MASTER CORO-IMARO
"CONTROL AND ROBOTICS"

2022 / 2023

Master Thesis Report

Presented by

Ke GUO

On 21/08/2023

Improvement of Visual SLAM System Based on Deep Learning Approaches

Jury

Evaluators:	Olivier KERMORGANT	Associate Professor (LS2N, ECN)
	Vincent FREMONT	Professor (LS2N, ECN)
Supervisor(s):	Thibault NEVEU	CTO (Visual Behavior)
	Daria DOUBINE	Machine Learning Engineer (Visual Behavior)

Abstract

Over the past forty years, the rise of automation and robotics has brought the SLAM problem to the forefront of both industrial and academic research. SLAM, a system for mapping unknown environments and tracking robot or sensor movement, has seen its visual variant, Visual SLAM, gain prominence due to its simple sensor configuration and applicability in areas like Augmented Reality and UAVs. With the advent of deep learning, which has shown prowess in sectors like industry, biology, and chip design, there's growing interest in its integration with visual SLAM. This paper delves into popular SLAM systems, the fusion of deep learning with vSLAM, and introduces the innovative PnP Transformer, a deep learning approach to the traditional PnP problem. Additionally, we explore the Bird's Eye View (BEV) for obstacle-free navigation and delve into the enhancement of Visual Odometry (VO) using techniques like SuperPoint+LightGlue. Our efforts culminate in a refined VO system, ensuring reliable autonomous navigation. This paper presents interesting insights into the integration of deep learning with VSLAM, offering a foundation for scholars keen on further exploration in this domain. Through innovative approaches and methodologies, we shed light on the potential synergies between these two fields. Our findings serve as a valuable starting point for those aiming to push the boundaries of VSLAM through the lens of deep learning.

Keywords: Visual SLAM, Deep learning, PnP, Transformer, BEV, Feature detection

Acknowledgements

Firstly I would like to thank all the teachers I met at the ECN for teaching me professionally and guiding me in my research direction to let me arrive here and choose this thesis. In particular, I would like to thank Professor Olivier KERMORGANT for his guidance in my major and the internship. I am also very grateful to my classmates, the meeting of minds and discussions from different countries (or different cities in the same country) gave me a deeper understanding of the diversity of the world.

Secondly, I really appreciate the company, Visual Behavior, for giving me this internship position, which I was lucky enough to do in my field of interest. I would also like to thank all of my coworkers for all the help and time they have given me during these six months. I am especially grateful to Daria DOUBINE for her guidance on the direction of my research and this report review, and for always being there to discuss possible directions with me whenever I was confused!

Finally, I would like to say thanks to my parents for their support, to my friends for accompanying me, and to my girlfriend LI Xinyi. Her upbeat personality always helps me get out of the gloom when I am experiencing anxiety. I am sincerely grateful to everyone I have met for making me become a better version of myself.

Notations

$z_{1:t}$:	A series of sensor measurement data from bringing up the robot to t
$u_{1:t}$:	A series of control data
m:	Map information
$x_{1:t}$:	Robot trajectory state
$\mathbf{x}_s$:	Stereo keypoints (u_L, v_L, u_R)
(u_L, v_L):	Coordinates on the left image
u_R:	Horizontal coordinate in the right image
$\mathbf{x}_m$:	Monocular keypoints (u_L, v_L)
$\mathbf{p}$:	2D point in the previous image in homogeneous coordinates
$\mathbf{P}$:	3D point in the current camera pose frame in homogeneous coordinates
$\mathbf{K}$:	Camera's intrinsic matrix
$\mathbf{R}$:	3x3 rotation matrix
$\mathbf{t}$:	3x1 translation vector
$\hat{R}$:	The predicted rotation matrix
$\hat{t}$:	the predicted translation vector
$[\mathbf{R}\|\mathbf{t}]$:	Camera's extrinsic matrix
pos:	The position of the token in the sequence
Q:	The matrix of queries
K:	The matrix of keys
V:	The matrix of values
d_k:	The dimensionality of the keys
T_0:	The input initial transformation
$pos3d$:	A 3D position

Abbreviations

SLAM:	Simultaneous Localization And Mapping
AR:	Augmented Reality
vSLAM:	Visual Simultaneous Localization And Mapping
UAV:	Unmanned Aerial Vehicle
ROS:	Robot Operating System
DL:	Deep Learning
EKF:	Extended Kalman Filter
IMU:	Inertial Measurement Unit
RANSAC:	Random Sample Consensus
RGB:	Red, Green, Blue
RGB-D:	Red, Green, Blue - Depth
LiDAR:	Light Detection And Ranging
ORB:	Oriented Rotated BRIEF
BRIEF:	Binary Robust Independent Elementary Features
RBPF:	Rao-Blackwellized Particle Filter
PnP:	Perspective-n-Point
BA:	Bundle Adjustment
DoF:	Degree of Freedom
CNN:	Convolutional Neural Network
2D:	2-Dimensional
3D:	3-Dimensional
GNN:	Graph Neural Network
VLAD:	Vector of Locally Aggregated Descriptors
PCA:	Principal Component Analysis
VO:	Visual Odometry
DPT:	Dense Prediction Transformer
ICP:	Iterative Closest Point
ToF:	Time of Flight
ViT:	Vision Transformer
EPnP:	Efficient Perspective-n-Point
MLP:	Multilayer Perception
SURF:	Speeded-Up Robust Features
FAST:	Features from Accelerated Segment Test
BRIEF:	Binary Robust Independent Elementary Features
SIFT:	Scale-Invariant Feature Transform
GT:	Ground Truth
NNDR:	Nearest Neighbor Distance Ratio
ReLU:	Rectified Linear Unit
MLP:	Multilayer Perceptron Model
W&B:	Weights&Biases
BEV:	Bird's Eye View
KM:	Kuhn-Munkres
onnx:	Open Neural Network Exchange

List of Figures

List of Tables

Contents

Introduction

With the rapid development of automation and the emergence of more and more robots in the last forty years, the SLAM problem has gradually come to the attention of experts in industry as well as in academia[11]. SLAM or Simultaneous Localization and Mapping is a system for obtaining the 3D structure of an unknown environment and robot or sensor motion in the environment[12]. Visual SLAM, which uses only visual information, is the most discussed of all because the sensor configuration is simple and the wide range of applications such as the AR(Augmented Reality)[13] and UAV(Unmanned Aerial Vehicle)[14].

With the development of deep learning techniques, deep learning has become an interdisciplinary subject. It is important to note that deep learning is performing well in industry, biology and chip design[15]. There are now also many roboticists discussing the possibility of combining deep learning with visual SLAM.

In the following literature view part, first, we introduce some common and popular SLAM systems and vSLAM(Visual Simultaneous Localization and Mapping). In recent research, there has been a notable convergence of deep learning methodologies with visual Simultaneous Localization and Mapping (vSLAM). This integration has been particularly evident in areas such as feature detection and matching, place recognition, and visual odometry. In the following sections, we will delve into each of these aspects to understand the advancements and implications of combining deep learning techniques with vSLAM.

A notable innovation in this report is the PnP Transformer. This model represents a paradigm shift in addressing the PnP problem through deep learning. Traditional PnP algorithms often rely on geometric methods to estimate camera pose from 2D-3D correspondences and the RANSAC method to remove outliers. However, the PnP Transformer seeks to leverage the representational capacity of deep neural networks to learn these correspondences and subsequently predict camera poses. While the results from early iterations of the PnP Transformer have been mixed, the model undeniably offers a fresh perspective and serves as a foundation for future research endeavors.

Another pivotal development is the exploration of the Bird's Eye View (BEV) and Visual Odometry (VO). BEV, as the name suggests, provides a top-down perspective of the environment, offering a comprehensive view devoid of obstructions typical in side or ground-level visuals. This perspective is invaluable for tasks like navigation and obstacle detection. On the other hand, VO plays a crucial role in tracking the motion of a camera relative to a static environment. In our quest to enhance Visual Odometry (VO) performance, my primary role was to identify and implement robust methods for improved motion estimation. After experimenting with various techniques, we settled on the SuperPoint+LightGlue combination for superior feature point extraction and matching. To further refine our results, we introduced a transformation filter to eliminate extreme or erroneous transformations and integrated odome-

try correction. This approach, leveraging both the robot's inherent odometry data and our VO estimates, ensured accurate and consistent pose estimations, even in challenging environments. Through these innovations, we achieved a more reliable and precise VO system, crucial for dependable autonomous navigation.

In conclusion, the fusion of SLAM with deep learning heralds a new era in robotics and automation. While challenges persist, especially in the seamless integration of traditional algorithms with neural architectures, the potential for groundbreaking advancements is palpable. Through this paper, we aspire to provide readers with a nuanced understanding of the evolving landscape of SLAM, deep learning, and their confluence.

Chapter 1

Visual SLAM System

1.1 SLAM System

SLAM involves building a model (the map) of the environment that the robot's onboard sensors(various types possible) are detecting while simultaneously estimating the state of the robot. In some cases, the pose of the robot serves as a description of its current state. However, other elements such as the robot's velocity, sensor biases, and calibration parameters may also be present. The map, on the other hand, is a visual depiction of important characteristics of the area in which the robot functions (such as the location of landmarks and obstacles). It is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of a robot's pose within it and this problem initially appears to be a chicken-and-egg problem. The build of the map needs the premise of knowing the exact pose of the robot. And a good map can be used to reduce the cumulative error of the robot's pose during its motion. How to solve this chicken-and-egg problem perfectly has recently been investigated by scholars of robotics and computer vision. This challenging problem in mobile robotics has been researched for more than three decades where scientists use different techniques to improve the computational speed, accuracy, and robustness of the algorithm.

A thorough historical review of the first 20 years of the SLAM problem is given by Durrant–Whyte and Bailey in their research [16]. It primarily refers to what is known as the classical age (1986–2004). In that classical age, probability-based algorithms were the most common such as extended Kalman filters (EKF)[17], Rao–Blackwellized particle filters[18], and maximum likelihood estimation[19]. They solved the fundamental difficulties related to effective and reliable data association. But how to obtain a real-time SLAM system and how to make better use of the vision sensors became the next research direction. The next era is known as the "age of algorithmic analysis" (2004–2015) and is partially introduced by Dissanayake et al. in [20]. Observability, convergence, and consistency were some of the core characteristics of SLAM that were studied during the algorithmic analysis phase. During this time, the primary open-source SLAM libraries were established, and it was also known how crucial sparsity is to creating efficient SLAM solvers. Most of these packages are implemented on the ROS. ROS is Robot Operating System which is an open-source robotics middleware suite. It is a collection of software frameworks for robot software development, it provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. With the rapid development and complexity of the robotics field, the need for code reusability and modularity has become stronger. The emergence of ROS is to comply with this demand and has a wide range of applications in industrial robots and intelligent robots. Table 1.1 roughly illustrates some of the common SLAM legacy algorithm packages in the ROS. It is important to note that these packages are primarily aimed at robots equipped

with LIDAR(Light Detection and Ranging) sensors.

Table 1.1: Different legacy packages related in SLAM on ROS

	Merits	Demerits	Principle
HectorSLAM	No need for odometer, high accuracy	Need a laser scanner with high update frequency and low measurement noise	Scan-matching (Gaussian-Newton equation)
KartoSLAM	Have a better performance when drawing in a larger environment	Require a large memory to store nodes	Spare Pose Adjustment(SPA)
3D_SLAM	Build a 3D map and get more information	Need a 3D lidar and better cpu to support a lot of calculations	Point-to-Plane (Iterative Closest Point)
GmappingSLAM	high accuracy in a small environment, low requirement for the scanning frequency of the lidar	Need a lot of particle simulations to get good results	Rao-Blackwellized particle filter

A general review of the main SLAM surveys to date is shown in Table 1.2, noticing that most recent surveys only cover some specific subfields of SLAM.

Table 1.2: The surveys and tutorials of SLAM[1]

Year	Topic	Reference
2006	Probabilistic approaches and data association	Durrant–Whyte and Bailey[16]
2008	Filtering approaches	Aulinas et al.[21]
2011	Observability, consistency and convergence	Dissanayake et al.[20]
2012	Visual odometry	Scaramuzza and Fraundofer[22]
2016	Visual place recognition	Lowry et al.[23]
2016	Theoretical aspects	Huang and Dissanayake[24]

The anatomy of a modern SLAM system could be separated into two main parts: the front end(some old papers call it the lower level) and the back end(some old papers call it the higher level). The front end abstracts sensor data into models that are used for estimating, and the back end makes inferences using the front end's abstracted data. A summary of this architecture is shown in Fig.1.1

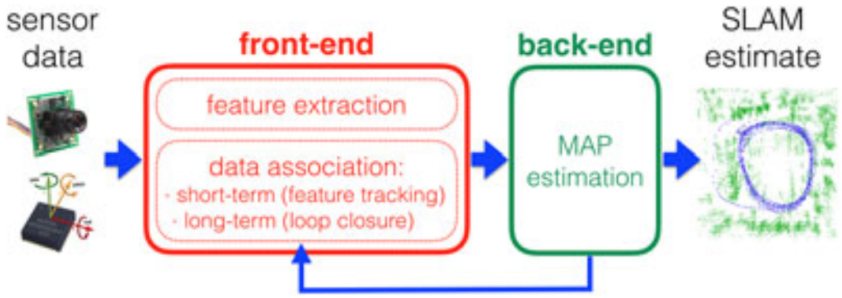


Figure 1.1: Front end and back end in a typical SLAM systems[1]

Explained in a mathematical way, the SLAM problem can be expressed as the following form:

$$p(x_{1:t}, m|u_{1:t}, z_{1:t}) \tag{1.1}$$

where $z_{1:t}$ is a series of sensor measurement data from bringing up the robot to t, $u_{1:t}$ is a series of control data, and the map information m and the robot trajectory state $x_{1:t}$ are what we want to estimate. Later on, depending on the different SLAM algorithms we can have different approaches to solving this mathematical problem. For a better understanding, we use *Slam_gmapping* package on ROS as an example to analyze this SLAM problem. *Slam_gmapping* is an algorithm based on RBPF (Rao-Blackwellized particle filter) method.

According to probability theory, the above formula can be simplified to:

$$p(x_{1:t}|u_{1:t}, z_{1:t})p(m|x_{1:t}, z_{1:t}) \tag{1.2}$$

In this way, the SLAM problem is reduced to two problems. Among them, the map construction of the known robot pose is a simple problem, so the estimation of the robot pose is a key issue.

In *slam_gmapping*, it uses particle filtering to estimate the pose of the robot and build a map for each particle. Therefore, each particle contains the robot's trajectory and the corresponding environment map. Here we use the Bayesian formula to simplify the calculation of the pose. (The specific derivation is quite complicated, here we only explain the results)

$$p(x_{1:t}|u_{1:t}, z_{1:t}) = \eta p(z_t|x_t)p(x_t|x_{t-1}, u_t)p(x_{1:t-1}|u_{1:t-1}, z_{1:t-1}) \tag{1.3}$$

where using $p(x_{1:t-1}|u_{1:t-1}, z_{1:t-1})$ to represent the robot trajectory at the previous moment, $p(x_t|x_{t-1}, u_t)$ is the kinematic model for propagation so that we can obtain the predicted trajectory of each particle. For each particle after propagation, the observation model $p(z_t|x_t)$ is used for weight calculation and normalization so that the robot trajectory at that moment is obtained.

But when the detected environment is quite large, we need tons of particles to estimate the pose, which is unreasonable in actual operation. In *Slam_gmapping*, it uses the maximum likelihood estimation to optimize this problem. The pseudocode of the Rao-Blackwellized Particle Filter Algorithm is shown in Appendix A.

1.2 Visual SLAM System

Because the sensor setup is straightforward and the technical challenges are greater than those of other approaches, SLAM utilizing simple cameras has been the subject of considerable discussion in recent years.[2] This only using the visual information as the input of the SLAM system

technology is referred to as visual SLAM(vSLAM). vSLAM algorithms have widely proposed in the field of computer vision[22], robotics such as UAV[14], and AR[13]. Although several vSLAM algorithms have been presented for various goals across various research communities, they essentially share the same fundamental technical concepts and can be used for various goals simultaneously.

In its most general form, vSLAM consists of several modules:

- Initialization
- Tracking
- Mapping
- Relocalization
- Global map optimization

In order to begin vSLAM, a specific coordinate system must be defined for 3D reconstruction and camera pose estimation in an unknowable environment. As a result, during initialization, it is important to first specify the global coordinate system before reconstructing a portion of the environment as a first map in the global coordinate system. Tracking and mapping are done after startup in order to continually estimate camera pose. To estimate the camera pose of the image with respect to the map, the reconstructed map is tracked in the image during tracking. This is accomplished by first locating 2D-3D correspondences between the image and the map using feature matching or feature tracking in the image. By resolving the Perspective-n-Point (PnP) problem[25], the camera posture is then calculated from the correspondences. It should be emphasized that the majority of vSLAM algorithms rely on intrinsic camera characteristics being calibrated and known in advance. With translation and rotation of the camera in the global coordinate system, a camera pose is therefore typically identical to extrinsic camera parameters. When a camera detects unmapped areas that haven't been mapped before, the map is enlarged by computing the 3D structure of the environment.

When tracking fails due to rapid camera motion or other disruptions, relocalization is necessary. In this situation, it is required to once again compute the camera pose in relation to the map. This problem is also called kidnapped robot problems in robotics. The other module is global map optimization. In general, the map contains cumulative estimating inaccuracy based on the camera movement's distance. The global map optimization is typically done to suppress the error. During this process loop closing is an necessary technique. A closed loop is first searched for a match between a current image and previously acquired images in the loop closing process. If the loop is found, it signifies that a previous view was captured by the camera. In this instance, it is possible to determine the cumulative error that happened during camera movement.

Pose-graph optimization, which optimizes camera postures, has been frequently utilized to suppress the accumulated error[26]. This technique builds a consistent graph to suppress the optimization errors by representing the relationship between camera poses as a graph. Through the optimization of both the camera and the map poses, bundle adjustment (BA)[27] is also used to reduce the map's reprojection error. This optimization technique is used in big environments to effectively reduce estimation mistakes. Due to the minimal accumulated error in small environments, loop closing is not always necessary.

Depending on whether all images are used in the vSLAM process, vSLAM can be classified into two types, feature-based methods, and direct methods. In feature-based methods, the features

of the front and back frames or the front and back keyframes need to be matched to calculate the pose relationship between the front and back frames. There are some typical feature-based methods such as MonoSLAM[28], PTAM[29] and ORB-SLAM[30], which are all based on the monocular camera. As an extension of PTAM, ORB-SLAM includes BA, vision-based closed-loop detection, and 7 DoF pose graph optimization. ORB-SLAM is widely regarded as the most complete feature-based monocular vSLAM system. The summary of feature-based methods is shown in Fig.1.2.

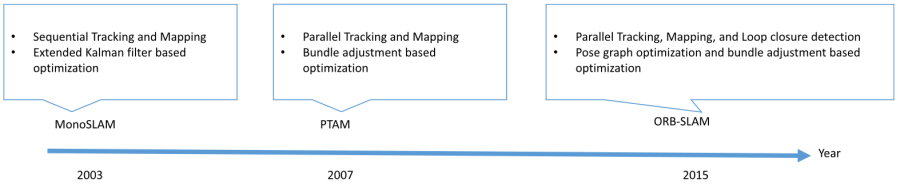


Figure 1.2: Summary of feature-based methods[2]

As opposed to the feature-based method, direct methods directly use an input image without any abstraction using handcrafted feature detectors and descriptors. In general, photometric consistency is used as an error measurement in direct methods whereas geometric consistency such as the positions of feature points in an image is used in feature-based methods. There are several typical direct methods algorithms such as SVO[31], DSO[32], LSD-SLAM[33] and DTAM[34]. Map density can be used to classify direct techniques. The summary of direct methods is shown in Fig.1.3. Dense methods use all depth values of every pixel in each keyframe to generate a map. In contrast to dense methods, semi-dense, and sparse methods focus on the applications based on the tracking of sensor poses and only use some of the pixels.

Figure 1.3: Summary of direct methods[2]

Compared with the feature-based methods, the direct method can reconstruct the whole 3D information but is more costly to calculate. In order to better implement a real-time SLAM system, we next discuss only the feature-based method. Depending on the camera setup, the vSLAM can be classified as monocular vSLAM, stereo vSLAM and RGB-D vSLAM. Based on the actual situation of my internship company, we will then describe in detail the ORB-SLAM2[3] which is a featured-based stereo vSLAM algorithm. (Although the ORB-SLAM3[35] is released it is more for the Visual-Inertial SLAM, so we do not introduce too much).

1.3 ORB-SLAM2

The ORB-SLAM2 is a comprehensive SLAM system with map reuse, loop closing, and relocalization modules for monocular, stereo, and RGB-D cameras. The system operates in real-time on common CPUs in a variety of environments. It has a light localization mode with zero-drift localization that uses visual odometry tracks for unmapped locations and matches to map points. The structure of the ORB-SLAM2 system is shown in Fig.1.4.

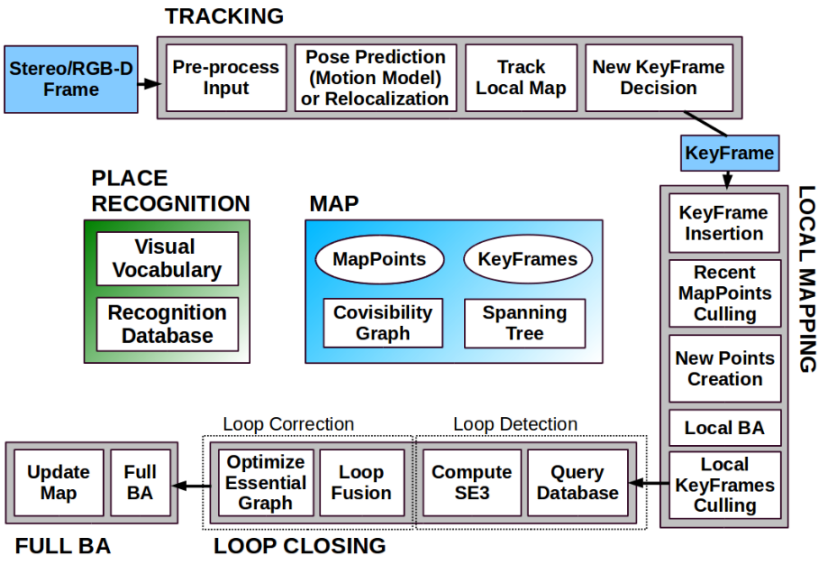


Figure 1.4: ORB-SLAM2 system threads and modules[3]

The system runs three main parallel threads:

- tracking to localize the camera with each frame by locating feature matches to the local map and reducing the reprojection error using motion-only BA.

- local mapping to manage the local map and optimize it.

- loop closing to detect large loops and correct the accumulated drift using a pose-graph optimization. Following the pose-graph optimization, this thread starts a fourth thread to execute full BA in order to determine the ideal structure and motion solution.

The system includes a place recognition module based on DBoW2[36] for loop detection as well as relocalization in the event of tracking failure (for example, an occlusion). The system keeps track of a minimal spanning tree connecting all keyframes and a covisibility graph[34] that connects any two keyframes by observing shared points. As a result, tracking and local mapping can act locally, enabling the use of huge environments. These graph structures also provide a framework for the pose-graph optimization process that is carried out when a loop is closed.

The system uses the same ORB features[37] for tasks involving place recognition, tracking, and mapping. These features exhibit strong invariance to camera auto-gain and auto-exposure, as well as variations in illumination, and are resistant to rotation and scaling. Additionally, they exhibit good precision/recall performance in bag-of-word place recognition and are quick to extract and match, enabling for real-time operation.

The system pre-processes the input to extract features at salient keypoint locations as shown in Fig.1.5.

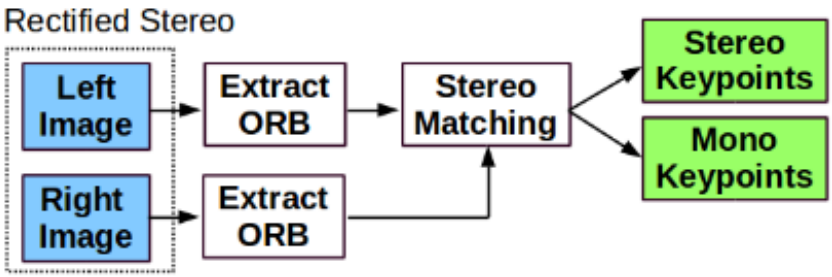


Figure 1.5: ORB-SLAM2 input pre-processing[3]

Stereo keypoints are defined by three coordinates $\mathbf{x}_s = (u_L, v_L, u_R)$, where (u_L, v_L) is the coordinates on the left image and u_R is the horizontal coordinate in the right image. Monocular keypoints are defined by two coordinates $\mathbf{x}_m = (u_L, v_L)$ on the left image and correspond to all those ORB for which a stereo match could not be found[3]. In order not to make this bibliography too difficult to understand, the mathematical derivation of other modules of the ORB-SLAM algorithm can be found in [30][3]. The comparison to the state-of-the-art shows that ORB-SLAM2 achieves in most cases the highest accuracy.

Chapter 2

Deep Learning Approaches in vSLAM

With the development of artificial intelligence technology, more and more disciplines are combined with deep learning technology, which is normally used to solve optimization problems and achieve outstanding performance by leveraging the advantages of deep learning. In this chapter, we introduce the application of several representative deep-learning techniques in the vSLAM system.

2.1 SuperPoint

SuperPoint[4] is a self-supervised algorithm for training interest point detectors and descriptors suitable for a huge number of multiple-view geometry problems in computer vision. In the sparse vSLAM system, the first step is to extract interest points as known as keypoints from images. Interest points are 2D locations in an image that should be stable and repeatable from different lighting conditions and viewpoints. In general, deep learning methods work better with annotation, i.e. supervised learning. But for the detection of interest points, the notion of interest point detection is semantically ill-defined. Therefore, SuerPoint provides a self-supervised method using self-training in place of human supervision to identify interest points in real photos. Instead of undertaking a significant human annotation effort, this method involves building a sizable dataset of interest point locations in real photos that are supervised by the interest point detector itself. The overview of the SuperPoint is shown in Fig.2.1.

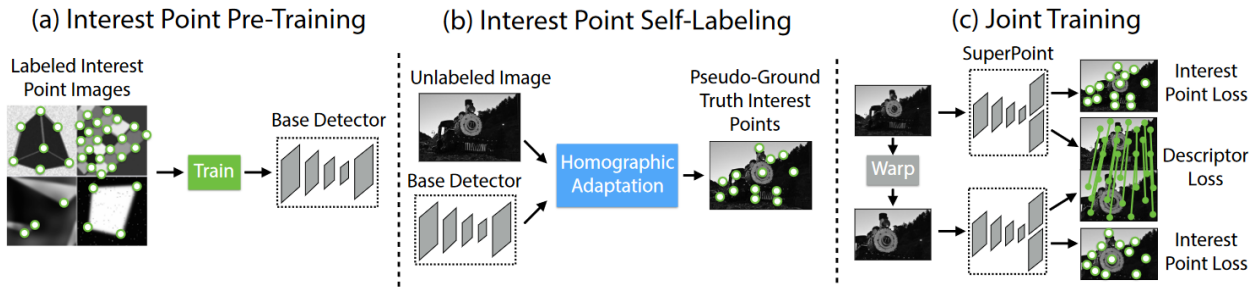


Figure 2.1: The overview of the SuperPoint[4]

It first trains a fully convolutional neural network on millions of samples from the artificial dataset called *Synthetic Shapes* in order to provide the pseudo-ground truth interest points(Fig.2.1(a)). The simple geometric shapes in the synthetic dataset have clear interest point positions. The resulting trained detector is known as MagicPoint. MagicPoint misses a lot of possible interest point sites when compared to traditional interest point detectors on a variety of visual textures and patterns. It created a multi-scale, multi-transform technique called homographic adaptation to close this performance difference on real photos.

Homographic Adaptation is designed to enable self-supervised training of interest point detectors. To aid an interest point detector observe the area from a variety of views and scales, it warps the input image many times. To improve the MagicPoint detector's effectiveness and produce the phony ground truth interest spots, the training process pairs it with homographic adaptation (Fig.2.1(b)). We called the resulting detector as SuperPoint since the resulting detections are more repeatable and activate on a wider range of stimuli.

After identifying reliable and repeatable interest locations, the most frequent procedure is to attach a fixed-dimensional descriptor vector to each point in order to use it for higher-level semantic tasks, such as picture matching. Consequently, it integrates SuperPoint with a descriptor subnetwork in the last step (Fig.2.1(c)). It gives rise to state-of-the-art homography estimation results when compared to LIFT [38]and ORB[37].

2.2 SuperGlue

SuperGlue[5] is a neural network that matches two sets of local features by searching for correspondences simultaneously and excluding locations that cannot be matched. SuperGlue could jointly analyze the underlying 3D scene and feature assignment based on a flexible context aggregation method with the attention mechanism. It proposes to learn the matching process from pre-existing features using a unique neural architecture termed SuperGlue, as opposed to learning better task-agnostic local features followed by straightforward matching heuristics and methods. The network sits exactly in the middle of the SLAM problem, which is generally divided between the visual feature extraction front-end and the bundle adjustment or posture estimation back-end(as mentioned in Fig.1.1). SuperGlue is a learnable middle-end as shown in Fig.2.2.

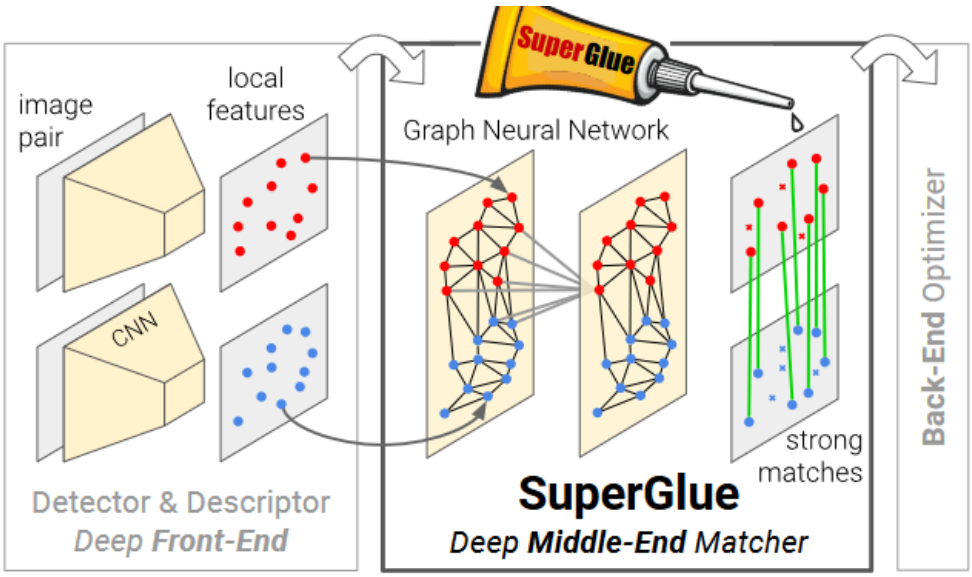


Figure 2.2: Feature matching with SuperGlue[5]

In this study, discovering the partial assignment between two sets of local features is referred to as learning feature matching. By resolving a linear assignment problem, which can be relaxed to an optimal transport problem and solved differently, it revisits the traditional graph-based matching technique. A Graph Neural Network (GNN) predicts this optimization's cost function. Drawing inspiration from the success of the Transformer[39], It draws on self- (intra-image) and cross- (inter-image) attention to maximize both the spatial relationships of the keypoints

and their visual appearance. This formulation gracefully handles occlusion and non-repeatable keypoints while enforcing the assignment structure of the predictions. It also enables to decrease in the cost to learn complex priors. This approach is trained from beginning to end from image pairs and discovers posture estimate priors from a sizable annotated dataset, giving SuperGlue the ability to analyze the 3D scene and the assignment. On the problem of pose estimation, when combined with SuperPoint(front-end), SuperGlue beats previously developed methods in challenging real-world indoor and outdoor environments.

2.3 LightGlue

As the above introduced, SuperGlue performs exceptionally better than the classical matching method for tasks like aerial matching, object pose estimation, and even fish re-identification. But the drawback of this Transformer-based architecture is it is quite computationally expensive. Especially for the VO problem, the efficiency of image matching is critical. Therefore the researchers proposed LightGlue[6] which is an upgrade and drop-in replacement version of SuperGlue with only benefits.

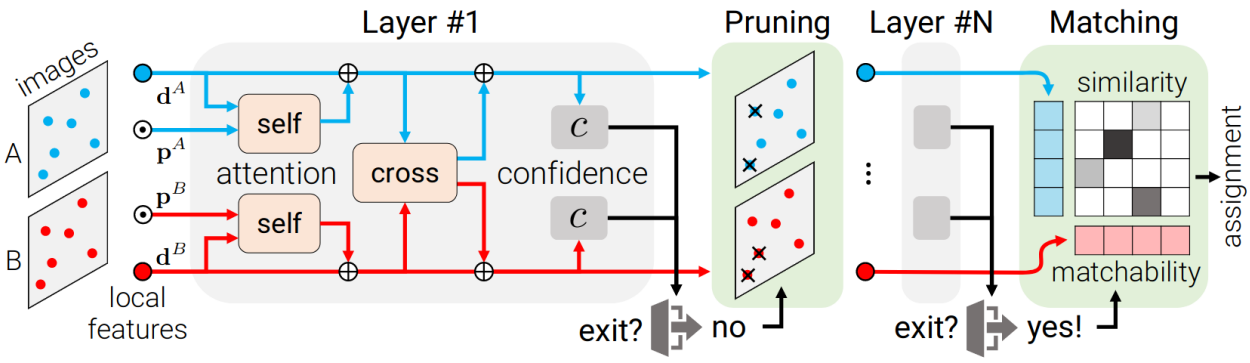


Figure 2.3: The LightGlue architecture[6]

The architecture of LightGlue is shown in Fig.2.3. The efficiency of LightGlue is achieved by two stages: 1) Initially, the model predicts a series of links or relationships (referred to as correspondences) following each computational block. This could be understood as the model identifying patterns or making connections after processing each set of information. 2) Subsequently, the model is given the capability to self-assess these identified patterns. This introspection process lets the model determine if it requires more computational effort to establish additional patterns or links. This two-step process ensures efficient computation by allowing the model to self-regulate its process based on the complexity of the data.

The LightGlue model further enhances this efficiency by filtering out non-matchable points in the early stages of the computation process. Non-matchable points are data points that can't be paired or linked with any other points. By discarding these points, the model can focus its computational resources on the covisible area, or the area where data points are visible together or have potential relationships. This streamlined focus ensures that computational resources are not wasted on data points that do not contribute to establishing meaningful patterns.

2.4 DPT

After getting the matched keypoints from the images, some specific pose estimation methods of VO such as ICP(Iterative Closest Point) or PnP(Perspective-n-Point) need the depth information of the keypoints to reconstruct the 3D points. In the absence of a depth camera like ToF(Time of Flight) camera, how to reconstruct the depth information from the images only is a tricky problem. DPT[7] or called Dense Prediction Transformer is designed to solve this problem.

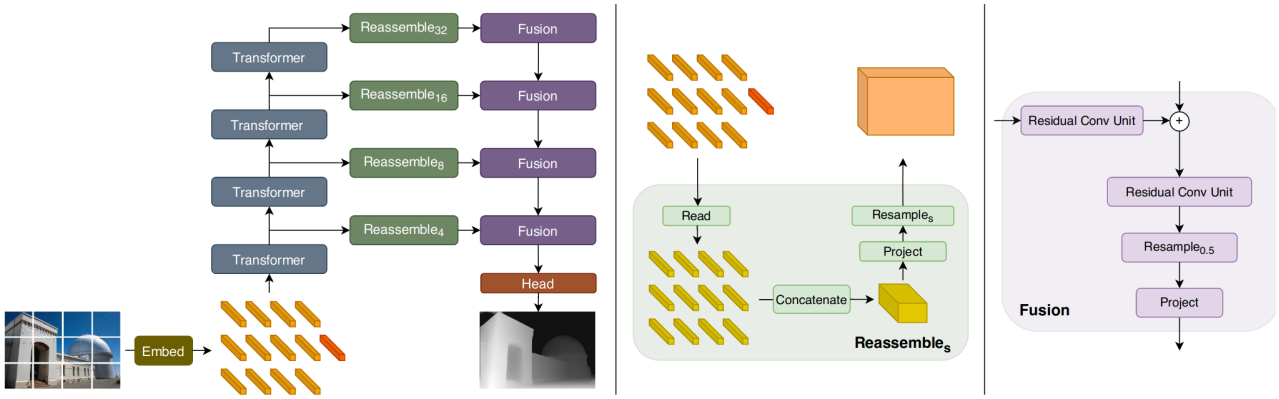


Figure 2.4: The DPT architecture[7]

The DPT architecture is shown in Fig.2.4. DPT takes the recent development architecture ViT(Vision Transformer)[40] which reads the input as a sequence of image patches (much like words in a sentence) and applies self-attention mechanisms to capture both local and global dependencies, as the backbone. In DPT, the ViT converts an image into a set of feature vectors, also known as a 'bag-of-words' representation. This set of feature vectors is then transformed back into image-like feature maps at different resolutions. Next, these feature maps are gradually merged into the final dense prediction through a convolutional decoder. The decoder utilizes convolutional layers, which are effective in maintaining spatial information, to decode the higher-level features back into the original image resolution.

These properties are particularly beneficial for dense prediction tasks. The global receptive field facilitates globally coherent predictions because the model can consider the full context when making a prediction for each point. Meanwhile, maintaining constant dimensionality throughout processing helps produce fine-grained predictions because detailed information is preserved throughout the model. This means that the DPT approach is naturally equipped to generate predictions that are detailed and coherent across the entire image.

2.5 TartanVO

TartanVO[8] is the first learning-based visual odometry model. One of the core elements of a visual SLAM system is visual odometry (VO). It falls within the categories of learning-based methods and geometric-based methods. The development of a solid and dependable VO technique in real-world situations for practical applications remains a difficult topic. On one hand, geometric-based methods are not robust enough in many real-life situations. On the other hand, although learning-based methods demonstrate robust performance on many visual tasks, including object recognition, semantic segmentation, depth reconstruction, and optical flow, it

is not so good on VO. The reasons for this are twofold: first, the existing VO models are trained without sufficient diversity, which is important for learning-based methods to be able to generalize. For example, a VO trained with indoor data, will perform poorly outdoors. Secondly, the learning-based method is a kind of black box method. Some basic aspects of the VO problem that is well-articulated in geometry-based VO theories are mostly ignored by the majority of the current learning-based VO models, such as the scale ambiguity and camera intrinsic parameters.

Therefore TartanVO designs an up-to-scale loss function to deal with the scale ambiguity of monocular VO and create an intrinsic layer (IL) in the VO model enabling generalization across different cameras. The model is substantially more resilient in difficult scenarios when compared to geometry-based approaches. The model is trained with the SLAM dataset *TartanAir*, which provides a huge number of diverse synthetic data in challenging environments. The overview of TartanVO is shown in Fig.2.5

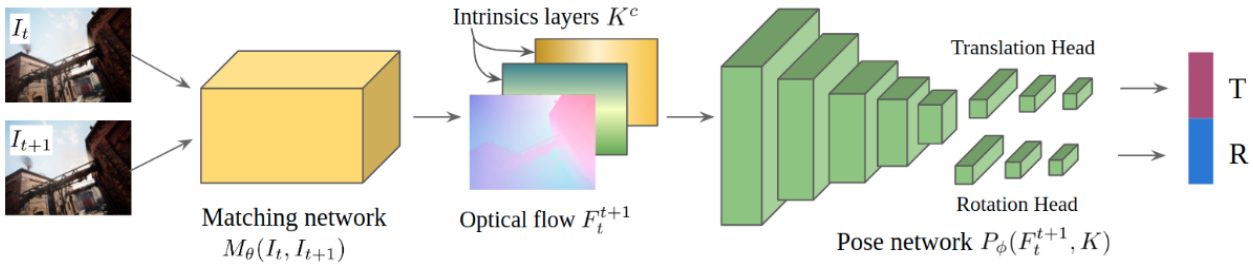


Figure 2.5: The overview of the TartanVO[8]

Chapter 3

PnP Transformer

3.1 Classical PnP

The PnP problem, or Perspective-n-Point problem, is a fundamental issue in the field of computer vision and photogrammetry. It is also the common method to solve the pose estimation problem during the VO. It refers to the task of estimating the pose of a calibrated camera given a set of n 3D points in the world and their corresponding 2D projections in the image. But more particularly in VO, we consider the pose relationship between the previous frame and the current frame. The general camera projection equation is shown as:

$$\mathbf{p} = \mathbf{K}[\mathbf{R}|\mathbf{t}]\mathbf{P} \tag{3.1}$$

where $\mathbf{p}$ is the 2D point in the previous image, in homogeneous coordinates (usually a 3-vector). $\mathbf{K}$ is the camera's intrinsic matrix, containing information about the camera's internal characteristics such as focal length and optical centers. $\mathbf{R}$ is the 3x3 rotation matrix and $\mathbf{t}$ is the 3x1 translation vector, both defining the camera pose. $[\mathbf{R}|\mathbf{t}]$ is the extrinsic matrix, showing the relation of the 3D points in the current camera pose frame to the previous frame. $\mathbf{P}$ is the 3D point in the current camera pose frame, in homogeneous coordinates (usually a 4-vector).

Therefore how to derive the extrinsic matrix $[\mathbf{R}|\mathbf{t}]$ from, $\mathbf{K}$, $\mathbf{p}$, $\mathbf{P}$ is named the Perspective-n-Point problem. Now there are ways to solve this problem like P3P[41] or EPnP[42]. It should be noted that the PnP problem can be relatively straightforward to solve when there is no noise in the data and when the correspondence between 3D points and 2D points is known. However, in real-world scenarios, these conditions are often not met, making the problem more complex. Noise can be present in the measurements and sometimes, it might not be clear which 3D points correspond to which 2D points, typically we could use the RANSAC(Random Sample Consensus) method to filter the outliers and get the local best pose solution. The pseudocode of RANSAC+EPnP is shown as:

Algorithm 1 RANSAC with EPnP Algorithm

Input:
- $\mathbf{p}$, the array of observed 2D points
- $\mathbf{P}$, the array of corresponding 3D points
- $\mathbf{K}$, the camera intrinsic matrix
- $iterationsCount$, the maximum number of RANSAC iterations
- $reprojectionError$, the maximum allowed reprojection error to treat a point as an inlier

Output:
- $\mathbf{R}$, the rotation matrix
- $\mathbf{t}$, the translation vector

inliers, the inliers that were used for final estimation
Process:
Initialize best score to infinity
Initialize best **R**, **t** to null
for $i = 1$ to *iterationsCount* **do**
Select a random subset of **p** and corresponding **P**
Call EPnP on the subset to estimate **R**, **t**
Compute the reprojection error for all points
Count the inliers falling within the *reprojectionError*
if count of inliers > best score **then**
Update the best score, best **R**, and best **t**
end if
end for
Refine the best **R**, and best **t** using all inliers
Return: best **R**, best **t**, inliers =0

And the EPnP algorithm pseudocode is shown as:

Algorithm 2 Efficient Perspective-n-Point (EPnP) Algorithm

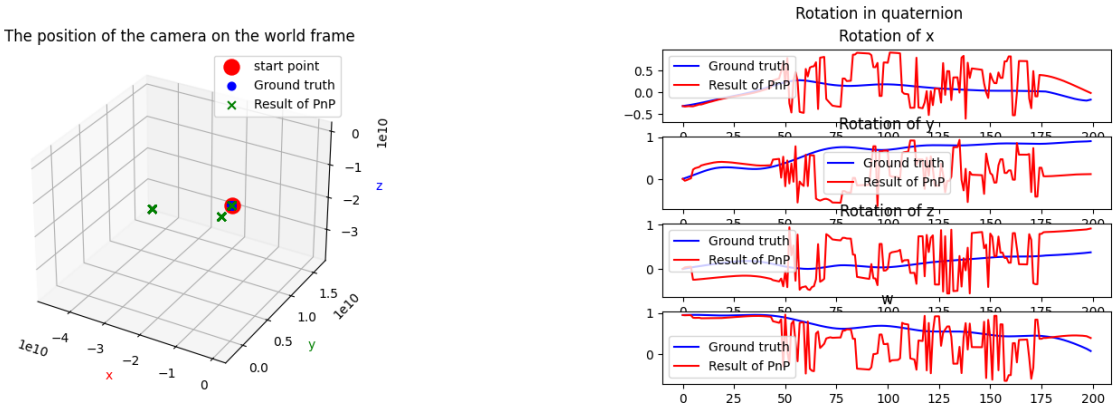
Input:
p, the array of observed 2D points
P, the array of corresponding 3D points
K, the camera intrinsic matrix
Output:
R, the rotation matrix
t, the translation vector
Process:
Construct the camera control points $C_i, i = 1, ..., 4$ from the centroid and covariance matrix of **p**
Construct the object control points $P_j, j = 1, ..., 4$ from the centroid and covariance matrix of **P**
Compute the weights w_{ij} that minimize the reprojection error with respect to the control points
Compute the matrix M from the weights w_{ij}, the object control points P_j, and the observed points **p**
Compute the SVD of M and obtain the solution for the translation and rotation
Return: **R**, **t** =0

Thanks to the package *opencv*, the above algorithm is very easy to implement in both Python and C++, as there are already encapsulated functions. Because the company projects are primarily based on Python development, we choose the built-in function *cv2.solvePnPRansac()* to implement the RANSAC with EPnP algorithm. After establishing the foundational structure of the PnP algorithm, another pivotal factor influencing its efficacy pertains to the extraction of keypoints, or called feature points. The robustness of these feature points is paramount. Here, robustness is characterized by two primary attributes: 1) the feature points should be uniformly dispersed across the image while maintaining a substantial count; and 2) across two successive images, the correspondence of these feature points should remain consistent, exhibiting minimal susceptibility to alterations in lighting and motion dynamics.

Two of the more common types of feature point extraction are ORB(Oriented FAST and Rotated BRIEF) and SURF (Speeded-Up Robust Features). ORB is a fusion of the FAST(Features

from Accelerated Segment Test) keypoint detector and the BRIEF(Binary Robust Independent Elementary Features) descriptor, enhanced for performance. It's known for its computational efficiency, primarily due to its use of binary descriptors, making it ideal for real-time applications. ORB employs a multi-scale approach for keypoint detection and uses the Hamming distance for faster matching. On the other hand, SURF, often viewed as a quicker alternative to SIFT(Scale-Invariant Feature Transform), uses a Hessian matrix-based approach for keypoint detection and integral images for rapid computation. While ORB's binary nature ensures speed, SURF's float-based descriptor provides a denser representation, potentially offering more distinctiveness in some scenarios. In essence, while ORB stands out in real-time scenarios due to its speed, SURF strikes a balance between robustness and computational efficiency.

To summarize, our experiments were tested on the TartanAir dataset using two different feature point extractors, ORB and SURF, and using *cv2.solvePnPRansac()* as the skeleton. However, because of the patent issue of the SURF feature extractor, after discussing with our colleagues we are not going to put the SURF feature extractor into our project, and we will mainly only discuss the performance of the ORB feature extractor on PnP problems in the following sections (although SURF is usually better because it provides more stable keypoints). The performance of the ORB+RANSAC+EPnP algorithm is shown in Fig.3.1.



(a) Trajectory

(b) Rotation in quaternion form

Figure 3.1: The performance of the ORB+RANSAC+EPnP algorithm

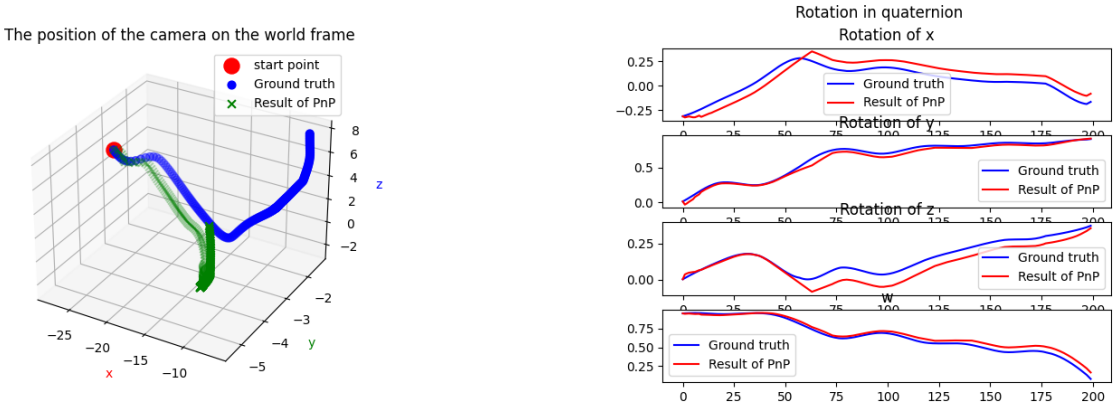
Based on the observations drawn from Fig.3.1b, it can be deduced that the efficacy of the proposed algorithm is suboptimal. Notably, there are frequent significant perturbations in the rotational estimation. Furthermore, as depicted in Fig.3.1b, the illustrated trajectory is the global trajectory, implying that any errors accrued in the initial stages can profoundly impact the ultimate outcome.

Therefore, we designed a transformation correction module to filter out overly large estimated poses, and we established separate thresholds for both translation and rotation. We first decompose the estimated pose returned by the PnP algorithm into a transformation vector and a rotation vector (this is quite simple in our project because the estimated pose returned by *cv2.solvePnPRansac()* is already a transformation vector and a rotation vector, note that the rotation here is not in the form of a quaternion but a rotation vector). We then compare the sup norm of each vector to its respective threshold. If either sup norm surpasses its threshold, we consider the estimated pose result too large which is unreasonable. We set the result to the previously optimized pose (which is logical because we assume the camera is moving slowly

and maintaining a certain continuity.) The sup norm of the vector:

$$\|\mathbf{v}\|_\infty = \max\{|v_1|, |v_2|, \ldots, |v_n|\} \tag{3.2}$$

Through testing, we determined that rotation and transformation thresholds of 1.0 and 0.8, respectively, yield optimal results. The result of the ORB+RANSAC+EPnP algorithm with the transformation correction module could be checked in Fig.3.2.

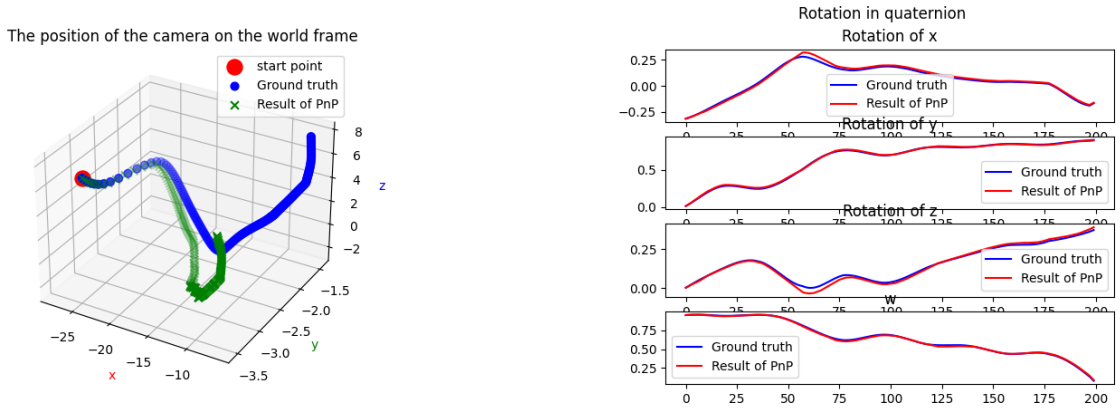


(a) Trajectory

(b) Rotation in quaternion form

Figure 3.2: The performance of the ORB+RANSAC+EPnP algorithm with the transformation correction module

It is easy to be noticed from the above figure that after adding the transformation module, the algorithm performance improved dramatically. But there are still some gaps between the GT(Ground Truth) and our result. Because the error is accumulated, we want to minimize the error at each step to improve the performance of the final algorithm. Then we speculated that the lack of filtering for feature points might be the issue, because, in the course of reading the paper, it revealed that feature point filtering is a prevalent practice in the field.[43]. Thus, we used the NNDR(Nearest Neighbor Distance Ratio) technique to implement the feature points filter module. The NNDR is used to determine the quality of matches between keypoints. For a given keypoint, the ratio is computed by taking the distance to the nearest neighbor (the best match) and dividing it by the distance to the second nearest neighbor (the second-best match). If the NNDR is over a set threshold, we will accept this match as a good correspondence for the following PnP algorithm. After using this module, the performance of our PnP algorithm is quite outstanding which is illustrated in Fig.3.3. In examining the results, it is evident that the estimated rotation curve closely aligns with the Ground Truth (GT). However, a minor discrepancy is observed in the translation component. Furthermore, it's worth noting that errors accumulated in the initial phases can culminate in a substantial deviation in the final outcome.

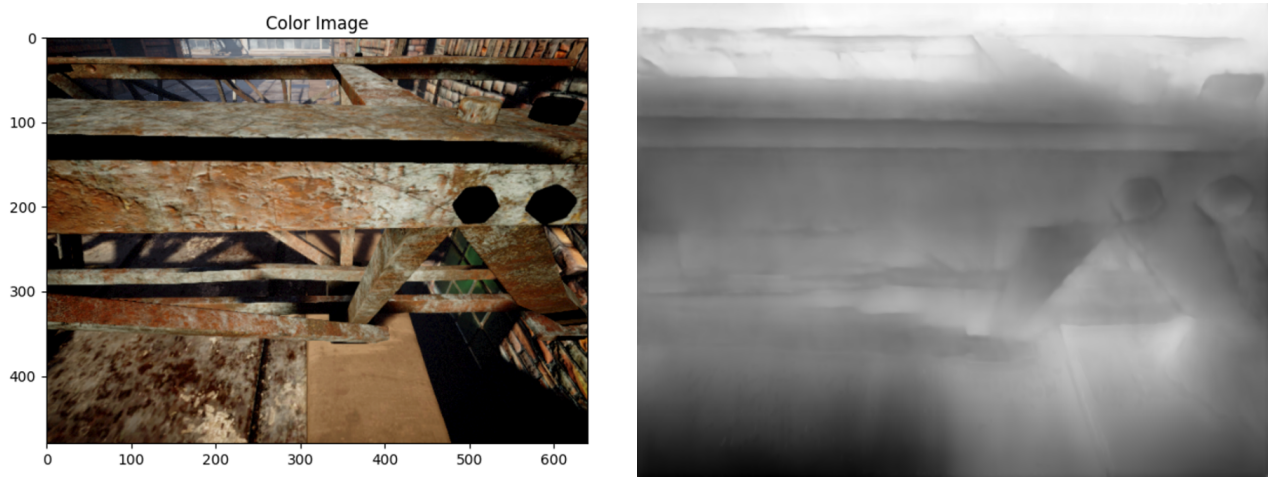


(a) Trajectory

(b) Rotation in quaternion form

Figure 3.3: The performance of the ORB+RANSAC+EPnP algorithm with the transformation correction module and feature points filter module

During the experiments, the primary sources of data are the original RGB images and their corresponding depth images, which are utilized to extract 3D feature points. The TartanAir dataset facilitates this process, as it readily offers precise depth images for every frame. However, a practical challenge arises when considering real-world applications: acquiring depth images is frequently a complex task. Fortuitously, thanks to our company's Alodepth framework, which incorporates depth estimation techniques like DPT, empowers us to directly derive the requisite depth maps from either monocular or stereo camera systems. This capability significantly enhances the versatility and applicability of our approach in diverse scenarios. The performance of Alodepth could be checked in Fig.3.4.

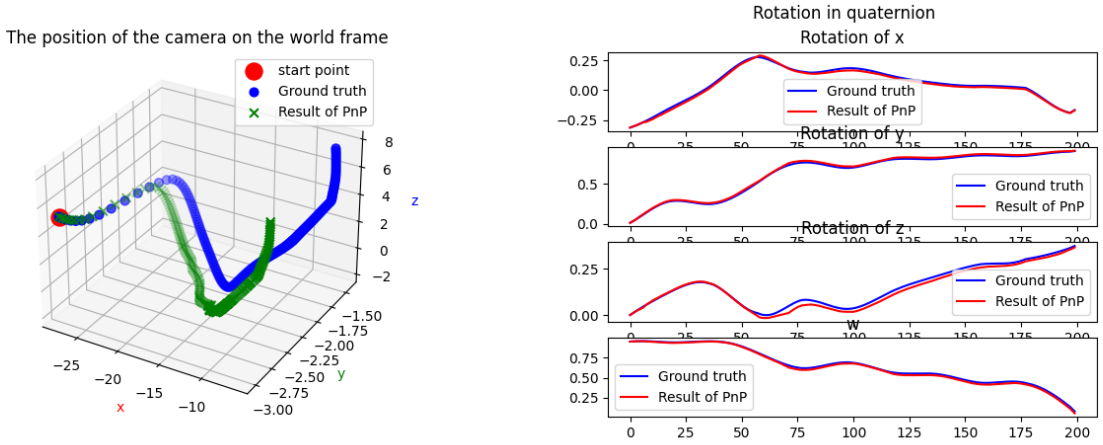


(a) Color image

(b) Alodepth image

Figure 3.4: The color image and its correspondence Alodepth image

And the performance of our PnP algorithm using Alodepth image is shown in Fig.3.5. It can be seen that our algorithm also performs consistently on Alodepth.



(a) Trajectory

(b) Rotation in quaternion form

Figure 3.5: The performance of the ORB+RANSAC+EPnP algorithm with the transformation correction module and feature points filter module using Alodepth

Overall, after adding the transformation correction module and the feature points filter module, our PnP algorithm can achieve more stable performance. However, throughout our experimental phase, we observed that while RANSAC is indispensable for ensuring robust performance, it is notably time-intensive. Considering the strong development of deep learning methods in computer vision in recent years, we want to design a deep-learning-based network to solve the PnP problem with less used time and higher accuracy.

3.2 Transformer Architecture

With the successful performance of deep learning algorithms in many fields, in recent years there are also many scholars focusing on using deep learning methods to solve PnP problems such as PnP-Net[44], EPro-PnP[45], etc.

After studying some related papers and discussing them, we finally decided to use the Transformer structure to solve this problem because of its better generalization and the ability to extract the information from the input data. The Transformer[9] architecture, originally developed for natural language processing tasks, has recently been applied to various computer vision tasks with promising results. The Transformer model is initially introduced in 2017 for handling sequential data, and it mainly consists of two main parts: the Encoder and the Decoder. The whole Transformer architecture is shown in Fig.3.6

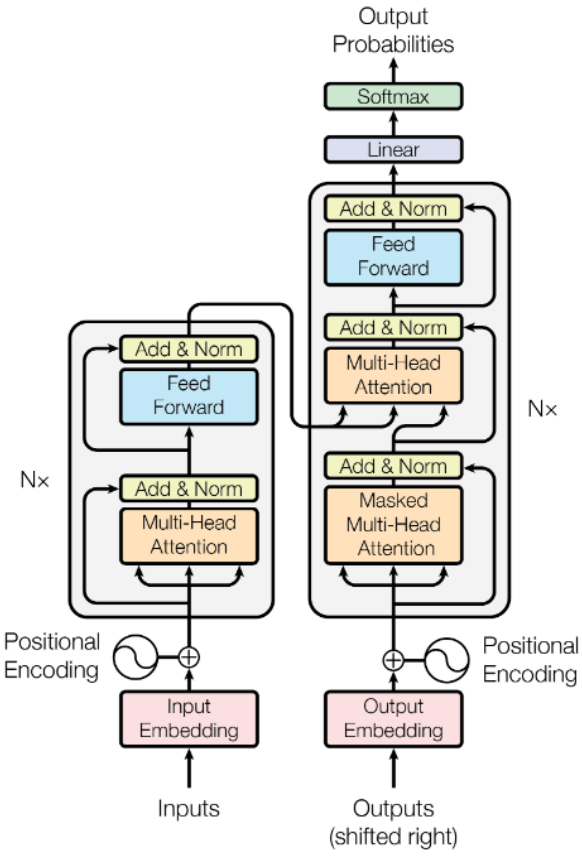


Figure 3.6: The Transformer architecture[9]

The left gray block is the Encoder block and the right one is the Decoder. The Encoder is made up of a stack of identical layers, each with two sub-layers: a multi-head self-attention mechanism, and a position-wise fully connected feed-forward network. The output from each sub-layer goes through a residual connection and is then normalized using layer normalization. The purpose of the Encoder is to process the input data and encode it into a continuous representation that holds the contextual information of the input. The Decoder also consists of a stack of identical layers. In addition to the two sub-layers found in the Encoder, the Decoder has a third sub-layer that performs multi-head attention over the output of the Encoder stack. Similar to the Encoder, each Decoder sub-layer has a residual connection around it followed by layer normalization.

Positional Encoding in the Transformer model is a crucial component that provides information about the position of each token in a sequence. Since the Transformer architecture, particularly its self-attention mechanism, does not have any inherent notion of the order of tokens, positional encoding is added to ensure that the model can account for the order or position of words in a sequence. The paper proposed using sinusoidal functions to generate positional encodings. The advantage of this method is that it can produce encodings for any position, even beyond those seen during training, allowing for the processing of sequences of varying lengths. The formula for sinusoidal positional encoding is shown:

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \\ PE_{(pos,2i+1)} &= \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right) \end{aligned} \tag{3.3}$$

where pos is the position of the token in the sequence, and i is the dimension.

The attention mechanism is another key part of the Transformer model, the simple one-head scaled dot-product attention mechanism is shown in Fig.3.7.

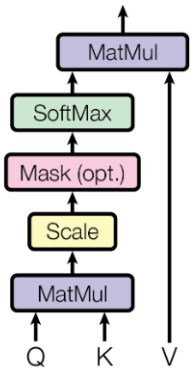


Figure 3.7: The attention mechanism[9]

The attention mechanism calculates the relevance of each input to each output. It uses a softmax function to give more weight to the most relevant inputs. The attention score between a query and a key is computed as the dot product of the query and key, divided by the square root of the dimension of the key which is used to avoid the gradient explosion problem. This score is then put through a softmax function to get the attention weights. The formula for the scaled dot-product attention is as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.4}$$

where Q is the matrix of queries, K is the matrix of keys, V is the matrix of values, $d_k$is the dimensionality of the keys. Here, the queries, keys, and values are all vectors. This mechanism allows the model to focus on different parts of the input sequence when producing each part of the output sequence, effectively allowing the model to "attend" to different parts of the input. The Transformer model also uses "Multi-Head Attention" to project the input and output multiple times with different learned linear projections, enabling it to focus on different positions and represent various types of information. The form of Multi-Head Attention is shown below:

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \\ \text{where head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{3.5}$$

where the projections are parameter matrices $W^{Q_i} \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W^{K_i} \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W^{V_i} \in \mathbb{R}^{d_{\text{model}} \times d_v}$, and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

3.3 PnP Transformer

Because of the success of the Transformer model on numerous tasks, not only textual tasks and even many computer vision tasks, and also because we found that no one has yet used the Transformer-based model on PnP problems. Therefore, we would like to design a Transformer-based model to solve the problems of classical PnP, which is easily affected by the outlier, the computation time is too long, and the robustness is not very good, and to improve the accuracy.

Our PnP Transformer model structure is illustrated in Fig.3.8

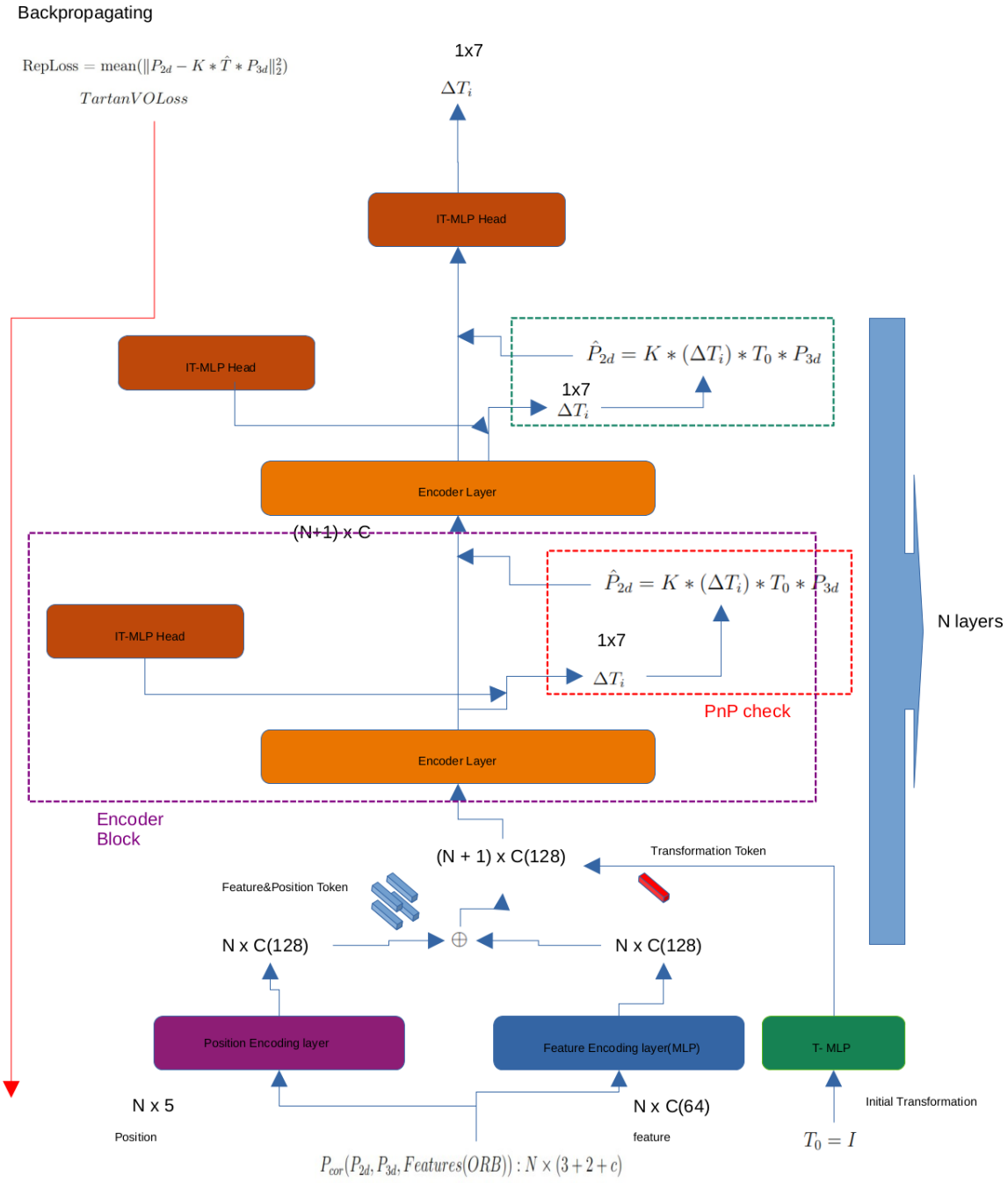


Figure 3.8: The PnP Transformer model structure

The input of the model is a batch of the keypoints(could choose after a keypoints filter or not) with their 3D positions in the current frame, 2D positions in the previous frame, and their keypoints feature descriptors respectively. We choose ORB features with 64 dimension feature descriptors for the input keypoints and therefore the input dimension is $(B, N, 69)$ where B stands for the batch size and N is the number of keypoints. Obviously, the number of keypoints varies for different images. Therefore, in order to ensure the consistency of the model, we randomly discard the keypoints that are more than N, and perform a zero-padding operation for keypoint inputs that are less than N. And additionally, we also input an identical transformation vector as the initial transformation. In the whole PnP Transformer model, the transformation is stored in a transformation vector and the rotation part is represented by

using quaternion form. Therefore, the transformation should be represented as:

$$T = (t_x, t_y, t_z, r_x, r_y, r_z, r_w) \tag{3.6}$$

And the input initial transformation is as:

$$T_0 = (0, 0, 0, 0, 0, 0, 1) \tag{3.7}$$

In the proposed architecture, the initial processing involves two primary encoding layers: the Feature Encoding and Position Encoding layers to extract the information from the model input. The former employs a simple MLP(Multilayer Perceptron Model), consisting of three linear layers. The activation functions for the initial two layers utilize the Rectified Linear Unit (ReLU) function. The Feature Encoding layer maps the feature points descriptor part of the model input into a 128-dimensional token representation. Concurrently, the Position Encoding layer, inspired by the original Transformer model's positional encoding, extracts spatial information from the input. While the canonical Transformer model was tailored for 2D positional data, our adaptation accommodates 3D spatial information, utilizing the following math formula:
Given a 3D position $pos3d$,

1. Compute the frequency term (same as in the 2D situation in the canonical Transformer model):

   $$idx_i = 10000^{\frac{2(i//2)}{num\_hid}}$$

   for i in range num_hid.

2. Expand and normalize the 3D positions:

   $$pos3demb_{b,n,d,h} = \frac{pos3d_{b,n,d}}{idx_h}$$

   where d is in 0,1,2 representing the x, y, and z coordinates, respectively.

3. Apply sinusoidal encoding (similar to 2D but extended for 3D):

   $$pos3dembc_{b,n,d,h} = \begin{cases} \sin(pos3demb_{b,n,d,h}) & \text{if } h \text{ is even} \\ \cos(pos3demb_{b,n,d,h}) & \text{if } h \text{ is odd} \end{cases}$$

4. Sum over the 3D coordinates to get the embedding:

   $$pos3dembcs_{b,n,h} = \sum_{d=0}^{2} pos3dembc_{b,n,d,h}$$

The sinusoidal encoding helps the model learn relative positions and attend to positions based on their relative distances. Then the outputs from these two layers are summed together to produce refined tokens, serving as foundational input for subsequent network operations.

As for the initial transformation input, it undergoes a process analogous to the Feature Encoding layer. Specifically, we employ a three-layer MLP to handle this input, whose overall structure and activation functions are the same as the Feature Encoding layer, except that the dimensions of the input and the dimensions of the intermediate layers are different. This MLP is tasked with transforming the initial transformation vector into what we term a "transformation token." Subsequently, this token is concatenated with the preceding feature token. The

amalgamated result serves as the input for the subsequent Encoder layer.

For the Encoder layer, we just follow the canonical Transformer model Encoder structure. This mechanism is mainly used to process spatial features from the input. It captures spatial relationships across different regions of the input using the self-attention mechanism, allowing each region to attend to other distant or nearby regions. This facilitates understanding of global and local patterns in the input. After processing through stacked Encoder layers, the resulting representation provides a comprehensive understanding of the input feature token. Meanwhile, we propose the PnP check module that transforms the output of the Encoder layer to a transformation at first and then follows the classical PnP to calculate the reprojection 2D position and embeds this 2D position information to the input of the next Encoder layer. We think by merging the classical PnP idea with the DL model, we could get a promising performance.

And finally, to extract the transformation from the output of the Encoder layer, we did not use the primitive Decoder layer but defined our own IT-MLP head to extract the useful information. The IT-MLP head starts with a flattening operation to reshape the input tensor into a one-dimensional tensor. Subsequently, the data flows through a series of linear transformations. The dimensions of these layers progressively decrease, starting from an input size and eventually outputting a tensor of size 7. Between each of these linear layers, we apply the ReLU activate function ensuring the introduction of non-linearity to the model. We believe that the IT-MLP head offers computational efficiency and simplicity compared to the Transformer Decoder. While the Decoder excels at handling sequence generation tasks and capturing long-range dependencies via its attention mechanisms, the IT-MLP head is streamlined for the specific transformation regression task which can be more memory-efficient and potential generalization.

At first, we want to use the classical reprojection loss as the loss function, the projection loss formula is shown as:

$$L_{\text{reproj}} = \sum_{i=1}^{N} \|p_i - \text{project}(P_i, \hat{R}, \hat{t}, K)\|^2 \tag{3.8}$$

where K is the camera intrinsic matrix, P_i represents a 3D point in the current frame coordinates, $\hat{R}$ is the predicted rotation matrix, $\hat{t}$ is the predicted translation vector, and p_i is the corresponding 2D point in the previous frame coordinates.

But then during the experiment, we found this loss is a bit hard to learn so we turn to an easier TartanVO loss[8]:

$$TartanVOLoss = L_p^{norm} = \|\frac{\hat{t}}{\max(|\hat{t}|, \epsilon)} - \frac{t}{\max(|t|, \epsilon)}\| + \|\hat{R} - R\| \tag{3.9}$$

where $\hat{t}$ and $\hat{R}$ represent the predicted translation and rotation, while t and R represent the ground truth respectively.

We can increase the model's ability to extract information from the input by adjusting the number of Encoder Blocks (the purple dashed part in Fig,3.8). And we also consider this part to be crucial, as our experiments were designed initially to train a network that could replace the RANSAC method to find inliers in the feature points of inputs with noise. The amount of model parameters of our PnP Transformer, when we chose 2 Encoder Blocks, is shown in Fig.3.9.

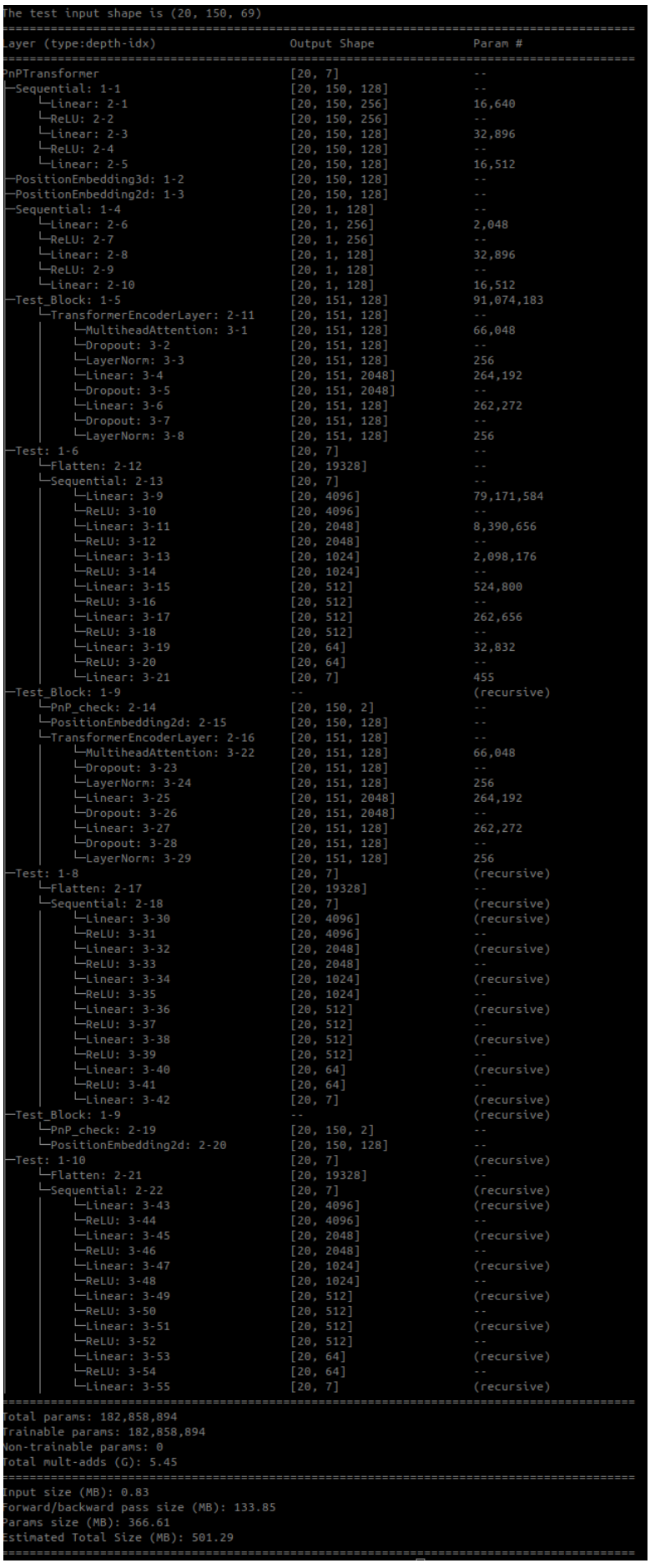
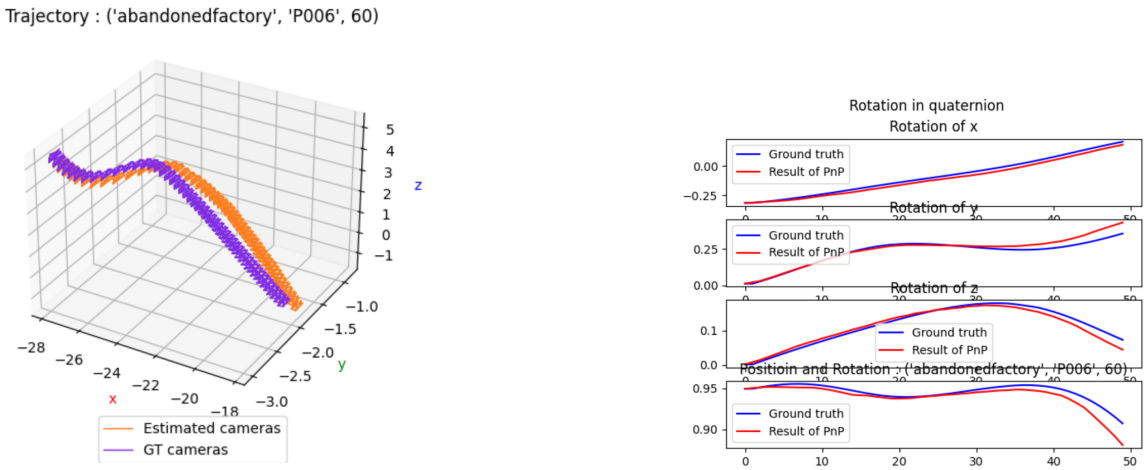


Figure 3.9: The parameters of the PnP Transformer when the number of Encoder Block = 2

3.4 Experiment

The experiments were validated on the TartanAir dataset, and the experimental device was a remote server paired with two RTX 3080 graphics cards. The model we used is basically as same as shown in Fig.3.9. And all the experiment process is stored in W&B(Weights&Biases).

Despite our extensive preliminary efforts in network design and literature review, and while we incorporated traditional PnP concepts into our PnP Transformer model, the outcomes often deviated from our initial expectations. For now, we could only get a good performance when we validate the performance on the same training trajectory the performance is shown in Fig.3.10.



(a) Trajectory

(b) Rotation in quaternion form

Figure 3.10: The performance of the PnP Transformer on the same training trajectory and validation trajectory

But when we train the model on the whole dataset, the loss is really hard to be converging as shown in Fig.3.11:

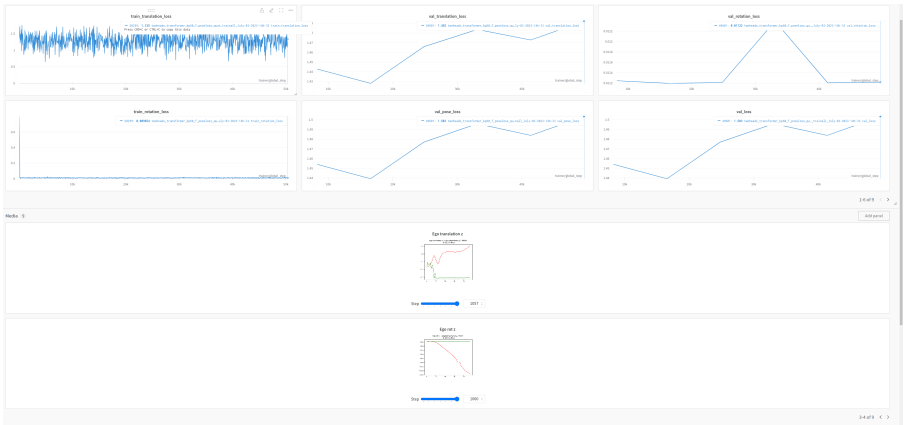
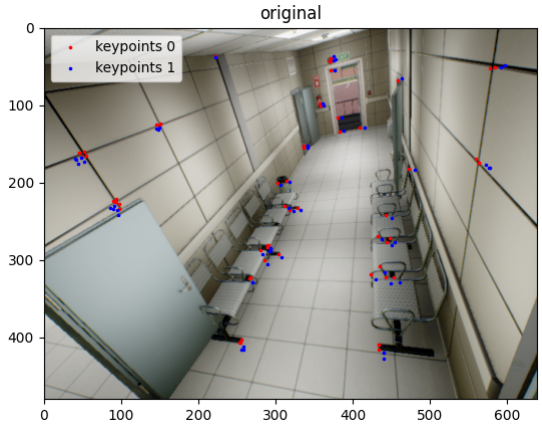


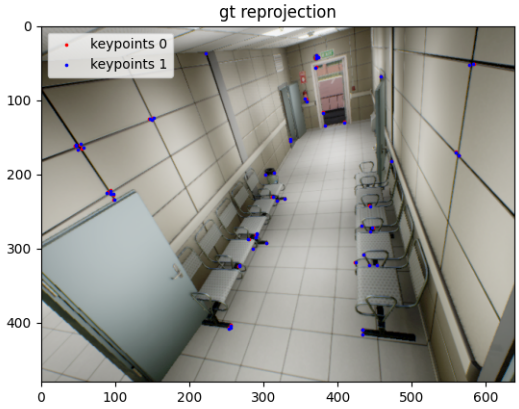
Figure 3.11: The performance of the PnP Transformer on the different training trajectory and validation trajectory

It could be found from Fig.3.11 that the rotation loss is very easy to converge to 0 but the translation loss is hard to learn which is quite opposite to what we expect. According to our

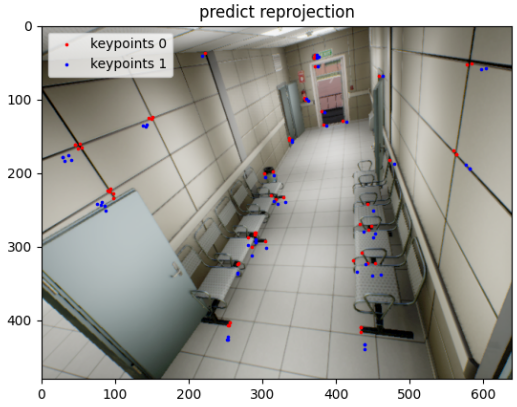
study, since rotations are stored in the form of quaternions in our training and because quaternions have their own properties, rotation should be more difficult to learn than translation. Subsequently, we utilize the forecasted pose to transform all the 3D feature points of the current frame and then reproject them onto the 2D points of the preceding frame to evaluate the efficacy of the predicted pose. The ensuing results, depicted in the subsequent Fig.4.2, reveal an intriguing observation: despite a reduction in rotation loss, the reprojected feature points deviate more from their accurate positions. This outcome is somewhat perplexing.



(a) Original keypoints



(b) GT Pose acts on keypoints



(c) Predicted Pose acts on keypoints

Figure 3.12: Check the performance of the pose on keypoints

After that, we also discussed and tried a lot of solutions, such as adding regularization terms to the model to avoid the model predicting too large transformations. as shown below:

$$Regularization = \delta * mean(\max(|translation| - 0.5, 0) + \max(|rotation| - 1.0, 0)) \quad (3.10)$$

where δ is the regularization ratio. Or use two independent MLP heads to predict rotation and translation separately, receiving inspiration from PnP-Net paper[44]. But the results are still not ideal. In the end, because of other development projects in the company, this project can only be stopped for the time being.

3.5 Conclusion and Future Work

Despite the fact that our proposed model did not fully meet our anticipated outcomes, it undeniably paves the way for a novel conceptual framework in the realm of the PnP problem

addressed through deep learning methodologies. This endeavor, while exploratory in nature, offers invaluable insights and foundational groundwork for future researchers venturing into this domain. A noteworthy observation from our experiments is that, when benchmarked against traditional PnP algorithms under identical training and testing conditions, our model exhibits superior performance. This suggests that the model is indeed capable of discerning and learning intricate patterns, even if such learning might be indicative of overfitting. Consequently, while there remains room for refinement and optimization, the preliminary findings underscore the potential of deep learning approaches in enhancing the efficacy of PnP solutions.

In light of our research and findings, I envision two primary avenues for future exploration:

- Enhancement of Keypoint Extraction and Matching: The efficacy of the PnP solution is intrinsically tied to the quality of input keypoints. If the input is replete with erroneous matches, resolving the PnP problem becomes inherently challenging. Traditional PnP methodologies employ the RANSAC technique to filter out such outliers. However, deep learning models, as currently structured, lack an analogous module. While our initial aspiration was for the PnP Transformer to autonomously discern and filter feature points, it might be pragmatic to commence with rudimentary conditions, progressively advancing towards more intricate scenarios. This iterative approach could potentially yield more robust and accurate solutions.

- Optimization of Positional Embedding Techniques: Cause I think the influence of the position information is greater than the descriptor information. And in the current model, we just follow the basic positional encoding method from Attention is All You Need paper[9]. It is originally designed for the text-based task, maybe it is better to try some other positional encoding method, especially for the space-oriented task like the proposed position encoding in LightGlue paper[6].

Chapter 4

BEV with VO

Since May, our company has pivoted its primary focus towards the development of BEV (Bird's Eye View) based projects. We offer our customers (mainly robotics companies, particularly those specializing in warehouse automation) the state-of-the-art robotic vision systems service. With the introduction of BEV systems by industry giants companies such as Tesla, we also offer our customers a vision-based (monocular or stereo camera) BEV system. In the later part of this internship, I was also mainly involved in the development related to this project. I was mainly in charge of a part of the development of the BEV and the improvement of the robustness of the BEV odometry.

4.1 BEV(Bird's Eye View)

Bird's Eye View (BEV) is a visualization technique that provides a top-down perspective, often used in automotive and robotics applications to offer a comprehensive view of the surroundings. This perspective is particularly valuable for tasks like navigation, obstacle detection, and path planning, as it presents a clear, overhead view without the obstructions that side or ground-level views might introduce.

BEVFormer[10] represents an evolution in the domain of Bird's Eye View (BEV) processing. Rooted in the foundational principles of BEV, BEVFormer leverages the transformative capabilities of transformer architectures, a paradigm shift in deep learning methodologies. This integration facilitates the intricate processing of BEV images, allowing for the extraction of nuanced spatial-temporal features. The inherent self-attention mechanisms of transformers enable BEVFormer to adeptly manage large-scale perception challenges, ensuring an efficient contextual understanding of the environment. BEVFormer can efficiently aggregate spatiotemporal features from multi-view cameras and historical BEV features. The architecture overview of BEVFormer is shown in Fig.4.1.

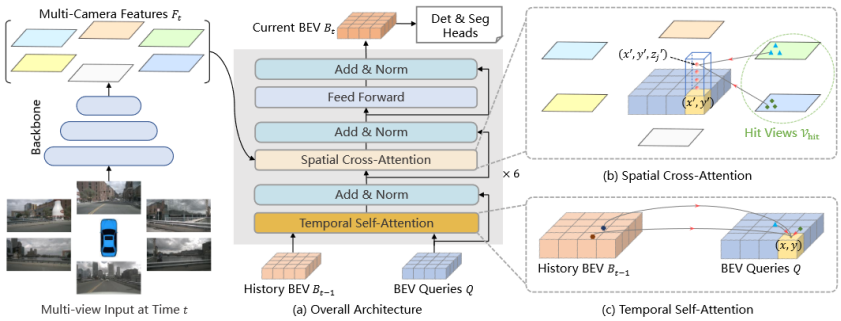
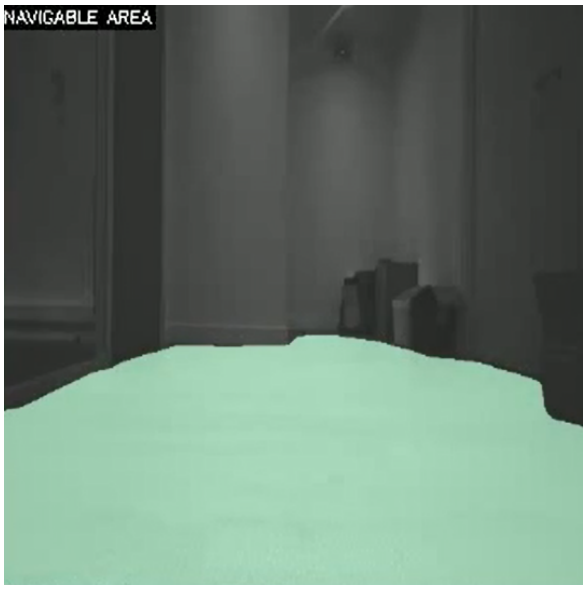


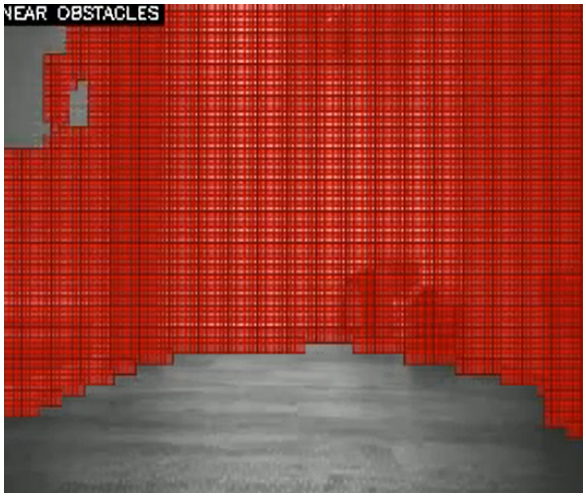
Figure 4.1: The architecture overview of BEVFormer[10]

In contrast to the BEVFormer approach that relies heavily on intricate deep learning techniques, our company's methodology for generating Bird's Eye Views adopts a more intuitive stance. Leveraging the capabilities of our proprietary *AloServing* framework, we efficiently extract depth as shown in Fig.4.2a, navigable area as shown in Fig.4.2b, and obstacles as shown in Fig.4.2c directly from images. Subsequent noise filtration processes refine this data, ensuring clarity and precision. By projecting the 3D spatial and semantic data onto a 2D plane, based on the camera's positioning, we can derive a comprehensive representation of the environment. Furthermore, heuristic algorithms are applied to the 2D obstacle points to generate convex hulls. The culmination of these processes results in the synthesis of a detailed BEV map as shown in Fig.4.2d, offering a clear and holistic view of the surroundings.

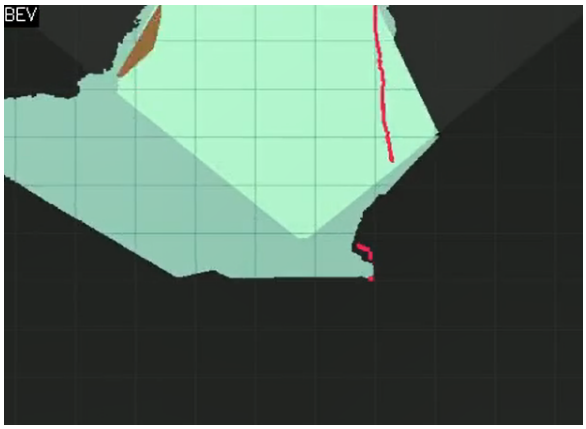
(a) Depth



(b) Navigable area



(c) Obstacles



(d) BEV

Figure 4.2: BEV example

4.2 VO Archiecture

Visual Odometry (VO) plays a pivotal role in the generation and optimization of Bird's Eye View representations. BEV provides a top-down perspective of the environment and VO, by tracking the motion of the camera relative to a static environment, offers the necessary

spatial and temporal context. This context ensures that the BEV representation is consistent and aligned over time, especially in dynamic scenarios. As vehicles or robots move through an environment, VO aids in stitching together consecutive frames to create a seamless and comprehensive BEV map. Moreover, in environments where global positioning systems might be unreliable or unavailable(like the indoor situation in which our clients meet usually), VO becomes the backbone for maintaining the accuracy and relevance of the BEV, ensuring safe and efficient navigation. Our VO structure is shown as Fig.4.3.

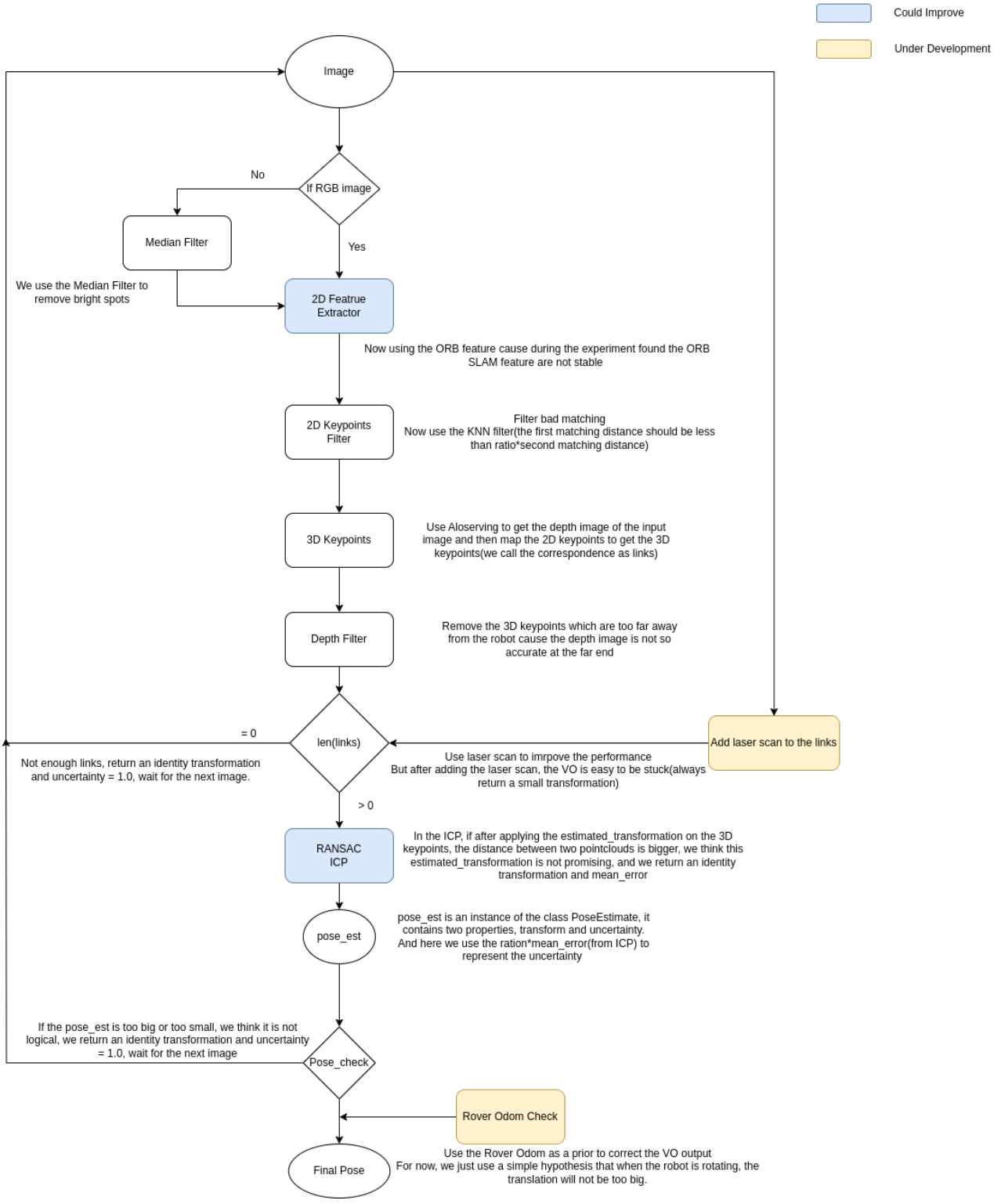


Figure 4.3: VO structure

In our VO, we mainly use the ICP method which is a widely-used algorithm for aligning two sets of points for pose estimation. ICP aims to minimize the difference between two clouds of points (a reference set and a source set) by iteratively revising the estimated transformation (rotation and translation) that aligns the source to the reference. The pseudocode is shown below:

Algorithm 3 Iterative Closest Point (ICP) Algorithm

Input:
$\mathbf{P}$, the source point set
$\mathbf{Q}$, the reference point set
T_0, the initial transformation
M, the maximum number of iterations
ϵ, the convergence threshold
Output:
T, the optimized transformation
Process:
$T \leftarrow T_0$
$error_{prev} \leftarrow \infty$
for $i = 1$ to M **do**
  for each point p in $\mathbf{P}$ **do**
    Find closest point q in $\mathbf{Q}$ to $T(p)$
    Associate p with q
  end for
  Compute transformation ΔT that minimizes $\sum_{p \in \mathbf{P}} ||\Delta T(p) - q||^2$
  Update $T \leftarrow T \circ \Delta T$
  $error \leftarrow \sum_{p \in \mathbf{P}} ||T(p) - q||^2$
  if $|error - error_{prev}| < \epsilon$ **then**
    break
  end if
  $error_{prev} \leftarrow error$
end for
Return: T =0

In our initial approach, we aimed to enhance the performance of ICP odometry by integrating laser scan point clouds with feature point clouds. It is important to note that the laser scan here is not radar-based in the traditional sense, but rather we select the depth information for a specific altitude range and then filter it to get it. However, during experimentation, we encountered challenges due to the absence of a filter for the laser scan point clouds. These unfiltered point clouds were often too dense, and some points were located at significant distances from the robot. This density and distance disparity adversely affected our translation estimation, causing the robot to remain stationary, especially in sparser environments where obstacles were far from the robot. To address this, we introduced a depth filter to reject those far-away point clouds and implemented a KM(Kuhn-Munkres) algorithm-based laser scan point cloud matcher. Despite these efforts, the results remained suboptimal. A notable limitation of the KM algorithm is its dependency on the previous point cloud, leading to a gradual reduction in the number of laser scan points until they deplete entirely when using this matcher.

Then we think, the main part is still to improve the original VO performance. Therefore, first, we add some transformation filters to remove those too less or too big, or bad estimation(after the transformation, the error between two point clouds is even bigger). And then we choose a more robust keypoint extractor SuperPoint[4]+LightGlue[6]. By using the onnx(Open Neural

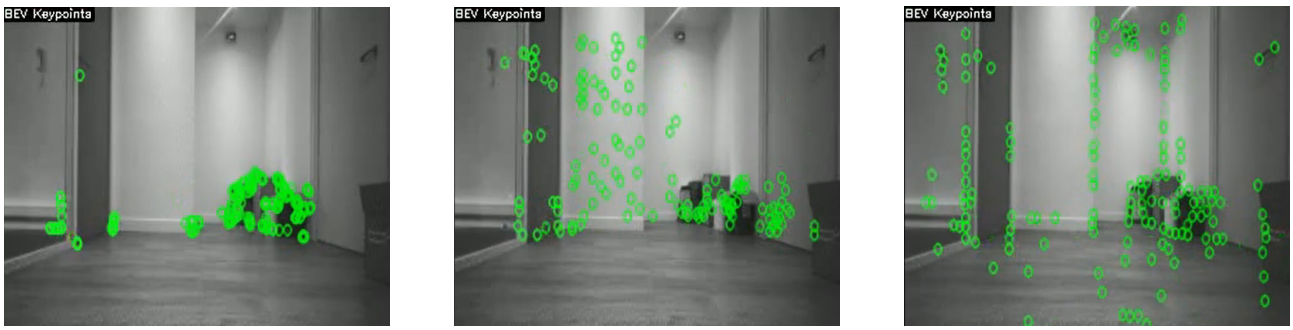
Network Exchange) model, we could get almost the same speed as the ORB feature extractor but more robust keypoints performance. During the experiment, we also tried the ORB_SLAM feature extractor[3], but the performance is still not as good as SuperPoint.

And for those robots with their own odometry, we propose a simple hypothesis that when the robot is rotating it will not translate a lot to use the odometry to correct the VO. Based on this method, we could even get better performance.

4.3 Experiment

This part of the experiment is mainly tested on datasets in the company environment and on customer datasets. But because of the confidentiality issue, I only put the company trajectory in the paper for demonstration purposes, and in fact, this VO we proposed has better performance on some customer datasets. The datasets in the company environment are mainly collected with the company's ROVER robot equipped with a stereo camera and its own wheel odometry.

From the three figures below, we could see the difference in the performance between the ORB feature, the ORB-SLAM feature, and SuperPoint+LightGlue. ORB features are recognized for their stability, but they tend to be more centralized in detection. This centralization is not favorable for pose estimation, particularly when there are inaccuracies in depth information. Such a concentration of feature points can lead to significant errors. ORB SLAM feature, on the other hand, addresses this centralization issue by selecting a specific number of feature points from each region. However, a drawback of the ORB SLAM feature is its lack of stability across consecutive frames, leading to considerable variations in detected feature points. In our research, the SuperPoint+LightGlue method emerged as the most optimal, offering both stability and a uniform distribution of feature point extraction and matching.



(a) ORB feature (b) ORB-SLAM feature (c) SuperPoint+LightGlue

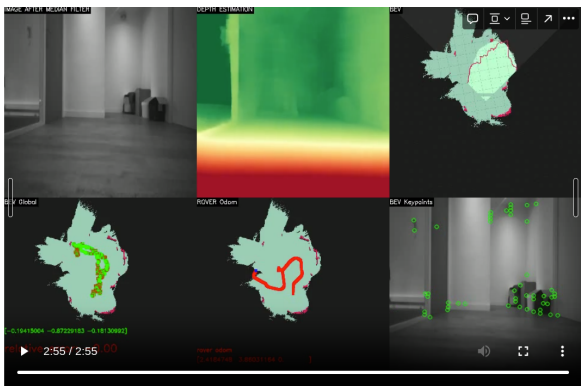
Figure 4.4: Different features performances

The operating time of each feature type is shown in the below table:

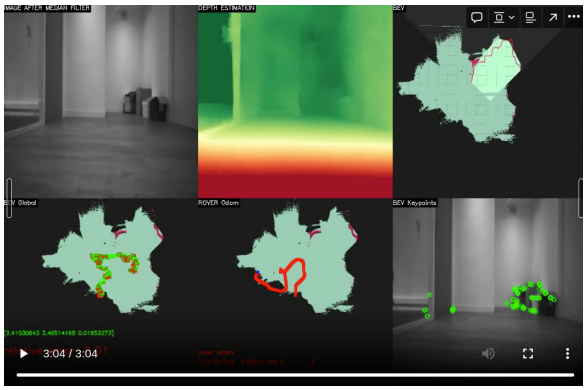
Feature Type	**Operating Time/Per Frame (s)**
ORB	0.280
ORB-SLAM	0.286
SuperPoint	0.304

Table 4.1: Operating time per frame for different feature types.

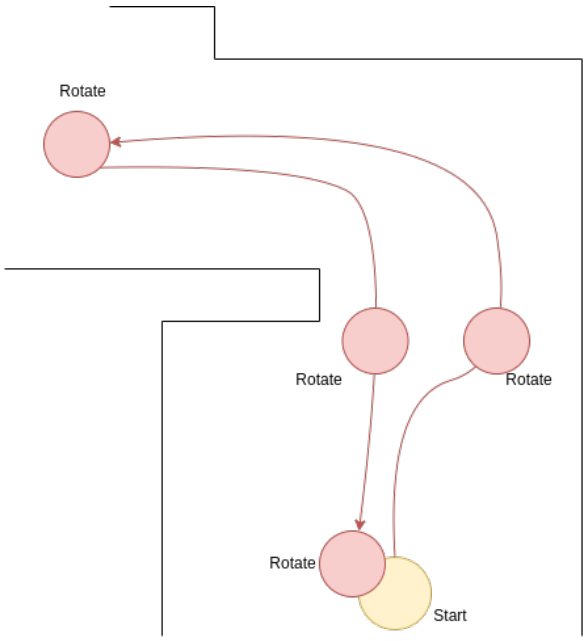
As shown in Fig.4.5, after using the SuperPoint+LightGlue, the VO performance is really close to the ground truth and much more competitive compared to the ORB feature.



(a) SuperPoint+LightGlue



(b) ORB



(c) Ground Truth

Figure 4.5: Different features performances

4.4 Conclusion and Future Work

Our exploration into Bird's Eye View (BEV) and Visual Odometry (VO) has highlighted their significant potential in the robotics and automotive sectors. While BEVFormer is the most popular method to generate the BEV, our company's unique approach showcases a different path to achieving it. Despite challenges in VO, the adoption of SuperPoint+LightGlue has proven promising. It is proved that SuperPoint+LightGlue could give better feature points not only in terms of stability but in uniform distribution as well and keep the same extracting and matching speed.

For now, I do not have a good idea about how to improve the VO over the previous modification. Maybe we could choose a much better sensor fusion module like EKF, but the precondition is that we have to calibrate the sensor.

Conclusion

In the span of the past four decades, the robotics and automation landscape has witnessed transformative advancements, with SLAM emerging as a cornerstone for understanding unknown environments. The simplicity and versatility of Visual SLAM, especially when applied to sectors like Augmented Reality and UAVs, have made it a focal point of research. The fusion of deep learning, a field already making significant strides in various domains, with visual SLAM, presents a promising frontier for robotic navigation. Noteworthy in this report is the introduction of the PnP Transformer, a novel approach to the traditional PnP problem, and the emphasis on the Bird's Eye View and Visual Odometry. Our endeavors, particularly in refining VO through techniques like SuperPoint+LightGlue, underscore the potential of these technologies in shaping the future of autonomous navigation, offering both precision and reliability in dynamic environments. The research in this paper provides some interesting ideas and some research bases for scholars who want to conduct subsequent research on deep learning in the direction of VSLAM.

Appendix A

Pseudocode of Rao-Blackwellized Particle Filter Algorithm

Algorithm 4 Rao-Blackwellized Particle Filter Algorithm

Input:
S_{t-1}, the sample set of the previous time step
z_t, the most recent laser scan
u_{t-1}, the most recent odometry measurment
Output:
S_t, the new sample set
$S_t = \{\}$
for all $s_{t-1}^{(i)} \in S_{t-1}$ **do**
$< x_{t-1}^{(i)}, w_{t-1}^{(i)}, m_{t-1}^{(i)} >= s_{t-1}^{(i)}$
//scan-matching
$x_t'^{(i)} = x_{t-1}^{(i)} \oplus u_{t-1}$
$\hat{x}_t^{(i)} = argmax_x p(x|m_{t-1}^{(i)}, z_t, x_t'^{(i)})$
if $\hat{x}_t^{(i)}$ = failure **then**
$x_t^{(i)} \sim p(x_t|x_{t-1}^{(i)}, u_{t-1})$
$w_t^{(i)} = w_{t-1}^{(i)} \cdot p(z_t|m_{t-1}^{(i)}, x_t^{(i)})$
else
//sample around the mode
for $k = 1, ..., K$ **do**
$x_k \sim x_j || x_j - \hat{x}^{(i)}| < \Delta$
end for
// compute Gaussian Proposal
$\mu_t^{(i)} = (0, 0, 0)^T$
$\eta^{(i)} = 0$
for all $x_j \in x_1, ..., x_K$ **do**
$\mu_t^{(i)} = \mu_t^{(i)} + x_j \cdot p(z_t|m_{t-1}^{(i)}, x_j) \cdot p(x_t|x_{t-1}^{(i)}, u_{t-1})$
$\eta^{(i)} = \eta^{(i)} + p(z_t|m_{t-1}^{(i)}, x_j) \cdot p(x_t|x_{t-1}^{(i)}, u_{t-1})$
end for
$\mu_t^{(i)} = \mu_t^{(i)} / \eta^{(i)}$
$\sum_t^{(i)} = 0$
for all $x_j \in x_1, ..., x_K$ **do**
$\sum_t^{(i)} = \sum_t^{(i)} + (x_j - \mu^{(i)})(x_j - \mu^{(i)})^T \cdot p(z_t|m_{t-1}^{(i)}, x_j) \cdot p(x_t|x_{t-1}^{(i)}, u_{t-1})$
end for
$\sum_t^{(i)} = \sum_t^{(i)} / \eta^{(i)}$
// sample new pose

$x_t^{(i)} \sim \mathcal{N}(\mu_t^{(i)}, \sum_t^{(i)})$
//update importance weights
$w_t^{(i)} = w_{t-1}^{(i)} \cdot \eta^{(i)}$
end if
//update map
$m_t^{(i)} = integrateScan(m_{t-1}^{(i)}, x_t^{(i)}, z_t)$
//update sample set
$S_t = S_t \cup < x_t^{(i)}, w_t^{(i)}, m_t^{(i)} >$
end for
$N_{eff} = \frac{1}{\sum_{i=1}^{N} (\tilde{w}^{(i)})^2}$
if $N_{eff} < T$ **then**
$S_t = resample(S_t)$
end if=0

Bibliography

[1] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. D. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, pp. 1309–1332, 2016.

[2] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual slam algorithms: a survey from 2010 to 2016," *IPSJ Transactions on Computer Vision and Applications*, vol. 9, pp. 1–11, 2017.

[3] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, pp. 1255–1262, 2016.

[4] D. DeTone, T. Malisiewicz, and A. Rabinovich, "Superpoint: Self-supervised interest point detection and description," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 337–33 712, 2017.

[5] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich, "Superglue: Learning feature matching with graph neural networks," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4937–4946, 2019.

[6] P. Lindenberger, P.-E. Sarlin, and M. Pollefeys, "Lightglue: Local feature matching at light speed," *arXiv preprint arXiv:2306.13643*, 2023.

[7] R. Ranftl, A. Bochkovskiy, and V. Koltun, "Vision transformers for dense prediction," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 12 179–12 188.

[8] W. Wang, Y. Hu, and S. A. Scherer, "Tartanvo: A generalizable learning-based vo," in *Conference on Robot Learning*, 2020.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[10] Z. Li, W. Wang, H. Li, E. Xie, C. Sima, T. Lu, Y. Qiao, and J. Dai, "Bevformer: Learning bird's-eye-view representation from multi-camera images via spatiotemporal transformers," in *European conference on computer vision*. Springer, 2022, pp. 1–18.

[11] A. T. Palacios, J. M. B. Cordero, M. R. Bello, E. T. Palacios, and J. MartínezGonzález, "New applications of 3d slam on risk management using unmanned aerial vehicles in the construction industry," *Drones - Applications*, 2018.

[12] R. Chatila and J.-P. Laumond, "Position referencing and consistent world modeling for mobile robots," *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2, pp. 138–145, 1985.

[13] M. Billinghurst, A. Clark, and G. A. Lee, "A survey of augmented reality," *Found. Trends Hum. Comput. Interact.*, vol. 8, pp. 73–272, 2015.

[14] J. J. Engel, J. Sturm, and D. Cremers, "Camera-based navigation of a low-cost quadrocopter," *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2815–2821, 2012.

[15] J. Dean, "1.1 the deep learning revolution and its implications for computer architecture and chip design," *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, pp. 8–14, 2019.

[16] T. Bailey and H. F. Durrant-Whyte, "Simultaneous localisation and mapping (slam) : Part ii state of the art," 2006.

[17] M. Li and A. I. Mourikis, "High-precision, consistent ekf-based visual-inertial odometry," *The International Journal of Robotics Research*, vol. 32, pp. 690 – 711, 2013.

[18] G. Grisetti, C. Stachniss, and W. Burgard, "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling," *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 2432–2437, 2005.

[19] Z. Sjanic, M. A. Skoglund, and F. K. Gustafsson, "Expectation-maximisation maximum likelihood estimation for inertial/visual slam," 2013.

[20] G. Dissanayake, S. Huang, Z. Wang, and R. Ranasinghe, "A review of recent developments in simultaneous localization and mapping," *2011 6th International Conference on Industrial and Information Systems*, pp. 477–482, 2011.

[21] J. Aulinas, Y. R. Pétillot, J. Salvi, and X. Lladó, "The slam problem: a survey," in *International Conference of the Catalan Association for Artificial Intelligence*, 2008.

[22] F. Fraundorfer and D. Scaramuzza, "Visual odometry : Part ii: Matching, robustness, optimization, and applications," *IEEE Robotics & Automation Magazine*, vol. 19, pp. 78–90, 2012.

[23] S. M. Lowry, N. Sünderhauf, P. Newman, J. J. Leonard, D. D. Cox, P. Corke, and M. Milford, "Visual place recognition: A survey," *IEEE Transactions on Robotics*, vol. 32, pp. 1–19, 2016.

[24] S. Huang and G. Dissanayake, "A critique of current developments in simultaneous localization and mapping," *International Journal of Advanced Robotic Systems*, vol. 13, 2016.

[25] D. Nistér and H. Stewénius, "A minimal solution to the generalised 3-point pose problem," *Journal of Mathematical Imaging and Vision*, vol. 27, pp. 67–79, 2004.

[26] R. Kümmerle, G. Grisetti, H. M. Strasdat, K. Konolige, and W. Burgard, "G2o: A general framework for graph optimization," *2011 IEEE International Conference on Robotics and Automation*, pp. 3607–3613, 2011.

[27] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment - a modern synthesis," in *Workshop on Vision Algorithms*, 1999.

[28] A. J. Davison, I. D. Reid, N. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, pp. 1052–1067, 2007.

[29] G. S. W. Klein and D. W. Murray, "Parallel tracking and mapping for small ar workspaces," *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 225–234, 2007.

[30] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "Orb-slam: A versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, pp. 1147–1163, 2015.

[31] C. Forster, M. Pizzoli, and D. Scaramuzza, "Svo: Fast semi-direct monocular visual odometry," *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 15–22, 2014.

[32] J. J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, pp. 611–625, 2016.

[33] J. J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *European Conference on Computer Vision*, 2014.

[34] R. A. Newcombe, S. Lovegrove, and A. J. Davison, "Dtam: Dense tracking and mapping in real-time," *2011 International Conference on Computer Vision*, pp. 2320–2327, 2011.

[35] C. Campos, R. Elvira, J. J. G. Rodr'iguez, J. M. M. Montiel, and J. D. Tardós, "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam," *IEEE Transactions on Robotics*, vol. 37, pp. 1874–1890, 2020.

[36] D. Gálvez-López and J. D. Tardós, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, pp. 1188–1197, 2012.

[37] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski, "Orb: An efficient alternative to sift or surf," *2011 International Conference on Computer Vision*, pp. 2564–2571, 2011.

[38] K. M. Yi, E. Trulls, V. Lepetit, and P. V. Fua, "Lift: Learned invariant feature transform," *ArXiv*, vol. abs/1603.09114, 2016.

[39] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *ArXiv*, vol. abs/1706.03762, 2017.

[40] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[41] M. Persson and K. Nordberg, "Lambda twist: An accurate fast robust perspective three point (p3p) solver," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 318–332.

[42] X. X. Lu, "A review of solutions for perspective-n-point problem in camera pose estimation," in *Journal of Physics: Conference Series*, vol. 1087, no. 5. IOP Publishing, 2018, p. 052009.

[43] W. Fang, Y. Zhang, B. Yu, and S. Liu, "Fpga-based orb feature extraction for real-time visual slam," in *2017 International Conference on Field Programmable Technology (ICFPT)*. IEEE, 2017, pp. 275–278.

[44] R. Sheffer and A. Wiesel, "Pnp-net: A hybrid perspective-n-point network," *arXiv preprint arXiv:2003.04626*, 2020.

[45] H. Chen, P. Wang, F. Wang, W. Tian, L. Xiong, and H. Li, "Epro-pnp: Generalized end-to-end probabilistic perspective-n-points for monocular object pose estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 2781–2790.