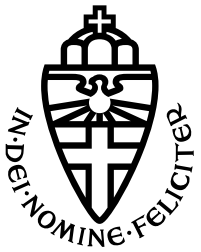
Radboud University Nijmegen



Faculty of Social Science

Frame-to-Frame ego-motion estimation for agile drones with Convolutional Neural Network

Master of Science Thesis

MSc in Artificial Intelligence (Robot Cognition)

Author:
Bogoda Arachchige Sameera Sandaruwan
(s1014012)

Supervisor:
Dr. Serge Thill
Radboud University Nijmegen

External Supervisor:
Dr. G.C.H.E. (Guido) de Croon
Delft University of Technology

April, 2020

Abstract

Openly available feature based Visual Odometery algorithms, have a high computational cost. This hinder the use of these algorithms in agile robotics platforms such as racing drones. A wide range of researchers have looked into, using data-driven learning methods such as Deep Learning, instead use of traditional feature based methods for developing a new class of visual odometery systems. These systems have mainly relied on datasets designed for low degree-of-freedom systems such as cars. Hence, even this new class of algorithms, suffer from high computational cost and inability to handle agile robotics sytems. Therefore, this research will look into developing a learning based visual odometry system, specifically designed for racing drones, which can handle high speed agile motion with low computational cost.

Keywords— Drone, Neural Networks, Visual-odometry, Ego-motion

Acknowledgement

I would first like to thank my first (and daily) supervisor Dr. G.C.H.E. (Guido) de Croon, whose constant support and encouragement along with all the feedback for the little and big things that helped me throughout the entire project. Without him listening to my problems and helping me deal with them, I would not have been able to finish this thesis. I also want to thank my second supervisor Dr. Serge Thill who consistently allowed this thesis to be my own work, but steered me in the right direction whenever I needed it. A big thanks to my two immediate supervisors, Fede Paredes Valles, Yingfu Xu without whom, designing and constructing this entire study would not have been possible, as their skills and feedback brought a lot on the table. I do not know how to extend my thanks to my family who stayed strong for me and encouraged me to give my all. Lastly, I want to thank some of my friends - Nilay Seth and Rohan Chotalal for always being there for brain storming problems and ideas, which fast tracked the progress of my thesis. Special thanks to Patrick Geneva from University of Delaware for giving me access to ros launch files for my analysis with OpenVINS.

Contents

List of Figures

List of Tables

Nomenclature

$\{\mathbf{B}\}$	Body reference frame
$\{\mathbf{C}\}$	Camera reference frame
$\{\mathbf{W}\}$	World reference frame
$\mathbf{A}$	Matrices
$\mathbf{p}, \vec{p}$	Vectors

Chapter 1

Introduction

An Unmanned Aerial Vehicle (UAV) is an aircraft with no human pilot on board. UAVs can be controlled by a remote or can be autonomously flown based on a pre-defined mission plan. Over the past 15 years, development and the use of UAVs have increased rapidly. Especially, a sub-category of UAVs called quadrotors (drones) has made their way to the consumer market. Quadrotors are widely applicable in military missions such as surveillance and reconnaissance, emergencies such as search and rescue, as well as scientific research including earth sensing and data collection. Additionally, their use can also be found in civilian activities such as photography and cinematography.

Recently, these quadrotors have found their way to the gaming sector. Specifically, this new use case of quadrotors aim to perform autonomous drone racing. Autonomous drone racing has the same entertainment appeal as other robotic competitions. This is mainly due to its high speed motion that entails a high risk of failure which can induce adrenaline for the spectators. On the other hand, these drone races have piqued the interest of robotic researchers due to its agile nature and real-world complexity, which is not available for ground-based robotics platforms. As a result, numerous universities and research laboratories around the world have started participating in such races [1, 5].

1.1 Brief introduction to autonomous drone racing

A drone race consists of flying through a set of checkpoints or gates and crossing the finish line as fast as possible. A map of the locations of the checkpoints is given to the participants beforehand with a probability of uncertainty. This mitigates the problem of hard-coding the gate positions inside the flight controllers by the teams and encourage the implementation of onboard localization algorithms. Therefore, the quadrotors must localize its location against the gates, plan trajectories and fly through them to get to the final gate with minimum time.

Successful implementation of a drone that is capable of autonomous navigation, challenges in two major research areas - perception and control. Perception algorithms use computer vision to perceive the surroundings to estimate the drone's position, velocity, and orientation in the 3-dimensional (3-D) space. Using the information provided by the perception system, control algorithms control the motor speeds, to correctly follow the trajectory to the goal. The execution of this pipeline poses challenges such as lighting conditions and motion blur for perception algorithms, trajectory generation and tracking for control algorithms and so on.

Research conducted in perception and control for drone racing directly translate to a wide variety of robotic systems such as robotic manipulators, control of self-driving cars, the control system of satellites and navigation system of Mars Rover [6]. Since quadrotors are agile robotic systems with a higher number of degrees of freedom, algorithms that work well with quadrotors have the potential to work well with the aforementioned systems with a relatively smaller amount of changes.

Figure 1.1: FlightGoggles simulator developed by MIT FAST Lab [1]

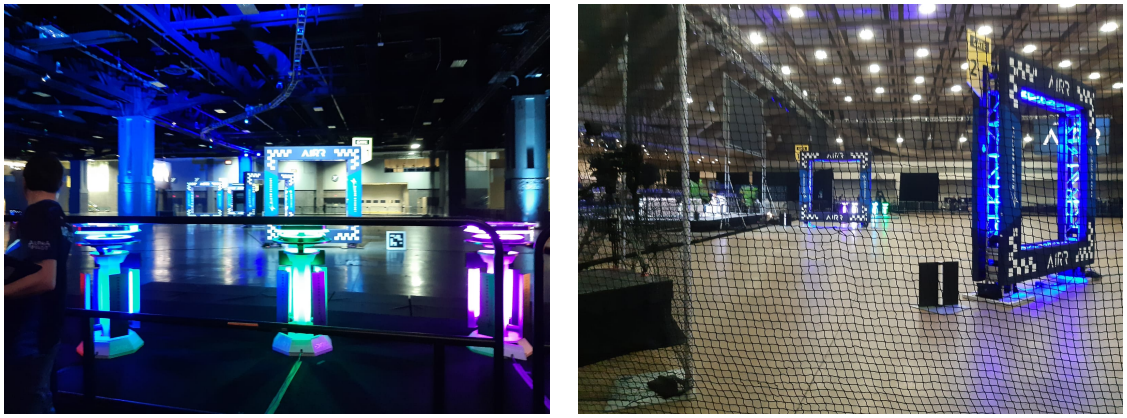


Figure 1.2: AlphaPilot Challenge 2019 arena

1.2 Motivation

The primary motivation of this thesis to bridge the gap between traditional robotics and Deep Learning (DL). The field of robotics still heavily relies on model-driven implementation approach. Hence, such robotics systems perform reliably within their constraints. On the other hand, this approach takes more time and effort. In the last eight years, DL has proven to be a valuable tool for developing data-driven[1] systems. Since AlexNet [7] won the ImageNet [8] competition by a significant margin in 2012, DL's popularity has risen dramatically[2]. Furthermore, DL systems have shown remarkable improvement in accuracy in a wide variety of fields.

As mentioned in the beginning of this thesis a sub-category of UAVs is quadrotors. Quadrotor flight controller pipeline consists of two main sub-modules, perception and control. Visual Odometry (VO)/Visual-Inertial Odometry (VIO)$^{(2.1.1)}$ algorithms are common in the industry in providing critical information about the world and its position, velocity and orientation to relevant system controllers. Most of the openly available VO/VIO algorithms are computationally intensive (section 1.2.1).

[1] Use large amount of data and machine learning to find relevant patterns within the data. This removes the time and effort taken for mathematical modeling.

[2] `https://pathmind.com/wiki/use-cases`

A large proportion of drones takes a small form factor format. Therefore, the number of computational resources available on-board is minuscule compared to traditional computers, which is why, aforementioned algorithms are not suitable for drones.

Consecutively this leads to an immense need in the industry for VO systems that have smaller computational and complexity footprints, and are capable of producing accurate enough outputs to be used for drone navigation. Therefore, this thesis will look into the research and implementation of the Neural Network (NN) based VO system that complies with the aforementioned criterion.

Furthermore, if the objectives of this thesis were accomplished, the neural network could be a potential addition to the drone race team at TU Delft. The team originated from the Micro Aerial Vehicle Lab[3] (MAVLab) of the Aerospace faculty and it consists of professors, Ph.D. students, and Masters students. Teams from the lab have consistently won podium positions in drone competitions such as IROS[4] and IMAV[5] in the past. Yet, one of the greatest achievements from the lab is becoming the world champions in the 2019 AlphaPilot challenge[6] organized by Drone Racing League and Lockheed Martin. The philosophy followed by MAVLab is to fly fast using simple but robust vision and control algorithms while trying to comply with time-optimal trajectories.

1.2.1 Traditional Visual Odometry pipeline

- Acquire input images: using either single cameras, stereo cameras, or omni-directional cameras.
- Image correction: apply image processing techniques for lens distortion removal, etc.
- Feature detection: define interest operators, and match features across frames and construct optical flow field.
  - Use correlation to establish correspondence of two images, and no long term feature tracking.
  - Feature extraction and correlation.
  - Construct optical flow field (Lucas–Kanade method).
- Check flow field vectors for potential tracking errors and remove outliers.
- Estimation of the camera motion from the optical flow.
  - Choice 1: Kalman filter for state estimate distribution maintenance.
  - Choice 2: find the geometric and 3D properties of the features that minimize a cost function based on the re-projection error between two adjacent images. This can be done by mathematical minimization or random sampling.
- Periodic re-population of track points to maintain coverage across the image.

The VO pipeline was taken from wikipedia[7].

1.3 Research question

Ego-motion is the estimation of the motion of a camera observed in an image sequence caused by the motion of the camera itself and by motions of objects moving in the scene [9]. VO is the process of determining the relative (between consecutive images) position and orientation of a robot by analyzing the associated camera images[8]. A traditional VO algorithm can perform with very high accuracy but it comes at the cost of higher computational expense. Furthermore, the accuracy deteriorates rapidly when these algorithms encounter blurry images, which is a common

[3] More information: `http://mavlab.tudelft.nl/`

[4] More information: `http://iros.org/`

[5] More information: `http://www.imavs.org/`

[6] More information: `https://lockheedmartin.com/en-us/news/events/ai-innovation-challenge.html`

[7] `https://en.wikipedia.org/wiki/Visual_odometry`

[8] More information: `https://en.wikipedia.org/wiki/Visual_odometry`

occurrence in high-speed motion as desired by drone racing. Section 2.1.1 will describe in detail how a traditional VO algorithm works.

After a certain period of time, when a drone is hovering or performing high-speed maneuvers, the estimation of the drone attitude changes due to several reasons. They are (a). mismatch between the Inertial Measurement Unit (IMU) sampling rate inside the sensor and sensory data acquisition rate of the micro-controller (b). existence of axis cross-coupling that depends on sensor alignment and (c). the motion of gravity vector at higher speeds.

Therefore the attitude controller will try to compensate for this error, which then leads to the drone wandering from its original position in 3D space. In this scenario, the images from an onboard camera clearly indicate that the drone was stationary, which is the reason for using images as the only input. Hence the motivation for using a VO system in a traditional drone flight controller pipeline (section 2.4) to avoid crashes. The estimated ego-motion using a VO system can be used with a traditional drone controller (Fig. 2.5) to produce accurate state estimations by reducing accumulated drift caused by IMU.

The purpose of this research is to develop a NN based VO system that only takes images as input and produces accurate estimates of ego-motion (section 2.1.3) at higher speeds. This research question is to determine if a NN based system can learn motions such as rotation and translation, use these motions to estimate relative pose between consecutive images.

1.3.1 Objectives

- Produce accurate VO estimations which is robust to blur, executes faster with less computational overhead
- Mitigate accumulated drift caused by IMU

1.3.2 Requirements of the system

- Minimum number of inputs (For this research only two consecutive monocular images. No Optical flow, depth map or IMU information as inputs)
- Smaller network (in size and complexity)
- Fast inference (For a flight with mean speed of 10 m/s, the vision system should work at least 60 Hz)
- Robust during high speed movements

1.4 Organization

Chapter 2 is a supplementary introduction chapter that explains the symbols, conventions and other relevant background information which is imperative for the understanding of this research. It also conveys definitions regarding quadrotor dynamics, Visual Odometry (VO).

Chapter 3 explains all the steps taken throughout the research. It starts with an explanation of the dataset and its attributes, followed by the explanation of the processing of data, neural network architectures, and the training process.

In chapter 4, the results of the research are expressed. At first, the evaluation metrics used are explained. Thereafter, an analysis section show the results obtained through different stages of tweaking and tuning. Furthermore, this chapter will also compare the output trajectory of the neural network with currently existing VO/VIO systems.

Chapter 5 elaborates on the results from chapter 4. In chapter 6, there will be in-depth explanations on the final outcome of the research and a few suggestions (that remain unexplored in this thesis) on how to improve the system.

Various NN architectures, hyper-parameters, and training techniques were explored and proposed during the thesis. The workflow followed was to first draft a working pipeline consisting of data processing, training, and evaluation. Python language was used for all coding purposes throughout the research.

Chapter 2

Background on Visual Odometry & Quadrotor Dynamics

2.1 A brief introduction to Visual Odometry

2.1.1 Visual Odometry (VO)

Odometry is the measurement of the traveled distance of a moving object such as a car or bicycle[1]. In the field of Robotics, odometry is a generalized term, and often refers to the estimation of not only the distance traveled but the entire trajectory of a moving robot. The parameterized version of odometry in robotics is referred to as the pose vector. Therefore, for every discrete timestep t, the pose vector can be expressed as $[x^t, y^t, z^t, \alpha^t, \beta^t, \gamma^t]$. Here, $[x^t, y^t, z^t]$ are the Cartesian coordinates while $[\alpha^t, \beta^t, \gamma^t]$ are the euler angles of the robot[2].

Visual Odometry is the process of determining the aforementioned pose vector (with respect to an initial position and attitude), of a camera by analyzing a sequence of images. VO is used in a variety of applications, such as mobile robots, self-driving cars, and unmanned aerial vehicles. Using a single camera and a dual-camera setup for VO is referred to as *Monocular*[3] *Visual Odometry* and *Stereo Visual Odometry*, respectively. This research will focus on the monocular approach.

Both of these approaches yield advantages and disadvantages. An immediate advantage of stereo is its ability to estimate the exact trajectory, while the monocular approach estimates the trajectory only up to a unique scale factor. Therefore, monocular VO algorithms can only estimate movements based on a certain normalized unit. On the other hand, algorithms based on the stereo approach can directly estimate the movements in meters. A disadvantage of the stereo approach is that if the ratio between the distance from the camera to an object and the distance between the cameras is much higher, the performance of the stereo approach comes to the same level as the monocular approach. Hence, using the stereo approach, on a small form factor system does not yield better performance over the monocular approach.

Furthermore, traditional VO pipelines (section 1.2.1) can be quite computationally expensive. Since all algorithms should run onboard the drone, such algorithms are not feasible [10]. As a result of the computational complexity, these algorithms can have high execution times which is not ideal for real-time control applications.

2.1.2 Visual-Inertial Odometry (VIO)

Visual-Inertial Odometry entails the use of a secondary sensor such as an IMU in addition to a camera to estimate the pose described in section 2.1.1.

2.1.3 Ego-motion

Ego-motion is the motion of the camera observed in an image sequence caused by the motion of the camera itself and by motions of objects moving in the scene [9].

[1] `https://en.wikipedia.org/wiki/Odometer`

[2] `https://avisingh599.github.io/vision/visual-odometry-full/`

[3] Monocular is sometimes referred to as "mono".

2.1.4 Deep Neural Network Based Visual Odometry

Unsupervised learning: Zhou et al. [11] have used the unsupervised learning approach by combining two separate networks that perform depth and pose estimation, simultaneously. Despite its accurate performance over the short version of ORB-SLAM [12], the model faces few shortcomings, such as the inability to explicitly perform scene dynamics and occlusion, the assumption of given camera intrinsics, and produced depth maps represent a simplified version of the underlying 3D scene.

Self-supervised learning: Sudeep et al. [13] was able to develop an end-to-end trainable ego-motion estimator, using self-supervised learning, which was robust enough to be used with different types of generic camera optics. Furthermore, this model could be bootstrapped by sensory information available through the robotic system, which was able to refine the training process. Even though the system was able to perform with high accuracy, the model was not able to perform in a resource-restricted environment. Lee et al. [14] have used a self-supervised learning-based novel approach for prediction of ego-motion. The model utilizes two networks that perform segmentation and optical flow estimation separately. The robustness of the model was increased by employing a two-layer segmentation approach.

The aforementioned related works have been conducted related to ground vehicles (KITTI [15]) or static camera systems. Therefore, they have not looked into the orientational freedom of the dynamic system, which is significantly important for a UAV. Furthermore, they have not focused on the computational cost of these networks, which again is critical for standalone robotic systems such as drones.

2.2 Frames of reference and Rotation matrices

2.2.1 Frames of reference

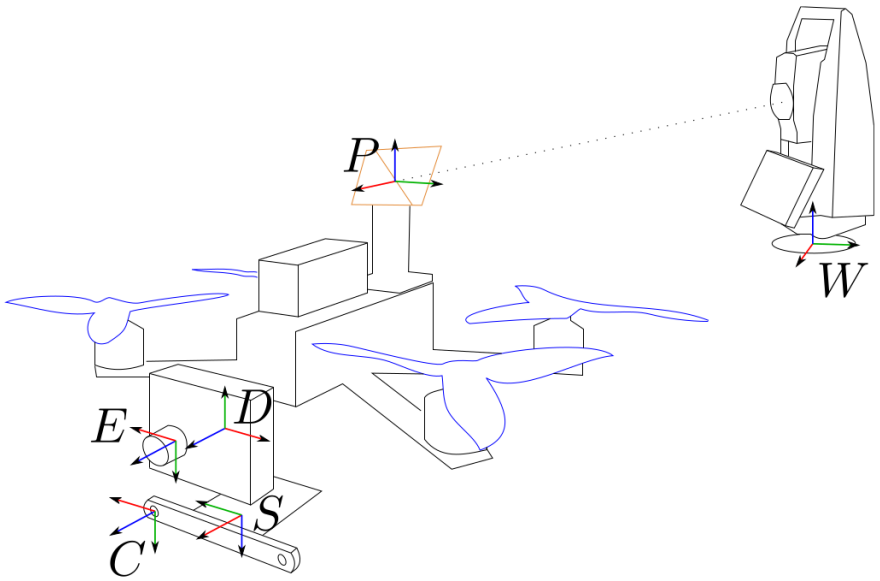


Figure 2.1: Reference frame used in this research. W: World frame, P: Center of Leica prism, the MS60 tracks the position of P relative to W. E: event camera and D: IMU on the mDAVIS sensor. C: standard camera and S: IMU on the Snapdragon Flight. For each frame, red, green and blue arrows represent the unit vectors in x, y and z, respectively. (courtesy of Robotics and Perception Group of University of Zurich [2])

In mathematics and physics, the right-hand rule is a common mnemonic for understanding orientation of axes in 3-dimensional (3-D) space and all the frames of reference in the thesis comply with this rule.

Reference frames are used to represent the Euclidean space in three-dimensions. These reference frames use a Cartesian coordinate system to specify a single point in space uniquely by a set of numerical coordinates. Each axis in a reference frame is perpendicular to each other. Any information represented in one frame of reference can be converted to another frame of reference by using rigid body transformation. There are three reference frames used in this research: World frame $\{\mathbf{W}\}$ (also know as the inertial frame), Body frame $\{\mathbf{B}\}$ and Camera frame $\{\mathbf{C}\}$.

2.2.2 Rotation matrices

A common method of denoting the orientation of one frame with respect to the other is by the use of Euler angles. The ***Proper Euler angles***[4] conventions involve sequential rotations, which means that the entities listed are the angles between each consecutive rotation, which usually follows the (z-y-z) sequence of rotation. In the Tait-Bryan representation (which is used in most aerospace applications), angles indicate orientation with respect to the original frame of reference (x-y-z, ...) rather than with respect to each sequential rotation. The angles ϕ, θ, ψ in equation 2.1 can be referred to as the angles in Tait Bryan representation[4]. Therefore, the Tait Bryan representation is used throughout the thesis to represent the orientation of the quadrotor. Rotation matrices can hence be given as:

$$\mathbf{R}_x(\phi)=\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}, \mathbf{R}_y(\theta)=\begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, \mathbf{R}_z(\psi)=\begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

Where ϕ, θ, ψ are the roll, pitch and yaw of the quadrotor with respect to the world frame of reference. The rotation matrix generated after following the x-y-z sequence can be given as:

$$\mathbf{R} = \mathbf{R}_z(\psi) \cdot \mathbf{R}_y(\theta) \cdot \mathbf{R}_x(\phi) \tag{2.2}$$

$$\mathbf{R} \in \mathrm{SO}(3)$$

A *Special Orthogonal group* (SO) refer to a group of rotations which suffice the following criterion [16];

$$\mathrm{SO}(n) = \mathbf{R}^T\mathbf{R} = \mathbf{R}\mathbf{R}^T = \mathbf{I},\ |\mathbf{R}| = 1$$

where, n is the number of dimensions (3). SO(3) form a *non-commutative group*[17].

$$\mathbf{Q} = \left[\begin{array}{c|c} \mathbf{R}_{3\times3} & \mathbf{t}_{3\times1} \\ \hline \mathbf{0}_{1\times3} & 1_{1\times1} \end{array}\right] \tag{2.3}$$

A *special Euclidean group* (SE) is a Euclidean space group that suffice the following criterion;

$$SE(3) = \left\{\mathbf{A} = \mathbf{Q},\ \mathbf{R} \in R^{3\times3}, \mathbf{r} \in R^3, \mathbf{R}^T\mathbf{R} = \mathbf{R}\mathbf{R}^T = \mathbf{I}\right\} \tag{2.4}$$

2.2.3 Rigid body transformation

Figure 2.2 shows, two rigid bodies (A, B) and each of them has their own frame of reference ($\{\mathbf{A}\}$, $\{\mathbf{B}\}$). Origin of the respective frames are indicated as $\mathbf{O}, \mathbf{O}'$.

To transform the position vector $\vec{p}$ in $\{\mathbf{B}\}$ to position vector $\vec{p}$ in $\{\mathbf{B}\}$,

$${}_A\vec{p} = {}^A\mathbf{R}_B \cdot {}_B\vec{p} + {}^A\mathbf{t}_B \tag{2.5}$$

where, ${}^A\mathbf{R}_B$ represent the rotation of $\{\mathbf{B}\}$ with respect to $\{\mathbf{A}\}$ and ${}^A\mathbf{t}_B$ represent the translation of the origin of $\{\mathbf{B}\}$ with respect to the origin of $\{\mathbf{A}\}$. The allows the description of $\{\mathbf{B}\}$ as seen from an observer in $\{\mathbf{A}\}$.

$$\mathbf{t} \in \mathbb{R}^3 \tag{2.6}$$

[4] `https://en.wikipedia.org/wiki/Euler_angles`

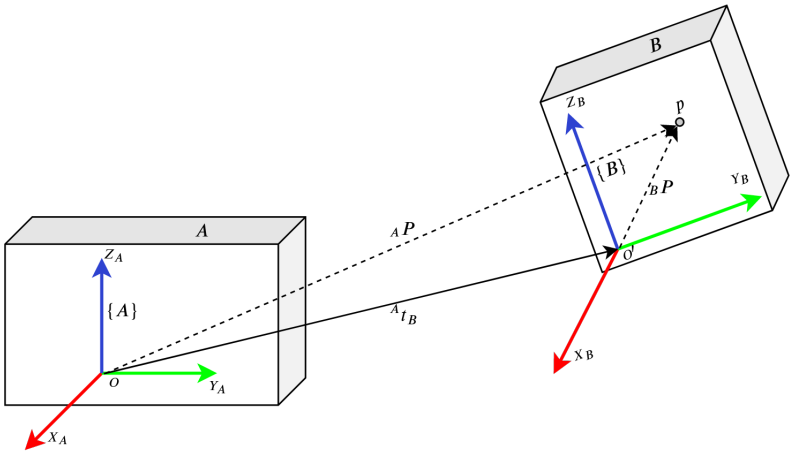


Figure 2.2: Schematics of the considered quadrotor model with the used coordinate systems and rotor forces

We denote the rotation matrix that converts from system $\{\mathbf{B}\}$ to $\{\mathbf{W}\}$ as ${}^{W}\mathbf{R}_{B}$ and the translation of system $\{\mathbf{B}\}$ with respect to system $\{\mathbf{W}\}$ as ${}^{W}\mathbf{t}_{B}$, respectively. This convention will be consistent throughout the research. To convert a vector $\vec{c}$ from the body frame $\{\mathbf{B}\}$ to the world frame $\{\mathbf{W}\}$ we use equation 2.7.

$$ {}_{W}\vec{c} = {}^{W}\mathbf{R}_{B} \cdot_{B} \vec{c} \tag{2.7} $$

Correspondingly, to convert a point $\vec{p}$ from the body frame B to the world frame W we use;

$$ {}_{W}\vec{p} = {}^{W}\mathbf{R}_{B} \cdot_{B} \vec{p} + {}^{W}\mathbf{t}_{B} \tag{2.8} $$

The homogeneous transformation matrix is constructed as follows;

$$ \mathbf{T}_{4\times4} = \left[\begin{array}{c|c} \mathbf{R}_{3\times3} & \mathbf{t}_{3\times1} \\ \hline \mathbf{0}_{1\times3} & 1_{1\times1} \end{array}\right] \in \mathrm{SE}(3) \tag{2.9} $$

2.2.4 Camera frame of reference

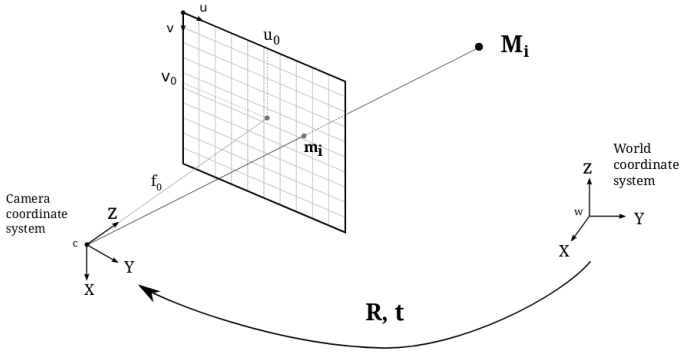


Figure 2.3: Pinhole camera model [3]

2.3 Quadrotor Dynamics

Three axes (X, Y, Z) are used to define the motion of a quadrotor. Roll, pitch and yaw are the rotation around each of these axes and they are denoted by ϕ, θ and ψ respectively (Figure 2.4). A quadrotor is an underactuated system, which means it have fewer controls than degrees of freedom [18]. A drone is a 6 Degree of Freedom (DoF) system. But it only has four control inputs (thrust, roll, pitch, yaw).

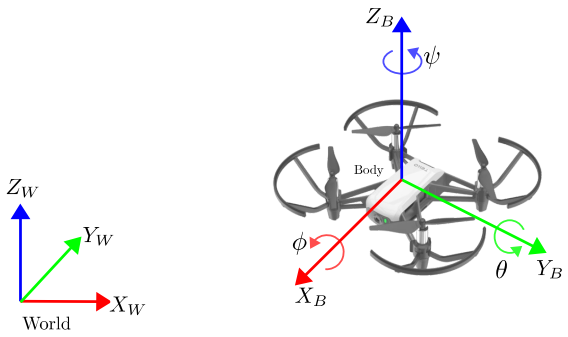


Figure 2.4: UZH data collection drone

Thrust controls the translational motion through $\mathbf{Z}_B$ axis. Roll and pitch control angular motion around $\mathbf{X}_B, \mathbf{Y}_B$ axes respectively. These translate to translational motion through $\mathbf{Y}_B, \mathbf{X}_B$ axes respectively. Yaw control the angular motion around $\mathbf{Z}_B$ axis.

2.4 Quadrotor flight controller

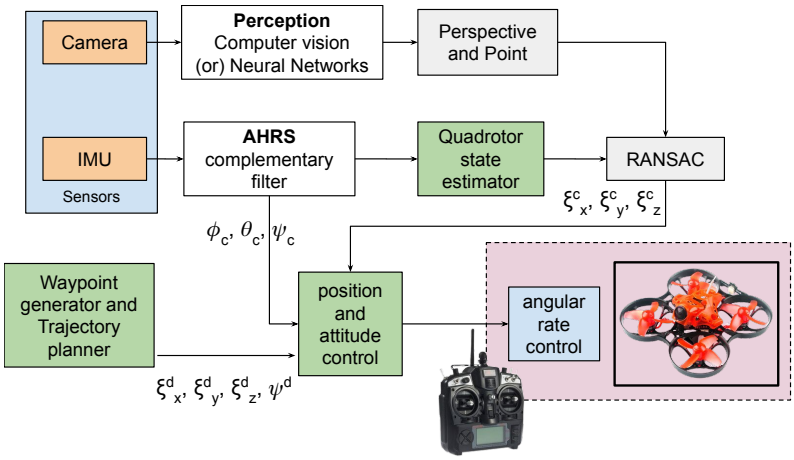


Figure 2.5: MAVLab's drone race flight controller pipeline (courtesy of Nilay Seth - MAVLab)

Chapter 3

Methodology

3.1 Dataset

For the purpose of this research, the FPV Drone Racing dataset [2] developed by the University of Zurich was used. The dataset consist of 27 drone flight sequences available in the dataset and 12 of them does not contain ground-truth. These sequences below to 4 different categories:

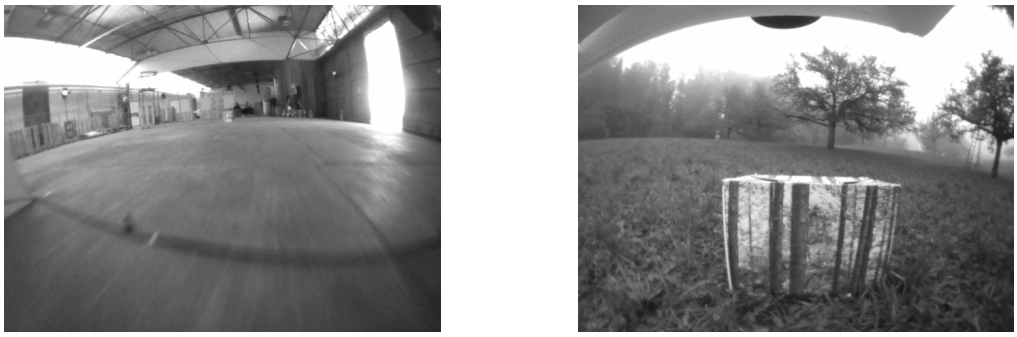
1. Indoor forward-facing
2. Indoor 45 degrees downward facing
3. Outdoor forward-facing
4. Outdoor 45 degrees downward facing

Related work that have been done in the past has focused more on either robotic systems that have less DoF (compared to a drone) such as cars or have used datasets that were not representative of racing drones. Conditions surrounding racing drones such as high speed motion (a). affect the IMU based state estimation and (b). produce blurred images which cause problems for traditional feature-based VO systems. The prevalence of these attributes and the drone race friendly flying method (FPV) used are the reasons for selecting this dataset for the research.

Figure 3.1: Indoor and outdoor environments in the dataset

In this research only category 1 will be used, which contains 9 drone flight sequences. Out of those 9, sequence numbers 8, 11 and 12 do not have ground truth data. Therefore, from the rest of the sequences, sequence numbers 3, 5, 6, 7, 9 will be used for training and sequence numbers 10 will be used for testing. Each sequence has a difficulty level (easy, medium, hard).

The dataset was collected by a quadrotor system equipped with different sensors (Fig. 3.3a). The drone was flown by a First-Person View (FPV) pilot on a few predetermined trajectories.



(a) Indoor environments (b) Outdoor environment

Figure 3.2: Dataset environments

Sequence #	Duration(s)	Length (m)	vmax (m/s)
3	54.63	287.12	9.5
5	50	156.47	4.87
6	32.93	223.27	12.52
7	73.2	**333.59**	12.78
8	**132.53**	259.16	5.26
9	34.04	157.07	11.42
10	33.43	149.36	9.49
11	24.02	85.68	10.32
12	31.98	124.07	**15.28**

Forward facing

Sequence #	Duration(s)	Length (m)	vmax (m/s)
1	73.99	150.71	4.36
2	55.77	218.9	6.97
3	**57.82**	119.82	3.53
4	47.36	168.06	6.55
9	40	215.58	11.23
11	22.96	125.21	**11.74**
12	51.25	124.56	4.33
13	42.49	166.62	7.92
14	43.66	**220.4**	9.54
16	15.49	58.72	7.69

Downward facing (45 degree)

Table 3.1: Indoor environment

Sequence #	Duration(s)	Length (m)	vmax (m/s)
1	49.63	258.23	8.55
2	36.9	220.88	10.13
3	**92.84**	**735.51**	14.04
5	22.21	189.63	**20.73**
6	34.83	338.2	19.42
9	43.15	314.41	10.68
10	59.6	455.63	12.58

Forward facing

Sequence #	Duration(s)	Length (m)	vmax (m/s)
1	24.49	**165.53**	**15.62**
2	**26.19**	143.13	10.68

Downward facing (45 degree)

Table 3.2: Outdoor environment

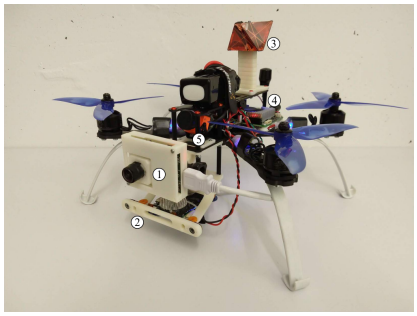
This means that the drone was at a constant altitude of 0 meters (the ground was at -1 meters). Therefore, the drone was at a constant pitch angle throughout a sequence only having drastic changes at the beginning and the end of the trajectory.

Furthermore, due to the method of flying (FPV), yaw motion has higher prominence compared to roll and pitch. Due to this preference of motion, even though the drone has 6 DoF, the drone race dataset contains a large amount of variance in Tx, Ty, and Rz (camera frame) axes, compared to the rest of the degrees of freedom. A similar situation arises in the KITTI dataset [15], where the data collection is done using a driving around a car, which leads to more variance in Tx, Ty, and Rz.

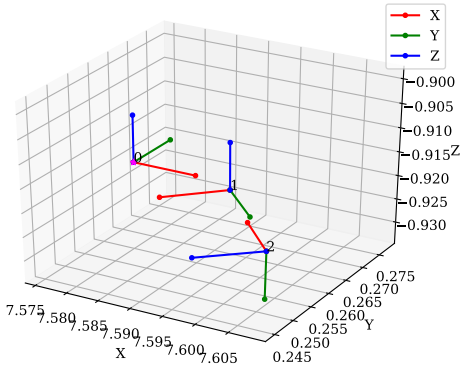
Compared to other available datasets (Table 3.3), the UZH-FPV dataset contains aggressive flight sequences which means a higher amount of motion blur is available. Furthermore, the drone was flown by an FPV pilot, which gives this dataset a unique perspective of motion. A similar approach is taken when developing self-driving cars [1].

The dataset consists of two environments, indoor and outdoor (Figure 3.1). The indoor environment is an unused airplane hangar, with an open space area of 30m $\times$ 15m $\times$ 6m. The indoor image sequence contains areas with saturated lighting conditions such as one part of the roof and

[1] NN based self-driving car systems use thousands of hours of videos of people driving cars to train the network. All these videos are taken from a FPV perspective.

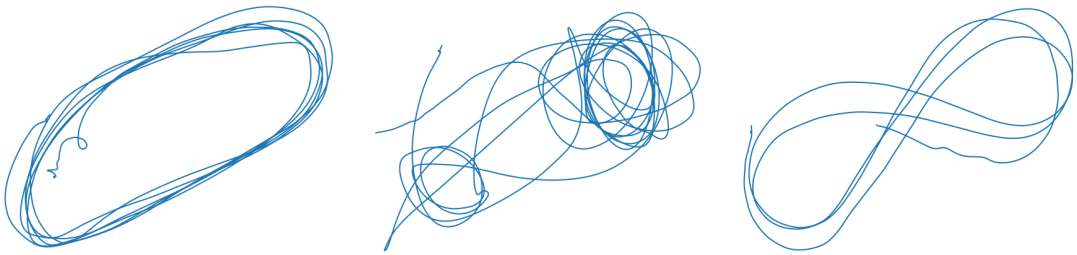


(a) The recording platform: ② Snapdragon Flight with fisheye stereo cameras ③ Omni-directional reflector that provides the ground truth position. ④ Up Board used for recording the DAVIS data. ⑤ FPV camera used by the human pilot. [2]



(b) World (*0*), body (*1*) and camera (*2*) reference frames in UZH dataset

Figure 3.3: UZH Flying Platform and its axes



(a) Sequence 3 (Easy) (b) Sequence 7 (Hard) (c) Sequence 10 (Medium)

Figure 3.4: Samples sequences from the dataset

	EuRoC MAV [19]	UPenn Fast Flight [20]	Zurich Urban MAV [21]	Blackbird [22]	**UZH-FPV Drone Racing** [2]
Environments	2	1	3	**5**[a]	2
Sequences	11	4	1	**186**	27
Camera *(Hz)*	20	40	20	**120**	30/50[d] + events
IMU *(Hz)*	200	200	10	100	**500/1000**[e]
Motor Encoders *(Hz)*	n/a	n/a	n/a	∽**190**	n/a
Max. Distance *(m)*	130.9	700	**2000**	860.8	340.1/923.5[f]
Top Speed *(m/s)*	2.3	17.5	3.9[b]	7.0	**12.8/23.4**[g]
mm Ground Truth *(Hz)*	20/100[c]	n/a	n/a	**360**	20

Table 3.3: Comparison of visual-inertial datasets. Notes: *a* - Visual environments are rendered in photorealistic simulation. *b* - Velocity from GPS. *c* - Rate of precise ground truth from Leica in Machine Hall sequences/Vicon Room sequences. *d* - Frame rate of Snapdragon Flight/mDAVIS. *e* - IMU rate of Snapdragon Flight/mDAVIS. *f* - Distance indoor/outdoor sequences. *g* - Speed for indoor/outdoor sequences, both are larger than existing indoor or outdoor sequences, respectively [2].

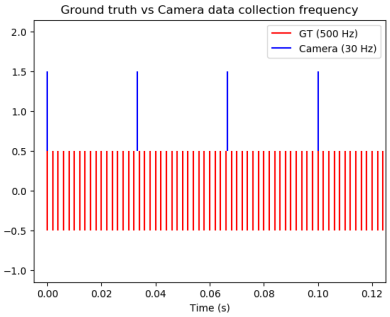
the entrance door to the hanger. Furthermore, it contains a low textured floor and a lot of motion blur. The outdoor environment was a large field with a subtle slope toward some woods, with several isolated trees.

3.1.1 Pre-processing

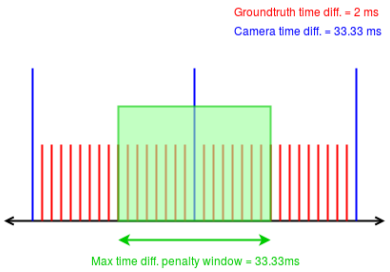
3.1.1.1 Spatio-temporal alignment

Ground-truth pose (translation and orientation of the drone) and image sequence file have two different time-stamps (start time of data collection from each sensor were different). Since the

data collection frequencies are different (ground-truth - 500Hz, camera - 30Hz), more ground truth samples were available compared to camera images. Therefore, the alignment process is required. Since camera images are the main input to the network, the alignment process was used to find the closest ground truth data to the corresponding images. After finding the closest ground truth sample per image, it was evident that the time difference between some pairs of ground truth and images were larger than one second. This was due to the different starting times of data collection by each sensor, which is indicated in Fig. 3.6. These outliers in the dataset would reduce the accuracy of the network. Therefore, a time difference penalty of 0.01667 seconds was introduced to remove the aforementioned outliers from the dataset. Figure [3.5a] shows the frequency of occurrence.



(a) Ground-truth vs camera data collection frequency



(b) Time alignment penalty

Figure 3.5: Time alignment of ground-truth and images

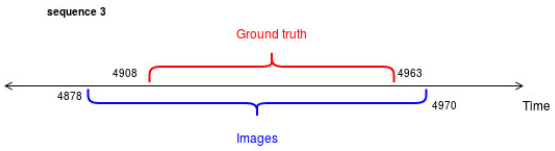


Figure 3.6: Dataset collection for different sensors

3.1.1.2 Frame to frame pose calculation

Dataset is recorded in the world frame $\{\mathbf{W}\}$. Since the input images to the network are recorded in camera frame $\{\mathbf{C}\}$, it will be easier to evaluate the performance of the network, if all ground-truth data was in the same frame.

First, we need to convert the world frame $\{\mathbf{W}\}$ ground-truth to camera frame $\{\mathbf{C}\}$ ground-truth.

$$ {}^{W}\mathbf{R}_B, {}^{W}\mathbf{t}_B \rightarrow {}^{W}\mathbf{T}_B \tag{3.1} $$

UZH dataset is repsented as in the equation equation 3.2.

$$ \{ {}^{W}\mathbf{T}_{B_1}, {}^{W}\mathbf{T}_{B_2}, \cdots {}^{W}\mathbf{T}_{B_N} \} \tag{3.2} $$

where, N is the number of samples in the dataset. Transformation of vectors in reference frame $\{\mathbf{B}\}$ to reference frame $\{\mathbf{C}\}$ can be done using the homogeneous transformation matrix indicated in equation 3.3,

$$ {}^{C}\mathbf{T}_B \tag{3.3} $$

Therefore, using equation 3.3 and 3.2, ground-truth data in $\{\mathbf{W}\}$ can be converted to $\{\mathbf{C}\}$ as follows;

$$ {}^{W}\mathbf{T}_C = {}^{W}\mathbf{T}_B \cdot ({}^{C}\mathbf{T}_B)^T \tag{3.4} $$

The ground-truth will now be represented as follows;

$$\{ {}^{W}\mathbf{T}_{C_1}, {}^{W}\mathbf{T}_{C_2}, \cdots {}^{W}\mathbf{T}_{C_N} \} \tag{3.5}$$

Now we need to convert the dataset to frame-to-frame presentation.

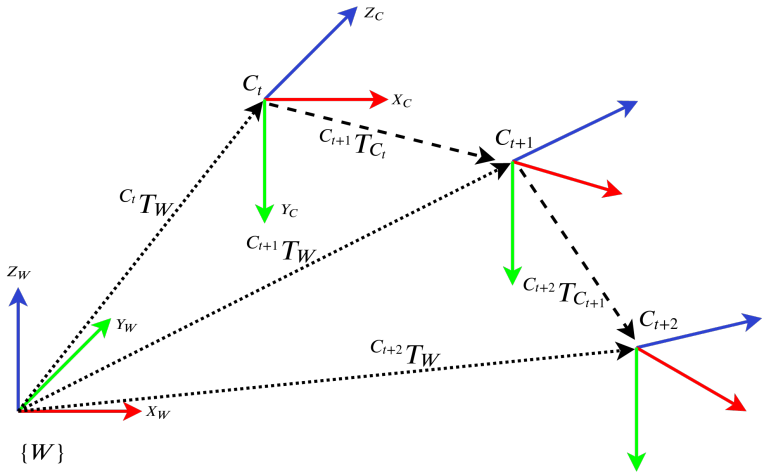


Figure 3.7: Frame-to-frame motion

$${}^{C_1}\mathbf{T}_{C_2} = ({}^{W}\mathbf{T}_{C_1})^{T} \cdot {}^{W}\mathbf{T}_{C_2} \tag{3.6}$$

which turns the dataset representation into the format represented by equation 3.7.

$$\{ {}^{C_1}\mathbf{T}_{C_2}, {}^{C_2}\mathbf{T}_{C_3}, \cdots {}^{C_N-1}\mathbf{T}_{C_N} \} \tag{3.7}$$

The frame to frame conversion was validated using the MATLAB Robotics toolbox developed by Peter I. Corke[23].

3.1.2 Data augmentation

3.1.2.1 Image mirror

Due to the method of flying, the dataset contains a large amount of motion in $-Y_C$ direction (circular trajectories). Flipping the image around the Y_C axis removes this unbalanced motion in the dataset. As shown in figure 3.8, when the image is flipped around Y_C axis, the direction of X_C, Z_C changes. Since the direction of Y_C does not change, its value does not change. The value of X_C will be flipped because of the direction change. Although, the direction of Z_C changes, the motion of the camera is still in the forward direction $(-Z_C)$. Therefore, the relative motion between frames in Z_C still has a positive value. Hence, it is not flipped. When it comes to rotations around each axis, values around Y_C and Z_C will be flipped.

3.1.2.2 Variable Relative Offset (VRO)

VRO is a (novel) data augmentation method specifically designed[2] for this research, to be used with frame-to-frame relative motion situations. The traditional method only considers the consecutive frames in an image sequence and calculate the relative motion between them (Fig. 3.7). Since the image capture rate (30 Hz) is constant for a camera, the magnitude of the relative motion in rotation and translation only depends on the velocity of the drone. This leads the dataset to become very specific to the dynamics of the drone.

Therefore, VRO is used to lower this specificity. VRO 1 related to the aforementioned frame to frame conversion. VRO 2 relates to considering frame n and frame $n+2$ for the frame to frame conversion. VRO 3 relates to considering frame n and frame $n+3$ for the frame to frame

[2]For the purpose of this research, a novel data augmentation method was conceptualized.

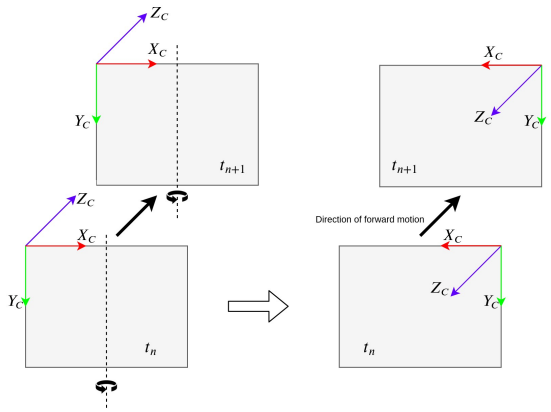


Figure 3.8: Dataset mirror (Translation)

conversion. Hence, this approach can be extended to any number of frame offset values. Figure 3.9 clearly articulates the VRO process. This removes the 30 Hz image collection rate from the dataset, which allows the network to learn a more generalized method of relative pose estimation.

This process of data augmentation can pair up images with very small or no common features, which will hinder the learning process of the network (Fig. 3.12). Especially due to the fact that this dataset contain, a lot of high-speed motion. Therefore, VRO 1 to 5 is used during training.

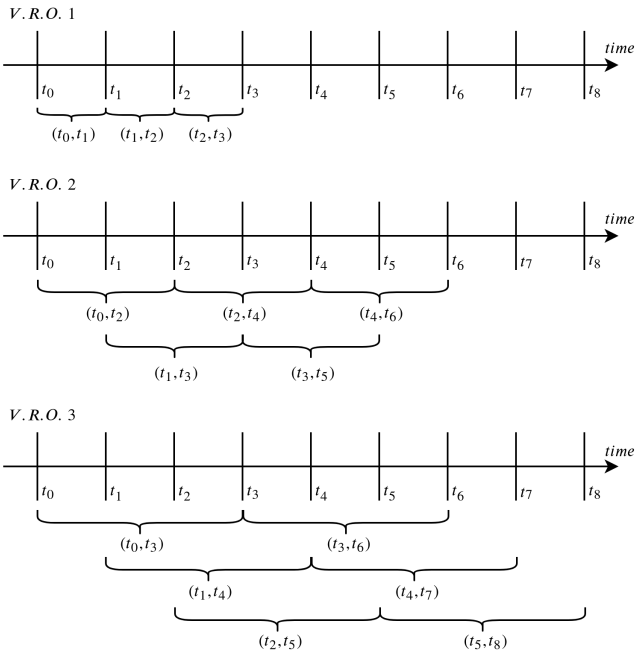
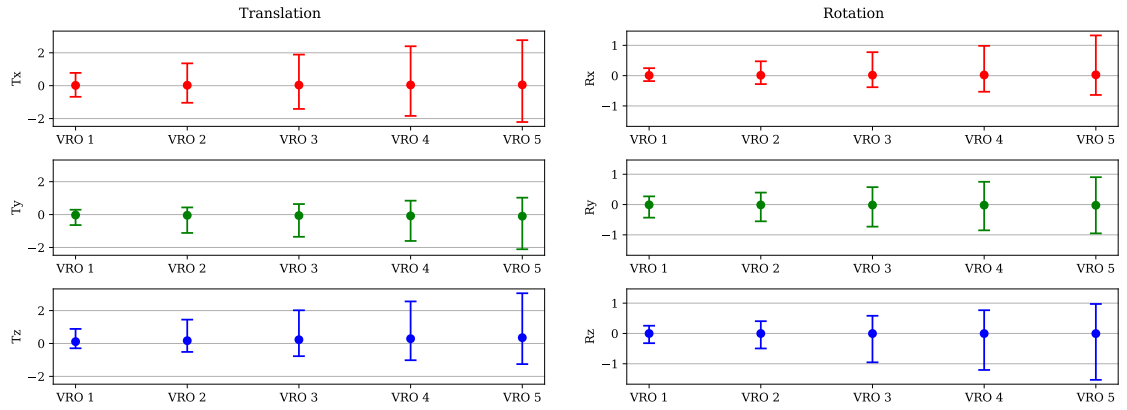


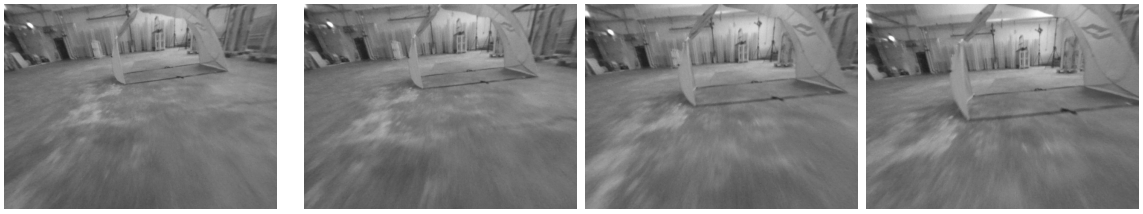
Figure 3.9: Dataset collection for different sensors

Figure 3.10 clearly shows that the ground-truth value distribution for Rx is much more compact compared to other motions.



(a) Translation (b) Rotation

Figure 3.10: Effects of VRO - Value distribution



(a) n (b) $n+1$ (c) $n+3$ (d) $n+5$

Figure 3.11: Effects of VRO (1 - 5)

(a) n (b) $n+6$ (c) $n+8$ (d) $n+10$

Figure 3.12: Effects of VRO (6 - 10)

3.1.2.3 Back to world (Camera-world)

$$^{W}\mathbf{T}_{C_2} = {}^{W}\mathbf{T}_{C_1} \cdot {}^{C_1}\mathbf{T}_{C_2} \tag{3.8}$$

$$\{ {}^{W}\mathbf{T}_{C_1}, {}^{W}\mathbf{T}_{C_2}, \cdots {}^{W}\mathbf{T}_{C_N} \} \tag{3.9}$$

3.1.2.4 Back to world (Body-camera)

$$^{W}\mathbf{T}_{B} = {}^{W}\mathbf{T}_{C} \cdot {}^{C}\mathbf{T}_{B} \tag{3.10}$$

$$\{ {}^{W}\mathbf{T}_{B_1}, {}^{W}\mathbf{T}_{B_2}, \cdots {}^{W}\mathbf{T}_{B_N} \} \tag{3.11}$$

3.2 Network architectures

Convolutional Neural Network (CNN)s have shown an amazing ability to recognize subtle features in images. Henceforth, all NN architectures tested in this thesis will be CNNs.

3.2.1 PoseNet

The network architecture used in this research was inspired by Tinghui Zhou et al. [11]. In that research, two networks were used to estimate the monocular depth and ego-motion separately. The encoder network from the ego-motion network was used as the starting point for the NN architecture. The selection of this architecture [11], was motivated by (a). the small form-factor of the encoder and (b). several researchers [24, 25, 26, 27] have used the same network architecture as their starting point to create networks that achieve state-of-the-art performance.

Convolution layers one and two have a kernel size of 7 and 5 respectively and all other convolutional layers have kernel size of 3, in the network. All convolution layers has a stride of 2 and ReLU activation except for the final layer. The output from the final convolution layer have the dimensions of (`batch-size, channels, height, width`), which then converted to (`batch-size, channels`) by taking the mean to height and width dimensions. All outputs of each layer are called feature maps (`fmap`). Such operation is unusual for CNNs, yet this allows, the network to take input images of any width and height, higher than what was used during training. How this effect the prediction accuracy is not investigated in this research. But it is an interesting topic to explore in the future.

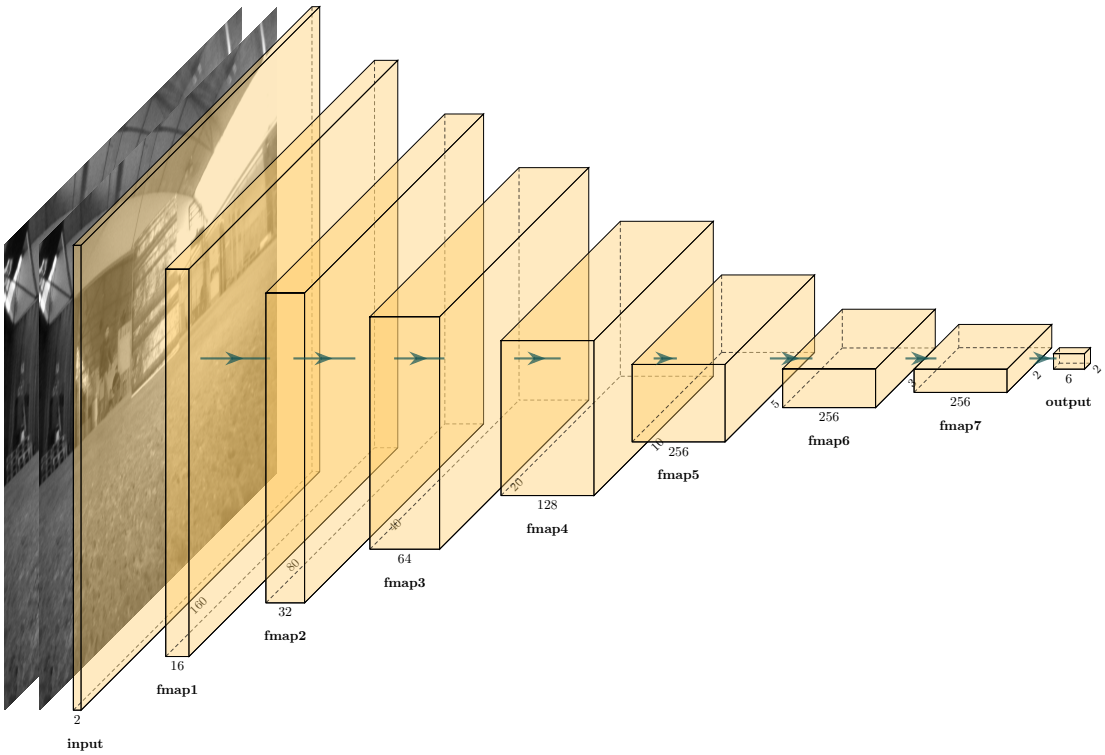


Figure 3.13: PoseNet architecture

3.2.2 PoseNet-L

PoseNet-L architecture is exactly similar to PoseNet until the 8th convolution layer. Thereafter, two fully connected (`fc`) layers each with 256 and 6 neurons respectively, were added to the network. These fc layers also consist of ReLU activation except layer one.

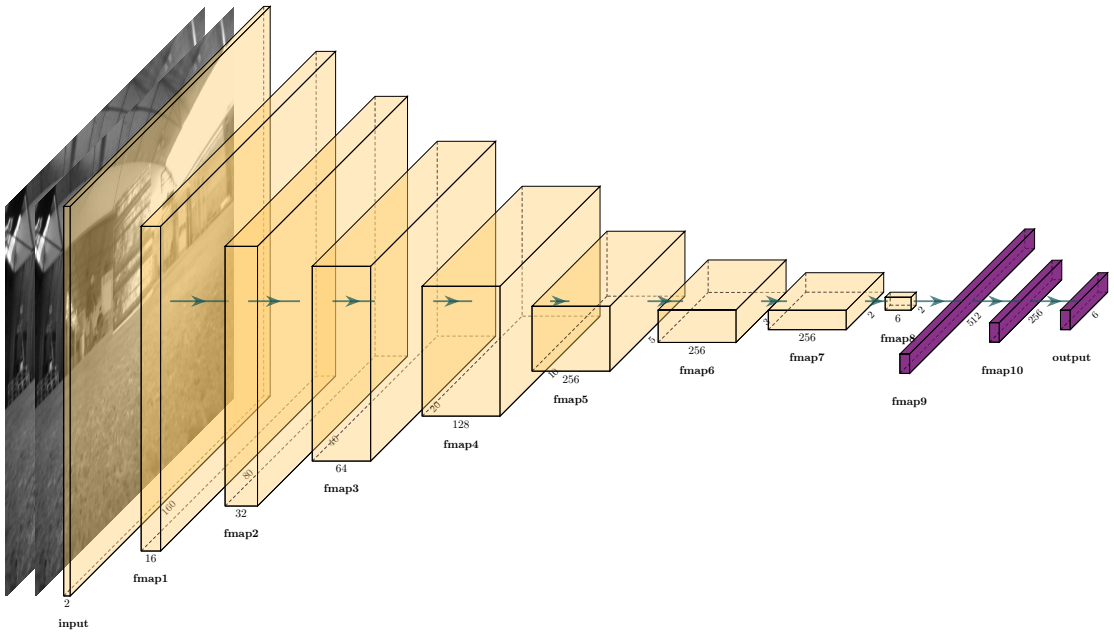


Figure 3.14: PoseNet-L architecture

3.2.3 VONet

VONet architecture was directly taken from work done by Mingqi Qiao et al.[4]. The unique aspect of this architecture is the processing of two input images separately and then converting the features maps into a single stream. This is somewhat similar to what is done in the traditional monocular VO pipeline where common features were detected between the two consecutive images to obtain the motion between them.

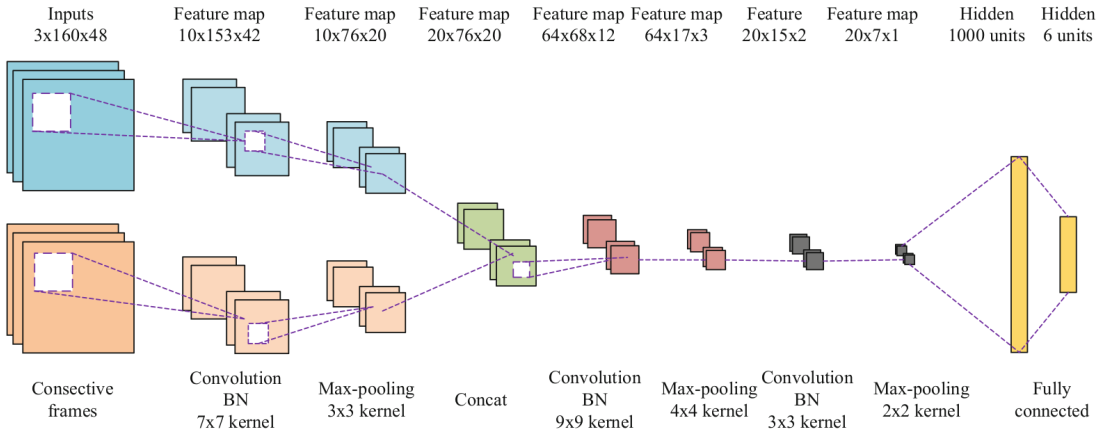


Figure 3.15: VONet architecture (Courtesy of [4])

3.2.4 PoseNet2

PoseNet2 is a novel architecture, which consists of eight convolution layers. The last convolution layer operates the same way as described in PoseNet. Convolution layers 1, 2, 3, 4 and 8 has a kernel size of 11, 9, 7, 5 and 1 respectively and the others have kernel size 3. Convolution layers 1, 3, 5 has max pooling and all layers consist of stride 1.

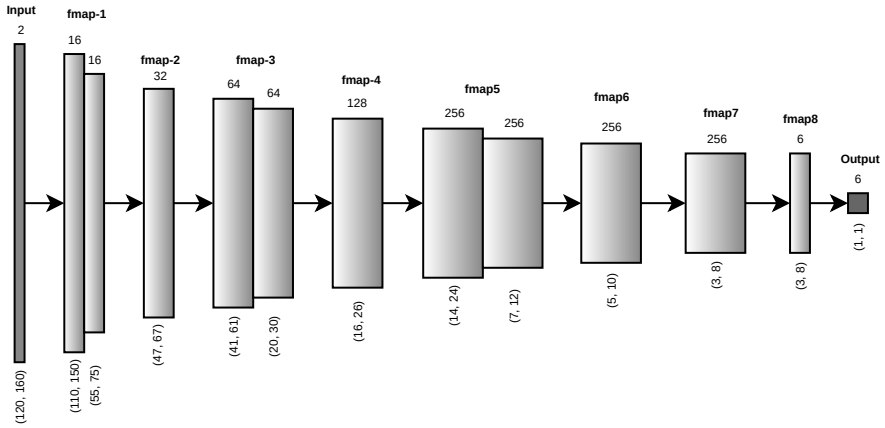


Figure 3.16: PoseNet2 architecture

3.2.5 PoseNet2-L

PoseNet2-L has the same architecture as PoseNet2 with the addition of a fully connected layer at the end.

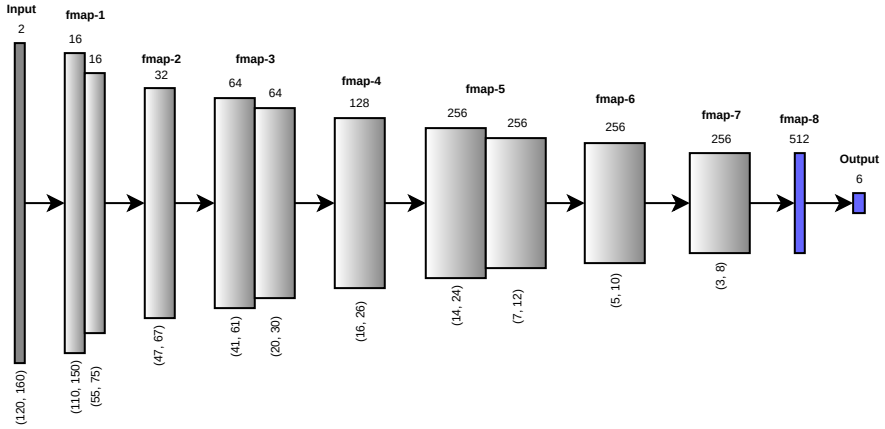


Figure 3.17: PoseNet2-L architecture

3.2.6 PoseNet2-TS

PoseNet2-TS is the Two Stream (TS) modification of the PoseNet2 architecture where the first three convolution layers are separated into two streams to process the two input images separately.

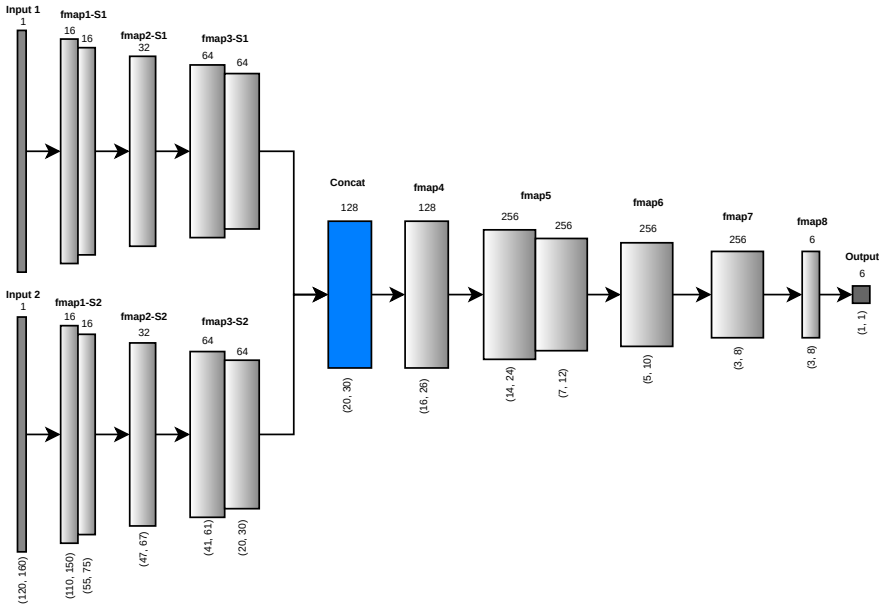


Figure 3.18: PoseNet2-TS architecture

3.3 Output representations

3.3.1 SE(3) pose

$$pose = [\ R, \vec{t}\ ] \tag{3.12}$$

Where, $R \in \mathbb{R}^{(3\times3)}$ and $\vec{t} \in \mathbb{R}^3$.

3.3.2 Euler pose

$$\vec{pose} = [\ tx, ty, tz, rx, ry, rz\ ] \tag{3.13}$$

Where, $\vec{pose} \in \mathbb{R}^6$.

3.4 Training

The system was implemented using the publicly available PyTorch [28] framework. During the training, we used the Adam [29] optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, learning rate of 0.0001 and mini-batch size of 32. The training typically converges after about 100 epochs. All the experiments are performed with image sequences captured with a monocular camera. The input images to the network were gray-scale, un-distorted and resized to 120×160 during training. This configuration is necessary also during inference. Mean Squared Error (MSE) between ground-truth and network prediction was used as the loss function. OpenCV [30] library was used for I/O operation of images. A learning rate finder[3] tool was used to find a suitable learning rate for different CNN architectures.

3.4.1 Hardware platforms

1. *Server:* This system is small server at the TU-Delft MAVLab. It consist of a octa-core Intel i7-7700 CPU with multi-threading, operating at 3.60GHz, and 16 GB of RAM. Ubuntu 16.04.4 LTS is used as the operating system. The server has two NVIDIA GTX-1080-Ti GPUs with driver version 384.130 and 11 GB VRAM, cuda version V9.0.176 and `Python 3.5.2`. This system was used for training purposes only.

[3] `https://github.com/davidtvs/pytorch-lr-finder`

2. *Laptop:* This system is a Asus Notebook FX503VD. It consist of a quad-core Intel Core i5-7300HQ CPU with multi-threading, operating at 2.50GHz, and 8 GB of RAM. Ubuntu 18.04.3 LTS is used as the operating system. The laptop has a NVIDIA GTX-1050 Max-Q GPU with driver version 430.50 and 2 GB VRAM, cuda version V9.0.176 and `Python 3.5.2`. This system was used for testing and evaluation purposes only.

Chapter 4

Results

4.1 Evaluation metrics

To evaluate the performance of the network, the estimated trajectories will be compared with the MSCKF [31] algorithm, which is a VIO based system. This comparison would not be a fair one, due to *Inertial* part being a huge advantage, VIO systems, perform much better compared to VO systems. Future comparisons should include purely visual algorithms, such as ORB-SLAM [12] and SVO [32], which were heavily tested and validated with datasets such as KITTI [15] and EuRoC [19]. It was not possible to implement these within the time frame of the thesis on the UZH Drone Race dataset [2].

For the evaluation of the final trajectory from the network, evo python package [33] and the UZH evaluation toolkit [34] were used, which will consist of the same sub-trajectory lengths for RPE estimation as the IROS 2019 FPV Drone Racing VIO Competition[1].

4.1.1 Absolute trajectory error

The absolute trajectory error directly measures the difference between points of the ground-truth and the final trajectory estimated by the algorithm [34]. Only translation information on the final trajectory are used for calculating ATE.

4.1.2 Relative pose error

Since frame-to-frame VO/VIO systems do not have a global reference (global initial position and attitude), the estimation quality can be evaluated by measuring the relative relations between the states at different times through-out the ground-truth and estimated trajectories [34]. Both translation and rotation information is separately used for calculating RPE. For the purpose of consistency, the relative pose errors at the sub-trajectory of lengths {40, 60, 80, 100, 120} meters are computed.

4.1.3 Prediction error rate

The prediction error rate looks at the correlation between the ground-truth value and the prediction error. This indicates the bias of the network towards the magnitude of the output values.

$$error = prediction - groundtruth \tag{4.1}$$

4.2 Network Analysis

For the analysis of the performance, two sequences were taken into account. Sequence 5 was available in the training set (consist of sequences 3, 5, 6, 7 and 9) and the other was not (sequence 10).

[1] `http://rpg.ifi.uzh.ch/uzh-fpv.html`

4.2.1 Network Architectures

In this section, the prediction accuracy of two CNN architectures will be compared. The learning rate of 1e-4, 100 epochs and VRO-1 dataset was used for all training scenarios for consistency.

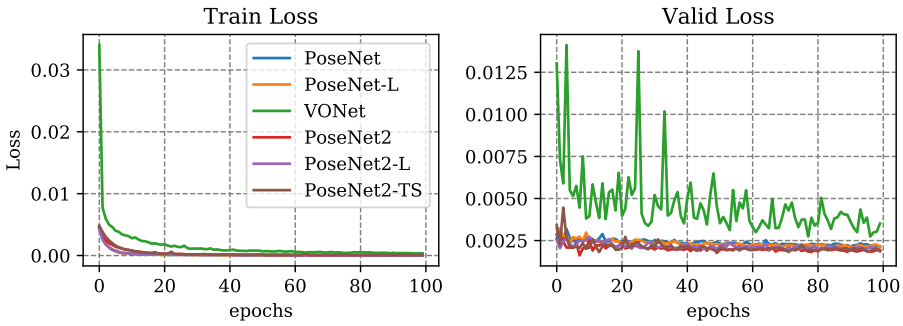
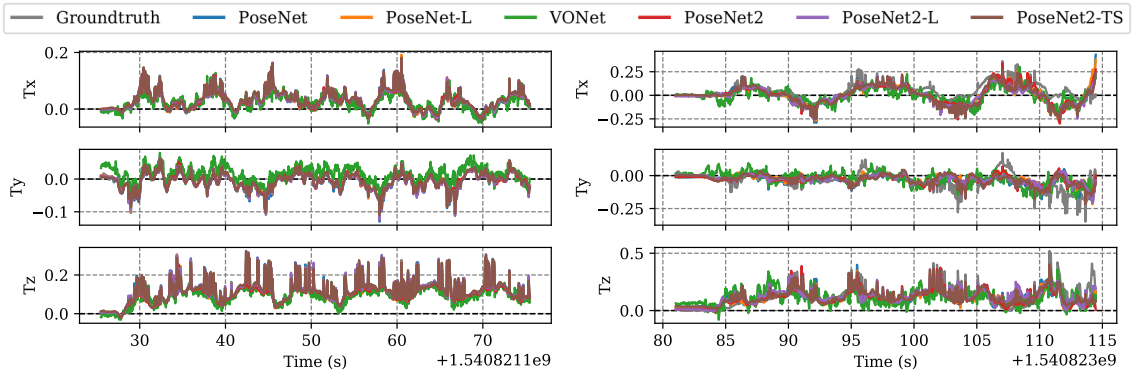
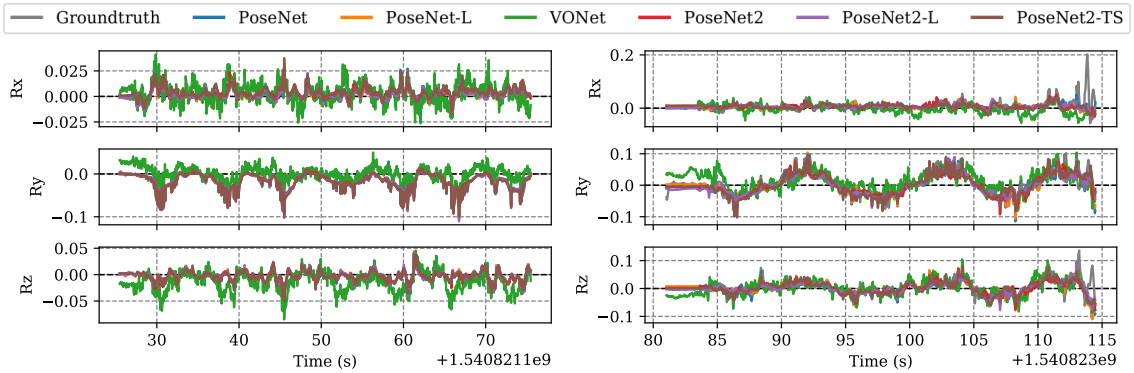


Figure 4.1: Network loss



(a) Sequence 5 (b) Sequence 10

Figure 4.2: Ground-truth vs network predictions (Translation)



(a) Sequence 5 (b) Sequence 10

Figure 4.3: Ground-truth vs network predictions (Rotation)

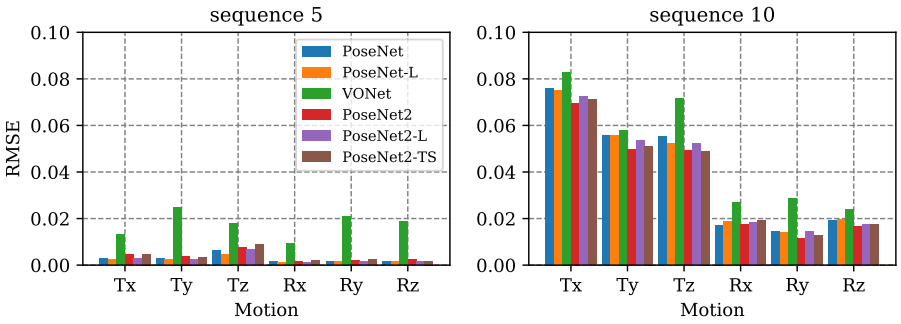
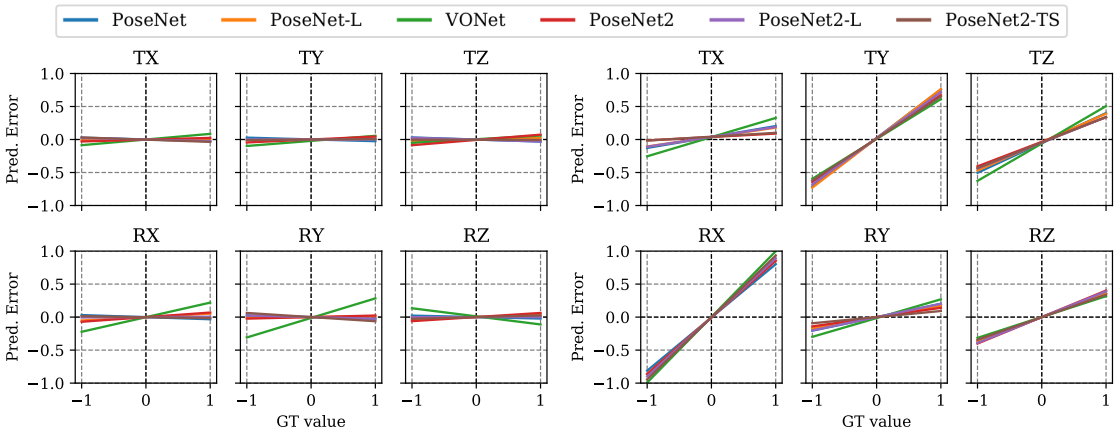


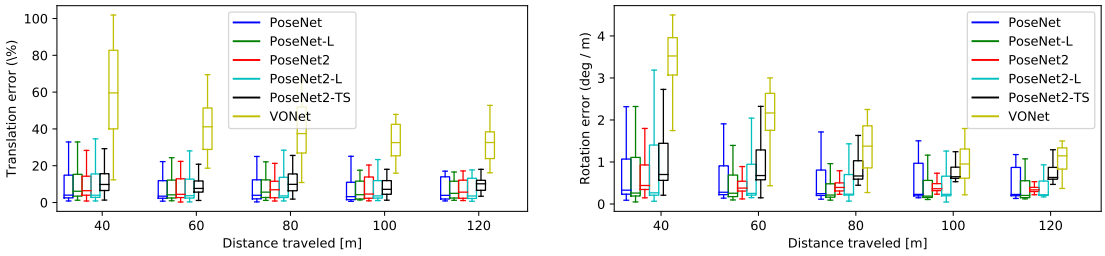
Figure 4.4: RMSE of motions for different models (GT vs prediction))



(a) Sequence 5 (b) Sequence 10

Figure 4.5: Prediction error rate

All models, struggle to learn the T_y and R_x motions. The higher prediction error rate for T_y stems from the fact that FPV pilots tend to use yaw motion for turning rather than roll. As for the higher error rate of R_x, it again comes down to the lack of motion in pitch.



(a) Overall translation error (percentage) (b) Overall rotation error (degrees per meter)

Figure 4.6: Overall relative pose error

Figure 4.6 indicate the overall (both sequence 5 and 10 combined) translation and rotation errors. For sequence wise (5, 10) errors, please refer to appendix A.2.1.

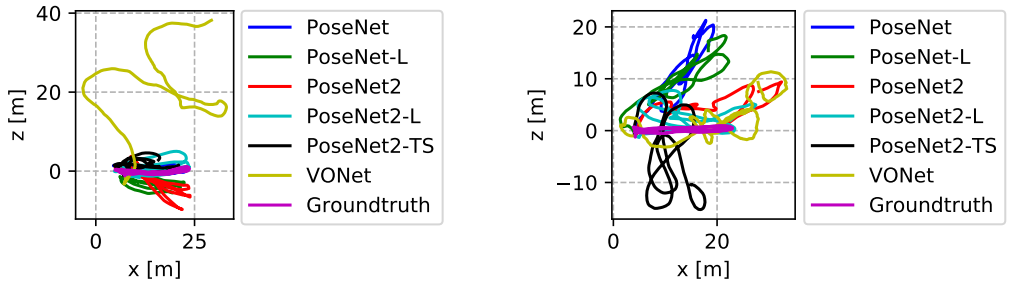
	SQ10	SQ5
PoseNet	13.847	**2.066**
PoseNet-L	15.025	3.903
PoseNet2	**11.166**	4.475
PoseNet2-L	16.389	2.366
PoseNet2-TS	11.475	7.136
VONet	13.663	32.535

ATE (RMSE)

	Translation (%)	Rotation (deg/meter)
PoseNet	**7.632**	0.601
PoseNet-L	7.821	**0.503**
PoseNet2	8.286	0.550
PoseNet2-L	8.657	0.588
PoseNet2-TS	10.346	0.898
VONet	42.860	1.959

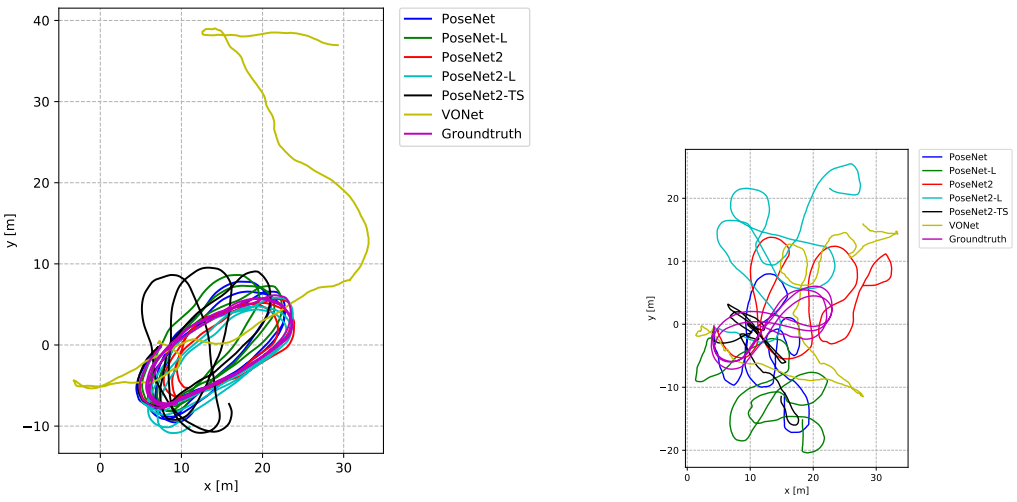
Relative pose error

Table 4.1: ATE/RPE for different network architectures



(a) Sequence 5 (b) Sequence 10

Figure 4.7: Trajectory (side view)



(a) Sequence 5 (b) Sequence 10

Figure 4.8: Trajectory (top view)

For sequence wise analysis, refer Appendix A.2.1

4.2.1.1 Profiling

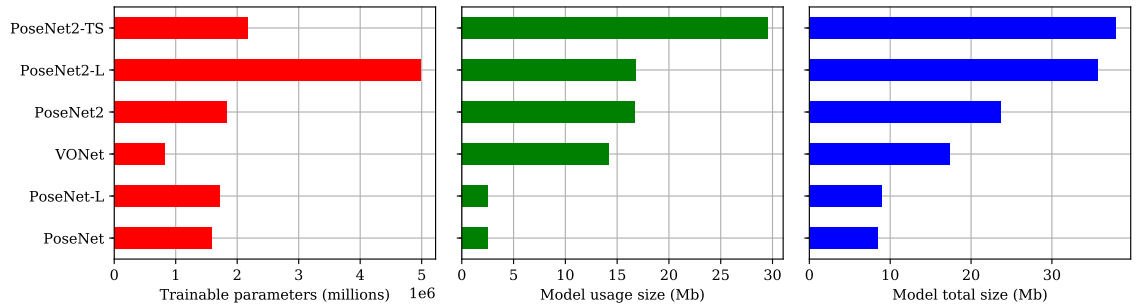


Figure 4.9: Model parameters and usage size

In figure 4.9, directly refer to the number of parameters inside the model. Model usage refers to the amount of memory is needed to run the model apart from trainable parameters, which is the memory needed to store the input images and the features maps, which directly depend on the size of the input images. The last graph refers to the combination of the first two graphs. The model size information in figure 4.9 was calculated using an open-source tool[2].

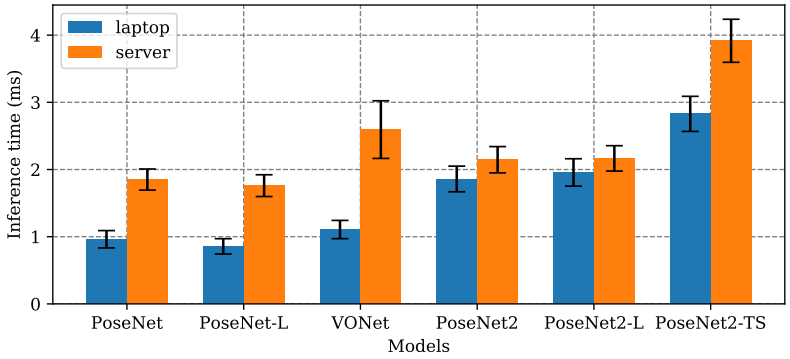


Figure 4.10: Inference time of different NN architectures (in different hardware platforms)

4.2.2 Effects of VRO

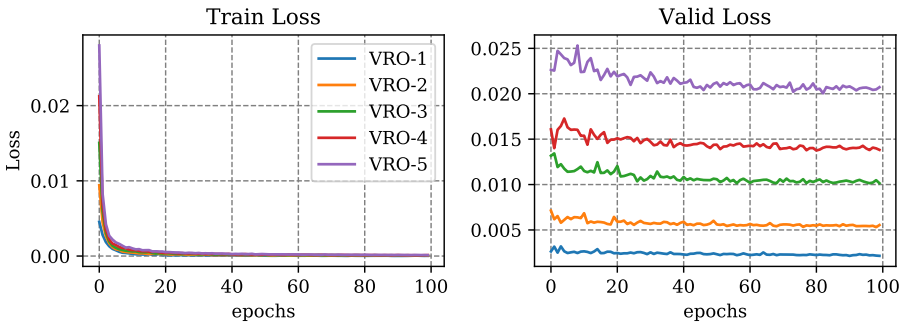


Figure 4.11: Network loss

[2] https://github.com/sksq96/pytorch-summary

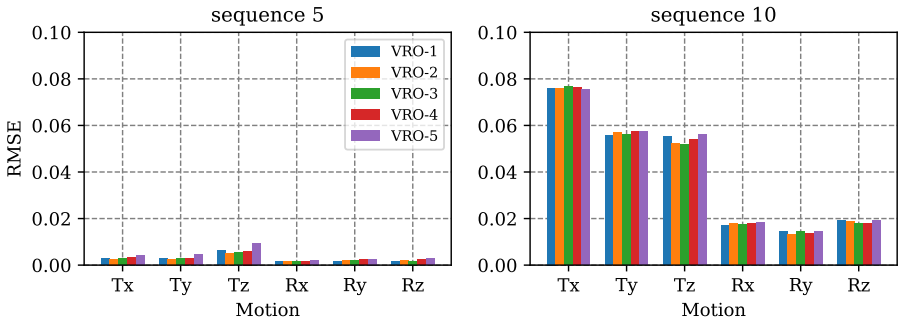
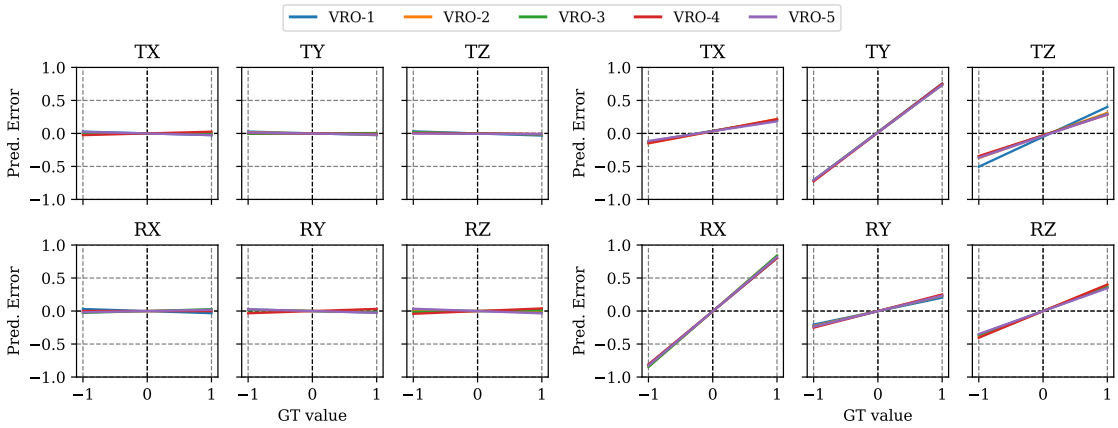


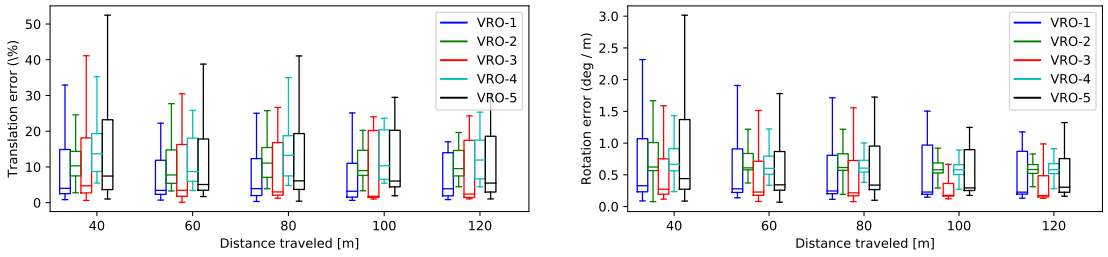
Figure 4.12: RMSE of motions for different VROs (GT vs prediction))



(a) Sequence 5

(b) Sequence 10

Figure 4.13: Prediction error rate



(a) Overall translation error (percentage)

(b) Overall rotation error (degrees per meter)

Figure 4.14: Overall relative pose error

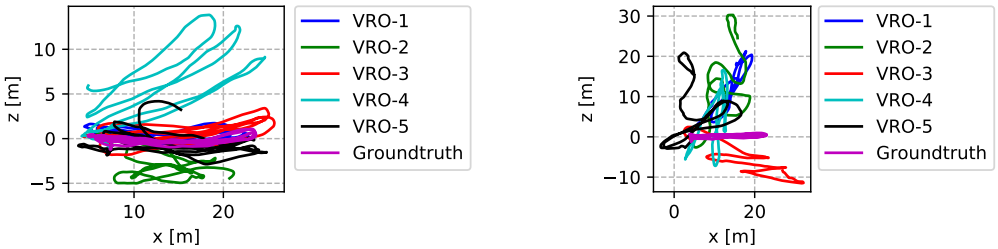
	SQ10	SQ5
VRO-1	**13.847**	**2.066**
VRO-2	15.868	7.980
VRO-3	18.594	2.268
VRO-4	26.809	13.490
VRO-5	29.313	15.270

ATE (RMSE)

	Translation (%)	Rotation (deg/meter)
VRO-1	7.632	0.601
VRO-2	11.698	0.783
VRO-3	**9.618**	**0.480**
VRO-4	24.631	1.720
VRO-5	25.467	1.887

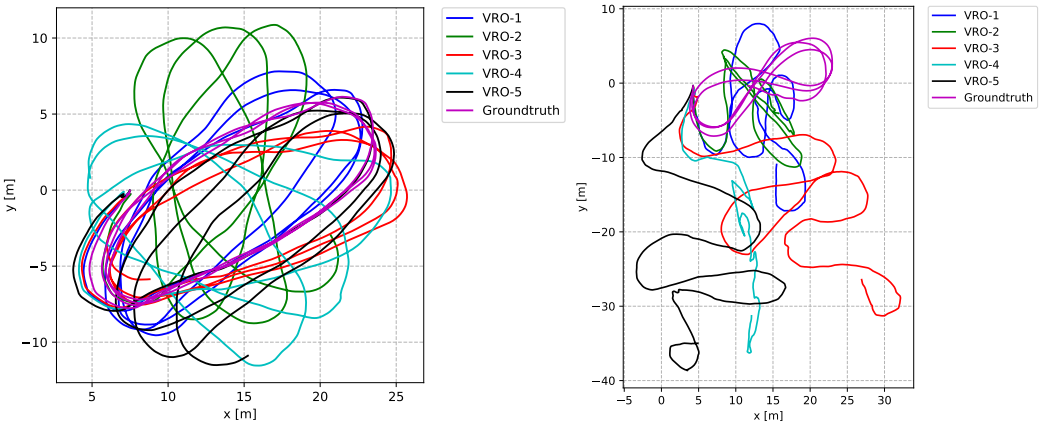
Relative pose error

Table 4.2: ATE/RPE for different VRO values



(a) Sequence 5 (b) Sequence 10

Figure 4.15: Trajectory (side view)

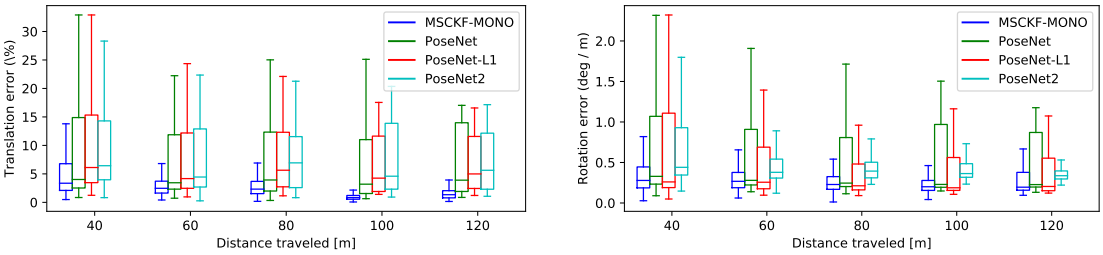


(a) Sequence 5 (b) Sequence 10

Figure 4.16: Trajectory (top view)

4.2.3 Comparison with open source VIO systems

MSCKF [31] works in both stereo and monocular modes, yet only the monocular version was used in this research for analysis. Since PoseNet and PoseNet-L models seem to perform the best, only those two models will be compared against MSCKF. For this analysis, MSCKF implemented on the OpenVINS [35] framework was used, which won the IROS 2019 FPV Drone Racing VIO Competition[3].



(a) Overall translation error (percentage) (b) Overall rotation error (degrees per meter)

Figure 4.17: Overall relative pose error

[3] http://rpg.ifi.uzh.ch/uzh-fpv.html

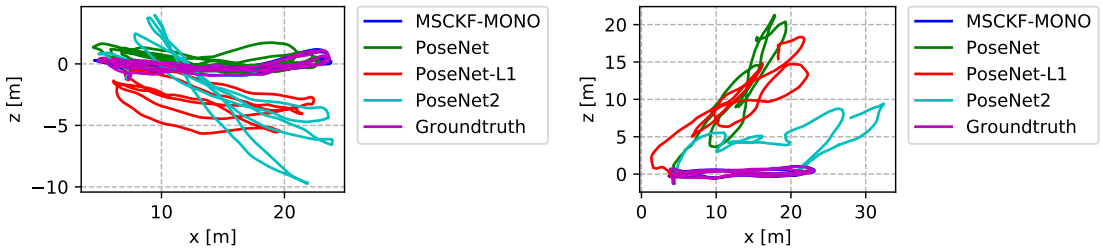
	SQ10	SQ5
MSCKF-MONO	**0.463**	**0.307**
PoseNet	13.847	2.066
PoseNet-L1	15.025	3.903
PoseNet2	11.166	4.475

ATE (RMSE)

	Translation (%)	Rotation (deg/meter)
MSCKF-MONO	**2.919**	**0.348**
PoseNet	7.632	0.601
PoseNet-L1	7.821	0.503
PoseNet2	8.286	0.550

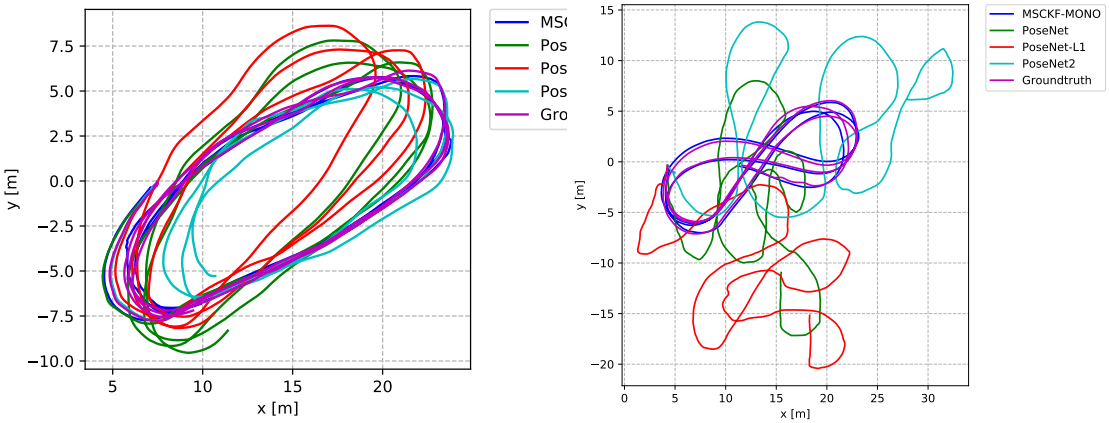
Relative pose error

Table 4.3: ATE/RPE for different VO/VIO systems



(a) Sequence 5 (b) Sequence 10

Figure 4.18: Trajectory (side view)



(a) Sequence 5 (b) Sequence 10

Figure 4.19: Trajectory (top view)

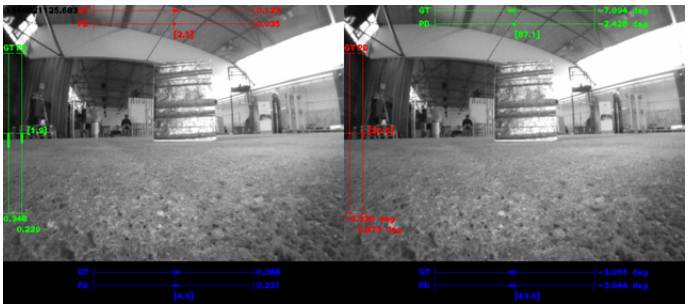


Figure 4.20: Prediction overlay on image sequence

Note: A video overlay of ground-truth vs Network predictions can be accessed through the following link `https://tinyurl.com/sameera-msc`.

Chapter 5

Discussion

5.1 Network Analysis

5.1.1 Network Architectures

Figure 4.1 indicates that both models do over-fit to the dataset quite fast (within 50 epochs). Figures 4.7 and 4.8 shows the output trajectory of the networks compared with the ground truth. The low generalization of networks can be clearly seen with the trajectory of sequence 10.

Accuracy wise, VONet seems to be the worst model. All other architectures perform consistently at the same level. PoseNet2 performs the best which is evident by the low RMSE value (Fig. 4.4) and lower prediction error rate (Fig. 4.5). PoseNet2-L and PoseNet2-TS models perform the same level as PoseNet2. Figure 4.2 and 4.3 shows that even though, the accuracy of the final trajectory (Fig. 4.7, 4.8) is low, the models seem to following the general direction of the motions, except VONet. However, when the ATE/RPE errors are considered (Table 4.1), PoseNet2 models are the most generalized model and, PoseNet and PoseNet-L models seem to perform well in RPE error overall.

5.1.1.1 Profiling

Since one of the objectives of this research is to discover a model that can operate in a resource-constrained environment, it is important to analyze both model size and their operation speed. Figure 4.9 shows that VONet architecture is the smallest in size (around 0.8 million parameters), yet it requires the 4th larges amount of operation memory. Therefore, overall, PoseNet and PoseNet-L architectures are the most suitable for the resource-constrained environment.

Figure 4.10 shows the execution times of each network architecture during inference for two different hardware platforms. All tested models run under 4 milliseconds per pair of images which means these models can run at least 250 Hz rate (theoretical maximum at 1000 Hz). To put this speed into perspective, during the AlphaPilot challenge, the NN based gate detection network (developed by MAVLab) ran at 50 Hz on an NVIDIA Jetson AGX Xavier[1] system. This speed was enough because the average speed of the drone was at around 10-12 m/s and the control loop ran at 1000 Hz.

Even though the server is more powerful compared to the laptop, all models consistently run slower there. The hardware platforms used for this profiling contain many more resources compared to an on-board computer of a racing drone. A deeper analysis would be to run the models on NVIDIA Jetson TX2[2] and JeVois smart machine vision camera systems [36], both of which are possible to be attached to a racing drone. This was not possible due to the time constraint of the thesis and both of these systems have their own NN frameworks (NVIDIA TensorRT[3] and DarkNet [37] respectively).

Considering the overall performance, PoseNet and PoseNet-L seem to be the best option for a NN based VO system to be executed on-board a drone.

[1] `https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit`

[2] `https://developer.nvidia.com/embedded/jetson-tx2`

[3] `https://developer.nvidia.com/tensorrt`

5.1.2 Effects of VRO

At first, the evidence obtained by observing ground-truth vs prediction RMSE (Figure 4.12) prediction error rate (Figure 4.13) suggests that the VRO data augmentation method does not have any effect on the network predictions. However, the ATE and RPE (Tabel 4.3) values suggest that VRO-3 performs better compared to other VRO values. Furthermore, this observation is confirmed by the final trajectories (Figure 4.15 and 4.16) for each VRO.

5.1.3 Comparison with open source VIO systems

MSCKF clearly dominate in all sectors of analysis. However, an interesting observation is that overall, the PoseNet architectures perform on par with MSCKF (RPE of Table 4.3). Furthermore, the lack of generalization of NN models is evident from the ATE sub-table of table 4.3.

Chapter 6

Conclusion

The purpose of this thesis was to investigate the possibility of accurate estimation of the frame-to-frame ego-motion using neural networks. The input to the network was two consecutive images capture by the camera and the relative pose between those images would be the output of the network.

As the results indicate in chapter 4, the network seems to learn and understand the motions it is required to learn, with a lower level of generalization. Therefore, more research needs to be done to further improve the accuracy of the network.

An ideal VO system would be if its prediction errors are withing a normal distribution with $\mu = 0$. However, a system would still be useful, if its prediction error bias is deterministic.

6.1 Limitations

The main output representation tested in this research is the ***Euler pose***, which contain three variables to represent the translation motion (t_x, t_y and t_z respectively). The existence of t_z in the pose vector means that the networks has to estimate motion in the direction of depth (in camera frame) as well. This is widely known to be difficult for a monocular system to do. Especially in absolute units (meters), which is the method used in this research. However, this allows the reconstruction of the final trajectory directly using the network predictions and rigid body transformation. Hence, the current network outputs heavily depend on the specific features it learnt about the environment, in the dataset.

Since, this research has shown that the network understands the basic motions required for visual odometry, a more practical way to utilize networks ability is to change the pipeline such that the output is reformed to $(t_x, t_y, r_x, r_y, r_z)$ where t_x and t_y are normalized to a unit vector.

This changes the position of the neural network in the VO based flight controller pipeline, where the outputs of the network has to be converted to 3D motion values by using another algorithm such as *Perspective-n-Points (PnP)* (fig. 2.5).

6.2 Future directions

An analysis should be conducted to determine which out of the six unique motions, contribute to the most and the least prediction errors. Since this research was conducted using the drone race dataset [2] from UZH, all the motions were simultaneously available through all sequences. Hence, it was not possible to isolate these motions out of this dataset. Therefore to truly understand how each motion affects the prediction error, we need to collect different data that contains each of these motions in isolation. This dataset should also include an isolated sequence that does not contain any motion (hovering or stationary on the ground), which will help us to determine how the network would handle such situations. However, we should keep in mind even VIO algorithms such as MSCKF the [31] requires the drone to have motion to produce accurate pose estimation results. Hence, it would be an interesting analysis.

6.2.1 Further work

- Add drone dynamics and geometric loss to the loss function.

- In alignment, use **weighted interpolation** instead of closest sample.
- Train the same neural network with the EuRoC dataset [19] to determine how it performs in lower average drone speeds.
- Consider more than 2 consecutive input images (*Refer to section 6.2.2*).
- Use a NN interoperability analysis tool such as Captum [38] to analyse the network on a deeper level (layer wise).
- Use a simulator to test with an actual simulator such as MIT Flight Goggles [1].
- Implement the system to work as self-supervised learning system.
- Change the architecture of the network to predict a probability distribution [13], rather than predicting a real value (regression).
- Due to the alignment process (3.1.1.1), a large number of samples were lost, where the drone is stationary. Therefore, the network has never seen the stationary motion before. Therefore, at the beginning and end of a sequence, the prediction error increases rapidly.
- The UZH dataset does not contain any other moving objects in the image sequences. Therefore, at this state, the network will assume that the drone is the only moving object in this environment, hence complying with the static world assumption. In the research conducted by Tinghui Zhou et al. [11], they mitigate this problem by introducing an explainability mask which indicates the areas of the original input image where the static world assumption was broken.

6.2.2 New architecture with multi consecutive input images

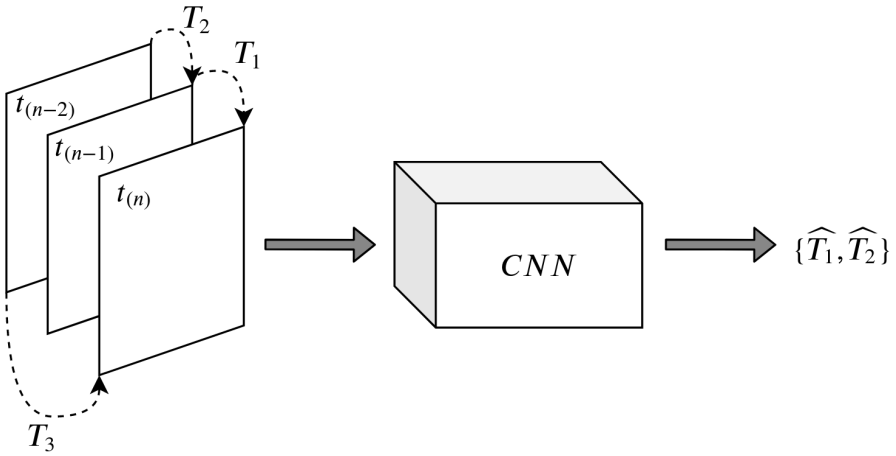


Figure 6.1: Multi input architecture

$$\mathbf{T}_1 = {}^{t_{n-1}}\mathbf{T}_{t_n}$$
$$\mathbf{T}_2 = {}^{t_{n-2}}\mathbf{T}_{t_{n-1}}$$
$$\mathbf{T}_3 = {}^{t_{n-2}}\mathbf{T}_{t_n}$$

$$\mathbf{T}_3 = {}^{t_{n-2}}\mathbf{T}_{t_{n-1}} \cdot {}^{t_{n-1}}\mathbf{T}_{t_n} = \mathbf{T}_2 \cdot \mathbf{T}_1$$

Therefore,

$$\widehat{\mathbf{T}_3} = \widehat{\mathbf{T}_2} \cdot \widehat{\mathbf{T}_1}$$

Three error values can be derived using the above equations;

$$
\begin{aligned}
err_1 &= \mathbf{T}_1 - \widehat{\mathbf{T}_1} \\
err_2 &= \mathbf{T}_2 - \widehat{\mathbf{T}_2} \\
err_3 &= \mathbf{T}_3 - \widehat{\mathbf{T}_3}
\end{aligned}
$$

These three error values induce a certain temporal correlation between multiple consecutive frames, which penalize the network to predict accurate relative pose values.

Bibliography

[1] W. Guerra, E. Tal, V. Murali, G. Ryou, and S. Karaman, "FlightGoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, Nov. 2019.

[2] J. Delmerico, T. Cieslewski, H. Rebecq, M. Faessler, and D. Scaramuzza, "Are We Ready for Autonomous Drone Racing? The UZH-FPV Drone Racing Dataset," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 6713–6719, May 2019.

[3] E. Riba, D. Mishkin, D. Ponsa, E. Rublee, and G. Bradski, "Kornia: an open source differentiable computer vision library for pytorch," in *Winter Conference on Applications of Computer Vision*, 2020.

[4] M. Qiao and Z. Wang, "Learning the frame-2-frame ego-motion for visual odometry with convolutional neural network," in *Computer Vision* (J. Yang, Q. Hu, M.-M. Cheng, L. Wang, Q. Liu, X. Bai, and D. Meng, eds.), (Singapore), pp. 500–511, Springer Singapore, 2017.

[5] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017.

[6] Y. Cheng, M. Maimone, and L. Matthies, "Visual odometry on the mars exploration rovers," vol. 1, pp. 903 – 910 Vol. 1, 11 2005.

[7] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Neural Information Processing Systems*, vol. 25, 01 2012.

[8] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[9] M. Irani, B. Rousso, and S. Peleg, "Recovery of ego-motion using image stabilization," pp. 454 – 460, 07 1994.

[10] J. Delmerico and D. Scaramuzza, "A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots," 05 2018.

[11] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6612–6619, July 2017.

[12] M. J. M. M. Mur-Artal, Raúl and J. D. Tardós, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[13] S. Pillai and J. J. Leonard, "Towards visual ego-motion learning in robots," *CoRR*, vol. abs/1705.10279, 2017.

[14] M. Lee and C. C. Fowlkes, "Cemnet: Self-supervised learning for accurate continuous ego-motion estimation," *CoRR*, vol. abs/1806.10309, 2018.

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

[16] R. Murray, Z. Li, and S. Sastry, "A mathematical introduction to robot manipulation," vol. 29, 12 2010.

[17] T. D. Barfoot, *State Estimation for Robotics*. Cambridge University Press, 2017.

[18] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms and Implementation*. 01 2005.

[19] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, 2016.

[20] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, "Robust stereo visual inertial odometry for fast autonomous flight," *CoRR*, vol. abs/1712.00036, 2017.

[21] A. L. Majdik, C. Till, and D. Scaramuzza, "The Zurich Urban Micro Aerial Vehicle Dataset," 2017.

[22] A. Antonini, W. Guerra, V. Murali, T. Sayre-McCord, and S. Karaman, "The blackbird dataset: A large-scale dataset for uav perception in aggressive flight," in *2018 International Symposium on Experimental Robotics (ISER)*, 2018.

[23] P. I. Corke, *Robotics, Vision & Control: Fundamental Algorithms in MATLAB*. Springer, second ed., 2017. ISBN 978-3-319-54413-7.

[24] Z. Yin and J. Shi, "Geonet: Unsupervised learning of dense depth, optical flow and camera pose," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[25] R. Mahjourian, M. Wicke, and A. Angelova, "Unsupervised learning of depth and ego-motion from monocular video using 3d geometric constraints," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[26] H. Zhan, R. Garg, C. Saroj Weerasekera, K. Li, H. Agarwal, and I. Reid, "Unsupervised learning of monocular depth estimation and visual odometry with deep feature reconstruction," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[27] T. Shen, Z. Luo, L. Zhou, H. Deng, R. Zhang, T. Fang, and L. Quan, "Beyond photometric loss for self-supervised ego-motion estimation," *CoRR*, vol. abs/1902.09103, 2019.

[28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[30] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

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

[32] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast semi-direct monocular visual odometry," 05 2014.

[33] M. Grupp, "evo: Python package for the evaluation of odometry and slam.." `https://github.com/MichaelGrupp/evo`, 2017.

[34] Z. Zhang and D. Scaramuzza, "A tutorial on quantitative trajectory evaluation for visual(-inertial) odometry," in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2018.

[35] P. Geneva, K. Eckenhoff, W. Lee, Y. Yang, and G. Huang, "Openvins: A research platform for visual-inertial estimation," in *Proc. of the IEEE International Conference on Robotics and Automation*, (Paris, France), 2020.

[36] L. Itti, "JeVois Smart Machine Vision Camera." http://jevois.org/, 2016.

[37] J. Redmon, "Darknet: Open source neural networks in c." http://pjreddie.com/darknet/, 2013–2016.

[38] N. Kokhlikyan, V. Miglani, M. Martin, E. Wang, J. Reynolds, A. Melnikov, N. Lunova, and O. Reblitz-Richardson, "Pytorch captum." https://github.com/pytorch/captum, 2019.

Appendix A

Supplementary material

A.1 Data Augmentation

A.1.1 VRO

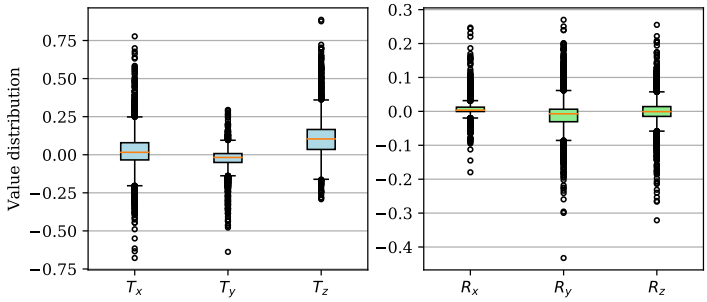


Figure A.1: Ground-truth value distribution - VRO 1 (Indoor)

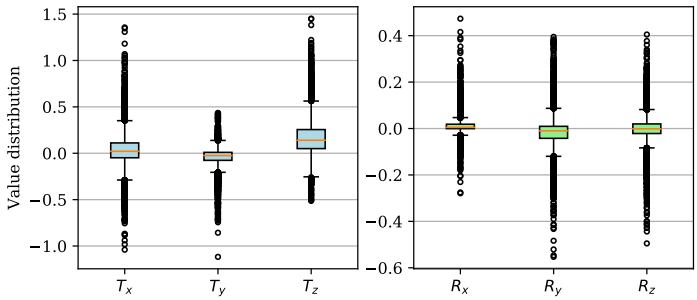


Figure A.2: Ground-truth value distribution - VRO 2 (Indoor)

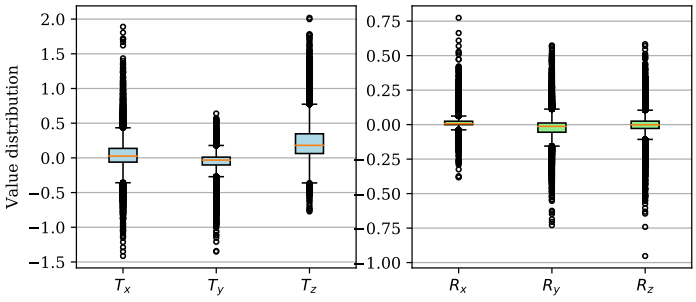


Figure A.3: Ground-truth value distribution - VRO 3 (Indoor)

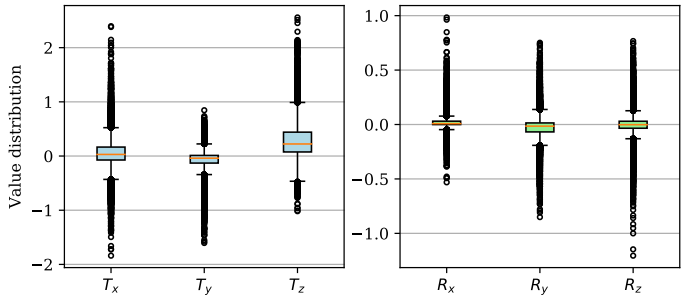


Figure A.4: Ground-truth value distribution - VRO 4 (Indoor)

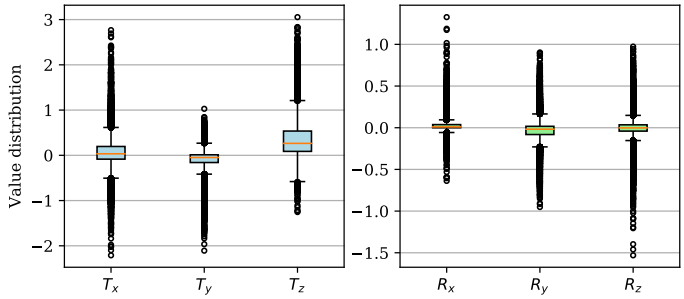


Figure A.5: Ground-truth value distribution - VRO 5 (Indoor)

A.2 Analysis

A.2.1 Model Comparison

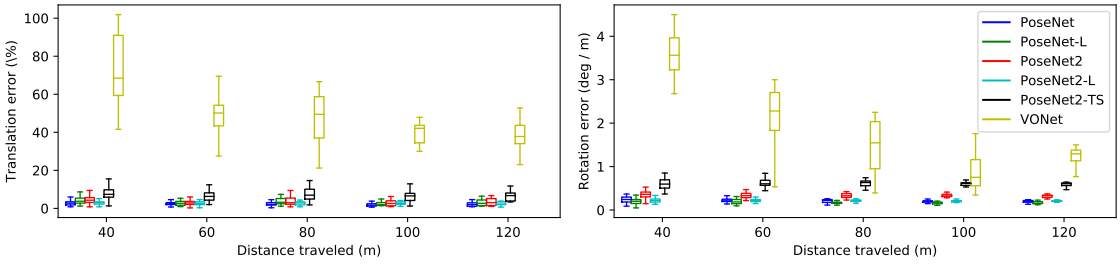


Figure A.6: Translation and rotation error (Sequence 5)

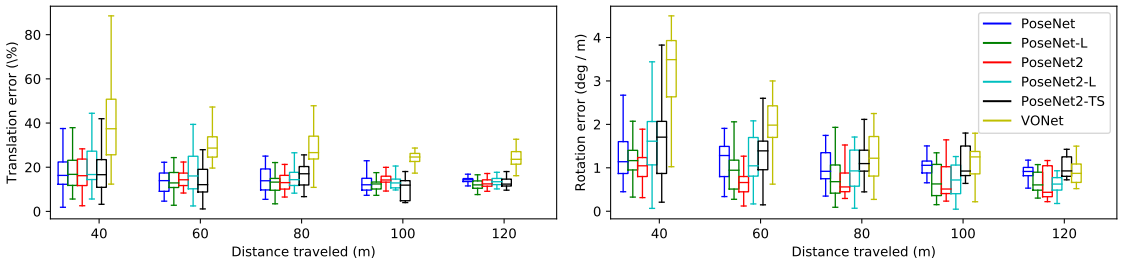


Figure A.7: Translation and rotation error (Sequence 10)

Notes

- Integration of my network output does not have to produce an accurate trajectory. Why?, because outputs of my network are used to remove the accumulated drift caused by IMUs that induce problems for traditional drone controllers. Therefore, this network will be used in a real-time manner in parallel with a traditional controller. Therefore, the final trajectory of the drone does not directly or completely depend on the network's output. Hence, it only has to produce generalized, accurate results on a real-time basis to assist the traditional controller.
- Networks ability to learn and distinguish rotation motions (degrees) and translation motions (meters) separately cannot be analyzed with this dataset. For that, a dataset with isolated motions is needed. The RMSE value difference in figure 4.4 between rotations and translation could be due to value distribution discrepancies rather than motion themselves.