

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Ego Motion from Video Data in Autonomous Vehicles

João Basto do Rosário



Mestrado em Engenharia Informática

Supervisor: Andry Maykol Pinto

July 18, 2023

Ego Motion from Video Data in Autonomous Vehicles

João Basto do Rosário

Mestrado em Engenharia Informática

July 18, 2023

Resumo

Os veículos autônomos têm o potencial para melhorar significativamente a segurança rodoviária e reduzir o congestionamento de tráfego. Para que isto possa acontecer, é crucial uma percepção precisa do ambiente que o rodeia. Esta percepção permite aos veículos tomar ações adequadas com base na situação em que se encontram. Uma parte essencial desta percepção é a estimativa da pose do veículo, tarefa que pode ser realizada através do uso de técnicas de odometria visual, que consistem na estimativa da pose 6D usando entradas visuais. Esta dissertação centra-se na melhoria destas técnicas, especificamente através da aplicação de modelos de transformers para processar dados de câmeras e optical flow.

No âmbito desta dissertação foram desenvolvidos três modelos: PoseFormer2D, PoseFormer3D e Spatio-Temporal PoseFormer. Tanto o PoseFormer2D como o PoseFormer3D podem utilizar codificações de posição aprendíveis ou fixas e utilizam um único bloco de transformador para calcular a atenção nas dimensões espaciais e temporais. Em contraste, o Spatio-Temporal PoseFormer calcula valores de atenção independentes para o espaço e o tempo, possibilitando um foco mais granular. Todos os modelos também podem incorporar optical flow além de imagens para integrar informações de movimento na estimativa da pose. Os modelos foram treinados com imagens RGB vindas do dataset público KITTI, e com o optical flow calculado com o modelo RAFT.

Os resultados dos modelos de transformers propostos, em condições normais de tráfego, ultrapassam o DeepVO em cerca de 85%, e são capazes de ultrapassar o ORB-SLAM3 na sequência 01 do conjunto de dados KITTI. Os modelos são capazes de fornecer estimativas precisas de pose 6D em situações simples e complexas onde existem objetos dinâmicos na cena.

Abstract

Autonomous vehicles hold the potential to significantly enhance road safety and reduce traffic congestion. To realize these benefits, an accurate perception of the surrounding environment is crucial. This perception allows vehicles to take appropriate actions based on the current situation. A key part of this perception is the estimation of the vehicle's pose relative to its environment, a task that can be achieved using visual odometry techniques, which allow for the estimation of 6D pose by using visual inputs. This dissertation focuses on improving these techniques, specifically through the application of transformer models to process camera and optical flow data.

Three models were developed: PoseFormer2D, PoseFormer3D, and Spatio-Temporal PoseFormer. Both PoseFormer2D and PoseFormer3D can take advantage of either learnable or fixed position encodings and utilize a single transformer block to compute attention across spatial and temporal dimensions. Conversely, Spatio-Temporal PoseFormer calculates attention values independently for space and time, enabling a more granular focus. This means that Spatio-Temporal PoseFormer has two independent transformer blocks which calculate the attention separately. All models can also incorporate optical flow in addition to images to integrate motion information into the pose estimation. The models were trained using stereo RGB images sourced from the public KITTI visual odometry dataset and with the optical flow calculated from the images using the RAFT model.

The results of the proposed transformer models, in normal traffic conditions, surpass DeepVO by around 85%, and are capable of surpassing ORB-SLAM3, a state-of-the-art geometric approach, in sequence 01 of the KITTI dataset. The models are capable of giving accurate estimates of 6D pose in simple and complex situations where there are dynamic objects in the scene.

Agradecimentos

Ao meu orientador, Andry Maykol Pinto, quero agradecer por me ter aceite para ser seu aluno e ter oportunidade de realizar o projeto a que esta tese estava ligada. Quero também agradecer por estar sempre disponível e conseguir tirar o melhor de mim.

Ao Pedro Nuno e à Maria Inês gostava de agradecer pelas reuniões de segunda feira, que embora me deixavam com mais dúvidas quando saía delas do que quando entrava, foram fundamentais para conseguir concluir a tese.

A todos as pessoas que tive o prazer de conhecer durante a faculdade, sejam professores ou colegas, quero agradecer por terem feito da faculdade uma ótima experiência.

Às pessoas que tive o prazer de conhecer na Bosch quero agradecer a paciência e a flexibilidade que sempre tiveram comigo. Gostava particularmente de agradecer ao Lucas, Diogo e Bernardo que me ajudaram imenso com todas as dúvidas que foram surgindo durante a tese.

Aos meus amigos mais próximos, quero agradecer por todos os momentos, conversas aleatórias e parvoíces que tive a felicidade de partilhar ao longo dos últimos 10 anos.

À minha namorada, Beatriz, quero agradecer, não só pelos momentos, conversas aleatórias e parvoíces, mas também por todo o carinho e apoio que me deu durante estes últimos 5 anos.

Por fim gostava de agradecer à minha família pelos valores que me inculcaram, desde pequeno, que me ajudou a ser quem sou hoje. Aos meus pais quero também agradecer por me darem todo o apoio do mundo, as oportunidades que me deram, sem elas não conseguia estar aqui.

João Basto do Rosário

Official Acknowledgements

This work was developed with the support of the host institution: BOSCH Portugal, and supervised at the institution by Bernardo Amaral.

This dissertation was supported by the European Structural and Investment Funds in the FEDER component, through the Operational Competitiveness and Internalization Programme (COMPETE 2020) [Project nº 047264; Funding Reference: POCI-01-0247-FEDER-047264]

João Basto do Rosário

“I’m a great believer in luck, and I find the harder I work, the more I have of it.”

Thomas Jefferson

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivation	2
1.3	Objectives	2
1.4	Document Structure	3
2	State of the Art	4
2.1	Visual Odometry	5
2.1.1	Geometric Methods	6
2.1.2	Learning Based Methods	15
2.1.3	Vision Transformers	26
2.2	Critical Review	31
3	Visual PoseFormer	33
3.1	Introduction	33
3.2	Data Processing	34
3.3	Model architecture	35
3.4	Loss Function	38
3.5	Optimizer	40
4	Experiments	41
4.1	Training and code details	41
4.2	Results	42
4.2.1	Training	42
4.2.2	Ego Motion Trajectories	45
4.2.3	Inference Time	54
5	Conclusions and future work	55

List of Figures

2.1	Types of Self Location, particularly, the different types of odometry.	5
2.2	Visual Odometry methodologies.	6
2.3	Main pipelines of geometric – feature-based VO [68]. Inside the dotted lines is the representation of the motion estimation algorithms that can be applied to the selected features.	7
2.4	Example of feature point extraction and matching using SIFT. ¹	7
2.5	Sequential pictures being taken from a moving camera. By tracking the motion of the chosen feature points (points in red), it is possible to solve the PnP problem and calculate how the camera, and in consequence, the agent, is moving.	9
2.6	Two images being captured by cameras O_L and O_R . Points P_L and P_R both correspond to point P viewed from different perspectives.	11
2.7	Main pipelines of geometric – appearance-based VO [68]. Inside the dotted lines is the representation of the motion estimation algorithms that can be applied to the selected features.	12
2.8	Architecture of DeepVO [52].	19
2.9	StackNet processing an image. ²	19
2.10	General overview of DVSO. ³	20
2.11	Overview of the D3VO architecture [74].	21
2.12	Self-Attention module [108].	22
2.13	A simple transformer architecture with encoder-decoder [108].	24
2.14	Patch and Positional Embedding [108].	25
2.15	Visual representation of segment and position embeddings [57].	26
2.16	Taxonomy of Vision Transformers [108].	28
3.1	An image from the KITTI dataset [32], where there aren't a lot of good static reference points, and there is a moving vehicle on the scene.	34
3.2	Optical flow for a pair of images. The color represents the direction of the movement; the saturation represents the value.	35
3.3	Architecture of the PoseFormer2D. The input stereo images are processed and transformed into a 1D vector representation where positional encodings are added. The tensor then goes through a transformer block composed of N transformer layers. The resulting tensor with the added attention values is used by the MLP heads to regress the 6D pose.	37
3.4	Architecture of the PoseFormer3D. The input stereo images are processed and transformed into a 1D vector representation where 3D positional encodings are added. The tensor then goes through a transformer block composed of N transformer layers. The resulting tensor with the added attention values is used by the MLP heads to regress the 6D pose.	38

3.5	Architecture of the Spatial-Temporal Transformer. The input stereo images are processed and transformed into a 1D vector representation where spatial position encodings are added. The tensor then goes through a spatial transformer block composed of N transformer layers. Before the output goes through another transformer block focused on finding temporal relationships, temporal tokens are added. The output of this block is fed to the MLP heads to regress the 6D pose. .	38
3.6	A single MLP Head, tasked with regressing a single value of the output 6D pose.	39
4.1	Training loss of PF2D, PF3D, and STPF models when trained with and without flow information. The 2D and 3D models were tested with and without learnable encodings. All the models were trained for 30 epochs with the Reduce On Plateau (ROP) optimizer scheduler.	42
4.2	Validation loss of PF2D, PF3D, and STPF with ROP scheduler.	43
4.3	Training loss of PF2D, PF3D, and STPF models when trained with the cosine annealing with warm restarts optimizer scheduler. The 2D and 3D models were also tested with and without learnable encodings.	44
4.4	Validation loss of the transformers when using the cosine annealing with warm restarts optimizer scheduler.	44
4.5	Plotted trajectory for the training sequences 00 (Urban), 01 (Highway), and 02 (Urban).	47
4.6	Plotted trajectory for the training sequences 05 (Urban), 08 (Urban), and 09 (Urban).	47
4.7	Plotted trajectory for the testing sequences 03 (Urban), 04 (Urban), and 06 (Urban).	48
4.8	Plotted trajectory for the testing sequences 07 (Urban) and 10 (Urban).	48
4.9	Translation error for sequences 00, 01, and 02.	49
4.10	Translation error for sequences 03, 04, and 05.	49
4.11	Translation error for sequences 06, 07, and 08.	49
4.12	Translation error for sequences 09 and 10.	50
4.13	Rotation error in degrees/100m for sequences 00, 01, and 02.	50
4.14	Rotation error in degrees/100m for sequences 03, 04, and 05.	50
4.15	Rotation error in degrees/100m for sequences 06, 07, and 08.	51
4.16	Rotation error in degrees/100m for sequences 09 and 10.	51
4.17	The STPoseFormer is better than DeepVO in all scenarios and manages to outperform the sota ORB-SLAM3 slightly in sequence 01. On sequence 02, the STPoseFormer suffers from drift problems, which are less pronounced on the ORB-SLAM3.	52
4.18	The STPoseFormer significantly outperforms the DeepVO on sequences 04 and 05. Sequence 06 shows a somewhat similar performance between the STPoseFormer and DeepVO.	52
4.19	Sequences 07 and 08. The STPoseFormer does not do very well on these sequences due to the accumulated errors. ORB-SLAM3 stands as the indisputable best method.	52
4.20	The STPoseFormer is better than DeepVO on all three sequences.	53
4.21	The STPoseFormer significantly outperforms the DeepVO.	53
4.22	The plotted unaligned sequences.	53

List of Tables

2.1	Table of five relevant methods. All the implementations are vision based. Each metric is taken from the respective paper where they were first described.	31
4.1	Quantitive comparison between the three developed architectures, ORB-SLAM3 without loop closure and DeepVO. Results are scaled using 6DoF. The PF2D uses flow information without learnable encodings and ROP optimizer scheduler. PF3D uses flow information without learnable encodings and ROP optimizer scheduler. STPoseFormer uses flow information and the ROP optimizer scheduler.	46
4.2	Average and standard deviation time for the proposed models	54

Abbreviations and Symbols

2D	Two dimensional
3D	Three dimensional
4D	Four dimensional
BRIEF	Binary Robust Independent Elementary Features
BRISK	Binary Robust Invariant Scalable
CNN	Convolutional Neural Network
DSO	Direct Sparse Odometry
FAST	Features from Accelerated Segment Test
FNN	Feed Forward Network
GPS	Global Positioning System
ICBM	Inter Continental Ballistic Missile
IMU	Inertial Measurements Units
ML	Machine Learning
MLESAC	Maximum Likelihood Estimator Sample Consensus
NLP	Natural Language Processing
ORB	Oriented FAST and Rotated BRIEF
PAAS	Position-Aware Attention Scaling
PF2D	PoseFormer2D
PF3D	PoseFormer3D
PnP	Perspective-n-Point
PROSAC	Progressive Sample Consensus
RANSAC	Random Sample Consensus
RGB-D	Red Green, Blue and Depth
RL	Reinforcement Learning
RNN	Recursive Neural Network
SIFT	Scale-invariant feature transform
SLAM	Simultaneous Location And Mapping
SPTF	Spatio-Temporal PoseFormer
SURF	Speeded Up Robust Features
ViT	Vision Transformer
VO	Visual Odometry
WHO	World Health Organization

Chapter 1

Introduction

1.1 Context

Vehicles have not only revolutionized the way we move, but they have also greatly expanded our access to the world, making distances less of a constraint and enabling unparalleled freedom and flexibility. In the past years, systems like adaptive cruise control, automatic lights, and others, were all developed with the experience of the user in mind. Autonomous driving technologies carry the potential to significantly reduce the number of traffic accidents [60] which claim more than 1.3¹ million lives annually. The majority of these incidents, around 90%, are due to human error. Autonomous vehicles are expected to substantially improve traffic flow², as the technology would enable vehicles to interact and coordinate with each other, distributing traffic evenly across available routes. Autonomous vehicles also have the potential to significantly alleviate the transportation challenges experienced by people with disabilities, promoting equality of rights across the world.

Governmental laws, economic challenges, and even philosophical reasons are all obstacles that companies have to deal with before a fully autonomous vehicle is available for public use. Until then, the work developed in controlled environments, simulations, and on limited scales forms the foundation of the knowledge in this rapidly progressing field. A crucial component of autonomous driving is ego-motion estimation [41]. Ego motion pertains to the process of determining a vehicle's relative positioning over time, and it plays a critical role in the understanding and anticipation of a vehicle's motion, laying the groundwork for decision-making and action execution in autonomous vehicles. Without accurate ego-motion estimation, an autonomous vehicle would struggle to accurately predict its movement, resulting in reduced operational efficiency and potentially compromising the entire system.

¹<https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries>.

²https://cinea.ec.europa.eu/publications/eu-road-safety-towards-vision-zero_en.

1.2 Motivation

Visual odometry is at a point where geometric techniques can accurately give pose estimations for most of the environments the vehicle may come across. The problem starts in environments where the conditions are not the best, and the mathematical principles behind traditional approaches start producing wrong or less accurate results. With no clear way to circumvent this issue, a clear necessity for implementing a more universal strategy starts emerging. This is why learning-based approaches present themselves as a prime solution for this issue. Machine learning has been exponentially growing, over the last couple of years, with improved, or even, entirely new architectures being developed at an unprecedented rate. With machine learning, a more general solution can be achieved since the trained models can learn how to more effectively use all the information on the images. Currently, the available learning-based techniques are not as good as the best geometric approaches, and this poses some questions, such as: *Are the currently employed learning approaches appropriate for odometry?*, *Is it necessary to create a new architecture from scratch?*. Considering these questions, the potential of Transformer architectures comes into view. Originally developed for Natural Language Processing (NLP) tasks, transformers have since been adapted to a variety of other domains, including the visual domain. Successful implementations of transformers have been observed in fields like image classification [71] [93], video classification [77], object detection [70] [85], semantic segmentation [98] [99] and pose estimation [104]. Driven by these considerations, this dissertation outlines a set of goals to be achieved, which will be elaborated in the subsequent section.

1.3 Objectives

This dissertation intends to explore the application of vision transformer models in visual odometry, with the ultimate objective of assessing their effectiveness and performance in this specific task. The proposed methodology relies on the KITTI visual odometry dataset [32] to train, test and evaluate the presented solution.

The following objectives are expected to be met during the dissertation:

- To improve the current scientific reviews on visual odometry and vision transformers.
- To develop learning-based architectures from scratch for visual odometry tasks, capable of doing reasoned ego-motion estimates.
- To base the architectures on vision transformer structures capable of producing the best possible estimates.
- To train and evaluate the models developed by comparing them with the performance of other visual odometry alternatives.

The work produced in this thesis uses cameras as a primary source of data. Thus visual odometry is the only type of odometry explored in more detail.

1.4 Document Structure

Chapter 2 provides a comprehensive overview of visual odometry techniques and introduces the concepts of transformers and visual transformers. Chapter 3 discusses the implementation of the developed transformers, including details on the loss function, optimization methods, and data processing techniques. Chapter 4 presents and interprets the experimental results from training the models on the KITTI odometry dataset. Chapter 5 concludes the thesis, summarizing key insights and discussing potential directions for future research in this field.

Chapter 2

State of the Art

In autonomous navigation, it is essential for a vehicle to understand how it is moving [40]. This understanding of its own motion relative to the surrounding environment is referred to as ego-motion. The GPS (Global Positioning System) is a commonly-used technology for self-location, as it can, with some degree of accuracy, determine a vehicle's position and speed using radio signals. The GPS was developed by the military in 1973 to locate Inter Continental Ballistic Missiles (ICBMs) [49]. However, only later, in 1980, was the technology made available for public use at a different carrier frequency [51]. GPS alone is not good enough to solve ego-motion, as its ability to, reliably and accurately, self-locate is subpar. Moreover, it has many other flaws, e.g., it does not provide information about angular velocity, only linear [67]; the strength and signal quality vary depending on the agent environment conditions [21], and with the atmosphere adding delay and noise, these problems are amplified even further [33]; even with newer satellites capable of handling the newer GPS III communication architecture, the error margin is still around 1.5 to 2 meters [11], which is still far from the desired fidelity. To conclude, GPS alone is not accurate enough to be used as the only way of self-location, and even if the GPS was capable of locating an agent with maximum accuracy and precision, an autonomous driving system that relies on GPS technology is not self-contained. What this ends up meaning is that there could be situations where the data required for estimating ego-motion is not available to the vehicle (e.g. in an underground park or in some other GPS-denied areas). These problems made researchers turn their attention elsewhere, and studies on GPS-denied odometry began to emerge.

Odometry is a set of techniques that allow the agent to estimate its position and orientation based on the distance traveled by its wheels or other means of locomotion [67]. Self-contained odometry allows the agent to solve odometry problems with access only to its onboard sensors, i.e., without relying on external signals, which, as seen above, can be unreliable. Odometry and ego-motion are intrinsically linked, with the main difference being their scope. Ego-motion usually refers to the more general concept of a vehicle movement related to its own frame of reference, while odometry is a set of techniques used to estimate that motion based on data. Different types of odometry techniques use various sensors, including wheel encoders, inertial sensors, radar, cameras (visual), and lasers. The combination of different odometry methods builds up the hybrid

techniques known as radar-inertial, visual-inertial, visual-laser, and visual-radar. Visual-inertial odometry approaches can also be further studied from two specific aspects, i.e., whether they are filter/optimization based or tightly/loosely coupled [67], as seen in Fig 2.1.

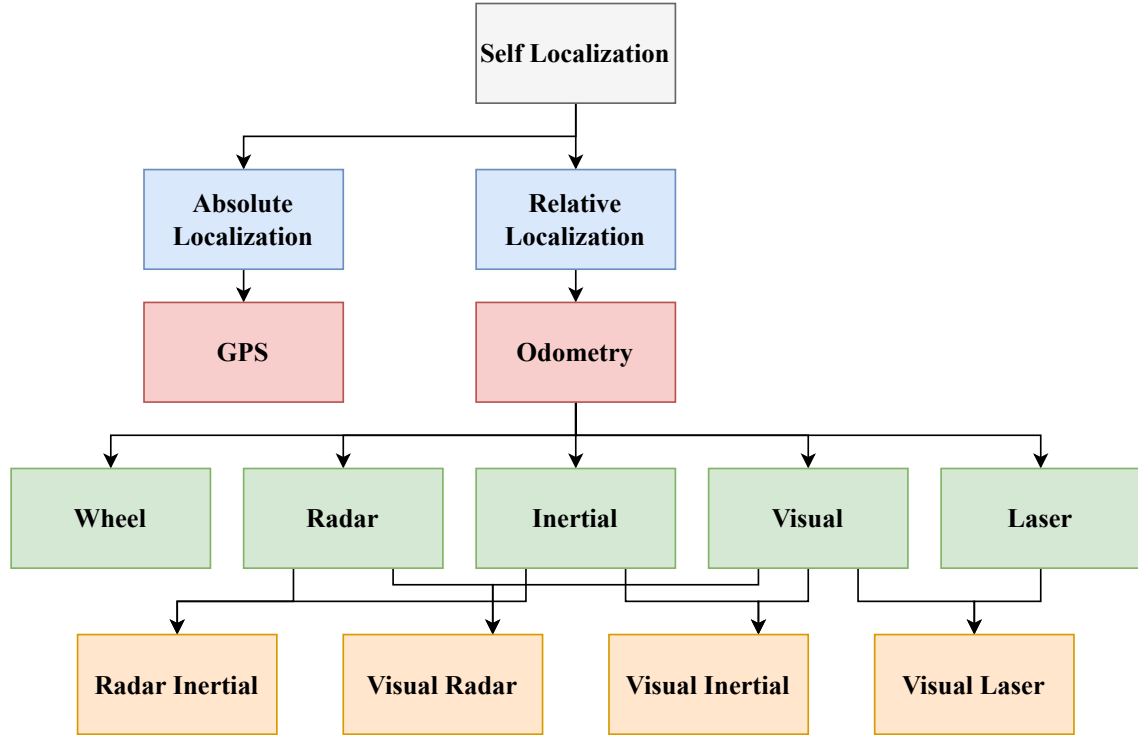


Figure 2.1: Types of Self Location, particularly, the different types of odometry.

2.1 Visual Odometry

Visual odometry (VO) is the name given to odometry techniques that use data from one or more cameras attached to the agent. VO techniques estimate the agent's pose incrementally by analyzing the motion changes captured in a series of images taken by the onboard camera(s) [29].

Visual odometry's first implementation is usually associated with the work done in 1980 by Moravec [1], where he developed a system that would let the lunar rover, built by Stanford University, dodge obstacles without requiring human instructions. However, only later, in 2004, did VO gain mainstream attention with Nister's paper [12]. Nister *et al.* improved the earlier implementations in several areas, and unlike previous works, this approach does not track features between frames but instead detects features independently in all frames and only allows matches between relevant features. Thenceforth, VO has significantly evolved alongside the increase in quality of image-capturing equipment and the seemingly endless progress of Machine Learning (ML) techniques.

Visual odometry approaches are divided into two groups [76] [105], figure 2.2:

- Conventional, also known as geometric.

- Non-Conventional, better known as learning-based approaches.

Geometric approaches can be further subdivided into three groups:

- Feature-based, or indirect approaches.
- Appearance-based, or direct approaches.
- Hybrid techniques, which use elements of the previous two approaches

Learning-based approaches are based on machine learning tools (i.e., a regression model) trained by VO parameters to estimate the motion [12][35][26]. Geometric and learning-based methods estimate motion based on the subsequent changes in the position of the cameras. Geometric approaches take advantage of specific feature points, like corners, while machine learning models can be tuned to either make use of feature points or be entirely self-taught, meaning that they can learn which components of the input are more important. Not needing to detect explicit features in the input ends up being one advantage of learning-based techniques. Initializing camera parameters is also not required, as the models can learn to extract that information on their own. Furthermore, correcting the scale of the estimated trajectories is not always needed [26], as the models can learn to capture that information on their own.

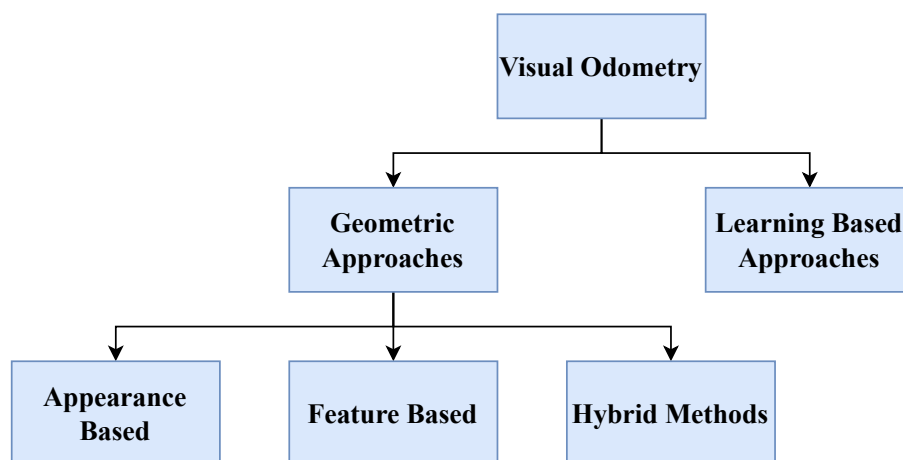


Figure 2.2: Visual Odometry methodologies.

2.1.1 Geometric Methods

Feature-based methods, figure 2.3, were the first to appear. The system developed by Moravec [1] matches distinct features (corners) between stereo images and triangulates them to the 3D world frame. Once the robot moves, these feature points are matched in the next frame to obtain the corresponding 3D points and generate motion parameters. This approach detects interest points, and as such, it is considered a feature-based method. Even if Moravec's implementation uses corners as feature points, there are other interesting collections of points that can be considered when trying to establish relations between successive images, like lines, curves, edges, and blobs

(constant areas different from their surroundings). Interest points, however, are not limited to these shapes as any point or collection of points that stand out from its surroundings can be categorized as a feature. Interest points are usually distinguishable due to their intensity, color, and texture.

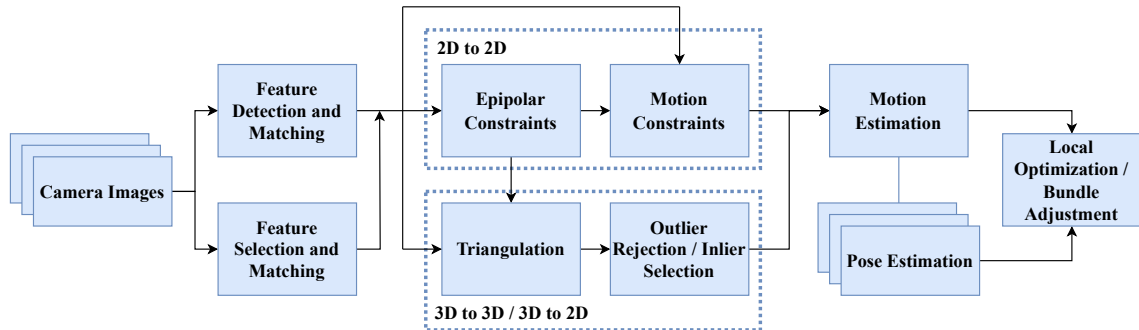


Figure 2.3: Main pipelines of geometric – feature-based VO [68]. Inside the dotted lines is the representation of the motion estimation algorithms that can be applied to the selected features.

Feature-based VO methods are generally considered robust to geometric distortions and poorly illuminated environments [9]. However, they may disregard some of the valuable information in the image by only extracting certain key features. Detecting and matching features between frames also requires a significant amount of computational resources, which is proportional to the number of features extracted. The most well-known algorithms for feature detection and description are the gradient-based SIFT [10] figure 2.4 and SURF [16]; the binary ORB [56] [28] (ORB is a combination of Orientated FAST [20] and Rotated BRIEF [25]), and BRISK [27] algorithms. Other algorithms only do detection, e.g., Harris [3], ShiTomasi [5], FAST [20], and others only do description, e.g., BRIEF [25]. The algorithm or combination of algorithms one should choose depends on their needs, with some algorithms being faster while others are more accurate but more computationally expensive.

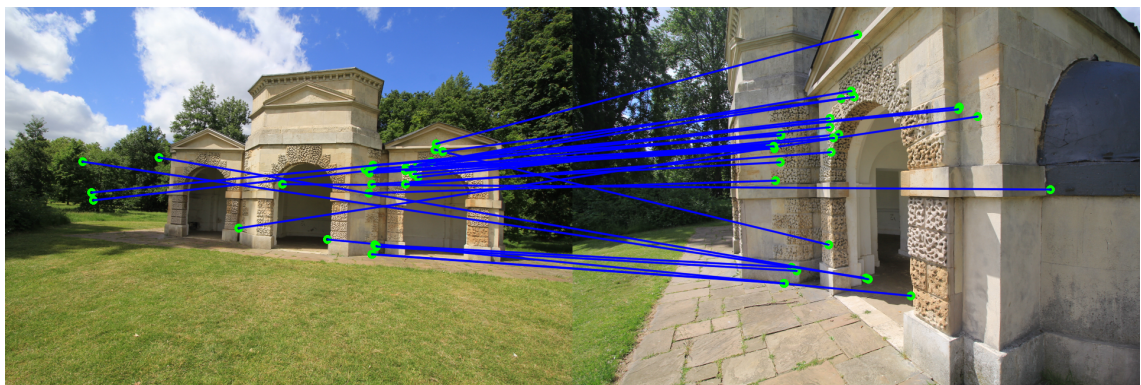


Figure 2.4: Example of feature point extraction and matching using SIFT.¹

Nevertheless, using the extracted feature points with no outlier-removing process tends to lead to poor results. Many techniques remove these outliers, the most well-known being RANdom

¹https://kind-of-works.com/CODE_matching.html.

SAmple Consensus (RANSAC) [2]. RANSAC is an iterative method based on a random voting scheme that tries to find optimal fitting for the data. Since RANSAC's creation, there have been other implementations of an outlier detection algorithm, e.g., PROSAC (PROgressive SAmple Consensus) [14], and MLESAC (Maximum Likelihood Estimator SAmple Consensus) [7].

After extracting the relevant key-point information from the image and removing outliers, there are three standard feature-based motion estimation methods: 2D to 2D (feature-to-feature matching), 3D to 2D (structure-to-feature matching), and 3D to 3D (structure-to-structure matching) [67] [76]. Whichever motion estimation method is used, 3D to 3D, 3D to 2D, or 2D to 2D, they all estimate the change in motion for every new image, allowing them to determine the complete trajectory of the camera and the associated agent. The 3D to 3D algorithm computes relative motion by specifying the features f_{k-1} and f_k from successive images I_{k-1} and I_k in 3D. To achieve this, it is necessary to triangulate 3D points at each time instant, i.e., by using a stereo camera [29]. A stereo camera is a camera with two or more lenses, which allows it to simulate human binocular vision and, therefore, capture three-dimensional images.

In the case of the 3D to 2D method, the features f_{k-1} are specified in 3D and their corresponding reprojections, f_k , in 2D. Capturing the images in 3D requires a stereo camera like in the previous 3D to 3D method or the use of two monocular (2D) cameras. To capture 3D features with monocular cameras, each feature f_k needs to be triangulated with the help of two adjacent camera views (e.g., I_{k-2} and I_{k-1}). This can be done by two other cameras or by a sequence of images that capture the same feature f_k . In other words, to capture a 3D feature f_k and project it on a 2D image representation I_k , an additional two other views are necessary. This adds up to a total of three different views (i.e. I_{k-2} , I_{k-1} , I_k). With the 3D points and their respective 2D image reprojection given, it is possible to estimate the pose of the calibrated camera(s). Finding the 6 DoF, or the pose, of a calibrated camera is the well-known Perspective-n-Point problem (PnP) [2]. The most common solution to this problem, P3P, exists when three different points are available between consecutive views like it is seen in I_{k-2} , I_{k-1} , and I_k , figure 2.5.

The 2D to 2D motion estimations have the features f_{k-1} and f_k specified in 2D image coordinates. The geometric relationship between these features is described by the epipolar constraint. The Epipolar constraints belong to a specific type of geometry known as Epipolar geometry. One fundamental concept of Epipolar geometry is the Epipolar line. The ray cast from the optical center of the view O_L , which intercepts the feature point P_L , can be seen from another view as the Line E_R . This line is called Epipolar Line. The interaction mentioned is shown in figure 2.6. To find the geometric relationships between two calibrated cameras or between two views of a single moving camera (rotations, translations), the correspondences between the feature points found have to be described. This can be done by calculating the essential matrix. The essential matrix E is described as follows:

$$E = [t_x]R. \quad (2.1)$$

Where R is a 3×3 rotation matrix and t_x is the skew-symmetric matrix obtained from the 3-dimensional translation vector t . The 3×3 t_x matrix has a very special property: for any vector

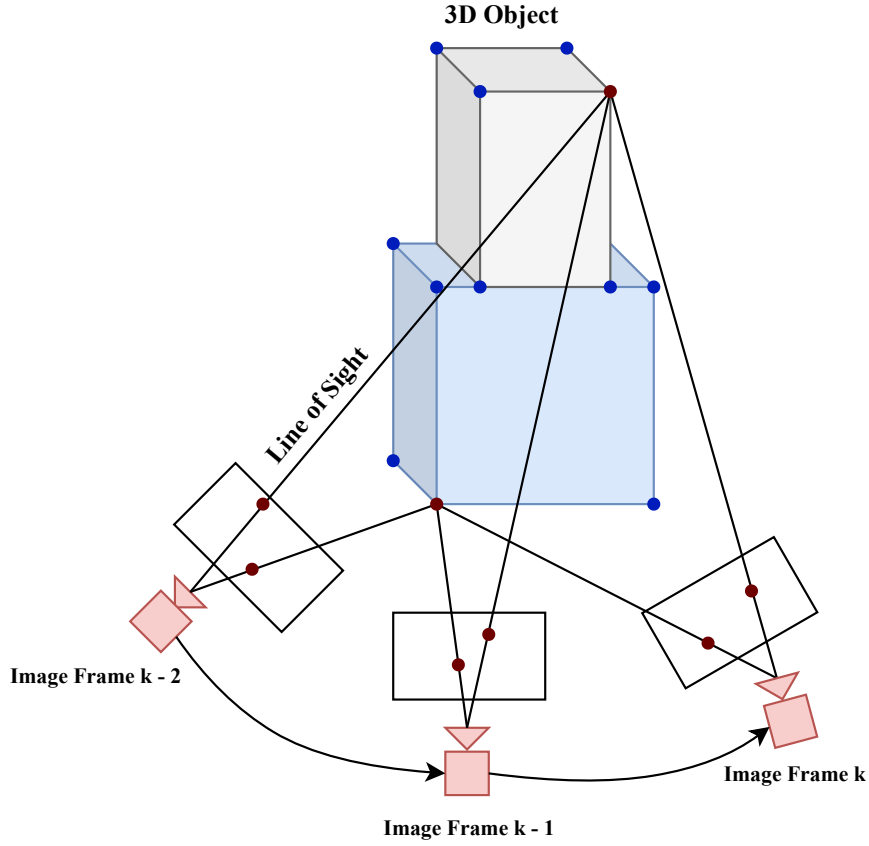


Figure 2.5: Sequential pictures being taken from a moving camera. By tracking the motion of the chosen feature points (points in red), it is possible to solve the PnP problem and calculate how the camera, and in consequence, the agent, is moving.

$v = (v_1, v_2, v_3)$, the cross product of t and v (written $t \times v$) can be computed using matrix multiplication, as follows: $t \times v = [t_x]v$. The resulting essential matrix encodes the epipolar constraints:

$$P_R^T E P_L = 0. \quad (2.2)$$

However, when one uses the essential matrix, there is no consideration for the camera's intrinsic parameters; but they need to be considered in order to correctly estimate the geometric relationships between different cameras or between different camera views. When considering the camera's intrinsic parameters matrix (K), another matrix, the Fundamental Matrix (F), describes the correspondences between feature points. The definition of the fundamental matrix is denoted as F in the equation below.

$$F = (K'^{-1})^T [t_x] R K^{-1}. \quad (2.3)$$

Here, K' and K are the intrinsic camera matrices for two different cameras; or the intrinsic camera matrices for the same camera at two different times.

The fundamental matrix also contains the epipolar constraints

$$P_R^T F P_L = 0. \quad (2.4)$$

The relationship between the fundamental matrix F , and the essential matrix E

$$F = (K'^{-1})^T E K^{-1}. \quad (2.5)$$

Once the matrix F is calculated, E can be obtained by the previous equations. Furthermore, by doing a singular value decomposition of matrix E , the rotations and translations of the camera views can be obtained. The real combination of R and t from the resulting four is found by doing what is often called a cheirality check. This process involves selecting the correct combination by using a physical constraint: a point in space, when viewed from both cameras, should appear in front of both cameras. By projecting a known corresponding point from both views into 3D space using each of the four possible combinations, it is possible to determine which combination produces a point that fulfills this condition. The correct combination is what gives accurate rotation and translation between the camera views. Afterward, the epipolar line, l_R , can be calculated by having access to a point P_L in a view and the fundamental matrix F .

$$l_R = F P_L. \quad (2.6)$$

Replacing $F P_L$ on equation 2.4, gives the equation

$$P_R^T l_R = 0, \quad (2.7)$$

which describes the epipolar constraint.

Given a set of matched points $\{x_i, x'_i\}$, and the relationship between F and x , where $\tilde{x}$ and $\tilde{x}'$ are the reprojection of those points in a 2D view; and knowing the following relation is true,

$$\tilde{x}'^T F \tilde{x} = 0, \quad (2.8)$$

it is possible to compute the fundamental matrix by an 8-point algorithm. Hugh Christopher Longuet-Higgins, in 1981, was the one who created this algorithm. More recently, David Nister provided, in 2003, the first efficient solution [13], which requires only five points.

Feature-based VO has some drawbacks, which makes it lackluster in some situations. Computing feature descriptors is very time-consuming, especially gradient-based ones like SIFT [10]. It also is impossible to do a 3D reconstruction of the environment (dense reconstruction) using only feature points.

Appearance-based VO techniques, figure 2.7, use other methods that estimate the camera's movement by minimizing photometric error and by comparing the intensity of the pixels in captured images. In contrast to feature-based VO, this method utilizes all of the geometric information from the captured camera images, which helps to reduce problems with aliasing that can occur in

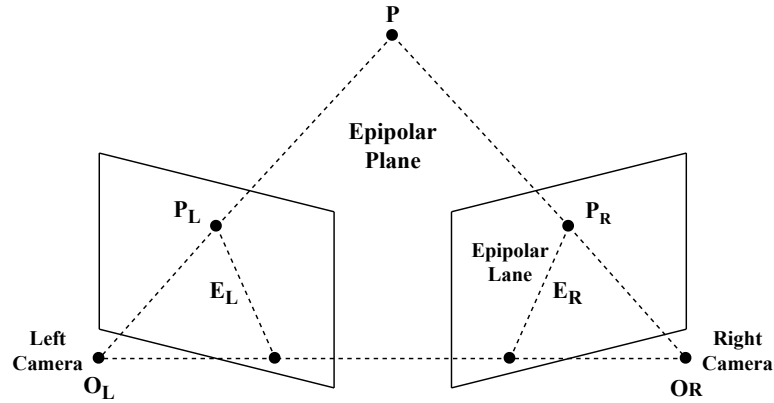


Figure 2.6: Two images being captured by cameras O_L and O_R . Points P_L and P_R both correspond to point P viewed from different perspectives.

scenes with similar patterns and improves the accuracy and reliability of the pose estimate, particularly in environments with low texture or low visibility, e.g., deserts [17]. The process of appearance-based VO can be divided into two main categories: region/template matching-based and optical flow-based methods [76].

Optical flow methods analyze the patterns of pixel intensity in an image sequence and use that information to compute the displacement of the pixels between frames. This allows for estimating the velocity and direction of movement of objects in the scene [36].

Regional-based VO methods divide the image into smaller regions and estimate the motion of each region independently. This can be more robust than optical flow in cases with large motions or deformations in the scene because it allows for more localized estimates of motion. However, it can also be more computationally expensive, requiring analyzing multiple regions rather than a single image. The work of Vatani *et al.* [24] shows a simple localized approach that relies on a constrained motion of a large vehicle, where downward-facing cameras are used to measure the pixel displacement and translate it into vehicle motion. An extension of this work was proposed by Yu *et al.* by utilizing a rotating template instead of a static template to find the translation and rotation between two consecutive images [30]. Since the development of this work, other implementations have used adaptive template matching [42], where the template is smaller, and its location varies concerning the vehicle acceleration.

To fully understand appearance-based VO methods, knowledge of 3D point projections, image warping, and photometric loss, ends up being crucial. The following mathematical expressions demonstrate how by using the three concepts mentioned, an appearance method can be implemented.

For a point,

$$X_W = [X, Y, Z]^T, \quad (2.9)$$

represented in the 3D world coordinate system, its projection on a reference image frame can be

described by,

$$\tilde{x}_{im_ref} = \frac{1}{\tilde{Z}} K X_W, \quad (2.10)$$

where first the homogenized coordinates of the point X_W are multiplied by the camera's intrinsic parameters matrix K , to obtain a 3D point camera coordinate in the system. The Z coordinate of this point is then divided by the homogenized Z coordinate of the point in the world coordinate system, $\hat{Z}$. It is now possible to project the 3D point to another frame with estimated R (Rotation), t (translation) and Z (depth) with

$$\tilde{x}_{im_warp} = K[\hat{R}|\hat{t}]\hat{Z}K^{-1}\tilde{x}_{im_ref}. \quad (2.11)$$

This builds an image if multiple 3D points and their respective projections are available. This process of finding each point projection is called warping. Whence the warping is obtained, so is the photometric loss. Because according to the brightness constancy assumption [4], if a perfect estimation of R , t , and Z exist, the warped image and the target image should be the same. In real-world scenarios, this is not possible, so there is always some loss, called photometric loss, L , described by the following equation

$$L = \frac{1}{HW} \sum_{i=1}^{HW} |I_{target}(i) - I_{warp}(i)|, \quad (2.12)$$

where I is the pixel intensities of the two images. By minimizing the photometric loss, the parameters R , t , and Z are optimized and provide better estimates of the change of motion. To do so one can use any non-linear optimization algorithms like the Levenberg-Marquardt [31] and Gauss-Newton methods. The latter was originally developed in the context of the method of least squares by Carl Friedrich Gauss around 1809.

Photometric loss can also potentially be used, as a loss function, for training a machine learning model to estimate the pose (position and orientation) and depth (distance from the camera) of objects in an image. These concepts will be explored further in chapter 3.

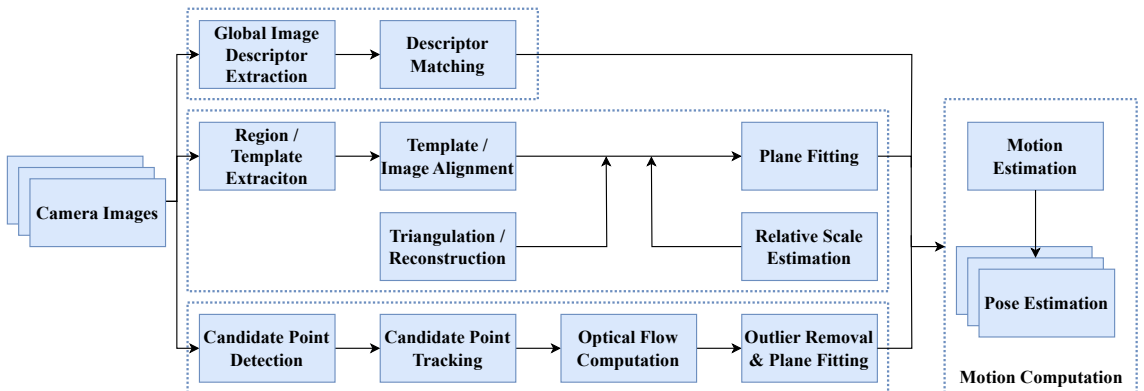


Figure 2.7: Main pipelines of geometric – appearance-based VO [68]. Inside the dotted lines is the representation of the motion estimation algorithms that can be applied to the selected features.

As both direct and indirect approaches have their strengths and weaknesses, there are hybrid techniques that try to obtain the best of both domains. As already mentioned, pure feature-based VO schemes are not considered to be robust since only a few features are detected and tracked. On the other hand, appearance-based VO does yield better results but has increased computational cost. Scaramuzza and Siegwart [23] proposed a hybrid method that features two systems. The first uses a feature-based approach with SIFT [10] and RANSAC [2] to estimate the translation of the vehicle. The second uses an appearance-based approach to get the rotation of the agent.

Feature, Appearance, or even hybrid approaches all have to face some similar problems, like drift. Drift happens due to the iterative nature of odometry, where errors accumulate over time and lead to poor estimation accuracy of visual odometry. To reduce the amount of drift, strategies like pose-graph optimization can be used. As seen previously, in VO, the camera poses are typically calculated by combining transformations between two consecutive views at times I_k and I_{k-1} . However, it is also possible to compute transformations between the current image I_k and the previous n image steps ($I_k, I_{k-1}, \dots, I_{k-n}$) or any other image steps (I_i, I_j). If these transformations are known, they can be used to refine the camera poses by incorporating them as additional constraints in a pose-graph optimization process.

In Pose-Graph representations, the camera poses are nodes, and the rigid-body transformations between the camera poses are the edges, which connect the nodes [19]. Each new transformation can be added as an edge, e_{ij} , to the pose graph, where $T_{e_{ij}}$ is the transformation between the poses i and j [31]. Pose graphs have a cost associated with each added edge e_{ij} defined by the following function [31]:

$$\sum_{e_{ij}} \|C_i - T_{e_{ij}} C_j\|^2. \quad (2.13)$$

The end goal of pose graphs is to minimize the cost of this function by finding the optimal camera pose parameters. It is important to note that the rotation part of the transformation makes the cost function nonlinear, so a nonlinear optimization algorithm must be used (e.g., Levenberg-Marquardt) [31]. Pose optimization graphs can be further improved by establishing connections between edges which are usually far apart and have accumulated a large amount of drift. Maintaining information about a previously observed location makes it possible to create a loop. This type of calculation is called loop detection [15]. Loop detection is an extensive problem with many possible implementations, but the ones that stand out the most are the ones that use local image descriptors [18] [22]. By representing images with bags of visual words, it is possible to measure the similarity between images by computing the distance between the images' word histograms.

Another way to correct drift is by using Windowed Bundle Adjustment [8]; a technique similar to pose-graph's idea of refining the estimates of the camera parameters. However, it also includes the optimization of 3D landmark parameters simultaneously. This method is suitable for situations where image features are tracked throughout more than two frames. It involves considering a window of n consecutive frames and optimizing the camera poses and 3D landmarks for this set

of frames. In bundle adjustment, the error function to minimize is the image reprojection error:

$$\arg \min_{X^i, C_k} \sum_{i,k} \|p_k^i - g(X^i, C_k)\|^2, \quad (2.14)$$

where p_k^i is the i th image point of the 3D landmark X^i measured in the k th image and $g(X^i, C_k)$ is its image reprojection according to the current camera pose C_k [31]. As the error function is nonlinear, like in pose-graphs, the optimization step needs to be done by nonlinear functions. One thing to keep in mind when using this technique is that the computational complexity is $O((qM + lN)^3)$ with M and N being the number of points and cameras poses and q and l the number of parameters for points and camera poses [31]. Keeping large amounts of camera poses can result in calculations that are not fast enough to achieve soft real-time (something is done in soft real-time when the window of time it takes to complete an action is small enough that it does not impact the intended behavior).

Before moving on to the learning-based methods, there are some interesting works worth exploring more in-depth [58]. Starting with ORB-SLAM2 [47], which is a geometric implementation from the family of ORB-SLAM algorithms, currently made up of three versions (ORB-SLAM, ORB-SLAM2, and ORB-SLAM3). ORB-SLAM2 is a complete Simultaneous Location And Mapping (SLAM) approach for monocular, stereo, and RGB-D cameras. ORB-SLAM2 extends the features of its predecessor, ORB-SLAM, by incorporating a robust and efficient system designed to work in real-time on standard hardware. The efficiency of ORB-SLAM2 arises from the use of the ORB feature descriptor. Specifically, ORB-SLAM2 involves three crucial stages: tracking, local mapping, and loop closing. The tracking phase estimates the camera trajectory in real-time by detecting ORB features in the scene and matching these with the current map. In parallel, the local mapping process refines the map, ensuring that it accurately reflects the current understanding of the environment. The loop closing module detects and corrects drifts in the trajectory and the map to ensure global consistency. Moreover, ORB-SLAM2 offers reliable initialization techniques for both monocular and stereo cameras, improving upon the vulnerability of monocular SLAM systems to initialization errors. The ability to run on monocular, stereo, and RGB-D cameras makes it versatile, lending itself to a wide range of applications, from robotics to augmented reality. The core strength of ORB-SLAM2 is its robustness and resilience in a variety of environments, including small-scale indoor areas and larger-scale outdoor settings. It achieves high-accuracy performance in mapping and localization tasks, even in low-texture or dynamic scenes, making it an effective choice for real-world deployment. It is important to note, however, that, like any SLAM system, ORB-SLAM2 faces challenges in handling extreme lighting conditions, fast camera motions, and feature-poor environments. As such, these are areas for future work and refinement.

ORB-SLAM3 [80] is the latest iteration in the family of ORB-SLAM algorithms, following ORB-SLAM and ORB-SLAM2. Like its predecessor, ORB-SLAM3 is a complete SLAM approach for monocular, stereo, and RGB-D cameras. ORB-SLAM3 builds upon the robustness and efficiency of ORB-SLAM2, which was designed to operate in real-time on standard hardware,

utilizing the ORB feature descriptor. The algorithm is equal to that of ORB-SLAM2, meaning it has the same three key stages: tracking, local mapping, and loop closing. ORB-SLAM3 extends the capability of its predecessors by incorporating advanced features, including support for inertial measurements units (IMUs), more robust visual-inertial SLAM, and the ability to handle pure inertial trajectory estimation. This latest version maintains the strength of reliable initialization techniques present in ORB-SLAM2, for monocular and stereo cameras, thereby addressing the vulnerability of monocular SLAM systems to initialization errors. Furthermore, the use of ORB-SLAM3 with monocular, stereo, and RGB-D cameras, as well as with IMUs, broadens its potential applications, from robotics to augmented reality and beyond. It shines in a variety of environments, be they small-scale indoor spaces or larger-scale outdoor settings, delivering high-accuracy performance even in low-texture or dynamic scenes. Nonetheless, it is important to mention that, like its predecessors, ORB-SLAM3 still encounters challenges in dealing with extreme lighting conditions, fast camera motions, and environments lacking features. These remain areas for further research and refinement. Despite these challenges, ORB-SLAM3's improvements make it a robust and versatile tool for real-world deployment in SLAM-based applications.

In 2016, the method Direct Sparse Odometry (DSO) [43] appeared. DSO is based on continuous optimization of the photometric error over a window of recent frames. In contrast to the traditional methods up until that point, DSO jointly optimizes for all involved parameters (camera intrinsic, extrinsic, and inverse depth values), effectively performing the photometric equivalent of windowed sparse bundle adjustment. DSO presented itself as a promising method since it combines the benefits of direct methods (seamless ability to use and reconstruct all points instead of only corners) with the flexibility of sparse approaches (efficient, joint optimization of all model parameters).

2.1.2 Learning Based Methods

Geometric approaches rely on explicit hand-crafted models of the environment and the camera's intrinsic parameters and use techniques such as feature matching, triangulation, and optimization to estimate the camera's pose. These methods are accurate and efficient but can be brittle and prone to failure in complex or dynamic environments. On the other hand, learning-based approaches focus on learning how to represent the environment and the camera's intrinsic parameters from data rather than assuming a pre-defined model. These methods are more robust and adaptable and can potentially handle complex and dynamic environments better than geometric approaches. There are clear advantages that learning-based approaches possess. There is no need to fine-tune camera parameters; learning-based approaches showcase robustness in tracking failure and scale drift [53]; by using stereo image pairs in the training phase, they are capable of recovering metric scale from a monocular image [46] [64]. However, they can be more computationally expensive and may require large amounts of labeled data for training.

Estimating odometry using non-conventional techniques is usually done by training deep neural networks driven by many image sequences. These techniques hope to benefit from the data-driven approach and the potential of deep neural networks, as seen in other areas where they are

being applied. Learning-based methods can be categorized as follows:

- Supervised Learning
- Unsupervised or Self-supervised Learning Approaches
- Reinforcement Learning Approaches
- Hybrid Methods
- End-to-End Learning Methods

it is essential to keep in mind that while these groups can help categorize different methods, there is significant overlap, and many methods could be considered as belonging to multiple groups. For instance, a method could be both an end-to-end method and a supervised learning approach.

Supervised learning approaches involve the training of a model with labeled data. In the context of VO, the labels could be the relative pose or position of the camera. The model then tries to learn a mapping from the input images to the correct pose, with the goal of minimizing the difference (or error) between its predictions and the true labels. The mapping function is learned during the training of the network, after which the model can be used to predict poses from new images. The most common types of Deep Learning models used are based on Convolutional Neural Networks (CNNs) [90], due to their strong capacity to learn complex patterns. CNNs apply a series of convolutions and nonlinear operations to the input image, effectively transforming the raw pixel values into a high-level representation that can be used for pose prediction. One work that pioneered the works in supervised deep learning-based VO is the PoseNet, introduced by Kendall *et al.* [45]. This architecture uses a CNN to regress the 6-DoF camera pose directly from a single RGB image, effectively bypassing traditional steps of feature extraction and matching. The novelty of PoseNet was capable of operating in real-time, attributes that have since become common in deep learning-based VO methods, but at the time were considered a novelty. PoseNet's architecture is based on GoogLeNet [39], modified to output a 7-dimensional vector representing the 3D translation and 4D quaternion orientation of the camera. Overall PoseNet is notable for being one of the first, supervised learning, implementations to show that deep learning can be applied successfully to ego-motion. While supervised learning approaches for VO can achieve good results, they also have their limitations. These can include difficulty in generalizing to different environments from the training data, reliance on large amounts of labeled data, and challenges in incorporating temporal information due to the primarily feed-forward nature of these methods.

Unlike supervised learning, unsupervised learning does not require labeled training data. Instead, these methods try to learn useful representations or structure from the input data directly. In the context of VO, unsupervised methods often involve learning to predict depth or ego-motion from images, guided by geometric constraints, such as photometric consistency, section 2.1.1. Photometric consistency, sometimes referred to as photometric error or photometric loss, is a fundamental concept in many unsupervised learning approaches for visual odometry and other

computer vision tasks. The central idea is rooted in the brightness constancy assumption, which stipulates that the intensity (or color) of a specific point in the world should remain consistent across different images, assuming the lighting conditions stay the same. Given two images, and the depth and ego-motion estimated by the networks, one image can be warped to the viewpoint of the other using these estimated values. The photometric consistency loss can then be computed as the difference in pixel intensities between the warped image and the target image. The networks are trained to minimize this loss. The basic form of the photometric consistency loss is given by:

$$L_{photo}(I_a, I_b, D, T) = ||I_a - I_b^w(D, T)|| \quad (2.15)$$

where I_a and I_b are the two images, D is the depth map estimated for I_a , T is the relative pose (translation and rotation) between the two camera views, and $I_b^w(D, T)$ is the image I_b warped into the viewpoint of I_a using the estimated depth D and pose T . Different variations and enhancements of this basic loss function can be found, for example, in the paper by Zhou *et al.* [54], where they propose a multi-scale photometric loss and use an SSIM (structural similarity) term for better performance. Still, while unsupervised methods alleviate the need for labeled training data, they come with their own challenges. These include dealing with issues such as occlusions, illumination changes, and dynamic objects that violate the assumptions of photometric consistency.

Reinforcement Learning (RL) is another type of machine learning where an agent learns to make decisions by taking actions in an environment to maximize some form of cumulative reward. In the context of VO, RL approaches typically involve learning a policy that guides the camera motion to maximize the accuracy of the estimated trajectory or other related objectives. Deep Reinforcement Learning (DRL) is a subfield of RL which brings together the decision-making capabilities of RL, and the high representation power associated with deep learning. In the context of VO, DRL can be used to make sense of images and decide how best to estimate the motion of the camera in an environment. DRL has some potential for advancing the state of VO, but it is a much more challenging approach to the VO problem. For instance, the unstable and divergent nature of learning in high-dimensional spaces makes it so that designing and tuning a DRL algorithm requires a delicate balance between exploration, where the agent probes the environment to discover profitable actions, and exploitation, where the agent uses the knowledge it has gained to optimize reward. Moreover, DRL typically requires extensive amounts of data and computation. The iterative process of trial and error through which RL operates means that considerable time and resources are often needed to train models. This is exacerbated in the context of VO, where obtaining labeled real-world data for complex environments can be a significant hurdle.

Hybrid Learning approaches attempt to combine the benefits of different learning paradigms, such as supervised and unsupervised learning, or geometric-based and learning-based methods. The goal is to leverage the strengths of each method to overcome their individual limitations. Hybrid approaches can potentially mitigate the weaknesses of individual learning paradigms. For instance, they can help to reduce the reliance on large labeled datasets, improve generalization to

new environments, and enhance the accuracy and robustness of VO estimates. The UnDeepVO model by Ruihao Li *et al.* [61], can be considered as a hybrid approach, as it learns to predict the depth and pose in a supervised manner, while photometric loss is minimized in an unsupervised way. Despite the potential of hybrid based approaches they also have their challenges. These include the difficulty of integrating different learning paradigms, the increased complexity of the resulting systems, and potential issues with scalability and efficiency.

Finally, End-to-End approaches aim to learn direct mapping from raw input data (e.g., images) to the desired output (e.g., pose). They can use supervised, unsupervised, reinforced, or hybrid learning paradigms. The significant alteration here is that the neural networks take full charge of the task, providing a more seamless processing pipeline. Instead of manually designing and linking together multiple processing stages, the network learns to execute the entire task from raw data to final output in a single pass. This direct approach often involves the usage of both Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). CNNs are generally employed at the beginning of these pipelines, serving as automatic feature extractors that transform raw image data into a more meaningful and compact representation. The sequential nature of visual odometry problem makes it a good fit for RNNs, which are designed to process temporal data by maintaining an internal state that can represent information from past inputs. End-to-End learning approaches provide a unified learning process that has the potential to streamline the pipeline, reduce error accumulation, and improve the overall performance. However, they require vast amounts of data and computational resources for training. Additionally, they often act as black boxes with their decision-making processes, making it challenging to diagnose and correct failures.

Similar to geometric methods. There are some architectures worth delving more deeply into. One of those is DeepVO [52]. DeepVO is one of the most well known learning-based approaches. DeepVO utilizes Convolutional Neural Networks (CNNs), figure 2.8 to achieve visual odometry. Unlike traditional approaches, which typically involve explicit feature extraction and matching, DeepVO leverages the power of deep learning to directly infer the 6-DoF (6 Degrees of Freedom) camera pose from raw sequential images. One of the most significant aspects of DeepVO is its ability to incorporate temporal information from sequential images, made possible by the integration of Recurrent Neural Networks (RNNs) with CNNs. The RNN, particularly the Long Short-Term Memory (LSTM) variant, allows the model to remember and utilize past frames to improve its current pose estimation, effectively handling the temporal nature of video sequences. DeepVO is sensor-agnostic and can work with monocular or stereo camera data, although it was initially designed and primarily tested with monocular input. Its end-to-end deep learning approach means it does not require explicit feature extraction and matching, which can be a strength when handling complex and less-textured environments where traditional feature-based methods may struggle. Nevertheless, as a data-driven approach, DeepVO's performance heavily relies on the availability of large, diverse, and high-quality training data, which can be a limiting factor. The system must be trained with a diverse range of environments, lighting conditions, and motion patterns to generalize well in real-world conditions. Furthermore, DeepVO, like most deep

learning-based approaches, requires significant computational resources, making real-time application challenging on devices with limited processing power. It also struggles to provide a globally consistent map, a common issue in VO-based systems that lack a loop-closure mechanism.

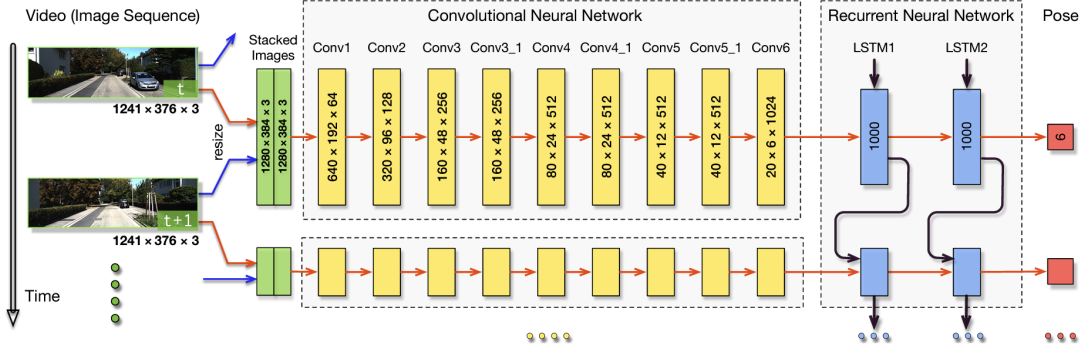


Figure 2.8: Architecture of DeepVO [52].

The mentioned DSO on the geometric methods section was used as a basis for the Deep Virtual Stereo Odometry (DVSO) [65] architecture. DVSO builds on the windowed sparse direct bundle adjustment formulation of monocular DSO by integrating it into a deep learning-based framework. DVSO achieves state-of-the-art results comparable to those achieved by stereo VO methods while only using monocular images. DVSO leverages the proposed StackNet, a semi-supervised monocular depth estimation network, to eliminate the scale drift due to scale unobservability. StackNet takes a left monocular image as the input and predicts both the left and right disparities in a virtual stereo setup, Fig 2.9.

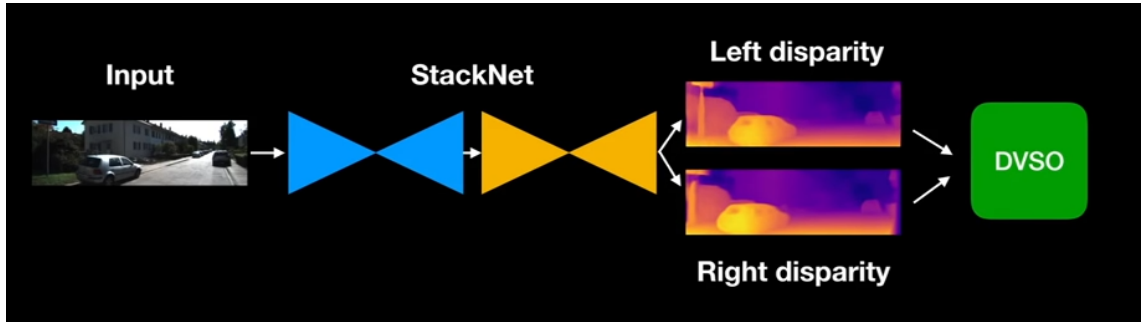
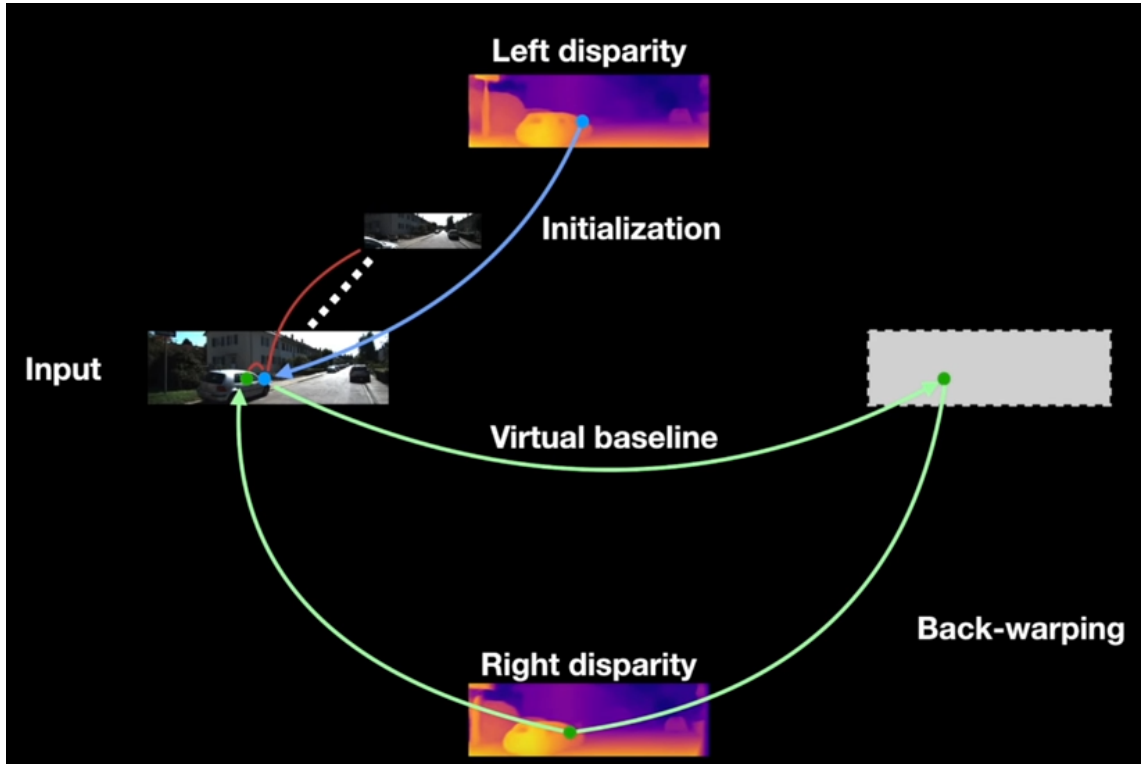


Figure 2.9: StackNet processing an image.²

For every new keyframe, the depth of reconstructed points is initialized using the left disparity. The optimization is done with a given virtual stereo baseline. The estimated points are then projected onto a virtual stereo right image. Then the projected right image is compared to the right disparity, which helps project the points to the left image. The loss function is given by optimizing the total photometric error, Fig 2.10.

²https://www.youtube.com/watch?v=sLZ0eC9z_tw.

³https://www.youtube.com/watch?v=sLZ0eC9z_tw.

Figure 2.10: General overview of DVSO.³

Another two years later, in 2020, Yang *et al.* [74], the same authors of DVSO, presented Deep Depth, Deep Pose and Deep Uncertainty, also known as D3VO.

D3VO [74] is a robust self-supervised framework for monocular visual odometry that exploits deep networks on three levels – deep depth (D), pose (T_t^{t-1}) and uncertainty (Σ) estimation. The predicted properties of depth (D), pose (T_t^{t-1}) and uncertainty (Σ) are then used in the tracking front-end, as well as in the photometric bundle adjustment backend, Fig 2.11. D3VO estimates depth using a Convolutional Neural Network (CNN) [6] called DepthNet [59] [55]. DepthNet is similar to StackNet in that it also predicts the uncertainty associated with the predicted depth estimates. Camera pose is estimated using PoseNet [45]. In this approach, the camera pose is represented as a combination of a translation, t , and a rotation q , in quaternion format, $p = [t, q]$.

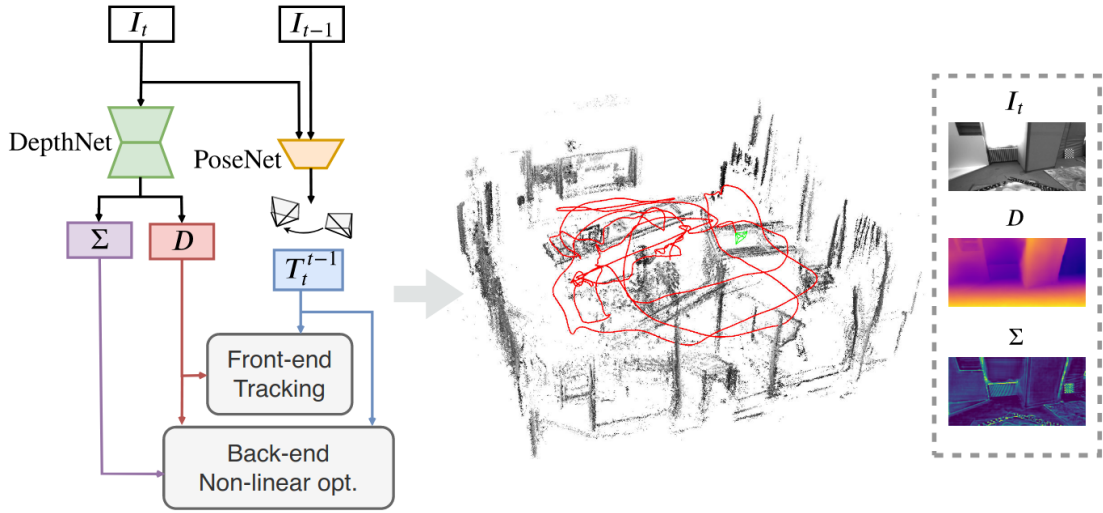


Figure 2.11: Overview of the D3VO architecture [74].

The learning based approaches explored until now show examples of how models use CNNs and RNNs to extract 6D pose from images. However, since the development of the Transformer architecture, they may not be the best suited to handle the task of VO. Transformers are a type of neural network architecture that was introduced in 2017 by Vaswani *et al.* on the paper "*Attention Is All You Need*" [50]. They are encoder-decoder models that process sequential data such as natural language text or time series data. Transformers have found immense success in natural language processing [57] [63] because they can capture complex relationships between different parts of the input using a mechanism known as multi-head self-attention. Since their creation, transformers have been used for various tasks like sentiment analysis, machine translation, word prediction, and summarization.

To understand self-attention and why it makes transformer models so unique, it is core to understand the difference between traditional encoder-decoder architectures and common attention mechanisms. As the name implies, in encoder-decoder architectures, there are two parts. The first part is the encoder. The encoder is usually composed of a network which is trained on input sequences to obtain a large summary vector c with a fixed dimension. This vector, called context, is usually a simple function of the last hidden state. The second part of the model is a decoder, another RNN, which generates predictions given the context c and all the previous outputs. In traditional encoder-decoder architectures, only the last hidden state (context vector c) from the encoder is passed on to the decoder. With this approach, in particular, the information acquired has to be compressed into a fixed-sized vector, which leads to information loss. Primarily the information found early in the sequence. A quick fix for this is the addition of bi-directional layers, which help solve the problem for shorter input sequences but do not do much in longer ones. Attention fixes this by allowing the decoder to attend to the whole sequence and, thus, use its entire context.

When Bahdanau *et al.* [34] first introduced attention, they found that the model could learn to focus on specific parts of the input sequence when generating the output sequence. Attention

allows a model to weigh the importance of different input parts when making a prediction, which can improve performance in specific tasks. Regular decoders without attention predict y_t given a fixed length context vector c and all earlier predicted words $\{y_1, \dots, y_{t-1}\}$. The fixed-length context vector is computed with

$$c = q(\{h_1, \dots, h_T\}) \quad (17)$$

where $h_1, \dots, h_T$ are the hidden states of the encoder for the input sequence $x_1, \dots, x_T$ and q is a non-linear function. The attention model, on the other hand, changes the context vector c that a decoder uses for translation from a fixed-length vector c of a sequence of hidden states $h_1, \dots, h_T$ to a sequence of context vectors c_i . Attention mechanisms compare input to external memories or queries, which can be considered an additional piece of information that the model uses to inform its predictions or output. For example, in Attention-based Neural Machine Translation, the encoder encodes the input sequence and stores it in memory (this can be done in many ways, as described previously). The decoder then uses this memory to help generate the output sequence.

Self-attention is fundamentally different from attention. Self-attention is focused solely on the input sequence and its relations. In other words, it allows the model to learn the importance of each input part. In practice, attention mechanisms can be used in conjunction with self-attention in sequence-to-sequence models like encoder-decoder architectures. For example, an encoder that uses self-attention to capture the relationships between different parts of the input sequence and then an attention mechanism that is used to align the encoder output to the decoder input can help the model generate the output sequence more accurately.

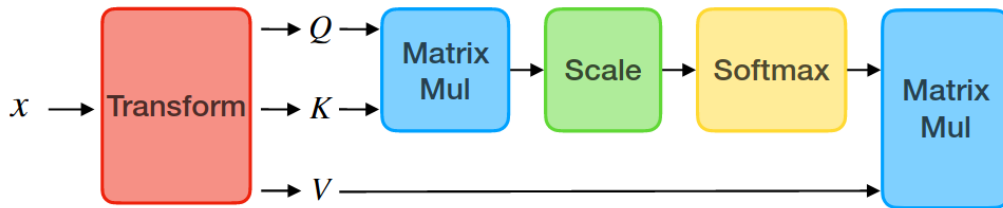


Figure 2.12: Self-Attention module [108].

Figure 2.12 shows the self-attention module processing the input X , which is a sequence of vectors represented as $(x_1, x_2, \dots, x_L)$, where $X \in \mathbb{R}^{L \times d}$ is true, and d is the embedding dimension of each vector. The model learns the relations between each vector by using three matrices; the Query (Q), Keys (K), and Values (V) matrices. These matrices are projected from X with linear layers. For instance, the query matrix, Q is obtained by projecting X with a linear layer W_q , i.e., $Q = XW_q$. Specifically, the attention weights, E , are calculated by the normalized product of Q and K^T with the softmax function [108],

$$E = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right). \quad (2.16)$$

The softmax function normalizes the attention weight of each query-key pair within $(0,1)$.

However, other functions can do neuron activation; ReLU (Rectified Linear Unit), the most commonly used one, and the sigmoid function. The purpose of this step is to categorize how important is each query-key pair. A pair with a value of 0 means useless information; on the other hand, a pair with a value of 1 means it is the most important. After this step, the output features, O , are enhanced by applying the attention weights E to V as follows [108]:

$$O = \text{Attention}(Q, K, V) = EV = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V. \quad (2.17)$$

What makes transformer models so powerful is the already mentioned multi-head self-attention. This specific type of attention gives the transformer model the ability to attend to different parts of the input sequence at the same time. To learn variant representations at different positions, the model transforms the input X into nh different representations (heads), denoted by $h_1, h_2, \dots, h_n$. The attention is first computed for each head h_i with Q^i , K^i , and V^i with projection matrices W_q^i , W_k^i , and W_v^i respectively.

$$h^i = \text{Attention}(Q^i, K^i, V^i). \quad (2.18)$$

Afterward, all the heads are concatenated to form the multi-head representations H as follows [108]:

$$H = \text{Concat}(h_1, h_2, \dots, h_{n_h}). \quad (2.19)$$

The concatenation of the multiple heads can be done in several ways. One way requires that the output of each attention head is converted into a 2D tensor with a specified number of rows and columns. Then, the outputs of all the heads can be concatenated along a specified axis, typically the last axis. Another way this can be done is by stacking the output of all attention heads one below the other and creating a 3D tensor. Next, H needs to be converted back to the dimension of d so that the projection matrix W^O is obtained, i.e., $O = HW^O$. As the number of relations between inputs is usually unknown, multi-head attention is commonly used to capture different connections between input elements in a data-driven manner [108].

The Transformer model proposed by Vaswani *et al.* [50] was made up of an encoder and decoder block. Each block can be decomposed further into 1) self-attention, 2) Position-wise Feed-Forward Networks, and 3) positional encoding.

Position-wise Feed-Forward Networks (FFNs) are mainly composed of Fully-Connected layers. Considering that the input X to the FFN is a 2D matrix of dimensions $N \times d$, where N is the number of tokens in the input sequence and d is the dimensionality of the input, the FFN can be written as:

$$\text{FFN}(X) = \delta(XW_1 + B_1)W_2 + B_2, \quad (2.20)$$

where $W_1 \in \mathbb{R}^{d \times h}$ and $W_2 \in \mathbb{R}^{h \times d}$ are the weight matrices, and $B_1 \in \mathbb{R}^{1 \times h}$ and $B_2 \in \mathbb{R}^{1 \times d}$ are the bias vectors. δ is the ReLU activation function, and h is the hidden layer dimension, usually set to $4d$ [108].

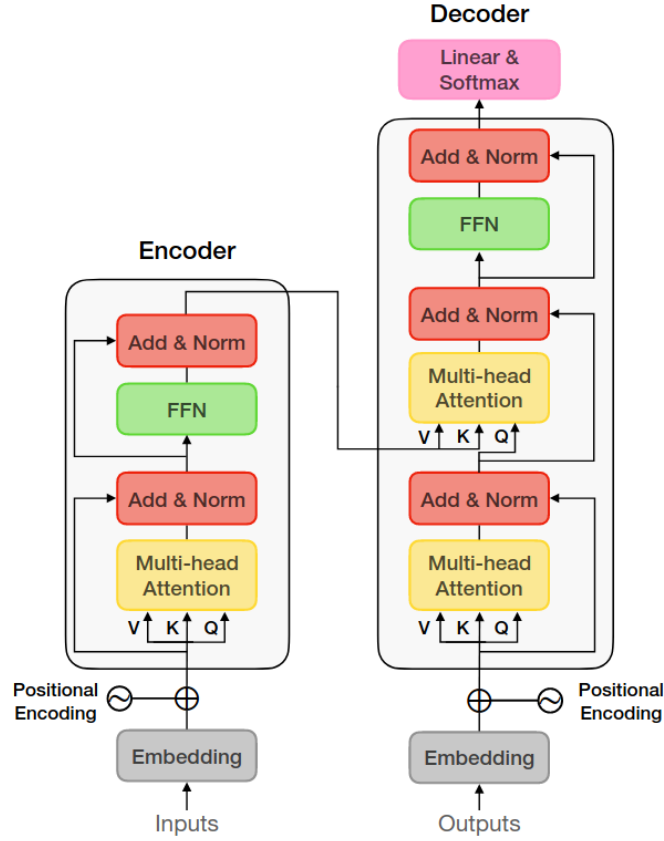


Figure 2.13: A simple transformer architecture with encoder-decoder [108].

Self-attention does not take into consideration the inputs' sequential order. To counteract this effect, some techniques can be applied. Positional encoding, figure 2.15, maintains the ordinal information by passing the input data through an embedding layer that converts the input tokens (e.g., words or subwords) into a continuous vector representation, denoted as X_p . Positional encodings can be further divided into fixed and learnable position encodings. The key difference is that learnable positional encodings have the ability to get updated with each pass through the model, allowing them to capture intricate positional dependencies specific to the trained data. This approach is present in the implementation proposed by Devlin *et al.*[57]. On their bidirectional Encoder Representations from Transformers (BERT) approach, a learnable class token $x_{class} \in \mathbb{R}^{1 \times (P^2 C)}$ is created and added to X_p . This information is then added to the output - $[x_{cls}; X_p]$ and provides the model with positional information [71], figure 2.14.

On the other hand, fixed encodings, such as those used in the original Transformer model, utilize sinusoidal positional encodings. These encodings add a unique positional signal to each input token, helping the model to understand the position of each token within the sequence. The principle behind sinusoidal positional encodings is to assign each position a vector that corresponds to a point in a high-dimensional sinusoidal wave. The unique property of this type of encoding is that it allows the model to infer the relative positions of the tokens in the sequence. Because the frequency of the sinusoidal wave varies along the dimensions of the positional encoding vector,

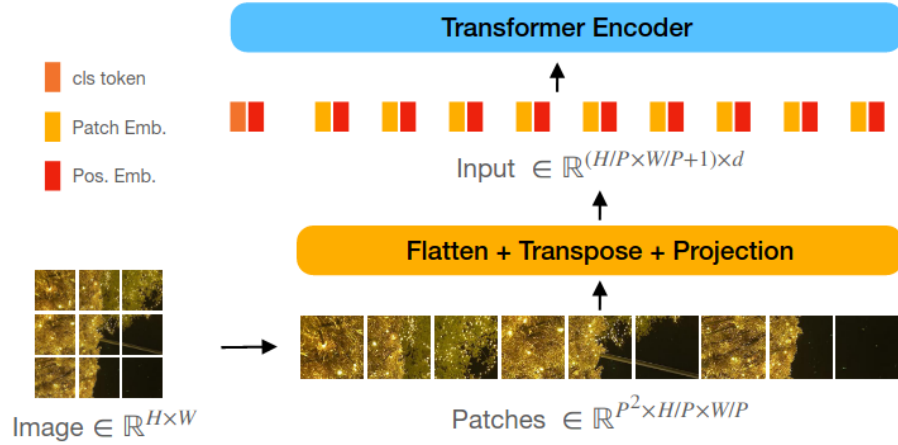


Figure 2.14: Patch and Positional Embedding [108].

the model can capture the periodic nature of the sequence and make accurate predictions. Despite their effectiveness, fixed encodings do have limitations. They do not allow for learning from the data and updating the encodings during training, which can restrict the model's ability to adapt to the specificities of the input data. Moreover, they may not be ideal for sequences that significantly exceed the length the model was originally trained on, as the positional encodings for those positions would be extrapolated and may not represent the positional information accurately.

Another way to keep track of the order of the inputs is through segment embeddings, figure 2.15. This can be done by incorporating additional information about the input sequence into the Transformer model. The basic idea is to add an extra embedding for each token in the input sequence, representing the segment that the token belongs to. The segment embedding can indicate different types of information, such as the source of the input (e.g., a question or an answer in a Q&A model) or the structure of the input (e.g., the beginning or end of a sentence in a language model). A segment embedding is passed along with the input tokens during training. The Transformer model learns to consider the difference in embedding when processing the tokens in the input sequences. In the particular case of transformers applied to visual odometry, segment embeddings can provide additional information about the structure and layout of the scene. One way to use segment embeddings in a transformer for visual odometry is to extract features from the images in the sequence, such as SIFT [10] or ORB [56] features, and then cluster these features into segments. The transformer can then use these segment embeddings, along with the other input data, to better understand the relationships between the images in the sequence and make more accurate estimates of the camera's ego motion.

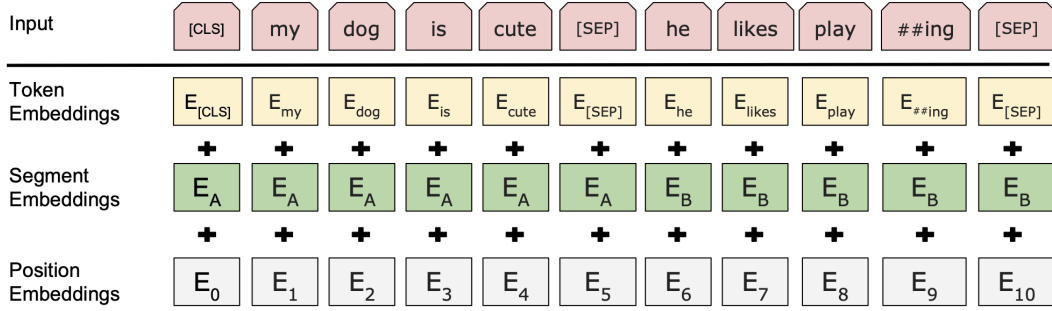


Figure 2.15: Visual representation of segment and position embeddings [57].

2.1.3 Vision Transformers

Following the immense success verified in natural language processing, Transformers applied to vision tasks started to appear. There are quite a few use cases for vision transformers, verified by the works developed in the following areas: image classification [71] [93], video classification [77], object detection [70] [85], semantic segmentation [98] [99] and pose estimation [104], but before Dosovitskiy *et al.* [71], there were some complex challenges that were affecting vision Transformers.

Image classification tasks require machine learning models to establish some connection between the different pixels of an image. If this task is approached naively, there would be connections to be made between every pixel of an image. And even if it is possible to achieve results with this approach, it is by far not the most appropriate, as establishing connections between high-resolution images with millions of pixels would be very time-consuming ($O(n^2)$). The first tries at implementing a "smart" algorithm were done by Parmar. Niki Parmar *et al.* [62], applied the self-attention only in local neighborhoods for each query pixel instead of globally, which, as seen in [66], [69], and [75], can completely replace the convolutions from Convolutional Neural Networks (CNNs). After Parmar *et al.* [62], Rewon Child *et al.* [34] implemented a Sparse Vision Transformer, which tried to attain global self-attention with the help of scalable approximations. However, these attempts failed to provide a simple enough transformer architecture that achieved comparable results to state-of-the-art CNNs. Dosovitskiy *et al.* [71] fixed these problems by proposing the following Transformer architecture made of three modules i) patch and positional embedding, ii) a Transformer encoder, and iii) multi-layer perceptron (MLP) head.

The patch and positional embedding is the first step of the Visual transformer architecture proposed by Dosovitskiy *et al.* [71]. This step transforms an image, $X \in \mathbb{R}^{C \times H \times W}$, where C is the number of channels (3 for RGB and 1 for grayscale), H is the height and W is the width, into a 1D sequence of vectors, represented by $X_p \in \mathbb{R}^{N \times (P^2 C)}$. In this equation, $P \times P$ is the resolution of each patch, and the number of patches is $N = HW/P^2$. After this, a linear layer projects the patches into patch embeddings, and thus the first step is almost complete, lacking only information about each patch position. To add this information, the Visual Transformer uses the same learnable

encodings which are applied to the BERT transformer mentioned above. The resulting patch embeddings with the added positional information is passed on through the transformer layers. The successive transformations that happen inside the transformer layers are described by the following mathematical expressions:

$$z_0 = [x_{cls}; x_p^1 E; x_p^2 E; \dots; x_p^N E] + E_{pos}, \quad E \in \mathbb{R}^{(P^2 C) \times D}, E_{pos} \in \mathbb{R}^{(N+1) \times D} \quad (2.21)$$

$$z'_\ell = \text{MSA}(\text{LN}(z_{\ell-1})) + z_{\ell-1}, \quad \ell = 1 \dots L \quad (2.22)$$

$$z_\ell = \text{FFN}(\text{LN}(z'_\ell)) + z'_\ell, \quad \ell = 1 \dots L \quad (2.23)$$

$$y = \text{LN}(z_L). \quad (2.24)$$

Where z_0 represents the initial input to the transformer layers, which consists of the class token x_{cls} and the patch embeddings $x_p^i E$ for i from 1 to N . $[x_{cls}; x_p^1 E; x_p^2 E; \dots; x_p^N E]$ denotes the concatenation of the class token and the patch embeddings. E is a learnable parameter and represents the embedding matrix that is used to transform the patches into the required dimensionality D . E_{pos} is the positional encoding added to the input sequence, and helps ensure that the positional encoding aligns with the size of the patch embeddings and class token. This entire first expression creates the initial sequence with the positional information to be fed into the transformer layers.

The second equation describes the standard transformer operations of normalization, self-attention, and addition of the residual connection. Here, z'_ℓ represents the output of the $\ell - th$ layer after applying the multi-headed self-attention (MSA) and layer normalization (LN), and adding the original input (residual connection) from the $(\ell - 1) - th$ layer. The segment, $\text{MSA}(\text{LN}(z_{\ell-1}))$ applies the layer normalization first, followed by multi-headed self-attention, where $z_{\ell-1}$ denotes the input to the $\ell - th$ layer. In the third expression, z_ℓ represents the output of the $\ell - th$ layer after applying a Feed Forward Network (FFN) after another round of Layer Normalization (LN) to z'_ℓ , while also adding the intermediate result z'_ℓ (residual connection). The last expression is the final normalized output of the transformer layers, which is ready to be fed into the next stage, either a classifier or some other form of prediction layer. The final output of the $\ell - th$ transformer layer, after applying LN to z_L , is described by y .

The visual Transformer developed by Dosovitskiy *et al.* [71] does not have a decoder module, as the model does not make an autoregressive prediction. Instead, the ViT (Visual Transformer) only has an encoder step, intending to find a better representation of each image patch. This representation is classified with the help of tokens. The other component of self-attention is the MLP Heads, which are a type of fully connected network, meaning that each node is connected to every other node in the subsequent layer, creating an architecture that consists of at least three components; an input layer, a hidden layer, and an output layer. MLPs can also be entirely made up of one or more linear layers. In the particular case of the standard ViT, two MLP layers with GELU [44] non-linearity [71] is used.

The Visual Transformer (ViT) introduced a novel approach to image classification, though

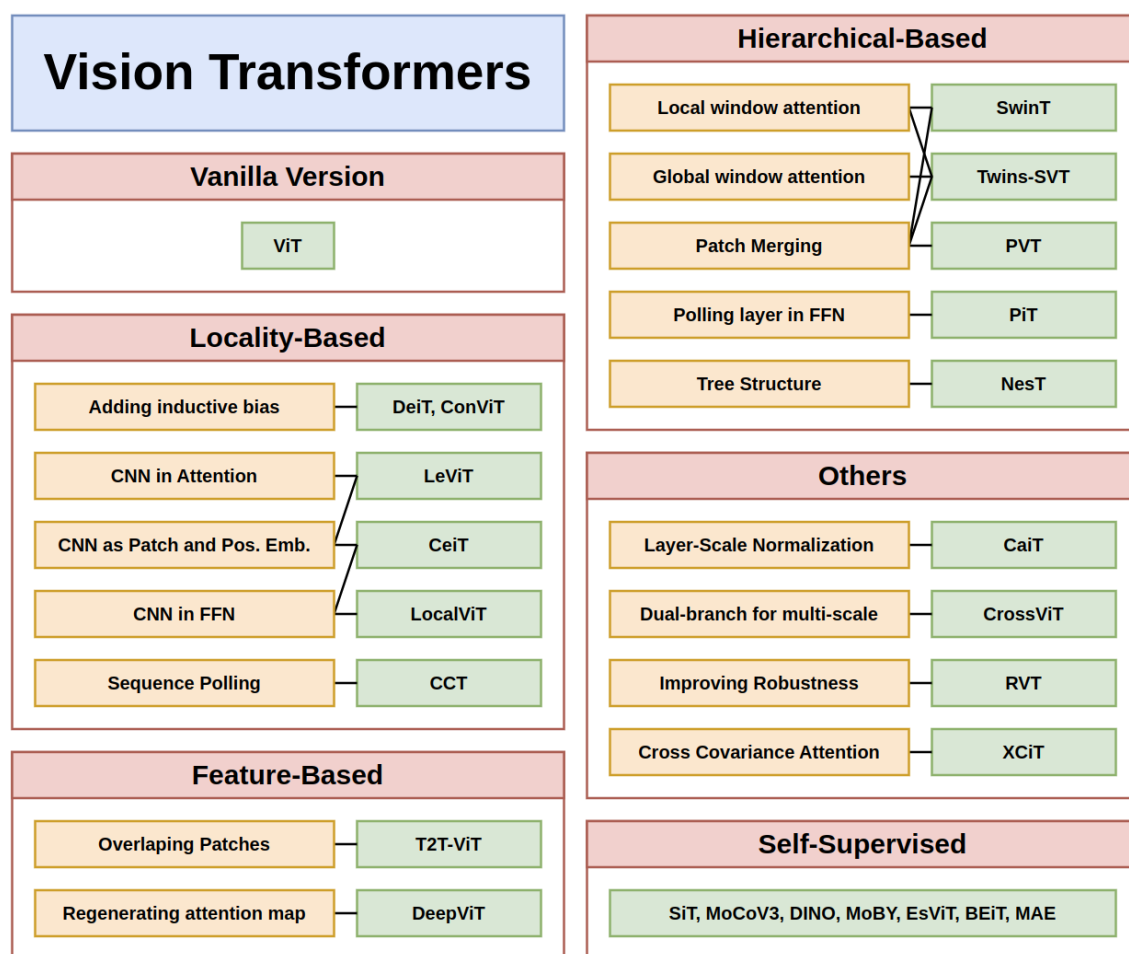


Figure 2.16: Taxonomy of Vision Transformers [108].

it presented several limitations. Notably, its dependence on large datasets for effective training and its restricted application beyond image classification, including tasks such as object detection and image segmentation. Despite these issues, the ViT initiated the exploration of Visual Based Transformers, leading to the development of various alternative architectures. These architectures aim to address the shortcomings of the ViT model, resulting in four primary categories: locality-based, feature-based, hierarchical-based, and self-supervised learning models. Additional models that do not strictly adhere to these classifications have also made significant contributions to the field.

A key limitation of the original Visual Transformer pertains to its limited use of locality, a crucial characteristic of Convolutional Neural Networks (CNNs). In response to this, the locality-based category emerged, creating the following example architectures, which include the Data-Efficient Image Transformer (DeiT) [73], ConViT [107], LeViT [86], CeiT [100], LocalViT [92], and CCT [87]. These models typically incorporate convolutional layers to extract local features, imitating CNNs' structure to some extent; this can decrease input size, expedite inference time, and help the models handle large images more efficiently. Each architecture provides unique strategies to bridge the gap between transformers and CNNs. For example, the DeiT model utilizes a

teacher CNN model and knowledge distillation for training a vision transformer, seeking to enhance its performance by reproducing the output of the teacher model during the student model's training phase, therefore integrating the inductive biases of the teacher model. The ConViT model, similar to DeiT, integrates the inductive bias of CNNs into its structure. However, in contrast to DeiT, ConViT employs Gated Positional Self-attention (GPSA) that can be initialized as a convolutional layer. LeViT and CeiT, on the other hand, derive their embeddings from convolutional layers, enabling local feature extraction and input size reduction. LeViT further reduces the input size in some attention blocks, thereby accelerating inference time. CeiT extends this concept by using convolutional layers to create query, key, and value tensors prior to self-attention, hence allowing the model to include local spatial information. LocalViT utilizes convolutional layers in each Feed Forward Network (FFN) to extract local features for every transformer block. This model applies varied activation functions and FFN layer architectures, facilitating the efficient processing of large images. Lastly, the CCT model extracts embeddings via convolutional layers, eliminating the need for positional embeddings. This approach highlights that transformers can operate effectively without extensive computational resources, making them applicable even in more constrained settings common in certain scientific domains with limited data availability.

Feature-based Visual Transformers emphasize feature diversification, such as token maps and attention maps, in vision transformers. By generating distinct feature maps, the model is enabled to extract a variety of features, enhancing its performance [108]. The advancement in CNN models can be largely attributed to the training of highly layered models, facilitated by innovative architecture designs. The depth of a CNN enables the learning of richer and more intricate representations of input images, improving performance on vision tasks. DeepViT [103] replicates this behavior, utilizing different heads as a foundation and re-formulating the attention maps through the transformation matrix. This approach promotes inter-head information communication, allowing the re-formulated attention maps to encode more extensive information. The Tokens-to-Token ViT (T2T-ViT) [101], in contrast, employs a unique form of tokenization. Rather than using the standard tokenization approach employed in ViT, it utilizes a progressive tokenization module that aggregates neighboring tokens into a singular token. This strategy aids in learning the local structural information of surrounding tokens and iteratively reduces the length of tokens. Another key finding from this study is that the attention backbone of ViT is not optimally designed for vision tasks in the same way CNNs are. It exhibits redundancy which leads to restricted feature richness and increased difficulties during model training.

The ViT model is known for its computational intensity. To address this, a range of models classified as Hierarchical Based have been proposed, each employing unique strategies to reduce the computation load. The Pyramid Vision Transformer (PVT) [97] is a pioneering pure Transformer backbone model designed specifically for pixel-level dense prediction tasks such as object detection and semantic segmentation. It utilizes a Spatial Reduction Layer to lower the dimension of K and V . The PVT model offers three primary advantages over ViT [71]: it can generate different channel feature maps at different stages; it can be easily integrated with most downstream task models; and it is more computation and memory efficient, capable of handling higher resolution

feature maps or longer sequences. In terms of parameter count, the PVT model has demonstrated superior performance compared to well-designed CNN backbones. The Pooling-based Vision Transformer (PiT) [89] employs a depth-wise convolutional layer for dimension reduction in its pooling layer. This design mirrors a principle widely used in CNNs, where spatial dimensional transformation is performed by pooling or convolution with strides. The Swin-Transformer [93] constructs hierarchical feature maps by merging image patches in deeper layers. The model's linear computation complexity relative to input image size is due to the computation of self-attention within each local window only. This structure allows the Swin-Transformer to achieve robust performance across a range of tasks, including image classification, object detection, and semantic segmentation. Its key component is the shifted window-based self-attention, which is both efficient and effective for vision tasks. The Twins-SVT, a result of the study in [84], also computes attention within a shifted window. However, it performs the global attention computation following the local window attention. The final architecture within the hierarchical-based category is the Nested Hierarchical Transformer (NesT) [102]. NesT divides an image into n blocks, merging every four blocks post a transformer layer. The model introduces Gradient-based Class-aware Tree-traversal to visualize the most critical path from child to root, illustrating how the model makes decisions based on a given input image. Interestingly, due to its tree structure, NesT can generate images by reversing the tree direction [108].

Several transformer architectures do not adhere to a specific classification type but propose interesting directions to enhance visual transformers.

The core innovation in CrossViT [82] lies in its encoder modules. Each encoder module in CrossViT is dual-branch, containing a large (L) and a small (S) branch. This structure enables the model to capture spatial information at varying scales, which are then fused through cross-attention. CaiT [96] introduces a significant modification in the normalization step. The model employs *LayerScale*, which multiplies the output vector of each residual block per channel, rather than using a single scalar. This form of normalization expedites convergence and enhances the training of deep models. CaiT also integrates a Class Attention Layer, calculating the attention between the class embedding and the comprehensive features to facilitate a deeper understanding of the inputs. Developed with a robustness emphasis, the Robust Vision Transformer (RVT) [94] proposes three main ideas: *i*) excluding the class token, deemed unimportant to the vision Transformer; *ii*) incorporating a CNN into the embedding and FFN layers; *iii*) utilizing more attention heads to extract more features. The model substitutes the conventional self-attention module with Position-Aware Attention Scaling (PAAS), adding a learnable matrix to self-attention to highlight the importance of each $Q - K$ pair. The Cross-Covariance image transformer (XCiT) [95] primarily targets reducing the self-attention time complexity. It accomplishes this by substituting the self-attention module with Cross-Covariance Attention, which applies the transposed version of self-attention. In this case, the self-attention is calculated on the feature channels, not on the tokens.

Self-Supervised Learning (SSL) implementations have also emerged, allowing models to learn from data without requiring annotations. Examples include SiT [78], MoCoV3 [83], DINO [81],

MoBY [99], EsViT [91], BEiT [79], MAE [88].

While each category of vision transformers presents promising characteristics for various tasks, including visual odometry, the Locality-based and Hierarchical-based architectures stand out due to their unique combination of features. The Locality-based models leverage convolutional ideas for the effective extraction of local features, which is integral to understanding spatial relations in visual odometry. Meanwhile, Hierarchical-based architectures employ hierarchical attention mechanisms that allow varying levels of detail to be captured, a crucial factor for accurate motion estimation.

2.2 Critical Review

In recent years, visual odometry (VO) has seen significant advancements with the advent of machine learning-based approaches. This progress comes after an era of geometric techniques, that, while beneficial in specific contexts, still lack the ability to generalize across diverse environments.

In table 2.1, it is possible to see how some of the explored methods stack up against each other.

	$t_{rel}(\%)$	$r_{rel}(^{\circ})$	Category
ORB-SLAM3	0.87	0.27	Geometric
D3VO	0.88	0.21	Learning
DVSO	0.9	0.21	Learning
ORB-SLAM2	1.15	0.27	Geometric
DeepVO	5.96	6.12	Learning

Table 2.1: Table of five relevant methods. All the implementations are vision based. Each metric is taken from the respective paper where they were first described.

As displayed in Table 2.1, the relative translational (t_{rel}) and rotational errors (r_{rel}) serve as performance indicators for various VO methods, namely ORB-SLAM3, D3VO, DVSO, ORB-SLAM2, and DeepVO. It is important to note how the learning-based approaches compare to geometric ones. Looking at the relative translational errors (t_{rel}), D3VO and DVSO, both learning-based techniques, do not significantly outperform ORB-SLAM2 and ORB-SLAM3, which are more traditional, geometric-based methods. In fact, the ORB-SLAM3 implementation even marginally surpasses D3VO. The same pattern is observed with the rotational errors (r_{rel}), suggesting that the advances in learning-based methodologies have yet to clearly surpass the traditional geometric methods in this regard. The exception to this pattern is DeepVO, another learning-based approach that exhibits significantly higher errors, both in translation and rotation, than any of the other methods in the table. This discrepancy further emphasizes that while learning-based VO techniques are undoubtedly promising, they are currently not a definitive improvement over traditional methods. This could be attributed to the relative novelty of learning-based VO, which has not been around as long as geometric techniques, and therefore has not had as much time for refinement and improvement. Another reason why learning-based approaches are still lagging behind may have to

do with the fact that the architectures employed, like CNNs, are not good at representing sequential information, while RNNs are computationally heavy and are incapable of capturing the image features. Even when the two are combined, the resulting architecture is not always optimal.

Given the limitations of the existing VO methods, it becomes evident that there is a significant opportunity for improvement, particularly with the use of transformer models. Vision transformers are particularly well-suited to extract features from images while efficiently handling sequential data, a task that traditional methods struggle to accomplish. Furthermore, unlike recurrent neural networks (RNNs), transformers stand to benefit from an increase in data size. Despite the substantial progress in vision transformers, certain issues persist. Specifically, *i*) there is an absence of universal pre-trained weights. The adaptation of self-attention in transformers, initially designed for NLP problems, to vision, is still not thoroughly explored. *ii*) Feature collapsing, which constrains the model's ability to produce diverse representations, can hinder training for deep vision transformers. While models like T2T-ViT and DeepViT address this issue by adding extra modules or utilizing learnable parameters, this can increase inference time — an undesirable outcome for applications such as ego-motion. Potential solutions involve modifying the architecture or training method or utilizing different data augmentations. *iii*) The model must be lightweight enough to operate on in-car computing technologies. *iv*) The issue of fixed positional embeddings size exists. Interpolation can be used to adjust embeddings size, but this might cause information loss when the input size deviates significantly from the training size. Currently, the most viable approach is to extract features directly from convolutional layers without adding positional embeddings, although this method lacks global positional information. *v*) Lastly, the model's robustness is critical. In real-world scenarios, input images may be altered or corrupted due to factors such as brightness, background, blur, noise, digital artifacts, or even malicious adversarial attacks.

Chapter 3

Visual PoseFormer

Traditionally, learning-based approaches estimate six-degree pose values - namely, x , y , z coordinates along with roll(ϕ), pitch(θ), yaw(ψ) angles. There are many ways to do this, but the problem can be simplified to two key components: *i*) extract features from the input images. DeepVO and D3VO are two learning-based designs that try to extract valuable information from the input images. DeepVO learns to extract random features by using CNNs, which may or may not be relevant, while D3VO explicitly extracts Depth and Optical Flow information. *ii*) Use the extracted information to regress continuous 6D pose values. This step is usually performed by a fully connected network like an MLP head.

3.1 Introduction

Due to the nature of the task, that is, visual odometry, ViTs can be seen as prime candidates to be the state-of-the-art deep learning approach since they have some very interesting features which could help them during the feature extraction step. ViTs, unlike traditional Convolutional Neural Networks (CNNs), which process data hierarchically, can utilize the power of the transformer layers to handle spatial and temporal dependencies in a global context manner. In other words, a patch of an image can be related to a completely different patch, something that can't happen in CNNs as they can only establish local relationships. Extracting the global context in ViTs is only made possible due to how self-attention works (explained in Chapter 2). This behavior is appreciated when dealing with visual odometry tasks, where understanding the interrelationship of different parts of the image and their evolution over time is very important for accurate pose estimation. An example of where this behavior can be beneficial is seen in figure 3.1, where different moving parts of the image, like a moving vehicle, can affect the outputs of a CNN, something which can be circumvented with the self-attention mechanisms, present in transformers. This type of situation is not the exception as most of the time the vehicle will have to face scenes with other moving objects. In these cases, the behavior of the transformer can prevent the model from giving too much importance to those objects which move at a different speed from the rest of the image and can negatively influence the quality of the predictions.



Figure 3.1: An image from the KITTI dataset [32], where there aren't a lot of good static reference points, and there is a moving vehicle on the scene.

After the transformer extracts some features, the next step is to regress 6D pose values, where, usually, and in the context of this thesis, a Multi-Layered Perceptron (MLP) is used. Still, it should be noted that a combination of linear layers and activation functions usually yields better results, allowing the MLPs to model non-linear dependencies contained in the data.

Combining vision transformers and MLPs is a compelling approach to solving odometry problems. The vision transformer is a competent architecture for handling the feature extraction part. Meanwhile, the MLPs act as a regressor, mapping the extracted features to specified continuous outputs. With enough computational resources and sufficient training data, this combination of networks has the potential to yield accurate odometry estimates.

3.2 Data Processing

All the work developed in this thesis was done on the colored KITTI odometry dataset [32]. KITTI incorporates a range of outdoor sequences captured in varying environments, which span urban and rural landscapes, as well as highway and residential regions. These sequences are organized into 22 colored stereo sequences. Ground truth (GT) trajectory data is available for 11 of these sequences (00-10), while the remaining sequences (11-21) are reserved for evaluation. Two cameras take images at a frequency equal to 10Hz. Each camera represents the left or right component of the stereo pair of RGB images, which facilitates the estimation of depth and 3D structure. The images do not have a fixed resolution, with some sequences containing images with resolutions of $Height, Width = 376, 1241$ pixels, while others have a resolution of $Height, Width = 370, 1226$. Having different heights and widths for images means that all images have to be resized. In this case, all images are resized to have $Height, Width = 256, 512$. The dataset further offers ground truth camera poses sourced from a highly precise GPS/IMU (Inertial Measurement Unit) system, LiDAR point clouds (which isn't important in this particular study), and calibration data for each sequence. The provided calibration data holds critical information regarding the camera's intrinsic and extrinsic parameters, assisting in the projection of 3D points onto the 2D image plane.

Odometry has an iterative nature, something that ends up reflected in the process of feeding images into the transformer model.

- At the most basic level, the model processes stereo images from consecutive instants. This results in pairs of images: two from instant t_1 and another two from instant t_2 , totaling four images for each prediction.
- Given that the KITTI dataset size is relatively small for transformer models to learn effectively, data augmentation methods were employed to enhance learning efficiency. Instead of using just two consecutive instances, instants that are more than one frame away are also used. Thus, the transformer models may receive moments t and $t + 1$, or t and $t + 2$, or t and $t + 3$, or t and $t - 1$.
- In addition to the stereo images, the model can also utilize optical flow [37][38][48] information as an extra input. This additional information is pre-calculated using the RAFT model [72], serving as another source of valuable data for the transformers. A visual representation of what optical flow looks like can be seen in figure 3.2

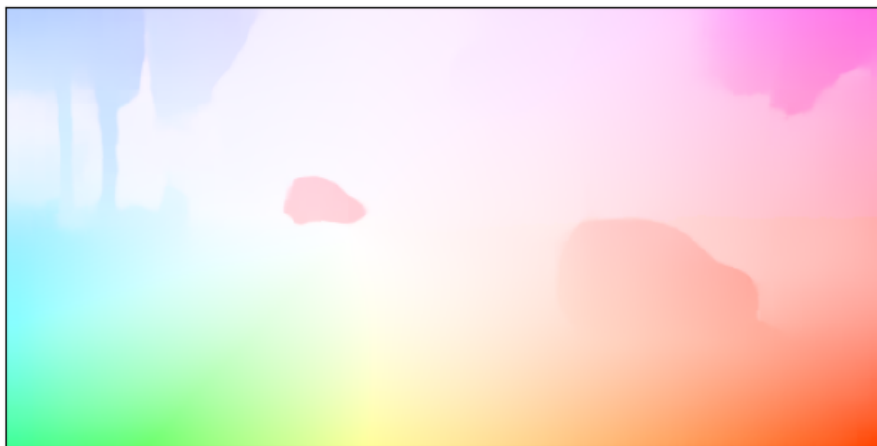


Figure 3.2: Optical flow for a pair of images. The color represents the direction of the movement; the saturation represents the value.

3.3 Model architecture

To thoroughly understand how transformer models perform at VO, three transformer models were developed from scratch and then tested. Different architectures mean that each model has something unique, which may grant it some advantage over the others, allowing for a better understanding of how each transformer component (different input shape, different attention mechanisms) affects the ability of the transformer to learn odometry tasks. Expanding upon these unique aspects, the specific characteristics of each of the three developed transformer models are described as follows:

- **PoseFormer2D (PF2D)**: This is the most basic version implemented, and it features 2D transformer layers followed by six MLPs tasked with regressing one of the 6D pose values.

The architecture can be used with fixed or learnable position encodings. Being a 2D architecture means that all the information is bundled into the channels' dimension, and this goes for images and optical flow.

- **PoseFormer3D (PF3D)**: This version has an extra time dimension in the input tensor. This means that information from instants t and $t + 1$ is stacked on the channels' dimension, as opposed to PF2D, where everything is bundled together.
- **Spatio-Temporal PoseFormer (STPF)**: This version is similar to PF3D since it also separates the information in instants but has two different transformer blocks, which attend to spatial and temporal dependencies differently.

Diving deeper into each architecture, the first to discuss is the PoseFormer2D (PF2D), depicted in figure 3.3. This architecture operates on two-dimensional data, encapsulating spatial information like the height and width of images. However, it does not incorporate the temporal dimension across different frames. As a consequence, time-varying information from multiple frames, such as a pair of stereo images at distinct moments, is merged into the channel dimension and represented as a 4-dimensional tensor $[Batches, Channels, Height, Width]$. The PF2D architecture is primarily composed of three steps. The initial step involves dividing the images into smaller patches. For an image I with dimensions (H, W, C) , a single patch of I is characterized by dimensions PH, PW, C , which represent the height, width, and channels of the patch respectively. This operation of creating patches can be symbolized as $partition(I)$, further detailed in two steps:

1. Calculate $H' = H/PH$ and $W' = W/PW$ by dividing the height and width of the image by the patch size.
2. For every $i \in \{0, 1, \dots, H' - 1\}$ and $j \in \{0, 1, \dots, W' - 1\}$, a patch $I_{ij} = I[i \cdot PH : (i + 1) \cdot PH, j \cdot PW : (j + 1) \cdot PW, :]$ is created. In this context, I_{ij} refers to a patch at position (i, j) , and the colon symbol $(:)$ at the end, indicates the incorporation of all channels in each patch.

Following the creation of patches, these segments are flattened into 1D arrays with dimensions $PH \times PW \times C$. Subsequently, the patches undergo Layer Normalization, a Linear Layer transformation, and another Layer Normalization, before positional encodings are added. PoseFormer2D has two types of mutually exclusive encodings which may either be tokens that are learned during training or fixed values generated using sine and cosine functions.

The next step, and arguably the most important one, is the calculation of attention scores and conversion of that to useful information. First, the model calculates three vectors from the input, the Query (Q), Key (K), and Value (V). It obtains these vectors by applying a linear transformation to the input described by the equations - $Q = W_q X, K = W_k X, V = W_v X$, where W_q, W_k , and W_v are the weight matrices learned during the training process, and X is the input tensor. The attention score is given by the expression 2.17, explored earlier on chapter 2. The last step is using the output of the transformer layers to regress the 6D pose through MLPs.

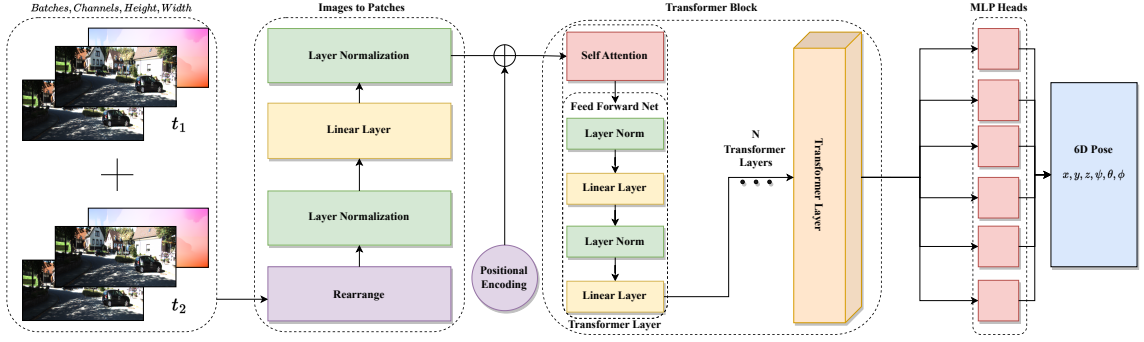


Figure 3.3: Architecture of the PoseFormer2D. The input stereo images are processed and transformed into a 1D vector representation where positional encodings are added. The tensor then goes through a transformer block composed of N transformer layers. The resulting tensor with the added attention values is used by the MLP heads to regress the 6D pose.

The second model, PoseFormer3D (PF3D), figure 3.4, expands on the PF2D, allowing the model to handle temporal data. This happens because the input now takes the form of $[Batches, Channels, Frames, Height, Width]$, where $Frames$ corresponds to the number of different pairs of stereo images fed to the model. The $partition(I)$ operation is slightly different in this case. Here each 3D patch V_{ijk} is equal to $V[i \cdot PF : (i + 1) \cdot PF, j \cdot PH : (j + 1) \cdot PH, k \cdot PW : (k + 1) \cdot PW, :]$, where PF is the frame patch size.

Besides that difference, both PF2D and PF3D have the same three main steps and follow the same principles as they are both based on the original transformer model proposed by Vaswani *et al.*[50] and also follow some important principles, like having global average pooling instead of class tokens, having no dropout, that are present in Beyer *et al.*[106]. The 3D model ends up being an extension of the 2D model. Besides using fixed positional encodings, both models can use learnable position encodings. Learnable position encodings can prove advantageous over fixed-value position encodings, since they are updated during training, and can potentially learn position information that is more useful for visual odometry. Learnable encodings are not necessarily better than fixed positional encodings, as each type has its advantages and disadvantages, *e.g.*, learning encodings can lead to overfitting and can also make the model take longer to converge.

Lastly, the Spatial-Temporal Pose Former (STPF), figure 3.5, is heavily inspired by Arnab *et al.* [77], and differs from the previous two models by having different transformer blocks attend different parts of the input tensor $[Batches, Channels, Frames, Height, Width]$. In the previous two implementations, a single transformer block calculates the attention values for the spatial and temporal dimensions. STPF is different since it has two blocks of transformer layers, where each block calculates attention values for its designated dimension - space or time. Furthermore, STPF's way of adding positional information is through learnable positional encodings. Since this model has transformer blocks that calculate attention values for space, the patches here are also different. For STPF, each image is converted to a patch where the time dimension is more important. This creates a patch T_{ijk} equal to $T[i \cdot PF : (i + 1) \cdot PF, (j \cdot PH : (j + 1) \cdot PH, k \cdot PW : (k + 1) \cdot PW), :]$

All three implementations - PF2D, PF3D, and STPF - have six MLP heads, figure 3.6 as their

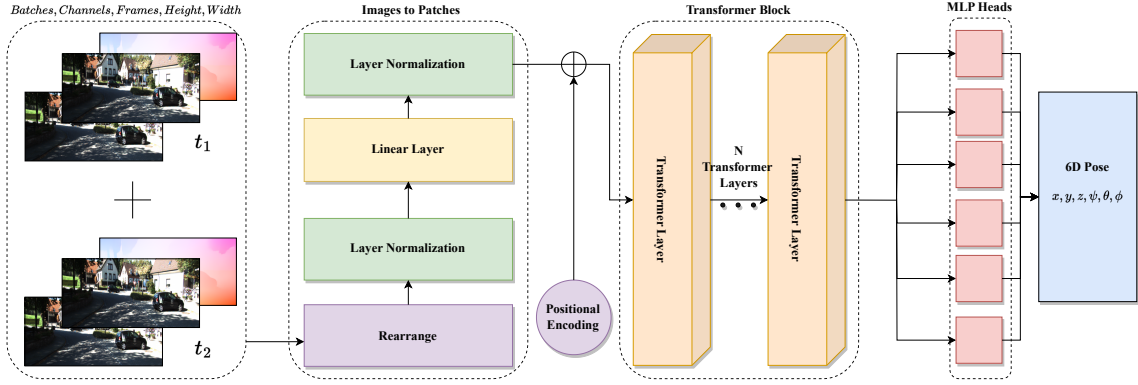


Figure 3.4: Architecture of the PoseFormer3D. The input stereo images are processed and transformed into a 1D vector representation where 3D positional encodings are added. The tensor then goes through a transformer block composed of N transformer layers. The resulting tensor with the added attention values is used by the MLP heads to regress the 6D pose.

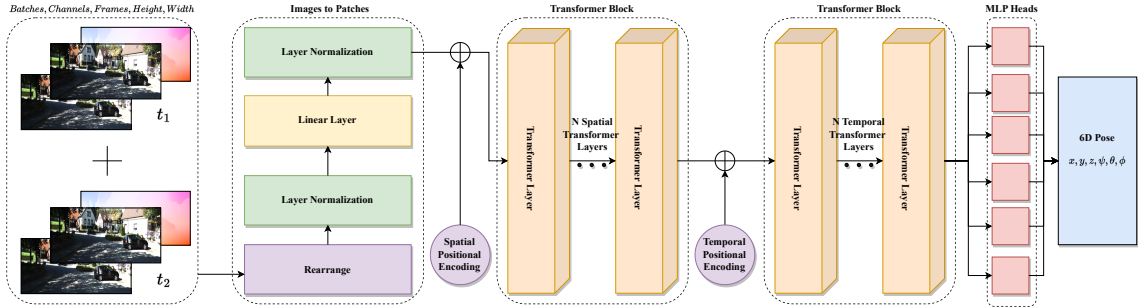


Figure 3.5: Architecture of the Spatial-Temporal Transformer. The input stereo images are processed and transformed into a 1D vector representation where spatial position encodings are added. The tensor then goes through a spatial transformer block composed of N transformer layers. Before the output goes through another transformer block focused on finding temporal relationships, temporal tokens are added. The output of this block is fed to the MLP heads to regress the 6D pose.

regressors where each head makes use of a combination of linear layers, layer normalization, and activation functions to regress one value of 6D pose, figure 3.6. In each MLP the input tensor has the dimension of size, dim , decreasing by half for each linear Layer. The final linear Layer has $\frac{dim}{4}$ input size and 1 output size. Each dropout layer has 0.5 chance to convert a value to zero. This amount of dropout helps to prevent overfitting during training, by forcing the model to learn redundant information.

3.4 Loss Function

In autonomous driving scenarios, the magnitude of longitudinal movements (moving forward and backward) is more prevalent than lateral (side-to-side) or height changes. However, the direction of each movement is also crucial for navigational purposes. A slight error in the direction over

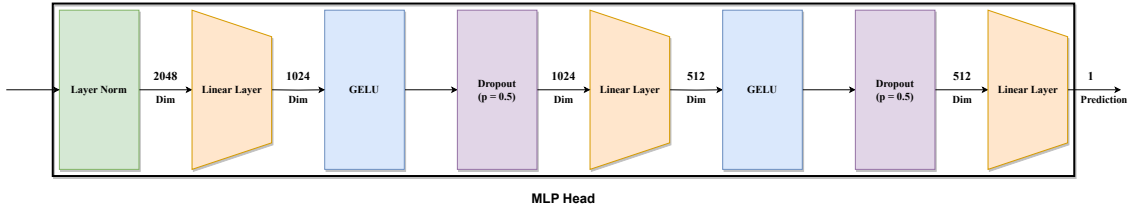


Figure 3.6: A single MLP Head, tasked with regressing a single value of the output 6D pose.

a long distance can result in significant positional discrepancy, potentially leading to a dangerous situation. With these constraints in mind, the following loss function was used to train the models proposed:

$$L = w_t \left(\left\| \frac{P_t}{\|P_t\|} - \frac{T_t}{\|T_t\|} \right\| + \lambda_1 |P_t - T_t| \right) + w_r (\|P_r - T_r\| + \lambda_2 |P_r - T_r|) \quad (3.1)$$

In machine learning, the loss function tells the model how far off the predictions are from the target values. In this case, where the model predicts 6D poses, including translations (3D) and rotations (3D), it makes sense to separate the loss into two parts: one for translations (T) and one for rotations (R). This allows the loss function to handle two parts independently and possibly assign different importance to them via the weights w_t and w_r .

For translations, it is essential to penalize differences not only in magnitude but also in direction. To accomplish this, the Euclidean distance (L2 norm) is computed between the normalized predicted and target translation vectors, effectively measuring the angular difference. Furthermore, the magnitude of the error, an absolute difference (L1 norm) scaled by a factor λ_1 , is also added, giving the total translation loss.

Similarly, for rotations, the aim is to penalize disparities between the predicted and target rotations. The first term of the rotational loss function is the L2 norm (Euclidean distance) between the predicted rotation and the target rotation, reflecting the disparity in the degree of rotation. For rotations there is no need to normalize the L2 norm since direction does not matter. This is followed by the absolute difference (L1 norm) scaled by a factor λ_2 to account for the error in the magnitude of the rotation.

Finally, the two losses for translation and rotation are combined into a total loss, considering the weights w_t and w_r for translations and rotations, respectively. This gives a meaningful single scalar value that can be used for optimization.

After testing, it was determined that superior results were achieved for certain ranges of the parameters w_t , w_r , λ_1 , and λ_2 . Specifically, w_t performed optimally within the range $[0.5, 1.5]$, w_r within the range $[1, 2]$, λ_1 within $[0.05, 0.5]$, and λ_2 within the range $[0.01, 0.1]$. These ranges provided a balance that minimized the loss and increased the model's predictive ability.

3.5 Optimizer

Optimization algorithms play an essential role in training machine learning models. These algorithms adjust the model's parameters iteratively to minimize the loss function's error, effectively learning from the data to improve the model's performance. Among the various optimizer algorithms tested, the AdamW optimizer was ultimately chosen for this dissertation due to how easy it is to use. It does not require much tuning to get good performance, a behavior that is appreciated in more complex problems like Learning-based VO.

AdamW is an extension of the Adam optimizer, a popular choice due to its adaptive learning rate, which can lead to faster convergence and improved generalization. The distinction of AdamW lies in its superior handling of weight decay, which aids in regularizing the model and preventing overfitting. Furthermore, AdamW also works better with learning rate schedulers because it decouples weight decay from the optimization steps. This means that as the learning rate decreases with a schedule, the weight decay effect does not diminish. Besides selecting an optimizer, managing the learning rate during training is crucial. This concept, known as learning rate scheduling, adjusts the learning rate over the training period, which can improve the model's final performance. The results to be presented employ two particular learning rate schedulers: ReduceLROnPlateau and CosineAnnealingWarmRestarts. The ReduceLROnPlateau scheduler reduces the learning rate whenever the model's performance plateaus; that is when it stops improving. This scheduler allows the optimizer to make finer adjustments to the parameters, avoiding the potential pitfall of overshooting the optimum. Consequently, it can contribute to faster convergence and prevent stagnation of the model's learning process. The CosineAnnealingWarmRestarts scheduler varies the learning rate according to a cosine function with periodic warm restarts. This strategy initiates with larger learning rates to encourage broad exploration of the parameter space, then progressively decreases the rate for precise fine-tuning of the model parameters. Periodic warm restarts reset the learning rate to its initial value, fostering exploration in potentially different regions of the parameter space.

Together, the AdamW optimizer and the use of these learning rate schedulers support an effective training process, balancing the need for broad exploration of the parameter space and fine-tuning, resulting in a well-optimized model.

Chapter 4

Experiments

This chapter evaluates the performance of the three proposed architectures, namely PoseFormer2D, PoseFormer3D, and Spatial-Temporal PoseFormer, under a consistent set of conditions. It also explores the impacts of different configurations within these models, including the effects of fixed positional encodings versus learnable encodings on the 2D and 3D PoseFormers and the addition or exclusion of optical flow information.

4.1 Training and code details

The KITTI dataset [32] provides eleven training sequences. Out of those eleven sequences, the models were trained on the sequences 00, 01, 02, 05, 08, and 09, as these contain over 70% of the images present on the dataset, thus representing it better. Sequence 07 was used as validation, and sequences 03, 04, 06, and 10 were excluded from training. All the images were resized to have a height equal to 256, and a width equal to 512. The patch size chosen, $P = 16$, is a divisor of both the height and width proposed.

Each time a model makes a prediction, it receives two consecutive frames and the respective optical flow (if so desired). The individual inputs are shuffled to create diverse batches of 16. This shuffling is performed on the level of the entire inputs, not within the paired images or optical flow themselves. This approach ensures that the sequences within the same batch are not directly sequential, providing the model with a broader range of data in each batch for better generalization. The batch size greatly impacts the memory used to train the model. Hence, it has to be chosen according to the GPU used to train the models. All the training, validation, and testing are done using the PyTorch and the PyTorch Lightning framework, which allows for the creation of modular training, validation, and testing loops and easy setup of an optimizer scheduler.

The computer used to train the models has a Ryzen 5800x3D CPU (8 cores, 16 threads) with up to 4.5GHz clock speed; the GPU used was an RTX4090 with 24GB of VRAM.

4.2 Results

This section will demonstrate each transformer model’s qualitative and quantitative performance. Additionally, it will provide an overall comparison against other state-of-the-art models.

To set a standard point of comparison between the different variations of the proposed models, they were trained for 30 Epochs with the same hyperparameters. The starting learning, $5e^{-5}$, stayed consistent for all the training tests.

4.2.1 Training

During training, as shown in Figure 4.1, the model computes the loss for every step. These individual losses are accumulated throughout an epoch, representing the total error the model experiences during that time. Once the epoch ends, an average loss is determined by dividing the total accumulated loss by the number of steps. However, this gives the average loss per batch, with each batch comprising 16 inputs. To get the loss per single input within a batch, this average is further divided by the batch size. For the validation phase, demonstrated in Figure 4.2, the calculation process is similar to training. However, the batch size during validation is set to 1. This means that the average loss computed at the end of an epoch is already indicative of the loss per single input, and no further division by batch size is necessary. These plots show the result of training with AdamW optimizer with the Reduce on Plateau scheduler. Both training and validation loss are important metrics to understand how the models are learning over time, but validation loss is a better metric to represent how the model would perform in real-world scenarios. Thus the lower the validation loss, the better.

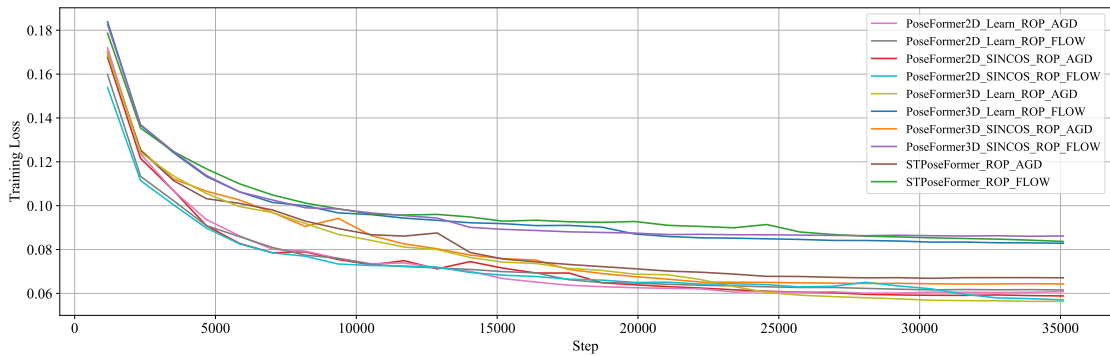


Figure 4.1: Training loss of PF2D, PF3D, and STPF models when trained with and without flow information. The 2D and 3D models were tested with and without learnable encodings. All the models were trained for 30 epochs with the Reduce On Plateau (ROP) optimizer scheduler.

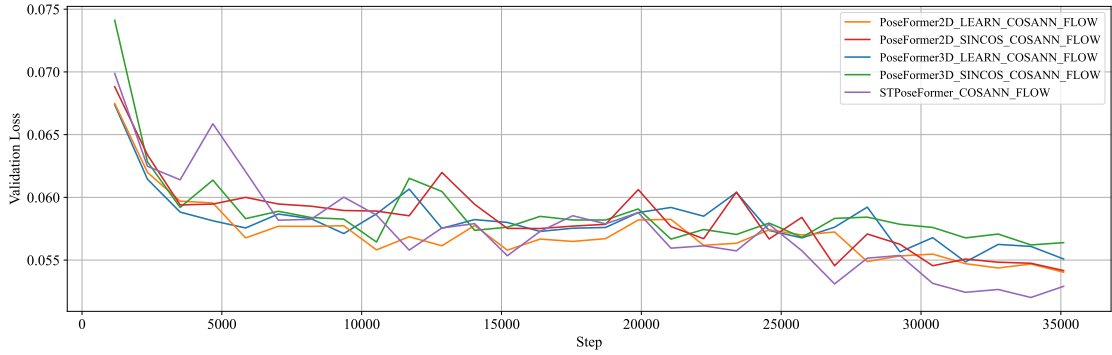


Figure 4.2: Validation loss of PF2D, PF3D, and STPF with ROP scheduler.

The graphs show that the worst-performing models, during validation, are the ones that do not have optical flow information. This is especially pronounced for the more complex models PF3D and STPF. Being more complex means a longer convergence time, and within a 30-epoch timeframe, they end up not reaching a satisfactory solution when using only images. It is also worth noting that in the graphs, the training loss can sometimes be lower than the validation loss for the same model. While this is not the typical behavior, it can happen in some of these models for two main reasons.

- The MLP heads, on the last layer of the transformer, have a high dropout value. High dropout prevents overfitting by making some values in the input tensors zero. This stimulates other neurons learning and prevents overfitting. More importantly, this zeroing of values does not happen during validation, which means that the model can use all the learned information when predicting on the validation dataset.
- As the validation data comes from a single sequence, that sequence can be overall simpler than some others present in the training, which can result in a lower validation loss.

All the variations were also trained with the cosine annealing with warm restarts optimizer scheduler. In this case, the optical flow datasets were the only ones used. This is due to their faster convergence, thus being easier to evaluate the models' performance. The scheduler has a cyclic learning rate that gradually decreases within each cycle (annealing phase) and then restarts at the beginning of each new cycle. Essentially, it warms up the learning rate at the start of each cycle and cools it down towards the end. Having this dynamic adjustment, allows the model to potentially escape local minima and converge toward a more optimal solution during training. For these tests, each annealing phase has a duration of $T_0 \times T_m^{n-1}$, where $T_0 = 500$, $T_m = 2$, and n is the n th annealing phase. Furthermore, the lowest learning rate that the optimizer reaches each annealing phase is $5e^{-7}$. For this experiment, the training and validation loss can be observed in the graphs 4.3 and 4.4.

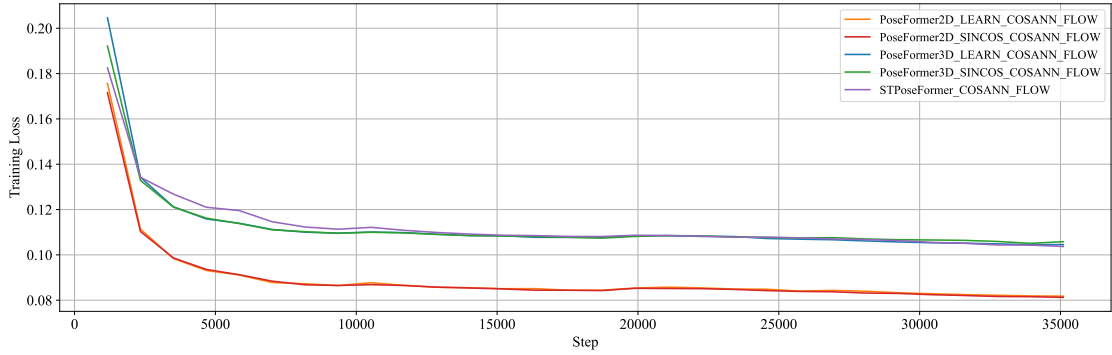


Figure 4.3: Training loss of PF2D, PF3D, and STPF models when trained with the cosine annealing with warm restarts optimizer scheduler. The 2D and 3D models were also tested with and without learnable encodings.

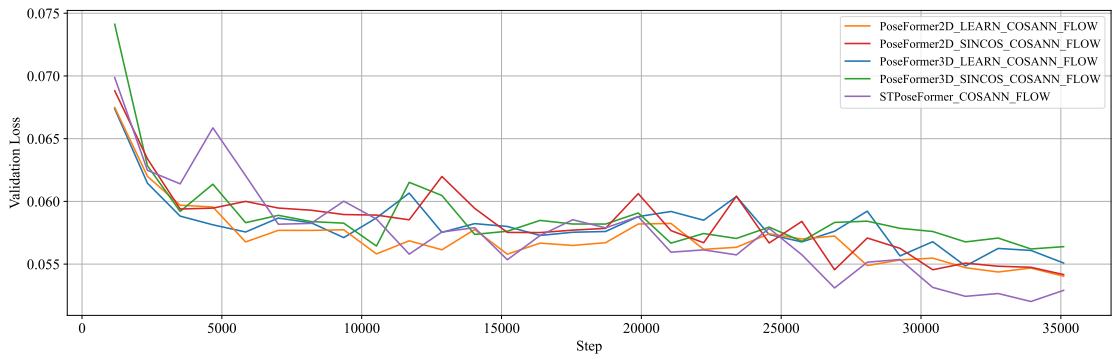


Figure 4.4: Validation loss of the transformers when using the cosine annealing with warm restarts optimizer scheduler.

The training results from the task of 6D pose estimation for visual odometry indicate that the Reduce On Plateau (ROP) scheduler outperforms the Cosine Annealing with Warm Restarts approach. Moreover, it is also interesting that the models using fixed positional encodings exhibited equivalent performance to those employing learnable positional encodings. This observation suggests that in the context of 6D pose estimation for visual odometry tasks, the choice between fixed and learnable positional encodings does not significantly affect the models' effectiveness, or that not enough training data was used to see any real difference. Despite the inherent adaptability of learnable encodings, and the theoretically advantageous property of fixed encodings to capture long-term dependencies, neither approach showed a distinctive advantage over the other. However, it is crucial to emphasize that this conclusion might be specific to this particular task and dataset, and the choice between fixed and learnable positional encodings could produce different outcomes in other scenarios. Analyzing the graphical data, all the transformer models evidently exhibit a learning tendency from the input data. Specifically focusing on the models trained with ROP, as they resulted in comparatively lower loss values, a majority of the PoseFormer2D variations display minimal validation loss in contrast to the remaining models. PoseFormer3D

generally shows the weakest performance, whereas STPoseFormer exhibits mixed results - being superior in some tests and inferior in others. This analysis suggests that PoseFormer2D, due to its less complex structure, converges more quickly than its counterparts. Conversely, PoseFormer3D, despite its more intricate information representation, takes longer to converge, and the benefits derived from this complexity appear negligible. The STPoseFormer model, despite its longer convergence time, appears to encapsulate the data patterns best. This suggests that with further optimization in training, such as providing more data to enhance model generalization, the performance of STPoseFormer could potentially improve significantly. Overall it seems that the low amount of training data seemed to favor the less complex PoseFormer2D. PoseFormer3D changes in architecture do not seem to improve the capabilities of the model, and the STPoseFormer seems to have some advantages over the other two architectures, but that did not make it stand out in any way.

4.2.2 Ego Motion Trajectories

The previous section compared the training and validation loss achieved by the variants of the proposed models. Analyzing the performance of the models based on how good the predictions are is a good starting point. However, KITTI has a set of standard evaluation metrics that better capture how good a technique is at doing VO. These metrics are:

- t_{err} : average translational error, which is measured in percentage (%) and calculated over 100 meters;
- r_{err} : average rotational error, measured in degrees per 100 meters ($^{\circ}/100m$);
- ATE: absolute trajectory error, expressed in meters. It corresponds to the root mean squared error (RMSE) between the estimated camera pose and the ground truth;
- RPE: relative pose error for rotations and translations. It represents the error between frames in meters for translations and in degrees for rotations.

Table 4.1, shows the quantitative performance of the trained models when compared with stereo ORB-SLAM3 (without loop closure) and DeepVO (the most well known end-to-end approach, similar to the transformers). From this point forward, only the best-performing version of each model is used. This means that for the PoseFormer2D and PoseFormer3D, no learnable encodings were used. All three models use optical flow information. Being the best means that the values present in the table are the lowest possible for each model.

The results are obtained using a 6 degrees of freedom (6DoF) alignment. The usage of 6DoF alignment is common in the field of visual odometry as it provides standard and consistent means to measure and compare the performance of different techniques by removing much of the drift and other errors which accumulate over time. The trained DeepVO model used for comparison is from an unofficial implementation¹, and was trained on the same sequences as the transformer

¹<https://github.com/ChiWeiHsiao/DeepVO-pytorch>

models (00, 01, 02, 05, 08, and 09); and the code used for ORB-SLAM3 comes from its official C++ implementation².

Method		Sequence										
		00	01	02	03	04	05	06	07	08	09	10
t_{err} (%)	ORB-SLAM3	0.852	1.748	0.763	0.978	0.442	0.643	0.843	0.844	1.022	0.849	0.659
	DeepVO	62.121	101.589	82.313	96.307	20.517	56.219	56.842	69.850	78.979	74.336	126.950
	PoseFormer2D	<u>8.688</u>	3.682	<u>6.568</u>	13.895	<u>13.927</u>	7.819	<u>13.445</u>	14.187	<u>8.854</u>	8.841	<u>9.574</u>
	PoseFormer3D	10.529	5.140	8.151	13.005	14.682	8.946	24.559	<u>13.101</u>	14.015	<u>8.039</u>	11.623
	STPoseFormer	10.424	<u>3.609</u>	7.722	<u>10.020</u>	15.615	<u>6.118</u>	20.769	13.222	10.747	8.466	12.724
r_{err} (%)	ORB-SLAM3	0.299	0.272	0.262	0.167	0.129	0.259	0.253	0.481	0.297	0.250	0.335
	DeepVO	34.051	12.939	31.198	24.626	7.152	35.269	33.577	59.872	33.381	31.960	26.607
	PoseFormer2D	<u>3.188</u>	1.190	<u>2.368</u>	<u>3.959</u>	<u>1.573</u>	3.804	3.986	10.945	<u>3.307</u>	3.340	<u>3.514</u>
	PoseFormer3D	3.888	1.226	2.835	5.468	2.375	3.912	8.966	8.136	5.465	2.854	4.262
	STPoseFormer	4.154	<u>1.118</u>	2.610	4.030	2.333	<u>1.917</u>	7.399	<u>6.229</u>	3.628	<u>3.005</u>	4.196
ATE (m)	ORB-SLAM3	5.163	<u>14.563</u>	7.434	1.386	0.217	2.148	2.165	1.401	3.418	3.245	1.360
	DeepVO	105.444	71.061	168.569	39.100	<u>12.424</u>	66.294	93.828	21.185	64.311	29.212	49.480
	PoseFormer2D	81.465	22.217	56.647	18.548	15.491	39.764	<u>58.018</u>	40.466	<u>62.398</u>	68.644	<u>15.836</u>
	PoseFormer3D	80.819	27.572	91.313	<u>11.355</u>	16.494	47.579	109.952	31.374	118.107	<u>29.759</u>	24.634
	STPoseFormer	<u>73.584</u>	13.356	<u>56.574</u>	12.800	17.572	<u>35.829</u>	89.842	<u>21.035</u>	64.908	66.103	24.624
RPE (m)	ORB-SLAM3	0.020	0.058	0.023	0.016	0.018	0.014	0.015	0.013	0.025	0.020	0.016
	DeepVO	0.687	2.340	1.281	0.742	0.355	0.698	0.907	0.593	0.839	0.842	1.120
	PoseFormer2D	<u>0.058</u>	<u>0.064</u>	<u>0.059</u>	0.107	<u>0.211</u>	<u>0.063</u>	0.169	0.073	<u>0.069</u>	<u>0.070</u>	<u>0.088</u>
	PoseFormer3D	0.068	0.099	0.074	<u>0.091</u>	0.223	0.067	<u>0.146</u>	<u>0.071</u>	0.078	0.086	0.091
	STPoseFormer	0.059	0.066	0.068	0.118	0.238	0.068	0.184	0.093	0.078	0.083	0.124
RPE (°)	ORB-SLAM3	0.059	0.037	0.050	0.042	0.032	0.041	0.033	0.038	0.045	0.046	0.053
	DeepVO	1.038	0.623	0.921	0.567	0.171	0.716	0.609	0.940	0.871	0.969	0.746
	PoseFormer2D	<u>0.181</u>	<u>0.103</u>	<u>0.158</u>	<u>0.153</u>	<u>0.103</u>	<u>0.136</u>	<u>0.124</u>	<u>0.187</u>	<u>0.160</u>	<u>0.152</u>	<u>0.177</u>
	PoseFormer3D	0.258	0.123	0.221	0.209	0.124	0.181	0.165	0.213	0.210	0.194	0.241
	STPoseFormer	0.263	0.118	0.224	0.216	0.128	0.189	0.162	0.235	0.217	0.198	0.249

Table 4.1: Quantitive comparison between the three developed architectures, ORB-SLAM3 without loop closure and DeepVO. Results are scaled using 6DoF. The PF2D uses flow information without learnable encodings and ROP optimizer scheduler. PF3D uses flow information without learnable encodings and ROP optimizer scheduler. STPoseFormer uses flow information and the ROP optimizer scheduler.

Examining the information in the table, ORB-SLAM3 consistently outperforms across all sequences and evaluation metrics, implying its superior capability in predicting camera pose relative to the ground truth. Being one of the state-of-the-art geometric approaches, ORB-SLAM3 will be utilized as a benchmark to assess the extent of deviation in the predictions of the proposed models.

Among the machine learning-based models, including DeepVO, PoseFormer2D, PoseFormer3D, and STPoseFormer, DeepVO tends to record the greatest errors across all sequences and metrics, suggesting that it is overshadowed by the proposed PoseFormer architectures. PoseFormer2D performs particularly well, displaying the best results among the machine learning models in terms of the t_{err} , while STPoseFormer is the best for the r_{err} . STPoseFormer, despite demonstrating slightly higher short-term errors in t_{err} , is also a very capable model, as evidenced by its relatively low ATE for sequences 00 and 01. PoseFormer3D, while not surpassing other models in any specific metric, still maintains steady and comparable results with the other PoseFormer variants. Overall the PF2D presents an increase in performance, when compared to DeepVO, of 87% and 87% for t_{err} and r_{err} respectively; PF3D an increase of 84% and 85%; and STPF an increase of 85% and 88%.

²https://github.com/UZ-SLAMLab/ORB_SLAM3

Following the performance analysis of the chosen methods against the selected KITTI metrics, visualizing their predicted trajectories is crucial. The subsequent figures, 4.5 and 4.6, illustrate the performance of the three transformer models, under 6DoF alignment on the training sequences.

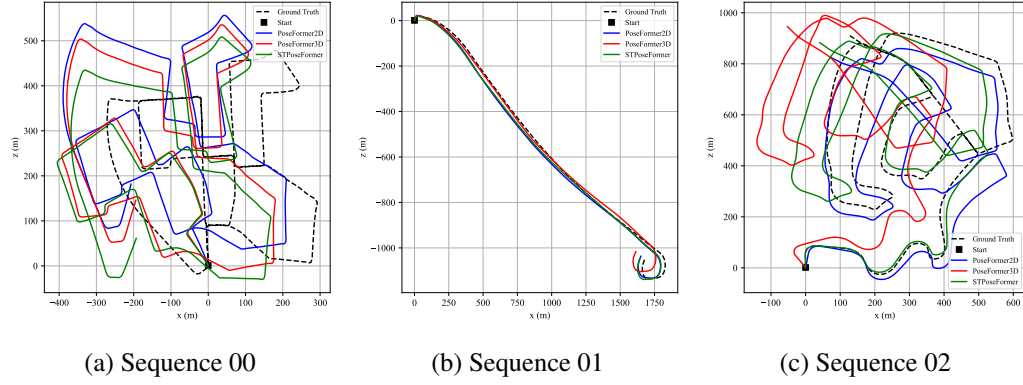


Figure 4.5: Plotted trajectory for the training sequences 00 (Urban), 01 (Highway), and 02 (Urban).

The initial three sequences demonstrate that all models possess some degree of accuracy in predicting trajectories. Both PoseFormer2D and STPoseFormer exhibited similar performance, while the PoseFormer3D fell short in comparison. Sequence 01, illustrated in figure 4.5b, represents a highway scenario that typically poses challenges for ego-motion estimation techniques due to an increased number of traceable feature points exiting the field of view between frames. Nevertheless, all three models managed to accurately predict the trajectory. On the other hand, sequences 00 and 02, though relatively simple, are very long and start to accumulate drift, as depicted in figures 4.5a and 4.5c.

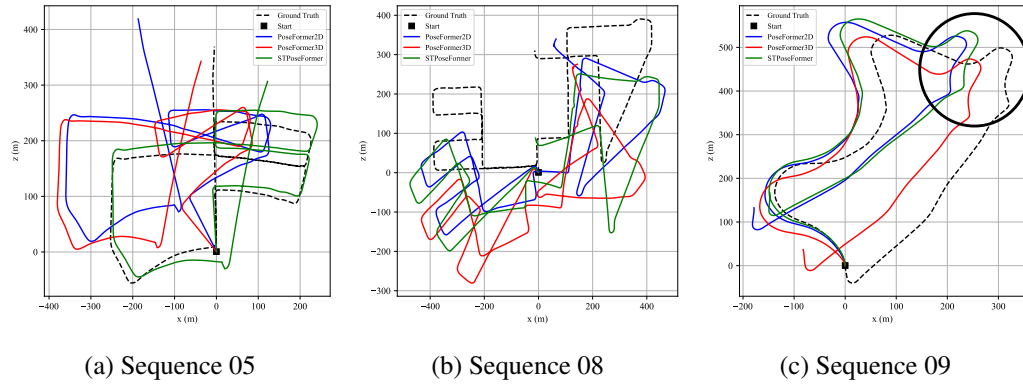


Figure 4.6: Plotted trajectory for the training sequences 05 (Urban), 08 (Urban), and 09 (Urban).

The second set of training sequences, despite their simplicity, poses a challenge due to their length and multiple loops, which tends to accumulate drift. In this context, STPoseFormer exhibits superior performance, specifically on sequence 05, while delivering comparable results to the other two models on sequences, 08 and 09. Sequence 09 (figure 4.6c) illustrates a significant error in the

turns marked by the black circle. This error is the result of interference from an external vehicle, which greatly affects the vehicle's own ego-motion estimates.

The subsequent figures, namely figure 4.7 and 4.8, provide a detailed analysis of the models' performance across the testing sequences.

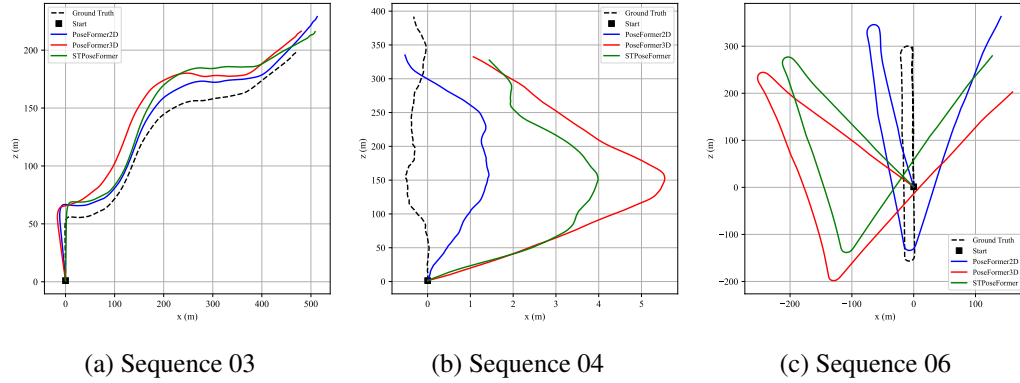


Figure 4.7: Plotted trajectory for the testing sequences 03 (Urban), 04 (Urban), and 06 (Urban).

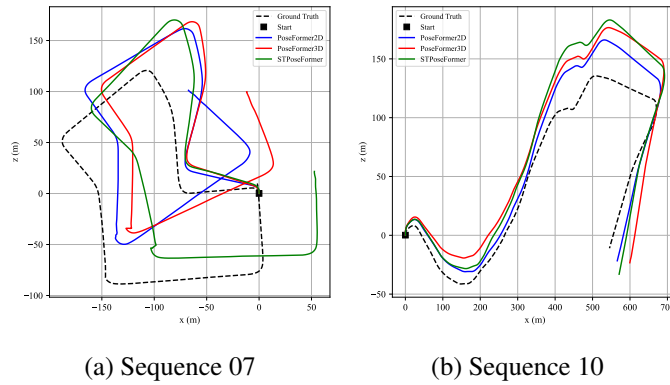


Figure 4.8: Plotted trajectory for the testing sequences 07 (Urban) and 10 (Urban).

From the plots, it can be inferred that the models have effectively acquired the ability to estimate ego-motion, as demonstrated by their capacity to predict the trajectory for all test sequences, albeit with varying degrees of precision. In the context of sequences 03 (figure 4.7a) and 10 (figure 4.8b), the models were successful in approximating the overall trajectory shape. However, they exhibit certain discrepancies in terms of translation magnitude, while largely maintaining the rotation component. Conversely, sequence 06 (figure 4.7c) manifests a considerable error during both U-turns. This can be attributed to the absence of similar turns in the training sequences, hence depriving the models of the necessary understanding to accurately predict the pose alteration.

To further explore how well the models are doing throughout the sequences, the plots in the figures 4.9, 4.10, and 4.11 show the translation error in percentage over the first eight hundred meters.

The translation error tends to increase as the path length increases. This is an expected behavior since the predictions accumulate small errors over time, resulting in bigger errors over

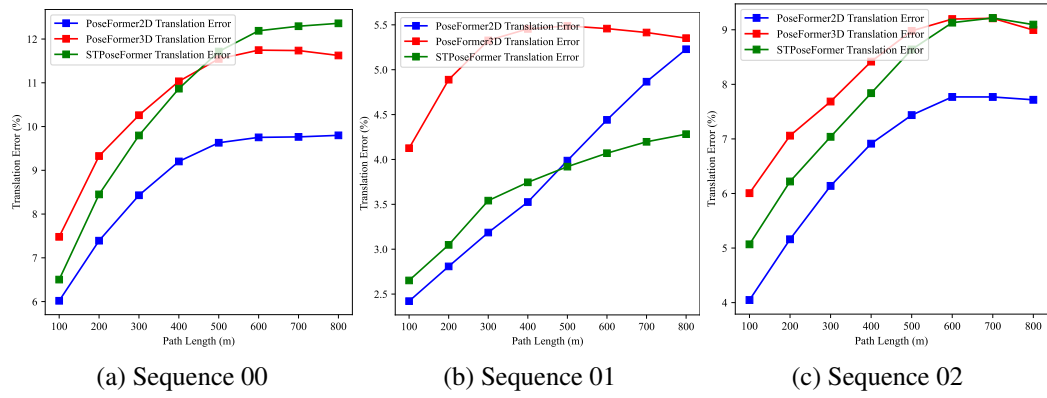


Figure 4.9: Translation error for sequences 00, 01, and 02.

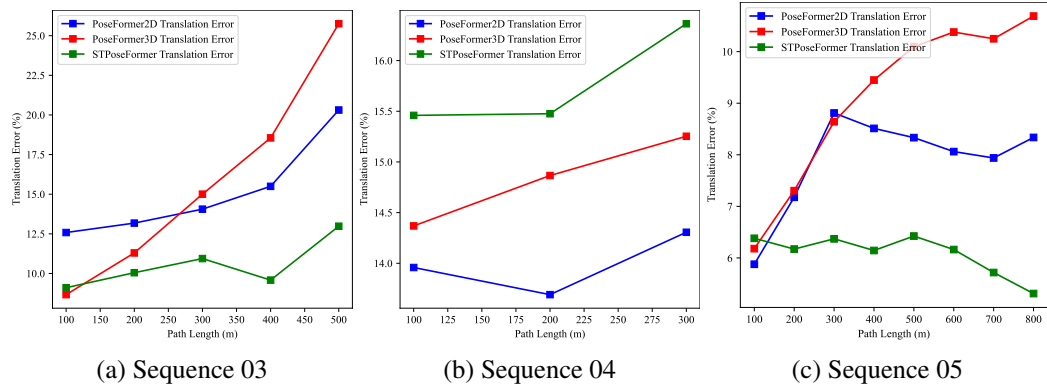


Figure 4.10: Translation error for sequences 03, 04, and 05.

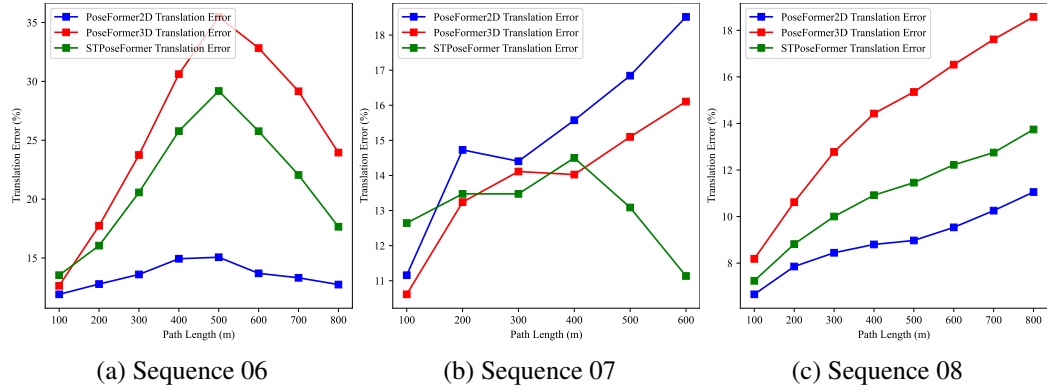


Figure 4.11: Translation error for sequences 06, 07, and 08.

longer distances. Sequence 06 is an exception since, over the 500-meter mark, the accumulation of translation errors leads to an overall better predicted global position, figure 4.9. This does not occur as a result of the models' accurate predictions. Instead, due to the accumulated drift, the predicted global position happens to align more closely with the actual global position. It is also possible to see that PoseFormer3D is the worst-performing model of the three. This goes along with the conclusions taken from the table, 4.1.

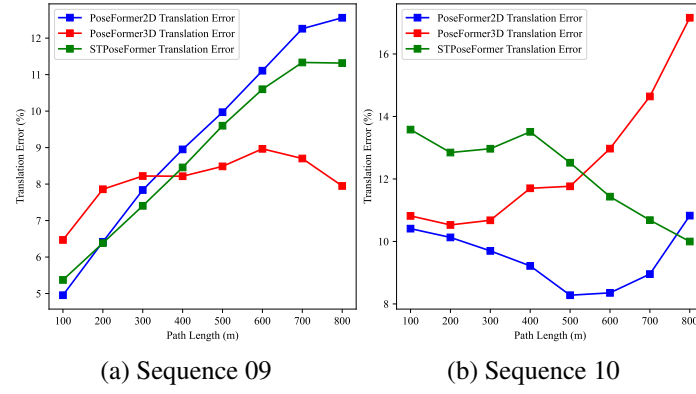


Figure 4.12: Translation error for sequences 09 and 10.

Evaluating the models' rotation errors over time is also important, and they are depicted in the following plots, figures

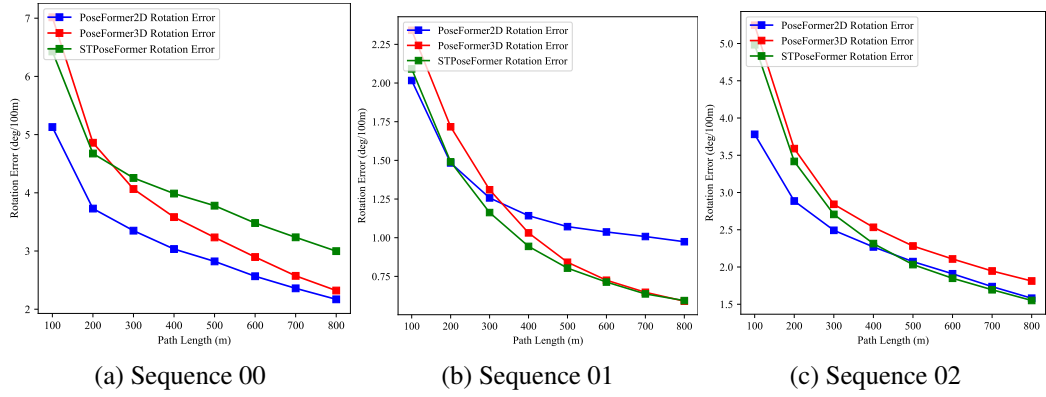


Figure 4.13: Rotation error in degrees/100m for sequences 00, 01, and 02.

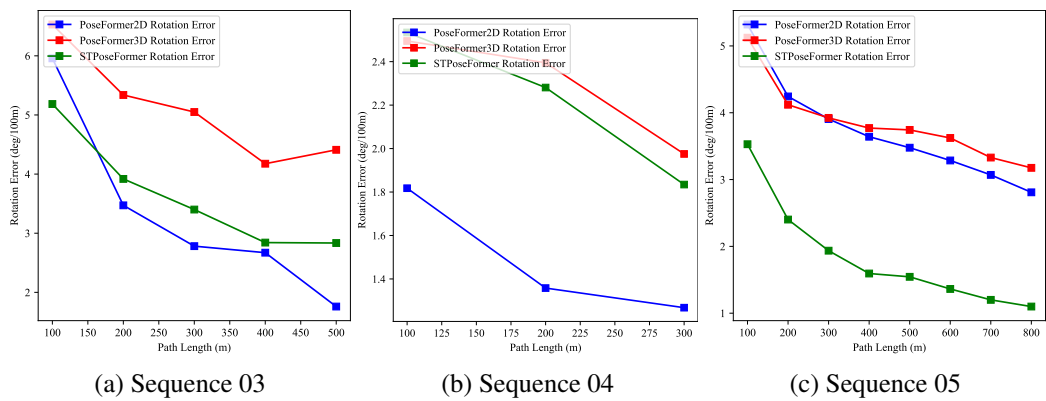


Figure 4.14: Rotation error in degrees/100m for sequences 03, 04, and 05.

The plots for the rotations show an interesting behavior, where over time, the error seems to decrease. This most likely is the result of the straights that the trajectory has. In other words, as time goes on, more and more straight segments start contributing to a lower overall rotation

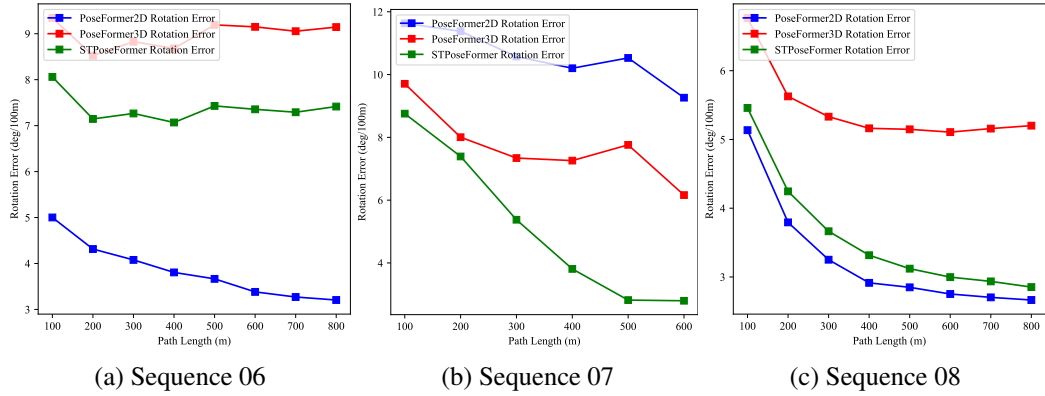


Figure 4.15: Rotation error in degrees/100m for sequences 06, 07, and 08.

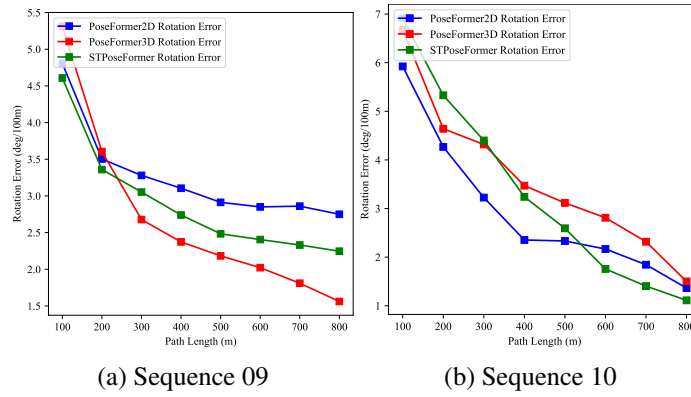


Figure 4.16: Rotation error in degrees/100m for sequences 09 and 10.

error. The results here are mostly similar to the translation plots, where STPoseFormer and PoseFormer2D are the better choices. PoseFormer3D shows good results but not as good as the other two.

To further cement the obtained results, it is important to see how the predicted trajectories measure up against DeepVO and ORB-SLAM3. The following plots, figures 4.17, 4.18, 4.19, demonstrate that, a qualitative comparison between the STPoseFormer, DeepVO, and ORB-SLAM3. The STPoseFormer, DeepVO, and ORB-SLAM3 are all under a 6DoF alignment.

The plots show that the STPoseFormer model consistently yields better trajectories than DeepVO. Moreover, the performance of the transformers is expected to significantly improve as more and more training data is available. Another thing worth noting is that ORB-SLAM3 is considerably better than STPoseFormer on the simpler sequences, but on sequence 01, arguably, the hardest sequence due to the nature of highways, the model performs better than ORB-SLAM3. This can be a very good indication that in harder sequences, where traditional mathematical approaches start failing, the STPoseFormer will be able to maintain a somewhat consistent performance.

To further cement the taken conclusion, the translation and rotation errors over time for the STPoseFormer, DeepVO, and ORB-SLAM3 have to be taken into account. They are present in the following figures, 4.20 and 4.21.

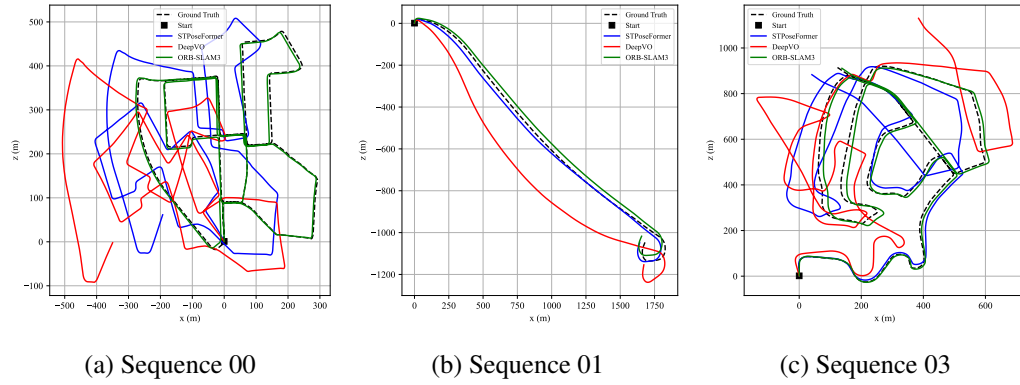


Figure 4.17: The STPoseFormer is better than DeepVO in all scenarios and manages to outperform the sota ORB-SLAM3 slightly in sequence 01. On sequence 02, the STPoseFormer suffers from drift problems, which are less pronounced on the ORB-SLAM3.

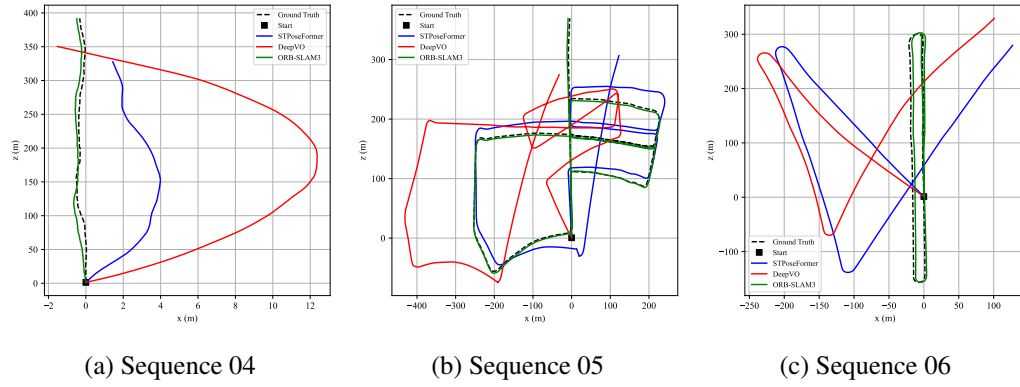


Figure 4.18: The STPoseFormer significantly outperforms the DeepVO on sequences 04 and 05. Sequence 06 shows a somewhat similar performance between the STPoseFormer and DeepVO.

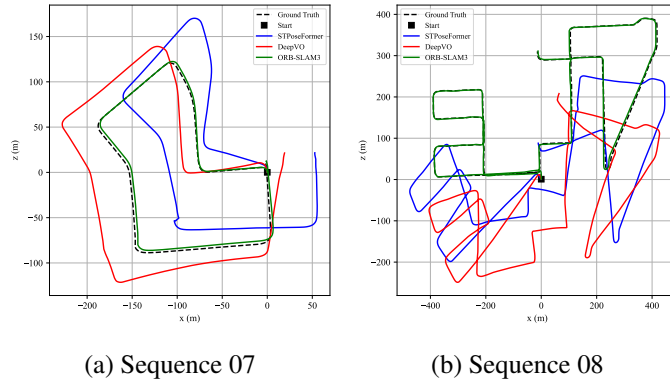


Figure 4.19: Sequences 07 and 08. The STPoseFormer does not do very well on these sequences due to the accumulated errors. ORB-SLAM3 stands as the indisputable best method.

The diagrams illustrate the superior performance of the suggested transformer model compared to DeepVO. Both the translation and rotation errors of the transformer model are significantly reduced compared to those of DeepVO. Meanwhile, ORB-SLAM3 consistently exhibits

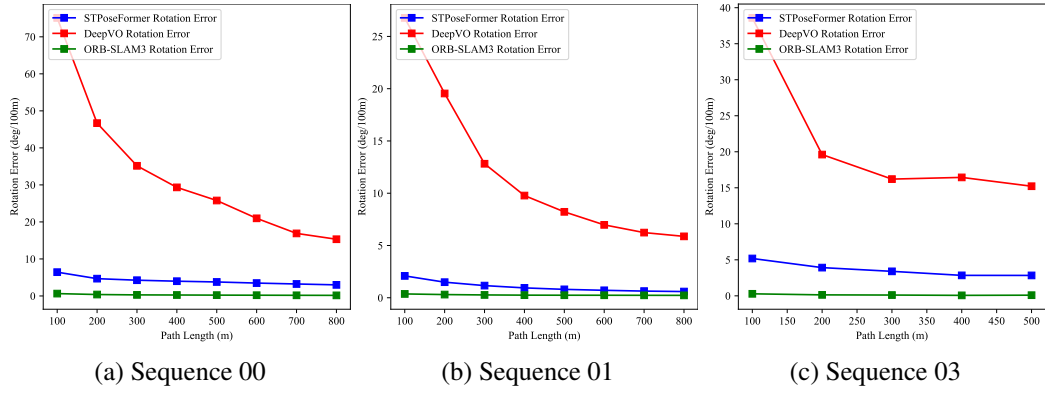


Figure 4.20: The STPoseFormer is better than DeepVO on all three sequences.

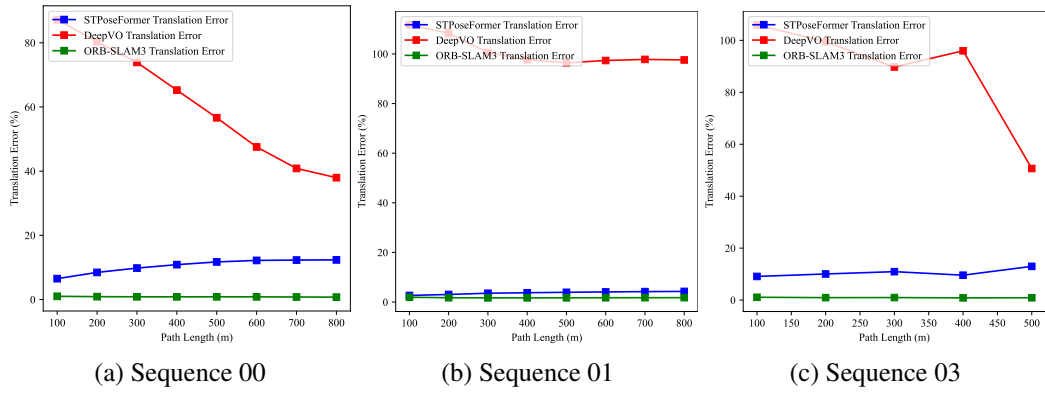


Figure 4.21: The STPoseFormer significantly outperforms the DeepVO.

almost zero error throughout the sequences presented.

The following figure, 4.22, shows a final comparison of the models' trajectories, without any alignment.

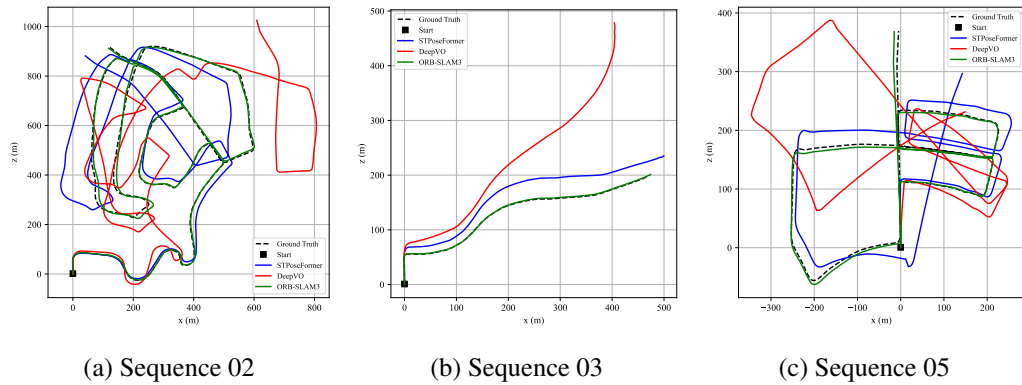


Figure 4.22: The plotted unaligned sequences.

The suggested STPoseFormer approach once again outperforms the DeepVO, exhibiting an enhanced ability to determine the scale of the scene. Like the transformer, ORB-SLAM3 also demonstrates this capability, given its use of stereo images.

In summary, the proposed transformer models demonstrate significant superiority over DeepVO - the other learning-based method. Moreover, the results from these transformer models align more closely with the state-of-the-art ORB-SLAM3 than they do with DeepVO.

4.2.3 Inference Time

Regardless of the accuracy of the Transformers, their utility would be negligible if they were unable to produce real-time predictions. As such, the computational efficiency, particularly the inference time, of these models is of considerable importance. They need to swiftly process and analyze incoming data to ensure their effective response to changing driving scenarios. Using KITTI as a benchmark, where images are captured at a rate of 10Hz or every tenth of a second, the inference time for each model should ideally not exceed this limit. The inference time for the proposed models is outlined in the following table 4.2:

Model	avg (ms)	std (ms)
PoseFormer2D	7.65	0.45
PoseFormer3D	8.66	0.41
STPoseFormer	11.65	0.31

Table 4.2: Average and standard deviation time for the proposed models

As the table shows, the proposed models running on the training hardware are more than capable of achieving real-time performance. It still is worth noting that the hardware used is considered to be top-of-the-line, at the time of writing this thesis. However, the size and cost of the hardware are not significant and can be easily used in modern vehicles. Furthermore, the used programming language - Python - is not a compiled language like C++. Thus, it is expected that implementations in C++ or other compiled language will have lower inference times.

Chapter 5

Conclusions and future work

The work done in this dissertation has demonstrated that transformer-based architectures possess a significant potential for predicting 6D poses. The alterations in attention mechanisms and how image segmentation into patches is done led to the successful development and training of three different transformer models: PoseFormer2D, PoseFormer3D, and Spatio-Temporal PoseFormer. These models exhibited good performance in the KITTI dataset, with results that are around 85% better than DeepVO. Moreover, the architectures demonstrated inherent flexibility, permitting adept handling of mixed spatial and temporal information in the channels dimension. This feature of the transformers is further supported by their capacity to effectively utilize extra data, as adding flow information helped the models converge faster and also yield better results.

The models proposed in this research offer robust solutions, but the performance is still not comparable to the more explored geometric methods. Still, they hold potential for further enhancements. Increasing the size of the training datasets would be a good starting point, as transformers benefit greatly from bigger datasets. Another key factor that can enhance the performance of these models is the choice of attention mechanism. For instance, the STPoseFormer incorporates modifications that facilitate separate attention to spatial and temporal elements. Despite this its performance, like the PF2D and PF3D models, relies on dot product attention, which, although a powerful tool, can potentially be improved by using other attention mechanisms.

In conclusion, the transformer architectures are capable of creating consistent, precise and accurate 6D pose estimates, far surpassing the DeepVO. Moreover, most of the shortcomings the transformer models are facing can potentially be solved by working on:

- Enhancing the dataset by increasing its size and refining its quality. A more extensive dataset would enhance the model's ability to manage a broader array of situations.
- Exploring alternate attention mechanisms. While dot-product attention provides a strong foundation, other attention strategies might enable the model to concentrate on diverse segments of the images more effectively.
- Architecting a model that utilizes optical flow and images more optimally. At present, the transformer merges the images and optical flow in the channels dimension, producing an

attention score. There may, however, be more efficient methods to accomplish this, such as computing distinct attention scores for optical flow and images.

Bibliography

- [1] Hans P. Moravec. “Obstacle avoidance and navigation in the real world by a seeing robot rover”. In: (Jan. 1980). DOI: [10.1184/R1/6557033.v1](https://doi.org/10.1184/R1/6557033.v1). URL: https://kilthub.cmu.edu/articles/journal_contribution/Obstacle_avoidance_and_navigation_in_the_real_world_by_a_seeing_robot_rover/6557033.
- [2] Martin A. Fischler and Robert C. Bolles. “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. In: *Commun. ACM* 24.6 (June 1981), pp. 381–395. ISSN: 0001-0782. DOI: [10.1145/358669.358692](https://doi.org/10.1145/358669.358692). URL: <https://doi.org/10.1145/358669.358692>.
- [3] C.G. Harris and J.M. Pike. “3D positional integration from image sequences”. In: *Image and Vision Computing* 6.2 (1988). 3rd Alvey Vision Meeting, pp. 87–90. ISSN: 0262-8856. DOI: [https://doi.org/10.1016/0262-8856\(88\)90003-0](https://doi.org/10.1016/0262-8856(88)90003-0). URL: <https://www.sciencedirect.com/science/article/pii/0262885688900030>.
- [4] Robert Woodham. “Photometric Method for Determining Surface Orientation from Multiple Images”. In: *Optical Engineering* 19 (Jan. 1992). DOI: [10.1117/12.7972479](https://doi.org/10.1117/12.7972479).
- [5] Jianbo Shi and Tomasi. “Good features to track”. In: *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 1994, pp. 593–600. DOI: [10.1109/CVPR.1994.323794](https://doi.org/10.1109/CVPR.1994.323794).
- [6] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: [10.1109/5.726791](https://doi.org/10.1109/5.726791).
- [7] Philip Torr and A. Zisserman. “MLESAC: A New Robust Estimator with Application to Estimating Image Geometry”. In: *Computer Vision and Image Understanding* 78 (Aug. 2000), pp. 138–156. DOI: [10.1006/cviu.1999.0832](https://doi.org/10.1006/cviu.1999.0832).
- [8] Bill Triggs et al. “Bundle Adjustment — A Modern Synthesis”. In: *Vision Algorithms: Theory and Practice*. Ed. by Bill Triggs, Andrew Zisserman, and Richard Szeliski. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 298–372. ISBN: 978-3-540-44480-0.
- [9] L.-P. Morency and Rakesh Gupta. “Robust real-time egomotion from stereo images”. In: *Proceedings 2003 International Conference on Image Processing (Cat. No.03CH37429)*. Vol. 2. 2003, pp. II–719. DOI: [10.1109/ICIP.2003.1246781](https://doi.org/10.1109/ICIP.2003.1246781).

- [10] David Lowe. “Distinctive Image Features from Scale-Invariant Keypoints”. In: *International Journal of Computer Vision* 60 (Nov. 2004), pp. 91–. DOI: [10.1023/B:VISI.0000029664.99615.94](https://doi.org/10.1023/B:VISI.0000029664.99615.94).
- [11] K. Maine, P. Anderson, and F. Bayuk. “Communication architecture for GPS III”. In: *2004 IEEE Aerospace Conference Proceedings (IEEE Cat. No.04TH8720)*. Vol. 3. 2004, 1539 Vol.3. DOI: [10.1109/AERO.2004.1367927](https://doi.org/10.1109/AERO.2004.1367927).
- [12] D. Nister, O. Naroditsky, and J. Bergen. “Visual odometry”. In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004*. Vol. 1. 2004, pp. I–I. DOI: [10.1109/CVPR.2004.1315094](https://doi.org/10.1109/CVPR.2004.1315094).
- [13] David Nistér. “Nistér, D.: An efficient solution to the five-point relative pose problem. IEEE-TPAMI 26(6), 756-770”. In: *IEEE transactions on pattern analysis and machine intelligence* 26 (July 2004), pp. 756–77. DOI: [10.1109/TPAMI.2004.17](https://doi.org/10.1109/TPAMI.2004.17).
- [14] O. Chum and J. Matas. “Matching with PROSAC - progressive sample consensus”. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. 2005, 220–226 vol. 1. DOI: [10.1109/CVPR.2005.221](https://doi.org/10.1109/CVPR.2005.221).
- [15] T. Bailey and H. Durrant-Whyte. “Simultaneous localization and mapping (SLAM): part II”. In: *IEEE Robotics Automation Magazine* 13.3 (2006), pp. 108–117. DOI: [10.1109/MRA.2006.1678144](https://doi.org/10.1109/MRA.2006.1678144).
- [16] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. “SURF: Speeded up robust features”. In: vol. 3951. July 2006, pp. 404–417. ISBN: 978-3-540-33832-1. DOI: [10.1007/11744023_32](https://doi.org/10.1007/11744023_32).
- [17] Frédéric Labrosse. “The visual compass: Performance and limitations of an appearance-based method”. In: *Journal of Field Robotics* 23.10 (2006), pp. 913–941. DOI: <https://doi.org/10.1002/rob.20159>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.20159>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20159>.
- [18] P. Newman, D. Cole, and K. Ho. “Outdoor SLAM using visual appearance and laser ranging”. In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006*. 2006, pp. 1180–1187. DOI: [10.1109/ROBOT.2006.1641869](https://doi.org/10.1109/ROBOT.2006.1641869).
- [19] Edwin Olson, John Leonard, and Seth Teller. “Fast iterative alignment of pose graphs with poor initial estimates”. In: June 2006, pp. 2262–2269. DOI: [10.1109/ROBOT.2006.1642040](https://doi.org/10.1109/ROBOT.2006.1642040).
- [20] Edward Rosten and Tom Drummond. “Machine Learning for High-Speed Corner Detection”. In: vol. 3951. July 2006. ISBN: 978-3-540-33832-1. DOI: [10.1007/11744023_34](https://doi.org/10.1007/11744023_34).
- [21] Daniele Borio, Laura Camoriano, and Letizia Lo Presti. “Impact of GPS acquisition strategy on decision probabilities”. In: *IEEE Transactions on Aerospace and Electronic Systems* 44.3 (2008), pp. 996–1011. DOI: [10.1109/TAES.2008.4655359](https://doi.org/10.1109/TAES.2008.4655359).

- [22] Friedrich Fraundorfer et al. “Visual word based location recognition in 3d models using distance augmented weighting”. In: (Jan. 2008).
- [23] Davide Scaramuzza and Roland Siegwart. “Appearance-Guided Monocular Omnidirectional Visual Odometry for Outdoor Ground Vehicles”. In: *IEEE Transactions on Robotics* 24.5 (2008), pp. 1015–1026. DOI: [10.1109/TRO.2008.2004490](https://doi.org/10.1109/TRO.2008.2004490).
- [24] Navid Nourani-Vatani, Jonathan Roberts, and Mandiam V. Srinivasan. “Practical visual odometry for car-like vehicles”. In: *2009 IEEE International Conference on Robotics and Automation*. 2009, pp. 3551–3557. DOI: [10.1109/ROBOT.2009.5152403](https://doi.org/10.1109/ROBOT.2009.5152403).
- [25] Michael Calonder et al. “BRIEF: Binary Robust Independent Elementary Features”. In: vol. 6314. Sept. 2010, pp. 778–792. ISBN: 978-3-642-15560-4. DOI: [10.1007/978-3-642-15561-1_56](https://doi.org/10.1007/978-3-642-15561-1_56).
- [26] Vitor Guizilini and Fabio Ramos. “Visual odometry learning for unmanned aerial vehicles”. In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 6213–6220. DOI: [10.1109/ICRA.2011.5979706](https://doi.org/10.1109/ICRA.2011.5979706).
- [27] Stefan Leutenegger, Margarita Chli, and Roland Y. Siegwart. “BRISK: Binary Robust invariant scalable keypoints”. In: *2011 International Conference on Computer Vision*. 2011, pp. 2548–2555. DOI: [10.1109/ICCV.2011.6126542](https://doi.org/10.1109/ICCV.2011.6126542).
- [28] Ethan Rublee et al. “ORB: An efficient alternative to SIFT or SURF”. In: *2011 International Conference on Computer Vision*. 2011, pp. 2564–2571. DOI: [10.1109/ICCV.2011.6126544](https://doi.org/10.1109/ICCV.2011.6126544).
- [29] Davide Scaramuzza and Friedrich Fraundorfer. “Visual Odometry [Tutorial]”. In: *IEEE Robotics Automation Magazine* 18.4 (2011), pp. 80–92. DOI: [10.1109/MRA.2011.943233](https://doi.org/10.1109/MRA.2011.943233).
- [30] Yang Yu, Cédric Pradalier, and Guanghua Zong. “Appearance-based monocular visual odometry for ground vehicles”. In: *2011 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*. 2011, pp. 862–867. DOI: [10.1109/AIM.2011.6027050](https://doi.org/10.1109/AIM.2011.6027050).
- [31] Friedrich Fraundorfer and Davide Scaramuzza. “Visual Odometry : Part II: Matching, Robustness, Optimization, and Applications”. In: *IEEE Robotics Automation Magazine* 19.2 (2012), pp. 78–90. DOI: [10.1109/MRA.2012.2182810](https://doi.org/10.1109/MRA.2012.2182810).
- [32] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite”. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.
- [33] Li Bian. “Study on ionospheric delay correction in GPS signal”. In: *2013 IEEE 11th International Conference on Electronic Measurement Instruments*. Vol. 1. 2013, pp. 79–83. DOI: [10.1109/ICEMI.2013.6743045](https://doi.org/10.1109/ICEMI.2013.6743045).

- [34] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2014. DOI: 10.48550/ARXIV.1409.0473. URL: <https://arxiv.org/abs/1409.0473>.
- [35] Thomas Ciarfuglia et al. “Evaluation of non-geometric methods for visual odometry”. In: *Robotics and Autonomous Systems* 62 (Aug. 2014). DOI: 10.1016/j.robot.2014.08.001.
- [36] A. M. Pinto. “Visual motion analysis based on a robotic moving system”. PhD thesis. Porto, Portugal: Univ. Porto, Dept. Eng., 2014. URL: <https://hdl.handle.net/10216/73552>.
- [37] Andry Maykol Pinto et al. “A Flow-based Motion Perception Technique for an Autonomous Robot System”. In: *Journal of Intelligent & Robotic Systems* 75.3 (Sept. 2014), pp. 475–492. ISSN: 1573-0409. DOI: 10.1007/s10846-013-9999-z. URL: <https://doi.org/10.1007/s10846-013-9999-z>.
- [38] Andry Maykol Pinto et al. “Unsupervised flow-based motion analysis for an autonomous moving system”. In: *Image and Vision Computing* 32.6 (2014), pp. 391–404. ISSN: 0262-8856. DOI: <https://doi.org/10.1016/j.imavis.2014.04.003>. URL: <https://www.sciencedirect.com/science/article/pii/S0262885614000638>.
- [39] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].
- [40] Andry M. Pinto, António P. Moreira, and Paulo G. Costa. “A Localization Method Based on Map-Matching and Particle Swarm Optimization”. In: *Journal of Intelligent & Robotic Systems* 77.2 (Feb. 2015), pp. 313–326. ISSN: 1573-0409. DOI: 10.1007/s10846-013-0009-2. URL: <https://doi.org/10.1007/s10846-013-0009-2>.
- [41] Andry Maykol Pinto, Paulo Gomes Costa, and António Paulo Moreira. “Introduction to Visual Motion Analysis for Mobile Robots”. In: *CONTROLO’2014 – Proceedings of the 11th Portuguese Conference on Automatic Control*. Ed. by António Paulo Moreira, Aníbal Matos, and Germano Veiga. Cham: Springer International Publishing, 2015, pp. 545–554. ISBN: 978-3-319-10380-8.
- [42] Mohammad Aqel et al. “Adaptive-search template matching technique based on vehicle acceleration for monocular visual odometry system: ADAPTIVE-SEARCH TEMPLATE MATCHING FOR MONOCULAR VISUAL ODOMETRY”. In: *IEEJ Transactions on Electrical and Electronic Engineering* 11 (Sept. 2016). DOI: 10.1002/tee.22299.
- [43] Jakob Engel, Vladlen Koltun, and Daniel Cremers. *Direct Sparse Odometry*. 2016. DOI: 10.48550/ARXIV.1607.02565. URL: <https://arxiv.org/abs/1607.02565>.
- [44] Dan Hendrycks and Kevin Gimpel. *Gaussian Error Linear Units (GELUs)*. 2016. DOI: 10.48550/ARXIV.1606.08415. URL: <https://arxiv.org/abs/1606.08415>.
- [45] Alex Kendall, Matthew Grimes, and Roberto Cipolla. *PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization*. 2016. arXiv: 1505.07427 [cs.CV].

- [46] Ruihao Li et al. “UnDeepVO: Monocular Visual Odometry through Unsupervised Deep Learning”. In: *CoRR* abs/1709.06841 (2017). arXiv: 1709.06841. URL: <http://arxiv.org/abs/1709.06841>.
- [47] Raul Mur-Artal and Juan D. Tardos. “ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras”. In: *IEEE Transactions on Robotics* 33.5 (Oct. 2017), pp. 1255–1262. DOI: 10.1109/tro.2017.2705103. URL: <https://doi.org/10.1109%2Ftro.2017.2705103>.
- [48] Andry Maykol Pinto et al. “Visual motion perception for mobile robots through dense optical flow fields”. In: *Robotics and Autonomous Systems* 87 (2017), pp. 1–14. ISSN: 0921-8890. DOI: <https://doi.org/10.1016/j.robot.2016.08.014>. URL: <https://www.sciencedirect.com/science/article/pii/S0921889016305176>.
- [49] P Srinivas and Anil Kumar. “Overview of architecture for GPS-INS integration”. In: *2017 Recent Developments in Control, Automation Power Engineering (RDCAPE)*. 2017, pp. 433–438. DOI: 10.1109/RDCAPE.2017.8358310.
- [50] Ashish Vaswani et al. “Attention Is All You Need”. In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: <http://arxiv.org/abs/1706.03762>.
- [51] Saffet Vatansever and Ismail Butun. “A broad overview of GPS fundamentals: Now and future”. In: *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*. 2017, pp. 1–6. DOI: 10.1109/CCWC.2017.7868373.
- [52] Sen Wang et al. “DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2017. DOI: 10.1109/icra.2017.7989236. URL: <https://doi.org/10.1109%2Ficra.2017.7989236>.
- [53] Tinghui Zhou et al. “Unsupervised Learning of Depth and Ego-Motion from Video”. In: *CoRR* abs/1704.07813 (2017). arXiv: 1704.07813. URL: <http://arxiv.org/abs/1704.07813>.
- [54] Tinghui Zhou et al. *Unsupervised Learning of Depth and Ego-Motion from Video*. 2017. arXiv: 1704.07813 [cs.CV].
- [55] Tinghui Zhou et al. *Unsupervised Learning of Depth and Ego-Motion from Video*. 2017. DOI: 10.48550/ARXIV.1704.07813. URL: <https://arxiv.org/abs/1704.07813>.
- [56] Kai Cao et al. “Visual Odometry Based on 3D-3D and 3D-2D Motion Estimation Method”. In: *2018 Chinese Automation Congress (CAC)*. 2018, pp. 3643–3648. DOI: 10.1109/CAC.2018.8623237.
- [57] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2018. DOI: 10.48550/ARXIV.1810.04805. URL: <https://arxiv.org/abs/1810.04805>.

- [58] Ana Rita Gaspar et al. “Urban@CRAS dataset: Benchmarking of visual odometry and SLAM techniques”. In: *Robotics and Autonomous Systems* 109 (2018), pp. 59–67. ISSN: 0921-8890. DOI: <https://doi.org/10.1016/j.robot.2018.08.004>. URL: <https://www.sciencedirect.com/science/article/pii/S0921889018301386>.
- [59] Clément Godard et al. *Digging Into Self-Supervised Monocular Depth Estimation*. 2018. DOI: [10.48550/ARXIV.1806.01260](https://arxiv.org/abs/1806.01260). URL: <https://arxiv.org/abs/1806.01260>.
- [60] André Leite, Andry Pinto, and Aníbal Matos. “A Safety Monitoring Model for a Faulty Mobile Robot”. In: *Robotics* 7.3 (2018). ISSN: 2218-6581. DOI: [10.3390/robotics7030032](https://www.mdpi.com/2218-6581/7/3/32). URL: <https://www.mdpi.com/2218-6581/7/3/32>.
- [61] Ruihao Li et al. *UnDeepVO: Monocular Visual Odometry through Unsupervised Deep Learning*. 2018. arXiv: [1709.06841](https://arxiv.org/abs/1709.06841) [cs.CV].
- [62] Niki Parmar et al. *Image Transformer*. 2018. DOI: [10.48550/ARXIV.1802.05751](https://arxiv.org/abs/1802.05751). URL: <https://arxiv.org/abs/1802.05751>.
- [63] Alec Radford and Karthik Narasimhan. “Improving Language Understanding by Generative Pre-Training”. In: 2018.
- [64] Nan Yang et al. “Deep Virtual Stereo Odometry: Leveraging Deep Depth Prediction for Monocular Direct Sparse Odometry”. In: *CoRR* abs/1807.02570 (2018). arXiv: [1807.02570](https://arxiv.org/abs/1807.02570). URL: <http://arxiv.org/abs/1807.02570>.
- [65] Nan Yang et al. *Deep Virtual Stereo Odometry: Leveraging Deep Depth Prediction for Monocular Direct Sparse Odometry*. 2018. DOI: [10.48550/ARXIV.1807.02570](https://arxiv.org/abs/1807.02570). URL: <https://arxiv.org/abs/1807.02570>.
- [66] Han Hu et al. *Local Relation Networks for Image Recognition*. 2019. DOI: [10.48550/ARXIV.1904.11491](https://arxiv.org/abs/1904.11491). URL: <https://arxiv.org/abs/1904.11491>.
- [67] Sherif A. S. Mohamed et al. “A Survey on Odometry for Autonomous Navigation Systems”. In: *IEEE Access* 7 (2019), pp. 97466–97486. DOI: [10.1109/ACCESS.2019.2929133](https://doi.org/10.1109/ACCESS.2019.2929133).
- [68] Shashi Poddar, Rahul Kottath, and Vinod Karar. “Motion Estimation Made Easy: Evolution and Trends in Visual Odometry: Theories and Applications”. In: Jan. 2019, pp. 305–331. ISBN: 978-3-030-02999-9. DOI: [10.1007/978-3-030-03000-1_13](https://doi.org/10.1007/978-3-030-03000-1_13).
- [69] Prajit Ramachandran et al. *Stand-Alone Self-Attention in Vision Models*. 2019. DOI: [10.48550/ARXIV.1906.05909](https://arxiv.org/abs/1906.05909). URL: <https://arxiv.org/abs/1906.05909>.
- [70] Nicolas Carion et al. *End-to-End Object Detection with Transformers*. 2020. DOI: [10.48550/ARXIV.2005.12872](https://arxiv.org/abs/2005.12872). URL: <https://arxiv.org/abs/2005.12872>.
- [71] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2020. DOI: [10.48550/ARXIV.2010.11929](https://arxiv.org/abs/2010.11929). URL: <https://arxiv.org/abs/2010.11929>.

- [72] Zachary Teed and Jia Deng. *RAFT: Recurrent All-Pairs Field Transforms for Optical Flow*. 2020. arXiv: 2003.12039 [cs.CV].
- [73] Hugo Touvron et al. *Training data-efficient image transformers and distillation through attention*. 2020. DOI: 10.48550/ARXIV.2012.12877. URL: <https://arxiv.org/abs/2012.12877>.
- [74] Nan Yang et al. *D3VO: Deep Depth, Deep Pose and Deep Uncertainty for Monocular Visual Odometry*. 2020. DOI: 10.48550/ARXIV.2003.01060. URL: <https://arxiv.org/abs/2003.01060>.
- [75] Hengshuang Zhao, Jiaya Jia, and Vladlen Koltun. *Exploring Self-attention for Image Recognition*. 2020. DOI: 10.48550/ARXIV.2004.13621. URL: <https://arxiv.org/abs/2004.13621>.
- [76] Yusra Alkendi, Lakmal Seneviratne, and Yahya Zweiri. “State of the Art in Vision-Based Localization Techniques for Autonomous Navigation Systems”. In: *IEEE Access* 9 (2021), pp. 76847–76874. DOI: 10.1109/ACCESS.2021.3082778.
- [77] Anurag Arnab et al. *ViViT: A Video Vision Transformer*. 2021. DOI: 10.48550/ARXIV.2103.15691. URL: <https://arxiv.org/abs/2103.15691>.
- [78] Sara Atito, Muhammad Awais, and Josef Kittler. *SiT: Self-supervised vision Transformer*. 2021. DOI: 10.48550/ARXIV.2104.03602. URL: <https://arxiv.org/abs/2104.03602>.
- [79] Hangbo Bao et al. *BEiT: BERT Pre-Training of Image Transformers*. 2021. DOI: 10.48550/ARXIV.2106.08254. URL: <https://arxiv.org/abs/2106.08254>.
- [80] Carlos Campos et al. “ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual-Inertial, and Multimap SLAM”. In: *IEEE Transactions on Robotics* 37.6 (Dec. 2021), pp. 1874–1890. DOI: 10.1109/tro.2021.3075644. URL: <https://doi.org/10.1109/tro.2021.3075644>.
- [81] Mathilde Caron et al. *Emerging Properties in Self-Supervised Vision Transformers*. 2021. DOI: 10.48550/ARXIV.2104.14294. URL: <https://arxiv.org/abs/2104.14294>.
- [82] Chun-Fu Chen, Quanfu Fan, and Rameswar Panda. *CrossViT: Cross-Attention Multi-Scale Vision Transformer for Image Classification*. 2021. DOI: 10.48550/ARXIV.2103.14899. URL: <https://arxiv.org/abs/2103.14899>.
- [83] Xinlei Chen, Saining Xie, and Kaiming He. *An Empirical Study of Training Self-Supervised Vision Transformers*. 2021. DOI: 10.48550/ARXIV.2104.02057. URL: <https://arxiv.org/abs/2104.02057>.
- [84] Xiangxiang Chu et al. *Twins: Revisiting the Design of Spatial Attention in Vision Transformers*. 2021. DOI: 10.48550/ARXIV.2104.13840. URL: <https://arxiv.org/abs/2104.13840>.

- [85] Yuxin Fang et al. *You Only Look at One Sequence: Rethinking Transformer in Vision through Object Detection*. 2021. DOI: 10.48550/ARXIV.2106.00666. URL: <https://arxiv.org/abs/2106.00666>.
- [86] Ben Graham et al. *LeViT: a Vision Transformer in ConvNet's Clothing for Faster Inference*. 2021. DOI: 10.48550/ARXIV.2104.01136. URL: <https://arxiv.org/abs/2104.01136>.
- [87] Ali Hassani et al. *Escaping the Big Data Paradigm with Compact Transformers*. 2021. DOI: 10.48550/ARXIV.2104.05704. URL: <https://arxiv.org/abs/2104.05704>.
- [88] Kaiming He et al. *Masked Autoencoders Are Scalable Vision Learners*. 2021. DOI: 10.48550/ARXIV.2111.06377. URL: <https://arxiv.org/abs/2111.06377>.
- [89] Byeongho Heo et al. *Rethinking Spatial Dimensions of Vision Transformers*. 2021. DOI: 10.48550/ARXIV.2103.16302. URL: <https://arxiv.org/abs/2103.16302>.
- [90] Pedro Nuno Leite and Andry Maykol Pinto. "Exploiting Motion Perception in Depth Estimation Through a Lightweight Convolutional Neural Network". In: *IEEE Access* 9 (2021), pp. 76056–76068. DOI: 10.1109/ACCESS.2021.3082697.
- [91] Chunyuan Li et al. *Efficient Self-supervised Vision Transformers for Representation Learning*. 2021. DOI: 10.48550/ARXIV.2106.09785. URL: <https://arxiv.org/abs/2106.09785>.
- [92] Yawei Li et al. *LocalViT: Bringing Locality to Vision Transformers*. 2021. DOI: 10.48550/ARXIV.2104.05707. URL: <https://arxiv.org/abs/2104.05707>.
- [93] Ze Liu et al. *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. 2021. DOI: 10.48550/ARXIV.2103.14030. URL: <https://arxiv.org/abs/2103.14030>.
- [94] Xiaofeng Mao et al. *Towards Robust Vision Transformer*. 2021. DOI: 10.48550/ARXIV.2105.07926. URL: <https://arxiv.org/abs/2105.07926>.
- [95] Alaaeldin El-Nouby et al. *XCiT: Cross-Covariance Image Transformers*. 2021. DOI: 10.48550/ARXIV.2106.09681. URL: <https://arxiv.org/abs/2106.09681>.
- [96] Hugo Touvron et al. *Going deeper with Image Transformers*. 2021. DOI: 10.48550/ARXIV.2103.17239. URL: <https://arxiv.org/abs/2103.17239>.
- [97] Wenhai Wang et al. *Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction without Convolutions*. 2021. DOI: 10.48550/ARXIV.2102.12122. URL: <https://arxiv.org/abs/2102.12122>.
- [98] Enze Xie et al. *Segmenting Transparent Object in the Wild with Transformer*. 2021. DOI: 10.48550/ARXIV.2101.08461. URL: <https://arxiv.org/abs/2101.08461>.
- [99] Zhenda Xie et al. *Self-Supervised Learning with Swin Transformers*. 2021. DOI: 10.48550/ARXIV.2105.04553. URL: <https://arxiv.org/abs/2105.04553>.

- [100] Kun Yuan et al. *Incorporating Convolution Designs into Visual Transformers*. 2021. DOI: 10.48550/ARXIV.2103.11816. URL: <https://arxiv.org/abs/2103.11816>.
- [101] Li Yuan et al. *Tokens-to-Token ViT: Training Vision Transformers from Scratch on ImageNet*. 2021. DOI: 10.48550/ARXIV.2101.11986. URL: <https://arxiv.org/abs/2101.11986>.
- [102] Zizhao Zhang et al. *Nested Hierarchical Transformer: Towards Accurate, Data-Efficient and Interpretable Visual Understanding*. 2021. DOI: 10.48550/ARXIV.2105.12723. URL: <https://arxiv.org/abs/2105.12723>.
- [103] Daquan Zhou et al. *DeepViT: Towards Deeper Vision Transformer*. 2021. DOI: 10.48550/ARXIV.2103.11886. URL: <https://arxiv.org/abs/2103.11886>.
- [104] Yiran Zhu et al. “PoseGTAC: Graph Transformer Encoder-Decoder with Atrous Convolution for 3D Human Pose Estimation”. In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. Ed. by Zhi-Hua Zhou. Main Track. International Joint Conferences on Artificial Intelligence Organization, Aug. 2021, pp. 1359–1365. DOI: 10.24963/ijcai.2021/188. URL: <https://doi.org/10.24963/ijcai.2021/188>.
- [105] Lucas R. Agostinho et al. “A Practical Survey on Visual Odometry for Autonomous Driving in Challenging Scenarios and Conditions”. In: *IEEE Access* 10 (2022), pp. 72182–72205. DOI: 10.1109/ACCESS.2022.3188990.
- [106] Lucas Beyer, Xiaohua Zhai, and Alexander Kolesnikov. *Better plain ViT baselines for ImageNet-1k*. 2022. arXiv: 2205.01580 [cs.CV].
- [107] Stéphane d’Ascoli et al. “ConViT: improving vision transformers with soft convolutional inductive biases*”. In: *Journal of Statistical Mechanics: Theory and Experiment* 2022.11 (Nov. 2022), p. 114005. DOI: 10.1088/1742-5468/ac9830. URL: <https://doi.org/10.1088%5C%2F1742-5468%5C%2Fac9830>.
- [108] Bo-Kai Ruan, Hong-Han Shuai, and Wen-Huang Cheng. *Vision Transformers: State of the Art and Research Challenges*. 2022. DOI: 10.48550/ARXIV.2207.03041. URL: <https://arxiv.org/abs/2207.03041>.