both yielded similar results, former can be found in Figure 21 in section 7 and the latter being depicted in Figure 7. Over-fitting is mainly visible for the translational values. Considering that the x-axis is the only axis for automobile data to have substantial translational value, its overfitting is of great significance and could have implications on the method when applied to drone data. It is for this reason that the results on the y and z axis were of smaller relevance during the following process. Similarly, only the yaw angle plays a significant role for automobile data, while drones may use pitch and roll in a more noteworthy fashion. The discrepancy between the angular and translational accuracy as well as the neural network overfitting led to an exploration for result improving adjustments, sticking with the KITTI database. Due to the validation and training accuracy of similar values, although low at around 20 to 30 %, for the roll and pitch angles it is believed that adjustments may lead to less overfitting.

However, considering that the original implementation of the SfmLearner converged to a higher accuracy with less overfitting it seemed appropriate to test whether the by SfmLearner applied sequences yield similar results. This concerns sequences 00 through 08 as training data, with both sequence 09 as well as sequence 10 as validation data. When applying this training and validation method it results in Figure 8, where it can be seen that yet again it results in limited accuracies and overfitting patterns, as well as discrepancies between translational and rotational outputs. It is for this reason that it is believed that using the priorly discussed sequences, is justified in the search for the improvement of the learning pattern.

5.2 Splitting the loss

By having a single array as output, and feeding this to a single mean squared error loss function, a skewed gradient may be a consequence. This could be a reason for the different patterns between the translational and rotational values found in Figure 8. To overcome this problem a new training method was found.

Instead of using a single mean squared error loss criterion for the whole array, where the largest value will have the largest influence. The forward motion is most prominent, so learning this translation well results in low loss values. The other motion axes and rotations matter less for the loss value, and thus it is hypothesizes that they are less well learned because of this fact. And thus it is expected to be more efficient when each degree of freedom is similarly trained by equalizing the training gradients. For this reason the array was split into six separate variables, each corresponding to one of the degrees of freedom. Each one of these six variables can be calculated using a separate mean squared error loss function conveying the total loss given by Equation 6.

The next step is to stabilise the learning process by awarding each output value the same influence on the

$$\mathbf{T}_{cam,UZH}^{imu} = \begin{bmatrix} -0.02822879 & -0.99960149 & 0.00001218 & 0.02172388 \\ 0.01440125 & -0.00041887 & -0.99989621 & -0.00006605 \\ 0.99949774 & -0.02822568 & 0.01440734 & -0.00048818 \\ 0. & 0. & 0. & 1. \end{bmatrix} \tag{4}$$

$$\mathbf{T}^{imu}_{cam,UZH} = \begin{bmatrix} 0.0148655429818 & -0.999880929698 & 0.00414029679422 & -0.0216401454975 \\ 0.999557249008 & 0.0149672133247 & 0.025715529948 & -0.064676986768 \\ -0.0257744366974 & 0.00375618835797 & 0.999660727178 & 0.00981073058949 \\ 0. & 0. & 0. & 1. \end{bmatrix} \tag{5}$$

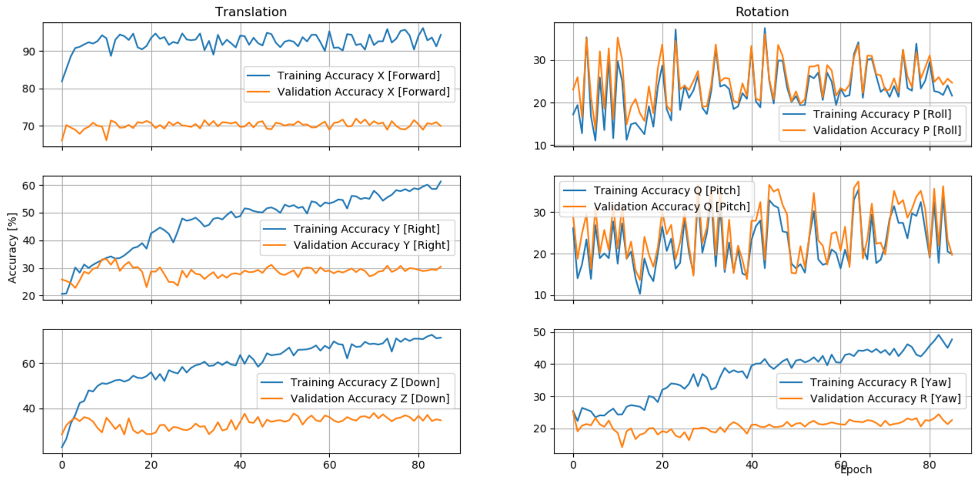

Figure 7: Translational and rotational results on supervised version of the SfmLearner. Trained on sequences 00-04 and 06-10, validated on sequence 05, of the KITTI database.

gradient. This was established by appointing weights, indicated by ω. in Equation 6.

The theory is build around the fact that the neural network is trained using various batches each with their own values for the six degrees of freedom. Therefore, each new batch requires different weights in order

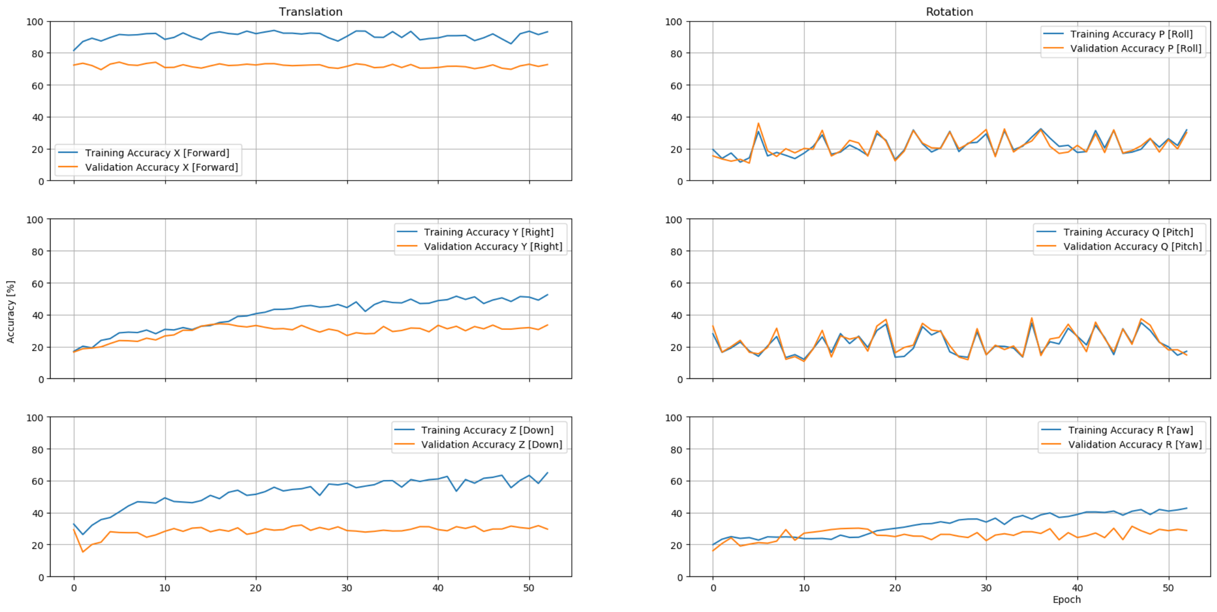

Figure 8: Translational and rotational results on supervised version of the SfmLearner. Trained on sequences 00 to 08, validated on sequence 09 and 10, of the KITTI database.

Table 1: SfmLearner Architecture

Layer	Input	Output	Kernel	Padding	Stride
One	16	32	7	3	2
Two	32	64	5	2	2
Three	64	128	3	1	2
Four	138	256	3	1	2
Five	256	256	3	1	2
Six	256	256	3	1	2
Seven	256	256	3	1	2
Out	256	6	1	-	-

Table 2: Supervised enhancement parameters for the SfmLearner

Parameter	Characteristic
Loss criterion	Mean squared error
Optimizer	Adam (0.9,0.999)
Layers	See Table 1
Activation function	ReLU
Batchnormilisation	Not in place

to be trained most optimally. Where it is most optimal to distribute the error values over all six degrees of freedom. These weights can be set according to the batch they belong to, by leveraging the known label values. Various variations for such a weight formula can be applied. The minimum or maximum value in the batch's label for each degree of freedom can for example be a factor in the determination of weight values. Here the minimum value would impose a logical advantage, as a division by the minimum value per variable would increase the effect of lower outputs (angles) over those with higher values (forward motion). However, it must be taken into account that a negative weight would destabilise the learning curve and so taking the absolute value may be a requirement for labels and outputs that can take a negative sign.

To confirm the possibility for success of splitting the outputs an initial test was conducted. In this test set up the translational outputs are omitted. To achieve this the output layer in Table 1 has an output size of three, which are then fed to the loss function as an array together with the respective rotational labels. The emerging accuracies are found in Figure 9. Unlike beforehand, a learning curve is displayed by the training accuracies for all angles. However, an overfitting pattern is shown with the validation accuracy converging to around 30% for each rotation.

This would suggest the possibility of improving the overall result for the rotational outputs. Considering the increasing accuracy for training data, up to 70%, was invisible in previously conducted training sessions for all

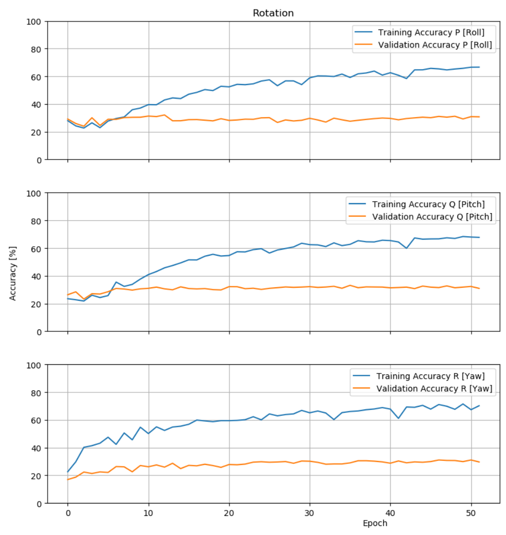

Figure 9: Results for a rotational output only supervised version of the SfmLearner. Trained on sequences 00-04 and 06-10, validated on sequence 05, of the KITTI database.

rotations it would seem that the loss value of the translational output overshadows those of rotational values. Thus the hypothesis proposed in this section seems to have been proven and the pursuit of splitting the output should eventually provide a more balanced result between rotational and translational output.

5.3 Results on UZH FPV

When adopting the above described methodology to the UZH FPV dataset it results in the learning process found in Figure 10. The neural network is trained on all forward facing sequences, both indoors as well as outdoor, except for the validation data which is the indoors sequence number five. The accuracy on the training data is determined using the indoor sequence nine. Hereby, it is specifically chosen to have indoor sequences for both the validation and training accuracy for a fair comparison, as textures in both scenarios may vary.

For all six degrees of freedom an overfitting effect is found. However, in comparison to the KITTI database the learning procedure is more balanced between the various translational and rotational outputs. The worst accuracies belong to the rotational outputs, with roll and pitch in particular.

Further research with previously described methods did not develop in a decrease of these overfitting results. Reason for this is estimated to be the sparsity of data. It is believed that neural networks require more data in order to be better suited for scenarios outside the scope of the data that it is trained on.

$$\mathcal{L}_{total}(x, y, z, p, q, r) = \omega_x \cdot \mathcal{L}_{MSE}(x) + \omega_y \cdot \mathcal{L}_{MSE}(y) + \omega_z \cdot \mathcal{L}_{MSE}(z) \\ + \omega_p \cdot \mathcal{L}_{MSE}(p) + \omega_q \cdot \mathcal{L}_{MSE}(q) + \omega_r \cdot \mathcal{L}_{MSE}(r) \tag{6}$$

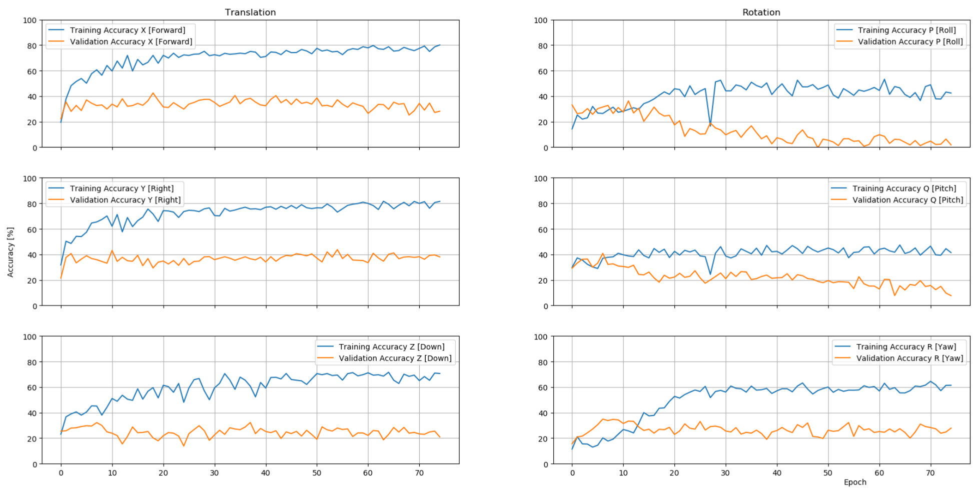

Figure 10: Results on a supervised version of the SfmLearner. Trained on sequences forward sequences, of the UZH-FPV database, except for *indoor*5, on which it is validated.

5.4 Results on EuRoC

EuRoC is a database that contains more data than the UZH-FPV database and thus may be less prone to the aforementioned overfitting problems. To put this hypothesis to the test the neural network was subjected to the EuRoC database, except for the machine hall 3 sequence which functioned as validation data. The sequence labeled as machine hall 5 was then used to determine the accuracy on the training data.

Results of this test are depicted in Figure 11. In comparison to previous results on the other databases, less overfitting occurs as can be seen from the decreased distance between the training and testing accuracy line. However, the accuracies of the angle estimations do not properly advance. Instead they stay relatively the same, and the yaw accuracy even shows a decreasing trend. It is for this reason that the training was stopped prematurely. However, conclusions can still be drawn. For instance, the decrease in overfitting could point to the effective use of a larger dataset. The worsening learning curve of the angles could be the aftermath of a decreased activity in the angular velocities in comparison to the UZH-FPV dataset.

Performance on the EuRoC dataset was then improved by using the splitted loss methodology which was previously constructed for the KITTI database, resulting in Figure 12. Here, the weight calculation in Equation 7 was adopted. Furthermore, the SfmLearner makes use of a 0.01 multiplication for the final output layer for each degree of freedom. However, as each output may not be in a similar range these were adjusted according to the output resulting in the array of 0.0001 and 0.01 rotation and translation respectively. This further increased the balance between the learning processes of the six outputs. As a result the decreasing yaw angle now improved, although only to 25%. Furthermore, the learning process is more balanced between the various outputs, resulting in a slightly delayed and volatile learning curve for the translational outputs. Unfortunately, these results still show overfitting.

Given the consistent overfitting of SfMLearner on the training data, it is hypothesized that, to achieve a neural network based visual odometry method that is applicable outside the scope of the training data, a new state-of-the-art dataset has to be created. This would need to contain at least an equal amount of data as that of the EuRoC database, as well as similar limits to the movement speeds as the UZH-FPV dataset.

6 The UrbanMAV Dataset

Considering that the existing datasets may be too small a new dataset was established. By operating a simulator and adopting points of improvements gathered from the previously described methodologies a state-of-the-art visual odometry focused dataset was created.

The limits of the translational and rotational mo-

Degree of freedom	Minimum	Maximum
$\frac{\Delta X}{\Delta s}$ [m/s]	-25.00	25.00
$\frac{\Delta Y}{\Delta s}$ [m/s]	-12.00	12.00
$\frac{\Delta Z}{\Delta s}$ [m/s]	-9.50	9.50
P [deg]	-45	45
Q [deg]	-45	45
R [deg]	-180	180
$\frac{\Delta P}{\Delta s}$ [deg/s]	-375	375
$\frac{\Delta Q}{\Delta s}$ [deg/s]	-375	375
$\frac{\Delta R}{\Delta s}$ [deg/s]	-375	375

Table 3: Flight envelope of the UrbanMAV Dataset

tion were greatly derived from the UZH-FPV dataset, as this is considered to contain aggressive sequences. After close analysis the limits were established to be those in Table 3. Single valued sequence makes use of a predetermined velocity and or angle. Initial positions as well as relative poses were randomly chosen using a uniform distribution.

Data is gathered using the Urban model environment of UnrealCV [16]. Commands can be given to this virtual world via a python client. These commands include setting the location of the camera, applying its angles as well as extracting images. As these frames are derived from a simulated camera, the methodology of camera calibration in subsection 2.1 is not required. By calculating relative poses, and from that the global positioning, within the scope of the above described limits the ground truth can be pre-determined, and the corresponding frames can be found accordingly.

With an eye on building viable data various sequence types are created. Most notably are those with a single degree of freedom. Where both EuRoC and UZH-FPV contain only sequences that adhere to all degrees of freedom in conjunction, the UrbanMAV dataset possesses data with isolated motions, so only one degree of freedom was active at a time. Multiple sequences for the translational movements exist, with varying velocities depending on the axis that is excited. With these sequences it is anticipated that independent outputs can be trained more conveniently, as previous sections demonstrated the adversity with which this is currently implemented. However, as means to avert overfitting, additional data sequences with various degrees of freedom are created as well.

These random sequences are constructed not as true flights, as with EuRoC and UZH-FPV, but as random image pairs throughout the viable locations in the simulated environment. Per image pair a random location is established, after which a relative pose for the camera is obtained from within the above described scope of limits. Using the pre-established global positioning, in particular the Euler angles, Equation 3 is applied to transfer the relative pose into the global frame with which the second camera position can be found. Using these two camera positions the frames that accompany the relative pose can be constructed and so a dataset with image pairs on random locations with random (relative) poses is built up. Due to its arbitrary values, it is expected that learning may be less prone to overfitting.

Due to the ease of creation of both random data with multiple degrees of freedom, as well as semi-random single degree of freedom data, a dataset of considerable size was created. All the sequences were created within the

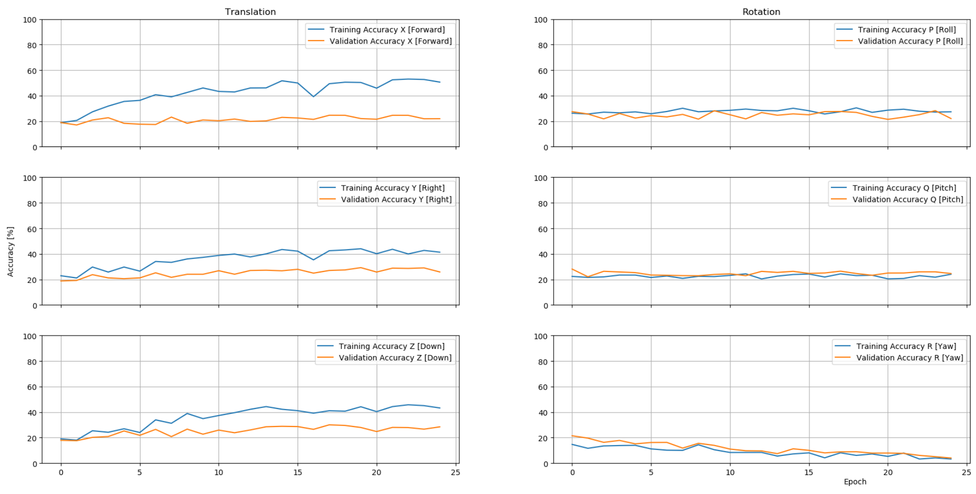

Figure 11: Results on a supervised version of the SfmLearner. Trained on sequences forward sequences, of the EuRoC database, except for *machinehall3*, on which it is validated.

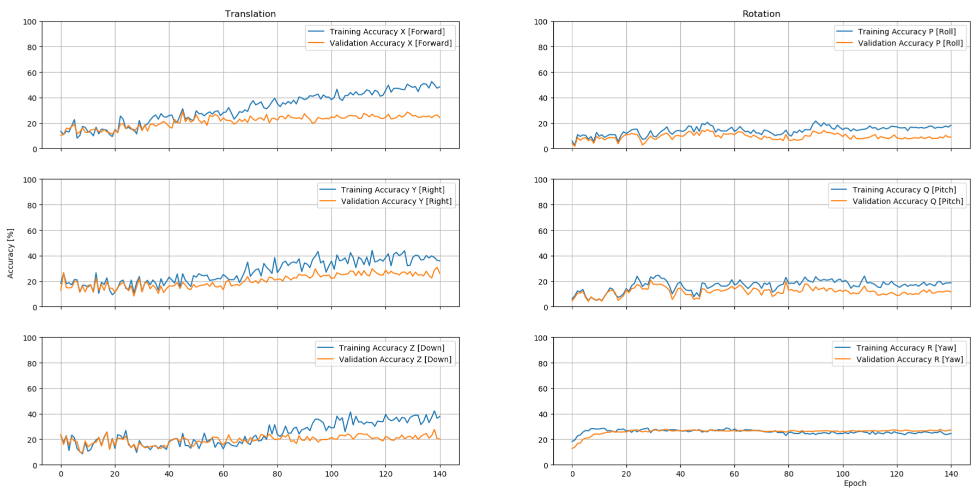

Figure 12: Results on a supervised version of the SfmLearner. Trained on sequences forward sequences, of the EuRoC database, except for *machinehall*3, on which it is validated.

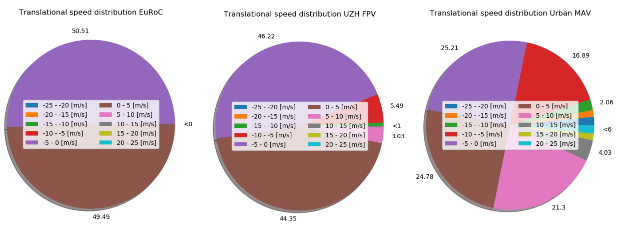

Figure 13: Distribution of the translational velocities for the EuRoC, UZH and Urban MAV datasets.

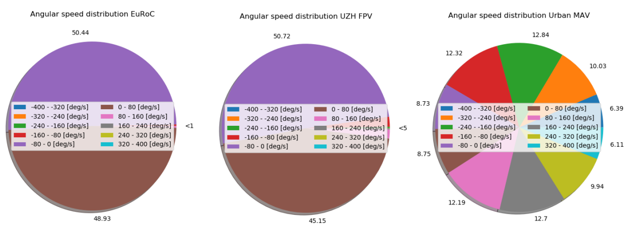

Figure 14: Distribution of the rotational rates for the EuRoC, UZH and Urban MAV datasets.

green area indicated by Figure 17.

The distribution of the translational values of the EuRoc and UZH are depicted by Figure 13. From these pie charts it is clear that the EuRoC dataset does not excite the translational values much. The UZH data reaches translational values of around 23 m/s. However, these are only found sporadically within the data. Eventhough they push the limits of the dataset, in essence it only has a minimal effect on the training of a network due to the overshadowing lower values. However, as these higher values do occur within the data they drive the accuracy of the network down. When looking at Figure 13 the pie-chart indicates the distribution of the UrbanMAV data. Although the limits are not dramatically higher than that of the UZH-FPV dataset, the upper boundaries are more frequently visited, thus delivering a more sensible arrangement of values.

Again it holds for both EuRoC and UZH-FPV that the distribution of angular velocities make for relatively low speeds throughout the datasets. This is illustrated by Figure 14. However, as showcased, the angular velocities of the Urban MAV dataset reaches the outskirts of the constraints in a close to uniform distribution, hopefully making the data more suitable for neural network based visual odometry methods.

However, as stated earlier not only the distribution of data but also the amount of data plays a vital role in determining whether a dataset is suited for neural networks, as smaller datasets are more prone to overfitting. Figure 16 showcases the differences in proportion of the state-of-the-art databases. It is important to take note of the unit of this graphical representation. As it states the number of images pairs are depicted, which differs from the total amount of images that a dataset may hold. Most importantly, the randomized data within the Urban MAV dataset is created solely per pair, and thus the total amount of images within these sequences is twice as large as is depicted within Figure 16.

An example of an image that is extracted from the Urban environment with the means of building one of the straight forward motions is found in Figure 18. The size of these images is 640 by 480. Notably, the road

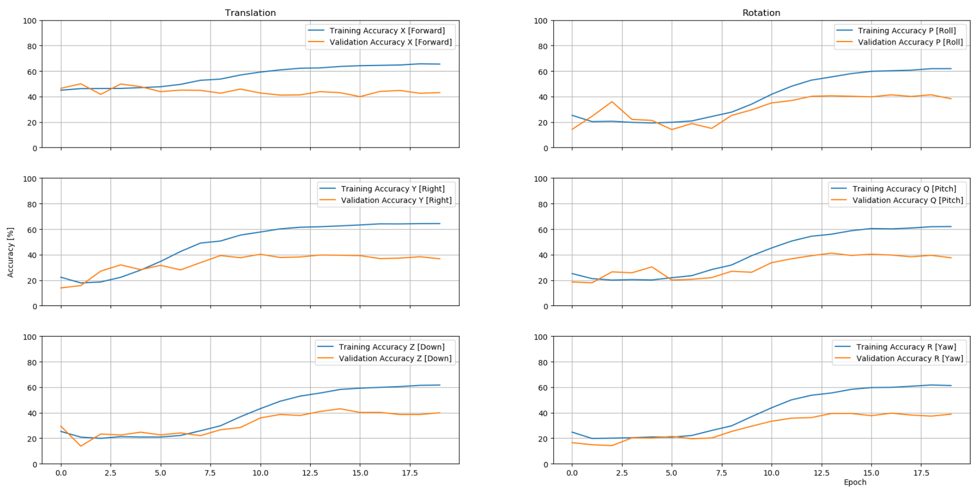

Figure 15: Translational and rotational results on supervised version of the SfmLearner. Trained on sequences the randomsequences 2 - 25, validated on random sequence 1, of the Urban MAV database.

is simulated to be covered with water using a reflective effect. Furthermore, leaves are on the ground giving the road more texture. These leaves also occasionally fall from the trees scattered around the map. As these trees, green dots on the map in Figure 18, sometimes are close or overlap the roads and therefore the pathways of the drone, these falling leaves are occasionally recorded. Although not beneficial for the determination of relative poses, it is believed that these will not cause much harm and are therefore left in the dataset. Instead, it is believed that in real-life situations similar occlusions may occur and thus the trained neural network would be less prone to false estimations in such scenarios. However, the dataset is unlike the real world as it misses motion blur at greater speeds and is only recorded within a single environment. In this aspect improvement can still be found.

In order to verify the applicability of this dataset a couple of tests were run. During these tests it became apparent that the new dataset indeed is capable of being trained on by the neural network. Moreover, the learning curve is more equally distributed between the six degrees of freedom. As a result splitting the loss methodology was not required and thus not used. This can be a huge advantage over the existing datasets as this means that weight functions will not require tweaking towards the dataset. Unfortunately, again training results in overfitting, where the accuracy training values reach 60% while validation accuracy only 40%, as seen in Figure 15. Due to the ease and smoothness of training it is believed that great results can be achieved using this dataset. An example of this is depicted in Figure 19, where it can be seen that the neural network

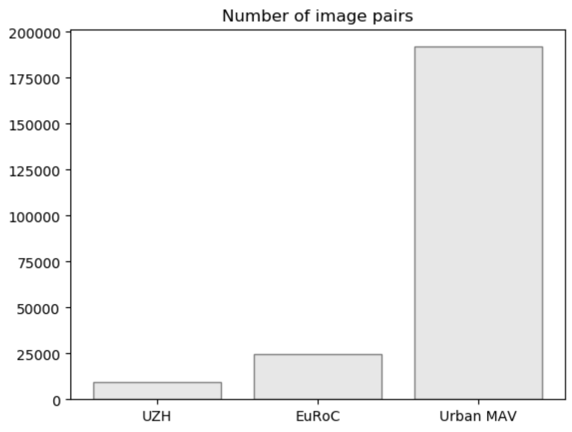

Figure 16: Representation of the amount of viable image pairs within the UZH-FPV, EuRoC and Urban MAV datasets.

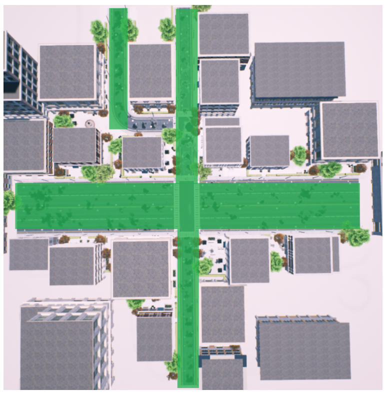

Figure 17: Map of the Urban model environment of UnrealCV [16]. Green indicates the area in which is flown for the creation of the UrbanMAV dataset.

Figure 18: Example of an image frame extracted from the Urban model environment of UnrealCV for the UrbanMAV dataset [16].

is already capable of converging to the correct range of values. Additionally, higher accuracies can be achieved by decreasing the learning rate step wise. The addition of the decoupled motions has not proven itself in improving these results. It is believed that a new extensive research into applying this data could shed light into this. Nonetheless, the resulting development in the smoothness of the learning process are promising and it is thus believed that this dataset will contribute to the development of neural network based visual odometry.

7 Conclusion

A neural-network-based visual odometry method was studied during which it was found that the state-of-the-art method SfMLearner over-fits when being trained on existing driving and flying datasets. Moreover, its outputs mainly converge for translational motions. Multiple solution avenues have been investigated. First off, an updated variation on learning a neural network in a supervised manner with multiple outputs was established. Weighing motion axes inversely to their occurrence in the dataset resulted in an improved balance of the learning curves of the outputs. Furthermore, it was established that existing datasets do not hold a sufficient amount of data in order to train supervised neural networks to reasonable accuracies. This was established from a variety of result enhancing methods. Hence, a new dataset was created that that contains fast motions, has both mixed motions and motions isolated per axis, and contains an amount of data exceeding what has been produced before. To accomplish this the UrbanMAV dataset was produced using simulated frames. Preliminary tests with this dataset have resulted in smoother, more equal learning curves for the different degrees of motion. For this the methodology of weighted losses was not required, which is a great improvement over the other datasets. However, it is believed that a decrease of overfitting and improvement of accuracies is still possible. Currently, decoupled motions have not yet shown this capability. Thus for future researches it is prescribed that this state-of-the-art dataset as well as the weighted multiple output method are used to improve upon the architectures of neural network based visual odometry.

References

[1] Reem Ashour, Tarek Taha, Fahad Mohamed, Eman Hableel, Yasmeen Abu Kheil, Malak Elsalamouny, Maha Kadadha, Kasturi Rangan, Jorge Dias, Lakmal Seneviratne, et al. Site inspection drone: A solution for inspecting and regulating construction sites. In *2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1–4. IEEE, 2016.

[2] UM Rao Mogili and BBVL Deepak. Review on application of drone systems in precision agriculture. *Procedia computer science*, 133:502–509, 2018.

[3] Paweł Podsiadło, Bartłomiej Bargiel, and Wojciech Moskal. Mountain rescue operations facilitated with drone usage. *High altitude medicine & biology*, 20(2):203–203, 2019.

[4] Dmitri Aleksandrov and Igor Penkov. Energy consumption of mini uav helicopters with different number of rotors. In *11th International Symposium" Topical Problems in the Field of Electrical and Power Engineering*, pages 259–262, 2012.

[5] Hyungpil Moon, Jose Martinez-Carranza, Titus Cieslewski, Matthias Faessler, Davide Falanga, Alessandro Simovic, Davide Scaramuzza, Shuo Li, Michael Ozo, Christophe De Wagter, et al. Challenges and implemented technologies used in au-

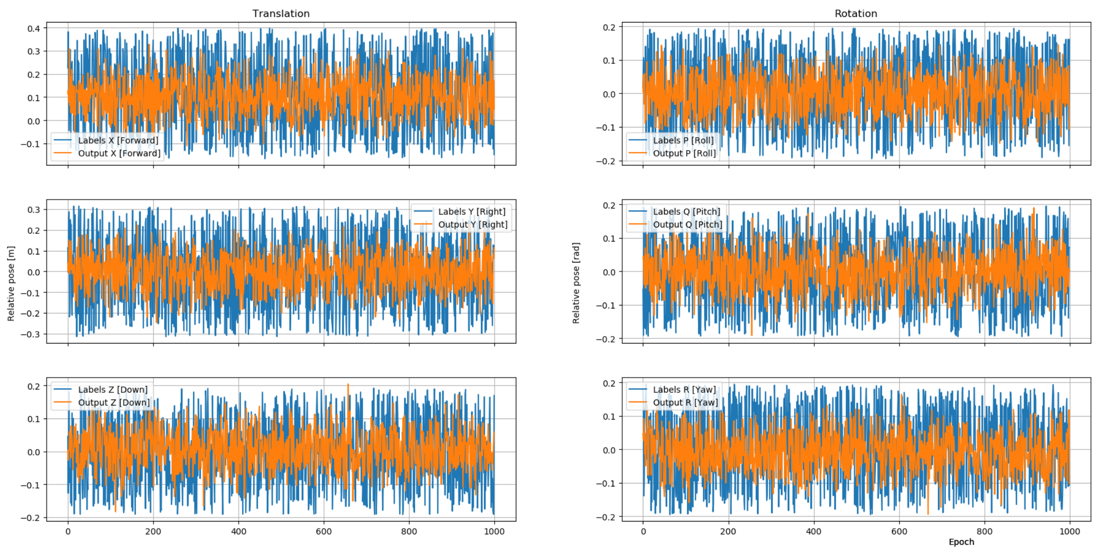

Figure 19: Outputs of neural network trained on the Urban MAV dataset.

tonomous drone racing. *Intelligent Service Robotics*, 12(2):137–148, 2019.

[6] Winter Guerra, Ezra Tal, Varun Murali, Gilhyun Ryou, and Sertac Karaman. Flightgoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality. *arXiv preprint arXiv:1905.11377*, 2019.

[7] Ke Sun, Kartik Mohta, Bernd Pfrommer, Michael Watterson, Sikang Liu, Yash Mulgaonkar, Camillo J Taylor, and Vijay Kumar. Robust stereo visual inertial odometry for fast autonomous flight. *IEEE Robotics and Automation Letters*, 3(2):965–972, 2018.

[8] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 15–22. IEEE, 2014.

[9] Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. pages 2043–2050, 2017.

[10] Kimberly McGuire, Guido De Croon, Christophe De Wagter, Karl Tuyls, and Hilbert Kappen. Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone. *IEEE Robotics and Automation Letters*, 2(2):1070–1076, 2017.

[11] Rui Wang, Stephen M Pizer, and Jan-Michael Frahm. Recurrent neural network for (un-) supervised learning of monocular video visual odometry and depth. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5555–5564, 2019.

[12] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1851–1858, 2017.

[13] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[14] Jeffrey Delmerico, Titus Cieslewski, Henri Rebecq, Matthias Faessler, and Davide Scaramuzza. Are we ready for autonomous drone racing? the uzh-fpv drone racing dataset. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6713–6719. IEEE, 2019.

[15] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35(10):1157–1163, 2016.

[16] Yi Zhang Siyuan Qiao Zihao Xiao Tae Soo Kim Yizhou Wang Alan Yuille Weichao Qiu, Fangwei Zhong. Unrealcv: Virtual worlds for computer

vision. *ACM Multimedia Open Source Software Competition*, 2017.

[17] Oleksandr Semeniuta. Analysis of camera calibration with respect to measurement accuracy. *Procedia Cirp*, 41:765–770, 2016.

[18] Yuzhen Hong, Guoqiang Ren, and Enhai Liu. Non-iterative method for camera calibration. *Optics Express*, 23(18):23992–24003, 2015.

[19] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library.* O'Reilly Media, Inc., 2008.

[20] Georgi Zapryanov and Iva Nikolova. An experimental comparative performance study of demosaicing algorithms on general-purpose gpus. In *Proceedings of the 9th Balkan Conference on Informatics*, pages 1–7, 2019.

[21] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

[22] Xue Ying. An overview of overfitting and its solutions. In *Journal of Physics: Conference Series*, volume 1168, page 022022. IOP Publishing, 2019.

[23] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Appendix A - Results on KITTI

The following pages serve to deliver additional information and graphs on top of that which is depicted in section 5. These sections dive into the usage of dropout and are a showcase of a part of the additional exploration that went into the above described research. Note that this only encompasses a narrow part of the additional work that went into this research.

To prevent or reduce overfitting the method of dropout can be applied. This procedure relies on dropping out a part of the connections between layers in order prevent the neural network from learning false patterns, due to stationary conditions, which may cause higher accuracies on training data compared to validation data. If these dropouts were to occur on the same positions this would lead to stationary conditions, in the form of a smaller architecture, on which overfitting could still apply. Instead, these dropouts occur on randomly selected locations [23].

Dropout of 0.25 means that 0.25 of the connections between the assigned layers are blocked. In the case of this experiment half of these connections are dropped out, which can be managed using functions of the Pytorch library [21]. The training and validation accuracies of the neural network subjected to 0.5 dropout are found in Figure 20. When comparing this to Figure 7 it can be seen that the dropout has an effect on the neural network that is three fold.

1. Forward motion, x-axis translation, still reaches similar accuracy values again while overfitting. However, the training accuracy starts at a lower value and takes a longer period of time in order to reach the preciseness.

2. A notable change is the elimination of overfitting patterns for the translation of the y and z-axis. Additionally and more importantly the same holds for the yaw angle. Unfortunately this is paired with a relatively low roughly equal to the limit of the validation accuracy established without dropout.

3. The accuracy of pitch and roll vary more frequently, although the training and validation accuracy still follow the same pattern. Also, the accuracy of pitch and roll overall is lower.

The first effect can be seen as an indication that high accuracies are still achievable even though half of the connections are severed randomly. Secondly, the removal of the overfitting effect can be seen as a positive and promising result. The third and final effect, although less impactful, shows the instability of the learning process due to the dropout. Which is highly likely the cause of the increased duration mentioned by the first effect.

From the current result two possible methods of improvement were established. Foremost an increase in layers can potentially establish an increase in accuracy. This may then help to acquire fewer errors in the yaw validation. However, considering that an increase in network architecture would lead to an increase in memory size requirement it is coherent with the research goal to look into other potential alterations in the learning process that may improve results.

Another area where changes can be made is the loss function. Currently the network has three translational and three rotational outputs in a single array. This array is then fed to Pytorch's mean squared error loss function, as described by subsection 5.1. However, the above described results show a disparity between the learning patterns of the six outputs. For this reason a trial was

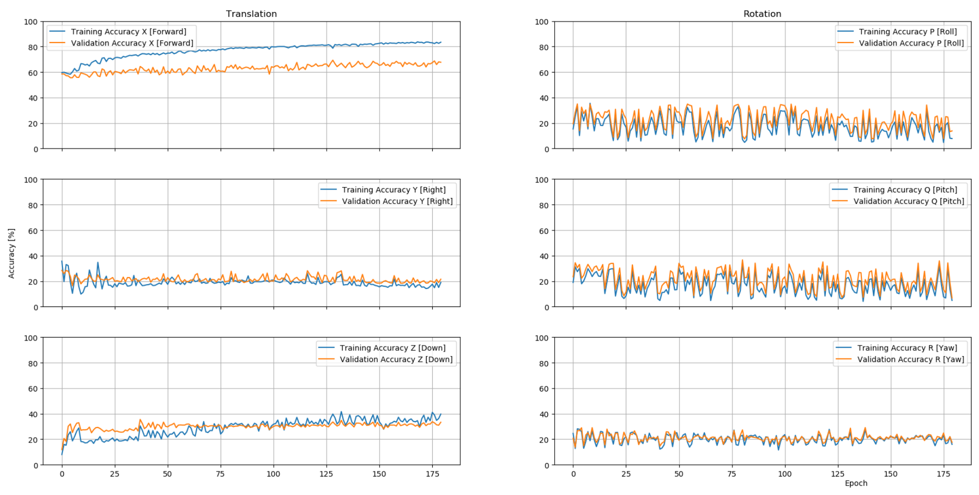

Figure 20: Translational and rotational results on supervised version of the SfmLearner. Trained on sequences 00-04 and 06-10, validated on sequence 05, of the KITTI database. With a dropout of 0.5.

conducted to severe the outputs into different variables, each with their own (mean squared error) loss function.

With the results of the dropout described above and the estimation that addition of layers may improve results a new architecture was set-up. Two additional convolutional layers were added with similar characteristics as layers six and seven in Table 1. The resulting accuracies are depicted by Figure 24.

Due to time constraints the training of this method was shortened in comparison to the test conducted previously, at 125 epochs in comparison to 175. Nonetheless the resultant accuracies are a clear depiction of the achieved outcome. Little to no difference is found between the patterns and accuracies taken on by the accuracies in Figure 24 when compared to Figure 20. Which is a clear indicator that this is not causing improvements and the factor that limits the current outcome may be found in the learning process or other characteristics of the neural network rather than the quantity of layers.

Again, a large difference in pattern is found between the accuracies of the translational and rotational predictions. For that reason the second improvement option opted when using a dropout, focusing on the adoption of another loss function method was adopted.

In order to test the capabilities of this method the above neural network was trained with a half drop out. Its results are depicted in Figure 22. Again, the yaw angle shows a increasing accuracy pattern. It shows an increase of validation accuracy over the results in Figure 9. However, this comes at the cost of not having the learning curve for the pitch and roll angles. Thus it shows the possibility to improve results by splitting the outputs. However, this will most probably require extensive regard to the balancing method.

Another case with drop out value 0.25 was tested resulting in Figure 23. Not only does this show an improved final accuracy for the validation of yaw, at 40%. It also shows a learning curve for the pitch and roll outputs. However, the start of this learning curve is delayed till the point where the yaw angle estimation is almost done with improving. This again clearly indicates the overshadowing of certain outputs over the training of others.

With the purpose of balancing the learning curves between the various outputs, a test was conducted using weight values of one divided by the absolute value of the minimal value of the batch of the respective output. This weight calculation is described by Equation 7, where $*$ is used to indicated any of the six output variables. Note that this intrinsically means that the lower variables get a relatively higher weight value. Which may be required as the differences between the labels and outputs also is intrinsically lower.

$$w_* = \frac{1}{min(\|labels[*]\|)} \tag{7}$$

The results of this test are depicted in Figure 25. Where it can be seen that the desired learning curve for each variable is only available for the translation in the x-axis. This could partly be due to improper balanced weight calculations. Thus the above described equation would have to be adjusted. Furthermore, it is apparent that the overfitting effect on the translational value of the x-axis is far less than during previous tests. However, it reaches far lower accuracies. The divergence of the training and validation learning curves is relatively low after follow a similar trail, with for example the peaks on epoch 23 and 43. Nonetheless, the lacking of a learning curve in the rotational data forms a deprivation. Beside creating more balanced weight calculations, it is estimated that the datasets for drones suffer less from the

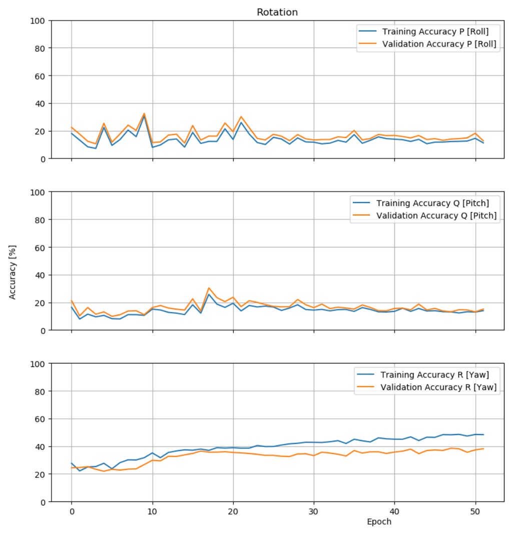

Figure 22: Results for a rotational output only on supervised version of the SfmLearner, with a dropout value of 0.5. Trained on sequences 00-04 and 06-10, validated on sequence 05, of the KITTI database.

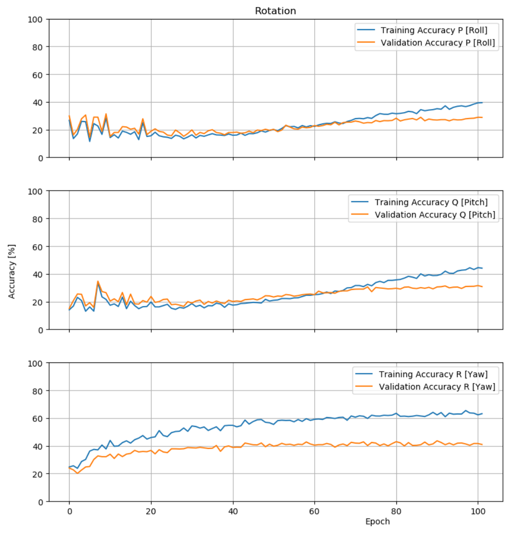

Figure 23: Results for a rotational output only on supervised version of the SfmLearner, with a dropout value of 0.25. Trained on sequences 00-04 and 06-10, validated on sequence 05, of the KITTI database.

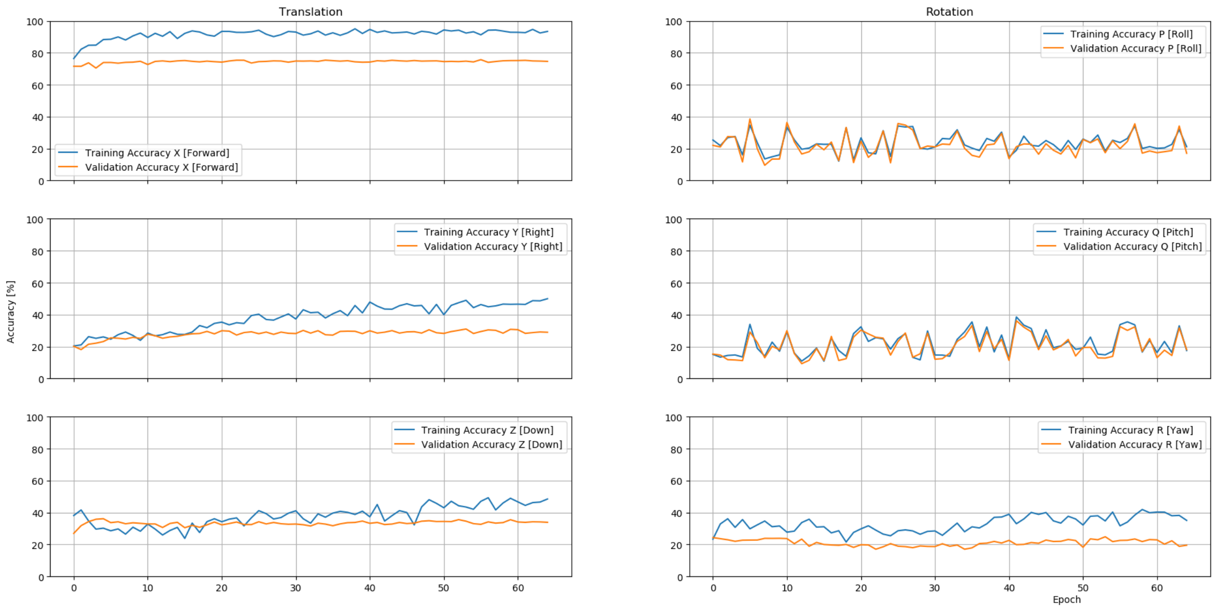

Figure 21: Translational and rotational results on supervised version of the SfmLearner. Trained on sequences 01-10, validated on sequence 00, of the KITTI database.

disparity between outputs, due to their higher usage of the y and z-axis translations as well as the pitch and roll rotations.

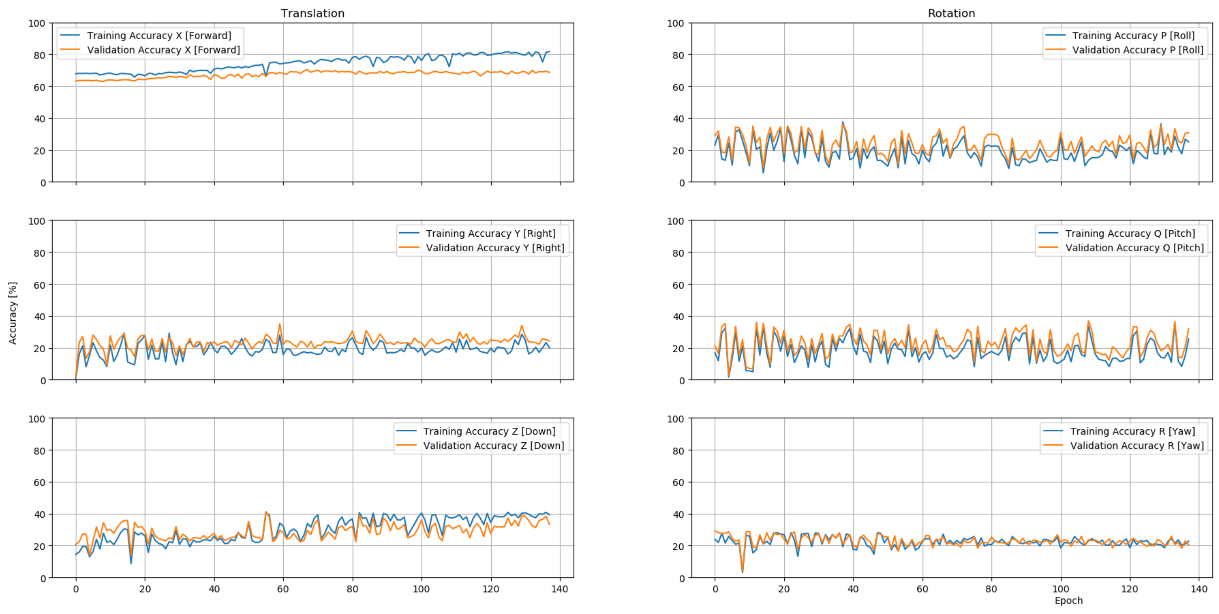

Figure 24: Translational and rotational results on supervised version of the SfmLearner. Trained on sequences 00-04 and 06-10, validated on sequence 05. With a dropout of 0.5, of the KITTI database. And two additional convolutional layers.

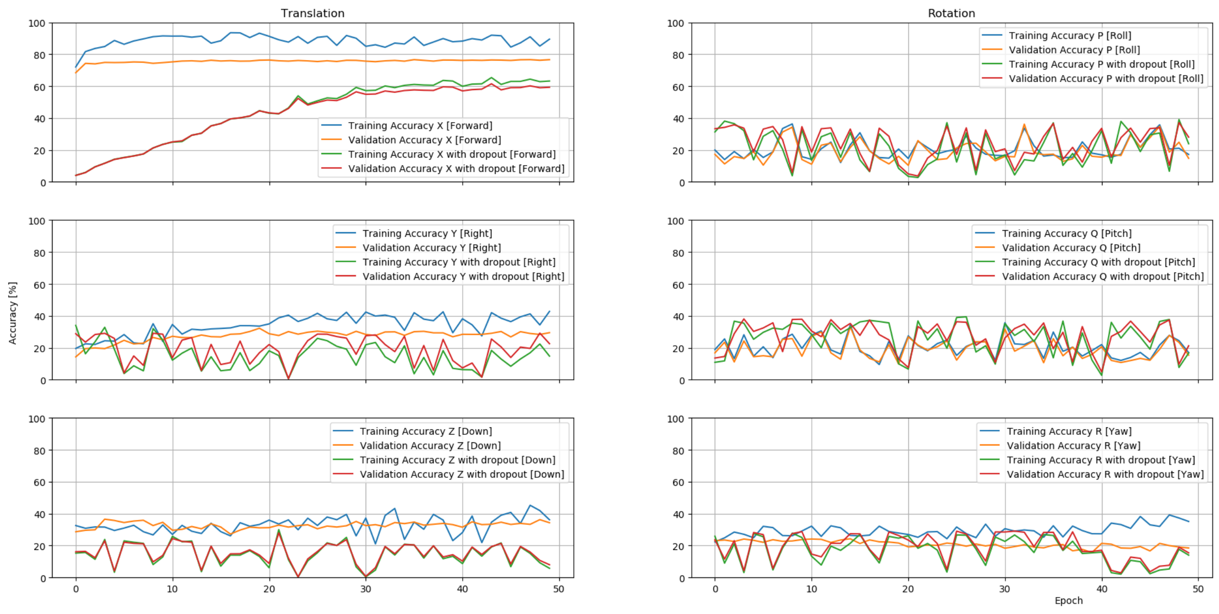

Figure 25: Translational and rotational results on supervised version of the SfmLearner. Trained on sequences 00-04 and 06-10, validated on sequence 05, of the KITTI database. Containing two training moments. First is the regular model, second uses a split output and respective weight of $\|min(label[\cdot])\|$ and a dropout of 0.5.

A Literature Review

This chapter contains the content of the preliminary thesis report. It has been written and graded for the course AE4020. Previously written in article style, it has been converged to the thesis format.

***Abstract* - Drones are continuously becoming more developed and are deployed to aid in challenging tasks. Autonomous drone races endeavor to expand the capabilities of micro air vehicles. This extensive literature study delves into the effectiveness of various visual odometry methods, revealing the competence of neural networks. From the literature studied, it becomes evident that appropriate datasets are essential when training these networks, especially when competing in highly dynamic autonomous drone races. Which raises the questions (1) do efficacious input sequences exist and in turn (2) is it possible to further improve upon the extant datasets. Both questions being answered through additional literature examinations. With this information the ground work is laid for a future thesis research, including a preliminary analysis of neural networks.**

keywords - Autonomous Drone Racing, Visual Odometry, Convolutional Neural Networks

A.1. Introduction

The challenge of MAV is to make them entirely autonomous and able to handle unknown or changing environments, using fast and dynamic movements provided by on-board sensing and calculation. These MAV can then prove useful in e.g. site inspection, agriculture and rescue missions [2–4]. Autonomous drone races (ADR) pose as a passageway that may advance these fast and dynamic MAV under various circumstances. These advancements are achieved by constantly requiring improved pose (relative positioning) estimations and control systems under constraint computational power. In ADRs a course is set out using gates, of which the locations may not be known beforehand, which must be traversed as quick and far as possible. The MAVs are generally lightweight at around 0,5 kg, but Li et al. (2020) proved it can be decreased to 72 g [13]. This calls for efficient coding of position determination methods, as small hardware with low CPU capacity is on board. VO, pose estimation by means of cameras, has shown great potential towards solving this problem [14, 15].

This paper is the result of an extensive literature study on VO methods, summarized by section A.2. With the advantages and drawbacks of readily available systems in mind, it hopes to achieve an answer to the question whether ego-motion estimation using deep neural networks has further potential. And in turn develop increased results during drone races. Considering that training datasets play a fundamental role for the appropriate implementation of neural networks, section A.3 gives an overview of available input datasets for machine learning methods. Furthermore, the conception of contemporary datasets, particularly through simulator footage, is discussed in section A.4. An evaluation on these informative chapters is given by section A.5, deducing the aptitude and potential improvements of neural networks for VO purposes. This aptitude is then displayed in section A.6, containing a baseline research including the preliminary modification process and results, laying the foundation for an anticipated thesis. To conclude, section A.7 summarizes this report and sheds light on other possible future researches.

A.2. Existing VO Methods

The following section will illustrate the current state-of-the-art VO methods. This knowledge will aid in establishing possible improvements and the position of future research.

A.2.1. Monocular and Binocular

The introduction by Wang et al. (2018) discusses a category divider for VO methods, dependant on the amount of cameras; monocular and binocular (stereo) [16]. They state that monocular VO structures prove to be cheaper and lighter, due to requiring fewer components. Monocular VO makes use of a single camera while stereo methods use two cameras, resulting in a higher accuracy for stereo approaches. According to Persson et al. (2015) this superiority in precision is due to the inherently easier problem for stereo VO. From which it can be deduced that monocular VO systems are generally more sophisticated by itself as they have to deal with larger problems [17]. Furthermore, stereo VO converges to monocular VO results as the distance between the cameras decreases relatively to the depth at which they measure [16]. Both methods are able to estimate trajectories close to that of the ground truth [18–20].

Due to the greater amount of hardware required for stereo solutions, less room for other on-board hardware such as batteries and CPU (central processing unit) is available, possibly limiting performance. It is for this reason that monocular solutions, although having a more arduous problem statement, are favored. It is believed that the intrinsic additional effort will lead to more substantial developments. However, analysis on literature containing stereo approaches are nevertheless encompassed in the following subsections, as they may likewise provide beneficial information for monocular techniques.

A.2.2. Geometric VO

Currently most high performing VO methods are based on geometric constraints, according to Wang et al. (2017) [11]. A number of these methods were examined by Poddar et al. (2018), who showed they can be separated into different groups [21]. The feature based geometric approach brings to light geometric features such as salient points, edges and blobs from images in order to derive position estimations by matching these features with previously attained frames. Secondly, appearance based VO relies on improving the photo-metric error, thus matching entire frames. By using the entire image, as opposed to a select few features, decreased aliasing errors for comparable appearing features can be achieved. These appearance based methods can again be branched into two main categories; optical flow based and region based matching.

Often the computer vision method by Geiger et al. (2011) is referred to as VISO2-M [22]. It makes use of a blob and corner detectors for tracking. With this it introduces a sparse feature matcher in order to construct 3D maps. Its results are compared to the method by Kitt et al (2010), showing a reduction in computational time with approximately equivalent accuracy [19].

Roberts et al. (2008) mapped optical flow, the inter-frame motion, in order to derive a velocity vector [23]. Due to the large vector describing the sparse optical flow, making it hard to generalize, and the varying dimensionality due to the Harris corner detection, K-Nearest-Neighbours (KNN) methods were required [24]. The KNN method selects a fixed amount of best input features. The platform which was used to record the data is described as car-like, as it can only yaw and translate forward. Furthermore, it moves at a maximum velocity of 1.5 m/s, which is relatively slow in comparison to MAVs [25, 26].

These geometric methods greatly rely on the detection and tracking of features. This dependency gives rise to the disadvantage of false correlations, due to erroneous thresholds [27]. Therefore, these geometric VO methods are only basal solutions that do not offer enough potential of advancement in the desired field.

A.2.3. Visual Inertial Odometry

In order to improve results of VO methods Inertial measurement units (IMU) can be applied, resulting in Visual inertial odometry (VIO). Kalman filters and sliding window estimators play a key role in merging the visual and inertial data.

A benchmark research by Delmerico and Scaramuzza (2018) evaluates several of these methods [28]. One of these is the Extended Kalman Filter (EKF) for non-linear problem statements regarding vision aided inertial navigation is introduced by Mourikis and Roumeliotis (2007) [29]. The method used is also referred to as Multi-State Constraint Kalman Filter (MSCKF). Sun et al. (2018) enhanced this method to a Stereo MSCKF [26]. Who then proved that S-MSCKF is able to withstand high velocities, aggressive movements and combinations of indoor and outdoor flights. However, at high speeds the method is proven to be outperformed by VINS Mono in RMSE value [30]. This does come at the cost of a higher CPU load for VINS Mono. For the creation of the dataset, Sun et al. (2018) made us of the Intel NUC6i7KYK. This computer kit weighs around 1.5 kg, which is more than three times the weight of the entire Parrot Bebop 1 used in the IROS 2017 ADRs, with higher CPU powers as result [7]. Thereby, similar calculations may not be possible in ADR situations.

The previously mentioned VINS Mono provides a sliding window non-linear optimization for tightly-coupled monocular VIO [30]. Camera-IMU calibration and IMU bias adjustment are also provided, using DBoW2 as loop-closure detection algorithm [31].

The integration of IMU data improve accuracy, by directly measuring linear acceleration vectors conceived from non-gravitational external forces. Furthermore, the additional available data helps in low textured and high speed scenarios such as ADRs [32]. However, VIO methods also have drawbacks e.g. the additional computational power usage, which leaves less computational power for other on-board calculations [33]. Under critical conditions that are introduced during ADRs, this computational power can become a highly limiting factor.

A.2.4. SLAM

Another improvement for VO methods is Simultaneous Localisation and Mapping (SLAM), which cultivates a pose graph out of the data that is gathered. Its main function is to reduce errors of other methods by adjusting pose estimations for drift [34].

A highlight in the progress of SLAM approaches is the ORB-SLAM method [35]. Its name is derived from the usage of ORB-features, which are binary descriptors, making it less susceptible to noise [36]. By using identical ORB-features for tracking, mapping and the closing of loops ORB-SLAM achieves a respectable efficiency. Loop closing is the act of recognizing landmarks and routes, which is an essential last step for SLAM methods in order to mitigate drift errors. Unfortunately, the method has difficulty with recognizing paths when traversing in opposite directions, resulting in unclosed loops [35].

SLAM is more often than not conducted in the front-end of the algorithms, forcing back-end codes to heavily rely on the accuracy of the SLAM system. Sünderhauf and Protzel (2012) proposed to overcome this problem by formulating the code so that the back-end is able to access and adjust the pose graph [37]. They also exploit the sparsity innate to SLAM problems to apply non-linear optimization, resulting in a relatively quicker solution for extensive SLAM problems, as opposed to filter-based systems such as EKF-SLAM and FastSLAM [38].

Although, SLAM proves useful due to cost effective and energy efficient equipment by using the on-board computer, it does hold various disadvantages. First of the partial reliance on planar grounds, similarly to the INAOE team during the IROS 2017 ADR, introduces complications in non-planar regions [7]. Secondly the SLAM algorithms can be computationally intensive resulting in low speeds as to safeguard the drone from crashing [7, 39]. Both may play a crucial role in ADRs and similar scenarios. A lack of potential for the ADR scenarios is therefore adjudge for SLAM methods.

A.2.5. Neural Networks

Pose estimation can be conducted by machine learning algorithms named neural networks. These networks are inspired by the human brain in an attempt to adhere to similar levels of skills. In the case of MAVs these skills would be best weight against those of professional pilots. This subsection hopes to introduce the currently existing VO neural networks, varying from stand-alone architectures to those who integrated the above described methods of IMU.

A multitude of network architectures are benchmarked by Sanket et al. (2020) [40]. The networks included are VanillaNet, ResNet, ShuffleNet, MobileNet and SqueezeNet [41–45]. It concludes that for small networks SqueezeNet is most optimal while for larger networks ResNet is the best choice with respect to accuracy, parameters and number of operations. However, the SqueezeNet was designed in a broad non-systematic process and therefore can still be improved upon [45].

Several methods exist that make use of optical flow (OF), the incremental differences between frames. Muller and Savakis (2017) proposed Flowdometry, a deep Convolutional neural network (CNN) based on such optical flow values [46]. The images containing these changes are the input of a CNN, whose output is the MAVs pose. The architecture is highly based on the FlowNetS network [47]. Results on the KITTI training dataset are compared to VISO2-M, SVR VO and P-CNN VO, which shows an improved translational error over VISO2-M and SVR VO, but a relatively poor rotational error [22, 48–50]. Overall, P-CNN VO proofs to have the lowest inaccuracy in this comparison with 8.96 % translational- and 0.0235 deg/m rotational errors.

Besides P-CNN VO Constante et al. (2015) propose the CNN-1b VO and CNN-4b VO architectures [50]. CNN-1b VO is a conventional CNN that is fed an eight-times down-sampled optical flow image. The CNN-4b VO instead divides the frame into quadrants, which each get down-sampled four-times to be fed to four sets of CNN filters, similar to those of CNN-1b VO. The final layer then uses all information from the four sets to provide a final single output. P-CNN VO merges CNN-1b and CNN-4b for both all-inclusive as well as region specific information. From the comparison made by Constante et al. (2015) it holds that P-CCN VO performs best, which can autonomously find the imperative visual cues as well as selecting the best estimator.

Although the results of Muller and Savakis (2017) and Constante et al. (2015) look promising, these methods would require a supplementary step [46, 50]. A separate neural network or algorithm would have to extract the OF from image sequences, while such methods could also directly extract poses [51]. The hardware of most micro air vehicles is likely still able to withstand the additional computations required, however fewer computations may leave room for other algorithms.

Optionally, some methods make use of Recurrent neural networks (RNN), which helps to stabilize the sequential output, by transferring information on previous frames into the pose computation of the current frame. As Wang et al. (2017) show with their DeepVO method [11]. However, Wang et al. (2017) also uncover a problem when the velocities of the validation data are higher than those during training. When validating the system experiences large drift errors. Therefore, DeepVO is not considered as a stand-in method for geometric VO methods [11]. Other approaches may also implement various RNN. The most commonly seen RNN is Long short-term memory (LSTM) [52]. LSTM's main focus is to keep error values constant by memorizing past values, so that errors do not flare up nor die out, thus overcoming fitting problems illustrated in Figure A.1[1].

[1]Overfitting and underfitting, Educative, `https://www.educative.io/edpresso/overfitting-and-underfitting`, (21 September 2020)

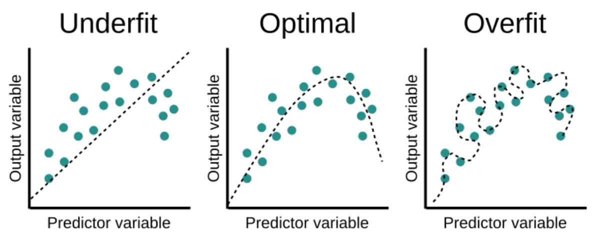

Figure A.1: Over- and under-fitting demonstration.

DeepVIO is a recent work that combines the previously discussed optical flow and IMU data to provide full trajectory estimates. It explores self-supervised CNN-Flow for monocular VIO [53]. Training is conducted using stereo images from which the 3D optical flow and 6-DOF pose form the constraints for the supervised 2D optical flow network. This network includes a LSTM approach for IMU data preintegration in order to increase the pose estimation accuracy. Fully Connected (FC) Fusion networks then fuse the information together as can be seen in Figure A.2. Although the application of 3D optical flow requires disadvantageous additional computations, the pioneering self-supervised learning shows great results when practiced with the KITTI and EuRoC datasets. On the KITTI database it shows vast improvements over ORB-SLAM-M and VINS [30, 35]. However, at some sequences VIOLearner is superior with regard to translational or rotational errors [54]. Furthermore, using the EuRoC datasets the DeepVIO is outperformed by ORB-SLAM and VINS. Nevertheless, considering DeepVIO results surpasses DeepVO on the EuRoC data, this method shows that beneficial IMU data effects also hold for neural network based methods [11].

Some methods Loquercio et al. (2019) deploy a CNN to traverse static and dynamic closed-loop drone racing tracks [55]. Its training on purely simulated environments is what makes this method distinct. By creating a variety of gates, illuminations and environments, referred to as domain randomization, the system becomes inherently more robust for changes when being deployed. Another characteristic of the method by Loquercio et al. (2019) is the determination of required waypoint coordinates in the UAVs body-frame. This rids the system of drift and enables dynamic courses. These waypoints are determined based on the visual position of the artificial gates, ensuring that knowledge of the global path is not required. The system is tested both in simulation as well as in real-life with both static and dynamic gates. The simulation test is performed against the VIO by Loianno et al. (2016), which seems to be a mismatch considering it has a downwards facing camera and its focus is on high speed and aggressive flight rather than traversing a course [56]. The real-life tests are conducted against two pilots with intermediate and professional experience respectively. In respect to the simulated test it is evident that lower speeds (3 m/s) are achieved by the system in real-life. In comparison the pilots are able to traverse the course with twice the velocity. Nonetheless, the algorithm has a higher chance of successfully completing the course and on that account is an indicator of the abilities CNN holds in conjunction with synthetic data.

Both a pose CNN, as well as a depth CNN, are utilized by Zhou et al. (2017) [12]. The method differentiates from others by applying an unsupervised learning approach, meaning that the input data is unlabelled and thus no ground truth is required when training. By synthesizing new frames the data from the pose CNN and depth CNN can be trained. However, as the future data is synthesized, predictions may go wrong causing errors in especially the rotational vectors. Also, unsupervised methods require more data and training time in comparison to supervised methods. Consider Sandaruwans master thesis in which a similar pose network is implemented using the EuRoC database with ground truth as labels. While the unsupervised method requires 150,000 iterations the supervised method converges after around 50 iterations [12, 57]. In the discussion of Sandaruwans thesis report he informs the reader about the fact that, as EuRoC consistently holds rotational or translational movements, it is hard to isolate certain movements. It is discussed that by doing so in future researches, a further understanding of VO based neural networks can be achieved, and consequently improved results may be obtained.

Similarly, an unsupervised method by Wang et al. (2019) is introduced which resorts to a depth estimation network, whose output is the input of the pose estimation network [58]. This network is visualised in Figure A.3, from which it can be concluded that the network does not only hold convolutional layers but also LSTM layers, a type of RNN. However, LSTM requires more memory in comparison to other RNN [59]. Which not only causes a longer training time but also may pose a memory shortage threat when used on small scale computers such as on a drone.

Another memory based neural network is introduced by Xue et al. (2019). They developed an initial RNN, using convolutional LSTM, as tracking architecture. After which memory-networks and refining-networks provide additional adaptive potential [60]. The convolutional LSTM (ConvLSTM) allows for improved spatial formulation as a result of which the recollection of knowledge is expanded. The memory-network is the result of an adaptive selection strategy, after which the selected data is stored for an extended period of time versus LSTM, creating similar effects as SLAM systems. This memory is afterwards used in the refining phase, to successfully achieve increased accuracy in comparison numerous state-of-the-art neural network VO methods. However, these results were achieved on a NVIDIA 1080Ti GPU, being a CUDA-based gaming GPU these do not represent the possible outcomes for drone computers [61]. Instead it is likely that the entire architecture, including the tremendous amount of memory, would put to much of a strain on the on-board processing power. This can possibly lead to unfavourable scenarios during competitions.

Instead of having a single six channel output for translation and rotation vectors, UnDeepVO splits up the layers prior to the output layer [62]. By doing so it creates two parallel stacks of fully-connected layers, each with corresponding three output channels. This decoupled architecture allows distinct weights to be developed, for rotational and translational vector determination, with improved performance as a result. Furthermore, learning of the UnDeepVO network is done unsupervised gaining the advantage of ground truth independence. Thus, unlabelled datasets can be used making for easier acquiring of training sequences. As previously discussed unsupervised methods require a more extensive training in comparison to supervised neural networks. While training is conducted on stereo image pairs, the final architecture is capable of determining position and depth values from two monocular consecutive frames.

Multifarious neural networks exist that are able to establish the pose values of drones, as depicted in Table A.1. An advantage of these methods is the availability due to low hardware requirements. CNN can be conducted with both monocular and stereo camera set-ups and do not necessarily require inertial measurement units, although these methods do exist. However, momentary correlations can be overlooked by neural networks. The inclusion of LSTM layers, or other recurrent architectures, may solve this problem, but in turn make for an increase in memory usage [58]. Furthermore, the accuracy of a neural network is the result of an extensive training. Protracted unsupervised training may be conducted, but is less than ideal for extensive research on various architectures under a time constraint.

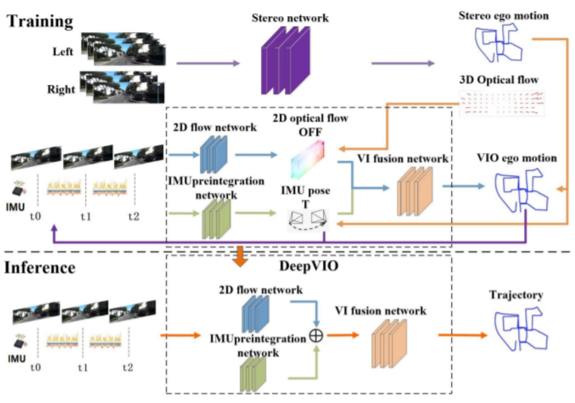

Figure A.2: DeepVIO pipeline [53].

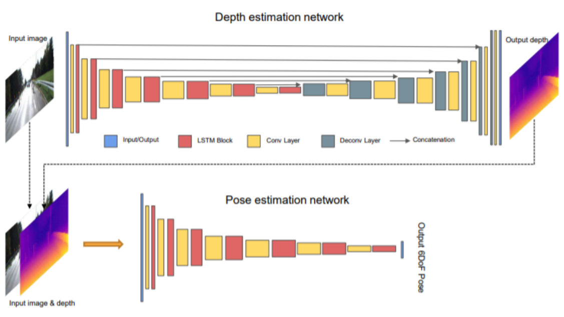

Figure A.3: Unsupervised depth and pose estimation pipeline [58].

Neural Network	Optical Flow	IMU	RNN / LSTM	Training data	Training method
Flowdometry [46]	√			KITTI	Supervised
P-CNN VO [50]	√			KITTI	Unsupervised
DeepVO [11]			√	KITTI	Supervised
DeepVIO [53]	√	√	√	KITTI & EuRoC	Self-supervised
PoseNet [12]				KITTI	Self-supervised

Table A.1: Overview of distinct Neural Networks and their characteristics.

Instead supervised learning can be implemented, which in turn requires appropriate datasets. Incorrect training undoubtedly leads to results of poor quality. It is for this reason that section A.3 further dives into existing datasets for VO purposes.

A.3. Extant Training Datasets

When learning neural networks are applied, proper datasets are a requirement. This section dives into the currently existing datasets that may be used for the training of neural networks for visual odometry purposes.

The TUM monoVO database is introduced by Engel et al. (2016) [18]. It holds 50 sequences spanning 105 minutes, whose frames have been recorded with two different cameras. It must be noted that some of the sequences have fish-eye like effects. Another disadvantage of this dataset is that it has no ground truth. Instead it begins and stops at the same location, which can be used to check for drift to a limited degree. This may not be sufficient for some method verification.

Dataset UAV123 is created by Mueller et al. (2016) and holds over 110,000 frames for UAV based tracking algorithms [63]. It improves upon existing tracking datasets by having a highly-dynamic low-altitude flight and annotations. The data is gathered using two different drones and using a synthetic images. In all three cases ground truth annotations are available, however these are the ground truths of the tracked object thus not improving the usability for the challenge at hand. Furthermore, in most cases the monocular camera is angled downward which obstructs the use of drones with a front facing camera.

Gomez-Ojeda and Gonzalez-Jimenez (2016) conduct an initial experiment using the Tsukuba dataset [20, 64]. Which is a computer generated stereo graphics dataset containing 1800 image pairs, with respective ground truth data, for learning-based approaches. However, this data is created for disparity recognition purposes and as such has a total of 256 levels of disparity. And thus is invalid for the training of visual odometry methods.

Another dataset used by Gomez-Ojeda and Gonzalez-Jimenez (2016) is KITTI, which is collected using a car, see Figure A.4. In total KITTI contains six hours of high-resolution stereo data [48]. However, in case of monocular VO method training the data of the left camera can be extracted. Relative position information between frames is given for these images, which is recorded by an on-board GPS/IMU system. A benchmark by Geiger et al. (2012), analysed various visual odometry methods using KITTI [65]. This benchmark shows that stereo approaches are able to decrease translational errors to around 5%, for monocular approaches this seems to be 10 %. However, it must be noted that only a single monocular VO method, VISO2-M, was investigated. Furthermore, a comparison made to the monocular odometry method UnDeepVo proves inferior results for VISO2-M, making the result less reliable [22, 62]. A drawback of the KITTI database is that automobile movements do not correspond to those of drones. KITTI is unlikely of emulating y-axis and agile rotational movements, while racing drones undergo high velocity 6 DOF motions.

Burri et al. (2016) offer a dataset named EuRoC MAV (2016) [66]. Similarly it holds both stereo images as well as the respective ground truth data. According to Sun et al. (2018) it has highly dynamic rotations and lighting changes [26]. Drawback to this data is the low top speed of 2.3 m/s.

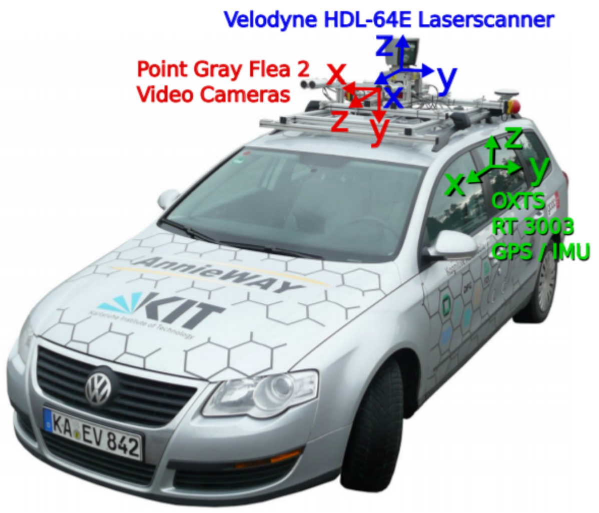
Velodyne HDL-64E Laserscanner
Point Gray Flea 2 Video Cameras
OXTS RT 3003 GPS / IMU
z
y
x
x
z
y
z
y
x

Figure A.4: Recording platform of the KITTI database [48].

A comparable dataset called UZH-FPV was introduced by Delmerico et al. (2019), which is especially developed for racing drones [15]. In total 27 sequences are available, spanning a 10 km flight path. Flights were conducted both inside and outdoors, with maximum velocities of 12.8 m/s and 23.4 m/s respectively. Furthermore, these flights were recorded with a downward- and front-facing camera. As is evident from Table A.2 primarily the front-facing is of interest as this can also be used to detect obstacles. Besides these images, ground truth (GT) has been made available. It must be noted that these images are delivered in grey-scale. Both forward facing data is available as well as under a 45 degree angle in stereo and mono configuration. Unfortunately, the flight paths are mostly forward, thus making the learning of rotational motions less likely.

In turn Sun et al. (2018) made available a database with higher speeds to test the boundaries of the established filter-based stereo VIO algorithm. To prove the robustness a top velocity of 17.5 m/s is achieved [26]. This database sets itself apart from others as it has a transition from outdoors to inside. Downside of the database is that no ground truth is available. Making it only valid for evaluations or unsupervised learning. Furthermore, the latter runs into the problem of not having enough data available, considering that only 700 m of flight are recorded.

Antonini et al. (2018) produced an agile dataset, named Blackbird, in order to allow perception evaluation during aggressive flights [67]. All flight movements are performed by a real-life drone while the respective visuals are created using the FlightGoggles simulator [25]. It covers a total of 860.8 m over 10 hours of flight time, with a maximum speed of 7 m/s. The Blackbird dataset contains five different environments. A disadvantage of the method with which the Blackbird dataset is recorded is that a 100% accurate ground truth value can not be guaranteed, potentially causing errors in training scenarios.

In order to train neural networks some researchers, such as Kaufmann et al. (2018), collect their own datasets [68]. The method from Kaumann et al. (2018) learns from global trajectories that pass through a set of gates, from which it is assumed that their positions are a known factor. In reality however, the final algorithm should be able to handle non-rigid environments. It was observed that by training on multiple rigid environments, with different trajectories, the system can operate accordingly. Likewise, advanced training sets can be created in order to test contemporary neural networks.

Evidently several datasets, see Table A.2, are in existence that provide image sequences for visual odometry systems. Although some are specifically made for MAV purposes, many of the available datasets are inept at the highly dynamic scenarios that characterize automatic drone races. In most

Dataset	Max speed [m/s]	Recording platform	Camera	Length [m]	Ground truth
Kitti [48]	25	Automobile	Stereo	39200	√
EuRoC [66]	2.3	MAV	Stereo	834.7	√
UZH-FPV [15]	12.8/23.4^{i}	MAV	Stereo	3988.3^{ii}	√^{iii}
Blackbird [67]	7	MAV	Virtually stereo	860.8	√^{iv}

Table A.2: Overview of select datasets. ^{i}Indoor/Outdoor. ^{ii}Front facing only. ^{iii}Not available for all sequences. ^{iv}Accuracy is not guaranteed.

cases due to discrepant velocities and rotations, resulting from e.g. dissimilar recording platforms. Lastly, no database contains exclusively rotational or translational data, which may greatly improve the accuracy of said vector determinations.

A.4. Datasets Production

To overcome drawbacks of the aforementioned methods new databases can potentially be created to build upon the strong points with regards to specific VO algorithms. This section dives into the world of database creation by using real-life recording and by employing simulators. Furthermore, the reality-gap dilemma that is innate to synthetic data is discussed, among innovations holding the aptitude of solving this complication.

A.4.1. Physical Data Formation

As with the training datasets considered in section A.3 careful precautions must be taken when creating an improved dataset for research purposes. The definition of aggressive flight are discussed for the UZH-FPV dataset [15]. Their interpretation on swiftness and difficulty of the flown track is the altitude dependent optical flow. With this logic low-altitude low-speed flights bear the same difficulty level as high-altitude high-speed flight. The collation in Figure A.5 shows the optical flow of several conventional datasets. It becomes evident that KITTI and EuRoC have a maximum optical flow of 700 px/s, while UZH-FPV reaches a flow of 1000 px/s. MVSEC likewise reaches flows of around 800-1000 px/s, however merely with a small number of sequences [69].

A.4.2. UAV Simulators

Instead of creating datasets through real-life recordings one may decide to prefer synthetic simulated data. As this requires fewer physical resources and more care can be taken into the creation of the data itself. Contemporary approaches of drone simulation are discussed by Mairaj et al. (2019) [70]. One of the research oriented MAV simulators they discuss is RotorS from ETH Zurich, a UAV simulator

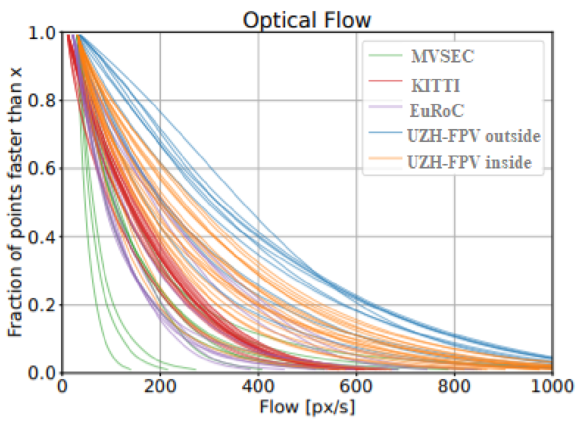

Figure A.5: An optical flow comparison of conventional datasets conducted for UZH-FPV [15].

based in the Gazebo environment [71, 72]. The main focus of this simulator is debugging rather than the creation of databases. RotorS also holds collision avoidance and path planning scenarios but no drone race capabilities. Furthermore, the visual output qualities are at a bare minimum according to Du Montcel et al. (2019), which may negatively influence the results of the training data [73]. Multiple similar performance driven simulators are discussed by Mairaj et al. (2019) that do not fulfil the demand of this research [70].

Unreal Engine is used to augment the Sim4CV simulator, which is specifically build for training and evaluation of computer vision (CV) algorithms [74, 75]. The simulator holds an interface for Matlab, C++ and Python. Which can be a high advantage for communication between the simulator and CV algorithms. However, the main focus of the simulators UAVs dynamics is for tracking algorithm evaluation and as a result is not focused on the high speed 6 DOF manoeuvres of racing drones. Making it inadequately suited for training networks for ADR purposes.

The benchmark by Mueller et al. (2016) produces a simulator which uses the Unreal Engine 4. The benchmark weights over one-hundred different UAV based trackers [63]. Creation of a new high fidelity simulator enables unbiased testing of the trackers. Due to its userbase the simulator is less suitable for the creation of datasets for ADRs. This mainly comes from the lack of motion blur, which is inherent when higher speeds are achieved and depth of field which may play a essential role in pose determination.

Microsoft's Aerial Informatics and Robotics (AIR) has developed AirSim [76]. This simulator is especially established for computer vision and deep learning ends, thus off the bat it shows much potential. Because of its focus it is equipped with quadcopter models. It supports Micro Air Vehicle Link (MAVLink), which is a recently developed communication protocol for autonomous vehicles, making for life-like databases [77]. Airsim makes use of flight controller firmware PX4, ROSFlight, and Hackflight. These take roll, pitch and yaw commands from which the required thrust and torques are computed for the specified quadcopter model. The modular design of AirSim introduces an advantage as elements can be implemented independently. Furthermore, the AirSim algorithm can be executed in Unreal Engine using a plugin bases making for high quality life-like frames [76]. In Mid-Air a low-altitude dataset is created using the AirSim simulator [78]. From this it becomes apparent that AirSim uses RenderTargets in order to render images, which invalidates motion blur. This can cause setbacks when trying to reach high speeds. Simulation framework AirSim Drone Racing Lab builds upon AirSim [79]. It is developed specifically for prototyping and verifying autonomous modules of racing drones, by decreasing the reality-gap with high fidelity. This includes drone gates production, establishment of races including enforcing rules with penalties and the tracking of race progresses. Furthermore, it builds upon Unreal Engine's graphics potential, leading to the addition of the aforementioned lacking motion blur. However, the dynamic computations are tightly coupled with the rendering engine, hindering simulation activity speeds.

Preliminary selections for the AlphaPilot Challenge teams were conducted using the FlightGoggles simulator [6, 25]. Its UAV-in-the-loop dynamics allows for fast and acrobatic flight performance. FlightGoggles is created with an eye on reinforcement learning methods. A modular construction, which holds the Unity game engine at the core, enables users to extract required in- and outputs through the FlightGoggles API [81]. Moving elements can be added to turn a rigid environment into a dynamic and more realistic environment. In addition to this it also enables for adding light chances. Furthermore,

Simulator	Rendering	Dynamics	Sensors	Virtual Reality	API
RotorS [71]	OpenGL	Gazebo	IMU, RGB, Depth		
Sim4CV [75]	Unreal Engine	PhysX	IMU, RGB, Depth, Segmentation		
AirSim [76]	Unreal Engine	PhysX	IMU, RGB		
FlightGoggles [25]	Unity	Flexible	IMU, RGB, Depth, Segmentation	√	
Flightmare [80]	Unity	Flexible	IMU, RGB, Depth, Segmentation	√	√

Table A.3: Overview of state-of-the-art drone simulators

High Definition Render Pipeline (HDPR) is used enabling for motion blur and depth of field, which can be vital in high speed pose estimation scenarios. FlightGoggles contains a single track layout within a abandoned factory environment. This may be disadvantageous when the ADR is not held in a similar environment, as training in a singular scenario can potentially create ad hoc solutions.

Most quadrotor simulators either excel in accuracy or speed. To overcome this problem Song et al. (2020) have developed Flightmare, whose function is to lay the trade-off in the hands of the user by incorporating a flexible physics engine [80]. A key feature in this is the separation of the physics and rendering engines. Flightmare contains three different environments being a warehouse, garage and forest. However, more environments can be acquired via Unity [2]. In order to connect with the simulator an API is made available for Python. Flightmare's characteristics, along with the previously discussed simulators, is depicted in Table A.3.

A.4.3. Jumping the Reality-gap

A draw back to FlightGoggles, emphasised by Sayre-McCor et al. (2018), is the lowered success rate in real-life in comparison to the simulated data [82]. According to Sayre-McCor et al. (2018) reason for this is the noisier visual data when using a real camera and the extra computational load due to image obtainment. Nedevschi (2019) addresses a similar perturbation in virtually created datasets [83]. Although the paper focuses on semantic segmentation, recognition of items in an image, the issues may be of concern to deep learning via synthetic frameworks in general. It speaks of a gap between reality and the virtual world. To close this gap it suggests the use of Virtual-Reality goggles. Which is derived from VR-Goggles [84]. Their take on the reality gap problem is to convert real-life images during the deployment phase to the virtual realm used in the learning phase. This would ensure the algorithm to be familiar with the noise, light and textures found in the incoming data, thus making adaptable to various environments.

The idea of converging images springs from CycleGAN [85]. CycleGAN introduces an algorithm that converts images to the works of painters such as Monet and van Gogh, horse images to zebras and pictures in winter conditions to those belonging to summer. More importantly the $G(X) = Y$ condition is closely revertible so that if $F(Y) = X$ it also holds that $F(G(X)) \approx X$. Similar consistency may prove useful when verification of the algorithms is required.

GeneSIS-RT effectively implements image-to-image rendering to train an algorithm to operate a quadcopter over a course of 60 m with reactive obstacle evasion [86]. In operation the algorithm takes the frames of a monocular camera as input. Training is conducted on data extracted from a simulation. To establish a higher semblance this synthetic data is translated to have a more realistic essence. This manifests a improved result for the ROC-curve and log-loss over exclusively simulated data and approaches results from pure real-world data. It must be taken into account that due to safety restrictions the drone does not reach speeds faster than 3.5 m/s. Moreover, it takes wide trajectories and often stops to avoid collision making for an adverse scenario to ADR.

A more ad hoc solution is the aforementioned AirSim Drone Racing Lab framework. This framework aspires to further decrease the reality-gap of AirSim by improving the dynamical and graphical fidelity [76, 79].

A.5. Deep Neural Networks Potential

From section A.2 it is evident that a variety of VO methods exist. First of subsection A.2.1 discussed the possibility of choosing monocular or stereo camera rigs. As it is desirable that the eventual method is easily adopted by a wide variety of drones, including those with less available resources, it stands to reason that monocular solutions fit best with the target group. Therefore, forthcoming conducted computational researches single out the monocular vision odometry methods.

[2]Unity Asset Store, accessed on 14-09-2020, `https://assetstore.unity.com/`

From here, multiple non-artificial intelligent procedures of pose estimation were examined. Each showing potential in resource rich environments, but lacking in some way. In general, geometric VO methods require exact thresholds, making them rather ad hoc solutions, which in the case of this research is less desirable. VIO and SLAM methods are designed to further improve accuracy results of pose estimation methods. As such they require an increased amount of computational power and VIO approaches need additional hardware space. Both assets are not abundantly available on micro air vehicles and must thus be approached with care.

Fortunately, neural networks have an aptitude for learning the solution to pose estimation problems, while requiring few on-board services. Although, IMU systems and SLAM algorithms may still provide aid, deep neural networks have been shown to be able to estimate relative poses, with only computational power and a monocular camera. It is for this reason that a baseline test for a prospective thesis research will be set-up in section A.6. To further improve the results during the thesis research, developing upon the existing training databases will be considered. To do this, the Flightmare simulator will be deployed [80]. Flightmare provides a flexible drone focused dynamic system, and high quality graphics modelling. Making the combination ideal for developing training data for deep neural network related visual odometry methods. Furthermore, it has the advantage of being able to acquire additional environments, potentially making the trained network less prone to changes when executed. In order to further improve upon the existing datasets, additional exclusively translational and rotational sequences can be created [57]. It is important to achieve compatible speeds during ADRs, while maintaining a consistent pose estimation. It is for this reason that velocities from 3.5 km/hr to 12.5 km/hr need to be acquired in the synthetic dataset. These velocity limits are estimated from the AlphaPilot teams speeds [25]. In order to test the dynamics against other datasets, the optical flow values can be benchmarked, as was done for the UZH-FPV database [15].

A.6. Preliminary Tests and Results

In order to create a substantial foundation for an ensuing thesis research this section presents the set-up and outcome of a baseline-study.

A.6.1. Selection of the Baseline model

As a starting point for the VO neural network described in section A.5, an elemental method was tested. The pose network from Zhou et al. (2017) functions as a guide for this initial research as it holds few exclusive characteristics as seen in Table A.1 [12]. For this baseline research an adapted version of PoseNet is applied in Python using Pytorch. The KITTI database is used by this baseline to train the network [12, 48]. The recording platform, as seen in Figure A.4, contains an IMU and GPS that have recorded ground truth values for odometry purposes. These ground truths are provided in a projection matrix format. Originally the pose network from Zhou et al. (2017) is not trained using this ground truth and instead makes use of synthesized views of future intervals which yields a self-supervised network [12]. However,for the baseline research a supervised learning method, using the ground truth, is implemented. This forms the baseline of the preliminary tests and the start of the research at hand. Convergence to a solution, capable of achieving same digit and sign solutions, would form a secure groundwork for a foreseen thesis. While creating the baseline model, the PoseNet structure of Zhou et al. (2017), implemented using TensorFlow, and an adapted version named PoseNet2 by Sandaruwans, implemented in Pytporch, were used for the verification of the network [12, 57].

A.6.2. Validation results

Validation of the baseline model was conducted using the KITTI database. The desired sign and digit requirement were achieved at roughly 80 to 90 % of the estimates during training of this method. It proved to have an average accuracy of up to 95 % for translational vectors and 40 % for rotational values while training, as can be seen in Figure A.6 and Figure A.7. The values depicted in these figures are estimated using Equation A.1, where λ is the measured accuracy and $\sigma_{out}, \sigma_{lab}$ are the absolute values of the output and labels respectively. Considering that these formulae do not uphold in case the

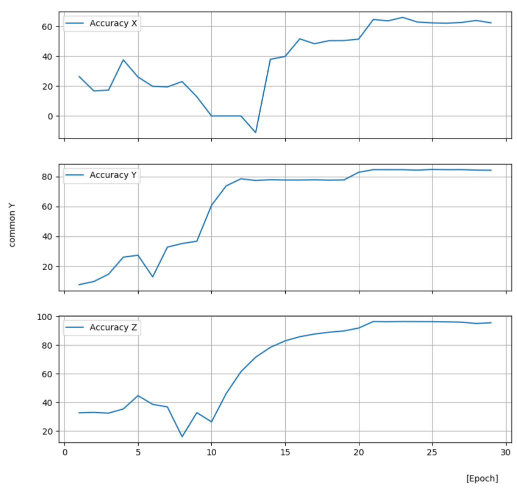

Figure A.6: Translational output of concatenated input.

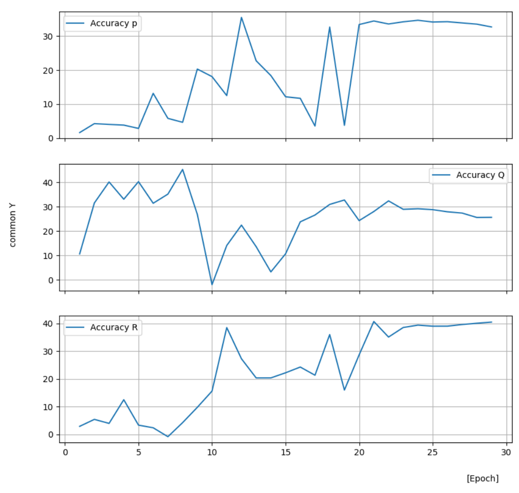

Figure A.7: Rotational output of concatenated input.

sign of the estimations are incorrect the accuracy is set to 0 for these situations. Unfortunately translational values in the X direction seem to deviate from an anticipated pattern. Reason for this could be over-fitting during the first ten epochs, which are conducted using sequence 3. Furthermore, accuracy values for the rotational vectors will require improvement. Possible solutions to these problem are discussed in the following subsection. Nonetheless, the results from this validation process make a strong argument for the potency of the neural networks, as equal accuracies in real-life scenarios should be achievable and improvements are likely [3].

$$\lambda = \begin{cases} 100 - 100 \cdot (\sigma_{out} - \sigma_{lab})/\sigma_{lab}, & \text{if } \sigma_{out} \geq \sigma_{lab} \\ 100 - 100 \cdot (\sigma_{lab} - \sigma_{out})/\sigma_{lab}, & \text{if } \sigma_{lab} \geq \sigma_{out} \end{cases} \tag{A.1}$$

A.6.3. Possible Future Improvements

When transactions between epochs took place the estimated values deviated to a greater extend from the ground truth. Reason for this could be the possible sudden changes in vector values between the end and beginning of a single sequence. Not being able to withstand such changes is an indicator of over- or under-fitting. Which in turn may be caused by oversights in the architecture, such as off target learning rate and weight decay values. The learning rate value determines how swift the network will learn and therefore also how fast convergence takes place. Mismanaged convergence speeds may be the cause of fitting errors. To counterbalance swift convergence weight decay values ensure that the weights are subjected to constant change so that over-fitting is less likely.

Some of the current validations took place by repeating single sequences, which may induce bias to certain weight values and therefore causing the network again to over-fit. This bias was already reduced by switching between sequences while training. It was then established that sequence 1 has a higher potency of inducing a bias, which may be cause by having a lack of divergent characteristics. To further improve learning situations, it is advised that during the thesis research the input order is randomized. It is expected that by doing so the network is less likely to converge to a solution that will only cover the most frequently and consecutively present values.

[3]Private communication with TU Delft Ph.D. student Y. Xu, `http://mavlab.tudelft.nl/people/`

Another potential solution to the fitting problem may be the application of LSTM layers, whose function is to memorize past situations. It is advised to examine the effects of RNN during the thesis further [58]. Furthermore, improvements of the accuracy values, especially in regards to the rotational vectors, may be achieved by splitting translational and rotational computations in the architecture, as is done by Li et al. (2018) [62].

A.7. Conclusion

Considering that the goal of ADRs is the advancement of MAV usage, monocular vision for pose estimation makes for a more widely applicable approach. The cheaper costs, reduced weights and space in comparison to stereo vision systems ease the accessibility, although intrinsically making for a more tenacious problem statement [17]. If this predication is overcome the results should be well worth the effort.

Various sub-optimal VO methods were examined. Geometric VO proved to be too dependant on threshold values. VIO shortcomings showed in weight and energy usage which may be scarce on micro air vehicles. Despite not sharing these disadvantages, SLAM does require increased computational capacity which may pose as a liability in critical scenarios inherent to ADR.

From further literature research it was evident that neural networks prove as an useful solution to the demanding monocular visual odometry problem. Neural networks were shown to provide accurate pose estimations, either single-handedly or by collaborating with IMU data and SLAM methods. Furthermore, LSTM layers were successfully implemented to increase accuracy values. In comparison to the existing methods, progress can be achieved by developing new training data with a primary focus on drone racing.

Where many of these extant methods are trained using the automobile based KITTI datasets, it is believed that using a drone based dataset will lead to improved results. Existing drone based datasets however, lack in various fashions, i.e. being too slow or lacking the separation of translational and rotational movements. Thus the creation of an improved training set is required. Various simulators are capable of creating such drone based synthetic data. After an extensive study it became indisputable that Flightmare is best equipped, due to its flexible dynamics allowing for the simulation of arduous scenarios [80].

A baseline study proved the underlying competence of PoseNet, an elementary pose estimating neural network [12]. It is believed that by advancing the current baseline's architecture, adding characteristics from other methods, but more importantly improving training data by simulating drone race specific sequences high-end results may be achieved. This concept forms the basis for a prospective thesis research, whose preliminary planning can be found in Appendix A.

Auxiliary future researches may look into the ability of odometry using point cloud data instead of camera frames. An effort is already made by DeepPCO, whose deep parallel neural network is responsible of achieving positive performances on KITTI sequences [87]. Point clouds are also adopted by Han et al. (2019) for DeepVIO. Who estimate 3D point clouds, from which 3D geometric labels are extracted for a supervised network [53]. Furthermore, Flightmare offers the ability of synthesizing point cloud data for training purposes [80].

B Discussion

After the extensive literature review found in Appendix A a plan was set-up to adopt drone datasets for a neural network based visual odometry instead of the more frequently used automobile based dataset named KITTI. As the visual odometry method would be applied to drones it seemed more suitable to work with a dataset that contains movements that depict those of a drone. Research with the UZH and EuRoC datasets proved that only minimal results were possible with an unfortunate amount of overfitting. Causing the neural network to work well for the data it had trained on but being unable to extent to validation data.

Nonetheless, a suitable new method was developed that enabled a more elegant way of training the neural network. By dividing the loss into multiple functions, each assigned to a single output. This enables a weighted loss function with a weight value tuned to the desired output for a smoothed learning curve. But more importantly, it enables a more appropriately shared distribution of learning initiative between the various translational and rotational variables. It is believed that this can be a powerful tool that will aid future researches. As it develops more clarity in the learning process of the neural network for the user, decreases time due to the simultaneously increasing accuracies and allows for a more divided desired output values.

Furthermore, it was discovered that the already existing datasets only have limited data. Not only do they contain a small amount of image pairs. The distribution of values for the translational and rotational speeds are highly skewed. Although EuRoC contains a vast amount of data, this data is highly centered around bare minimum velocities. For the UZH-FPV dataset this misrepresentation of values is less high than EuRoC but still too much to properly train a neural network. In order to overcome this problem an advanced and equally distributed dataset was created. By creating random image pairs the distribution of relative poses was entirely under control, and thus shows no bias as with the extant datasets.

This dataset was created using a simulated urban environment, resulting in the Urban MAV naming of the dataset. By optimally incorporating various Python functions a process was created that could create a random dataset within the desired constraints. In total around 25,000 random image pairs were established, subdivided in 25 sequences. However, it was theorised that additional sequences with decoupled motions, containing only a single variation of motion, would be able to improve the learning process. To test this hypothesis an additional 25,000 image pairs per decoupled motion were created. Making for a dataset containing vastly more usable image pairs in comparison to the already existing databases. Additionally, these decoupled motion are too created in a controlled randomised fashion, in order to guarantee the lack of skewness within the data.

Initial testing proved a noticeable improvement over the previously mentioned dataset. Even without the splitting of the loss methodology a more balanced distribution of the learning curves was apparent. Reason for this probably stems from the fact that the data itself is more equally distributed in values, making the loss values less prone to connecting more to one output. Additionally, as larger relative pose values are more frequently met the dataset could be adjusted so that changes found in angular and translational motions are more in line with each other, thus taking away the need of weighting these loss values. However, the addition of decoupled motions has not yet shown a positive effect. Most likely this is due to the tremendous amount of zero values within the data, making it harder for the neural network to take a hold of a proper learning strategy. It is most likely that further investigation may uncover how to overcome this hurdle, improving the results tremendously.

For future research it would be advised to use the newly created, state-of-the-art dataset in order to dive into the various methods that exist in the improvement of neural network based visual odometry. It is expected that long-short-term-memory may prove highly potent, and can show its true power in combination with the decoupled motion sequences. As assigning weights to the various losses is no longer required in order to train a neural network with the Urban MAV dataset, the splitting of the loss may not play a vital role within researches based around this dataset. Nonetheless, it is believed that splitting and assigning weights to losses can improve the training of neural networks with multiple outputs, both inside as well as outside the scope of visual odometry.

Bibliography

[1] Voir DR Brunstetter and Megan Braun. The implication of drones on the just war tradition. *art. cité*, 2011.

[2] Reem Ashour, Tarek Taha, Fahad Mohamed, Eman Hableel, Yasmeen Abu Kheil, Malak Elsalamouny, Maha Kadadha, Kasturi Rangan, Jorge Dias, Lakmal Seneviratne, et al. Site inspection drone: A solution for inspecting and regulating construction sites. In *2016 IEEE 59th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1–4. IEEE, 2016.

[3] UM Rao Mogili and BBVL Deepak. Review on application of drone systems in precision agriculture. *Procedia computer science*, 133:502–509, 2018.

[4] Paweł Podsiadło, Bartłomiej Bargiel, and Wojciech Moskal. Mountain rescue operations facilitated with drone usage. *High altitude medicine & biology*, 20(2):203–203, 2019.

[5] Dmitri Aleksandrov and Igor Penkov. Energy consumption of mini uav helicopters with different number of rotors. In *11th International Symposium" Topical Problems in the Field of Electrical and Power Engineering*, pages 259–262, 2012.

[6] Philipp Foehn, Dario Brescianini, Elia Kaufmann, Titus Cieslewski, Mathias Gehrig, Manasi Muglikar, and Davide Scaramuzza. Alphapilot: Autonomous drone racing. *arXiv preprint arXiv:2005.12813*, 2020.

[7] Hyungpil Moon, Jose Martinez-Carranza, Titus Cieslewski, Matthias Faessler, Davide Falanga, Alessandro Simovic, Davide Scaramuzza, Shuo Li, Michael Ozo, Christophe De Wagter, et al. Challenges and implemented technologies used in autonomous drone racing. *Intelligent Service Robotics*, 12(2):137–148, 2019.

[8] Elias Mueggler, Henri Rebecq, Guillermo Gallego, Tobi Delbruck, and Davide Scaramuzza. The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam. *The International Journal of Robotics Research*, 36(2):142–149, 2017.

[9] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, pages I–I. Ieee, 2004.

[10] Kishore Reddy Konda and Roland Memisevic. Learning visual odometry with a convolutional network. In *VISAPP (1)*, pages 486–490, 2015.

[11] Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. pages 2043–2050, 2017.

[12] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1851–1858, 2017.

[13] Shuo Li, Erik van der Horst, Philipp Duernay, Christophe De Wagter, and Guido CHE de Croon. Visual model-predictive localization for computationally efficient autonomous racing of a 72-g drone. *Journal of Field Robotics*, 2020.

[14] Christian Forster, Zichao Zhang, Michael Gassner, Manuel Werlberger, and Davide Scaramuzza. Svo: Semidirect visual odometry for monocular and multicamera systems. *IEEE Transactions on Robotics*, 33(2):249–265, 2016.

[15] Jeffrey Delmerico, Titus Cieslewski, Henri Rebecq, Matthias Faessler, and Davide Scaramuzza. Are we ready for autonomous drone racing? the uzh-fpv drone racing dataset. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6713–6719. IEEE, 2019.

[16] Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. End-to-end, sequence-to-sequence probabilistic visual odometry through deep neural networks. *The International Journal of Robotics Research*, 37(4-5):513–542, 2018.

[17] Mikael Persson, Tommaso Piccini, Michael Felsberg, and Rudolf Mester. Robust stereo visual odometry from monocular techniques. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 686–691. IEEE, 2015.

[18] Jakob Engel, Vladyslav Usenko, and Daniel Cremers. A photometrically calibrated benchmark for monocular visual odometry. *arXiv preprint arXiv:1607.02555*, 2016.

[19] Bernd Kitt, Andreas Geiger, and Henning Lategahn. Visual odometry based on stereo image sequences with ransac-based outlier rejection scheme. In *2010 ieee intelligent vehicles symposium*, pages 486–492. IEEE, 2010.

[20] Ruben Gomez-Ojeda and Javier Gonzalez-Jimenez. Robust stereo visual odometry through a probabilistic combination of points and line segments. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2521–2526. IEEE, 2016.

[21] Shashi Poddar, Rahul Kottath, and Vinod Karar. Evolution of visual odometry techniques. *arXiv preprint arXiv:1804.11142*, 2018.

[22] Andreas Geiger, Julius Ziegler, and Christoph Stiller. Stereoscan: Dense 3d reconstruction in real-time. In *2011 IEEE intelligent vehicles symposium (IV)*, pages 963–968. Ieee, 2011.

[23] Richard Roberts, Hai Nguyen, Niyant Krishnamurthi, and Tucker Balch. Memory-based learning for visual odometry. In *2008 IEEE International Conference on Robotics and Automation*, pages 47–52. IEEE, 2008.

[24] Konstantinos G Derpanis. The harris corner detector. *York University*, pages 1–2, 2004.

[25] Winter Guerra, Ezra Tal, Varun Murali, Gilhyun Ryou, and Sertac Karaman. Flightgoggles: Photo-realistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality. *arXiv preprint arXiv:1905.11377*, 2019.

[26] Ke Sun, Kartik Mohta, Bernd Pfrommer, Michael Watterson, Sikang Liu, Yash Mulgaonkar, Camillo J Taylor, and Vijay Kumar. Robust stereo visual inertial odometry for fast autonomous flight. *IEEE Robotics and Automation Letters*, 3(2):965–972, 2018.

[27] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 15–22. IEEE, 2014.

[28] Jeffrey Delmerico and Davide Scaramuzza. A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots. pages 2502–2509, 2018.

[29] Anastasios I Mourikis and Stergios I Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3565–3572. IEEE, 2007.

[30] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[31] Dorian Gálvez-López and Juan D Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, 2012.

[32] David Schubert, Thore Goll, Nikolaus Demmel, Vladyslav Usenko, Jörg Stückler, and Daniel Cremers. The tum vi benchmark for evaluating visual-inertial odometry. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1680–1687. IEEE, 2018.

[33] Xiaochen Qiu, Hai Zhang, Wenxing Fu, Chenxu Zhao, and Yanqiong Jin. Monocular visual-inertial odometry with an unbiased linear system model and robust feature tracking front-end. *Sensors*, 19(8):1941, 2019.

[34] Pablo F Alcantarilla, Luis M Bergasa, and Frank Dellaert. Visual odometry priors for robust ekf-slam. In *2010 IEEE International Conference on Robotics and Automation*, pages 3501–3506. IEEE, 2010.

[35] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[36] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[37] Niko Sünderhauf and Peter Protzel. Towards a robust back-end for pose graph slam. pages 1254–1261, 2012.

[38] JZ Sasiadek, A Monjazeb, and D Necsulescu. Navigation of an autonomous mobile robot using ekf-slam and fastslam. pages 517–522, 2008.

[39] Runze Liu, Jianlei Yang, Yiran Chen, and Weisheng Zhao. eslam: An energy-efficient accelerator for real-time orb-slam on fpga platform. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[40] Nitin J Sanket, Chahat Deep Singh, Cornelia Fermüller, and Yiannis Aloimonos. Prgflow: Benchmarking swap-aware unified deep visual inertial odometry. *arXiv preprint arXiv:2006.06753*, 2020.

[41] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[43] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, pages 116–131, 2018.

[44] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[45] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[46] Peter Muller and Andreas Savakis. Flowdometry: An optical flow and deep learning based approach to visual odometry. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 624–631. IEEE, 2017.

[47] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazırbaş, Vladimir Golkov, Patrick Van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. *arXiv preprint arXiv:1504.06852*, 2015.

[48] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[49] Thomas A Ciarfuglia, Gabriele Costante, Paolo Valigi, and Elisa Ricci. Evaluation of non-geometric methods for visual odometry. *Robotics and Autonomous Systems*, 62(12):1717–1730, 2014.

[50] Gabriele Costante, Michele Mancini, Paolo Valigi, and Thomas A Ciarfuglia. Exploring representation learning with cnns for frame-to-frame ego-motion estimation. *IEEE robotics and automation letters*, 1(1):18–25, 2015.

[51] Yang Wang, Peng Wang, Zhenheng Yang, Chenxu Luo, Yi Yang, and Wei Xu. Unos: Unified unsupervised optical-flow and stereo-depth estimation by watching videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8071–8081, 2019.

[52] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[53] Liming Han, Yimin Lin, Guoguang Du, and Shiguo Lian. Deepvio: Self-supervised deep learning of monocular visual inertial odometry using 3d geometric constraints. *arXiv preprint arXiv:1906.11435*, 2019.

[54] E Jared Shamwell, Kyle Lindgren, Sarah Leung, and William D Nothwang. Unsupervised deep visual-inertial odometry with online error correction for rgb-d imagery. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[55] Antonio Loquercio, Elia Kaufmann, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: From simulation to reality with domain randomization. *IEEE Transactions on Robotics*, 36(1):1–14, 2019.

[56] Giuseppe Loianno, Chris Brunner, Gary McGrath, and Vijay Kumar. Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and imu. *IEEE Robotics and Automation Letters*, 2(2):404–411, 2016.

[57] Bogoda Arachchige Sameera. Frame-to-frame ego-motion estimation for agile drones with convolutional neural network. Master's thesis, Radboud University Nijmegen, 4 2020.

[58] Rui Wang, Stephen M Pizer, and Jan-Michael Frahm. Recurrent neural network for (un-) supervised learning of monocular video visual odometry and depth. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5555–5564, 2019.

[59] Hojjat Salehinejad, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078*, 2017.

[60] Fei Xue, Xin Wang, Shunkai Li, Qiuyuan Wang, Junqiu Wang, and Hongbin Zha. Beyond tracking: Selecting memory and refining poses for deep visual odometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8575–8583, 2019.

[61] Craig Warren, Antonios Giannopoulos, Alan Gray, Iraklis Giannakis, Alan Patterson, Laura Wetter, and Andre Hamrah. A cuda-based gpu engine for gprmax: Open source fdtd electromagnetic simulation software. *Computer Physics Communications*, 237:208–218, 2019.

[62] Ruihao Li, Sen Wang, Zhiqiang Long, and Dongbing Gu. Undeepvo: Monocular visual odometry through unsupervised deep learning. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 7286–7291. IEEE, 2018.

[63] Matthias Mueller, Neil Smith, and Bernard Ghanem. A benchmark and simulator for uav tracking. In *European conference on computer vision*, pages 445–461. Springer, 2016.

[64] Martin Peris, Sara Martull, Atsuto Maki, Yasuhiro Ohkawa, and Kazuhiro Fukui. Towards a simulation driven stereo vision system. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 1038–1042. IEEE, 2012.

[65] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012.

[66] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35(10):1157–1163, 2016.

[67] Amado Antonini, Winter Guerra, Varun Murali, Thomas Sayre-McCord, and Sertac Karaman. The blackbird dataset: A large-scale dataset for uav perception in aggressive flight. In *International Symposium on Experimental Robotics*, pages 130–139. Springer, 2018.

[68] Elia Kaufmann, Antonio Loquercio, Rene Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: Learning agile flight in dynamic environments. *arXiv preprint arXiv:1806.08548*, 2018.

[69] Alex Zihao Zhu, Dinesh Thakur, Tolga Özaslan, Bernd Pfrommer, Vijay Kumar, and Kostas Daniilidis. The multivehicle stereo event camera dataset: An event camera dataset for 3d perception. *IEEE Robotics and Automation Letters*, 3(3):2032–2039, 2018.

[70] Aakif Mairaj, Asif I Baba, and Ahmad Y Javaid. Application specific drone simulators: Recent advances and challenges. *Simulation Modelling Practice and Theory*, 94:100–117, 2019.

[71] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. Rotors—a modular gazebo mav simulator framework. In *Robot Operating System (ROS)*, pages 595–625. Springer, 2016.

[72] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004.

[73] Thibaut Tezenas Du Montcel, Amaury Negre, Jose-Ernesto Gomez-Balderas, and Nicolas Marchand. Boarr: A benchmark for quadrotor obstacle avoidance based on ros and rotors. 2019.

[74] Martin Mittring. The technology behind the unreal engine 4 elemental demo. *part of "Advances in Real-Time Rendering in 3D Graphics and Games," SIGGRAPH*, 2012.

[75] Matthias Müller, Vincent Casser, Jean Lahoud, Neil Smith, and Bernard Ghanem. Sim4cv: A photo-realistic simulator for computer vision applications. *International Journal of Computer Vision*, 126(9):902–919, 2018.

[76] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pages 621–635. Springer, 2018.

[77] Anis Koubaa, Azza Allouch, Maram Alajlan, Yasir Javed, Abdelfettah Belghith, and Mohamed Khalgui. Micro air vehicle link (mavlink) in a nutshell: A survey. *IEEE Access*, 7:87658–87680, 2019.

[78] Michael Fonder and Marc Van Droogenbroeck. Mid-air: A multi-modal dataset for extremely low altitude drone flights. In *Conference on Computer Vision and Pattern Recognition Workshop (CVPRW)*, June 2019.

[79] Ratnesh Madaan, Nicholas Gyde, Sai Vemprala, Matthew Brown, Keiko Nagami, Tim Taubner, Eric Cristofalo, Davide Scaramuzza, Mac Schwager, and Ashish Kapoor. Airsim drone racing lab. *arXiv preprint arXiv:2003.05654*, 2020.

[80] Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio, and Davide Scaramuzza. Flightmare: A flexible quadrotor simulator. *arXiv preprint arXiv:2009.00563*, 2020.

[81] John K Haas. A history of the unity game engine. 2014.

[82] Thomas Sayre-McCord, Winter Guerra, Amado Antonini, Jasper Arneberg, Austin Brown, Guilherme Cavalheiro, Yajun Fang, Alex Gorodetsky, Dave McCoy, Sebastian Quilter, et al. Visual-inertial navigation algorithm development using photorealistic camera simulation in the loop. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2566–2573. IEEE, 2018.

[83] Sergiu Nedevschi Bianca-Cerasela-Zelia Blaga. Semantic segmentation learning for autonomous uavs using simulators and real data, 2019.

[84] Jingwei Zhang, Lei Tai, Peng Yun, Yufeng Xiong, Ming Liu, Joschka Boedecker, and Wolfram Burgard. Vr-goggles for robots: Real-to-sim domain adaptation for visual control. *IEEE Robotics and Automation Letters*, 4(2):1148–1155, 2019.

[85] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.

[86] Gregory J Stein and Nicholas Roy. Genesis-rt: Generating synthetic images for training secondary real-world tasks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7151–7158. IEEE, 2018.

[87] Wei Wang, Muhamad Risqi U Saputra, Peijun Zhao, Pedro Gusmao, Bo Yang, Changhao Chen, Andrew Markham, and Niki Trigoni. Deeppco: End-to-end point cloud odometry through deep parallel neural network. *arXiv preprint arXiv:1910.11088*, 2019.

C
Code Overview

This appendix gives an outline of the most vital codes used within this research. First off, the python code that was used to train neural networks is presented. After which, the implementation of dataset creation is displayed. These introduce only the general outline of the code and are combined with additional lines or functions to serve their final purpose, according to the settings and data used at the time.

Neural network training

The following python code was used to train the neural network. Throughout the thesis this code has been subjected to an uncountable amount of changes. It is for this reason a more generalised, and through that process decommissioned, version of the program is depicted below.

```
import winsound
import torch
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader, Dataset
import accuracycalculation
from PIL import Image
import math as mt
from pandas import DataFrame
import matplotlib.pyplot as plt
from torch.nn.init import xavier_uniform_, zeros_
import torch.nn as nn
import torch.optim as optim
from scipy.spatial.transform import Rotation as R
import numpy as np
import time

tic = time.perf_counter()

dev = torch.device(”cuda”) if torch.cuda.is_available() else torch.device(”cpu”)

#TRAINING PARAMETERS
newmodelnum = 0
num_epochs = 251
learning_rate = 2e-4

#TRAINING DATA PARAMETERS
loadnewdata = True
batch_size_val = 25
shuffle_bool = True
mirror_bool = False
values = [] #the list ’values’ indicates the relevant sequences during training
for i in range(2,26):
    values.append(’Random/Randommotion’+str(i))

#VALIDATION DATA PARAMETERS
loadvaldata = True
validation_link = [’Random2’]  #dataloaderEuRoC_MH1_1_noshuffle_syncGT.pth    dataloaderEuRoC_MH3Med_1_noshuffle_syncGT.pth  dataloaderEuRoC_V11_1_noshuffle_syncGT
trainvalidation_link = [’Random1’]
batch_size_validation = 1
shuffle_bool_validation = False
mirror_val= False

#Strings for saving and loading the pth files
loadstr1,loadstr2 = ’./backupmodelOWN/UrbanMAV_posenet_’ , ’epochs_allmin10traintest13_MultipleOutput_e4_forwardonly.pth’
savestr1,savestr2 = ’./backupmodelOWN/UrbanMAV_posenet_’ , ’epochs_allmin10traintest13_MultipleOutput_e4_forwardonly.pth’


def Write(validation_accuracy,accuracylist,trainvalidation_accuracy,LossList): #Saving accuracy values
    h = open(”UrbanMAVaccuracybackupMultipleOutput.py”,”w”)
    h.write(”””
from numpy import nan
validation_accuracy =”””+str(validation_accuracy)+”””
accuracylist = ”””+str(accuracylist)+”””
trainvalidation_accuracy = ”””+str(trainvalidation_accuracy)+”””
lossvalues = ”””+str(LossList)+”””
    ”””)
    h.close()

def CreateTrainloader(batch_size_val=batch_size_val,shuffle_bool=shuffle_bool,values=values,mirror=mirror_bool):
    # Function for the creation of dataloaders, ensures relative poses are calculated and image pairs are made.
    # Became irrelevant with the Urban MAV
    train_data = []
    transform_train = torchvision.transforms.Compose([torchvision.transforms.ToTensor()]) #torchvision.transforms.Resize((128,416))
    for j in range(len(values)):
        print(values[j])
        if values[j][:6] == ’Random’:
            data_path_gt = ’NieuweDataset/UrbanMAVDataSet/’+values[j]+’/GroundTruths/cameragroundtruth.txt’
            table_labs = []
            f = open(data_path_gt,”r”)
            lines = []
            for line in f:
                line = line.strip()
                line = line.split(” ”)
                lines = []
                for value in line:
                    value = float(value.strip())
                    lines.append(value)

                labs =  [lines[4],lines[5],lines[6],lines[1],lines[2],lines[3]]
                table_labs.append(labs)
            f.close()

            # Load training data
            train_dataset0 = torchvision.datasets.ImageFolder(
                root=’NieuweDataset/UrbanMAVDataSet/’+values[j]+’/Images/Storage0/’,
                transform=transform_train
            )
            train_dataset1 = torchvision.datasets.ImageFolder(
                root=’NieuweDataset/UrbanMAVDataSet/’+values[j]+’/Images/Storage1/’,
                transform=transform_train
            )
            for ii in range(0,len(train_dataset0)):
                train_data.append((np.array(train_dataset0[ii][0][0]),np.array(train_dataset1[ii][0][0]), table_labs[ii]))
        else:
            data_path_gt = ’NieuweDataset/UrbanMAVDataSet/’+values[j]+’/GroundTruths/cameragroundtruth.txt’
            table_labs = []

            f = open(data_path_gt,”r”)
            lines = []
            for line in f:
                line = line.strip()
                line = line.split(" ")
                lines = []
                for value in line:
                    value = float(value.strip())
                    lines.append(value)

                labs =  [lines[4],lines[5],lines[6],lines[1],lines[2],lines[3]]
                table_labs.append(labs)
            f.close()
            # Load training data
            train_dataset = torchvision.datasets.ImageFolder(
                root='NieuweDataset/UrbanMAVDataSet/'+values[j]+'/Images/Storage/',
                transform=transform_train
            )
            for ii in range(1,len(train_dataset)):
                train_data.append((np.array(train_dataset[ii-1][0][0]),np.array(train_dataset[ii][0][0]), table_labs[ii]))
    trainloader = DataLoader(train_data, batch_size = batch_size_val, shuffle=shuffle_bool, drop_last=True, num_workers=0)
    return trainloader

def conv(in_planes, out_planes, kernel_size=3):
    return nn.Sequential(
        nn.Conv2d(in_planes, out_planes, kernel_size=kernel_size, padding=(kernel_size-1)//2, stride=2),
        nn.BatchNorm2d(out_planes),
        nn.ReLU()
        #nn.ReLU()#,
        #nn.Dropout2d(0.7)
    )

class PoseExpNet(nn.Module):
    def __init__(self, nb_ref_imgs=1, output_exp=False):
        super(PoseExpNet, self).__init__()
        self.nb_ref_imgs = nb_ref_imgs
        self.output_exp = output_exp



        conv_planes = [16, 32, 64, 128, 256, 256, 256]
        self.conv1 = conv(1, conv_planes[0], kernel_size=7)
        self.conv2 = conv(conv_planes[0], conv_planes[1], kernel_size=5)
        self.conv3 = conv(conv_planes[1], conv_planes[2])
        self.conv4 = conv(conv_planes[2], conv_planes[3])
        self.conv5 = conv(conv_planes[3], conv_planes[4])
        self.conv6 = conv(conv_planes[4], conv_planes[5])
        self.conv7 = conv(conv_planes[5], conv_planes[6])
        #self.conv8 = conv(conv_planes[5], conv_planes[6])
        #self.conv9 = conv(conv_planes[5], conv_planes[6])
        self.pose_pred = nn.Conv2d(conv_planes[6], 6*self.nb_ref_imgs, kernel_size=1, padding=0)

    def init_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d) or isinstance(m, nn.ConvTranspose2d) or isinstance(m, nn.Linear):
                xavier_uniform_(m.weight.data)
                if m.bias is not None:
                    zeros_(m.bias)

    def forward(self, inputs):
        out_conv1 = self.conv1(inputs)
        out_conv2 = self.conv2(out_conv1)
        out_conv3 = self.conv3(out_conv2)
        out_conv4 = self.conv4(out_conv3)
        out_conv5 = self.conv5(out_conv4)
        out_conv6 = self.conv6(out_conv5)
        out_conv7 = self.conv7(out_conv6)
        pose = self.pose_pred(out_conv7)

        pose = pose.mean(3).mean(2)
        pose = 0.01*pose.view(pose.size(0), self.nb_ref_imgs, 6)


# Load model
model = PoseExpNet()
if newmodelnum != 0:
    model.load_state_dict(torch.load(loadstr1+str(newmodelnum)+loadstr2))
model.to(dev)

criterion1 = nn.MSELoss(reduction='mean') #Creating the loss criterion
criterion2 = nn.MSELoss(reduction='mean')
criterion3 = nn.MSELoss(reduction='mean')
criterion4 = nn.MSELoss(reduction='mean')
criterion5 = nn.MSELoss(reduction='mean')
criterion6 = nn.MSELoss(reduction='mean')

optimizer = optim.Adam(model.parameters(), lr=learning_rate, betas = [0.9,0.999]) #Loading the optimizer Adam

print("start gathering data...")

validation_accuracy = [[],[],[],[],[],[]]
trainvalidation_accuracy = [[],[],[],[],[],[]]
running_accuracy = [[],[],[],[],[],[]]
value_count = -1

#trainloader = torch.load('dataloaderEuRoC.pth') #Additional method of saving data before running for repetitive usage

if loadnewdata:
    trainloader = CreateTrainloader()

if loadvaldata:
    trainloadervalidation = CreateTrainloader(batch_size_val=batch_size_validation, shuffle_bool = shuffle_bool_validation, values = validation_link, mirror = mirror_val)
    trainloadertrainval = CreateTrainloader(batch_size_val=batch_size_validation, shuffle_bool = shuffle_bool_validation, values = trainvalidation_link, mirror = mirror_val)

toc = time.perf_counter()
seconds = str(round((toc - tic)%60))
minutes = str(round((toc - tic)/60%60))
hours = str(round((toc - tic)/60/60))
print('gathering completed in '+ hours + ' : ' + minutes +' : '+ seconds)
print("start training")
LossList= []
contflag = True
epoch =0
while epoch <= num_epochs:  # loop over the dataset multiple times
```

```
    running_loss = 0.0
    running_accuracy2 = [[],[],[],[],[],[]]
    running_accuracy3 = [[],[],[],[],[],[]]

    for i, (input1,input2,labels) in enumerate(trainloader, 0):
        inputs = torch.cat((input1,input2),1).unsqueeze(1)
        inputs = inputs.to(dev)
        optimizer.zero_grad()

        labels = torch.transpose(torch.as_tensor([list(labels[0]),list(labels[1]),list(labels[2]),list(labels[3]),list(labels[4]),list(labels[5])]),0,1)
        outputs = model(inputs).squeeze()
        if np.isnan(outputs[0].detach()[0].cpu().numpy()):
            model = PoseExpNet()
            model.load_state_dict(torch.load(savestr1+str(newmodelnum+epoch)+savestr2))
            model.to(dev)
            contflag = False
            break
        else:
            loss1 = criterion1(outputs[:,0], labels.float().to(dev)[:,0])
            loss2 = criterion2(outputs[:,1], labels.float().to(dev)[:,1])
            loss3 = criterion3(outputs[:,2], labels.float().to(dev)[:,2])
            loss4 = criterion4(outputs[:,3], labels.float().to(dev)[:,3])
            loss5 = criterion5(outputs[:,4], labels.float().to(dev)[:,4])
            loss6 = criterion6(outputs[:,5], labels.float().to(dev)[:,5])

            loss = loss1+loss2+loss3+loss4+loss5+loss6

            loss.backward()
            optimizer.step()

            LossList.append(loss.item())

            # Accuracies
            for JJ in range(batch_size_val):
                acc_outp = [’%.5f’ % elem.cpu() for elem in list(outputs[JJ].detach())]
                acc_labs = [’%.5f’ % elem for elem in list(labels[JJ].detach().numpy())]
                for j in range(6):
                    acc2 = accuracycalculation(acc_outp[j],acc_labs[j])
                    running_accuracy2.append(acc2)

    if contflag:
        for j in range(6):
            running_accuracy[j].append(sum(running_accuracy2[j])/len(running_accuracy2[j]))

        PATH = savestr1+str(newmodelnum+epoch+1)+savestr2
        torch.save(model.state_dict(), PATH)
        epoch += 1

        ###VALIDATION
        model.load_state_dict(torch.load(PATH))
        model.eval()
        validation_loss = 0.0
        validation_accuracy2 = [[],[],[],[],[],[]]
        trainvalidation_loss = 0.0
        trainvalidation_accuracy2 = [[],[],[],[],[],[]]

        for i, (input1,input2,labels) in enumerate(trainloadertrainval, 0): #Maybe used to signify only a limited set of sequences for the training accuracy

            inputs = torch.cat((input1,input2),1).unsqueeze(1)
            inputs = inputs.to(dev)

            labels = torch.transpose(torch.as_tensor([list(labels[0]),list(labels[1]),list(labels[2]),list(labels[3]),list(labels[4]),list(labels[5])]),0,1)
            outputs = model(inputs).squeeze()

            # Accuracies
            for JJ in range(batch_size_val):
                acc_outp = [’%.5f’ % elem.cpu() for elem in list(outputs.detach())]
                acc_labs = [’%.5f’ % elem for elem in list(labels[0].detach().numpy())]
                for j in range(6):
                    acc2 = accuracycalculation(acc_outp[j],acc_labs[j])
                    trainvalidation_accuracy2.append(acc2)

        for j in range(6):
            trainvalidation_accuracy[j].append(sum(trainvalidation_accuracy2[j])/len(trainvalidation_accuracy2[j]))

        for i, (input1,input2,labels) in enumerate(trainloadervalidation, 0): #Validation accuracy determination

            input1 = input1.sub(torch.mean(input1)).div(torch.std(input1))
            input2 = input2.sub(torch.mean(input2)).div(torch.std(input2))

            inputs = torch.cat((input1,input2),1).unsqueeze(1)
            inputs = inputs.to(dev)

            labels = torch.transpose(torch.as_tensor([list(labels[0]),list(labels[1]),list(labels[2]),list(labels[3]),list(labels[4]),list(labels[5])]),0,1)
            outputs = model(inputs).squeeze()

            # Accuracies
            for JJ in range(batch_size_val):
                acc_outp = [’%.5f’ % elem.cpu() for elem in list(outputs.detach())]
                acc_labs = [’%.5f’ % elem for elem in list(labels[0].detach().numpy())]
                for j in range(6):
                    acc2 = accuracycalculation(acc_outp[j],acc_labs[j])
                    validation_accuracy2.append(acc2)

        for j in range(6):
            validation_accuracy[j].append(sum(validation_accuracy2[j])/len(validation_accuracy2[j]))

        model.train()
        Write(validation_accuracy,running_accuracy,trainvalidation_accuracy,LossList)
    contflag = True

print(’Finished Training’)

winsound.MessageBeep(15)
toc = time.perf_counter()
seconds = str(round((toc - tic)%60))
minutes = str(round((toc - tic)/60%60))
hours = str(round((toc - tic)/60/60))
print(’running time ’+ hours + ’ : ’ + minutes +’ : ’+ seconds)
```

Database creation

The python code depicted in this section was used to develop the Urban MAV database. As multiple functions exists that aid the distribution of values over the six available degrees of freedom, only the function that creates the random database is shown. This should give enough indication on the process that went into creating this dataset. This function was called with randomly initialisation variables.

```
import numpy as np
import cv2
from unrealcv import client
import os, sys
import time
import math as mt
from PIL import Image
from navpy import angle2quat
from navpy import quat2dcm
from scipy.spatial.transform import Rotation as R


client.connect()
if not client.isconnected():
    print('not running')

def flyto(sequencename,x,y,z,p,q,r,seqnumb):
    loc = {'x': x, 'y':y, 'z': z};
    rot = {'roll': p, 'pitch':q, 'yaw':r};
    client.request('vset /camera/0/location {x} {y} {z}'.format(**loc))
    client.request('vset /camera/0/rotation {pitch} {yaw} {roll}'.format(**rot))
    client.request('vget /camera/0/lit Database/'+sequencename+'/Storage/Images/Image'+str(seqnumb)+'.png')
    img = Image.open('/UrbanCity/WindowsNoEditor/UrbanCity/Binaries/Win64/Database/'+sequencename+'/Storage/Images/Image'+str(seqnumb)+'.png').convert('L')
    img.save('UrbanMAVDataSet/'+sequencename+'/Images/Storage/Images/Image'+str(seqnumb)+'.png')
    os.remove('/UrbanCity/WindowsNoEditor/UrbanCity/Binaries/Win64/Database/'+sequencename+'/Storage/Images/Image'+str(seqnumb)+'.png')


def randomflyto(sequencename,x,y,z,p,q,r,dx,dy,dz,dp,dq,dr,seqnumb):
    loc = {'x': x, 'y':y, 'z': z};
    rot = {'roll': p, 'pitch':q, 'yaw':r};
    client.request('vset /camera/0/location {x} {y} {z}'.format(**loc))
    client.request('vset /camera/0/rotation {pitch} {yaw} {roll}'.format(**rot))
    client.request('vget /camera/0/lit Database/'+sequencename+'/Storage0/Images/Image'+str(seqnumb)+'.png')
    img = Image.open('/UrbanCity/WindowsNoEditor/UrbanCity/Binaries/Win64/Database/'+sequencename+'/Storage0/Images/Image'+str(seqnumb)+'.png').convert('L')
    img.save('UrbanMAVDataSet/'+sequencename+'/Images/Storage0/Images/Image'+str(seqnumb)+'.png')
    os.remove('/UrbanCity/WindowsNoEditor/UrbanCity/Binaries/Win64/Database/'+sequencename+'/Storage0/Images/Image'+str(seqnumb)+'.png')

    loc = {'x': x+dx, 'y':y+dy, 'z': z+dz};
    rot = {'roll': p+dp, 'pitch':q+dq, 'yaw':r+dr};
    client.request('vset /camera/0/location {x} {y} {z}'.format(**loc))
    client.request('vset /camera/0/rotation {pitch} {yaw} {roll}'.format(**rot))
    client.request('vget /camera/0/lit Database/'+sequencename+'/Storage1/Images/Image'+str(seqnumb)+'.png')
    img2 = Image.open('/UrbanCity/WindowsNoEditor/UrbanCity/Binaries/Win64/Database/'+sequencename+'/Storage1/Images/Image'+str(seqnumb)+'.png').convert('LA')
    img2.save('UrbanMAVDataSet/'+sequencename+'/Images/Storage1/Images/Image'+str(seqnumb)+'.png')
    os.remove('/UrbanCity/WindowsNoEditor/UrbanCity/Binaries/Win64/Database/'+sequencename+'/Storage1/Images/Image'+str(seqnumb)+'.png')

def fly2(x,y,z,p,q,r):
    loc = {'x': x, 'y':y, 'z': z};
    rot = {'roll': p, 'pitch':q, 'yaw':r};
    client.request('vset /camera/0/location {x} {y} {z}'.format(**loc))
    client.request('vset /camera/0/rotation {pitch} {yaw} {roll}'.format(**rot))

def writegtcamera(sequencename,deltalist):
    h = open("UrbanMAVDataSet/"+sequencename+"/GroundTruths/cameragroundtruth.txt","w")

    for delta in range(len(deltalist[0])):
        dx = deltalist[0][delta]
        dy = deltalist[1][delta]
        dz = deltalist[2][delta]
        dp = deltalist[3][delta]
        dq = deltalist[4][delta]
        dr = deltalist[5][delta]
        h.write(str(delta)+" "+str(dx)+" "+str(dy)+" "+str(dz)+" "+str(dp)+" "+str(dq)+" "+str(dr)+"\n")
    h.close()

def writegtglobal(sequencename,globgtlist):
    h = open("UrbanMAVDataSet/"+sequencename+"/GroundTruths/globalgroundtruth.txt","w")

    for delta in range(len(globgtlist[0])):
        x = globgtlist[0][delta]
        y = globgtlist[1][delta]
        z = globgtlist[2][delta]
        p = globgtlist[3][delta]
        q = globgtlist[4][delta]
        r = globgtlist[5][delta]
        h.write(str(delta)+" "+str(x)+" "+str(y)+" "+str(z)+" "+str(p)+" "+str(q)+" "+str(r)+"\n")
    h.close()

def random(xmin,xmax,ymin,ymax,zmin,zmax,sequencelength,seqcount):
    tic = time.perf_counter()
    sequencename = 'Randommotion'+str(seqcount)

    if os.path.isdir("UrbanMAVDataSet/"+sequencename+'/GroundTruths'):
        print('directory already exists')
    else:
        os.makedirs("UrbanMAVDataSet/"+sequencename+'/GroundTruths')

    if os.path.isdir("UrbanMAVDataSet/"+sequencename+'/Images'):
        print('directory already exists')
    else:
        os.makedirs("UrbanMAVDataSet/"+sequencename+'/Images/Storage0/Images')
        os.makedirs("UrbanMAVDataSet/"+sequencename+'/Images/Storage1/Images')

    FPS = 30
    GB = round(sequencelength*33/100000,3)

    esttime = round(sequencelength*0.9*2)
    seconds = str(mt.floor((esttime)%60))
    minutes = str(mt.floor((esttime)/60%60))
    hours = str(mt.floor((esttime)/60/60))
    print('estimated running time ' + hours + ' : ' + minutes +' : '+ seconds)
    print('to create around ' + str(sequencelength) + ' image pairs')
    print('folder size estimated at around ' + str(GB) + 'GB')

    deltalist = [[],[],[],[],[],[]]
    globgtlist = [[],[],[],[],[],[]]

    pmax = 45
    qmax = 45
    rmax = 180

    dpmax = 335
    dqmax = 335
    drmax = 335


    dxmu = 350
    dxsigma = 850
    dymax = 950
    dzmax = 575
    for i in range(sequencelength):
        x = round(np.random.randint(xmin,xmax))
        y = round(np.random.randint(ymin,ymax))
        z = round(np.random.randint(zmin,zmax))

        p = round(np.random.randint(-pmax,pmax))
        q = round(np.random.randint(-qmax,qmax))
        r = round(np.random.randint(-rmax,rmax))

        roll,pitch,yaw = p*np.pi/180,q*np.pi/180,r*np.pi/180

        rm = np.array([[np.cos(yaw)*np.cos(pitch), np.cos(yaw)*np.sin(roll)*np.sin(pitch)-np.cos(roll)*np.sin(yaw), np.cos(roll)*np.cos(yaw)*np.sin(pitch)+np.sin(roll)*np.sin(yaw)
                       [np.sin(yaw)*np.cos(pitch), np.sin(roll)*np.sin(yaw)*np.sin(pitch)+np.cos(roll)*np.cos(yaw), np.cos(roll)*np.sin(yaw)*np.sin(pitch)-np.cos(yaw)*np.sin(roll
                       [-np.sin(pitch), np.cos(pitch)*np.sin(roll), np.cos(roll)*np.cos(pitch)]])

        dAngle = np.array([np.random.randint(-dpmax,dpmax+1)/FPS,np.random.randint(-dqmax,dqmax+1)/FPS,np.random.randint(-drmax,drmax+1)/FPS]) #camera frame
        dTrans = np.array([(np.random.randint(-dxsigma,dxsigma+1)+dxmu)/FPS,np.random.randint(-dymax,dymax+1)/FPS,np.random.randint(-dzmax,dzmax+1)/FPS]) #camera frame

        translation = np.matmul(rm,np.transpose(dTrans)) # to global frame
        dx = translation[0]
        dy = translation[1]
        dz = -translation[2]

        dp = dAngle[0]
        dq = dAngle[1]*np.cos(roll)
        dr = dAngle[1]*np.sin(roll) + dAngle[2]

        randomflyto(sequencename,x,y,z,p,q,r,dx,dy,dz,dp,dq,dr,i)

        #Global Frame, total positioning
        x,y,z,p,q,r = x+dx,y+dy,z+dz,p+dp,q+dq,r+dr
        gAngle = [p,q,r]
        gTrans = [x,y,z]

        for ii in range(3):
            globgtlist[ii].append(gTrans[ii]/100) #convert to meters
            deltalist[ii].append(dTrans[ii]/100) #convert to meters
            globgtlist[ii+3].append(gAngle[ii]*np.pi/180) #convert to radians
            deltalist[ii+3].append(dAngle[ii]*np.pi/180) #convert to radians

    writegtcamera(sequencename,deltalist)
    writegtglobal(sequencename,globgtlist)
    toc = time.perf_counter()
    seconds = str(mt.floor((toc - tic)%60))
    minutes = str(mt.floor((toc - tic)/60%60))
    hours = str(mt.floor((toc - tic)/60/60))
    print('running time '+ hours + ' : ' + minutes +' : '+ seconds)

    h = open("UrbanMAVDataSet/"+sequencename+"/GroundTruths/DataDescription"+sequencename+".txt","w")
    h.write("""
"""+sequencename+""":
Random dataset with maximum absolute value for p,q,r = """+str(pmax)+""","""+str(qmax)+""","""+str(rmax)+""" degrees at staring point.
Relative poses are limited to an absolute change of dp,dq,dr = """+str(dpmax)+""","""+str(dqmax)+""","""+str(drmax)+""" [degrees],
and dy,dz = """+str(dymax)+""","""+str(dzmax)+""" meters per second.
dx is determined using a normal distribution with sigma and mu = """+str(dxsigma)+""" and """+str(dxmu)+""" respectively.
Data is gathered at """+str(FPS)+""" FPS.
    """)
    h.close()
```