Absolute Localization by Mono-camera for a Vehicle in Urban Area using Street View

Li Yu

THÈSE DE DOCTORAT

de l'Université de recherche Paris Sciences et Lettres
PSL Research University

Préparée à MINES ParisTech

Localisation Absolue par Mono-caméra d'un Véhicule en Milieu Urbain via l'utilisation de Street View

Ecole doctorale n°432

SCIENCE DES METIERS DE L'INGENIEUR

Spécialité Informatique temps-réel, robotique et automatique

COMPOSITION DU JURY :

M. Patrick RIVES
INRIA Sophia Antipolis, Rapporteur

M. Paul CHECCHIN
Institut Pascal, UCA, Rapporteur

Mme. Samia BOUCHAFA
Univ. Evry Val d'Essonne, Présidente du jury

M. Fabien MOUTARDE
MINES ParisTech, Examinateur

M. Cyril JOLY
MINES ParisTech, Examinateur

M. Guillaume BRESSON
Institut VEDECOM, Examinateur

Soutenue par Li YU
le 06/04/2018

Dirigée par **Fabien MOUTARDE**
Encadrée par **Cyril JOLY**

MINES ParisTech | PSL

庄子曰：天地有大美而不言，四时有明法而不议，万物有成理而不说。

Nature has its own great beauty, yet does not speak of it; The four seasons have their clear-marked regularity, yet utter nothing of it; All things have their principles of growth, yet explain nothing of them.
Zhuangzi (369 BC - 286 BC)

Acknowledgements

I am going to present my "remerciements" in French as most people involved are French-speaking.

Ces trois années de doctorant ont étés énormément enrichissantes pour ma part, que ce soit dans la recherche scientifique ou dans ma vie. Ils m'ont permis d'aborder une thématique profondément et d'échanger avec des personnes interculturellement.

Je tiens tout d'abord à remercier mon directeur de thèse Dr Fabien Moutarde pour m'avoir permis d'effectuer cette thèse au Centre de Robotique (CAOR) et pour sa grande expérience et disponibilité. Ces travaux de thèse n'auraient pas le succès sans l'implication de mon encadrant Dr Cyril Joly, qui m'a fourni beaucoup d'aides précieuses et idées innovantes tout au long de ma thèse. Le temps passé dans V005, notre salle de discussion, sera un souvenir inoubliable dans ma vie.

Je tiens également à remercier mon responsable à l'Institut VEDECOM, Dr Guillaume Bresson, pour m'a guidé dans la recherche et m'intégré dans l'équipe, ainsi que pour ses relectures d'articles qui m'ont permis de progresser. Peu de doctorants ont eu la possibilité comme moi de mettre en relation la théorie avec les applications industrielles dans un cadre de travail excellent.

Je remercie le jury de thèse, sans qui la conclusion de ces travaux aurait été impossible. Je remercie les rapporteurs de thèse, Dr Patrick Rives et Dr Paul Checchin. Ils ont lu mon écrit en détail attentivement et formulé des critiques constructives. Je remercie également Dr Samia Bouchafa pour sa présence au jury de ma soutenance de thèse.

J'aimerai aussi remercier tous mes collègues et anciens collègues côtoyés au CAOR et à VEDECOM. Ce sont ces personnes que je vais essayer de remercier plus personnellement : Zayed Alsayed, Laurene Claussmann, Xiangjun Qian, Housem Nouira, Marc Revilloud, Xuanpeng Li, Fernando Garrido, Xavier Roynard, Mohamed-Cherif Rahal, Arthur Gaudron, Florent Chiaroni, Daniele Sportillo et Mathieu Nowakowski. Les trois ans sont passés très vite et très agréable grâce à eux.

Enfin je souhaiterais exprimer ma gratitude à China Scholarship Council (CSC) pour la bourse de Master-PhD de 5 ans en France. Je remercie mes parents et mes soeurs, vous êtes loin de France, mais je sens toujours vos soutiens et vos amours sans conditions (谢谢你们一直以来对我无私的爱与支持). Enfin, je remercie mes aims en général, pour m'avoir toujours encouragé à poursuivre mon rêve.

Abstract

In a work made at Centre de Robotique and Institut VEDECOM, we studied robust visual urban localization systems for self-driving cars. Obtaining an exact pose from a monocular camera is difficult and cannot be applied to the current autonomous cars. Rather than using approaches like Global Navigation Satellite Systems, Simultaneous Localization And Mapping, and data fusion techniques, we mainly focused on fully leveraging Geographical Information Systems (GIS) to achieve a low-cost, robust, accurate and global urban localization.

The development of public GIS's has brought us a new horizon to address the localization problem but their tremendous amount of information, such as topological, semantic, metric maps, Street Views, depth maps, 3D cadastral maps and High Definition maps, has to be well analyzed and organized to extract relevant information for self-driving cars. Our first task was to design a robotic accessible offline database from a dense public GIS, namely Google Maps, which has the advantage to propose a worldwide coverage. We make a compact topometric representation for the dynamic urban environment by extracting four useful data from the GIS, including topologies, geo-coordinates, panoramic Street Views, and associated depth maps. At the same time, an online dataset was acquired with a low-cost camera equipped on VEDECOM vehicles. In order to make spheric Street Views compatible with the online imagery, an image warping and interpolation based transformation is introduced to render rectilinear images from Street Views.

We proposed two localization methods: one is a handcrafted-feature-based computer vision approach, the other is a convolutional neural network (convnet) based learning technique. In computer vision, extracting handcrafted features is a popular way to solve the image based positioning. We take advantages of the abundant sources from Google Maps and benefit from the topometric offline data structure to build a coarse-to-fine positioning, namely a topological place recognition process and then a metric pose estimation by a graph optimization. The only input of this approach is an image sequence from a monocular camera and the database constructed from Google Maps. Moreover, it is not necessary to establish frame to frame correspondences, nor odometry estimates. The method is tested on an urban environment and demonstrates both sub-meter accuracy and robustness to viewpoint changes, illumination and occlusion. Moreover, we demonstrate that sparse Street View locations produce a significant error in the metric pose estimation phase. Thus our former framework is refined by synthesizing more artificial Street Views to compensate the sparsity of original Street Views and improve the precision.

The handcrafted-feature-based framework requires the image retrieval and graph optimization. It is hard to achieve in a real-time application. Since the GIS offers us a global scale geotagged database, it motivates us to regress global localizations from convnet features in an end-to-end manner. The previously constructed offline database is still insufficient for a convnet training. We hereby augment the originally constructed database by a thousand factor and take advantage of the transfer learning method to make our convnet regressor converge and have a good performance. In our test, the regressor can also give a global localization of an input camera image in real time.

The results obtained by the two approaches provide us insights on the comparison and connection between handcrafted feature-based and convnet based methods. After analyzing and comparing the localization performances of both methods, we also talked about some perspectives to improve the localization robustness and precision towards the GIS-aided urban localization problem.

Résumé

Dans un travail réalisé au Centre de Robotique et à l'Institut VEDECOM, nous nous sommes intéressés aux systèmes robustes de localisation visuelle en milieu urbain pour la voiture autonome. Obtenir une pose exacte à partir d'une caméra monoculaire est difficile et insuffisant en terme de précision pour la voiture autonome actuelle. Plutôt que d'utiliser des approches comme la navigation par satellites, la Cartographie et Localisation Simultanées (SLAM), et les techniques de fusion de données, nous nous sommes concentrés sur l'utilisation de Systèmes d'Information Géographiques (SIG) pour concevoir une approche fiable, précise et absolue de localisation en milieu urbain.

Le développement de SIG publics nous a apporté un nouvel horizon pour résoudre le problème de la localisation, mais ses informations, telles que les cartes topologiques, sémantiques, métriques, les Street Views, les cartes de profondeur, les cartes cadastrales 3D et les cartes en haute définition, doivent être bien analysées et organisées pour extraire les informations pertinentes pour une voiture autonome. Notre première tâche consistait à concevoir une base de données hors ligne accessible par un robot à partir d'un SIG public dense, à savoir Google Maps, qui a l'avantage d'avoir une couverture mondiale. Nous générons une représentation topométrique compacte de l'environnement urbain dynamique en extrayant quatre données utiles du SIG, y compris : les topologies, les géo-coordonnées, les Street Views panoramiques et les cartes de profondeur associées. Dans le même temps, un ensemble de données en ligne a été acquis par une caméra à bas prix équipée sur les véhicules de VEDECOM. Afin de rendre les Street View sphériques compatibles avec l'imagerie en ligne, une transformation basée sur l'interpolation d'image est introduite pour obtenir des images rectilignes à partir de Street Views.

Nous proposons deux méthodes de localisation : l'une est une approche de vision par ordinateur basée sur l'extraction de caractéristiques, l'autre est une méthode d'apprentissage basée sur les réseaux de neurones convolutionnels (convnet). En vision par ordinateur, l'extraction de caractéristiques est un moyen populaire de résoudre le positionnement à partir d'images. Nous tirons parti de Google Maps et utilisons ses données topo-métriques hors ligne pour construire un positionnement grossier à fin, à savoir un processus de reconnaissance de lieu topologique puis une estimation métrique de pose par optimisation de graphe. La seule entrée de cet algorithme est une séquence d'images provenant d'une caméra monoculaire et la base de données construite à partir de Google Maps. De plus, il n'est pas nécessaire d'établir des correspondances d'image à image, ni d'utiliser l'odométrie. La méthode a été testée en environnement urbain et démontre à la fois une précision sous-métrique et une robustesse aux changements de point de vue, à l'illumination et à l'occlusion. Aussi, les résultats montrent que les emplacements éloignés de Street Views produisent une erreur significative dans la phase d'estimation métrique. Ainsi, nous proposons de synthétiser des Street Views artificielles pour compenser la densité des Street View originales et améliorer la précision.

Cette méthode souffre malheureusement d'un temps de calcul important. Étant donné que le SIG nous offre une base de données géolocalisée à l'échelle mondiale, cela nous motive à régresser des localisations globales directement à partir d'un convnet de bout en bout. La base de données hors ligne précédemment construite

est encore insuffisante pour l'apprentissage d'un convnet. Pour compenser cela nous densifions la base d'origine d'un facteur mille et utilisons la méthode d'apprentissage par transfert pour faire converger notre régresseur convnet et avoir une bonne performance. Le régresseur permet également d'obtenir une localisation globale à partir d'une seule image et en temps réel.

Les résultats obtenus par ces deux approches nous fournissent des informations sur la comparaison et la relation entre les méthodes basées sur des caractéristiques et celles basées sur le convnet. Après avoir analysé et comparé les performances de localisation des deux méthodes, nous avons également abordé des perspectives pour améliorer la robustesse et la précision de la localisation face au problème de localisation urbaine assistée par SIG.

List of Abbreviations

AI	Artificial Intelligence
SAE	Society of Automotive Engineers
LIDAR	Light Detection and Ranging
RADAR	Radio Detection And Ranging
GNSS	Global Navigation Satellite System
V2V	Vehicle to Vehicle
V2I	Vehicle to Infrastructure
GPS	Global Position System
IMU	Inertial Measurement Unit
SLAM	Simultaneous Localization And Mapping
CNN	Convolutional Neural Network
GIS	Geographic Information System
HD	High Definition
BOW	Bag of (visual) Words
ECEF	Earth-Centered Earth-Fixed
UTM	Universal Transverse Mercator
WGS	World Geodetic System
DoF	Degree of Freedom
DoG	Difference of Gaussian
MSER	Maximally Stable Extreme Region
PnP	n Point Pose problem
GT	Ground Truth
RTK	Real Time Kinematics
OSRM	Open Source Routing Machine
Relu	Rectified Linear Unit
PCA	Principle Component Analysis
SGD	Stochastic Gradient Descent
Adam	Adaptive moment estimation
AWS	Amazon WebService
LSTM	Long Short-Term Meomery

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Background and Motivation

1.1.1 Autonomous Vehicle

From a desert self-driving car Stanley [196] winning in 2006 DARPA Grand Challenge till Google's Waymo car cruising 2.5 million miles autonomously on complex city streets in 2017, the autonomous vehicle industry has been rapidly evolving with the technology on the cutting-edge of mobile robotics, computer vision, artificial intelligence (AI), control systems and mechanical engineering. The arrival of a self-driving car is regarded as a tremendous upcoming robotics-driven social and economic change in this century. Its influence will go beyond transportation (the potential reduction of traffic collisions), beyond urban planning (the management of traffic flow) and even change our human's future in incredible fields. Autonomous car projects not only attract traditional car manufacturers, academic institutions, and giant technical companies, but also trigger a boom of start-ups oriented in machine learning, vision and perception area.

A self-driving car is able to sense its environment and navigate without any human intervention. According to the Society of Automotive Engineers' (SAE) classification, this is the highest autonomous level (Level 5). Most currently developed intelligent vehicles still stay at Level 4 (excluding some shuttles that reach Level 5 in restricted areas), namely the high automation with limited human inter-

Figure 1.1: Examples of autonomous vehicles: Stanley (left) and Waymo (right).

vention. There are still many tasks for us to accomplish before self-driving vehicles take over. A fully autonomous car must integrate advanced technologies together, including perception, trajectory planning and dynamic control systems [150]. All these modules help an autonomous car to answer its "own ultimate philosophical questions": "where am I", "where am I going" and "How am I going".

- **Where am I?** An autonomous car must understand its surrounding environment profoundly and perception makes a difference. *Perception* refers to the ability of a self-driving car to collect information from the environment. The information is a contextual understanding of the environment, such as drivable areas, obstacles' locations, traffic signs and objects' semantic classification. This task is achieved by integrating and fusing a large number of diverse sensors, like cameras, LiDARs (Light Detection and Ranging), RADARs (Radio Detection And Ranging), ultrasonics, GNSS (Global Navigation Satellite System), etc. Different sensors have different pros and cons, thus fusion techniques can make full use of the complementary merits of each sensor. After understanding the environment, the autonomous vehicle should determine its pose (position and orientation). Localization is one of the fundamental capabilities to enable the self-driving.

- **Where am I going?** To decide where to go is also a main task for self-driving vehicles. The car needs to process a series of decision-makings to achieve its goals, which is known as *path planning*. Path planning typically aims to follow a given itinerary in a dynamic environment while avoiding any obstacles and optimizing time costs and other constraints. It is usually divided into the mission planner (searching route in the path network), motion planner (setting a sequence of actions) and behavioral planner (making decisions based on road rules).

- **How am I going?** *Control* is the competence for a self-driving car to execute the planned goals by generating necessary commands to the actuators and using feedback to evaluate and improve its motion. Feedback control is a very classic controller, which processes the measured system responses and compensates any deviations from the desired purpose. Owing to the feedback, a car is able to react and adapt the planned path. The major tasks of the control are composed of path tracking (a geometric moving from a start to a goal) and trajectory tracking (a moving with the velocity and steering information).

Perception, planning and control work and interact with each other to construct the core software system in the self-driving car. Also, the communications such as Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) are also new prevalences in intelligent transportation systems. Then with the modules depicted in Figure 1.2, the car is able to interact with the dynamic environment.

1.1.2 Current Localization Systems

In this thesis, we mainly focus on the vehicle's perception and localization in urban environment. As mentioned before, *localization* refers to the problem to determine the pose of a vehicle with respect to the environment, also known as a pose estimation

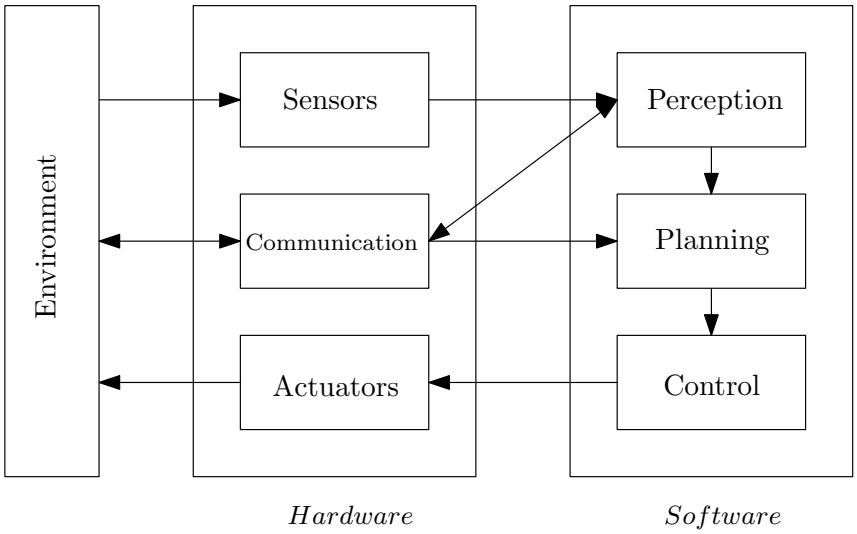


Figure 1.2: A typical self-driving car system with hardware and software settings and its interactions.

problem. Compared to other environment, there are more complexity and dynamic changes in urban environments. Obtaining an exact pose in urban environments is still difficult and unfeasible for the current self-driving car in every situation. Here we give a brief review of recent urban localization systems.

Using the GNSS, *e.g.* Galileo, GPS (Global Position System) and Beidou, and some proprioceptive sensors, like Inertial Measurement Unit (IMU), is one of the most popular ways to localize a vehicle. The precision varies from a few millimeters to ten meters depending on the signal strength, the post processing, the weather and the product quality. In urban area, a common IMU based GPS suffers a lot from the "canyon degradation" [45] and cannot reach the high accuracy required by an autonomous driving car.

The past two decades have seen a substantial progress of another localization method, namely Simultaneous Localization And Mapping (SLAM) [14, 55]. The principle of SLAM is to keep track of a mobile object and concurrently build up a map of the unknown explored environment. It estimates the pose in a map by using the observed environmental landmarks, such as visual features or laser scans, and locates new coming measurements. In most case, the localization is based on the exploring environment and it thereby is not possible to determine the absolute position directly. Generally, SLAM algorithms also leverage the vehicle's odometry to keep the consistency between where the features are predicted and where they are measured by sensors. To some extent, SLAM is more complex and versatile than it appears. Using different sensors will capture different types of features that are likely to open up totally different worlds. It ranges from monocular, stereo, omni-directional camera to 2D or 3D laser scanners. The advantages and limits of different sensors have been a major driver of new algorithms. The Bayes' filtering and optimization/smoothing are two major methods for a SLAM problem. The main issue in SLAM is the loop closure problem, which means previously visited places are observed again by sensors. A detected loop closure can correct the incremental odometry error and update the map. In the urban environment, the dynamic changes make this problem more challenging.

It has also been shown that recent learning based methods, such as random forests and convolutional neural network (convnet or CNN), are capable to regress the pose from images directly without complex feature extraction, inlier matching process and graph optimization [175]. The convnet achieves visual localization in an end-to-end manner. Thus, after an offline training procedure, these algorithms can be run in real time with an embedded computer in a vehicle. It shows a great promise in future real-time localization systems but it asks for large scale training data to realize a suitable model.

Modern autonomous vehicles generally use data fusion techniques to realize localization, that is, optimization based SLAM algorithms fusing proprioceptive and exteroceptive sensors with a previously built map. Typical used sensors include stereo cameras, LiDAR, IMU and GPS. LiDAR can provide 3D depth features without the illumination influence and camera is able to produce rich appearance features without 3D measurements. Their fusion with SLAM algorithms can make full use of the advantages of each other and construct a robust perception system for a self-driving car. Localization can finally achieve a decimetric accuracy after merging data from proprioceptive sensors and map matching. However, it is still tricky to evaluate the confidence of different data sources in the fusion mechanism and the localization quality depends a lot on sensors' price. It is not possible to use all expensive sensors for industrialization. Moreover, a mapping building process seems arduous and redundant if a world-wide map already exists.

1.1.3 Geographic Information Systems

Map-aided or map-building algorithms take advantage of local features to achieve centimeter accurate localization, particularly in the SLAM problem. However, a sparse representation of captured features in a map built from SLAM is not sufficient to model the real urban environment and dense 3D point clouds are expensive to obtain. In our daily life, the environment has been already explored and represented by the *Geographical Information Systems* (GIS), which is a temporal and spatial database to display, analyze and manage many different kinds of data related to geodetic positions on earth's surface [31]. We have seen tremendous advances in the GIS field, see Figure 1.3, in particular from some high-tech companies such as Google Maps, Apple's 3D Maps, Bing maps and so on. They can provide but do not limit to the metric, topological and semantic maps, High Definition (HD) maps[1], Street Views, depth maps, 3D urban cadastral models, etc. All these visual, topological, spatial, geographic and real-time traffic information are combined and projected according to their positions. Nowadays GIS have become more and more precise in a unified global representation due to the constant boost from companies in autonomous field. For example, more and more pixel-accurately, instance-specifically and location-precisely annotated street-level imagery datasets are released and shared in public to empower the global autonomous transportation.

[1]Strictly speaking, a HD map is also GIS with a highly precise 3D road network data features including painted lanes, traffics signals and infrastructures, 3D building models, etc.

Figure 1.3: Diverse data provided by existing GIS: (a) Metric and topological maps of Google Maps; (b) HD live maps created by HERE Maps; (c) 3D city models in Flyover project of Apple Maps; (d) Pixel-wise, instance-specific annotations for 130 million street-level images of the Mapillary Dataset; (e) A Street View from Google Maps; (f) A depth map of Street View from Google Maps.

1.1.4 Objectives and Problematics of the Thesis

The state of the art and the development of GIS have brought us a novel horizon to address the urban localization. An intuitive idea for localization is that, rather than SLAM, we do not need to build and update map if a GIS is integrated. This motivates us to re-think how to use the GIS as an alternative to simplify and improve the SLAM for industrialization. The objective of this thesis is to achieve a metric and robust urban localization system that is easy to deploy and affordable to apply in an autonomous car with the aid of GIS at a global level. That means the urban environment will be given and represented *a priori* by the current GIS. It is not a complementary fusion part like the mentioned road map matching, we only and fully leverage a given GIS to estimate a vehicle pose. It opens a new picture and also produces two major aspects to deal with.

- GIS aims to offer people various information and specific demands, such as positioning, path planning, live traffic situation and satellite images. However, all information are not directly accessible for a mobile robot. Which GIS database is adopted, how to extract useful information from the dense GIS and how to represent them in a robotic readable and efficient way should be our primary task to cope with. Hereafter we call it as an offline database construction. The extraction and representation will not only affect the localization accuracy but also determine the sensor choices and data fusion process.

  To our best knowledge, there is few research available in this field. In this thesis, Google Maps is chosen as our GIS database for its planet-scale coverage with billions of publicly accessible panoramic imagery, depth maps and other data. Then a monocular camera is determined as a single input sensor for two reasons: a) immense street-level images are available and constantly updated in Google Maps; b) a monocular camera is a low-cost but fully informative sensor, that makes our algorithms affordable and easy to be deployed in real urban localization as long as a GIS is provided.

- Rather than a classic SLAM problem supposing an origin to estimate the relative pose, our case is to localize a vehicle in a given human map with the absolute reference. The constructed GIS database enables a vehicle to reduce the time cost of map building and a visit of the environment beforehand. Our localization performance largely depends on the association between captured features and the offline database. The data association and loop closure are still tricky to deal within a SLAM problem even if all features are generated inherently by a same robot. In our case, features are from two different sources and the consistent data association requires more subtle algorithms.

1.2 Context of the Thesis

This thesis is jointly supervised by the Centre for Robotics (CAOR) at MINES ParisTech and Institut VEDECOM. CAOR has devoted its efforts to robotic autonomy, perception, multi-sensor fusion, analysis and comprehension of scenes, geometric and photometric reconstruction in the context of mobile robotics. The Institut VEDECOM, namely “Véhicule Décarboné Communicant et sa Mobilité” in French, is one

of the institutes for the Energy Transition within "Plan d'Investissement d'Avenir" organized by the French government, which is dedicated to researches and developments on carbon-free, sustainable and communicating individual mobility.

1.3 Contributions of the Thesis

As stated in above sections, the objective of this dissertation is to develop a metric global localization in urban environment with only a monocular camera and the Google Maps data. Contributions to this purpose are summarized below. We will detail the relevant contributions in the end of every chapter, and present a consolidated summary in Part V.

- **A robotic accessible database from a dense GIS is designed.** In Part II, we present a topometric representation of the dynamic urban environment from the well-known Google Maps. We extract 4 useful data from this dense GIS including topologies, geo-coordinates, panoramic Street Views and associated depth maps as stated in Chapter 5. According to different methods, we represent all these data in a compact way of the Bag of visual Words (BOW) as stated in Chapter 7, or as high-level CNN features in Chapter 9. The constructed database is then used for the online localization system.

- **A simple two-level localization system using handcrafted features is proposed.** In computer vision, extracting points of interest as handcrafted features is a popular way to solve data association in images. We fully leverage the abundant sources from Google Map and benefits from topometric data structure to build a coarse-to-fine positioning, namely a topological place recognition process and then a metric pose estimation by graph optimization. The only input of this approach is an image sequence from a monocular camera and the database constructed from Google Maps. Moreover, it is not necessary to establish frame to frame correspondences, nor odometry estimates [221]. The method is tested on an urban environment and demonstrates both sub-meter accuracy and robustness to viewpoint changes, illumination and occlusion.

  Although a considerable proportion of localization estimates achieve a 2m accuracy, the discontinuity still disturbs the robustness of the system. We show that the sparsity of Street View locations tends to induce significant errors in the metric pose estimation phase. Thus our former framework is refined with the construction of augmented Street Views database that compensates the sparsity of Street Views [220] and improves the localization precision (detailed in Chapter 7).

- **A convnet is trained with the augmented database from Google Maps to regress continuous poses from monocular images.**

  Convnets are very powerful and becoming a dominant method in computer vision. Many learning based localization methods obtain effective results in the literature. Considering the GIS offers us a global geotagged database, it motivates us to leverage this large data to train a localization convnet in an end-to-end manner. The previously constructed database for handcrafted

feature approaches is full of rich information but still insufficient for a convnet training. We hereby augment the originally constructed database by a factor of a thousand and make use of transfer learning methods to train our convnet regressor. In our test (detailed in Chapter 9), the regressor can also give a global localization of an input camera image in real time, which provides insights on the comparison and connection between handcrafted feature based and CNN based methods. After analyzing and comparing the localization performance of the two methods, we discuss promising directions towards the GIS-aided urban localization problem.

1.4 Thesis Outline

The thesis is organized in 5 main parts as follows:

- Part I: Some preliminaries about pose estimation, rigid transformation and optimization pipelines are illustrated. State-of-the-art concerning the visual localization is reviewed, specially the GIS-aided localization systems.
- Part II: We introduce some typical GIS and the extraction of useful information from Google Maps. A topometric representation is designed based on the extracted information. We also demonstrate how to rectify panoramic imagery to perspective image by image transformation. Due to the discrete distribution and the limited number of GIS dataset, image synthesis is used to augment the database.
- Part III: This part describes our first localization system based on handcrafted features. After analyzing its accuracy and robustness, a refined algorithm is proposed to improve the accuracy and smoothness of localization by augmenting the set of geo-tagged images using image synthesis.
- Part IV: A convnet based pose regressor algorithm is proposed. We show how to largely augment proper training data by image synthesis from existing GIS data and how to benefit from the power of transfer learning to make the model converge. Then, a comparison of handcrafted and convnet methods is discussed. We also give an overall summary of the visual urban localization and some future perspectives on the GIS -aided localization systems.

Part I

State of the Art

Chapter 2

Preliminaries

In this chapter, we introduce some preliminary geometric notions used in the remainder of this thesis. Visual localization studies the geometric relationship between a 3D scene and the images captured from a moving camera, which is the interplay between two fundamental transformations: the rigid body motion which models the camera motion, and the perspective projection that involves in image formation. We first present the coordinates systems used throughout the thesis. Then some foundations of rigid transformation, image formation and projective geometry are summarized.

2.1 Coordinate Systems

We can use different coordinate systems to present the localization. In autonomous mobility, two common used systems are the geodetic coordinate system and the Cartesian coordinate system, as depicted in Figure 2.1.

A geodetic coordinate system, is a coordinate system in geography that represents every absolute location on the earth by a set of geodetic datum as *latitude*, *longitude* and *elevation*. Latitude of a location is the angle between the equatorial plane and the straight line linking this location to the center of the earth. Longitude of a location is the angle east or west of a referenced meridian to a meridian passing

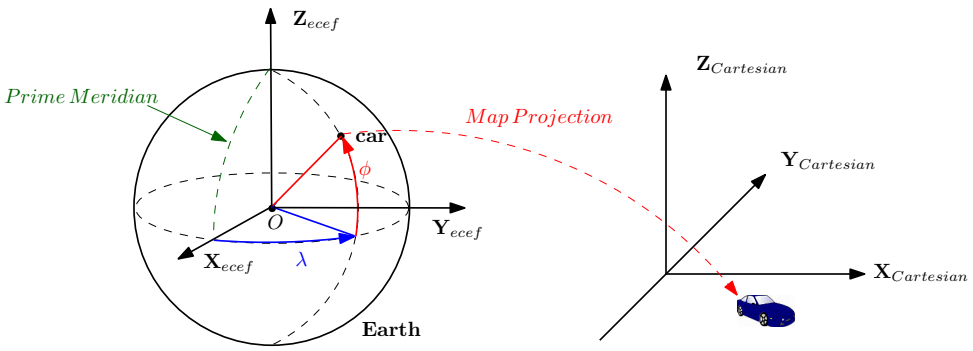


Figure 2.1: Illustration of coordinates: the geodetic coordinates on left and the local-level Cartesian coordinates obtained from the map projection on right. Latitude and Longitude are denoted by ϕ and λ respectively with the aid of the earth-centered earth-fixed (ECEF) coordinates.

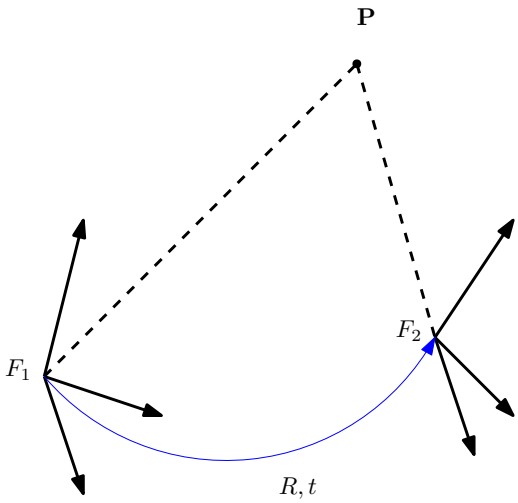


Figure 2.2: The rigid transformation between frame F_1 and F_2.

through this location. According to the datum system, the elevation is measured as a dynamic height relative to a known reference height. In the World Geodetic System (WGS84) [83], a default datum in the GPS, the elevation is measured from the mean sea level. We often use the combination of latitude and longitude to fix a vehicle's global position on the earth face.

In the driving scenario, Cartesian coordinates are often used for the reason of mathematical simplification and road-following convenience. In the Cartesian frame, we introduce x, y and z as the position coordinates. The origin of the frame is a reference to be set freely. We can convert the geodetic coordinates to the Cartesian coordinates by a map projection, like the Universal Transverse Mercator (UTM) and Lambert conformal conic projection [70, 83].

2.2 Geometry Notions

2.2.1 Rigid Transformation

A rigid transformation of a vector space includes translations, rotations, reflections, or their combination, which preserves distances between every pair of points. Rigid transformations of n-dimensional space form the basis of Euclidean geometry. We call the set of all n-dimensional rigid transformations as the Euclidean group denoted as $\mathbb{E}(n)$. A rigid transformation group excluding reflections, that is to say transformation preserves both distances and orientations, is called Special Euclidean space denoted as $\mathbb{SE}(n)$.

For two orthonormal frames F_1 and F_2 in the 3-dimensional Euclidean space, the rigid transformation from F_1 to F_2 can be represented by the homogenous matrix $\mathbf{T} \in \mathbb{SE}(3) \subset \mathbb{R}^{4\times4}$ such as,

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \tag{2.1}$$

where $\mathbf{R} \in \mathbb{SO}(3) \subset \mathbb{R}^{3\times3}$ is a rotation matrix in the Special Orthogonal group and $\mathbf{t} \in \mathbb{R}^3$ is a translation vector.

Normally this matrix also defines the transformation between two frames. For a 3D point $\mathbf{P}_1$ in frame F_1 as $\mathbf{P}_1 = [x, y, z]^T \in \mathbb{R}^3$, it can be transferred into frame F_2 by the transformation matrix $\mathbf{T}$:

$$\overline{\mathbf{P}_2} = \mathbf{T}\overline{\mathbf{P}_1} \text{ or } \mathbf{P}_2 = \mathbf{R}\mathbf{P}_1 + \mathbf{t} \tag{2.2}$$

where $\overline{\mathbf{P}_1} = [x, y, z, 1]^T$ is the homogeneous coordinates of the point $\mathbf{P}_1$ for the linear transformation. In the Special Orthogonal group, the rotation matrix has following property:

$$\mathbf{R}^T\mathbf{R} = \mathbf{R}^{-1}\mathbf{R} = \mathbf{I} \quad \det(\mathbf{R}) = 1 \tag{2.3}$$

As such, the inverse transformation matrix can be represented as:

$$\mathbf{T}^{-1} = \begin{bmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{t} \\ 0 & 1 \end{bmatrix} \tag{2.4}$$

2.2.2 Pose Representations

Pose is composed by rotation and translation. As we can see, the transformation matrix above with 16 parameters is not a compact representation. Since the rotational motion in $\mathbb{SO}(3)$ can be represented by various ways, such as rotation matrix, axis-angle, Euler angles, quaternions, we will have different pose representations as well [15, 127].

The axis-angle method parameterizes a rotation by a unit vector $\mathbf{n}$ indicating the direction of a rotation axis, and an angle θ describing the magnitude of the rotation about the axis. Rodrigues' formula is frequently used to convert the axis-angle to the rotation matrix, as stated in Equation 2.5:

$$\mathbf{R} = \cos\theta\mathbf{I} + (1 - \cos\theta)\mathbf{n}\mathbf{n}^T + \sin\theta[\mathbf{n}]_\times \tag{2.5}$$

where $[\mathbf{n}]_\times$ is a skew symmetric matrix of a vector $\mathbf{n}$ as:

$$[\mathbf{n}]_\times = \begin{bmatrix} 0 & -n_z & n_y \\ n_z & 0 & -n_x \\ -n_y & n_x & 0 \end{bmatrix} \tag{2.6}$$

Then the transformation can be presented by only 6 parameters by adding 3 elements. Actually, Rodrigues' formula is a compact form to express the rotational relationships between the Lie group and Lie algebra as will be exposed later in this section [186].

Both rotation matrix and axis-angle representation are not intuitive. Instead, Euler angles is a familiar way to describe the orientation of a rigid body by three angles. A well known convention is the Tait-Bryan angles that introduces yaw, pitch and roll for a moving body. The disadvantage of this representation is the gimbal lock problem where a singularity (losing one degree of freedom) occurs when pitch approaches $90°$ or $-90°$. This situation appears to be rare in an autonomous car as the car usually does not pitch up to these angles.

Compared with Euler angles, a unit quaternion $\mathbf{q} = \{q_0, q_1, q_2, q_3\}$ with $\|\mathbf{q}\| = 1$ is used to represent rotation and avoid the singularity. Since it is compact with only 4 parameters, this representation is widely used in the pose estimation. Conversion between different representations is possible excluding the singular case.

In the pose estimation problem like SLAM, we often construct an optimization model to estimate the optimal $\mathbf{R}$ and $\mathbf{t}$. However, the inner properties as depicted in Equation 2.3 make it difficult to solve the model. Thus Lie theory is used to transfer the pose estimation as an optimization problem without constraints.

Consider a continuously moving rigid body with a twist motion vector $\boldsymbol{\xi} \in \mathbb{R}^6$ with instantaneous speed (assuming in a timestamp of Δt) in translation $\mathbf{v} = [v_x, v_y, v_z]^T$ and in rotation $\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z]^T$.

As we known, $\mathbb{SO}(3)$ and $\mathbb{SE}(3)$ are Lie groups. Every Lie group has an associated Lie algebra, which is the tangent space around the identity element of the group. The Lie algebra is a vector space generated by differentiating the group transformations along the chosen directions in the space at the identity transformation. Such that, we have

$$\boldsymbol{\xi} = \int_0^{\Delta t} (\mathbf{v}, \boldsymbol{\omega}) dt \in \mathfrak{se}(3) \tag{2.7}$$

The exponential map from $\mathfrak{se}(3)$ to $\mathbb{SE}(3)$ is the exponential matrix. Thus the pose related to the vector $\boldsymbol{\xi}$ can be represented as follows with the Taylor expansion:

$$\mathbf{T}(\boldsymbol{\xi}) = \exp([\boldsymbol{\xi}]_\wedge) = \sum_{i=0}^{\infty} \frac{1}{i!} ([\boldsymbol{\xi}]_\wedge) \tag{2.8}$$

where the operation $[\cdot]_\wedge$ is defined as:

$$[\boldsymbol{\xi}]_\wedge = \begin{bmatrix} [\boldsymbol{\omega}]_\times & \mathbf{v} \\ 0 & 0 \end{bmatrix} \tag{2.9}$$

We summarize the above equations and their relationships in the Figure 2.3. We will use these representations in the remainder of the thesis and they are also helpful to explain the state-of-the-art algorithms.

2.3 Camera Geometry

2.3.1 Perspective Projection

In this section, we discuss the process of the image formation in terms of geometry. The ideal perspective projection by a pinhole model has been well studied [75]. What matters in the thesis is to understand the geometry and projection process in three coordinate systems: the world coordinate system (w) , the camera coordinate system (c) and the image plane (i), see in Figure 2.4. Instead of giving a rigorous introduction, here we only summarize the overall geometric relationships.

Consider a generic point in the world frame as $\mathbf{P}^w = [X^w, Y^w, Z^w]^T$ and its corresponding coordinates in the camera frame as $\mathbf{P}^c = [X^c, Y^c, Z^c]^T$ and in the image plane as $\mathbf{p}^i = [x^i, y^i]^T$. The geometry of the image formation model depends on the rigid transformation between the camera and world frame (also known as *extrinsic* parameters) and the camera *intrinsic* parameters, as illustrated by the

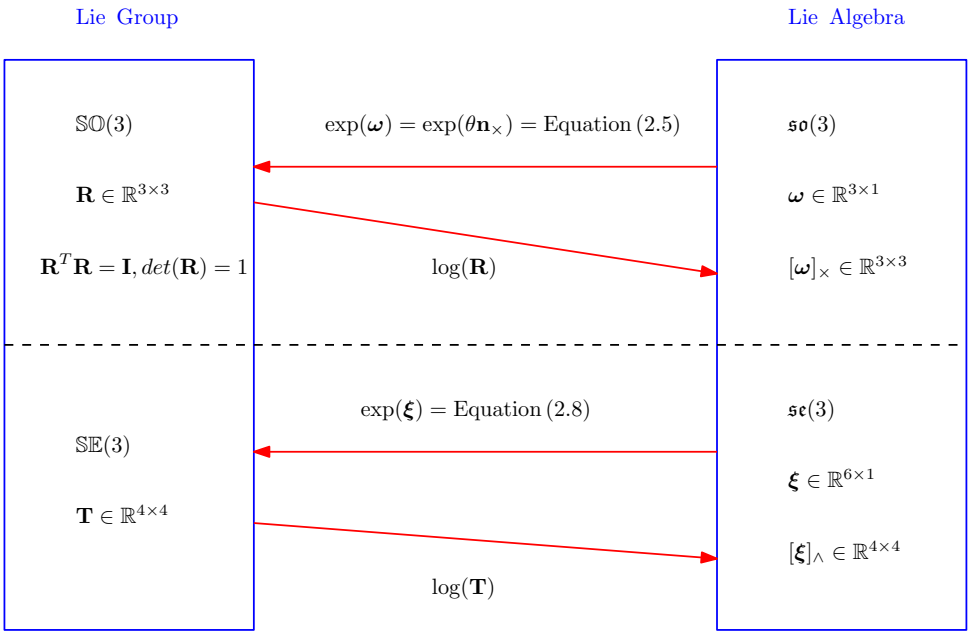


Figure 2.3: The relationships among $\mathbb{SO}(3)$, $\mathbb{SE}(3)$, $\mathfrak{so}(3)$ and $\mathfrak{se}(3)$.

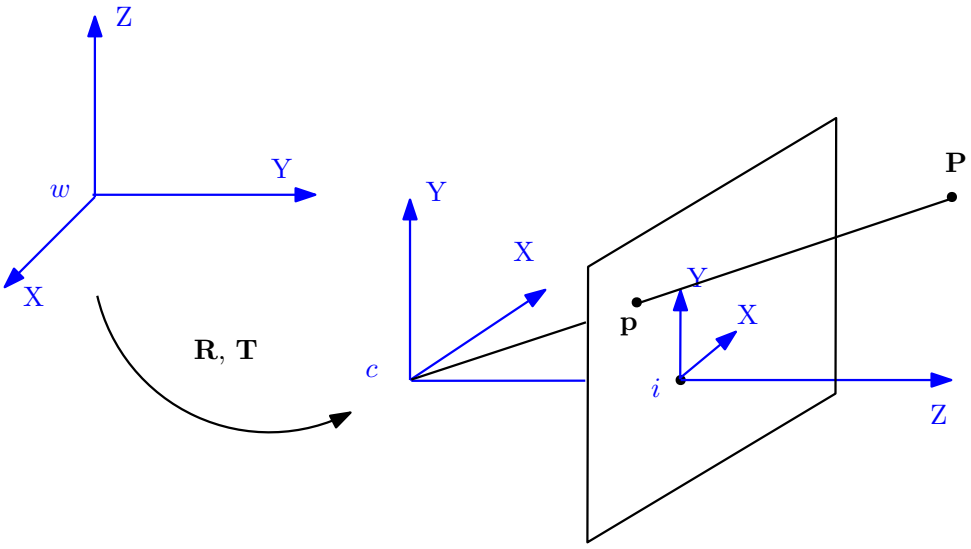


Figure 2.4: Illustration of the camera geometry: w is the world coordinate system, c is the camera coordinate system and i is the image plane.

following equation:

$$\lambda \begin{bmatrix} x^i \\ y^i \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X^w \\ Y^w \\ Z^w \\ 1 \end{bmatrix} \tag{2.10}$$

where the parameters f_x, f_y, s ,u_0 and v_0 are intrinsic parameters that can be obtained by well-developed calibration algorithms [24, 172]. (f_x, f_y) are two scalars of focal length along the two image axes, (u_0, v_0) are the coordinates of the principal point, and s describes the skewness of the two image axes. We call $\mathbf{R}$ and $\mathbf{t}$ extrinsic parameters, which present the coordinate transformation between the camera and the world coordinates. $\lambda \in \mathbb{R}_+$ is a positive (non-zero) scale factor and the depth of the point along the optical axis Z^c. The equation above can be also expressed in the homogenous matrix form[1].

$$\boldsymbol{\lambda}\overline{\mathbf{p^i}} = \mathbf{M}\overline{\mathbf{P^w}} = \mathbf{K}[\mathbf{I_{3\times 3}}, \mathbf{0_{3\times 1}}]\overline{\mathbf{P^c}} = \mathbf{K}[\mathbf{I_{3\times 3}}, \mathbf{0_{3\times 1}}]\mathbf{T}\overline{\mathbf{P^w}} \tag{2.11}$$

where $\mathbf{M}$ is the perspective projection matrix, $\mathbf{K}$ and $\mathbf{T}$ describe the matrix of the camera intrinsic and extrinsic parameters respectively.

Notice that the projection from the camera frame to the pixel image plane has two stages for convenience: first the matrix $[\mathbf{I}_{3\times 3}, \mathbf{0}_{3\times 1}]$ is used to introduce a normalized image plane located at the focal length $f = 1$, where the pinhole is mapped to the origin of the image plane; then an additional transformation is realized from the normalized coordinates to the pixel coordinate system.

2.3.2 Undistorsion

The pinhole camera is not sufficient to model the real image formation process. Real cameras use curved lenses that are fixed with offset to form an image. Image distortion tend to be introduced in the transformation with such cameras. Distortion will change both the shape and size of 3D objects. Therefore, it is necessary to analyze camera images and undo the distorsion.

There are two main types of distortion. One is the radial distortion, which is caused by the curved lenses where light rays will bend more or less at edges of lenses. In this way, the lines or objects in images appear more or less curved than they actually are. The other is the tangential distortion that makes an image look tilted or the objects seem further or closer. It occurs when a camera's lens is not fixed perfectly parallel to the image plane.

Radial distortion can be well measured by a n-order polynomial function as follows [211].

$$\overline{\mathbf{p}}_{corrected} = (1 + k_1 r^2 + k_2 r^4 + k_3 r^6 + ...)\overline{\mathbf{p}} \tag{2.12}$$

where r is the pixel distance with respect to the projection center (see Figure 2.5), k_i are the distorsion coefficients and $\overline{\mathbf{p}}_{corrected}$ are the corrected positions of raw distorted pixels $\overline{\mathbf{p}}$. All these parameters can be obtained during the camera calibration [172].

[1]Note that in the remainder of this thesis, the homogenous form is denoted by $\overline{(\cdot)}$.

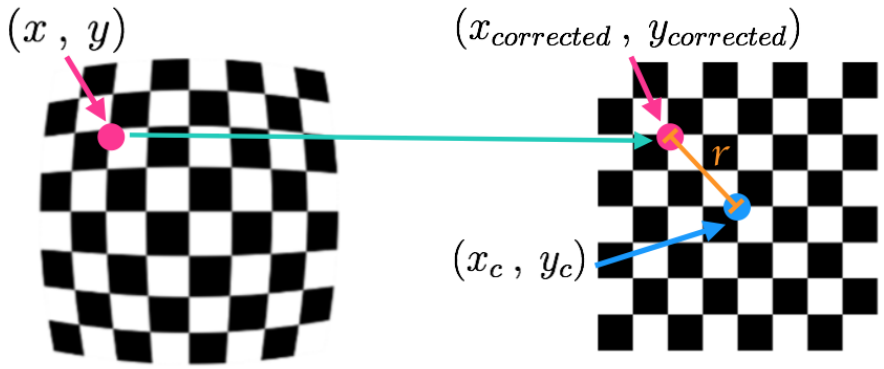


Figure 2.5: Points in a distorted and undistorted (corrected) image.

The tangential distortion can be modeled in a similar way. Here we will not dive into details for a better readability. In practice, we choose OpenCV five distortion coefficients $[k_1, k_2, p_1, p_2, k_3]$ [24] to correct both radial and tangential distortion as a pre-processing for all images obtained online from the camera.

2.4 Conclusion

In this chapter, we briefly introduced three major preliminaries: basic coordinate systems, the pose representations, rigid transformations and image formation. The global visual localization in urban area relies on both the global geodetic and local Cartesian coordinate systems. Pose transformation and image formation are the basic theory to understand our algorithms. All the notions in this chapter will be employed frequently in the continuation of the thesis.

Chapter 3

Review of Visual Localization

3.1 Introduction

Keeping in mind the previous foundations of the pose estimation, we will talk about the state-of-the-art in visual localization. Accurate localization is a prerequisite for most autonomous navigation and intelligent driving systems. The problem of visual localization is an old problem in both mobile robotics and computer vision. Cameras are low cost sensors and easy to use making them the focus of an active research community. However, visual localization is a very versatile problem, which constantly evolves, taking new techniques from various research fields, ranging from probability theory, computer vision, artificial neural network to even biology[1].

According to different criteria, we can divide current visual localization into several classes. We can distinguish different methods by the sensors they use, the precision they obtain, or simply by the environments they are applied in Figure 3.1 illustrates a taxonomy of the visual localization problem, without claiming to involve all.

In this chapter, we mainly focus on the mathematical algorithms behind the subclass of place recognition and metric visual localization. Due to the context of the thesis, the GIS-aided visual localization approaches are also presented particularly. We give a global introduction of relevant methods in visual localization and propose some inspiration to realize the objective of the thesis.

3.2 Place Recognition

Place recognition, also known as *appearance-based localization* [224], is a very common problem in visual localization. It asks the mobile object to recognize known places in the environment. The known places can have been perceived by the same camera in the past or be given by a set of geotagged images. In general, the precision of place recognition remains at a topological level. Nowadays, the photo sharing communities, like Fliker [153] and Instagram, generates thousands of geotagged images everyday and the Google Street View covers nearly every urban street in the world [223].

[1]Moser *et al.* won the 2014 Nobel Prize in Physiology and Medicine for the discovery of place and positioning grid cells.

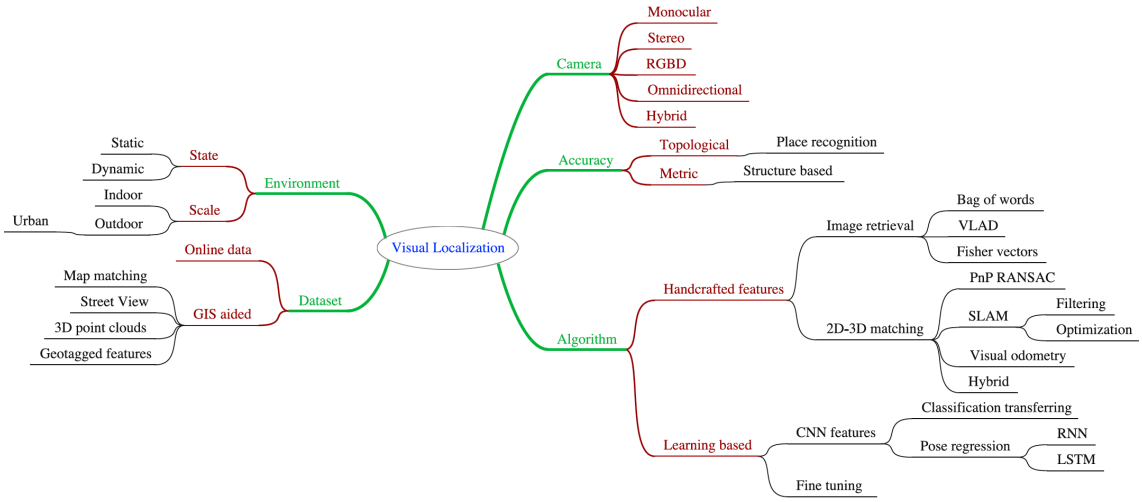


Figure 3.1: Taxonomy of visual localization problem.

As a consequence, the place recognition, as a method to map and organize photos on the globe, is undergoing intense study in computer vision. Major challenges in this area are caused by:

- Temporal changes due to lighting, weather, season, structures and moving objects [95, 187];
- Spatial changes [103] due to viewpoints or occlusions;
- Perceptual aliasing by inter or intra repetitions (*e.g.* chain stores with uniform decorations) structures, also known as saturations of features [202].

As illustrated in the taxonomy, we can solve place recognition by two broad ways: handcrafted-feature-based and leaning based algorithms. These are two different methods for representing images, but the typical pipeline in place recognition in the two cases follows the image retrieval idea [205] as shown in Figure 3.2. That is to say, we define an "image representation extractor" f to encode a *query*[2] image $\mathbf{Q}$ as $f(\mathbf{Q})$. The known places are represented by a set of geotagged images as $\mathbf{D} = [\mathbf{V}_1, \mathbf{V}_2, \cdots, \mathbf{V}_i]$. All query and database images are described by the same extractor f in the image representation space. Then we use a similarity function s to find the best matching database image for a given query image and transfer its geotag to localize the query image.

$$i^* = \underset{i \in 1 \cdots N}{\operatorname{argmax}}\, s(f(\mathbf{Q}), f(\mathbf{V}_i)) \tag{3.1}$$

The performance of this pipeline primarily depends on the extractor and the similarity function. The image retrieval started in the early 1990s in the computer vision society and was inspired from the text retrieval. Till now many different extractors are proposed, ranging from image texture, color, object, scene, various local or global descriptors to CNN features (*cf.* Figure 3.3). Before 2012, the bag of words (BoW) model [44, 179] with SIFT [125] was a very successful and predominantly studied feature based method. In recent years, it seems to be overtaken by the CNN methods, especially when dealing with a massive scale of dataset [231].

[2]Normally, a "query" image is captured online with respect to the offline image in a database.

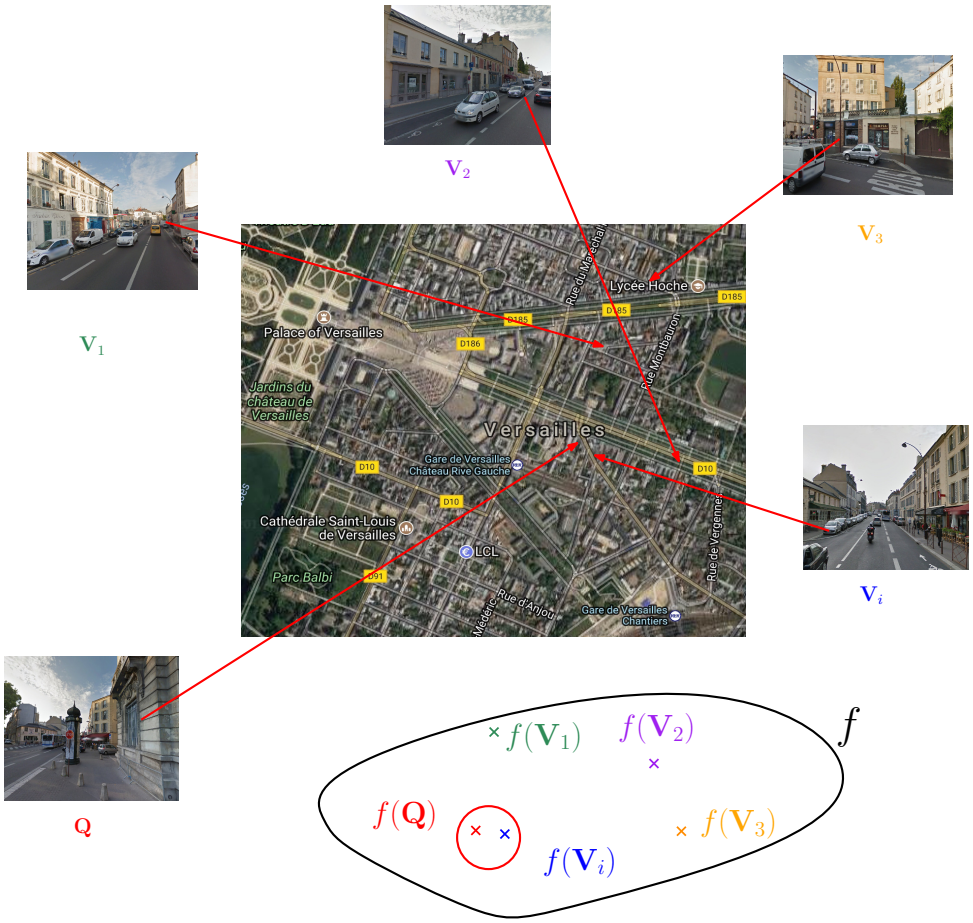


Figure 3.2: The image retrieval problem: Given a number of dataset images from visited places and a query image, we use an encoding function f to map all images in a new location space and decide the query image's position.

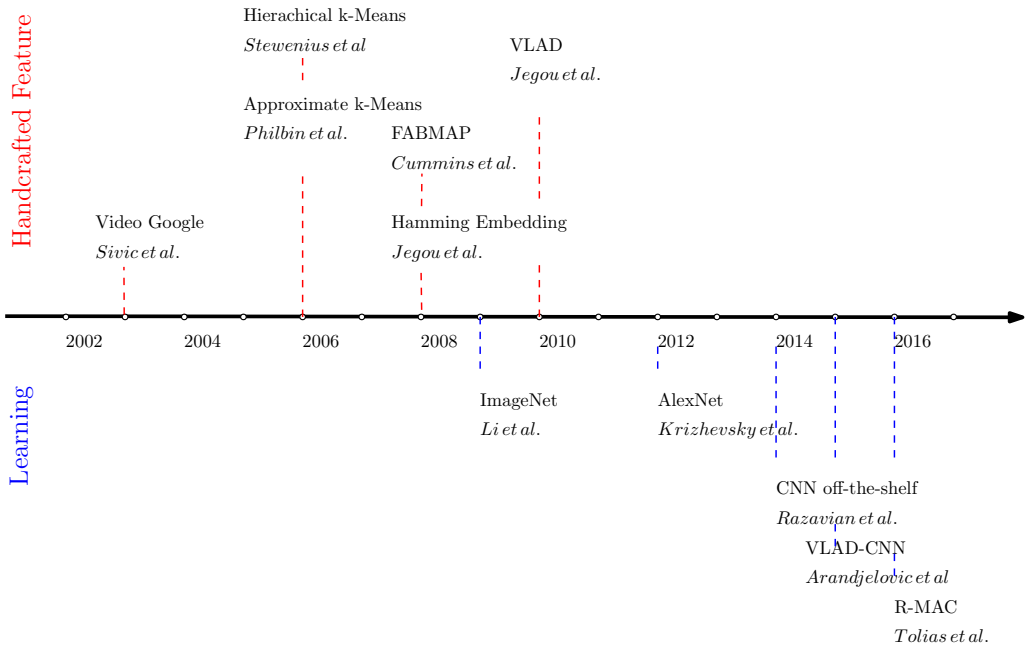


Figure 3.3: Milestones of image retrieval in place recognition.

3.2.1 Handcrafted-feature-based Place Recognition

The term "*handcrafted*" means a method designed by human instead of being the result of a machine-learning algorithm. In computer vision, it focuses on the feature detectors or descriptors designed according to the computer vision theory, such as SIFT, SURF [17], BRIEF [27], FAST [161] and CenSurE [3]. Each detector and descriptor has its own advantages and disadvantages. Most detectors are keypoint centric, including Harris, Hessian [140] and FAST detector, some are regional, like the Difference of Gaussian (DoG), the Maximally Stable Extreme Region (MSER) detector and the affine Harris-Laplace detector. With a set of detected regions, descriptors are used to encode the local contents. Apart from SIFT, we can also use PCA-SIFT to speed up matching by reducing the dimension. RootSIFT [8], SURF, BRIEF, DenseSIFT and PHOW [207] are also tested in image retrieval as well. We will compare these handcrafted features profoundly in the Chapter 6.

The handcrafted-feature-based place recognition consists in three phases: local feature extraction, dictionary training and parameterization. In the local feature extraction, we often use one type of feature detector like DoG, Hessian-Afine or dense patch to extract keypoints and one type of descriptor, *e.g.*, SIFT, SURF or BRIEF, to represent a large set of database images [138, 207]. Then a large collection of descriptor vectors are clustered by employing the flat k-means [74] or the hierarchical k-means [208]. This stage is a combination of dictionary training and parameterization. The centroids of the clusters are registered as the *visual words* and they form a compact and discriminative set of descriptor values called the *dictionary*, or *codebook*. The k-means method encodes features according to its distances to K visual words. K is a hyper parameter set up by ourselves. Each visual word is partitioned in the center of a subspace of the dictionary like the "Voronoi cell". For a query image, the same detectors and descriptors are used and the extracted feature vectors are classified within the dictionary. As such, every observed descriptor will be assigned to a similar visual word in the dictionary. Then the query image is described by a K-dimensional histogram vector about the occurrence of visual words, by counting how often individual visual words occur. There are several variations to normalize the distribution, *e.g.* L2 norm distance. In order to distinguish the importance of every visual word, the vector components are often weighted by the overall occurrence of the word. One of the most popular weighting schemes is the term frequency-inverse document frequency (TF-IDF). The TF-IDF weight increases proportionally to the number of times a visual word appears in the database, but is often offset by the frequency of the word, which is able to adjust for the fact that some words appear more frequently in general. After this weighting, we can use inverted file indexing to map the query image to the indices of candidate images in the database with similar visual worlds. Please note that both query and database images can be represented according to the obtained dictionary.

Suppose we have N images in the database $\mathcal{D} = \{d_1, \cdots, d_N\}$, we use DoG and SIFT to extract features with 128 dimensions and then cluster them into K centroids by the K-means method. So the dictionary is denoted by $\mathcal{C} = \{c_1, \cdots, c_K\}$ with K visual words (centroids). The i^{th} database image d_i, with $i \in [1, \cdots, N]$ can be described by a descriptor set as $\mathcal{X}^i = \{x_1, \cdots, x_L\}$. The size L represents the total number of detected features in this image. Each local descriptor x_l, with $l \in [1, \cdots, L]$, is assigned to its nearest visual word as $c_j = \mathrm{NN}(x_l)$, with $j \in [1, \cdots, K]$. The NN means the nearest neighbor assignment algorithm. Thus the occurrence

of the j^{th} visual word c_j in i^{th} image is the sum over all 0/1 assignment (by an indicator function $\mathbb{1}$), denoted as:

$$\mathrm{TF}(c_j, d_i) = o_j^{d_i} = \sum_l \mathbb{1}(c_j = \mathrm{NN}(x_l)) \tag{3.2}$$

To simplify, we use (c_j, d_i) to describe the j^{th} visual word c_j in i^{th} image (d_i) in the database $\mathcal{D}$. The obtained occurence is also known as the TF (term frequency) of a visual world in one image. The TF thereby is a local weight.

The raw $K-$dimensional histogram vector of this image is computed by,

$$\mathrm{H}(d_i) = [o_1^{d_i}, \cdots, o_K^{d_i}] \tag{3.3}$$

Alternatively, IDF measures the rarity of a given visual word through the whole database, which is computed by:

$$\mathrm{IDF}(c_j) = \log \frac{N}{n_j}, \text{where}\, n_j = \sum_{d_i \in \mathcal{D}} \mathbb{1}(o_j^{d_i} > 0) \tag{3.4}$$

where N is the total number of images in the database as we stated before.

Thus, the TF-IDF weight for a visual word c_j in image d_i is calculated by:

$$\omega(c_j^{d_i}) = \mathrm{TF}(c_j^{d_i})\mathrm{IDF}(c_j) \tag{3.5}$$

Then the final histogram vector is weighted by multiplying the TF-IDF value. This is a simple pipeline of the feature-based method, known as bag of word (BoW). Main open issues about this paradigm include:

- how to decide the size of the dictionary;
- how to generate the visual words efficiently;
- how to deal with the tradeoff between feature reduction and information loss;
- how to identity the false-positives, which means two places appear very similar but they are in totally different positions in the environment. False-positives occur a lot due to feature saturations in large scale urban environments.

BoW can be improved in many aspects. Geometric verification with RANSAC [152], by generating tentative feature correspondances, can be introduced to reject false-positives. RANSAC is used to calculate the affine transformation for each matching repeatedly by verifying the number of inliers that fit this transformation. RANSAC has efficiency problems thus some effective strategies are proposed to refine the matchings in terms of geometric angle and scale, such the weak geometrical consistency (WGC) [89], Hough pyramid matching (HPM) [11] and WGC with Hamming Embedding [89], etc. In FABMAP [46, 47], authors tried to explore the mutual dependencies among the visual words and take the correlations as an advantage to prevent the false-positives. Also, if a query image has weak GPS tags, the tags can provide a weak supervision to filter out definite negatives [171, 223]. In order to manage a large database of visual words, some tree structures are also developed to precisely model the mutual information in a dictionary, such as Chow-Liu tree [35]. Angeli *et al.* [5] also proposed to train a dictionary online by adding visual words incrementally.

As we stated before, when a dictionary is generated, region descriptors are "hard-assigned" (by the NN algorithm) to the nearest visual word in terms of Euclidean distance of k-Means, see Equation 3.2. The hard assignment only takes account of the closest visual words but ignores the uncertain visual word (*i.e.* when two or more candidate visual words are all close to x_l) or the plausible word (*i.e.* when all visual words are too far away from x_l but the NN algorithm still assigns one unrepresentative visual word to x_l). To avoid this, soft-assignment [32, 153] is proposed to pay attention to these two types of visual words with a frequency smoothing process.

The technique of BoW represents the set of features by a sparse histogram. Jégou *et al.* [90] designed a low-dimensional but dense descriptor called the Vector of Locally Aggregated Descriptors (VLAD), which enables a common memory to process place recognition in large image datasets. VLAD [49] also uses a local descriptor (SIFT) at the beginning to do clustering work. Rather than BoW recording a sparse vector of word occurence counts, VLAD accumulates the residuals (vector differences between query features and visual words). Thus each image can be represented by an aggregation of residual vectors assigned to each visual word. Mathematically, the VLAD can be represented by

$$v_{j,l} = \sum_{j \in K, l \in L} (c_j - x_l), \text{ only when } c_j = \mathrm{NN}(x_l) \tag{3.6}$$

where indices $j = 1, \cdots, K$ and $l = 1, \cdots, L$ index the visual word and the local descriptor component.

Rather than a rough binary assignment in the BoW, VLAD use the normalized distance vector $v_{j,l}$ to represent the relationship between every visual word $_j$ to each descriptor x_l. Then, VLAD aggregates all vectors into one vector with a small dimension by principal component analysis (PCA) to represent one database image. This approach is a simplification of the Fisher vector representation [165]. Achieving the same accuracy as BoW, it outperforms in terms of memory usage and rotation or scale invariance.

3.2.2 Learning-based Place Recognition

Here, the learning-based methods are only limited to CNN-based retrieval techniques, overlooking the typical machine learning classifiers like support vector machines. We cannot deny that CNN-based retrieval methods are replacing the handcrafted detectors and descriptors gradually in the computer vision community. Recent results show that CNN features are more generic, discriminative and powerful than handcrafted features in the image recognition, scene classification and semantic segmentation tasks [19]. Since the thesis focuses on a metric visual localization while CNN methods in place recognition yield topological results by an end-to-end way, we will not dive deep into the learning-based place recognition.

In a simple way, a CNN model can be regarded as a set of non-linear functions involving series of hierarchical layers of convolution, pooling and non-linearities[3]. Each layer is able to capture a different level in the hierarchy of a scene. For instance, the first layer as the bottom one in the hierarchy can learn simple shape

[3]These terms will be explained insightfully in Chapter 8, in brief, pooling is used to reduce the spatial size of the feature representations.

features like lines, edges and blobs and the subsequent layers tend to learn more complex shapes like combinations of lines, and eventually CNN discovers the high-level characteristics of the object. Therefore, the top-level layers are less general-purpose than the layers before it. Each convolution+pooling layer is composed of a set of CNN filters that split an image into very small patches at multiple scale levels, and then max-pools these patches within neighborhoods. Filters and max-poolings serve respectively as the detectors and descriptors in the handcrafted feature [231]. We will come back to explain this scheme elaborately in the Chapter 8. For now it suffices to understand that CNN features preserve hierarchical characteristics from an image.

The CNN-based image retrieval can be divided into two parts: using pre-trained CNN models and using fine-tuned CNN models. Since 2012, many successful CNN architectures are pre-trained on the ImageNet (containing 1.2 million images with $1,000$ categories) [50] for a classification task. Currently popular CNN architectures are proposed to extract features, such as VGG [178], GoogleNet [190] and ResNet [78]. These models are pre-trained on large-scale recognition database, including Places [232] and ImageNet. With the help of these models, we can extract different types of features for place recognition. We can use the descriptors from the *fully connected layer* (FC)[4] layer of the network, or rely on the low-level descriptors from the intermediate layers. Recent studies [107, 230] show that the top-level descriptors underperform in place recognition and image retrieval due to the fact that they are more specific to the particular image classification task they have trained for.

As we stated before, the descriptors are extracted by filtering small patches at multiple scale levels. They can be regarded as a set of SIFT features. Therefore, the previously introduced encoding methods can be conducted as well, for instance, Arandjelović *et al.* [9, 222] encoded all CNN features into VLAD representation and instead Kulkarni *et al.* [107] adopted the BoW encoding. Apart from the encoding way, pooling is another way to create global discriminative features to represent images. Well known work includes the Maximum activations of convolutions (MAC) descriptors that and its evolution proposed by Tolias*et al.* [198].

Although many impressive image retrievals have been produced by pre-trained CNN architectures, fine tuning a CNN model for a specific training dataset is still a popular method. In the pre-trained CNN methods, extracted features are mainly for a classification task and these features are able to distinguish between different semantic categories but are not robust to intra-class variability. They are lack of geometric invariance compared with handcrafted features. However, in the image retrieval or place recognition, we need to identify and discriminate particular objects even if they are of the same semantic type [67].

Without employing pre-trained deep networks as a black box to generate features, we can produce image-level descriptors by leveraging a deep architecture trained for the image retrieval in an end-to-end manner, such as the PlaNet [212]. For the fine tuning methods, instead of using ImageNet with various class labels, we are able to learn a CNN model from some instance-oriented datasets, including Oxford5k [152], Holidays [89], Tokyo 24/7 dataset [9] and Multiview RGBD dataset [110]. Please note that clean training data is the key to the success of the fine-tuning methods.

[4]A fully connected layer is a standard, non-convolutional layer, where all inputs are connected to all output neurons. In some deep learning libraries, it is also referred as *dense layer*.

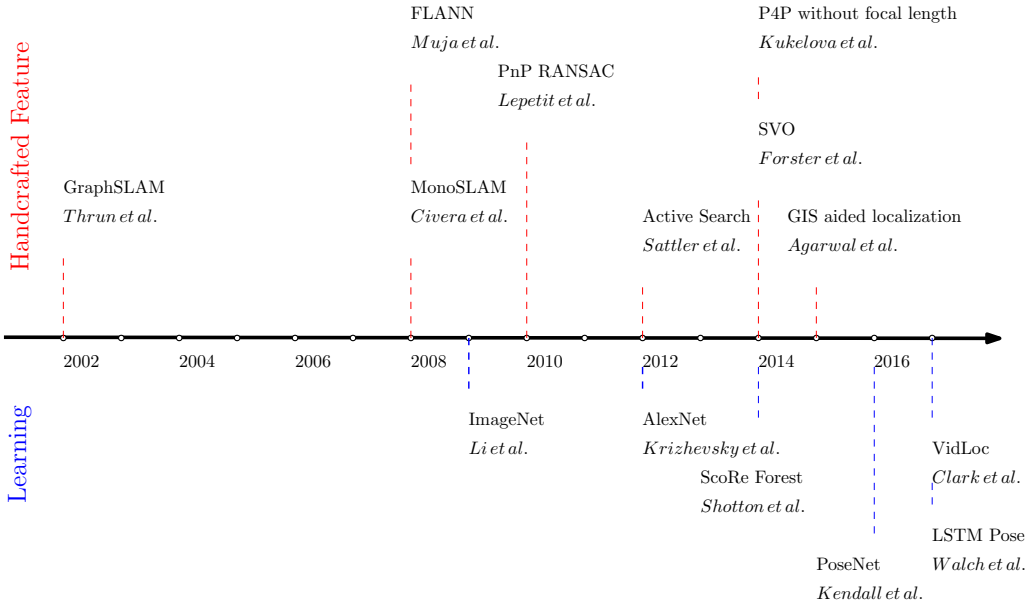


Figure 3.4: Milestones of metric visual localization.

CNN Siamese model [157] is also an option by leveraging matching or not-matching image pairs into a CNN model to learn positive matching in the place recognition. Moreover, Gordo *et al.* [68] proposed a global feature representation for image search by aggregating many region-wise descriptors.

3.2.3 Summary

In this section, we described handcrafted-feature-based and CNN-feature-based methods to solve the place recognition problem. It seems they are two distinct approaches but they share the same paradigm: extracting distinctive local features, aggregating features into an image-level vector and indexing or compressing vectors for computational efficiency. Some work [92] also prove that geometric matching with local features is an effective way to improve CNN training.

3.3 Metric Visual Localization

Instead of determining which place is visible in a query image as the place recognition problem, *metric visual localization* techniques, also known as *image-based* or *structure-based*, aim at computing the exact position and orientation of a query image. Similar to place recognition, metric visual based localization approaches can also be divided into handcrafted-feature-based and deep-learning-based methods. Figure 3.4 illustrates the milestone studies in the field. Since obtaining an accurate pose is the objective of the whole thesis, we will take a close look at 2D-3D matching approaches, particularly in the big aspect of current localization techniques, such as SLAM and visual odometry. The deep-learning localization approaches are also a big part in this section as they completely forego the handcrafted feature matching and try to regress the camera pose directly. Thereby we will discuss different representations of deep learning localization and introduce their advantages and disadvantages compared with traditional feature matching methods.

3.3.1 Handcrafted-feature-based Metric Localization

A simple handcrafted-feature matching method for image-based localization is usually done by the following pipeline:

- the scene is represented by a 3D structure-from-motion model;
- a set of keypoints are extracted in both query and database images, *e.g.* SIFT;
- their descriptors are matched by nearest neighbor search algorithms, *e.g.* FLANN [144];
- matching outliers are rejected by the ratio test [125] and geometric verification using constraints from the homography or the fundamental matrix, *e.g.* RANSAC.

Many state-of-the-art methods estimate the pose from 2D-3D matching between 2D features in the query image and 3D points in the database. Thus, the descriptor matching plays a vital role in the pose accuracy. Handcrafted features are normally high-dimensional which makes it time consuming to compute nearest neighbor matching. In fact, searching the reliable matches in high-dimensional vectors is always the most computationally expensive part in computer vision algorithms, especially when the dataset is large scale. Multiple nearest neighbor search algorithms are available, including kd-trees [144, 145], hierachical k-means trees [145], product quantization [91, 96] and inverted multi-index techniques [13]. Muja *et al.* [144] proposed automatic configurations to determine the fastest approximate nearest neighbor algorithm for specific dataset. They released this research as an open source library called FLANN incorporated in OpenCV. Apart from nearest neighbor algorithms, Lowe's ratio test [125] can also be used to reject wrong or ambiguous match. After establishing 2D-3D matching, we estimate the camera pose by solving the n-Point Pose problem (PnP). Assuming we have n 2D-3D correspondences noted as $(\mathbf{x}_i, \mathbf{X}_i)$, the PnP problem aims at recovering the pose $[\mathbf{R}, \mathbf{t}]$ with the relationship as stated in the Equation 2.10:

$$\mathbf{K}[\mathbf{R}, \mathbf{t}]\mathbf{X}_i = \lambda \mathbf{x}_i \tag{3.7}$$

Please note that the PnP problem can be divided into two categories depending on whether the camera intrinsic matrix $\mathbf{K}$ is known or not. If the camera is well calibrated, three correspondences are sufficient to estimate the pose efficiently by solving a fourth degree polynomial by a 3-point pose (P3P) solver. The P3P-RANSAC solver has been extensively studied in the state of the art [57, 72, 102, 115]. However, for images in photo-sharing community, intrinsic parameters are usually missing and the pose estimation turns to be difficult and inefficient, which requires solving multivariate polynomials with at least four 2D-3D correspondences [26, 75, 106, 170].

Concerning RANSAC, there are many different variants to improve the performance of the standard one. Torr *et al.* proposed MLESAC [204] which adopts the sampling solutions to maximize the likelihood rather than the number of inliers. Chum *et al.* [37] presented a $T_{d,d}$ test to reject bad putative solutions generated randomly by a standard RANSAC. They also introduced a locally optimized RANSAC that speeds up the process by reducing the number of samples according to inlier probability. Other alternatives leveraging state-of-the-art algorithms for

various modules, such as PROSAC [36], Randomized RANSAC [37], USAC [159] and SCRAMSAC [166], have also better performance than a standard RANSAC.

Actually, the above pipeline is just a classic pipeline [144] for a 2D-3D matching method. In recent years extensive studies have been conducted with a focus on how to determine descriptor matching for large scale data structures [34, 119, 120, 167, 168, 189], how to enable mobile devices, including tablets and smartphones, to process these highly consumptional localization algorithms [121, 126, 136]. Thereby, datasets and benchmarks with millions of 3D points are generated for test, for example Landmark 1K dataset [120], and some city datasets including Dubrovinik [120], Vienna [88], Rome [120], Aachen [169] and San Francisco [32]. To deal with large scale datasets efficiently and effectively, some remarkable matching schemes are proposed, including kd-tree search [167], prioritized point to feature matching with existing visibility graph [119], vocabulary based prioritized search [167] and active search [168].

It is impossible not to talk about SLAM and visual odometry, which are two primary techniques in handcrafted based visual localization systems. Thereby some details about those two techniques are discussed below.

SLAM

SLAM plays such an important role in the localization and mapping system for autonomous car, that it requires to be presented here. Since the birth of modern SLAM at the 1986 IEEE ICRA conference [183], various forms of techniques have been used to solve it, ranging from Bayesian probability theory to computer vision, and even biological science. We can divide the SLAM problem by the sort of map it creates, by the mathematical algorithms adopted, or even by the sensor employed. Yet, considering the context of our thesis, we will only talk about visual SLAM. Actually, 2D-3D matching visual localization is the foundation for the handcrafted-feature-based pose estimation problem. A pure 2D-2D feature matching without the depth information or scale factor cannot recover the pose. In many visual SLAM, depth can not be directly inferred from a single camera. Instead, it can be computed through optical flow [155] between successive image frames. So strictly speaking, it is still a 2D-3D matching pipeline. The optical flow is well studied: given an image sequence of a rigid 3D scene taken from a moving camera, it is possible to compute both a scene structure and a camera motion up to a scale factor. The moving camera observes every scene feature repeatedly and captures a ray of view from the feature to its optic center. The parallax angle [229] of the captured rays is measured to estimate the depth.

As we mentioned, we will make use of a given pre-existing map to develop our urban localization system instead of building a map by ourselves. The idea appears to deviate from the conception of SLAM in the following aspects: rather than the classic SLAM problem supposing an origin to approximate the pose, our case is to localize a vehicle in a GIS dataset with absolute references. The vehicle is not required to visit all the environment beforehand. Moreover, we have to consider how to represent the GIS dataset in a robotic accessible way: namely in a metric, topological or hybrid type. However, if we adopt handcrafted-feature-based method to solve the urban localization, it is meaningful to bring some inspirations from current successful SLAM algorithms. Here we give an introduction to the state-of-the-art of SLAM and then we discuss some related methods that are useful in our

case.

Given a mobile state vector $\mathbf{x}$, its interaction with the environment or the map $\mathbf{m}$ is measured by sensors as $\mathbf{z}$ and a control action as $\mathbf{u}$.

Then we characterize the evolution of the state and the process of measurement at timestep t as two parts.

- State transition probability: $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t)$
- Measurement probability : $p(\mathbf{z}_t|\mathbf{x}_t, \mathbf{m}_t)$

This temporal generative model is also known as a hidden Markov model or dynamic Bayesian network. The motion models describe the state transition part between poses and the corresponding odometry measurement. The odometry along the whole time is denoted by $\mathbf{u}_{1:T}$.

$$\mathbf{u}_{1:T} = \mathbf{u}_1, \ldots, \mathbf{u}_t, \ldots, \mathbf{u}_T \tag{3.8}$$

A nonlinear motion model f is defined as:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) + \mathbf{w}_t, \tag{3.9}$$

where $\mathbf{w}_t$ is the Gaussian noises in the motion model represented by $\mathbf{w}_t \sim \mathcal{N}(0, \sum_t)$.

In the same way, the sensor measurements are written as,

$$\mathbf{z}_{1:T} = \mathbf{z}_1, \ldots, \mathbf{z}_t, \ldots, \mathbf{z}_T \tag{3.10}$$

A nonlinear measurement model h is denoted as:

$$\mathbf{z}_{t+1} = h(\mathbf{x}_t, \mathbf{m}_t) + \lambda_t \tag{3.11}$$

where λ_t is the measurement error with $\lambda_t \sim \mathcal{N}(0, \Lambda_t)$.

The SLAM framework is constructed by motion model, sensor model as well as data association process. Based on above models, there are various SLAM algorithms developed due to different quantities estimated, summarized in Table 3.1. In *filter based* SLAM,[5] we only estimate the single pose at current time t, that is $p(\mathbf{x}_t, \mathbf{m}_t|\mathbf{u}_{1:t}, \mathbf{z}_{1:t})$; while *optimization based* SLAM [6] calculates *a posteriori* estimation over the full trajectory, namely $p(\mathbf{x}_{1:T}, \mathbf{m}_t|\mathbf{u}_{1:t}, \mathbf{z}_{1:t})$. The *full* SLAM estimates $\mathbf{x}_{1:T}$ and map structure $\mathbf{m}_t$ during the whole path all at once. However, currently we also encounter incremental smoothing SLAM approaches that realize online estimation of $\mathbf{x}_{1:t}$ during the full trajectory up to the current time t. In fact, *filter-based* approaches, such as Extended Kalman Filter (EKF) or Particle Filter [195], have dominated SLAM community for a long time. However, due to the inherent sparsity of the SLAM problem in a nonlinear least square optimization, *smoothing* SLAM problem can be solved as efficiently as the one based on filters. SLAM researchers prefer to express *optimization-based* SLAM problem by a graph representation known as *GraphSLAM* [195].

EKF-monoSLAM [38, 39, 48] is one of famous filter-based visual SLAM, which adopts the inverse depth algorithms to estimate the 3D scene. Considering the depth

[5] Also known as *online* SLAM

[6] Also known as *full* or *smoothing* SLAM

Estimated Quantity	Algorithm
$p(\mathbf{x}_t, \mathbf{m}_t \vert \mathbf{u}_{1:t}, \mathbf{z}_{1:t})$	Filtering
$p(\mathbf{x}_{1:T}, \mathbf{m}_T \vert \mathbf{u}_{1:T}, \mathbf{z}_{1:T})$	Smoothing
$p(\mathbf{x}_{1:t}, \mathbf{m}_t \vert \mathbf{u}_{1:t}, \mathbf{z}_{1:t})$	Incremental Smoothing

Table 3.1: Different probabilistic estimation

information is available in the GIS, it is not necessary for us to follow this scheme. Optimization-based SLAM is also called GraphSLAM, because this technique is often represented as node-edge graph, see Figure 3.5. Nodes in sequence are linked by edges that encode the movement of the robot given by odometry and corresponding observations. As the optimization is a frequent method in the localization problem, here a detailed state of the art is introduced. Optimization based SLAM typically consists of front-end and back-end subsystems. The front-end identifies the constraints based on sensor data that is referred to as data association problem; the back-end corrects the poses of the robot to obtain a map of the environment consistent with the given constraints. In robotics, many problems can be modeled as a spring-mass system in Figure 3.5(c) and solved by finding the minimum of a general cost function in the following form:

$$\mathrm{F}(\mathbf{x}) = \sum_{\langle i,j \rangle \in \mathcal{C}} \underbrace{e(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij})^\top \, \Omega_{ij} \, e(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij})}_{F_{ij}} \tag{3.12}$$

In above function, $e(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij})$ is an error function that computes the difference between the expected observation $\mathbf{x}_j - \mathbf{x}_i$ and the real observation $\mathbf{z}_{ij}$; Ω_{ij} represents the inverse covariance matrix. Note that $\mathbf{x}_i$ and $\mathbf{x}_j$ are generic parameter blocks that represent every node in Figure 3.5(c).

The optimal value $\mathbf{x}^*$ will be calculated by:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \, \mathrm{F}(\mathbf{x}) \tag{3.13}$$

In general, the function $\mathrm{F}(\mathbf{x})$ can be simplified by local approximation using popular Gauss-Newton or Levenberg-Maraquart algorithms. With derivation to the modified $\mathrm{F}(\mathbf{x})$, we can acquire the solution by adding the increments to the initial guess. To apply this general model into GraphSLAM, we can determine the most likely pose and map:

$$(\mathbf{x}^*, \mathbf{m}^*) = \operatorname{argmin} \underbrace{\sum_i \| f(\mathbf{x}_i, \mathbf{u}_i) - \mathbf{x}_{i+1} \|^2_{\Sigma_i}}_{Odometric\,Constraints} + \underbrace{\sum_{ij} \| h(\mathbf{x}_i, \mathbf{z}_{ij}) - \mathbf{x}_j \|^2_{\Lambda_{ij}})}_{Close\,Loop\,Constraints} \tag{3.14}$$

The typical approximation methods like Gaussian-Netwon, Levenberg-Maraquart, Gauss-Seidel relaxation or gradient descent provide acceptable results to minimize non-linear error problem. But they suffer from registration errors during loop closure and tend to fail in large environments due to a greedy maximization step. From the formulations of the GraphSLAM paradigm, each parameter $\mathbf{x}_i$ consists of a translation vector $\mathbf{t}_i$ and a rotation component $\mathbf{R}_i$. The translation $\mathbf{t}_i$ clearly forms an Euclidean space, in contrast, the rotational vector spans over the non-Euclidean 2D or 3D rotation group $\mathbb{SO}(2)$ or $\mathbb{SO}(3)$. To avoid singularities, the spaces are usually

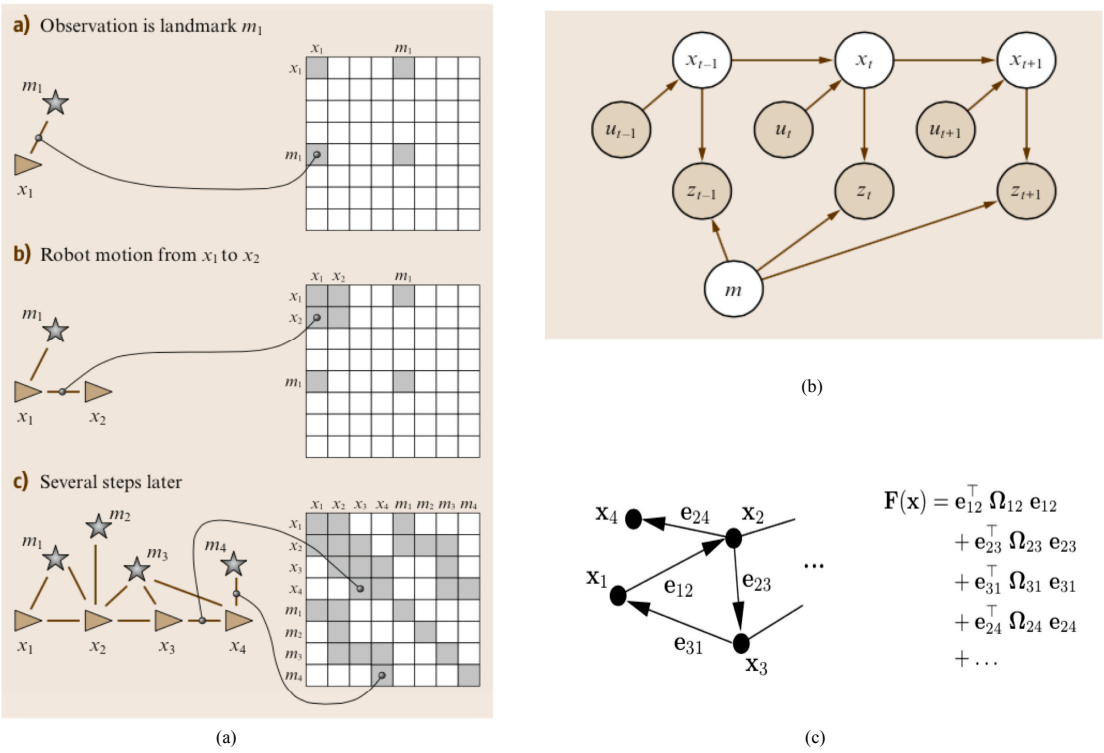


Figure 3.5: Three representations of Optimization based SLAM: (a) Transition graph and corresponding constraints; (b) Graph model of SLAM; (c) General mathematical representation of Graph optimization [108]. Image (a),(b) courtesy of [194].

described in an over-parameterized way, *e.g.* by rotation matrices or quaternions as mentioned in Chapter 2. Another idea is to consider the underlying space as a *manifold*: HOG-MAN algorithm [69] proposed a new hierarchical optimization solution on manifolds. It constructs different levels of the hierarchy to represent the original problem at different levels of abstraction. The bottom corresponds to the original input, while higher levels capture the structural information of the environment. In the g2o algorithm [108], a similar representation is adopted to avoid the singularities. Moreover it takes advantage of the special structure of the Hessian matrix to reduce the system complexity by using Schur complement. COP-SLAM [52] optimizes pose graph by constructing a novel pose-chains system by employing an extended version of trajectory bending with respect to the pose graph, pose chain system considers the relative pose displacement and corresponding transformation uncertainty. From a fundamental perspective, the chain can consist of transformations belonging to a Lie group whose exponential and logarithmic mappings are computable for all its elements.

Moreover, we can draw inspiration from the various map representations in SLAM to represent our GIS data. Different sensors used in SLAM algorithms will build up quite different map types. Generally speaking, the representation of robotic maps is divided into metric, topological and hybrid forms. The metric framework is created on 2 or 3 dimensional space with precise coordinates of objects. This representation is useful but noise prone. The topological framework is a node-edge graph that only describes objects and their distance. This representation is of low space complexity but difficult to construct in a large-scale environment.

Feature map represents the environment only with the positions of salient features or landmarks in the scene, *c.f.* Figure 3.6(a). In monocular visual SLAM, a feature map can be stored in a single vector as $\mathbf{m} = [\ f_1, f_2, \dots\ ]^T$, that only contains coordinates of features f_i. Although this approach is a sparse representation of the environment, newly discovered features will instantly augment the dimensions of the map vector.

Occupancy grid map is another way to model the environment as an array of cells, as depicted in Figure 3.6(b). Each cell holds a value to measure the probability that this region is occupied by an obstacle. It is widely used in obstacle avoidance and path planning algorithm. Also, it is suitable to represent dense structure of environment and can be generated and updated from sonar or laser scan finder data. Although it can achieve good results, this approach is difficult to apply in a wide and dense environment due to the resolution requirement and calculation burden.

Pose graph (see Figure 3.6(c)), is a representation of the robot's trajectory as a graph structure in which nodes represent robot poses and the edges between poses represent the spatial constraints. In contrast with the feature or grid map, this map is ego-centered and hardly concerns landmark or obstacle information outside. Yet, we can attach outside information into nodes and integrate odometry information or loop closure into edges. Since this data structure is close to the concept of GraphSLAM, we can find its applications in many optimization based algorithms. Moreover, this map is widely combined with occupancy grid map as a hybrid representation. The size of the pose graph should be accounted likewise due to its big influence on computational cost.

A hybrid map is often referred to as a metric topological representation[7]. This kind of map is often used to manage large-scale environments. The insight is that local metric maps are stored in the nodes of a global pose graph map. The difficulty of hybrid map is to achieve good scaling properties at a global level. Meanwhile, fusion between metric and topological information must be accordant with geometry constraints [20].

Visual Odometry

Apart from the SLAM paradigm, visual odometry [148] can also estimate the structure and motion directly from raw images. In visual odometry, the local or global intensity features are used to render the pose transformation instead of the geometry of handcrafted features. Visual odometry outperforms feature-based methods in robustness and less-structured environment, or in camera defocus and motion blur situations. However, it is influenced by the accumulated errors introduced during every frame-to-frame motion. Instead of minimizing re-projection error in the visual SLAM, the photometric error is computed and optimized in visual odometry algorithms. As for the pipeline of visual odometry, we denote $\mathbf{I}_{0:t} = \{\mathbf{I}_0, \dots, \mathbf{I}_t\}$ to represent a set of images taken up to step k. The rigid body transformation between two adjacent time $k-1$ and k is defined as,

$$\mathbf{T}_k = \begin{bmatrix} \mathbf{R}_{k,k-1} & \mathbf{t}_{k,k-1} \\ 0 & 1 \end{bmatrix} \tag{3.15}$$

[7]Also known as the combination of at least two different maps.

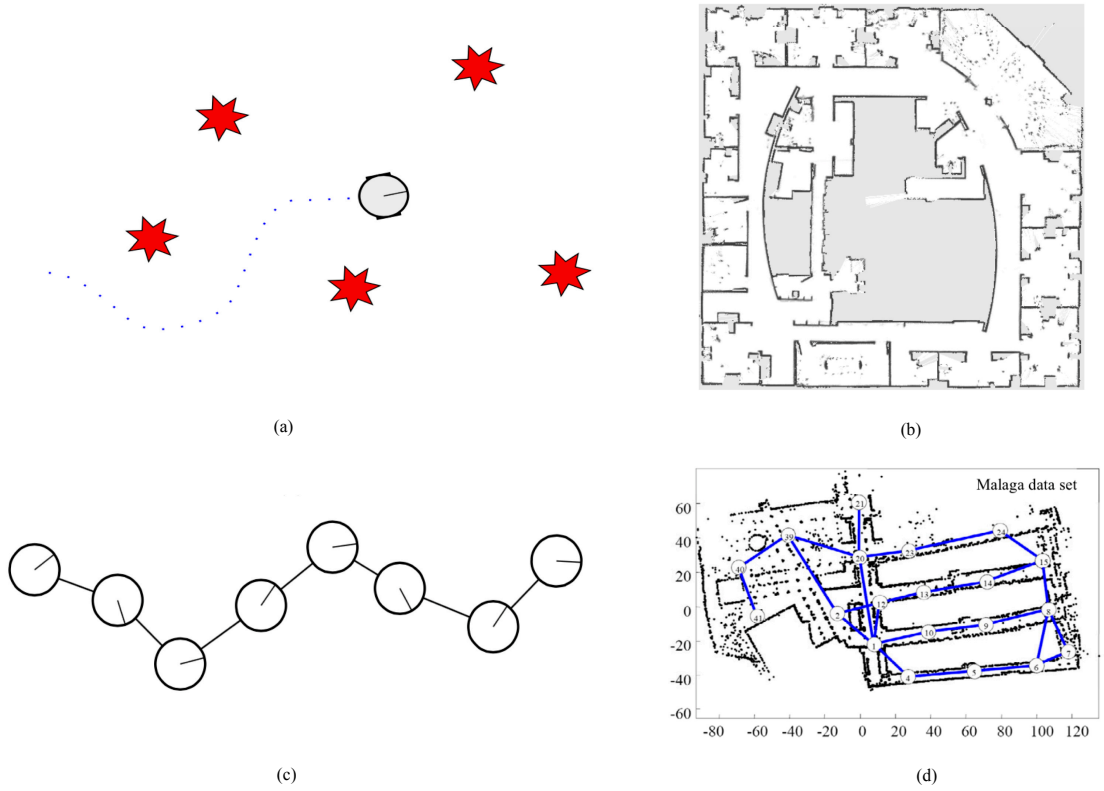


Figure 3.6: Illustration of general representations of map: (a) Metric feature based map contains only coordinates of distinctive features; (b) In 2D occupancy grid map, black regions are occupied, white are free of obstacles and gray are unobserved; (c) In pose graph, nodes contain orientation and position while edges contain relative relationship; (d) Hybrid map usually combines occupancy gird map with pose graph; Images (a),(b), (c) and (d) are coming from [195] and [20] respectively.

$\mathbf{R}_{k,k-1}$ and $\mathbf{t}_{k,k-1}$ describe the rotation matrix and translation vector respectively. We can use optimization technique to compute the relative motion $\mathbf{T}_k$, using either the intensity information of all pixels or only salient and repeatable feature across images $\mathbf{I}$. Therefore, if initial camera pose $\mathbf{C}_0$ is given, we can recover the full trajectory of camera $\mathbf{C}_{0,\ldots,t}$ pose after pose according to the equation below:

$$\mathbf{C}_n = \mathbf{C}_{n-1}\mathbf{T}_n \tag{3.16}$$

Visual odometry method is applied a lot to deal with a stream of image sequences captured by a camera. Most current visual odometry methods fail or cannot work effectively in outdoor environments with big illumination changes. They are more susceptible to the shadows and directional sunlight [7]. In our case, the GIS dataset was captured long time ago and in a cross season way, visual odometry thereby would not be a good option for our thesis.

3.3.2 Learning-based Metric Localization

Similar to place recognition, we now see more and more algorithms based on deep learning methods in the metric visual based localization. learning-based methods usually require a long-time training process but once models are generated, localization is often obtained in an end-to-end fashion with a low memory demand. Deep learning is a data oriented method which requires huge training dataset. Although many training dataset are labeled by SLAM methods, we believe a comparison between the handcrafted feature and CNN feature are meaningful.

Shotton *et al.* [177] employed a regression forest to estimate the camera pose from a single RGB-D image relative to a known 3D scene. A scene coordinate regression forest (ScoRe Forest) is trained to learn 2D-3D correspondence using RGB-D images with known camera poses. Every image pixel is labeled by scene coordinates inferred from depth and camera pose information. Once the correspondences between pixel coordinates and the 3D scene are learned, a pool of pose hypotheses is generated based on the correspondances and the RANSAC is used to refine a pose from the pool. This work avoids the traditional pipeline like feature detection, description and matching. Brachmann *et al.* [23] improved this approach with a single RGB image by modeling and reducing the uncertainty during the learning processing, by formulating the problem using a conditional random field to generate pose [134], and by changing the hypothesis selection of RANSAC to be differentiable and learnable [22].

PoseNet [100] is another impressive approach to train a naive end-to-end model to regress camera position and orientation with the single input of pose-tagged RGB images. It relocalizes query images with a 2m and 3° accuracy in outdoor scenes. It is able to learn CNN features with large context and contours invariant to poses and tolerate indoor/outdoor environment, motion blur, unknown camera intrinsics, weather changes, dynamic objects, etc. However, authors found it is difficult to learn both position and orientation at the same time and a large number of training dataset is required to obtain a good performance. Furthermore, geometric [99] and probabilistic [98] extensions to this algorithm are also explored. The experiments of [131] show that, traditional CNN architectures work better in scene coordinate regression and slightly less accurately in pose estimation compared to the test-time efficient Random Forests and ForestNets.

Walch *et al.* [43] leveraged the Long-Short Term Memory (LSTM) [82] units on the fully connected (FC) output in PoseNet to reduce the overfitting effects. This technique is capable of selecting the most useful context of feature correlations for the pose estimation and improves the previous accuracy by 5-50% depending on different datasets. Inspired by PoseNet, VidLoc [40] presented a bidirectional CNN-Recurrent Neural Network (RNN) model to perform pose estimation of a video clip, which outperforms PoseNet regarding the temporal consistency and the smoothness of the localization. Other improvements are also proposed, including RGB-D camera pose regression [118] with a dual PoseNet stream on RGB image and depth respectively and CNN map compression [43].

In the metric visual based localization, deep-learning-based methods are still pioneers and the localization inferred from dense context CNN features still remains coarse precision compared to the traditional handcrafted feature approaches. For the handcrafted-feature-based methods, the limitation mainly lies in the sparse scene representation: the number of correctly matched features primarily impacts the accuracy of the pose estimation.

3.4 GIS-aided Visual Localization

After introducing the two main categories of visual localization, we will give a particular focus on the GIS-aided visual localization since it is very close to our motivations. We are interested in the representations of GIS in the state-of-the-art which enable the urban navigation of a vehicle without exploring the environment beforehand. Our aim is to leverage the GIS to replace the consistent map building process in current urban localization systems.

As far as we know, there is few literature about GIS-aided localization systems. We also find that Google Street View is the most frequent used GIS database. In [128], a ground-air localization system is realized by matching images acquired from a MAV (Micro Aerial Vehicle) with Street View images. This system aims at solving the urban place recognition problem from aerial vehicles without GPS signals. The localization works at place recognition level and is able to deal with extreme changes caused by viewpoint, illumination, perceptual aliasing and over-season variations. Google Street View is converted into an image-feature-based representation by a virtual-view generation method called ASIFT [8]. Further they improved the accuracy using precise 3D cadastral models of buildings in [128]. Vaca-Castano et al. [206] estimated the geo-localized trajectory of a camera in an urban environment with the similar feature matching scheme as [128], but they smoothed the whole trajectory using a MST[9] to eliminate noisy results. Taneja et al. [193] employed a recursive Bayesian framework on the matching scheme to perform continuous positioning of a vehicle inside the discrete structure of Street View. Instead of matching, in [223], query images are localized by an indexed tree which is constructed by SIFT descriptors of $100,000$ geotagged Google Street View images and the GPS location of a query image is searched by a voting method. Since common feature detectors and descriptors fail to realize robust city-scale matching between query images and Street View, invariant or informative descriptors are extensively

[8] Affine Scale-Invariant Feature Transform

[9] Minimum Spanning Trees

studied, see [12, 60, 174, 201, 218].

However, all above approaches make use of Street View to formulate a place recognition problem instead of a metric localization. In a city environment, Street View imagery is able to serve as a reference database for the metric visual localization. For example, Zhang *et al.* [227] proposed a position estimation by triangulating the current image and matching features with two or more geo-tagged images from the Street View database. In [88], the vehicle position is calculated by applying a structure from motion algorithm with the accurate 3D urban models from Google Earth. Agarwal *et al.* [2] utilized Google Street View as a source of accurate geo-tagged imagery to allow robots to localize with a global reference. The localization problem is modeled as a non-linear least square estimation in two offline stages. The first estimates the 3D position of tracked feature points from short monocular camera sequences. The second computes the rigid body transformation between the Street View panoramas and the estimated points. The algorithm is validated in real urban scenario by using data from a Google Tango tablet. The accuracy of the localization is significant, about 40% within 1m and 60% within 1.5m lower than the mobile devices (5 to 8.5m) [225] from a cellular network and GPS.

The second frequently used GIS is OpenStreetMap, a free community-driven map. A prior information in OpenStreetMap, such as aerial images [109, 117], map networks [25, 58, 149], geo-referenced semantic objects [156, 158] are interoperated into a SLAM or visual odometry procedure to realize the localization. As such, map building is still essential and GIS only serves as a drift corrector for existing SLAM algorithms.

Recent approaches leverage Street View or OpenStreetMap in two ways: either place recognition or epipolar localization. The former focuses on fast retrieval and robust matching with the Street View database using invariant descriptors or appropriate data structures. The latter appears complex but metric results are often promising. They are all based on the hypothesis that the geo-referencing of Street View imagery is correct as a ground truth. The Street View database is widely used for semantic classification and recognition in the machine-learning field as the training dataset [215, 216]. With respect to learning-based localization, Street Views are often used as query images to test instead of training, such as in PoseNet and the Tokyo 24/7 dataset. It is probably caused by the discrete distribution and limited quantity of Street View. Moreover, only the geo-reference and Street View imagery are used in these algorithms. A well known city environment should also contain topological information as well. Is it possible to combine topological information with Street View to improve localization performance?

Meilland *et al.* [10, 42, 132, 133] realized an urban navigation using the visual odometry with a pre-processed spherical imagery. The spherical imagery is constructed using consecutive images associated with 3D points and tagged with precise positions along the trajectory. The consecutive images are also well calibrated with the fixed transformation relationship, especially the orientations between them. It serves as a reference to render the global localization and reduce the incremental error of visual odometry. Even though they constructed a global reference imagery by themselves, their node-edge representation of the spherical images and the topological information denoted by consecutive orientations inspire us a lot.

Inspired from the GIS-aided localization systems, our idea is to utilize GIS's accurate geotagged imagery to develop a rapid and precise localization system. Instead

of GIS-aided SLAM methods, we tend to make full use of depth information, topological information and imagery to localize a vehicle on a GIS map directly instead of building a map. From the handcrafted-feature-based methods, we can leverage the BoW scheme to obtain a coarse positioning for a query image from vehicle and refine the localization by the epipolar geometry. On the other hand, we will explore the use of geotagged GIS imagery as a training data to infer the pose directly.

3.5 Conclusion

This chapter addressed two main categories in the visual localization problem: place recognition and metric visual localization. Place recognition is a topological localization task while structure based localization aims at estimating the accurate metric pose. However, from the algorithmic point of view, they both can be solved by two aspects: classic handcrafted-feature-based and recent deep learning-based methods. Then many state-of-the-art techniques have been introduced and their advantages and disadvantages have been described. In particular, SLAM and visual odometry methods are presented due to their important roles in current autonomous vehicle field. We also reviewed some GIS-aided localization systems. They often leverage one single source of the GIS as a complement for their positioning. It inspires us to fully make use of the abundant GIS database.

After this literature review, we have a clearer view to propose solutions to our problems, such as how to represent GIS data in a robotic way from the SLAM methods; how to choose localization schemes, such as optimization or filtering methods; how to extract interesting features, including local point of interest or CNN high-level features.

Current visual localization approaches open up a big picture to guide us to design localization algorithms. In the following chapters, we will firstly present the system for the experimental test and the construction of an adapted GIS, then a traditional handcrafted-feature-based and CNN feature based algorithms will be proposed respectively.

Part II

Online Data Acquisition and Offline Database Construction from Geographic Information System

Chapter 4

Online Data Acquisition

4.1 Introduction

The main objective of this thesis is to realize a metric global visual localization in the urban area with an existing map data. According to the state-of-the-art presented in Chapter 3, we know that query images and database images are required. Thereby, we divide the data acquisition part into the *online data*[1] captured by a camera equipped on our own vehicle platform, and the *offline data* collection and reconstruction from Google Street View. Images acquired online are compared to database images from Street View with the aim of finding the relative pose transformation.

4.2 Review of current online dataset

At the beginning of the thesis, we favored existing online datasets so as to have a comparison with other methods. We did a survey[2] of the popular datasets, such as New college and Oxford city center datasets [46, 182], Rawseeds datasets [30], Malaga urban datasets [21] , Marulan datasets [151], Karlsruhe urban sequences [62], KITTI datasets [63] and MIT DARPA datasets [84]. According to the objective, our online dataset should be monocular image sequences with ground truth captured in urban environments where Street View data is available. Throughout the research (see in Table 4.1), we hardly found an ideal dataset, for example, the city of the KITTI dataset has no Street View; the MIT DARPA datasets mainly captured highway scenes; the Oxford city center and Malaga urban datasets have no ground truth for the image sequence. As a consequence, we constructed our own platform for the online dataset acquisition. Here we introduce the platform and sensors used, available data captured, and the test area respectively, then we illustrate some data examples.

[1]Readers might be confused about the concept of "online" since our following test is actually conducted offline but as close as possible to the real-time conditions. The term "online" serves as a contrast to the "offline" database construction from GIS.

[2]This survey was done in November, 2014 and now some suitable datasets are released, like Udacity urban datasets in 2016.

Dataset	GPS	GT	Street View	Urban Area	Path length	Images
New College (2008)	+	-	+	+	2km	Monocular RGB: 640*480
Oxford City	+	-	+	+	28km	Monocular RGB: 640*480
New College (2009)	+	-	+	-	2.2km	Stereo RGB: 512*384
Rawseeds datasets	+	+	+	-	1.9km	Monocular RGB: 320*240
Malaga urban datasets	+	-	+	-	36.8km	Stereo RGB: 1024*768
KITTI	+	+	-	+	>50km	Stereo RGB: 1392*512
Ford campus	+	+	+	-	>6km	Omni RGB: 1600*600
Karlsruhe sequences	+	+	-	+	6.9km	Stereo B/W: 1344*391
MIT DARPA	+	+	+	+	90km	Monocular RGB: 376*240

Table 4.1: A comparison of some datasets regarding the presence (+) or not (-) of GPS sensors, ground truth (GT), the Street View, the kind of cameras on the vehicle, the captured environments and the dataset path lengths.

4.3 Experimental Setup

Concerning the objective of the thesis, only image sequences captured in urban environment are used as input for our localization algorithm. In order to evaluate our algorithm, every image should be geo-tagged with precise position as ground truth. Thus a camera and a positioning system are mandatory in the experimental setup.

4.3.1 System Platform

In practice, we use two vehicles developed by VEDECOM as test platforms (see Figure 4.1), one is based on a Renault Zoé model and the other on a Renault Scénic. Both vehicles are equipped with laser sensors, front and rear cameras, RTK (Real Time Kinematics) GPS, 6 DoF IMU, IEEE 802.11p communication devices, as well as integrated processors and computers.

The only difference between them is the front camera setup. In the Zoé model, a stereo camera is embedded at the position of the front rear-view mirror. After the calibration, we can use either left or right eye image as an input captured by a monocular camera. In the Scénic model, we configured two monocular cameras manually in order to face street facades. Similarly, images filmed by either left or right camera can be regarded as an input. All sensor data are recorded, collected, processed, and fused into "Socket_Atlans.output" CAN (Controller Area Network) in RTMaps 4 (*cf.* Figure 4.2). The configuration can be measured during the calibration process. In the review of GIS-aided visual localization systems, we find that the online capturing cameras are usually set in the front of the car for place recognition tasks, while for metric structure based localization, they are often set to face street facades. Owing to the two platforms, we are able to explore the most suitable set-up in our case.

The autonomous car based on Zoé is the most recent model designed by VEDECOM. All sensors are reassembled together and hard to modify manually. Its inner stereo camera is strongly affected by illumination reflections, which is mainly used to detect the road or other vehicles according to VEDECOM. On the other hand, the capturing cameras on the Scénic are fixed to look sidewards (i.e. recording the building facades) in order to capture sufficient urban features.

All query images are captured in grayscale by the modular MIPSEE camera for

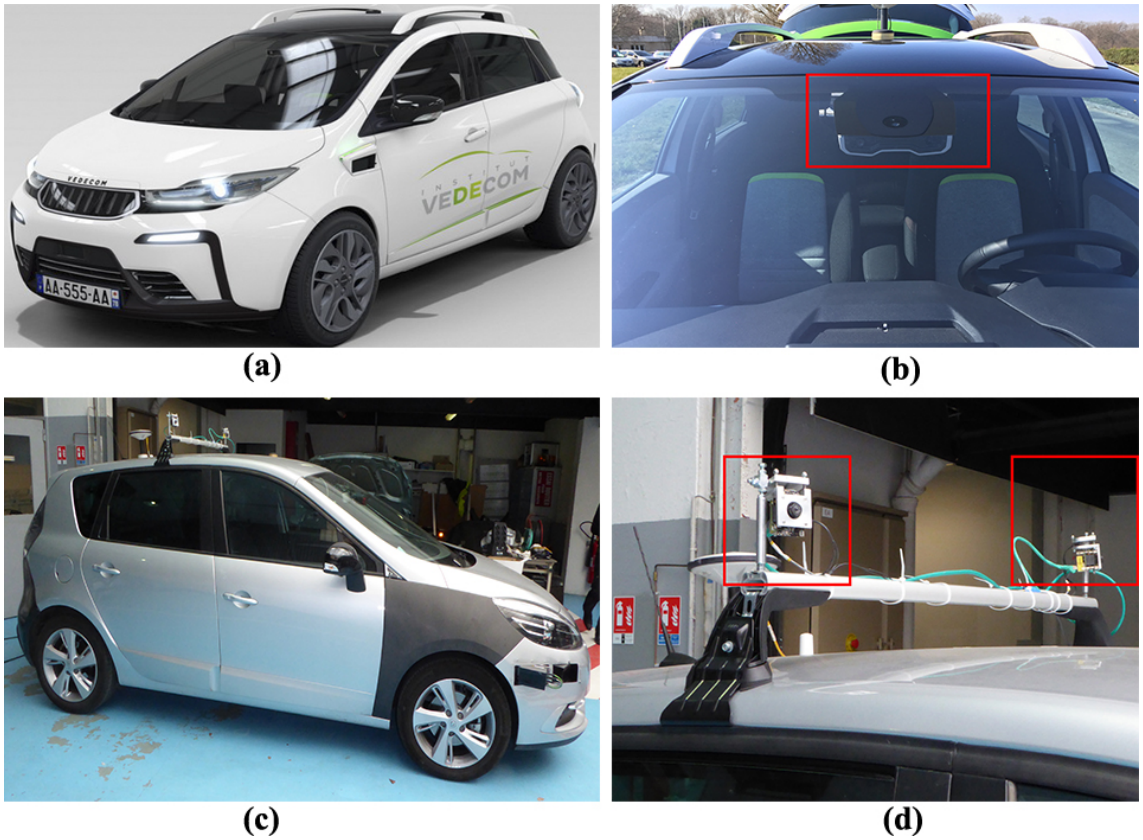


Figure 4.1: Platforms for online data capturing: (a) the autonomous vehicle Renault Zoé at VEDECOM; (b) the inner front stereo camera on the Zoé car; (c) the vehicle Renault Scénic at VEDECOM; (d) two cameras mounted on roof of the Scénic car.

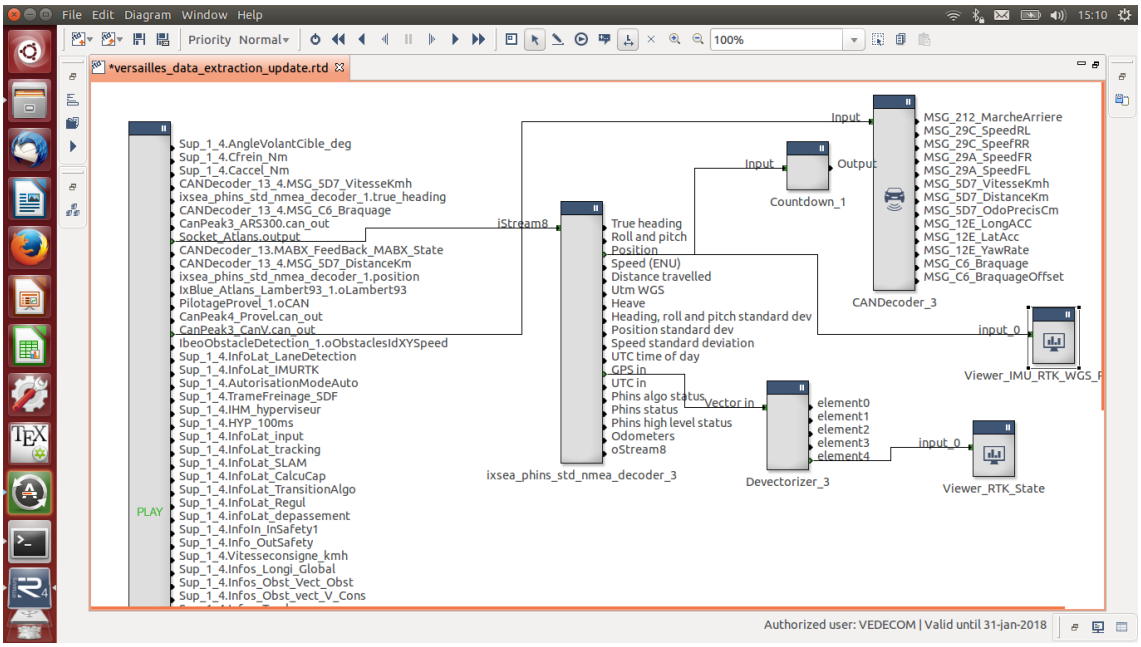


Figure 4.2: RTMaps 4.0 is used to record and preprocess the captured datasets. Besides the RTK GPS, we can also obtain many other parameters, like true heading, roll and pitch.

both vehicles, see Figure 4.3. It is a low-cost camera with 640×480 resolution, a 57.6° field of view (FoV) and a 20 frame-per-second frequency[3]. The localization ground truth is recorded by a RTK-GPS coupled with a highly accurate IMU. The global position can achieve a centimetric accuracy. However, the precision is strongly affected in urban environments where indirect measurements or non-line-of-sight phenomenon can occur [111]. Moreover, as illustrated in Figure 4.3(b), the position of the RTK-GPS is different from that of the camera. Thus, a rigid body transformation is necessary to transfer the RTK-GPS to the camera frame. We do this transformation according to the calibration and it is expected that the ground truth is the pose of the camera frame since the pose annotations of Google Street View correspond to the panorama point of view.

The camera and RTK-GPS have totally different frequencies. RTMaps just records all data in an asynchronous way. We synchronize the RTK-GPS on images based on timestamps and a simple interpolation.

4.3.2 Test Area

After preparing the platform, the online dataset has been recorded on several urban streets in the city center of Versailles in France, and covers a total length of roughly 5 km for each vehicle. The area was primarily chosen because of its good urban appearance (see Figure 4.5), high performance of RTK GPS and the traffic accessibility. Figure 4.4 shows a roughly processed test area describing the running trajectory.

Notice that not every recorded images are tested in our algorithm. Trajectory segments are selected carefully according to the following criteria:

[3]http://www.advansee.com/products

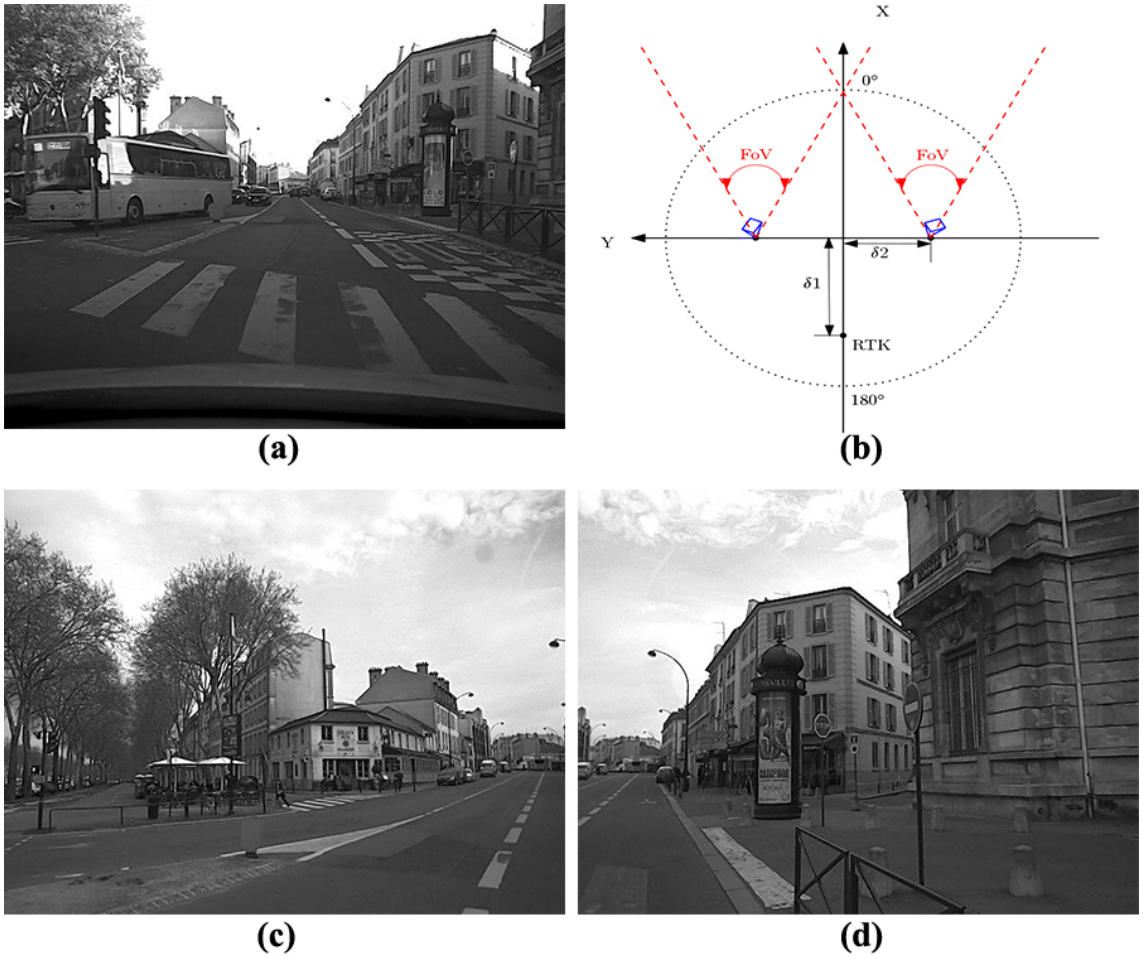


Figure 4.3: Illustration of online image examples and the capturing system: (a) online image captured by the left ego-camera on the vehicle Zoé; (b) birdview of the camera configuration on the vehicle Scénic; (c) a left image recorded by the Scénic; (d) a right image recorded by the Scénic.

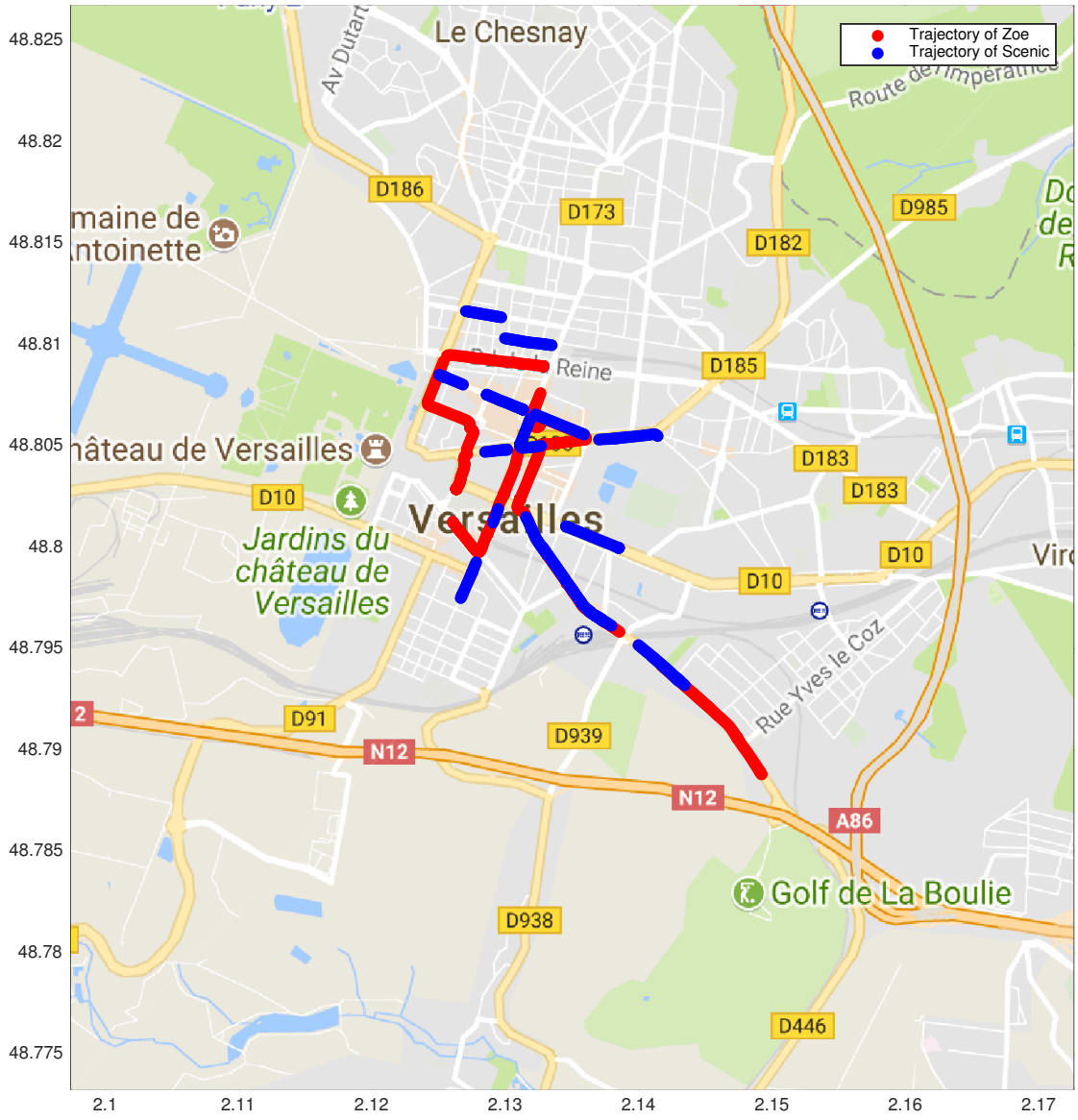


Figure 4.4: Illustration of the test area of two vehicles: the trajectory of Zoé and Scénic in red and blue respectively. Please notice some overlapping test areas are covered.

- The test area has an urban or at least a sub-urban appearance without the noise caused by broad vegetations[4]. For example, segments without urban features in the test area will no be tackled, *cf.* Figure 4.5;
- The RTK GPS should have a sufficient accuracy (good satellite visibility, low standard deviation, etc.).
- The coverage and depth quality of Google Street View should remain good in the same area, see next chapter.

4.4 Conclusion

In this chapter, we presented the online data collection in urban environment. Compared with the offline database construction presented in next chapter, the online data acquisition is simple to employ. We recorded image sequences from a well calibrated monocular camera and each image is geotagged by a precise RTK GPS. Then we filter out some trajectory segments with big noises or low quality.

[4]Vegetation is common in urban environment but if an scene is covered by tall trees extensively and this scene lasts for a long time, it is very tricky and even impossible for navigation by simple visual localization methods.

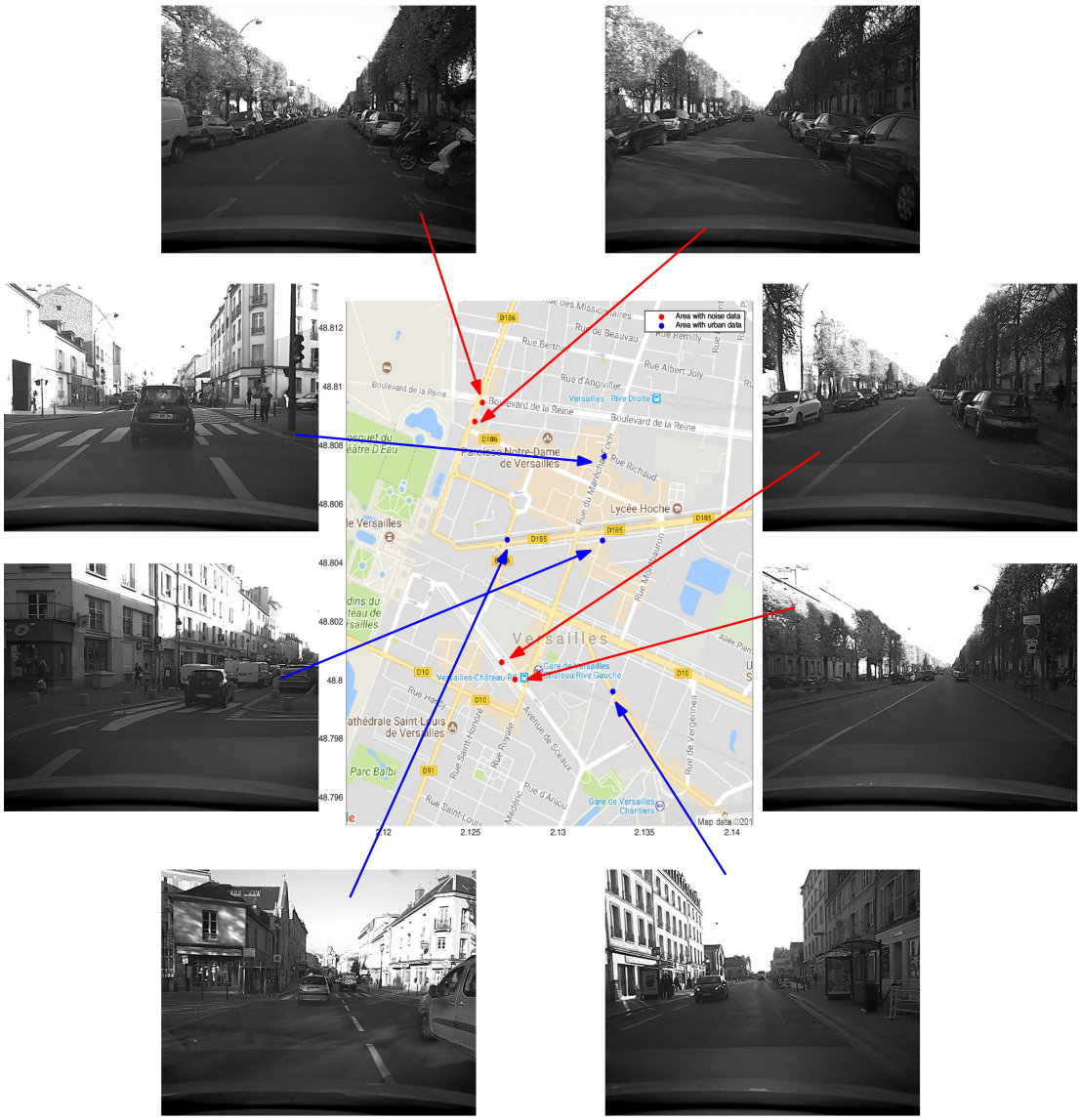


Figure 4.5: Image examples captured by Zoé with good urban appearance in blue spots and with noise in red spots. Since the camera is installed behind the windscreen inside the car, the captured image are all affected a lot by parasite reflections. The noise data lacks urban features due to big vegetations and they look similar even if they are captured in totally different places. It is very difficult to realize visual localization as stated in the Chapter 3 since the depth quality is low. We will overlook this scenario in the test area.

Chapter 5

Offline Database Construction and Urban Environment Modeling

In most current visual localization algorithms, the explored environment is represented by online captured data directly. However, in this thesis, we decide to represent and model the environment from the existing (offline) data, and then leverage this constructed model to localize the online data. If the existing data is at a world-wide scale, like a standard GIS, it is likely to realize a global visual localization. However, the urban environment is usually complicated, dynamic and full of illumination changes and moving objects. Therefore, we should carefully design the offline database construction, especially to decide how to represent this environment and from which type of data in the GIS.

Although the GIS generally records important data and filter out other useless information, the datasets are still too dense and various to deal with. Modeling the urban environment with the aid of the existing data is an important task for this thesis. In this chapter, we are going to explain how to choose an ideal existing GIS, how to extract useful data from it, how to make the data compatible with the online datasets and finally how to represent the urban environment in a compact way.

5.1 Geographic Information System

The term "*Geographic Information System*" was introduced by Roger Tomlinson in 1968 [199] when the computer hardware made digital topography and cartography possible. He also created the first operational GIS in Canada. The past decades have seen a big development in GIS. Now GIS's have become integrated representations of various captured spatial and geographic data, such as geodetic positions, 2D maps, panoramic images, depth maps and 3D cadastral models, etc. It is also an interactive platform to capture, store, represent, manage, update and process all geo-information.

GIS's play an essential role in our daily life with their public services from companies like Google Maps, Bing Maps, HERE, OpenStreetMap, Baidu Maps, TomTom and Mappy. Some more accurate GIS services are also accessible from national mapping organizations like the IGN (Institut National de l'Information Géographique et Forestière) in France. A good GIS asks for sophisticated hardware, software, comprehensive algorithms, precise capturing and frequent updates. We will not go

	GoogleMaps	HERE	Bing Maps	OpenStreetMap	Baidu Maps	TomTom	Mappy
Geo-data	+	+	+	+	+	+	+
Depth	+	+	-	-	+	+	-
2D Maps	+	+	-	-	+	+	-
HD Maps	-	+	-	-	-	+	-
3D Models	+	+	-	-	-	-	-
Live Maps	+	+	+	-	+	+	+
Street Imagery	+	-	-	-	+	-	-
Public Access	+	+	+	+	-	-	-
Route Planer	+	+	+	+	+	+	+

Table 5.1: A synthesis of current popular GIS.

further on how to design a qualified GIS at global level. But a survey is conducted as follows to understand pros and cons of current world-wide and precise GIS's towards the applications on the autonomous vehicle domain. We discuss the following items in various GIS's and analyze their coverage degree and accuracy. Note that "+" denotes the presence of elements, while "−" is the absence.

- Geo-data is the precise geodetic coordinates annotated to the captured imagery or laser scans;
- Depth provides 3D information of interesting points or surfaces in the scene;
- 2D map is a well-projected topological and metrically scaled description;
- HD map is a highly-defined machine-readable combined with a precise sub-lane level representation of road network,*e.g.* HERE HD Map.
- 3D Models are the cadastral landscape composed by 3D buildings;
- Live map is the dynamic traffic conditions and hazard warnings delivered by the real-time transportation statistics, which is widely used in motion planning;
- Street imagery provides urban scenes from positions along urban or rural streets, *e.g.* Google Street View;
- Public access is very important for our thesis but not all GIS's are free to access, for example, HERE is only for commercial clients;
- Route planner is designed to give one or several optimal trajectories from the starting point to destination considering time costs or other subjective requirements.

In GIS's, all information are normally organized and visualized according to its geo-data. The geo-data is frequently expressed according to its latitudes and longitudes in WGS84 system, which acts as key index variables to link other spatial or geographic information together [71, 160]. Considering the public access, accuracy and coverage, we choose Google Maps as our GIS data sources. As illustrated in Table 5.1, Google Maps nearly offers us all kind of useful information except HD maps. Nowadays, HD maps turn to be a powerful and even an essential mapping asset to support connected ADAS (Advanced Driver Assistance Systems) and highly

Figure 5.1: Google Street View capturing equipments: (a) From left to right depicts Street View Trekker, Street View Trolley for indoor acquisition, Street View Trike and Street View Car; (b) Google's R7 panoramic camera system.

automated driving use cases. Taking HERE as an example, its HD Live Map is self-maintained and built up by map layer, activity layer and analytics layer. Activity and analytics layers enhance the dynamic motion planner and comfortable control for automated vehicle. Its map layer is a well-designed street representation for positioning, localization and autonomous maneuvering, including slope, curvature, lane markings, roadside objets. However, the solution of HERE is not available at global level. Moreover, it will not furnish enough low level information to allow us to build a compact representation of the environment, since the main aim of HERE HD map is to realize the lane level localization. After the review, we found sufficient information in Google Maps to enable us to construct a global and light representation. As we focus on the vision-based localization, the main data source resides in the Street View of Google Maps.

5.2 Google Street View

Street View has been launched as a part of Google Maps since May 2007 and aims to gather global-scale street-level imagery with accurate geo-locations [6]. Till now Street View has expanded to over $3,000$ cities worldwide. Numerous data-collection platforms are developed, such as trekker, trolley, walking and vehicle apparatus (see Figure 5.1(a)) and the on-hand capturing camera system which are regularly updated. All the imagery of urban environment is collected from a car-mounted panoramic camera system dubbed R7 accompanied by Laser scanners. The R7 system is a ruggedized rosette of 15 identical, outward-looking cameras with 5-megapixel CMOS image sensors and low-flare, controlled-distortion lenses. All custom cameras are synchronized to increase the field of view and enable to collect fully-explored images even capturing the side-walks of a narrow street. The image set at the same timestamp is processed as a panorama by end-to-end stitching and privacy removal algorithms.

Then an accurate position estimate of the vehicle is associated to the panorama along a street map. The pose estimate is obtained by sensor fusion: GPS, wheel encoder and IMU data are logged to obtain raw pose estimates; then a offline Kalman

Figure 5.2: An extracted Street View of the Arc de Triomphe by setting parameters as 640 × 320 resolution, latitude= 48.8738, longitude= 2.2950, 0° heading, 0° pitch and 120° field of view.

filter algorithm and a batch optimization algorithm called GPO [6] are used to render a smoother and locally accurate pose. The final pose is transformed into a road-positioning data by a probabilistic graphical model of the road network [6].

The laser range data is aggregated and simply recorded by fitting it into a coarse mesh to model the dominant façades in the scene. This mesh is used to render a panoramic depth map by tracing rays from each panorama positon. Each pixel in the depth map represents a lookup into a table of 3D plane equations that allows us to reconstruct the real depth value. This data is also used to determine the consistent 3D cadastral models in Google Earth by fusion and filtering techniques [200]. The Street View team compacts panoramas and depth maps using lossless compression which allows a quick transportation over the network and a real-time representation at the frontend APIs.

5.2.1 Data Extraction

Till now, we know the main data of Google Street View consists in geodetic data, spheric images and associated depth maps. However, when we access one location at Google Street View web API, we normally get perspective images and their positions. In this way, our first task is to figure out how Google stores and represents all the information. In fact, once the R7 system has captured several rectified images in one location. These images are reorganized (segmented) in several units (a small image of 512×512 resolution called "tile"), these tiles are well placed in grids of a spherical bitmap and are stitched into a high resolutional panorama. In the visualization, we only access the tiles but in the backend of Street View, all the geodetic data, depth information and panorama are encoded and stored into a compact `XML` source file[1], as shown in Figure 5.3. In this way, we can access the Street View data by two methods, one is to download directly from the API, the other is to obtain the source file and decode the `XML` data.

In the literature, most researchers [129, 147, 164, 215] downloaded perspective Street Views (tiles) from the API directly, which provides an interface to access

[1] This `XML` file is Base64-encoded and zlib-compressed.

images for certain coordinates. We need to specify some parameters beforehand:

- Latitude and longitude;
- Vehicle heading, also known as global yaw;
- Vehicle pitch, a up-down angle;
- Field of view (up to 120°);
- Resolution (up to 640×640 pixels).

Take the location of the Arc de Triomphe as an example. We fix random parameters into the following API access link and get a perpective Google Street View shown as Figure 5.2: `https://maps.googleapis.com/maps/api/streetview?size=640x320&location=48.8738,2.2950&fov=120&heading=0]&pitch=0`

As we mentioned, this perspective image is one tile projected in the spherical grid of Street View panorama. Regardless of the copyright watermark, this acquisition method is still not suitable since Google APIs prohibit consecutive downloads and will block the user's IP-address if excessive accessing is detected. Therefore, we parse the available `XML` files behind the API with a Python script to extract and calculate the raw data. We can retrieve this raw data by requesting one of the following URLs: `http://maps.google.com/cbk` or `http://cbk0.google.com/cbk`. In the same way, we add some parameters to the URL: "`output`" is the format of output in `XML` or `json`; "`ll`" means latitude and longitude; "`dm`" is a boolean value to access depth map data or not and "`pm`" is a boolean value to include panorama data or not. Take the same location of the Arc de Triomphe as an instance to obtain its corresponding raw data as illustrated in Figure 5.3. Please note that the coordinates of the Street View are just the closest position to the input coordinates, namely, they will not be consistent.

`http://maps.google.com/cbk?output=xml&ll=48.8738,2.2950&dm=1&pm=1`

From the Figure 5.3, we can extract a lot of useful information about this Street View, for instance:

- Panorama pixel resolution, $13,312 \times 6,656$;
- Panorama identity no., `BbTzlrUC13Cuq2B_Eluleg`;
- Capturing global yaw, 286.62°;
- Optimized latitude and longitude, $48.874192, 2.295204$;
- Capturing date, $2014 - 8$;
- Elevation angles in different formats;
- Original latitude and longitude;
- Neighboring panorama ids and their yaw angles;
- Encoded panorama pixel and its depth map.

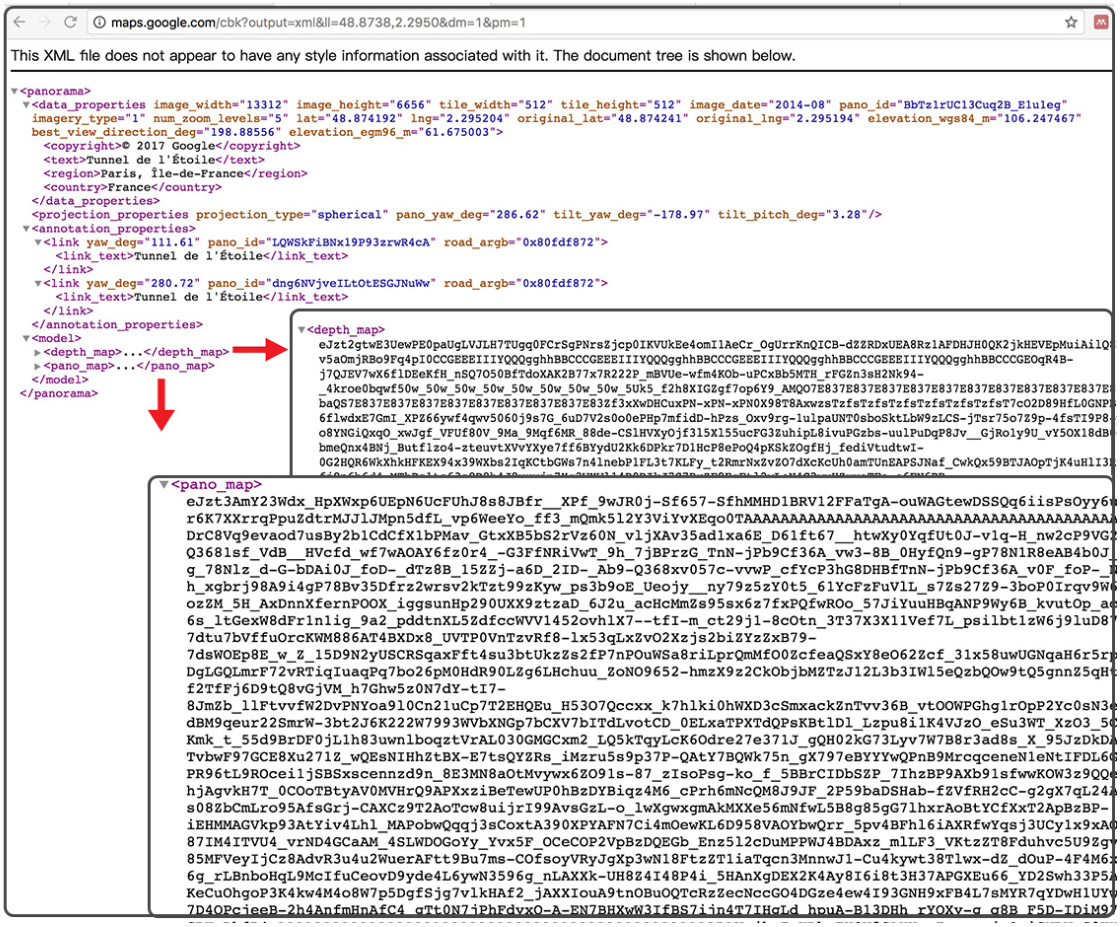


Figure 5.3: An extracted XML source file of the Street View and associated depth map around latitude= 48.8738 and longitude= 2.2950.

However, the chunk of the panoramic image and its associated depth map are still encoded. After decoding and uncompressing according the Base 64 rule, we can obtain a panorama directly but the depth map should be reconstructed. The interpreted panorama has a very high pixel resolution by capturing a 360° horizontal and 180° vertical field-of-view in a spherical projection [135, 215]. More importantly, all panoramas are geotagged with optimized GPS coordinates and cover the street in a nearly uniform successive way. The topological information of a panorama is given by the global yaw angle, which measures the rotation angle in clockwise direction around the R7 system locus relative to the true north. In fact, the GPS position of Street View is highly precise due to a careful global optimization.

As we stated, we can not retrieve a depth map directly after decoding. Actually, we only obtain some organized values, including:

- The depth map image size, the default value is 512×256[2] ;
- The total number of *planes* in a panorama;
- An integer list is assigned to every plane, know as the plane *normal vector* $\mathbf{n}$;
- The plane index to every pixel in the panorama;
- The distance *plane.d* from every plane to the camera center O, see more in Figure 5.4.

At first, we were confused about these parameters and their meaning. After analyzing the capturing system, we found that the associated depth map stores the distance denoted as *plane.d* and the orientation of various points denoted as $\mathbf{n}$ in the scene coming from laser range scans [135]. It only encodes the scene's structural surfaces (known as the *planes*) by their normal directions and distances, allowing to map building facades and roads while ignoring smaller dynamic entities. With the geometry of the planes, the index of the plane of each pixel and the distances, we can compute the real depth of every pixel. Algorithm 1 illustrates the pipeline for the depth map extraction and visualization. It is clear that the number of planes differs for each panorama.

Consider a pixel $\mathbf{p} = [x, y]$ in the depth map, with $x \in [0, w-1]$, $y \in [0, h-1]$, where w and h being the width and height of the depth map, namely 512×256. We normalize the image pixel into $[0, 1]$ space and then turn it to the yaw and pitch in the spherical coordinate system. In order to recover the ray direction $\mathbf{s}$ in Figure 5.4, we convert the spherical coordinates to Cartesian coordinates. With the plane distance and the directional vector, the dot product is used to calculate the absolute depth. Notice that all values with a depth of zero are considered as points at infinity. To visualize, we simply normalize the depth between 0 and 1 and map it to grayscale by multiplying with 255.

[2]The size is smaller than its panorama.

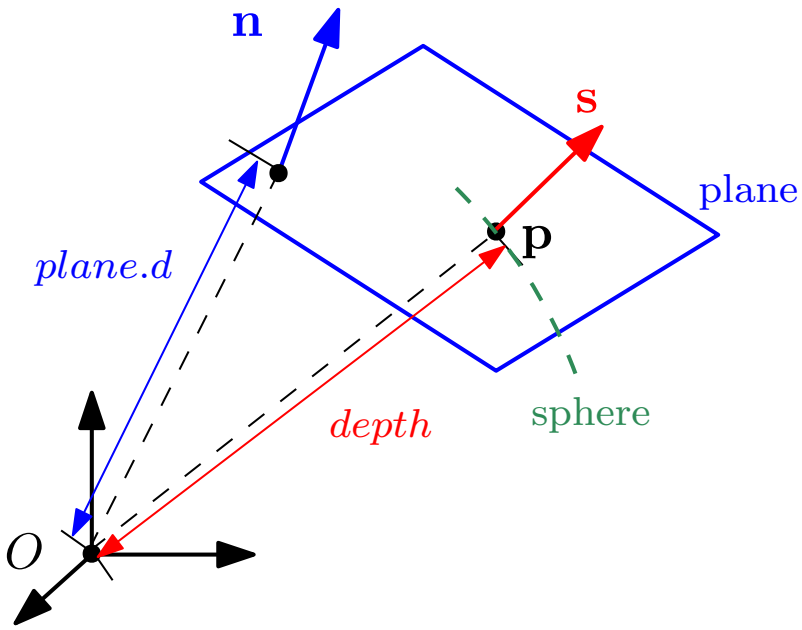


Figure 5.4: Illustration of the parameters in the depth map.

Algorithm 1 Depth Map Computing & Visualization Algorithm

Require: x, y, w: customized width; h: customized height; $Indices$; $planes$
Ensure: $depthMap$, $Intensity$

for all x, y, $indices$ **do**
  $planeIndex \leftarrow Indices[y * w/512 + x/256]$
  $yaw : \theta \leftarrow \frac{w-x-1}{w-1} * 2\pi$
  $pitch : \phi \leftarrow \frac{h-y-1}{h-1} * \pi$
  $spherical\ vector : \mathbf{s} \leftarrow [\cos(\theta)\sin(\phi), \sin(\theta)\sin(\phi), \cos(\phi)]^T$ *cf.* Equation. 5.5
  if $planeIndex > 0$ **then**
    $plane \leftarrow planes[planeIndex]$
    $depth = \frac{plane.d}{\mathbf{s} \cdot plane.\mathbf{n}}$
    $depthMap[y * w + (w - x - 1)] \leftarrow abs(depth)$
  else
    $depthMap[y * w + (w - x - 1)] \leftarrow \infty$
  end if
end for
for all x, y,$pixels$ **do**
  $depth \leftarrow depthMap[x + y * w]$
  $Intensity[x + y * w] \leftarrow depthMap[x + y * w]/100 * 255$
end for

Figure 5.5: An example of extracted Street View panorama at location $[48.801516, 2.131556]$ in the test area.

Figure 5.6: A depth map is associated with Figure 5.5 computed by Algorithm 1.

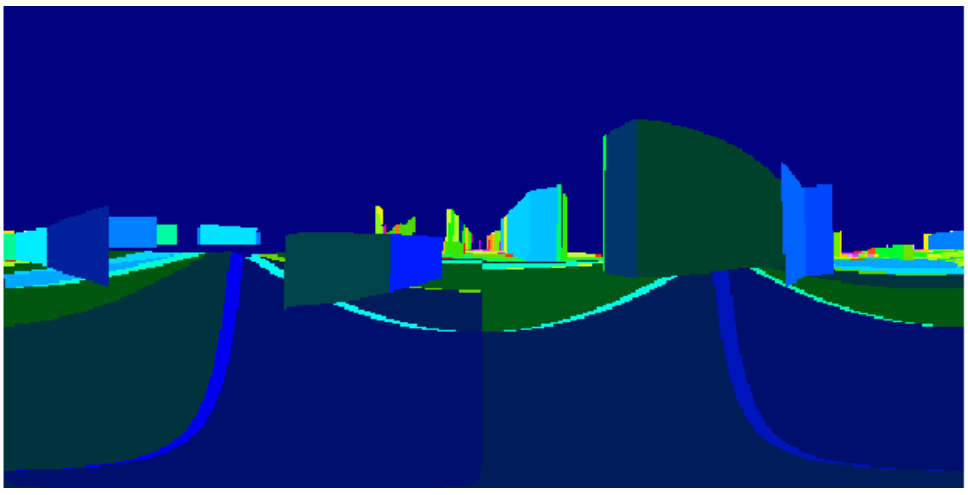


Figure 5.7: 86 different depth planes of Figure 5.6 are illustrated in colormap.

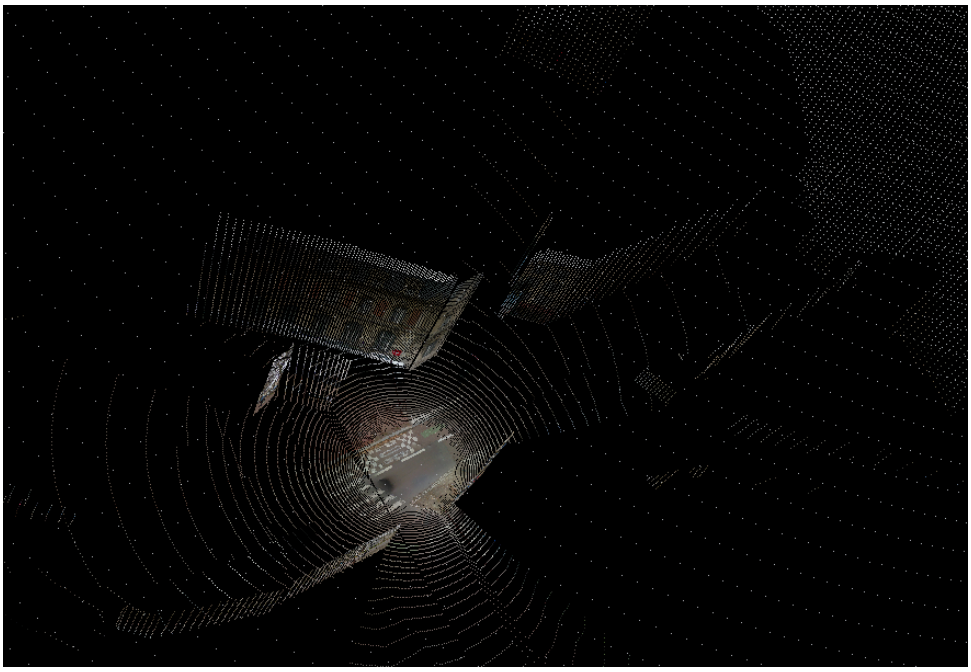


Figure 5.8: The point cloud with a $88,604,672$ size is generated from the panorama of Figure 5.5 and its associated depth map. Note that for depth with value zero, we multiply by a fixed depth 250 to visualize the point cloud.

Figure 5.5, 5.6 , 5.7 and 5.8 depict the extracted panorama, depth map, depth plane and point cloud respectively at the location $[48.801516, 2.131556]$[3] in the test area. For the sake of bandwidth saving, the depth map is sampled down to 512×256 pixels but recovering similar size to the panorama is easy. The depth map provides a coarse 3D structure of the scene with a relatively low accuracy (see more in [203]). For this reason, we neglect ineffective pixels with depth > 200m.

According to the test area of the online data acquisition, we use OSRM (Open Source Routing Machine) [85] to simulate a high-level planning of the test trajectory. Then we are able to use the planed coordinates to download all panoramic images and associated depth maps along the test trajectory. Moreover, we add a searching radius (12m in practice) to ensure all Street Views are detected in a fixed trajectory. The test area are covered by 478 useful Google Street Views in total, see Figure 5.9. Every blue blob stands for a discrete location where a Street View is available.

5.2.2 Assumptions and Challenges

The quality of the offline database construction strongly depends on the Street View GIS. As such, some assumptions on the GIS must be fulfilled.

- Adequate spatial availability of Street View image is required. According to our work, the average distance between consecutive panoramas is around 6 to 16m in Google Street View. It may differ a lot in one-way, two-way lanes or at traffic corners. Indeed, the Google car can pass several times and record

[3]Please note in the remainder of the thesis, all geodetic locations are represented in a form $[latitude, longitude]$ with decimal units.

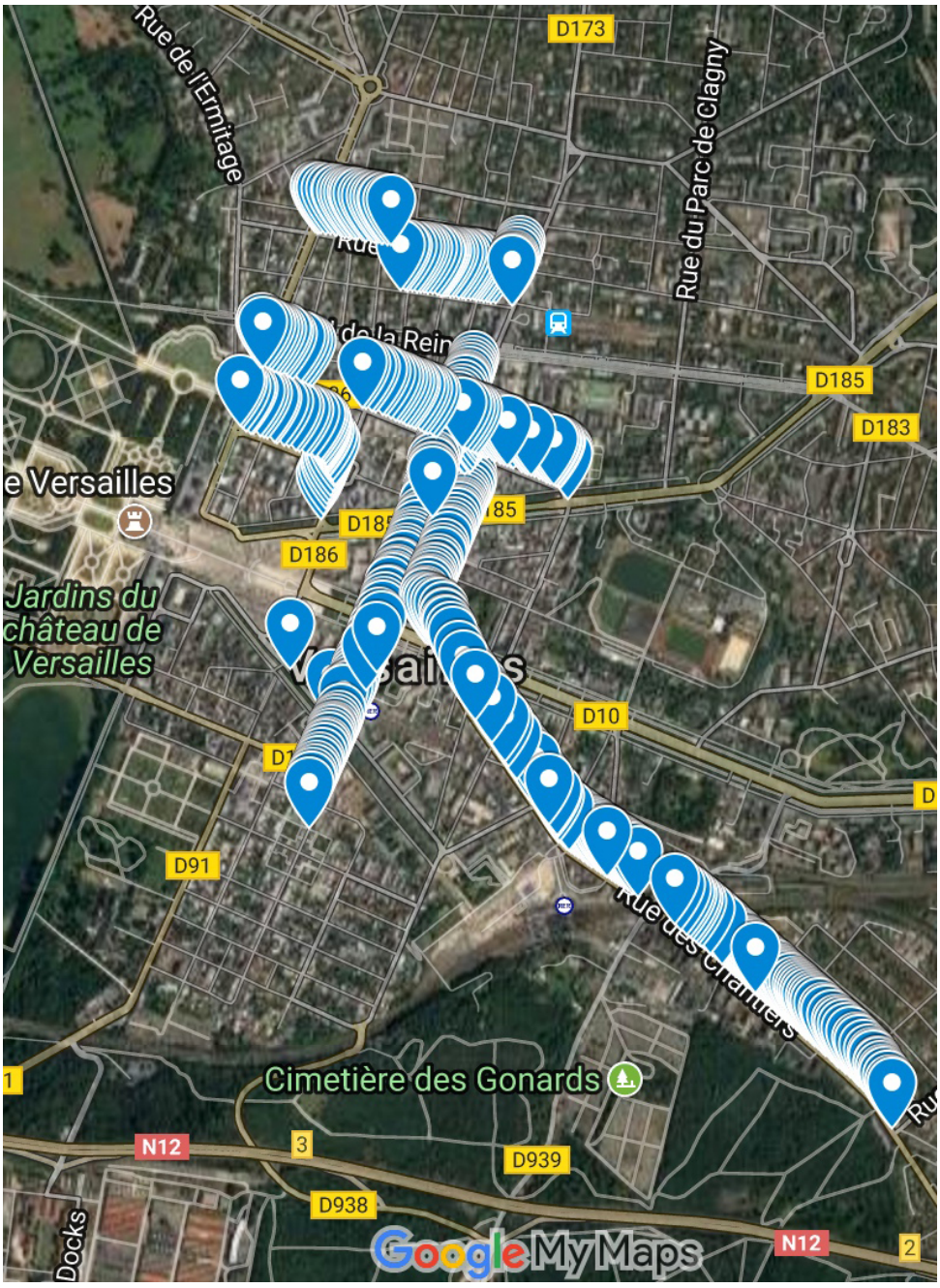


Figure 5.9: Illustration of the coverage of Google Street View in the test area.

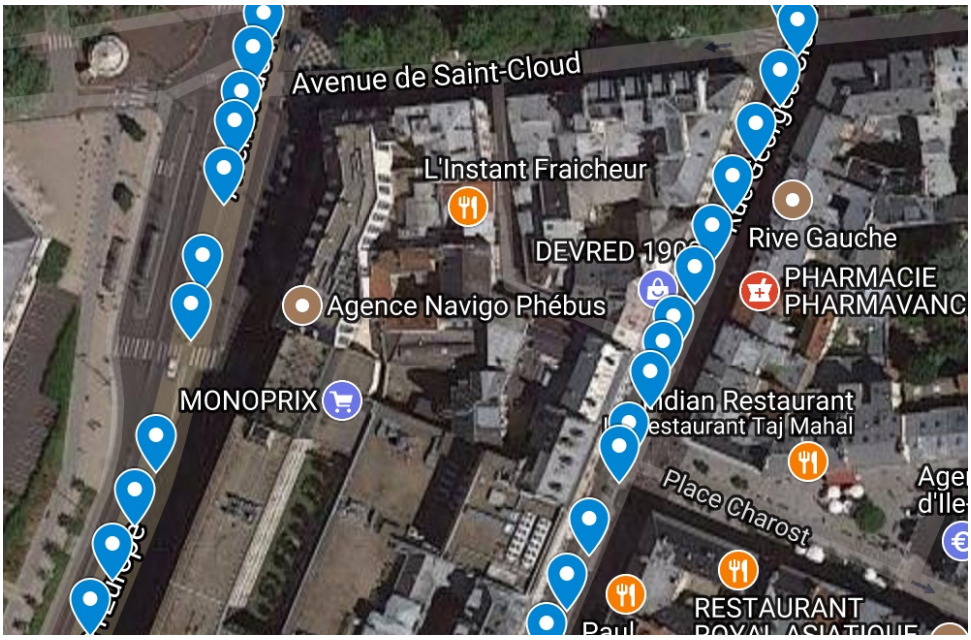


Figure 5.10: The Street View panoramas are distributed with a nearly uniform distance in the test area.

more panoramas in two-way lanes or at intersections. In the city of Versailles, we suppose that this assumption is satisfied, as shown in Figure 5.10.

- The most recent Street View data are used in our experiment. During the test, we found out that Google updates its Street View frequently (at least once a year) at Versailles. Google also combines panoramic images with different capturing dates together, for example Figure 5.5 was captured in November of 2016 but its neighboring panorama at location $[48.80056, 2.136449]$ in Figure 5.11 was recorded in July.

- We consider the geo-reference of the Street View as accurate enough to be used for localization. We verified the geo-position of Street View with the RTK-GPS and confirm that the global position of Street View is precise as Google stated in [6], while it exists slight deviations between the depth map and panorama, as illustrated in Figure 5.12. This phenomena appears particularly in the sub-urban or broad area.

5.2.3 Panorama Transformation

After obtaining the raw dataset from the GIS, it is necessary to make sure that the dataset is compatible with the online acquired data. It is obvious that panoramas differ from our online mono-camera images in both size and visual appearance. According to the state-of-the-art in Chapter 3, a direct way to estimate the pose from different types of imagery exists neither in handcrafted-feature-based nor learning-based methods. A pre-processing of the raw data is thereby necessary to construct an offline database compatible with the online dataset. To succeed in the metric localization, panoramas can be transformed into a set of overlapping or unrelated

Figure 5.11: A Street View panorama at location $[48.80056, 2.136449]$ is extracted in summer in the test area.

Figure 5.12: Example of the discrepancy between panorama and depth map of Figure 5.5. The red area illustrates the coverage of depth map without value 0.

cutouts (rectilinear images) to remove the large angle distortion, coming from the generation of the panorama.

Street View, as a panoramic image in the spherical coordinates, is stitched from plenty of small perspective images as mentioned before. Our task is to recover customized perspective images from the panorama. This pipeline is often referred as image transformation.

Before diving into the details of various projection and coordinate changes, we firstly introduce some foundations used in the image transformation, specially about warping function and interpolation.

Warping function and Interpolation

Let us consider an image $\mathbf{I}_1$ with a $m \times n$ pixel resolution in the camera frame F_1. The pixel position $\mathbf{p}_1 = (u, v)$, with $u \in [0; m[$ and $v \in [0; n[$, is associated to the image's intensity function, denoted as $\mathbf{I}_1(\mathbf{p}_1)$. Suppose that for every pixel $\mathbf{p}_1$, the depth along the optical axis (λ in the Equation 2.10) is well known. As such, the 3D point in this frame $\mathbf{P}_1$ can be defined as $\mathbf{P}_1 = f(\mathbf{p}_1, \lambda)$ when given camera intrinsic parameters. Hence the set $\mathbf{S} = (\mathbf{I}_1, \boldsymbol{\lambda})$ is called the *augmented image*, including both image intensity and depth map $\boldsymbol{\lambda}$.

Now we have another camera that observes the same scene from a user fixed point of view in the camera frame F_2, with a transformation matrix as $\mathbf{T} \in \mathbb{SE}(3)$ relative to the frame F_1, as shown in Figure 5.13. If the pose $\mathbf{T}$ is given, it is possible to synthesize a new image $\mathbf{I}_2$ from the image $\mathbf{I}_1$ in the frame F_2 with a warping function [181], denoted as:

$$\mathbf{p}_2 = w(\mathbf{T}; \mathbf{S}) : \mathbb{SE}(3) \times \mathbb{R}^3 \to \mathbb{R}^2 \tag{5.1}$$

The function $w(\cdot)$ transfers the pixel $\mathbf{p}_1$, associated with the 3D point $\mathbf{P}_1$, to the synthesized image $\mathbf{I}_2$ with a rigid transformation $\mathbf{T}(\mathbf{R}, \mathbf{t})$, and then followed by the corresponding projection, such as the perspective (*cf.* Equation 2.10), conic, spheric, cylindric model. Indeed, the given image $\mathbf{I}_1$ can also be projected from these different models. Please note that for the simplification, we do not define the projection in the warping function but it is obligatory. The process to render a new image from an augmented image is called "*image synthesis*", the new image is known as the "*virtual image*". There is a special case when $\mathbf{T}(\mathbf{R}, \mathbf{t} = 0)$. It means that the camera in the frame F_2 stays in the same position of the camera of the frame F_1 but the rotation is changed. This special case is also known as "*back projection*".

In general, the point coordinates $\mathbf{p}_2$ hardly lies into a pixel directly, in other words $\mathbf{p}_2 \notin \mathbb{N}^2$. The intensity of the synthesized image ought to be interpolated from the corresponding intensity of $\mathbf{p}_1$, as follows.

$$\mathbf{I}_2(\mathbf{p}_2) = \mathbf{I}_1(\mathbf{p}_1) \tag{5.2}$$

There are many ways to interpolate image pixels in the literature, like the fast nearest, bilinear and bicubic algorithms. In our work, we adopt the bilinear interpolation for its smooth performance and fast computation.

The bilinear interpolation [154] looks for a pixel's intensity with its 4 nearest pixel values that are located in diagonal directions (see Figue 5.13). We define a function to render the integer value from a real number as, $int(\cdot) : \mathbb{R}^2 \to \mathbb{N}^2$. A

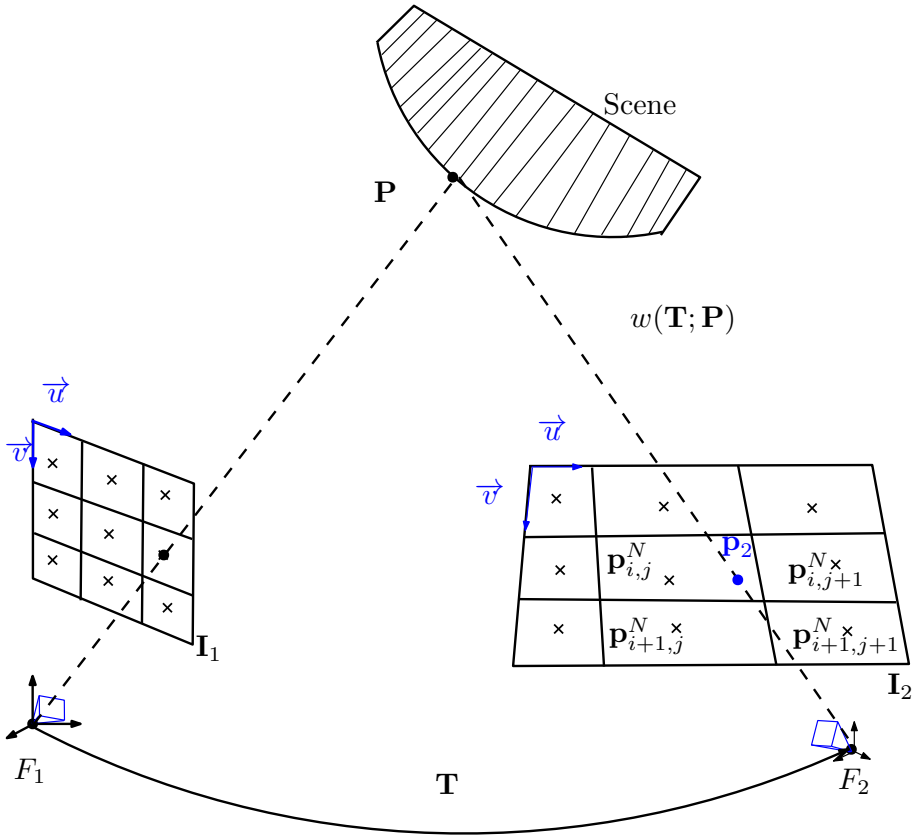


Figure 5.13: Transformation of camera in the scene. $\mathbf{I}_1$ is an augmented image in the frame F_1 with depth information. We are able to synthesize a new image $\mathbf{I}_2$ in the frame F_2 by bilinear interpolation.

supposed pixel position is defined by $\mathbf{p}^N = floor(\mathbf{p}_2)$ [4]. The coefficients of the bilinear interpolation are defined as:

$$\boldsymbol{\alpha} = \mathbf{p}_2 - \mathbf{p}^N = [\alpha_u, \alpha_v] \in [0; 1] \tag{5.3}$$

The pixel value interpolated from the coordinates $\mathbf{p}_2$ is defined $\mathbf{p}^N_{i,j}$ as:

$$\mathbf{I}(\mathbf{p}_2) = \begin{bmatrix} 1-\alpha_v, & \alpha_v \end{bmatrix} \begin{bmatrix} \mathbf{I}(\mathbf{p}^N_{i,j}) & \mathbf{I}(\mathbf{p}^N_{i,j+1}) \\ \mathbf{I}(\mathbf{p}^N_{i+1,j}) & \mathbf{I}(\mathbf{p}^N_{i+1,j+1}) \end{bmatrix} \begin{bmatrix} 1-\alpha_u \\ \alpha_u \end{bmatrix} \tag{5.4}$$

Panorama Backprojection

In our case, the image $\mathbf{I}_1$ is a Street View panorama with a spheric projection. We aim at transforming this panorama into a set of overlapping or unrelated cutouts (rectilinear images) at the same position. Hence, it is the special case in the image warping. As such, we build a back projection model by standard ray tracing with bilinear interpolation to extract perspective images. With this model, all panoramas are .

We assume several virtual pinhole cameras, for instance 6 cameras, with the intrinsic calibration matrix $\mathbf{K}$, mounted at the centre of the spherical coordinates $\mathbf{S}$ with a user fixed pitch δ and roll ζ, local yaw angle η_{local}[5] changing by $[0°, 60°, ..., 360°]$. The number of virtual cameras, intrinsic matrices and heading, pitch, yaw angles are free to select, yet empirically the more identical they are to the actual on-board monocular camera, the better performance expected.

Suppose a 3D point $\mathbf{P} \in \mathbb{R}^3$ defined in the sphere coordinates $\mathbf{S}$ at point O (*cf.* Figure 5.14). The projection of this 3D point on the unit sphere is represented by $\mathbf{p}^*$. It can also be projected directly on a virtual camera image plane $\mathbf{I}(\mathbf{p})$ with a pixel value interpolated from $\mathbf{p}^*$.

The 3D point $\mathbf{P}$ in homogeneous coordinates using spherical parametrization (θ, ϕ, ρ) with $\theta \in [0, 2\pi]$ and $\phi \in [0, \pi]$, is denoted as:

$$\overline{\mathbf{P}} = \begin{bmatrix} x_S \\ y_S \\ z_S \\ 1 \end{bmatrix} = \begin{bmatrix} \rho\cos(\theta)\sin(\phi) \\ \rho\sin(\theta)\sin(\phi) \\ \rho\cos(\phi) \\ 1 \end{bmatrix} \tag{5.5}$$

It is easy to deduce the conversion relationship between the Cartesian and the defined spherical coordinates, as follows.

$$\begin{bmatrix} \theta \\ \phi \\ \rho \end{bmatrix} = \begin{bmatrix} \arctan 2(\frac{y_S}{x_S}) \\ \arctan(\frac{\sqrt{x_S^2+y_S^2}}{z_S}) \\ \sqrt{x_S^2+y_S^2+z_S^2} \end{bmatrix} \tag{5.6}$$

[4]For example, given $\mathbf{p}_2 = [50.11, 60.99]$, $\mathbf{p}^N_{i,j} = floor([50.11, 60.99]) = [50, 60]$. $\mathbf{p}_2$ will not lie on the edges of a panoramra since the back projection is a centering downsample.

[5]Please distinguish: the local yaw angle η_{local} is the angle from which we extract an image from the panoramic image and the global yaw angle η_{global} in the remainder is the orientation of the vehicle when taking the panorama.

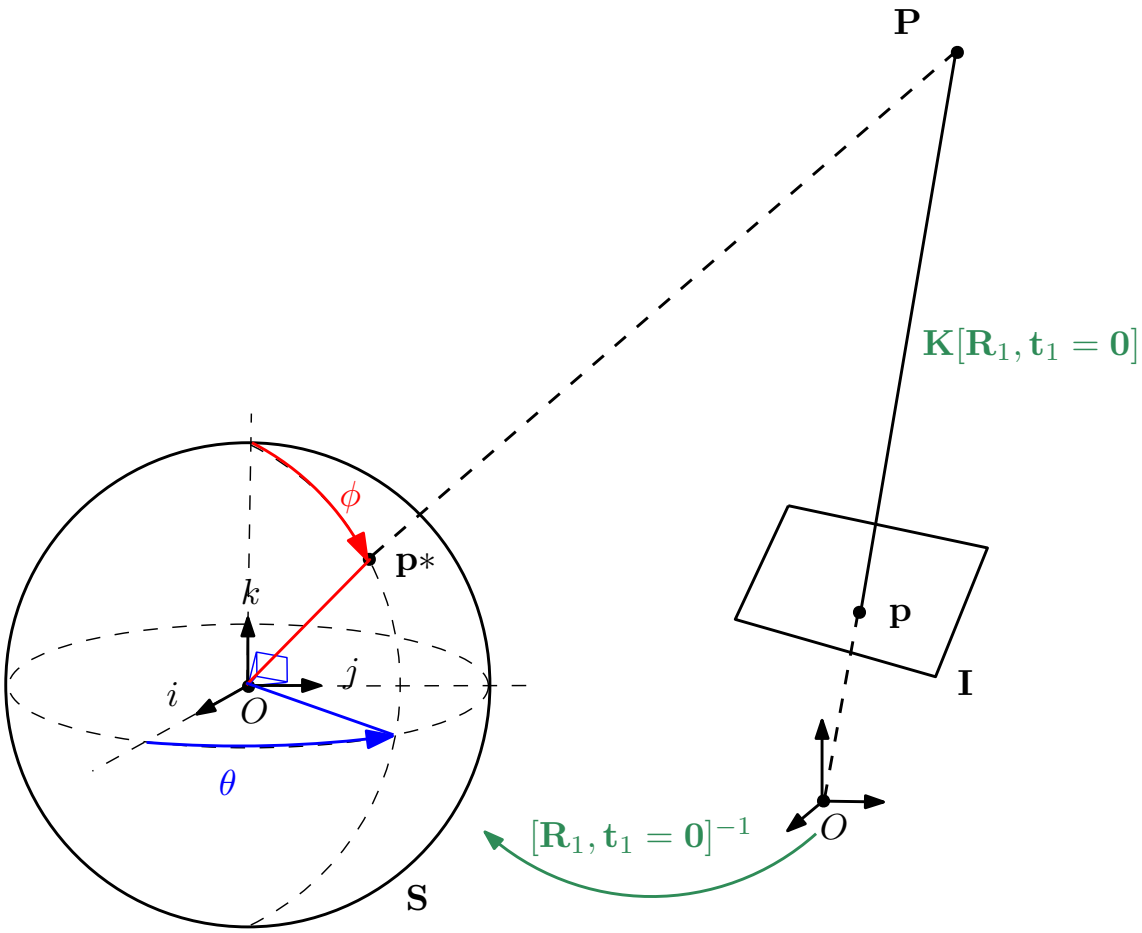


Figure 5.14: Back-projection model: virtual cameras are constructed at the point O and pixels in the image plane $\mathbf{I}$ are bilinearly interpolated from the panorama sphere. The local yaw offset changes according to the direction of θ.

Therefore, the projection in a unit sphere is represented as,

$$\overline{\mathbf{p}^*} = \frac{\mathbf{P}}{\|\mathbf{P}\|} = \begin{bmatrix} \cos(\theta)\sin(\phi) \\ \sin(\theta)\sin(\phi) \\ \cos(\phi) \\ 1 \end{bmatrix} \tag{5.7}$$

According to the Equation 2.10, the perspective projection is represented as:

$$\mathbf{p} = \mathbf{K}\frac{\mathbf{R}_1(\delta,\zeta,\eta_{local})\mathbf{P}}{z(\mathbf{R}_1(\delta,\zeta,\eta_{local})\mathbf{P})} = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \frac{\mathbf{R}_1(\delta,\zeta,\eta_{local})\mathbf{P}}{z(\mathbf{R}_1(\delta,\zeta,\eta_{local})\mathbf{P})} \tag{5.8}$$

with the focal length f and the principal point (u_0, v_0). We choose the convention that the z axis is the optical axis and normalize rotated 3D points to the unit plan $(z = 1)$. These intrinsic parameters are fixed according to our on-board camera. The camera extrinsic matrix is deduced from the above configuration, namely the local heading, pitch and yaw setup. $\mathbf{R}_1$ returns a rotation matrix through the Rodrigues' formula.

As stated in the Equation 5.2, the intensity of virtual images can be sampled

from the warping function with

$$\mathbf{S}(\mathbf{p}^*) \xrightarrow{w} \mathbf{I}(\mathbf{p}) \tag{5.9}$$

Due to the high resolution of the Street View panorama, we can interpolate values of intensity in virtual images perfectly, as illustrated in Figure 5.15 and Figure 5.17. The corresponding perspective depth map can be addressed likewise, see Figure 5.16 and Figure 5.18. It is obvious that the more rectilinear images are back projected, more overlapping views and detailed scenes appears in the result.

Figure 5.15: Rectilinear images are back projected from the panorama in Figure 5.11 with local yaw angle η changing by $[0°, 60°, ..., 360°]$. The virtual cameras' FoV, focal length and resolution are all the same as the online camera MIPSEE.

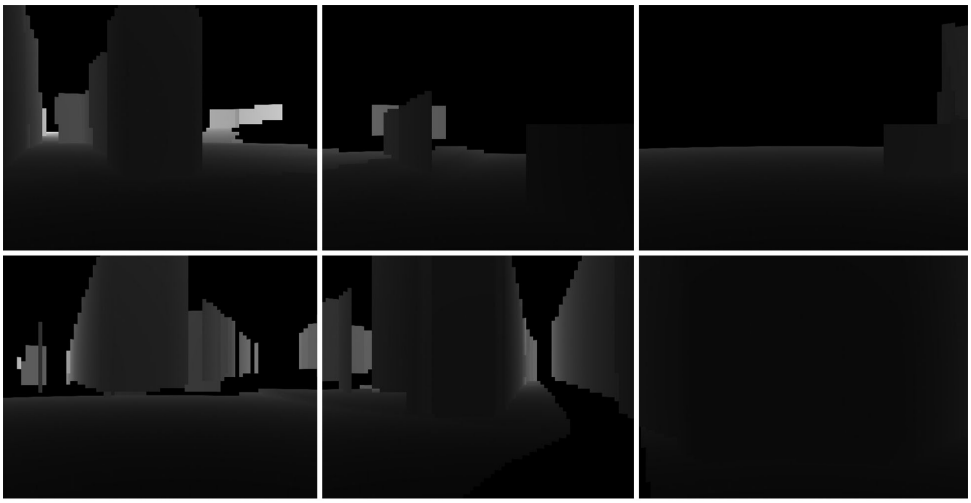


Figure 5.16: Rectilinear depth maps are associated with Figure 5.15.

As we known, most of related researches rely on Google API to generate perspective images by setting a virtual camera's parameters, such as FoV, pitch and yaw. Compared with the provided API, this method works in a more robust and flexible way, since we can set the intrinsic parameters of the visual camera as exactly those of the monocular camera employed for online dataset acquisition.

Figure 5.17: Rectilinear images are back projected from the panorama in Figure 5.11 with local yaw angle η_{local} changing by $[0°, 30°, ..., 360°]$.

5.3 Topometric Representations

So far, we have discussed how to choose our GIS, how to extract useful data from it and how to make the data fit to our online datasets. Now we will answer the final question in the offline database construction: how to use the data to represent the urban environment.

In the Chapter 3, we reviewed localization approaches taking advantages of Street Views and some robot-accessible representations of the environment from the SLAM. Inspired by the learning phase in [132], the given environment can be modeled as a database learned in advance. We propose a compact topometric representations of a given urban environment: all Street Views are represented by nodes, whose connecting edges contain the topological information by the global yaws (see Figure 5.19). Moreover, the geo-references of Street Views act as supplementary information to enhance the robustness of the localization. Through this model, three forms of information are extracted from the dense environment:

- Dense image information from Street View: some popular algorithms, such as the structure from motion, enable us to obtain the pose of a vehicle; the problem turns to realize matching between the nonstandard Street View imagery and image sequence acquired by an online vehicle.

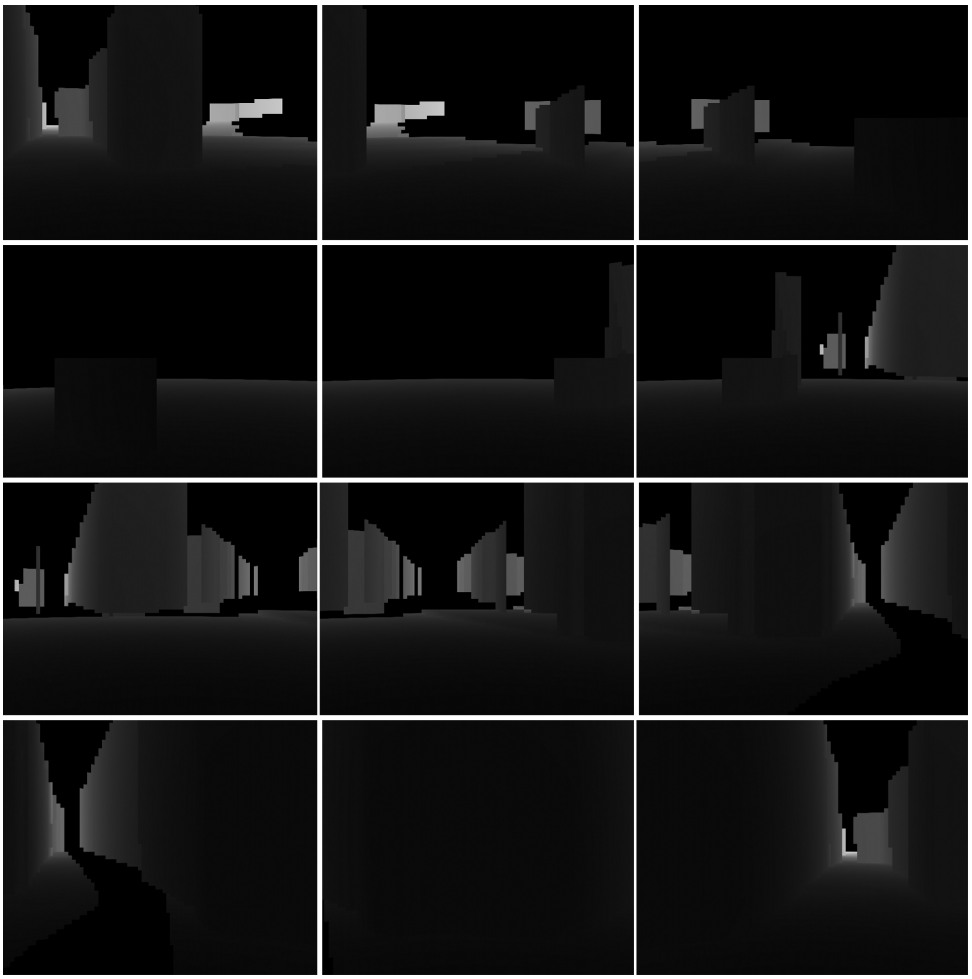


Figure 5.18: Rectilinear depth maps are associated with Figure 5.17.

- Direct geo-reference of Street View: namely latitude and longitude coordinates, its precision is expressed in decimal degree[6]. Google Street View achieves a very high precision (6 decimal places and 111.32 mm error at the equator, and normally in urban city such precision can be guaranteed.)[7]. For a vehicle urban localization, such precision is completely sufficient.

- Topology information: edges can represent the topology of urban streets if nodes are set up appropriately. The global yaw degree η_{global} plays an important role in the topological representation. Because in city-like environment the topology limits the vehicle's motion boundary and thus allows to predict and prepare the next Street View reference. In this way, we only need to download local environment information instead of global cities.

An interesting aspect will emerge according to our proposed model, we call it as the triangular error consistency problem (see Figure 5.20). It is understandable that our own vehicle can find several Street View nodes around it. Hence, we are able to obtain several rigid transformations $[\mathbf{R}_i, \mathbf{t}_i]$ among the triangular relationships. Thus, how to leverage several Street Views and find the coherence from this

[6]Decimal Degree: `http://www.wikipedia.org/wiki/Decimal_degrees`
[7]*Ibid.*

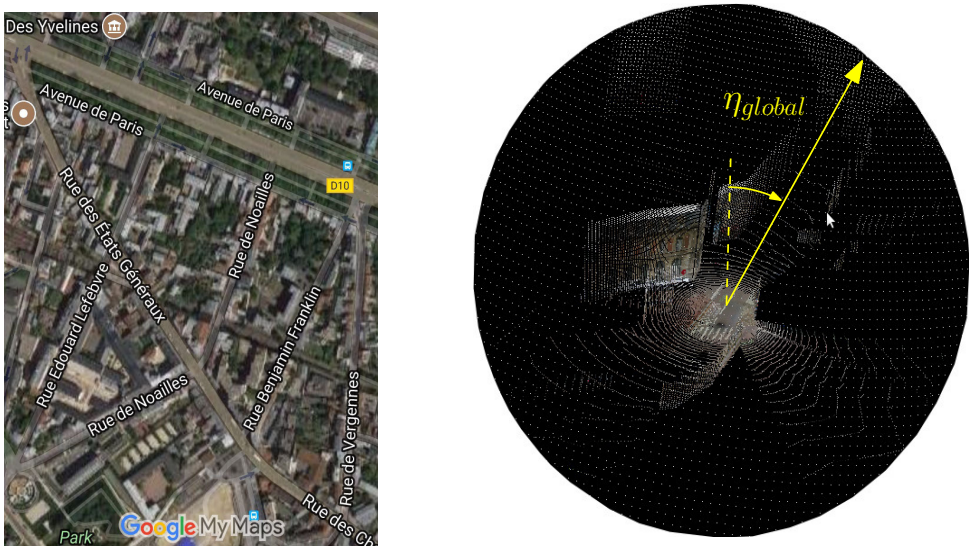


Figure 5.19: Useful information extracted from Google Street View: map topology and augmented spherical image.

triangular relationships to determine the final pose, will also be an important task in the following sections.

5.4 Conclusion

Using an existing database to represent a given urban environment is a highlight of innovation of our thesis. We described why we choose Google Street View as our database, what major data structure it contains, how the datasets are extracted and augmented, how we adapt the data for our online acquisition and what kind of representation we construct in a given environment. These proposed methods are not only based on the state-of-the-art but also our own trial and error. Once the online and offline data are well prepared, we are going to deal with the pose estimation problem from two main aspects: handcrafted-feature-based and learning-based methods.

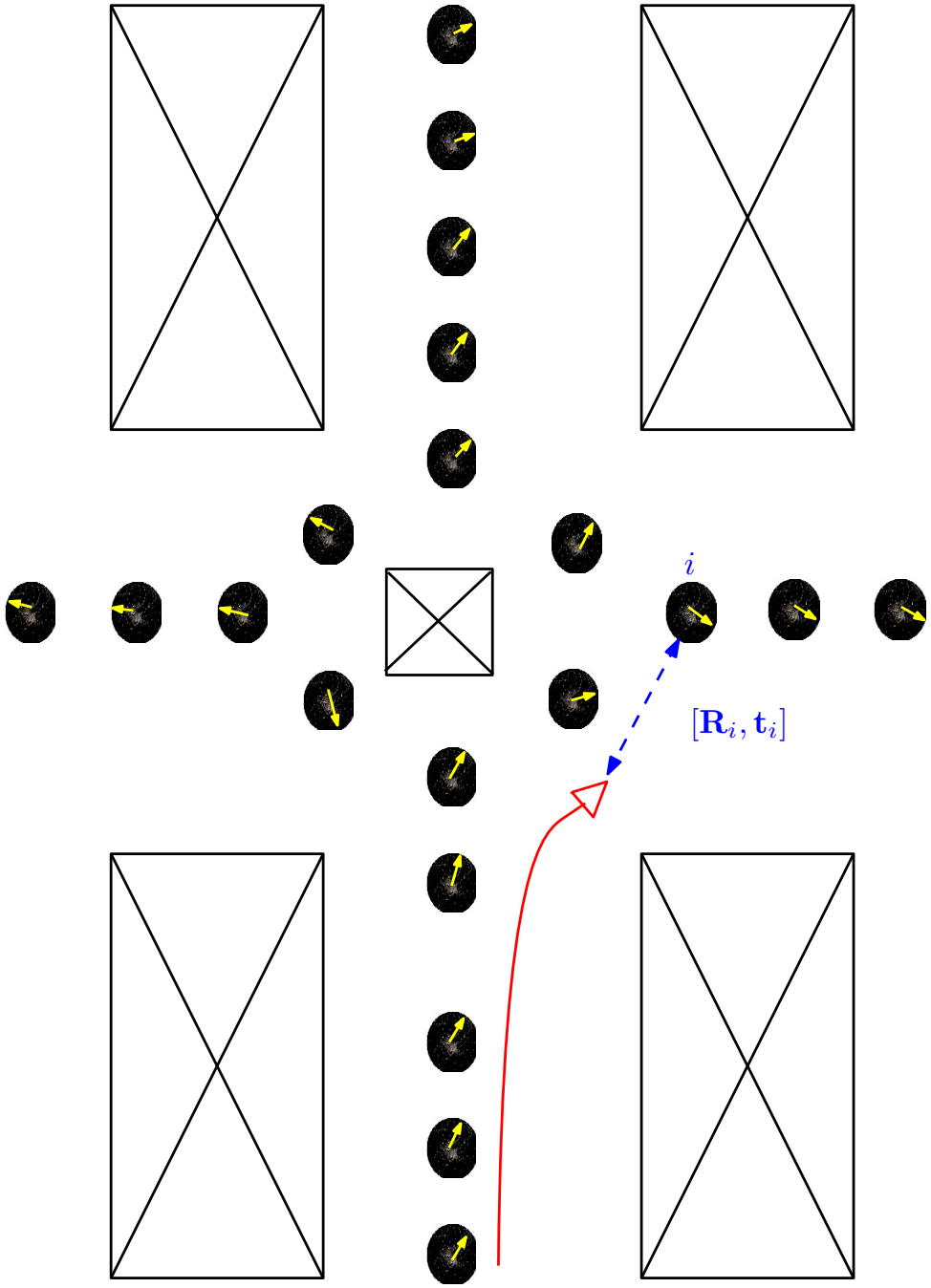


Figure 5.20: Google Street View data are used to model a given environment: consecutive panoramas and depth maps are constructed as geo-tagged augmented spheres with dense visual and metric information; the global yaw angle is used to represent the topology of trajectory.

Part III

Localization using Handcrafted Features

Chapter 6

Handcrafted Features for Urban Localization

6.1 Introduction

Handcrafted features refer to interesting image primitives (*e.g.* points, curves, edges and small image patches) and image structures (*e.g.* color, intensity and textures). The image primitives, also known as *local features* or *keypoints*, are not only used to find correspondences between images with important changes in viewpoints, occlusions and wide baseline matching, but also offer image-patch representations in the image retrieval. The image structures, also called *global features*, allow to describe the image content as an entire object instead of compact patches. Normally, global features are often used for lower level applications, such as object detection, while local features are widely applied in object recognition[1] and also play a key role in our following localization systems.

Research on local feature extraction and description has started since 1954 [125] and till now we have various feature detectors and descriptors in the literature. They focus on different primitives, invariances, accuracy, efficiency and robustness. How to choose suitable feature detectors and descriptors remains the top priority for a pose estimation task. Thereby, in Chapter 6, we first clarify our localization problem and objectives, then we analyze several challenging scenarios by testing some handcrafted features. These scenarios are useful to evaluate the performance of various features and select the potential ones. After choosing the feature extraction method, the details about the positioning system will be carried out in the Chapter 7.

6.2 Formal Problem Description

As we stated in Part II, we captured online data along a 5km trajectory with 478 Street View panoramas. The main goal is to find an algorithm to map each captured image (query image) to its corresponding Google Street View images (referenced images) and then estimate the pose from these correspondences. We call this process a

[1] "Object detection" and "Object recognition" are often mistaken to be the same thing but there is a distinct difference between them: "object detection" refers to detecting the presence of a particular object (probably unknown) in a given image while "object recognition" is a process of identifying an object in a given image.

corse-to-fine localization system with a combination of place recognition and structure based image localization. The formal problem is described as a prototype in Algorithm 2.

Algorithm 2 Input & Output of Handcrafted-feature-based Localization Algorithm

Input: Rectified images from Street View $\mathbf{R} = \{\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_n\}$ and their depth maps $\mathbf{D} = \{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_n\}$

Input: Query images $\mathbf{Q} = \{\mathbf{Q}_1, \mathbf{Q}_2, \ldots, \mathbf{Q}_m\}$ captured by a vehicle driving in the city

Output: For a query images $\mathbf{Q}_i$, the good matches $f(\mathbf{Q}_i) \in \mathbf{R}$ is find, where f is a surjective mapping function, like the description in Figure 3.2.

Output: Global pose of the query image: $solvePnP(\mathbf{Q}_i, f(\mathbf{Q}_i), \mathbf{D}(f(\mathbf{Q}_i))$

In this method, a query image captured by our camera equipped on the vehicle is associated with geotagged Street View images sharing high appearance similarities. As such, the vehicle's position is restricted around an intersected area geotagged by a matched Street View. At this stage, an efficient and robust retrieval method should be developed, which should work well even if high intra-similarities exist in Street View imagery. As we stated before, our goal is to identify, for a given query image, Street View images with a similar appearance. Since this stage does not require a metric accuracy, but only to identify the closest Street View images, the mapping function (namely $f(\mathbf{Q}_i) \in \mathbf{R}$ in the Algorithm 2) should render several proper candidates. Thus, classic place recognition methods like SIFT-BoW, VLAD or Fisher Vectors, FabMap will be good options as the mapping function as we stated in Chapter 3.

Secondly, since the depth maps are integrated in Street Views, we can refine the before-mentioned location by computing the vehicle's pose transformation *w.r.t* Street View images by solving the PnP problem[2]. With the pose transformation, the query image can be localized from the geo-reference of Street View directly. According to the distribution of Street View panoramas, we can deduce that many query images will share the same Street View. If we can extract sufficient correct feature correspondences, the function $solvePnP(\mathbf{Q}_i, f(\mathbf{Q}_i), \mathbf{D}(f(\mathbf{Q}_i))$ should be able to compute consistent poses for query images sharing the same referenced Street View.

6.3 Challenges in Image Search and Localization

In the data acquisition process, we have already ignored some trajectories without important urban appearances. Yet, difficulties still exist for both coarse place recognition and structure based localization. The core idea of our method is to find detectable and discriminable locations of handcrafted feature in both query and referenced images. Before giving further insights in feature extraction and matching, we explore both online and offline data to outline the challenges of the underlying image-search between two databases.

[2]Recall the "Solving the n-Point Perspective problem" in Chapter 3: given a set of n ($n \geq 3$) 3D points in a world reference frame and their corresponding 2D image projections as well as the calibrated intrinsic camera parameters, determine the 6 DOF pose of the camera in the form of its rotation and translation with respect to the world.

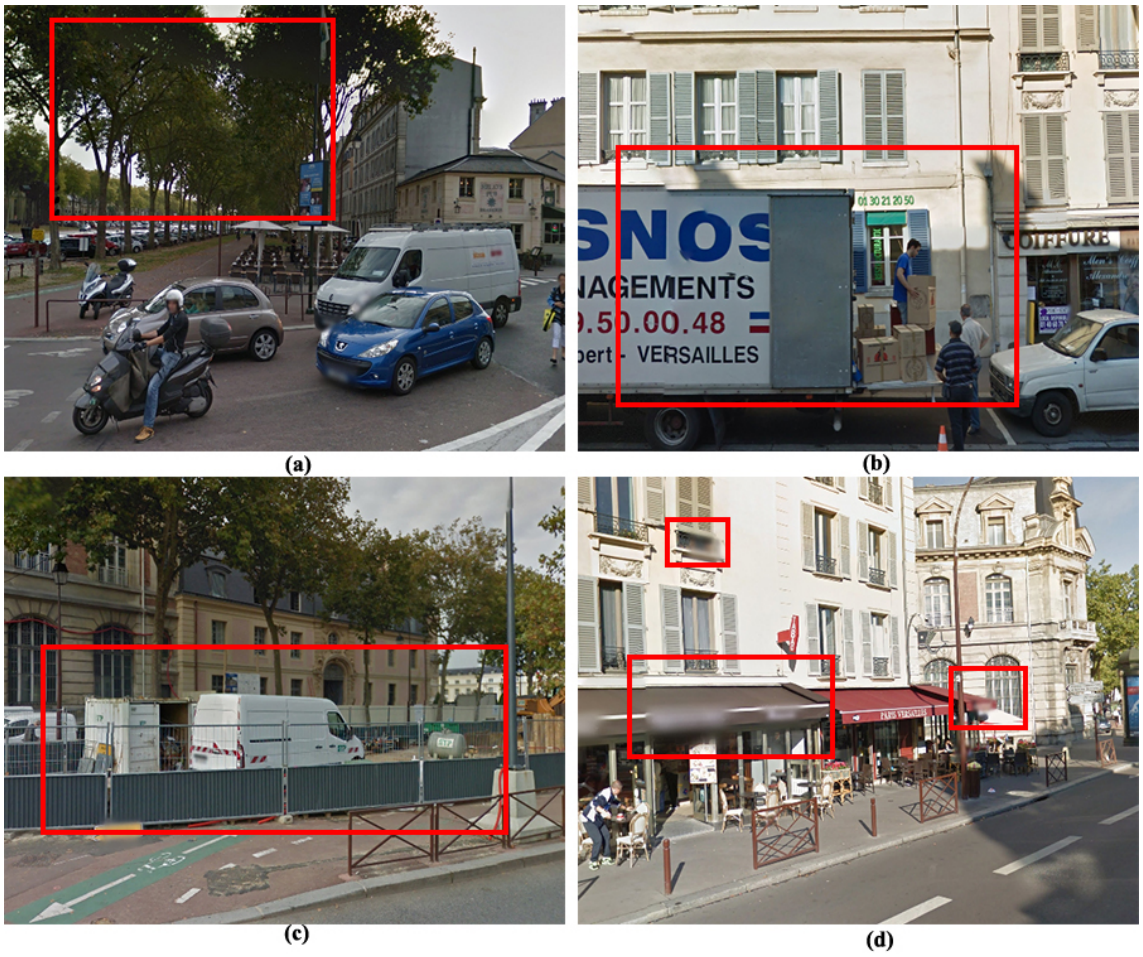


Figure 6.1: Example of noises in the Google Street View dataset: vegetation coverage, moving cars, temporary constructions and privacy blurs.

- **Noises in Datasets.** The urban environment is very dynamic and contains a lot of noise, *e.g.* moving cars, pedestrians and trees, which makes a robust searching and matching difficult. For offline data, except these dynamic objects, Google also blurs some parts of images containing car number plates, human faces and shop names in order to protect privacy and anonymity, see in Figure 6.1. In online data, the setup of acquisition can cause other disturbances. The position of the camera inside the Zoé car can capture reflections on the front window and generate bad illumination issues, as illustrated in Figure 4.5 in Chapter 4.

- **Outdated Street Views.** The Google Street View database is composed of images recorded in different seasons and years. For instance, it exists panoramas captured in both summer of year 2014 and winter of 2012 for our test area (*cf.* Figure 6.2), whereas the online dataset was obtained in 2015. It will lead to the problems of over-season changes (*e.g.* substantial vegetation and illumination differences) and the structural changes in the environment (*e.g.* building construction and destruction).

- **Self-similar and Repetitive Structures.** Man-made geometric and repetitive structures are very common in urban environments, including the win-

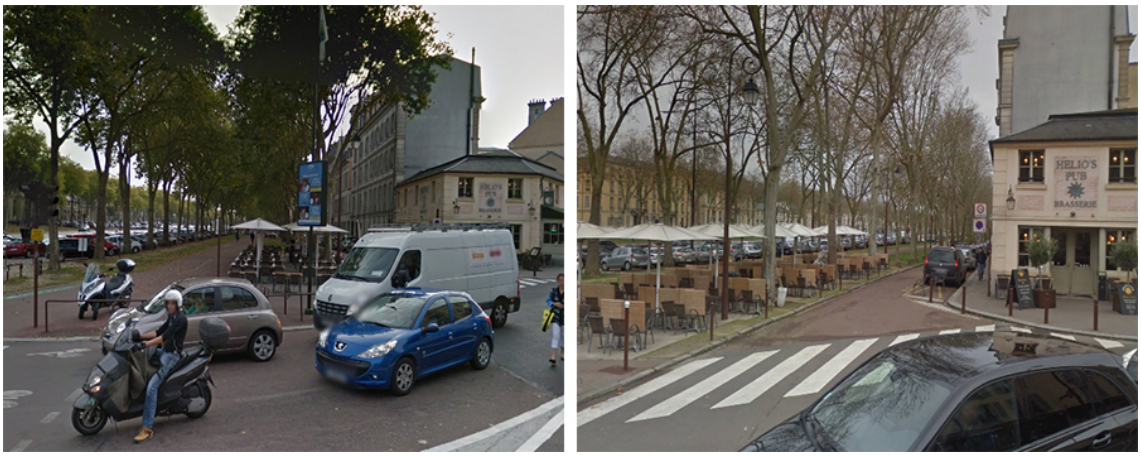


Figure 6.2: Over-season imagery is mixed in the Google Street View dataset.

dows, doors and even whole facades and buildings, see Figure 6.3. These identical elements are one of the main sources that can lead to failures in image searching and feature matching. If the coarse image searching fails, the metric process of feature matching is meaningless.

- **Intra-similar Street Views.** Most image searching algorithms use a winner-takes-it-all scheme meaning that the most similar referenced image is the one with the highest number of features in common with the query image. However, identical scenes are captured by several subsequent panoramas at different scales, as demonstrated in Figure 6.4. This intra-similarity can break up this kind of image searching scheme due to identical scenes. In fact, this challenge is similar to the triangular error consistency problem as mentioned in Chapter 5.1. We should think carefully to design the localization algorithm: either by adopting the best candidate from winner-takes-it-all scheme, or leveraging several similar offline candidate images.

6.4 Handcrafted Features Correspondences

Once the referenced images are found, the standard process for image localization is to match query and reference images with local feature descriptors: detecting feature points, extracting descriptors around feature points, matching descriptors and selecting inliers (ratio test, geometric consistency check, double check). In this section, we dive into some properties of current features and the matching process. The global localization accuracy is obtained from the transformation matrix using the good correspondences between the online query and Street View datasets. A suitable feature is crucial to obtain proper correspondences. Therefore, we manually select some challenging urban scenarios as samples to evaluate state-of-the-art features and matching techniques.

6.4.1 Feature Extraction and Description

In computer vision, keypoints are detected and extracted from two images to be matched by characterizing distinctive and salient local patches. Then, for each

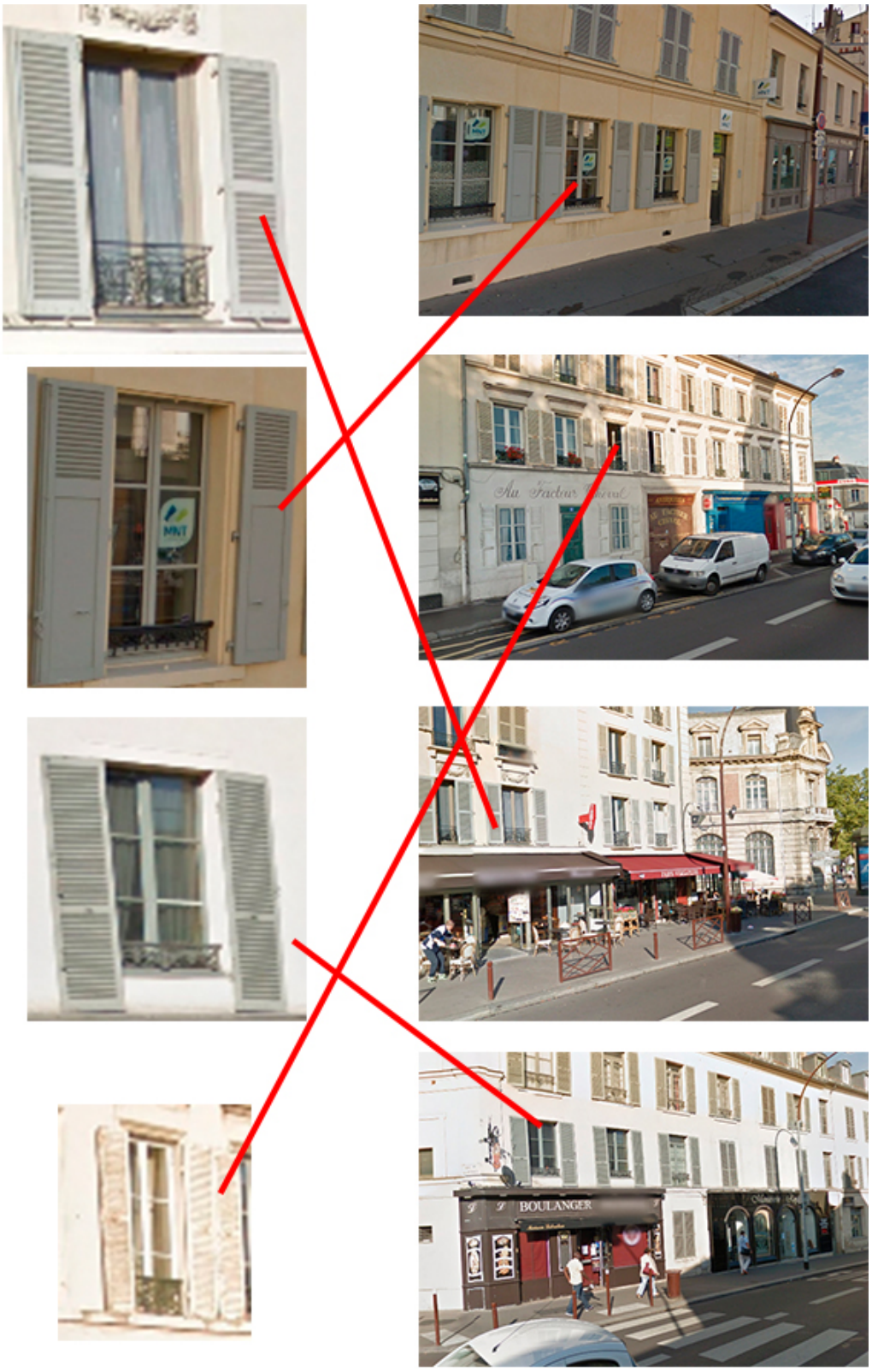


Figure 6.3: Example of repetitive windows from different Street Views.

Figure 6.4: Intra-similarity of three different Street Views in the offline dataset.

Local Feature Category	Methods	Examples
Edge detection	Differentiation based	Sobel, Laplacian of Gaussian, Canny edge detector
Corner detection	Differentiation/Template based	Harris, Shi-Tomasi, SUSAN, FAST
Blob detection of interest point	Differentiation based	LoG, DoG, Hessian-Laplacian, SIFT, SURF, Cer-SURT
Blob detection of interest region	Template/Segmentation based	ORB, BRISK, FREAK, MSER

Table 6.1: Some representative feature detection methods and their categories..

keypoint in these two images, a descriptor is computed to represent the intensity gradient information in the region around the keypoint.

There are three major steps in the detection process involving scale space scanning, keypoint localization and orientation assignment. The main idea behind these steps is to identify the intensity peak changes in the images at different scales. Most classic detectors follow this principle, including Harris corner [73], Difference of Gaussians (DoG) [130], Harris Laplacian [137], Laplacian of Gaussians (LoG) [130], SUSAN (Smallest Univalue Segment Assimilating Nucleus) [184] corner detection, FAST [162], etc. Table 6.1 covers some classical feature detectors in the field of visual tracking and localization.

The local patches can be edges, corners, points or regions. The intensity changes in the local patches can be calculated by *differentiation based methods*: namely the gradient operators of intensity (first-order differentiation) or the maximal magnitude operator (the second-order differentiation). The differentiation based methods are one of the fundamental building blocks in handcrafted feature extraction. The most common example is used in edge detection: once a gradient image is calculated, pixels with large gradient values become potential edge pixels. Moreover, in order to make a keypoint invariant to rotational changes, an orientation based on the local image gradient direction is often assigned to a keypoint. For instance, SIFT and SURF detectors have proven their good invariant performance across scale and viewpoint changes.

We can also create a size-fixed template and compare its intensity with the local patches to find keypoints, known as *template based technique*. This technique is often applied to detect blobs and corners. Blobs can be regarded as compact objects of approximately the same intensity and the corners are sharp turn of the contour. Their shapes thereby can be modeled as templates.

Segmentation based methods generate reliable segmented regions in images based on not only pixel intensity but also color and structure information in the image. For example, MSER [51] can detect connected regions characterized by almost uniform intensity, surrounded by contrast background. This kind of methods is very useful in the blob detection.

After the keypoint detection, descriptors are used to identify a keypoint, like a fingerprint. For example, in the implementation of SIFT, a 128-dimensional feature vector is constructed by normalizing a 16×16 region around the keypoint to a 4×4 histograms with 8 bins of gradient orientations.This vector is served as a description to identify this SIFT feature. While in SURF, the description vector has be reduced to only 64 dimensions which are represented by 4×4 neighborhood regions with 4 bins of orientations. Different representations will lead to different performances. Intuitively, the time consumption to compute a SIFT descriptor is significantly higher than the one of SURF. Since binary descriptors are faster to compute and require less memory storage, they are widely used in real time applications, such as BRIEF [27], ORB [163], BRISK [116] and FREAK [4]. In general, binary descriptors are generated by comparing the detected local patches with a fixed template. The result of comparison can be described by a binary string of the intensity difference. More sophisticated way is to construct many pairs of points around the keypoint and adopt the pairs to do the comparison work. It is natural that the binary computation is more efficient. There are also some approaches in the literature that can tackle big illumination changes like LIOP (Local Intensity Order Pattern) descriptors [210].

A so called handcrafted feature is composed of detector and descriptor. We can have many kinds of combinations in form of "detector+descriptor", such as "SIFT+SIFT" (SIFT feature), "SURF+SIFT", "SURF+SURF" (SURF feature), "MSER+SIFT", "FAST+BRIEF" (ORB feature), etc. The different features are normally evaluated by their invariance to both geometric and photometric transformations, and the execution time. Here, we only reviewed the performance of the recent handcrafted features (Table 6.2) in the context of SLAM, Visual Odometry and Visual Tracking. Since these features are widely adopted in SLAM and visual tracking tasks, we believe that they are capable of dealing with geometric and photometric changes to some degree. Moreover, they can be potential choices in our case as well.

6.4.2 Robust Feature Matching

The matching between two descriptors is usually carried out by a suitable distance metric, such as Euclidean or Hamming distance. In the literature, the brute-force and FLANN algorithms are two most widely used approaches in the feature matching: in the brute-force matching every feature of one image is compared to that of another image, and in the FLANN algorithm the nearest neighbor search is employed to speed up matching by reducing the searching space.

Outliers have to be removed. Several techniques exit but *ratio test* is the simplest way. Ratio test is conducted by comparing the distance to the closest match to that of the second-closest match. This measure works effectively because correct matches need to have the significantly closer match than the closest incorrect match to achieve reliable matching. According to the object recognition implementation of Lowe *et al.* [125], when the ratio between the correct matches and the closest incorrect match is greater than 0.8, 90% outliers are eliminated with less than 5% inliers discarded. If the raw correspondences are sufficient, it is not serious to discard inlier matches. However, since the descriptors of repetitive or similar structures in man-made environment are close, a lot of correct correspondences will be removed

by the ratio test. Secondly, a *symmetric check* (also known as double check or cross check) can be used to reject the outliers. A good matching pair is obtained only if they are best correspondences for both left-to-right and right-to-left matching[3]. This alternative is not useful to cope with the repetitive problem in the urban environment. In addition, we can obtain the reliable matches by checking the *geometric consistency*. The fundamental or a homography matrix[4] can be estimated using a RANSAC to reject outliers. All inlier matches satisfying the epipolar constraint make up a support set for the RANSAC model. This process is repeated for finite times and the biggest support set in the model is considered as the most likely correspondences.

We can leverage all previous schemes to ensure more robust matchings when the number of raw correspondences is high enough. Since the number of candidate matchings would be reduced a lot after every checking scheme, there is a risk that we cannot obtain enough good matchings to solve the PnP problem if too many schemes are employed.

According to the main feature matching pipeline exposed here, we use the reviewed features (*cf.* Table 6.2) to test our challenging scenarios (see Figure 6.5). We evaluate the features by matching two scenes with normal point views (Scenario #1), big different point views (Scenario #2), similar urban appearance (Scenario #3) and big illumination changes (Scenario #4).

We do not put the emphasis on the basic criteria rotation, scale and affine invariance. Normally such metrics should been evaluated since the birth of a handcrafted feature[5]. Moreover big transformations appears very rare in the urban driving and the reviewed features have already been proved to be capable of solving their respective urban localization issues. Therefore, we put more attention on the inlier matching performance, namely average number of detected features in two images, remained inliers after the inlier selection and the efficiency.

In the evaluation, we calculated the number of detected features in two images of different scenarios, the matched value after the ratio test and symmetric check, the final matching inliers after a following RANSAC check and the time cost for each scenario. The obtained values are registered in vectors, *e.g.* $[7386, 6292, 2, 0, 2.16]$. The comparative result is illustrated in Table 6.2. In Scenario #1, features with SIFT, SURF and Harris detectors obtain stable final matching inliers after a RANSAC check, while most binary based features (ORB and FREAK) failed. As we know, binary based features are often implemented in real time and they are not effective to deal with big intensity changes caused by time or weather. In addition, when FAST detector is combined with binary or template based descriptors, a lot of false matchings are computed and FAST fails with the SIFT descriptor. For the test of Scenario #1, it is observed that even if the FAST and binary based features are able to detect a lot of keypoints but they have a poor performance to obtain inlier matchings. In Scenario #2, features with SIFT, SURF and Harris detectors are still invariant to the big point view changes. There should be no matchings obtained in Scenario #3 and only "SIFT+SURF" feature rejected all false matching for this

[3]For example, we can swap the training and query descriptors to realize the cross check in OpenCV.

[4]For example, we can use 8-point *findFundamentalMat* or 5-point $findHomography$ algorithms to apply this check.

[5]For example, the widely-adopted Oxford matching dataset [139] is regarded as a basic benchmark to evaluate the transformation invariance of a proposed feature.

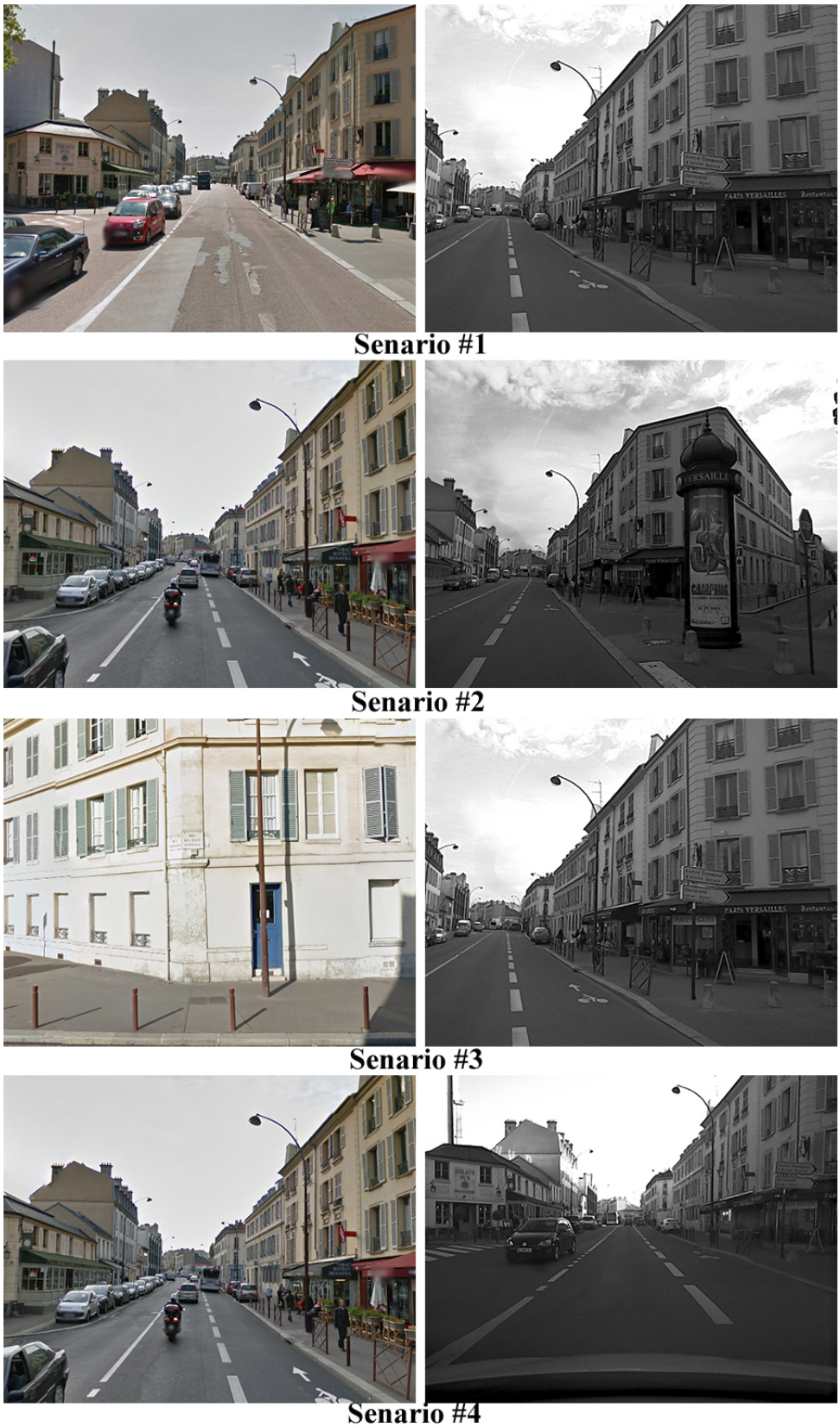


Figure 6.5: Google Street View on the left and online image from Zoé / Scénic on the right. Note that: the two images describe the same scene existing lots of repetitive facade windows under normal point views in Scenario #1; the two images describe the same scene with big different viewpoints in Scenario #2; the two images describe two completely different scenes in Scenario #3, where no correspondences should be found normally; the two images describe the same scene with big illumination changes in Scenario #4.

test. Concerning Scenario #4, LIOP descriptor does not show evident advantages to deal with the big illumination problem. Results from the MSER feature appear terrible since it does not work for all challenging scenarios. In summary, Table 6.2 demonstrates that differentiation based features, like SIFT and SURF, outperform both the template and segmentation based features. However, when we illustrated a classic feature matching pipeline with SIFT-SIFT feature on the first scenario, we found that the ratio test, symmetric check and the 8-point RANSAC rejection are not able to eliminate all outliers: after the ratio and symmetric check, we obtain 40 matchings and then a RANSAC finds a support set with 10 correspondences including outliers (see Figure 6.8). It is probably caused by repetitive landmarks in the scene and the different brightness makes the matching more difficult. In next section, we visualize the vectors to have a better understanding of the raw results.

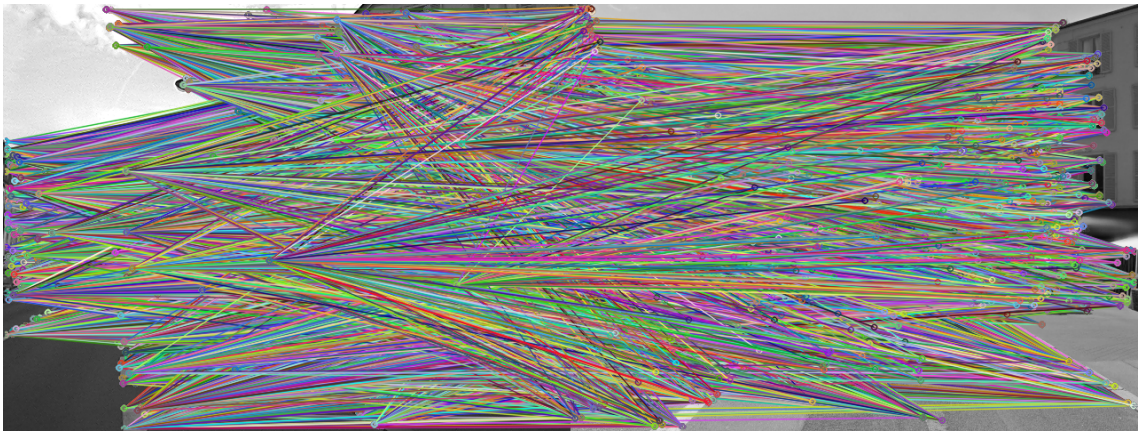


Figure 6.6: FLANN matching with SIFT-SIFT features before the ratio test and symmetric check.

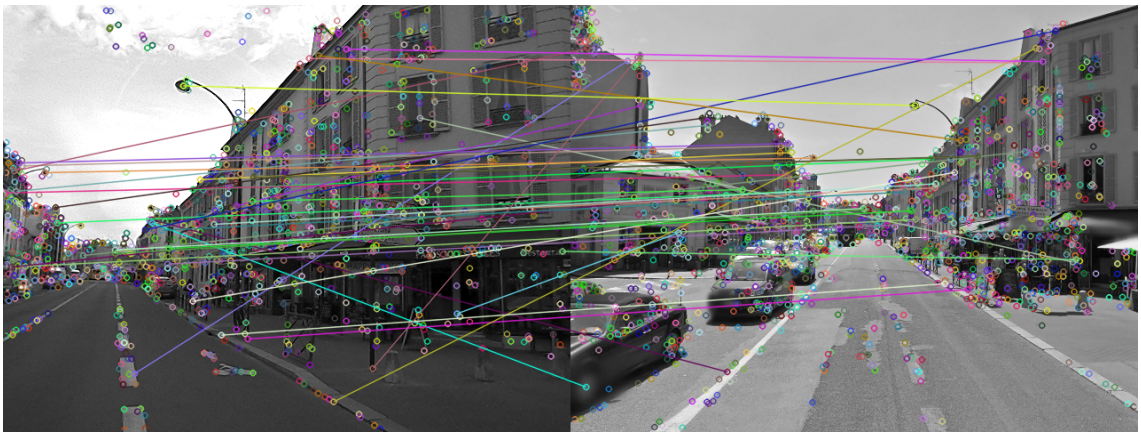


Figure 6.7: FLANN matching with SIFT-SIFT features after the ratio test and symmetric check.

Detector	Descriptor	Reference	Implementation	Scenario 1	Scenario 2	Scenario 3	Scenario 4
FAST	4×4 patch	Forster et al. [59]	Tracking, VO	[7386, 6292, 35, 21, 2.81]	[7285, 6186, 93, 6, 3.24]	[7386, 5507, 42, 18, 2.22]	[5430, 6186, 42, 29, 3.36]
FAST	SIFT	Konolige et al. [104]	VO, SLAM	[7386, 6292, 2, 0, 2.16]	[7285, 6186, 19, 13, 2.12]	[7386, 5507, 2, 0, 2.00]	[5430, 6186, 23, 16, 1.87]
Harris	SURF	Carrera et al. [28]	Tracking	[289, 833, 21, 9, 0.21]	[209, 863, 25, 10, 0.22]	[289, 923, 13, 7, 0.21]	[260, 863, 20, 9, 0.20]
FAST	ORB	Skrypnyk et al. [180]	VO	[7386, 6292, 37, 16, 3.83]	[7285, 6186, 75, 30, 3.83]	[7386, 5507, 21, 8, 0, 3.10]	[3865, 6186, 40, 11, 2.20]
ORB	ORB	Mur-Artal et al. [146]	ORB-SLAM	[500, 500, 8, 0, 0.16]	[500, 500, 20, 5, 0.15]	[500, 500, 4, 1, 0.16]	[500, 500, 48, 10, 0.15]
SURF	LIOP	Minshkin et al. [141]	Scene Matching	[2514, 2298, 13, 8, 1.62]	[2240, 2451, 39, 9, 1.93]	[1280, 1128, 9, 7, 2.37]	[994, 1732, 33, 12, 1.85]
SIFT	SIFT	Karlsson et al. [97]	Visual SLAM	[1280, 1518, 40, 10, 1.07]	[1250, 1732, 40, 9, 1.14]	[1280, 1128, 11, 7, 0.98]	[994, 1732, 57, 12, 1.06]
SURF	BRISK	Jiang et al. [94]	VO	[2514, 2298, 13, 7, 1.51]	[1250, 1732, 15, 9, 1.44]	[1280, 1128, 7, 7, 2.0]	[994, 1732, 26, 8, 1.09]
SURF	BRIEF	Heinly et al. [80]	Scene Matching	[2514, 2298, 19, 9, 1.20]	[2240, 2451, 25, 7, 1.15]	[2514, 2370, 8, 7, 1.59]	[1520, 2451, 21, 9, 0.93]
SURF	SURF	Engelhard et al. [56]	SLAM	[2514, 2298, 30, 9, 2.04]	[2240, 2451, 42, 10, 2.00]	[2514, 3677, 14, 8, 2.59]	[1502, 2451, 50, 9, 1.72]
SURF	SIFT	Gil et al. [64]	SLAM	[2514, 2298, 11, 7, 4.54]	[2240, 2451, 13, 7, 4.40]	[2514, 3677, 0, 0, ∞]	[1502, 2451, 19, 9, 3.75]
MSER	SIFT	Leong et al. [81]	Loop closure	[385, 313, 1, 0, 1.17]	[314, 425, 1, 0, 1.61]	[385, 414, 0, 0, 1.52]	[235, 425, 3, 0, 1.40]
BRISK	BRISK	Leutenegger et al. [116]	SLAM	[443, 603, 2, 0, 0.16]	[431, 690, 6, 0, 0.17]	[443, 576, 4, 0, 0.16]	[390, 690, 5, 0, 0.17]
BRISK	FREAK	Hartmann et al. [76]	SLAM, VO	[443, 603, 0, 0, ∞]	[431, 690, 1, 0, 0.20]	[443, 576, 0, 0, ∞]	[390, 690, 2, 0, 0.20]
FAST	BRIEF	Gálvez-López et al. [61]	Place recognition, VO, SLAM	[7386, 6292, 29, 13, 3.93]	[7285, 6186, 73, 28, 3.91]	[7386, 5507, 10, 7, 3.24]	[5430, 6186, 28, 15, 2.24]

Table 6.2: A comparison of some classic handcrafted features in the literature. We calculate the number of detected features in two images of different scenarios, the matched value after the ratio test and symmetric check, the final matching inliers after a following RANSAC check and the time cost for each scenario. These values are registered in a vector, *e.g.* $[7386, 6292, 2, 0, 2.16]$. For the work of Forster et al. and Minshkin *et al.*, we use their codes on Github directly. Note that if the RANSAC check cannot be conducted, the time cost is noted as ∞. Some bad and good results are highlighted in red and blue respectively.

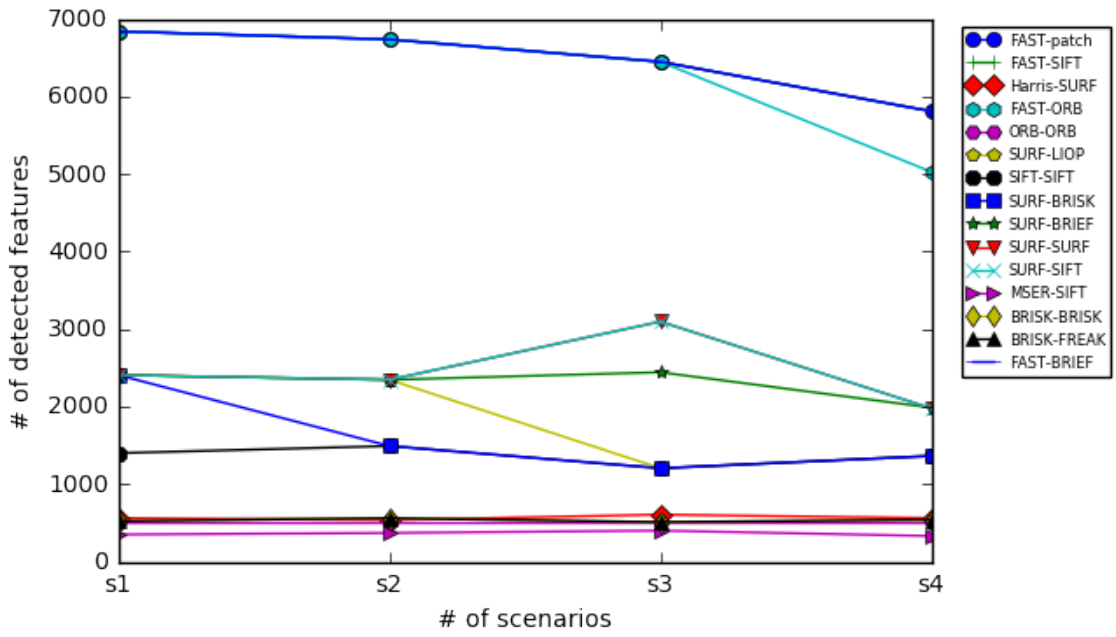


Figure 6.9: The average detected feature numbers of both images in 4 scenarios by the methods of Table 6.2. All the features are able to extract enough keypoints for all scenarios. FAST based features can detect more local keypoints than others for its simple template.

Figure 6.8: FLANN matching with SIFT features after ratio test, symmetric check and 8-point RANSAC selection.

6.4.3 Discussion

We plot the results of reviewed features in the Figure 6.9-6.11. Our aim is to find a robust handcrafted feature which is robust to big viewpoint[6] and brightness changes, able to distinguish repetitive structures at a relatively high speed. Therefore, this potential feature should compute as many matching inliers as possible in the Scenario #1, #2 and #4, but less and even no matchings in the Scenario #3. After the comparison, we found that:

[6]Note that: the offline imagery is back-projected from panoramas which lead to viewpoint changes compared to the online dataset.

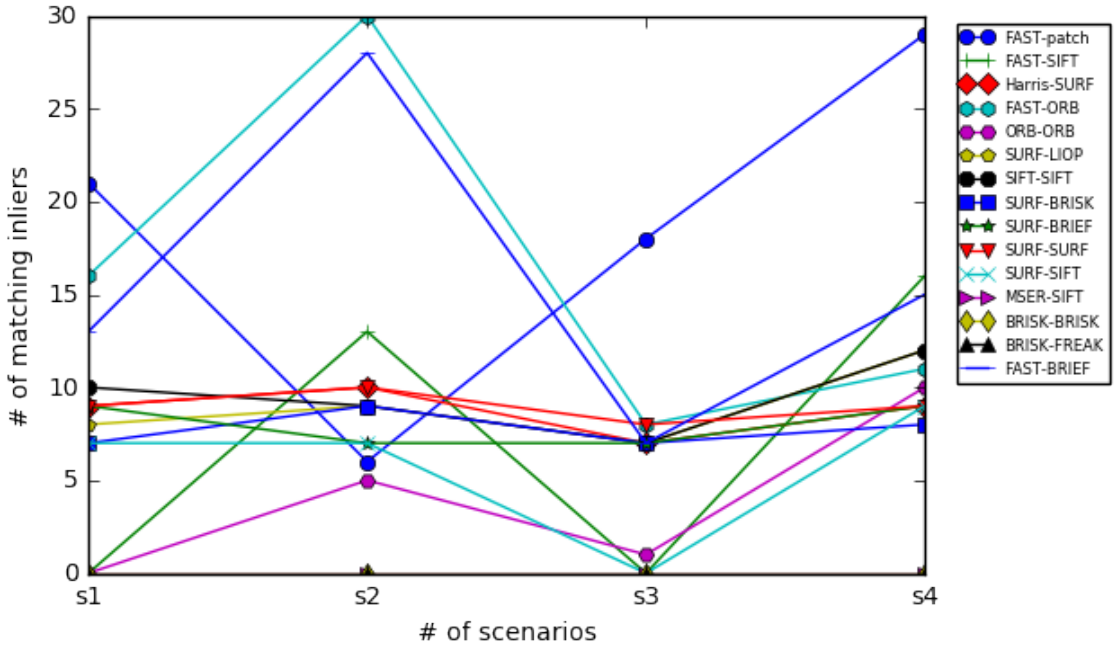


Figure 6.10: The detected inliers after the ratio test, symmetric check and RANSAC rejection using the features listed in Table 6.2. Please note the "inliers" mentioned here are just the matches after the conventional checks, not the real good matches.

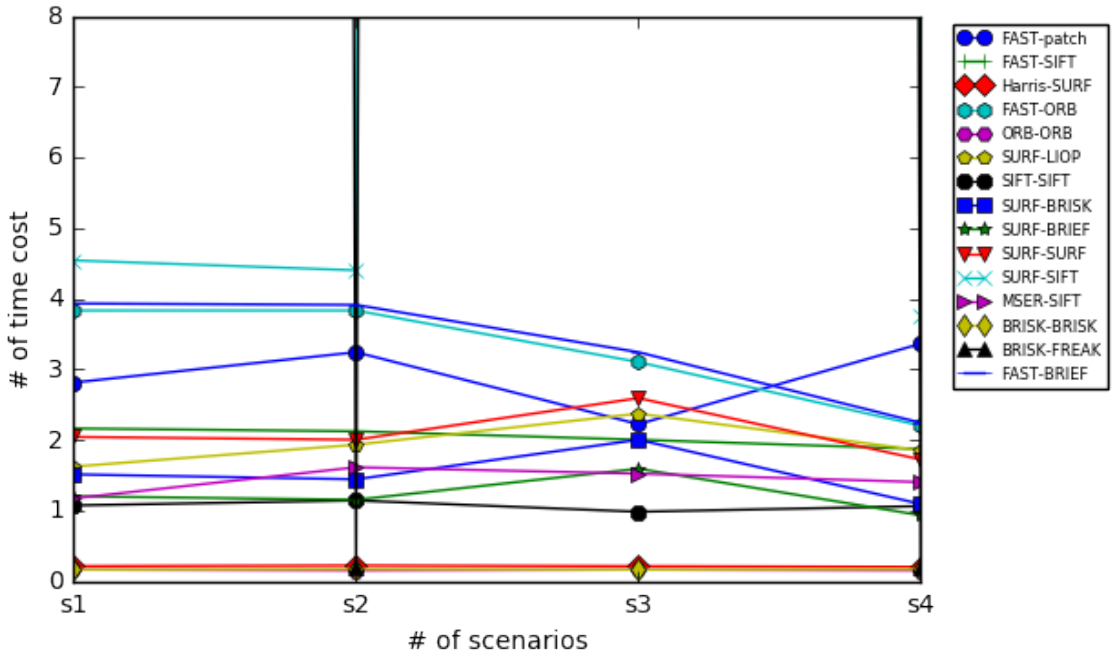


Figure 6.11: Time cost in second for detection, description, matching and inlier selection for different features in Table 6.2. Please notice when the inlier selection does not work, the time is noted as infinity.

- FAST and Harris corner detector can extract many keypoints in a short time but the obtained inliers are not sufficient for a pose estimation (*cf.* Figure 6.9 and Figure 6.10). These detectors are not invariant to the urban repetitive structures like the Scenario #3. It would be better to adopt differentiation based descriptors rather than binary ones to represent these detectors. Because of the large number of detected keypoints, the time cost for the matching is no longer an advantage for these simple detectors, see Figure 6.11.

- Template based features (binary features) are not suitable for our case due to the substantial illumination and over-season changes, they failed to extract good enough matchings in all four scenarios, see Figure 6.10. These features are mainly implemented on the real time tracking and VO tasks where the intensity can be regarded constant between consecutive camera frames. They are not effective for the matching between the cross-date images.

- Considering the illumination problem of Scenario #4, the LIOP descriptor and the MSER detector do not show significant superiority to deal with the severe intensity changes. As depicted in Figure 6.12, there are still wrong matchings with the SURF-LIOP feature after the outlier rejection process. In the urban area, illumination and brightness changes are very common. In literature, the histogram equalization [32, 86] is often used to enhance the contrast of the matched images prior to feature extraction. As shown in Figure 6.13, this technique spreads out the most frequent intensity values in the referenced image to equalize the global contrast in the processed image.

- After the analysis, we observed that SURF-SIFT and SURF-SURF features outperform other features, especially regarding the final correct inlier numbers. SIFT descriptors take a longer time to compute but more robust inliers compared to the SURF descriptor. Moreover, SURF-SURF cannot cope with the repetitive urban appearance (see Figure 6.19). Figure 6.14-6.20 display the performance of the SURF-SIFT and SURF-SURF features in four scenarios[7].

- SURF-SIFT or SURF-SURF features are potential choices for the metric localization algorithm since they ensure the invariance to rotation, scale and brightness. Yet, the matching inliers are too few to adopt the *solvePnP* method. It is likely that some of these matching inliers might lack the depth information from the Street View database, *e.g.* traffic lights and lanes. As such, we can not estimate pose from these inliers.

[7]Note that there is no final inlier matchings are obtained by the SURF-SIFT feature in Scenario #3 and we thereby do not display this process.

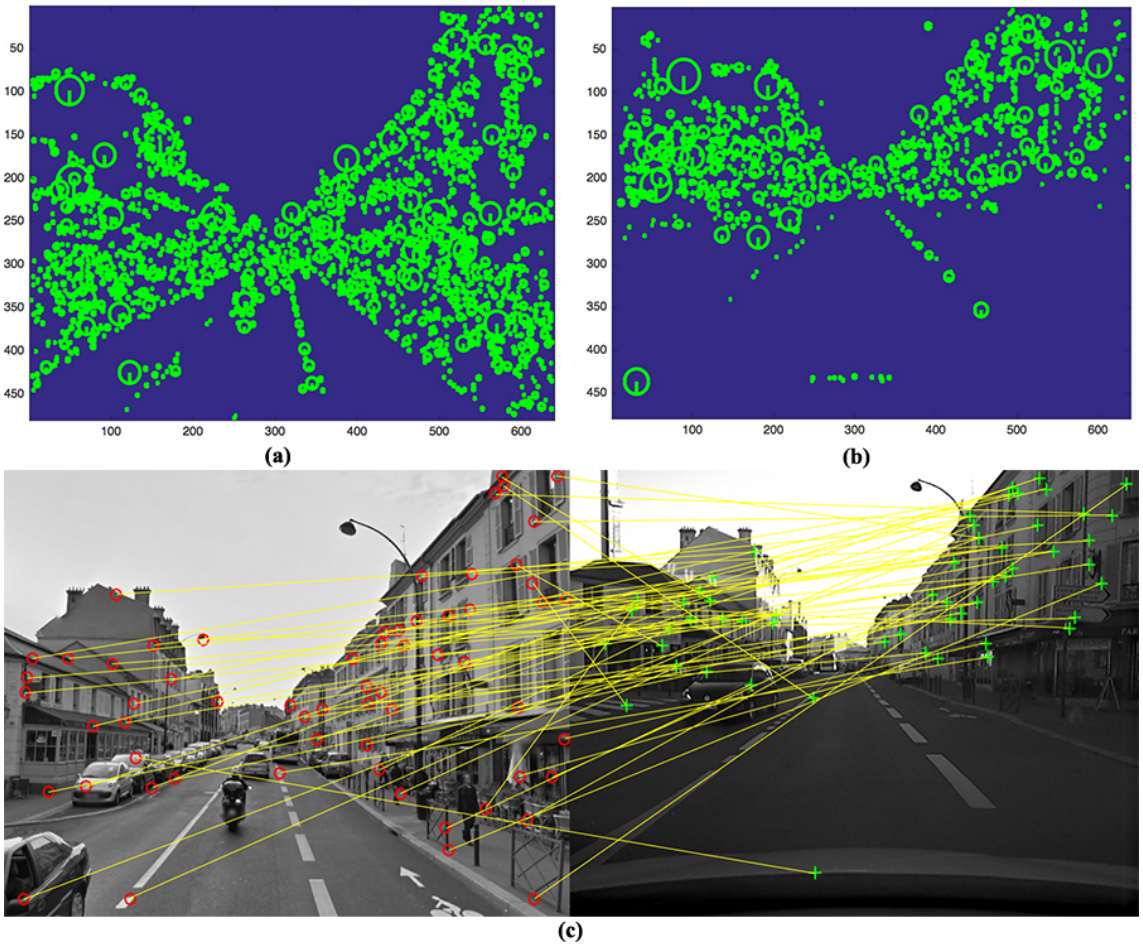


Figure 6.12: (a) and (b) illustrate the local and overall intensity ordinal information of the local patch which are captured by the LIOP descriptor with the VLFeat library [207]. (c) displays the matched inliers after the ratio test, symmetric and RANSAC check. We can see there are still lots of false correspondences.

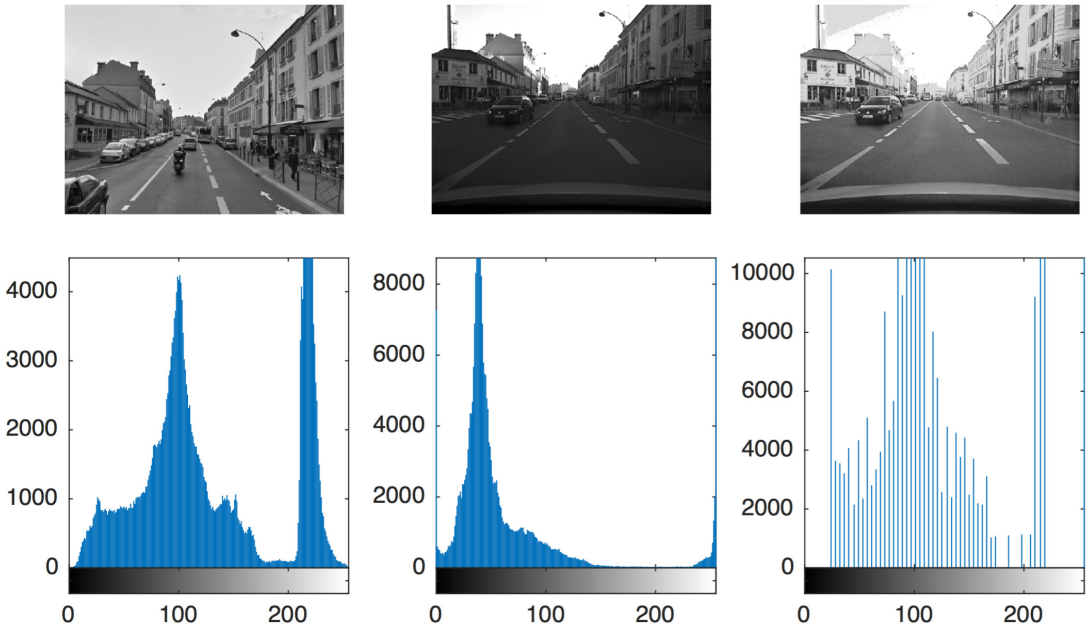


Figure 6.13: The concepts of histogram equalization to improve the contrast of matched images. The right image is equalized from the middle original image according to the intensity distribution of the left Street View image.

Figure 6.14: Final inliers detected by the SURF-SIFT feature in Scenario #1. As the matches are correct and enough, SURF-SIFT works well in normal point view.

Figure 6.15: Final inliers detected by the SURF-SIFT feature in Scenario #2. SURF-SIFT still performs well for big different viewpoint but the detected number is small.

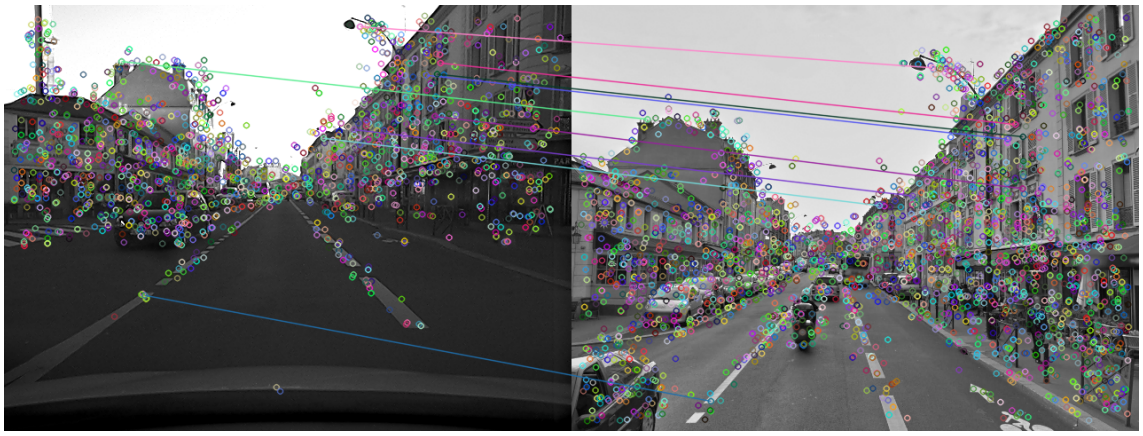


Figure 6.16: Final inliers detected by the SURF-SIFT feature in Scenario #4. The feature can still extract good matches under the big illumination change.

Figure 6.17: Final inliers detected by the SURF-SURF feature in Scenario #1. SURF feature works in normal point view but some wrong matches exist due to the repetitive elements in the urban environment.

Figure 6.18: Final inliers detected by the SURF-SURF feature in Scenario #2. A lot of matches are detected but there are still false matches.

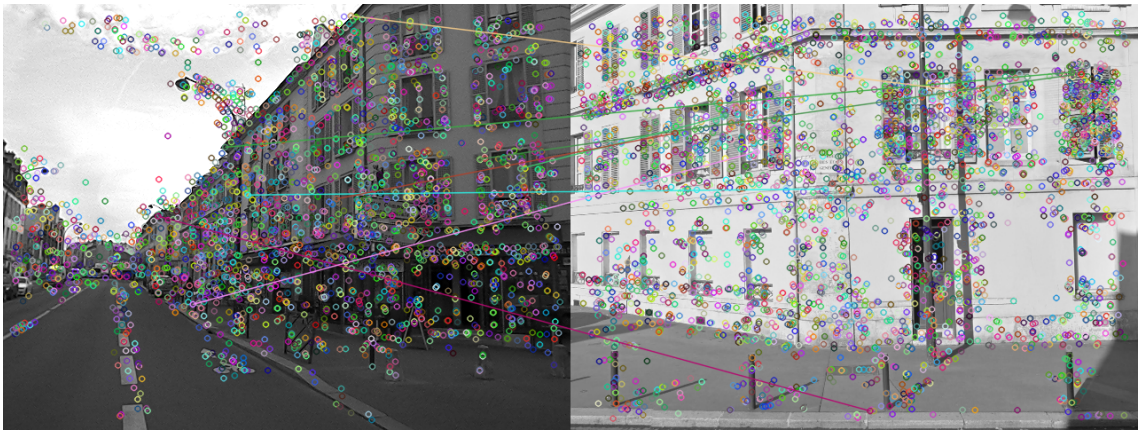


Figure 6.19: Final inliers detected by the SURF-SURF feature in Scenario #3. In fact, no correspondences should be calculated in this case. These false matches are probably caused by the similar urban appearances.

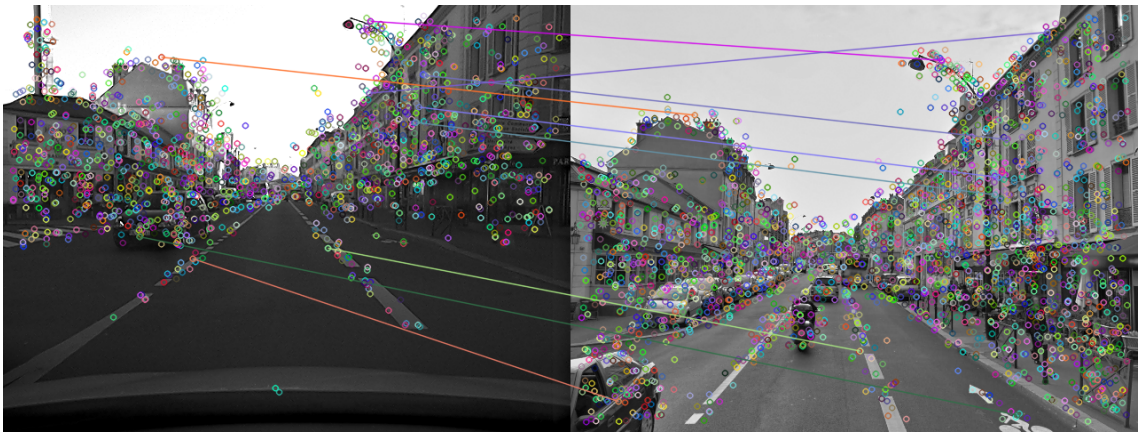


Figure 6.20: Final inliers detected by the SURF-SURF feature in Scenario #4. Here SURF feature does not work for big illumination changes.

6.4.4 Virtual Line Descriptor kVLD

As we stated, matching detected features based on the similarity of their descriptors provides both inliers and outliers. Eliminating those outliers while preserving inliers remains challenging for images captured in urban area, which have ambiguities and strong transformation. Ambiguities are caused by repetitive patterns or lake of texture and transformation arises by some occlusions. A RANSAC based feature matching is usually capable enough to deal with these problems in the context of SLAM and VO (see in Table 6.2). Yet it works well only when the inlier rate is high. In our challenging scenarios, the inlier rate is often low which may degrade the RANSAC based check. Even some better alternatives like PROSAC [36] or USAC [159] can not well suited for inlier rates lower than 50%. Only a few methods like ORSA (Optimal Random Sampling Algorithm) [142] can treat an inlier rate of 10%. Moreover, the geometric check with RANSAC often suffers from a limitation to estimate the fundamental matrix: it cannot remove outliers corresponding to points that have matches near their epipolar line but far from the correct location.

Therefore, we get out of the context of SLAM or VO, we choose a more well-performing handcrafted feature which is a graph based feature matcher, which is called k-Virtual Line Descriptor (kVLD) [123]. With this feature, a graph matching method is conducted rather than the standard pipeline. In a graph matching, several graphs are constructed where their vertices are feature points and the edges are the pairwise relations. Then a vertex correspondences between the two graphs are sought according to certain matching criteria. Usually, a second order photometric graph matching is applied where the matching constraints are the relative distance between two points combined with the relative orientation.

kVLD is a geometric and photometric consistent descriptor. Several salient keypoints are detected by SIFT detector at first. The idea of kVLD relies on the fact that, given a potential match $(P_i, P_i^{'})$, see in Figure 6.21, if there are at least K paires $(P_{jk}, P_{jk}^{'}), k \in \{1, \cdot, K\}$ are geometric virtual-line consistent with the $(P_i, P_i^{'})$, $(P_i, P_i^{'})$ is considered as a correct match. In the experiment, authors proved that $K = 3$ is able to guaratee a good performance. To be more robust, the kVLD algorithm use a RANSAC invariant ORSA to reject iteratively matches which have less than K geometric consistences in all potential virtual line matches. One important thing in kVLD is to represent every virtual line, such as (P_i, P_j). We can simply regard the virtual line as a concatenation of all SIFT descriptors. The line is described by the covering strip between two points, where the strip is represented by connected SIFT patches. The final strip descriptor includes a small-size histogram of gradients and an orientation. Virtual line descriptors are then matched by a second order graph matching technique. In the end, an ORSA is added to select more robust virtual line matches.

Figure 6.22-6.25 display the initial SIFT-like features detected by the kVLD and Figure 6.26-6.29 illustrate some robust matching examples using the kVLD graph matching and a ORSA in the four scenarios. We found that kVLD descriptor outperform all the previously reviewed features. It is not necessary to use the histogram equalization to counter the illumination effect in the Zoé dataset with kVLD descriptor. kVLD seems to be not perfect as illustrated in Figure **??**. However, as we stated in the problem description, we can render several Street View candidates for an online VEDECOM image. In this way, we believe kVLD can work robustly to

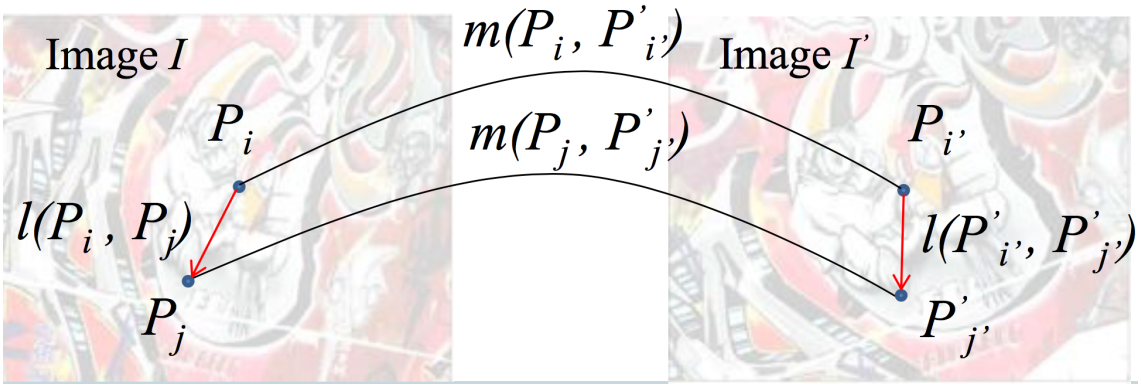


Figure 6.21: Intuition of the kVLD descriptor: for any two detected points P_i, P_j in the image I, and P_i', P_j' in the image I', the lines $l(P_i, P_j)$ and $l(P_i', P_j')$ are unlikely to be similar unless both matches (P_i, P_i') and (P_j, P_j') are correct. Image credit of [123]

.

detect enough 2D-3D correspondences to solve the PnP problem in the next stage.

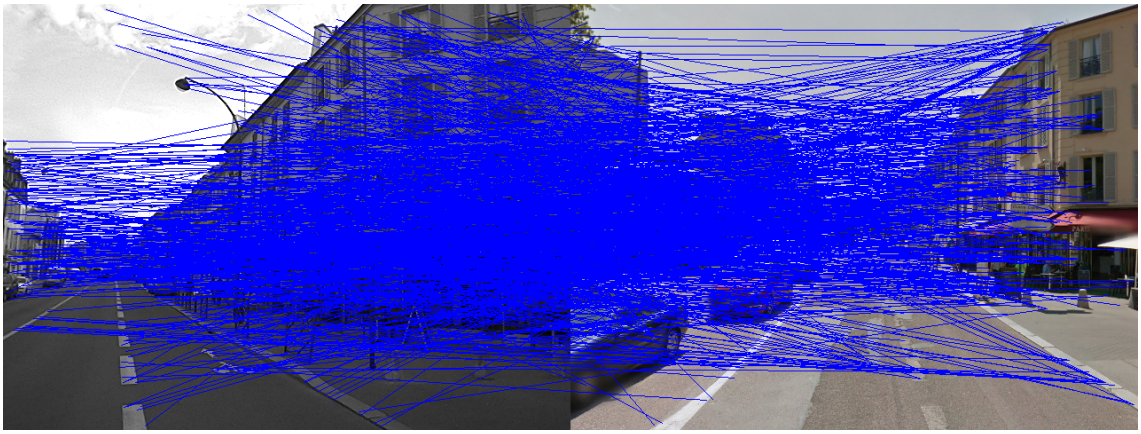


Figure 6.22: Initial features detected by the kVLD in Scenario #1.

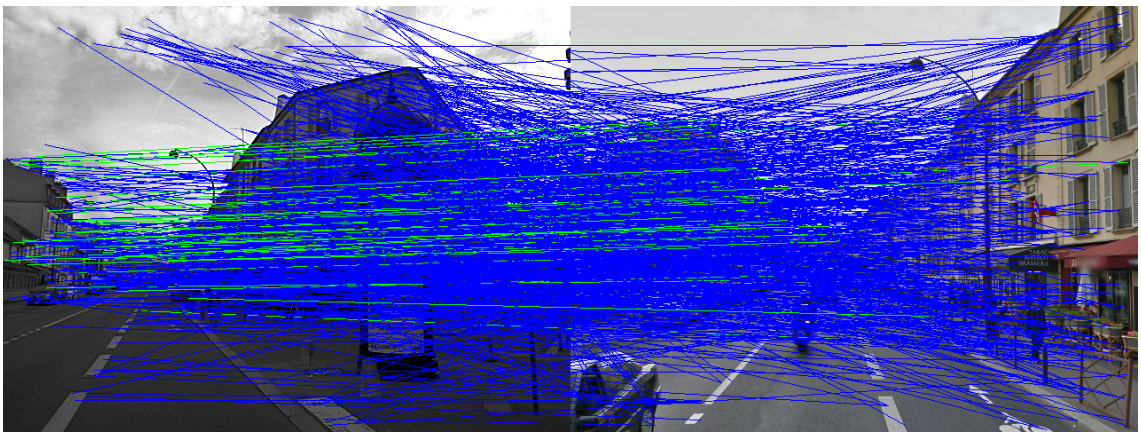


Figure 6.23: Initial features detected by the kVLD in Scenario #2.

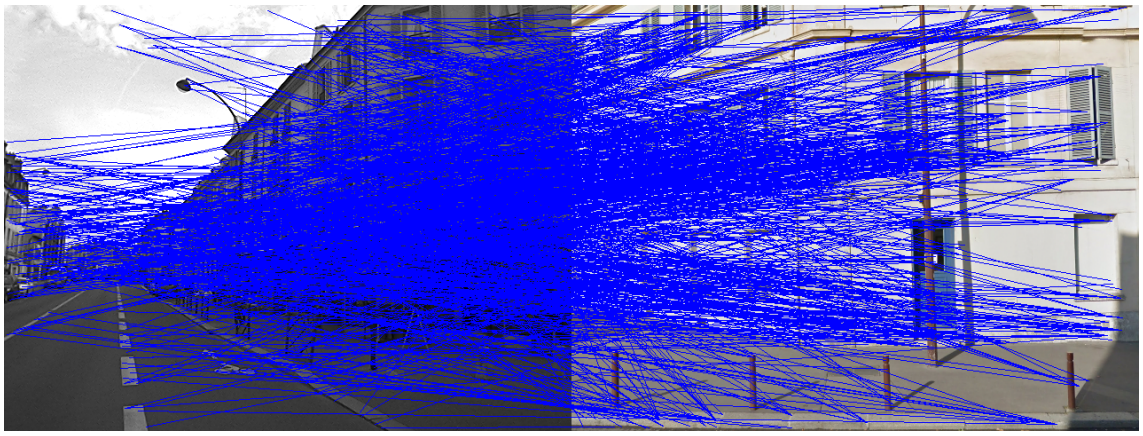


Figure 6.24: Initial features detected by the kVLD in Scenario #3.

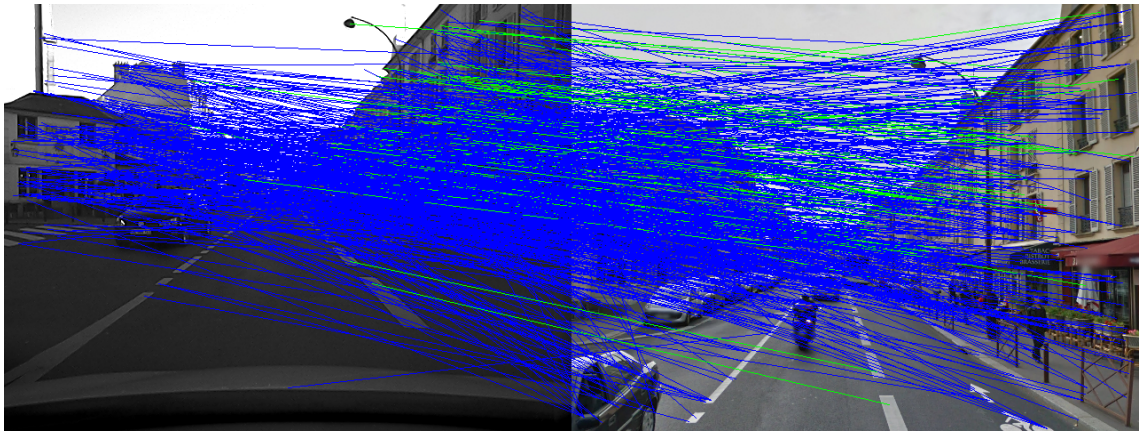


Figure 6.25: Initial features detected by the kVLD in Scenario #4.

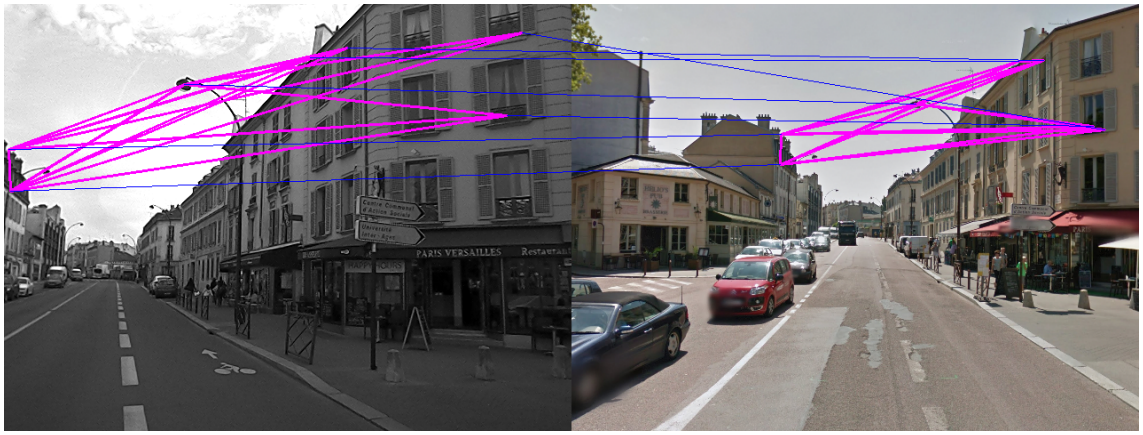


Figure 6.26: Final inliers detected by the kVLD feature in Scenario #1. Please note that the matches obtained by graph matching are displayed in blue lines and the final matches verified by ORSA in green. Final valid VLDs are in magenta. In this scenario, kVLD doest not work well due to too many repetitive facade windows.

Figure 6.27: Final inliers detected by the kVLD feature in Scenario #2. Many valid kVLDs are kept by ORSA.

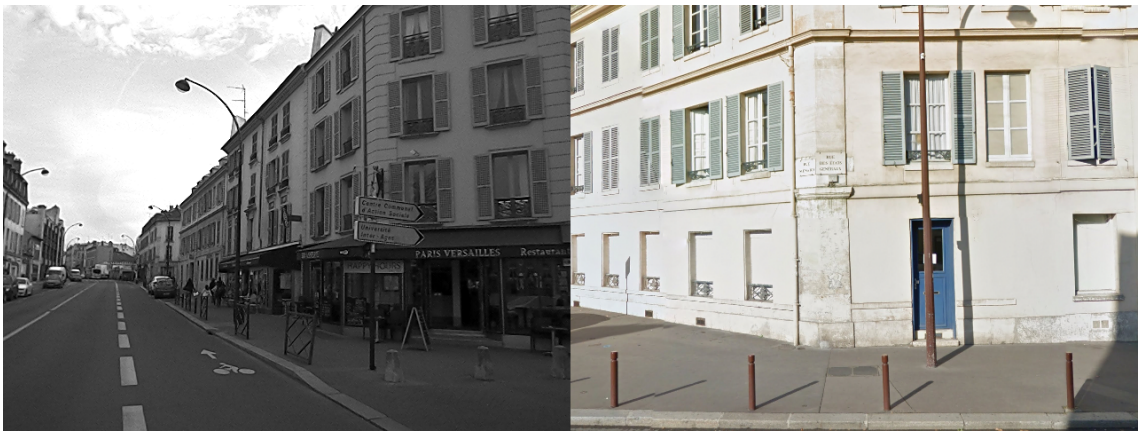


Figure 6.28: Final inliers detected by the kVLD feature in Scenario #3. In fact, no correspondences should be calculated in this case.

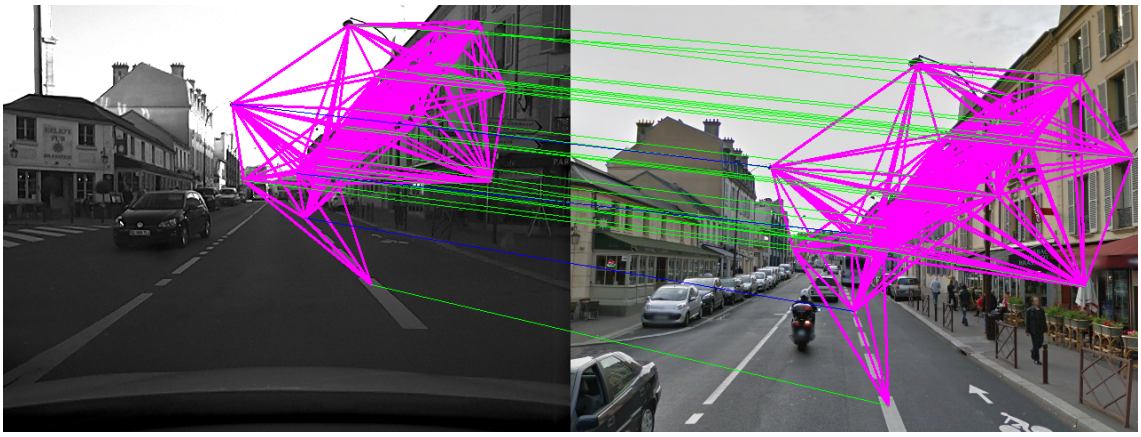


Figure 6.29: Final inliers detected by the kVLD feature in Scenario #4. Many valid kVLDs are kept by ORSA even with a big illumination change.

6.5 Conclusion

In this chapter, we described the formal problem and proposed to solve this problem by the handcrafted-feature-based methods. We briefly described that a coarse to fine localization would be a potential localization algorithm. We outlined the challenges to realize image search and image matching by going through our captured online and offline datasets. Since feature matching is a crucial part for this algorithm, it is necessary to adopt well-performed handcrafted features and robust matching algorithms. Therefore, we focused on studying the current used handcrafted features in the state-of-the-art of SLAM, VO and tracking. We manually selected some difficult matching scenarios to evaluate the performance of these features. We put the emphasis on the inlier correspondences and the robustness to the image noise, illumination and baseline. Only some differentiation based features, SURF-SURF and SURF-SIFT, outperformed in these scenarios but they seem to lack an adequate number of matchings for the $solvePnP$ techniques. Finally, we tested a combined feature (SIFT detector with Virtual Line descriptor kVLD) and it demonstrated good performances to obtain robust matchings for the pose estimation.

Chapter 7

Topological and Metric Localization

7.1 Introduction

In this chapter, we present a handcrafted feature based metric global localization in the urban environment only with a monocular camera and the Google Street View database. As we stated in the Chapter 6, our algorithm is able to leverage the abundant sources from the Street View and benefit from its topometric structure to build a coarse-to-fine positioning, namely a topological place recognition process and then a metric pose estimation by local bundle adjustment (LBA).

In the place recognition process, we first evaluate the popular image retrieval methods with our Street View datasets, namely the simple Bag of Words (BoW) and FABMAP. It demonstrates that the current state-of-the-art approaches fail on the sparsely distributed Street Views. Hence we introduce a SIFT-MESR combined BoW method which outperforms the previously described algorithms. In this phase, all offline imagery is trained as a combined BoW database. On the online stage, we extract top similar Street View images from the BoW by comparing with the query image[1] and then estimate relative poses between each other. Moreover, a LBA is employed to obtain the global metric localization from all estimated poses and corresponding 2D-to-3D matching constraints. The feature extraction and correspondences are realized by the kVLD descriptor.

The method is implemented on the urban test area and demonstrates both submeter accuracy and robustness to viewpoint changes, illumination and occlusion. Although a considerable proportion of the localization achieves a 2m accuracy, the discontinuity of Street Views still affects the robustness of the system. We observe that a Street View located far away from the query image ($>$ 8m in the test) generates a significant error in the metric pose estimation phase. Thus, we extend this framework with the construction of an augmented Street Views database in order to compensate the sparsity of Street Views and improve the localization precision. Due to this augmentation, easy handcrafted features, such as SIFT or SURF, are able to obtain robust matchings. We also demonstrate that this approach significantly improves the submeter accuracy and the robustness to important viewpoint changes, illumination and occlusion.

[1]To clarify, the query image is also known as the input image or online image from the VEDECOM cars. Street View image is known as the database image, offline image or referenced image in this thesis.

7.2 Localization with Street View

7.2.1 Overview

Our localization system is divided into two phases, as described in the flowchart of Figure 7.1. The Street View panoramas are prepared and their rectilinear views are generated and represented offline before an actual online vehicle localization.

As we stated in Chapter 5.1, the panorama should be transformed into a set of overlapping or unrelated cutouts to reduce the large angle distortion. Therefore, we assume 8 virtual pinhole cameras, with an intrinsic camera matrix $\mathbf{K}$, mounted in the centre of a unit sphere $\mathbf{S}$ with a user fixed pitch and heading, local yaw angle changing by $[0°, 45°, ..., 360°]$. The number of virtual cameras, intrinsic matrix and pitch/yaw/roll degree are free to select, yet empirically the more identical they are to the actual on-board camera, the better performance expected. Thus in practice, our virtual camera shares the same intrinsic parameters as the on-board camera. After this artificial generation, the Street View database becomes 8 times larger in quantity. Other extracted GIS data, such as topology, geodetic data, depth maps are organized according to Chapter 5.1. After this offline stage, we can dive into our coarse to fine localization systems, namely a place recognition and then a metric pose estimation.

7.2.2 Place Recognition

In this phase, all generated imagery is trained as a compact database for the image retrieval. As we know, there are many algorithms to realize this task. We thereby evaluate the classic SIFT-based BoW and FABMAP algorithms, which are successfully used in traditional street-level localization cases.

Evaluation of State-of-the-Art Approaches

Since the image retrieval training is a time consuming process, we only select a trajectory segment in the test area for evaluation: 892 online images captured from the Scénic right camera. In this urban segment, there are 29 discrete Street View panoramas covering this area. After the back projection, $29 \times 8 = 232$ rectified images are rendered from panoramas, which are all labeled by the geo-localizations and global yaw angles. All the online images should have a corresponding Google Street View image with a close geo-position and a similar global yaw angle. In our test, we use these parameters to generate our ground truth. We assume that if a query image is located close to a rectified Street View and they share a close global yaw angle, these two images can capture the same scene. This rectified Street View is thereby regarded as the ground-truth matching of the query image[2]. In practice, we set the neighborhood of the position and yaw angle between two matched images to 15m and 10° respectively. Therefore, one query image might have several referenced Street Views in such neighborhood boundary, see Figure 7.2.

[2]To simplify, our ground truth is roughly estimated based on yaw angles and position labels. As we know, even if the yaw angles and positions are close, a query image might not share similar visual scene with the Street View due to dynamic traffic. A proper ground truth should be verified manually.

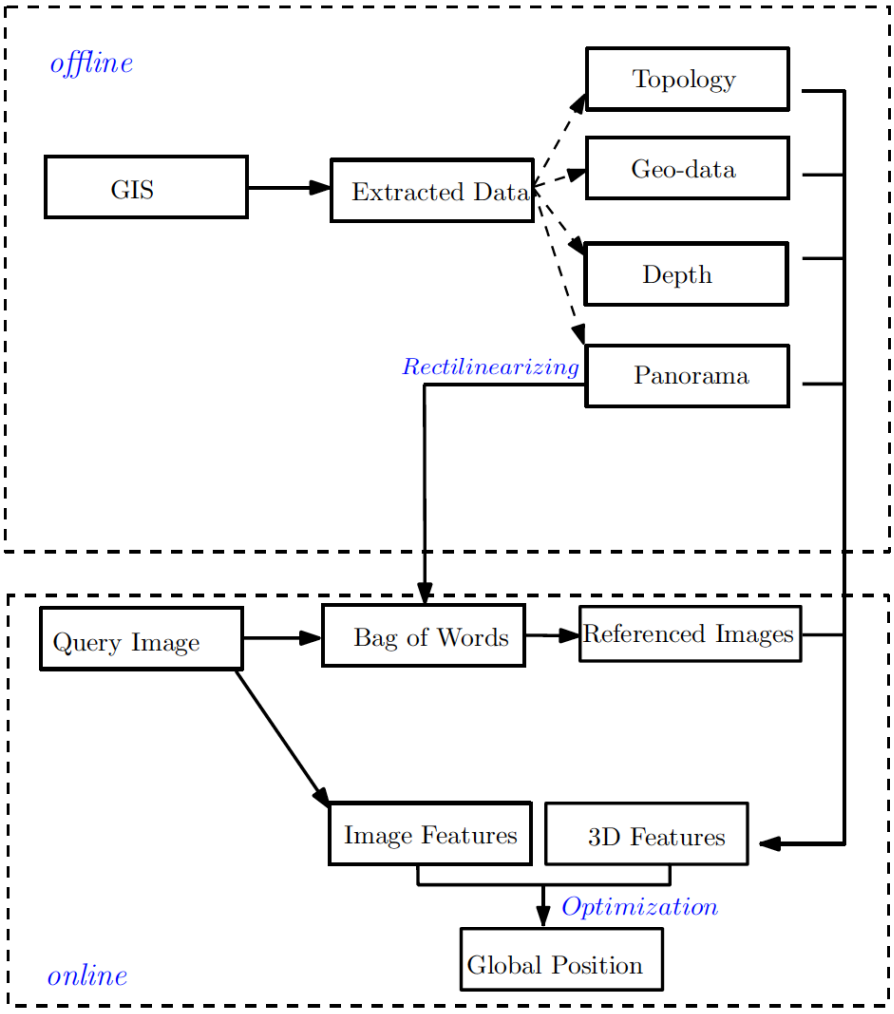


Figure 7.1: Flowchart illustrates workflows between different modules.

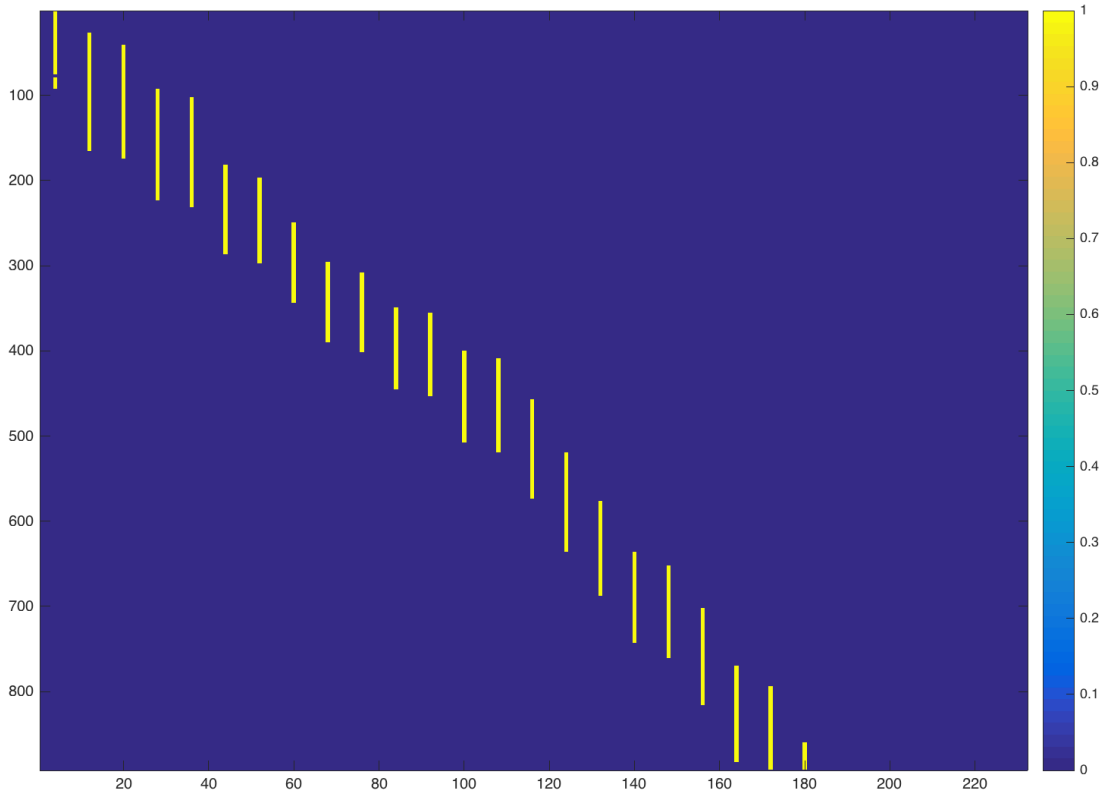


Figure 7.2: The confusion matrix of ground truth for 892 query images and 29×8 Street Views: this data is estimated according to the positions and yaws between Street Views and online images. Without manual labeling, we cannot guarantee it exists scene overlapping between two images.

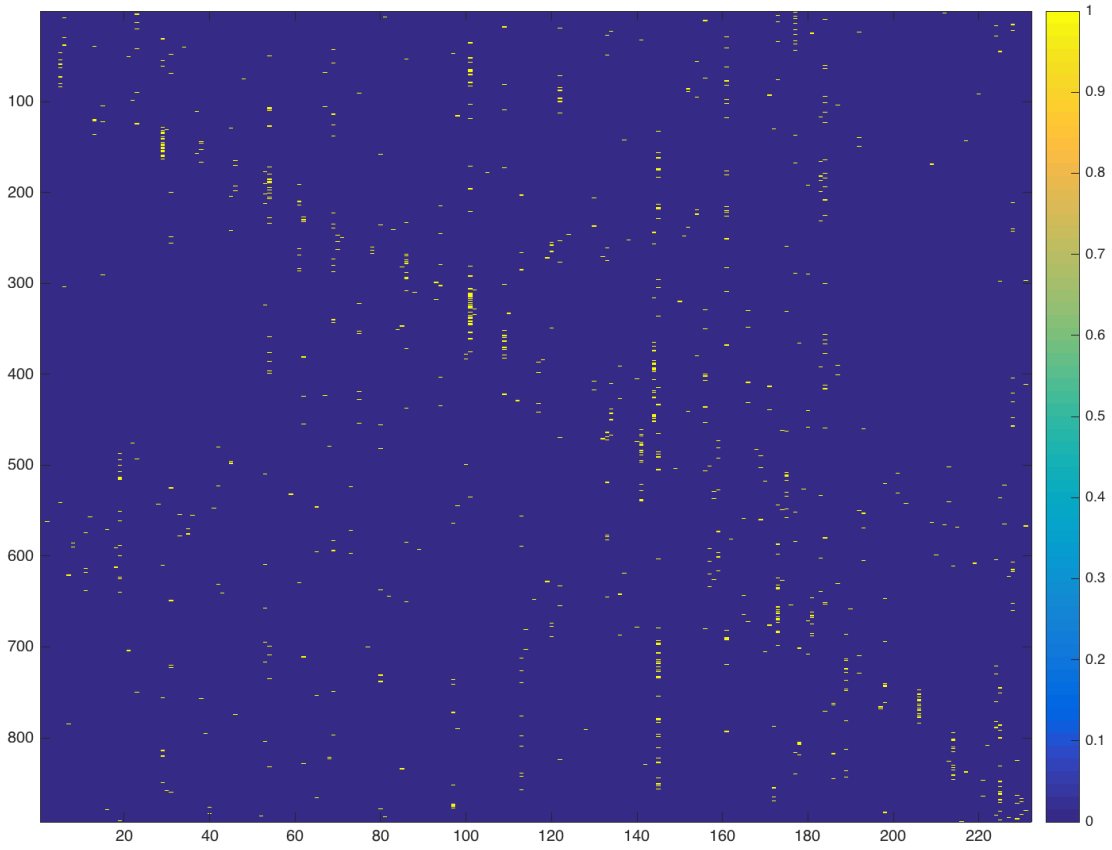


Figure 7.3: The confusion matrix is computed by a classic SIFT BoW for 892 query images and 29×8 Street Views. As observed, this method can detect several correct matchings but there are still many false positives.

The state-of-the-art algorithms have been evaluated in the form of confusion maps, colored as heat maps where a dark blue represents no visual similarity while a dark red is a complete similarity. An ideal place recognition algorithm would have a coincident confusion matrix to the ground truth matrix. In this evaluation, we use the classic SIFT based BoW and openFABMAP [66] algorithms. We also test different configuration parameters to improve the results. In the BoW, we use a hierarchical vocabulary tree by training a total of 10000 visual words in the dictionary. The Term Frequency-Inverse Document Frequency (TF-IDF) (*cf.* Chapter 3) and cosine distance are used for the weighting and the metric evaluation. Cosine distance is denoted by Equation 7.1.

$$cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|} \tag{7.1}$$

where $\mathbf{x}$ and $\mathbf{y}$ are two vectors. This metric is based on the judgment of orientation and not magnitude: two vectors with the same orientation have a cosine similarity of 1, two vectors at 90° have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independently of their magnitude. It is often normalized in positive space bounded between $[0, 1]$. In openFABMAP, the trained vocabulary is also set to 10000 and the other parameters follow the default settings of the authors. In BoW, the cosine metric can compute very small values and it affects a clear visualization of the confusion matrix. We thereby find the maximum of the cosine similarity and only visualize all values above $0.8 * maximum$, *cf.* Figure 7.3. In the openFABMAP, the matching similarity is based on probability theory and it only preserves big likelihoods. Thus we visualize the result directly as shown in Figure 7.4.

As observed, openFABMAP is able to detect more potential candidates compared to the classic BoW method. It is an improved version of BoW where the co-appearance probability of certain visual words is modeled in a probabilistic framework. However, compared to the ground truth, they both have a poor performance in our cases when the training data is limited but the test data is relatively enormous. In many classic applications of FABMAP, their training datasets are usually video sequences with a high frame frequency, which makes it possible to detect enough distinctive visual words. Instead, here only 29 discretely distributed panoramas are used to train the vocabulary. This training dataset is too few to deal with a relative large scale area and construct co-appearance probability for the test procedure. In some panoramas, main scenes would be occluded by mobile vehicles and vegetations and it leads to a loss of important visual words. The next paragraph hence introduces a new approach as displayed in Figure 7.5, outperforming the previously described algorithms.

Combined Bag of Words

As we stated before, we tested different configuration parameters during the evaluation of the BoW, but the result did not improve a lot. We thought that a simple BoW can not learn enough distinctive visual words from a small training dataset. An ideal visual vocabulary must be built up in both BoW and FABMAP. In this building process, we can improve the performance from two aspects: one is to change the feature types to detect more invariant and distinctive visual words, the other

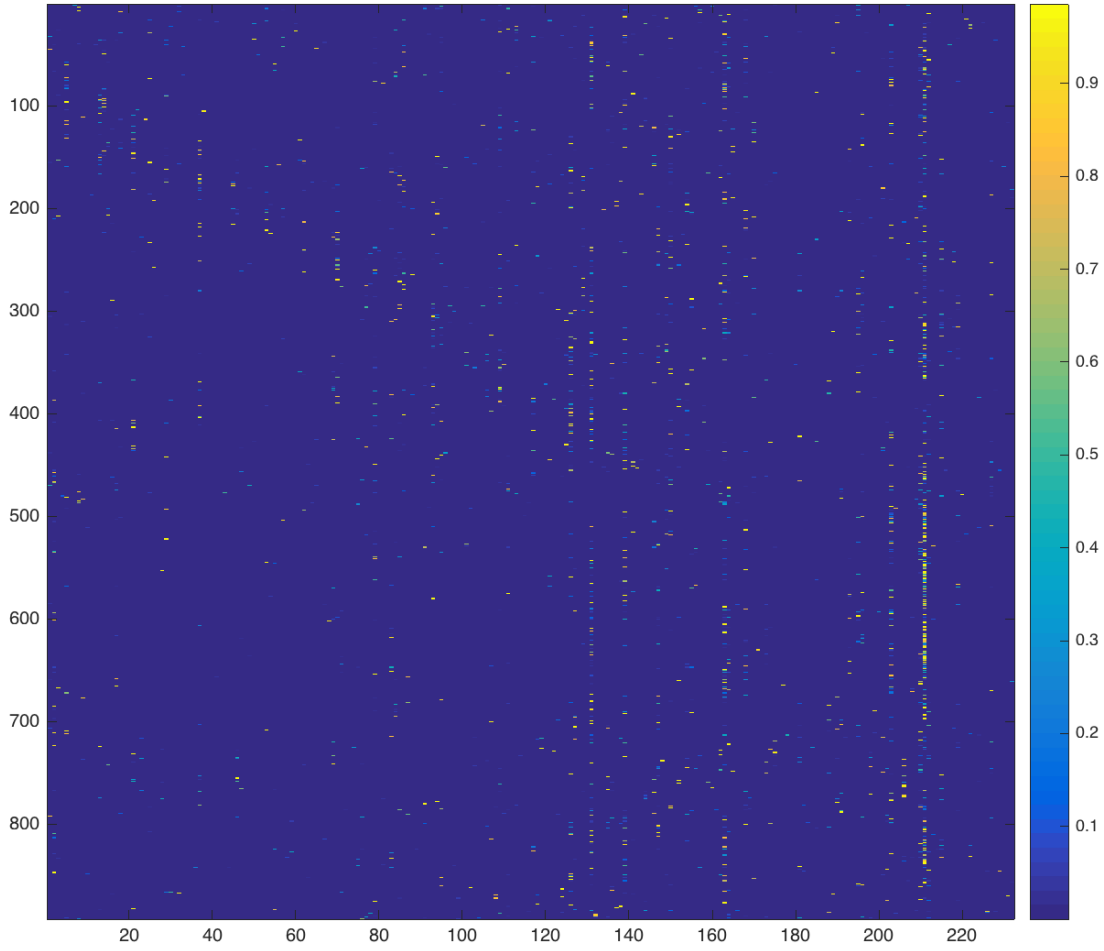


Figure 7.4: The confusion matrix is computed by the openFABMAP algorithm for 892 query images and 29×8 Street Views. As observed, this method can detect several correct matchings but there are still many false positives.

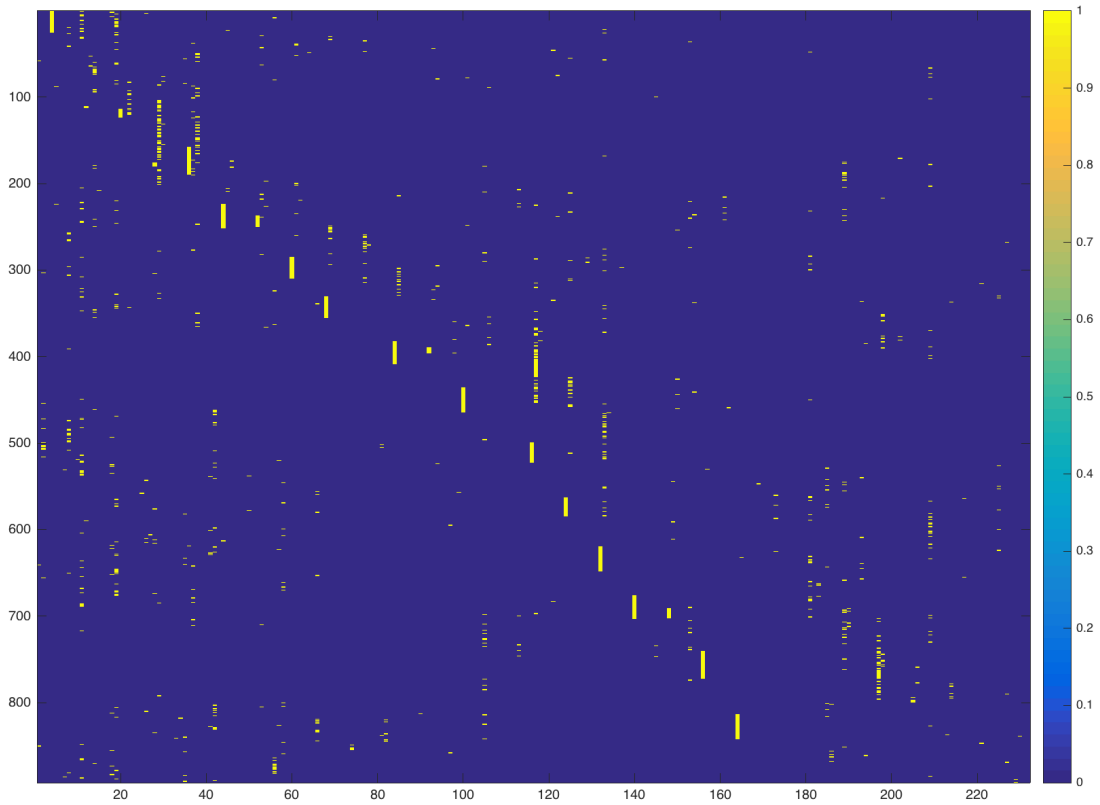


Figure 7.5: The confusion matrix is obtained by our proposed combined BoW for 892 query images and 29×8 Street Views. Results are closer to the ground truth than openFABMAP and the classic SIFT Bow.

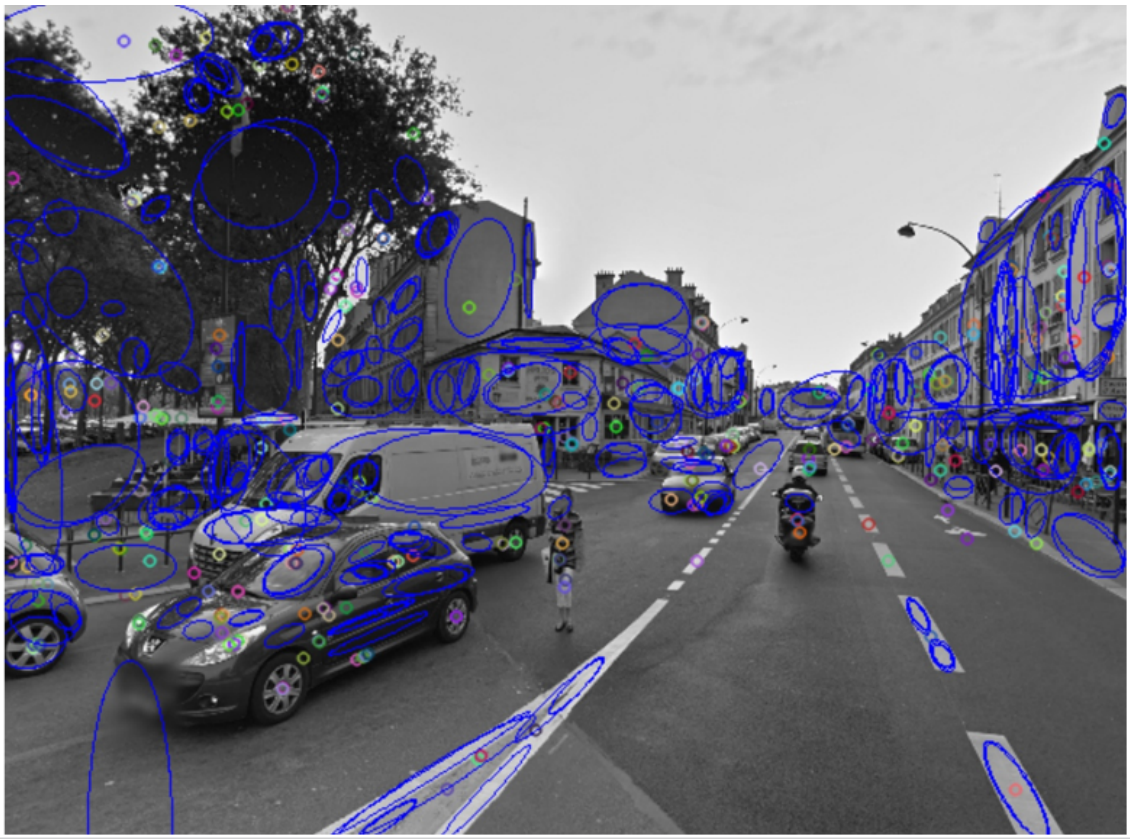


Figure 7.6: An example of the extraction of SIFT (colorful circles) and MSER features (blue ellipses).

is to determine an optimal number of visual words to control the trade-off between being distinctive and being robust.

Considering the changes in viewpoint, illumination and occlusion, we construct two independent dictionaries generated from the SIFT and MSER features, and then normalize them into one dictionary. Figure 7.6 illustrates an example of the extraction of both SIFT and MSER features. The final dictionary can thus take into account both local and regional feature descriptors. Our combined BoW outperforms the previously evaluated algorithms. We follow the typical BoW retrieval techniques to represent database images as numerical vectors quantized by feature descriptors, and to perform a hierarchical clustering (K-means) of the image descriptors as a classic BoW algorithm, see Figure 7.7. After the weighing strategies, all images are represented by the visual words. A topological localization is estimated according to a distance criterion based on the vectors similarity. In our test, we also use the classic TF-IDF reweighing and the efficient cosine similarity distance metric. As shown in Figure 7.5, the combined BoW outperforms the simple SIFT BoW when some regional visual words are integrated with keypoint visual words. Presumably, the regional visual words help to add more constraints to select better corresponding images.

However, there is still several false positives in this method and in practice, we adopt the geometric consistency (the performance of matching features) to check that the referenced image is matched with the query image, especially for the initialization. Moreover, in the online step, we also check the distance between two

Representation of all Street Views		
Bag No.	1	2
Feature Extraction		
Detector	SIFT	MSER
Descriptors	368979	75394
Parameterization		
Size of bag	10000	1800
IF-ITF reweighing		
Combination of 2 bags		
Search by the cosine similarity metric		

Figure 7.7: Procedure of the combined bag of words and its setup of parameters.

referenced images corresponding to consecutive online frames to avoid outliers.

7.2.3 Database Construction

As a rule of thumb, the bigger the database is, the slower will be the information retrieval from it. The aim now is to facilitate the run-time search even if a metropolis database is constructed. In a natural manner, we explore the intra database similarities and integrate the potential topological information to reduce the computational cost. More importantly, even if our place recognition does not detect the optimal similar referenced image, we can leverage the intra database similarities to avoid missing the correct one.

Figure 7.8 shows the intra-similarity between images within the whole database $D_{DB\times DB}$ as well as the relationship between the database and query images $D_{DB\times Q}$. The results, detailed in the figure caption, show a certain regularity to locate top similar images *w.r.t* a query image. To some degree, these regularities reflect the topology and inner connections in the GIS, *e.g.* all database images are downloaded successively according to the Street View route planner. Intuitively, neighboring panoramas share similar appearances. Their similar rectilinear images also share an analogous global yaw angle regarding the north direction. We denote these high similarities by yellow parallel lines and vertical dark lines in $D_{DB\times DB}$ and $D_{DB\times Q}$ respectively.

All these regularities make our database a sparse "searching map", which is advantageous to significantly reduce the number of image comparisons in the online phase. It is summarized as follows:

- In a piecewise route database $D_{DB\times DB}$, for each database image I_r we register its top k similar database images' location $I_{0\ldots k}$ in an array $D_{DB\times r}$.
- A maximum appearance distance $dist_{max}$ is defined as the farthest radius around a panorama where its neighboring panoramas share an overlapping view. Normally, 6 to 8 nearest panoramas share a common scene around 50

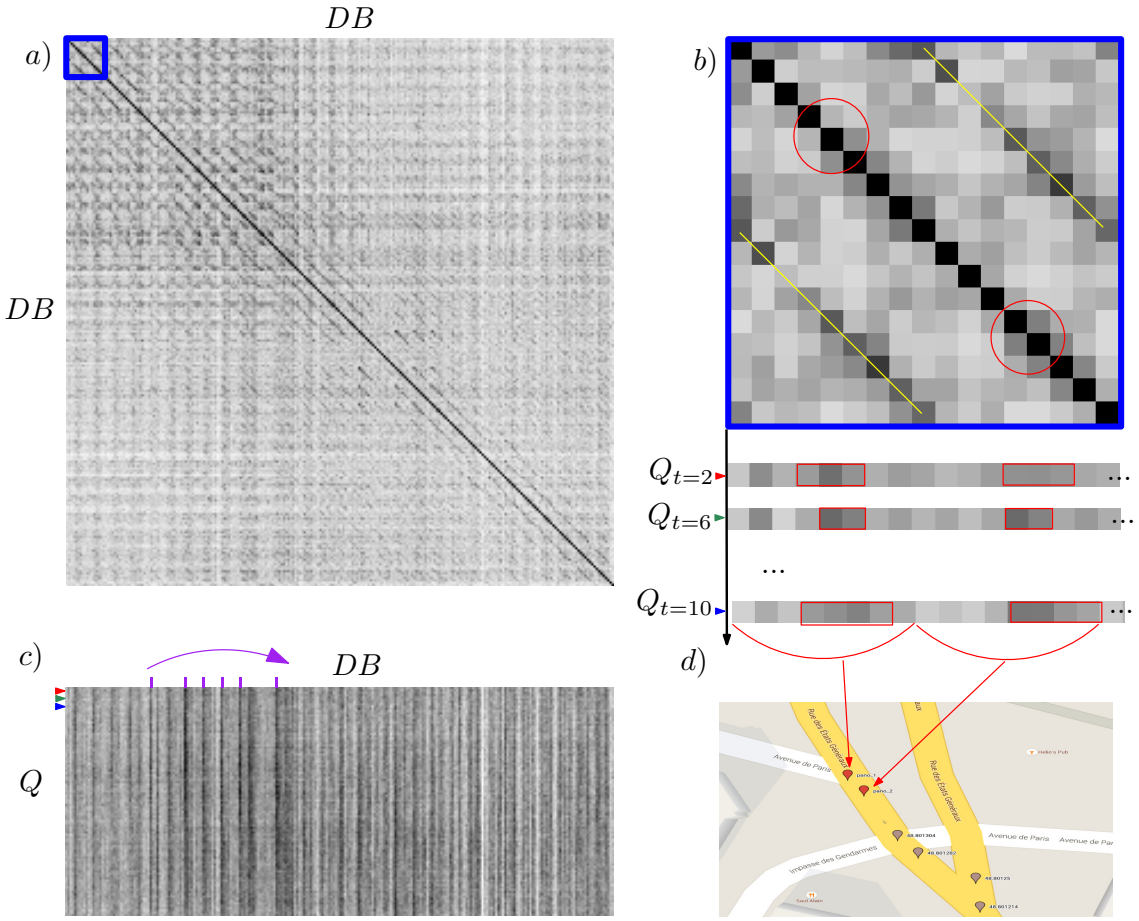


Figure 7.8: Illustration of distance matrices and intra-relations: *a*) The intra-database symmetric matrix $D_{DB\times DB}$. The rows and columns represent 232 rectilinear views generated from 29 panoramas in a 350m urban route, and the matrix intensity is computed by mapping the cosine similarity from $[0, 1]$ to $[0, 255]$, therein, the darker pixels depict the higher similarities and diagonal values are always equal to 255. *b*) The above matrix is a close-up view *w.r.t* the blue rectangle in *a*), it shows two obvious yellow lines with high intensity, that are parallel to the diagonal line indicating a location regularity for similar images in the matrix. *c*) The distance matrix $D_{DB\times Q}$ rows represent the 232 database images and columns represent a query sequence with only 100 frames. We only display 100 frames instead of a whole sequence with 892 frames. Several darkest vertical lines are highlighted by purple marks, meaning that a short query sequence can find its most similar database images only in few range of the database as shown by the purple arrow. Let us consider the candidate locations when teh query time steps equal to $t = 2, 6, 10$. The close-up of these timestamps is shown in *b*). We can find that the successive or close frames have the similar referenced panorama, but the best similar referenced image would be different certain yaw offsets even in the same panorama. The red cycles and rectangles represent top similar candidates from the same panorama. *d*) The panoramas are searched at the time steps of *b*).

to 70 meters. In $D_{DB\times DB}$, the $dist_{max}$ can be fixed as 48 or 64 matrix index steps.

- At the beginning, we can localize the first incoming query image by searching all the database images. Then for an ordinary query image, its searching range can be narrowed to all $D_{DB\times r}$ within the threshold $dist_{max}$.

According to our experiment, the above approach can avoid nearly 90% of the comparisons without any retrieval loss compared to a whole database search. A symmetric comparison matrix is computed from the database in order to reduce the retrieval time in the online phase. It also serves as an index reference in the optimization step. In the online stage, for every vehicle query image, we extract top similar images from the database as the topological localization and then estimate relative poses between each other. Finally, a LBA is employed to obtain the global metric localization from all estimated poses and corresponding 2D-to-3D matching constraints.

7.2.4 Image based Metric Localization

In the previous sections, a candidate set of images has been retrieved from the database and an intra similarity matrix of the database is computed. In order to realize a metric localization, high-inlier feature correspondence between camera images and the candidate set should be guaranteed. Other than a statistic comparison with the BoW, the conventional perspective matching is done by the following pipeline [75]: (a) a set of keypoints are extracted in both query and database images, *e.g.* SIFT; (b) their descriptors are matched by nearest neighbor search algorithms, e.g. FLANN; (c) matching outliers are rejected by the ratio test and geometric verification using constraints from the homograhy or the fundamental matrix, e.g. 8-point RANSAC algorithm. As stated in Chapter 7, we choose a more well-performing handcrafted feature, KVLD, to construct 2D-3D matchings and then use them to compute poses between referenced images and the current one.

Pose Estimation and Optimization

Once true positive referenced images are searched for the query image, the 2D-3D matching and PnP problem can be easily solved. The PnP problem has been well studied in computer vision. Since we also take advantage of the intra cosimilarity matrix to find several referenced images, a local graph optimization can be adopted to estimate an optimal pose. In this way, the referenced Street Views act as the "keyframe " in the visual-inertial SLAM to compute a consistent trajectory.

Figure 7.9 depicts our pose estimation and global metric localization process. The vehicle captures images at camera states x_{t-1} and x_t consecutively. No odometric input is integrated between successive states. At the state x_t, the best database image r_1 associated to the query image is retrieved through the topological localization. As mentioned before, via the database $D_{DB\times r}$, the top $k-1$ analogous database images to the best one can be found, denoted as $[r_2, r_3, \ldots, r_k]$. We suppose the query image shares an overlapping view with these $k-1$ images as well. The constraints between them are found by the accurate matching features. With

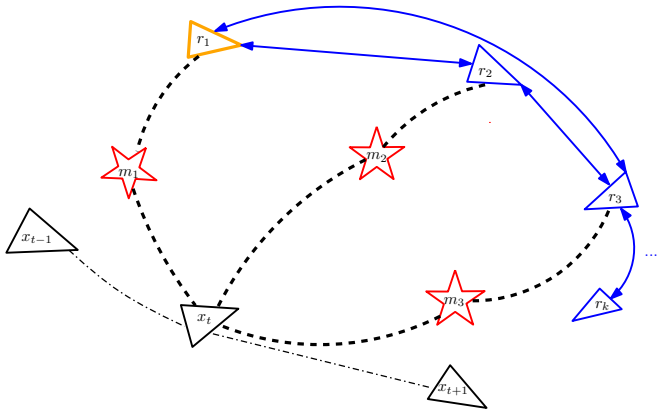


Figure 7.9: Illustration of Local Bundle Adjustment to estimate the global position of the vehicle. The vehicle, the best similar database image and other top $k-1$ similar database images are the triangles respectively colored in black, orange and blue. The red stars represent good matching features between the query and database images.

the help of depth maps in the database, the 6 DoF pose of the vehicle $\Theta = (\mathbf{R}, \mathbf{t})$, parametrized in Lie algebra $\mathbb{SE}(3)$, is computed by minimizing the reprojection error between each pair of images, *i.e.*:

$$\Theta^{\star} = \arg\min_{\Theta} \sum_{i} \pi \left(\| \mathbf{m}_i - \mathbf{P}\left(\mathbf{M}_i, \Theta\right) \| \right) \tag{7.2}$$

where $\mathbf{P}\left(\mathbf{M}_i, \Theta\right)$ is the image projection from the 3D point $\mathbf{M}_i$. In our case, $\mathbf{M}_i$ comes from Street View and $\mathbf{m}_i$ is the corresponding 2D point from the query image. The 2D-to-3D correspondence is improved by applying an invariant RANSAC (an ORSA in the kVLD feature). π is s a M-estimator based on the Tukey Biweight function [228]. We use the function π to substitute the squared-error of the residuals. It is more immune to the noise caused by imperfect correspondences because it increases less steeply than quadratic functions [124]. Moreover, the Tukey Biweight function can also suppress the outliers. 95% of the outliers can be suppressed when tuning the constant $t = 4.68$ [217]. In practice, we use this M-estimator based algorithm in an existing Library of CoSLAM proposed in [233] and the parameter $t = 3$. After this process, we can obtain k poses between the query image and its k similar reference images from Street View.

$$\pi\left(x\right) = \begin{cases} t^2 \,/\, 6 \left(1 - \left[1 - \left(\frac{x}{t} \right)^2 \right]^3 \right) & \text{if } |x| \leq t \\ t^2 \,/\, 6 & \text{if } |x| > t \end{cases} \tag{7.3}$$

As we stated in Chapter 5.1, there are estimation errors for every pair 2D-3D matchings and it causes the triangular error consistency problem (see Figure 5.20). In order to get a consistent result, we put all the k poses and 2D-to-3D correspondences into a LBA to refine the vehicle's pose and its global position. We use the Levenberg-Marquart algorithm in the optimization framework *g2o* [108] as our non-linear least square solver. The error regarding the feature detection is assumed to follow a Gaussian distribution. The k geotagged views extracted from panoramas forms the constraints in an optimization problem as shown in Figure 7.9 and their

camera configurations are given as before. This process is realized in the cartesian coordinates with the UTM projection. Since UTM values are very big, we select a starting point as a reference and normalize all the large UTM positions (in both online and Street View datasets) by subtracting this reference. The final pose is converted back to the real UTM and its geo-coordinates in WGS84.

We use many techniques to obtain a good final pose estimate: an invariant RANSAC to remove the outliers in 2D-3D matchings; a M-estimator based RANSAC to estimate k poses, and a LBA to optimize the final pose estimation. As mentioned, the PnP problem requires $n \geq 3$ correspondences. Since we adopt several optimization schemes, it is clear that the more inlier matchings we obtain, the better our algorithm performs. In the experiment, we set the minimum number of feature matchings (validated by ORSA) as 3. The estimated pose remains until a new localization is computed based on this threshold. Moreover, before employing the LBA algorithm, we compute the topological distance between the estimated pose to its most similar Street View (referred as the"the estimated topological distance"). If the estimated topological distance is larger than 12m, we keep the pose calculated before. These two thresholds are fixed according to our experiments and the following evaluation will demonstrate the reason.

Figure 7.10 shows a metric global localization in the tested segment where 892 query images and 29 panoramic Street Views are detected. According to our criteria, there are 153 metric positions obtained with an average error of $3.37m$ $w.r.t$ the RTK-GPS (78.5% of them within a 2m accuracy). An example of matches between the query and the retrieved Street Views is depicted in Figure 7.11. As observed, our algorithm can retrieve 3 Street Views for an online image but only the Street Views from the "pano2" and "pano3" can extract validated matchings with it (namely $k = 2$ in a LBA). We can thereby compute 2 initial poses from them and then optimize the final pose by a LBA.

To conclude, we summarize this coarse to fine localization system in Algorithm 3.

Evaluation and Discussion

The performance of our coarse-to-fine algorithm depends closely on both the topological and the metric localization part. In fact, if the coarse topological localization fails, the further metric part based on it works in vain. However, in the city scenario, it is difficult to construct a strict ground truth to evaluate the topological localization. As expected, the visual overlap must exist between the query and database image for the BoW algorithm to work well. Therefore we can evaluate the topological localization based on the geometric consistency. We assume that the topological localization works only if a query image lies in a 20m[3] radius around the Street View and there are also at least 3 matchings between them. The boundary radius distance (referred as "the real topological distance") is calculated between the RTK-GPS corresponding to the query images and the geodetic data of the retrieved Street View. After verifying the pairs matches along a street with 29 panoramic Street Views, our combined BoW method is able to recognize 100% visually similar database images corresponding to query images.

Regarding the metric localization performance, although the accuracy is higher

[3]In the Google Map API, 20m is a default searching boundary to find a closest Street View for a customized position input.

Figure 7.10: Bird's-eye view of the metric global localization. The red points represent the locations of the Street View cameras. The red and blue lines mark RTK-GPS ground truth and the estimated positions of the monocular camera respectively.

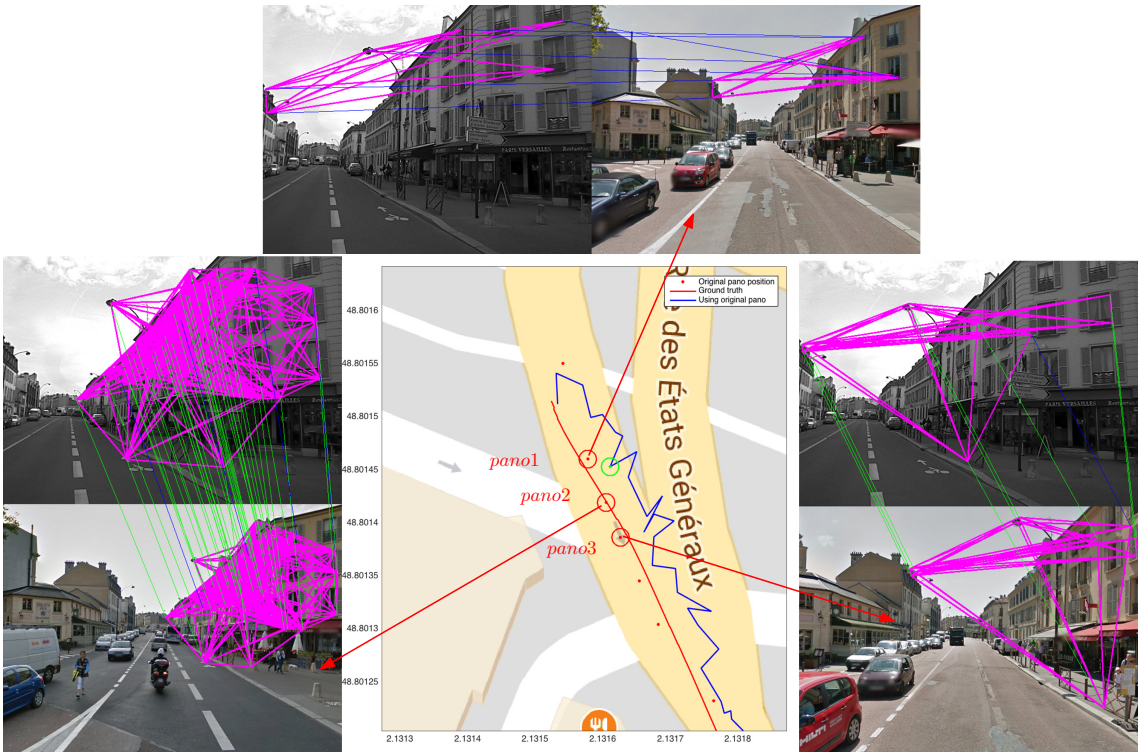
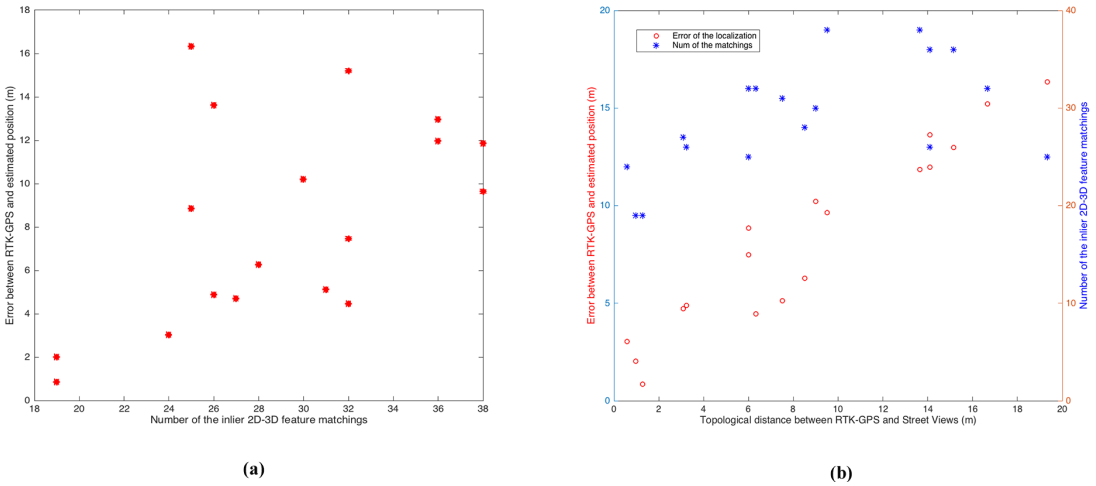


Figure 7.11: An example of 2 pairs of validated matches between a monocular image and its retrieved Street View images by kVLD descriptors. The positions of Street Views and the estimated localization are displayed in red and green circles respectively. kVLAD matches verified by ORSA are the green lines and those verified by graph matching are the blue lines.

Algorithm 3 Metric global localization in the urban area

Input: Street View panoramas $\mathbf{S} = \{S_1, S_2, \ldots, S_n\}$ and their depth maps $\mathbf{D} = \{D_1, D_2, \ldots, D_n\}$
Input: Query images $I_q = \{I_1, I_2, \ldots, I_m\}$ captured by a vehicle driving in the city
Output: Global location of the vehicle

for $i \leftarrow 1$ **to** n **do**
  Rectilinear processing of $\mathbf{S}$ and $\mathbf{D}$
  Feature extraction and BoW training
  Construction of DB as the final BoW dictionary
end for
Calculation of the intra-distance matrix $D_{DB\times DB}$ and speed-up matrix $D_{DB\times r}$ ▷ *cf.* Section 7.2.3
Up to here the algorithm is implemented offline.
for $t \leftarrow 1$ **to** n **do**
  Parametrize I_q by BoW
  Search I_q's best similar image I_r in DB
  Get the top k similar database images of I_r in $D_{DB\times r}$
  Histogram Equalization if possible: $I'_{t/r} = E\left(I_{t/r}\right)$
  for $j \leftarrow 1$ **to** k **do**
    Feature matching
    Pose estimation Θ_j ▷ *cf.* Equation 7.2
  end for
  $\Theta^\star = LBA\left\{\Theta_j\right\}$
  Recover the global position at step t
end for



(a) (b)

Figure 7.12: Metric error analysis with respect to inlier-match numbers and the topological distance: (a) The number of feature matchings does not affect the error a lot in the metric localization when place recognition works well. (b) The error of the metric localization depends a lot on the topological distance between the query images and their referenced Street Views, as shown by the red curve. The further away a query image is located from its referenced Street View, the less accuracy our system obtains. The blue curve illustrates that there is no obvious relationship between the topological distance and the number of matchings.

than the ordinary autonomous GPS (average error of 8m) [213], readers may still wonder about the discontinuity (only 153/892 frames localized). We analyze the cause of errors of the metric localization in two aspects: the topological distance between the chosen Street View and the position of the query image, and the number of inlier matches between the query and Street View image. The inlier-match number reflects the feature quantity that is used in the 2D-to-3D pose estimation process, and the real topological distance between the query and Street View image can describe the feature quality. Commonly, the less the further the features are tracked, the less accurate the localization is. We select all query images that retrieve the same Street View located at the coordinates $[48.801631, 2.131509]$ (namely "pano2" in Figure 7.11) and obtain the relationships in Figure 7.12. Our effective BoW algorithm guarantees the visual overlaps between query images and retrieved Street Views inside a 20m neighborhood. In this way, the inlier-match number does not affect much the error in the metric localization. However, the errors appear in a cumulative trend when the real topological distance increases between the query and the Street View. Despite the fact that the optimization improves the accuracy, many further away features only show small motions and can cause disastrous errors (even going past the bound of the topological localization). As a result, we set up the criteria of "the estimated topological distance" to eliminate lots of localization like this. We can make our localization more accurate by setting this distance to a small value but it will cause big jitters and provided less location estimates. In the test, we found 12m is a good compromise, which is also close to the average spatial distance between two successive panoramas.

7.3 Localization with augmented Street Views

As observed in the Figures 7.10 and 7.12, many online frames cannot be localized because of the discontinuity in the Street View dataset. The topological distance can affect the accuracy of the metric localization. It motivates us to make the topological distance smaller by augmenting the number of Street Views. Since there are both panoramas and associated depth maps (*i.e.* massive 3D points) in the constructed GIS dataset, we can generate artificial images by reprojecting the original panorama and depth map to arbitrary locations, see the image warping in Chapter 5.2.3. In order to reduce the sparse localization of Street View, we can synthesize virtual views along the trajectories.

In this section, we describe our refined localization algorithm in detail. As illustrated in Figure 7.13, the system is divided in two phases: in the offline stage, 4 useful types of data are extracted from Street View, including topology, geo-coordinate, panorama and associated depth map. We pre-process every panorama by rendering rectilinear images from its camera locus and at same time by generating virtual views deviating away from that locus. All rectilinear images are used to build a dictionary by a BoW algorithm. In the online stage, for every query image, we retrieve its most similar rectilinear images (namely referenced images in Figure 7.13) from the dictionary. Once referenced images are found, their neighboring virtual views are also used to construct 2D-to-3D transformation constraints. The global metric localization is also obtained by a LBA on all the constraints. More technical details are given in the following sections.

7.3.1 Database Augmentation

A city-scale Street View database is already too immense to deal with. Therefore, a reasonable way to augment virtual views is required and we should also consider how to efficiently render virtual views from the original GIS database, how to encode and index the whole database, and how many virtual views to be generated.

For the image synthesis, the depth map ought to be precise. In fact, we lose a large amount of the depth information in sub-urban, or highly-vegetation covered areas. Hence, this method should be strictly used in urban environment.

Instead of fixing virtual cameras in the centre of a unit sphere to get rectilinear images, the camera position of new virtual views is translated on the segments where VEDECOM vehicles were driven, see Figure 7.14. We are able to generate virtual views from a panorama in any position. Since vehicles are evolving along the road in urban areas, we only generate virtual images both forth and back along the trajectory. Inaccurate or missing depth information naturally causes artifacts and absent pixels, for example, sky pixels often disappear. Rendering virtual views from multiple panoramas can potentially improve the quality of the virtual views. However, here we only use a simple synthesizing process since a lot of artifacts are usually located in moving objects, such as vehicles or pedestrians, which are not the same to those that appear in our own query images. Our compact preprocessing enables every panorama to have suitable synthesized virtual panoramas in its neighborhood. Then rectilinear virtual views can be obtained from virtual panoramas via our backprojection model.

The rendering pipeline is as follows: for every pixel in a virtual panorama, a ray

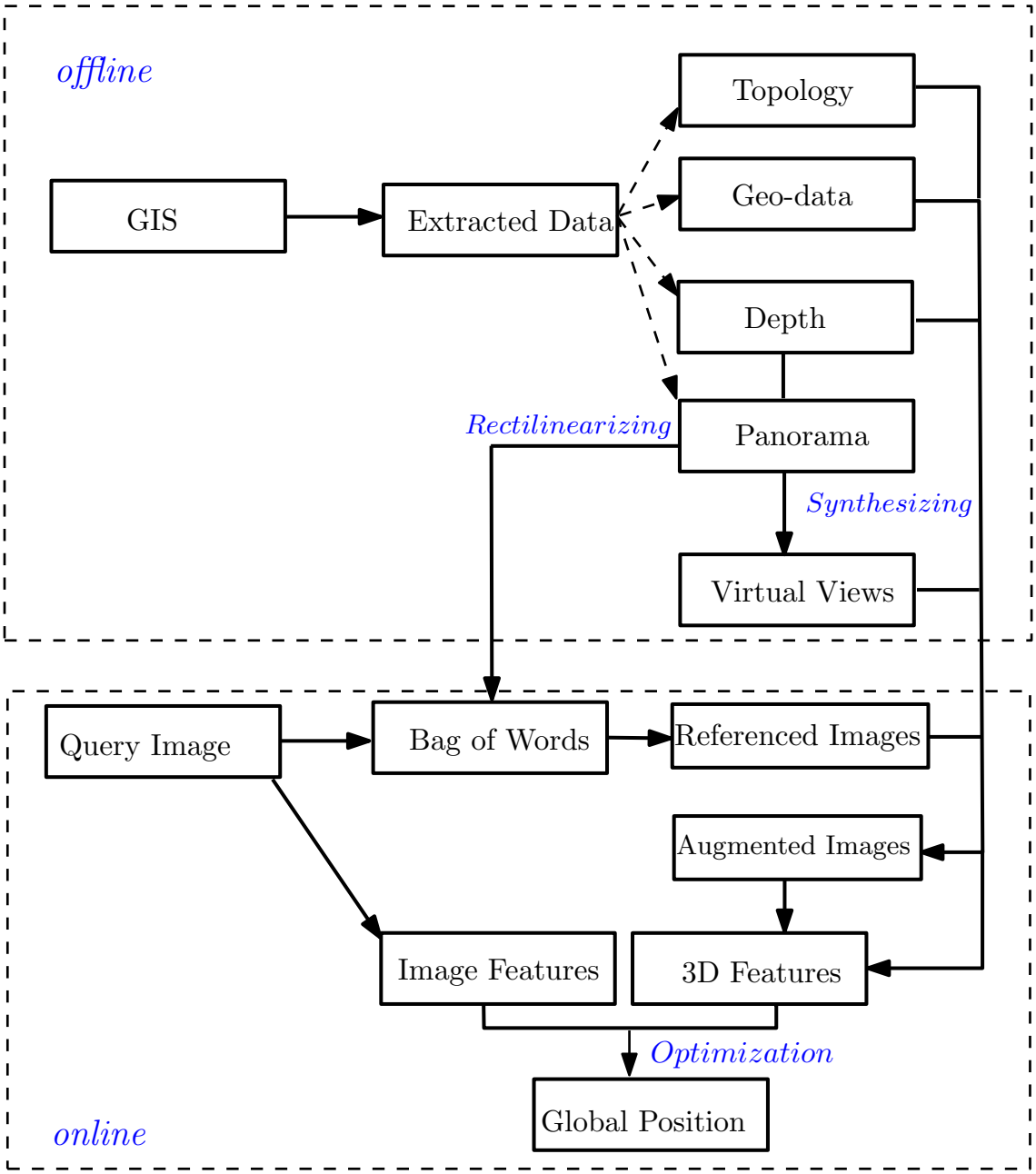


Figure 7.13: Flowchart illustrates workflows for the localization with augmented Street Views.

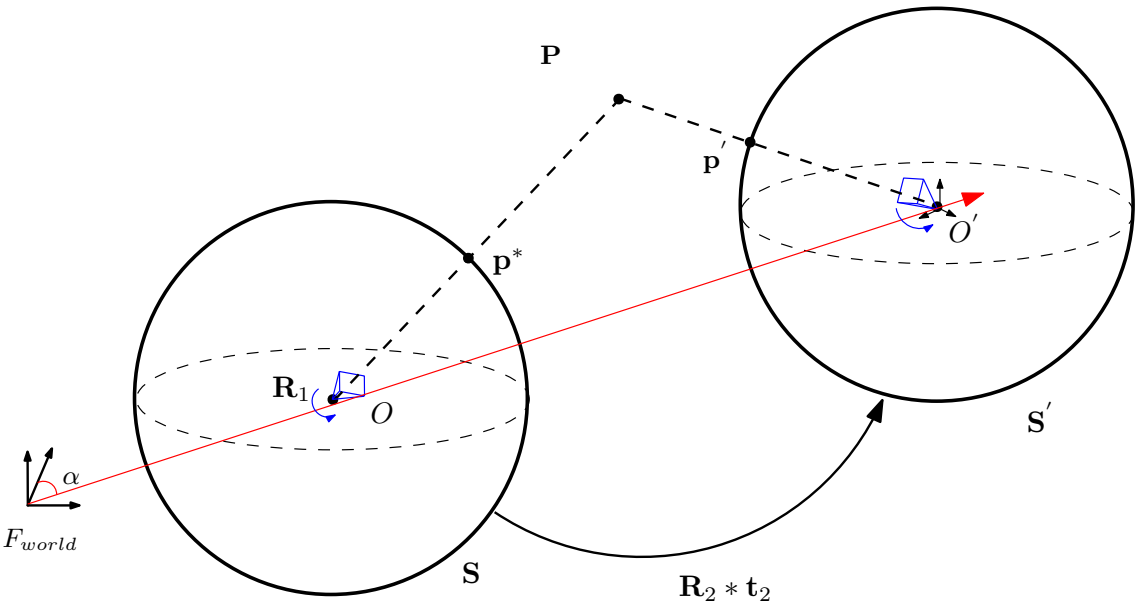


Figure 7.14: A virtual panorama at centre point $O^{'}$ is constructed from the original panorama at point O.

is cast from the centre of a virtual camera and intersects the planar 3D structure of its closest panorama. The intersection is then projected back to the spheric panorama and then the depth map is updated according to the transformation. Next, using the back-projection model, we extract rectilinear virtual views from the virtual panorama. Pixel values are still rendered by the bilinear interpolation. For more detail, please refer to the depth image based rendering algorithm [143].

As illustrated in Figure 7.14, we translate an original panorama $\mathbf{S}$ located at a point O to form the virtual panorama $\mathbf{S}^{'}$ at point $O^{'}$. The translation $\mathbf{t}_2$ is realized along the trajectory direction, which is defined by a global yaw angle α in our GIS data. Here, $\mathbf{t}_2$ is denoted in an East-North-Up world coordinates F_{world} by:

$$\mathbf{t}_2 = \begin{bmatrix} l\sin(\alpha) \\ l\cos(\alpha) \\ 1 \end{bmatrix} \tag{7.4}$$

with l as the distance between two panoramas as $OO^{'}$. l is a parameter customized by ourselves. A 3D point $\mathbf{P}$ is the intersection between the virtual ray and the planar 3D structure of its closest panorama. Thus it can be defined in the sphere coordinates at point $\mathbf{S}$ with a depth information ρ in a homogenous form as:

$$\overline{\mathbf{P}} = \begin{bmatrix} \rho\cos(\theta)\sin(\phi) \\ \rho\sin(\theta)\sin(\phi) \\ \rho\cos(\phi) \\ 1 \end{bmatrix} \tag{7.5}$$

Therefore, the point $\overline{\mathbf{P}}$ can be projected to $\mathbf{p}^{'}$ on the virtual unit sphere $\mathbf{S}^{'}$by transforming to the coordinates of S (at center O):

$$\mathbf{p}^{'} = \frac{\mathbf{P} + \mathbf{R}_2(\alpha)\mathbf{t}_2}{\|\mathbf{P} + \mathbf{R}_2(\alpha)\mathbf{t}_2\|} \tag{7.6}$$

Figure 7.15: Recall the Figure 5.11: a Street View panorama at location $[48.80056, 2.136449]$ is extracted in summer in the test area.

where $\|\mathbf{P} + \mathbf{R}_2(\alpha)\mathbf{t}_2\|$ is the updated depth $\rho^{'}$ and $\mathbf{R}_2$ is the rotation matrix deduced from the global yaw angle α. Thus, $\mathbf{R}_2(\alpha)\mathbf{t}_2$ describes the relative translation between two panoramas. The intensity value at $\mathbf{p}^{'}$ will be interpolated from $\mathbf{p}^{*}$ if satisfying $\rho^{'} > 0$. Then, rectilinear virtual views are registered by the former back-projection model.

In order to lessen the influence of absent pixels, we create more virtual pinhole cameras (12 in practice) to capture more details in virtual views for a good matching. Figure 7.16 shows a generation example of 12 virtual views from the panorama of Figure 5.11 (recalled in Figure 7.15)at location $[48.80056, 2.136449]$ by moving forward 1m to $[48.801499, 2.131564]$ along its topology.

7.3.2 Refined Result and Discussion

The intrinsic parameters of the virtual cameras are also fixed according to our own MIPSEE camera, which is similar to our backprojection model in Chapter 5.1. In order to qualify the metric accuracy, we only selected localization runs where the RTK-GPS reached a below-to-20cm precision. The same urban trajectory example of Section 7.2 is tested, see Figure 7.17. For every panorama, we generated its virtual views both forward and backward. In the experiment, virtual views with too much missing pixels (due to the limit of the synthesizing process) are rarely selected as referenced images for a query image. Yet in an indirect way, the augmented intra cosimilarity matrix helps to find more potential top matching referenced images for a query image, namely some much closer virtual views are used in the PnP procedure.

Translation Distance Evaluation

The localization accuracy decreases with the increase of the distance to its referenced Street View. Thus, the translation distance l must be chosen carefully. The main criteria to fix the distance between a panorama and the virtual views depends on the following aspects:

- Null pixels and artifacts will be produced if the synthesized views are far away from the rendered panorama. Normally, the farther they are, the more pixels

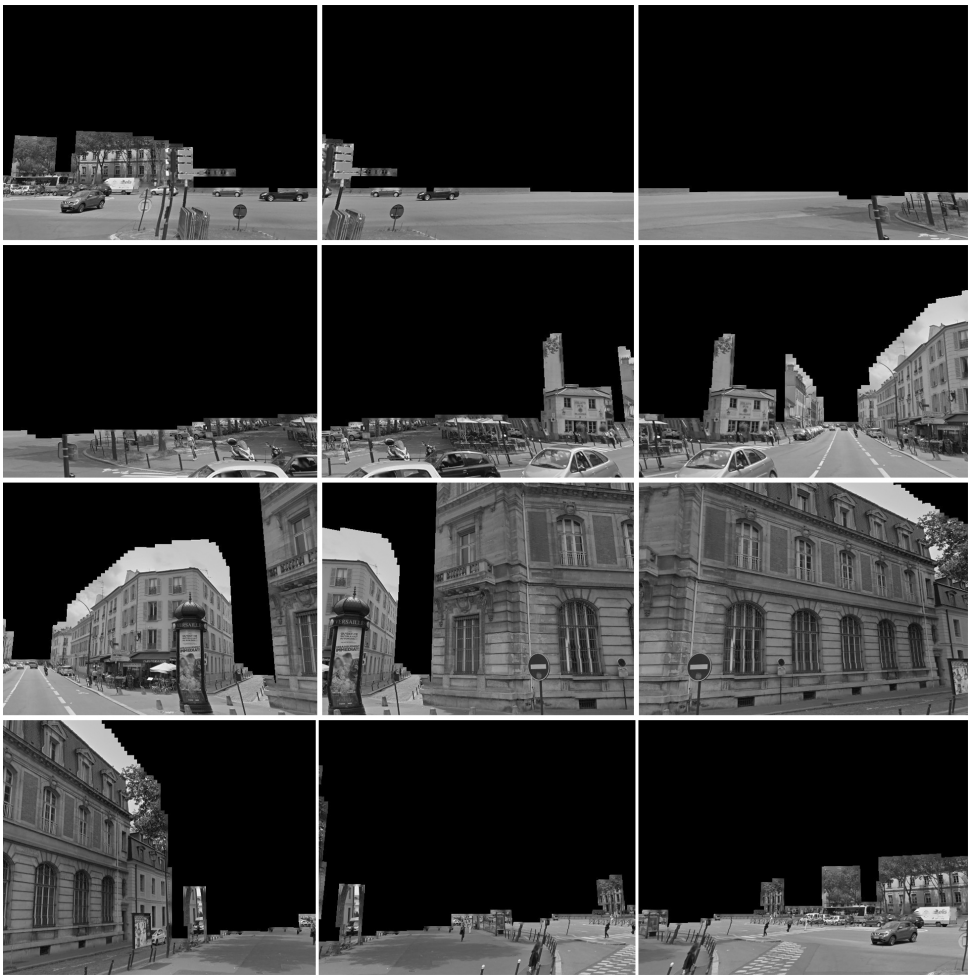


Figure 7.16: Rectilinear synthesized views from the panorama of Figure 5.11. The black pixels lack the depth information.

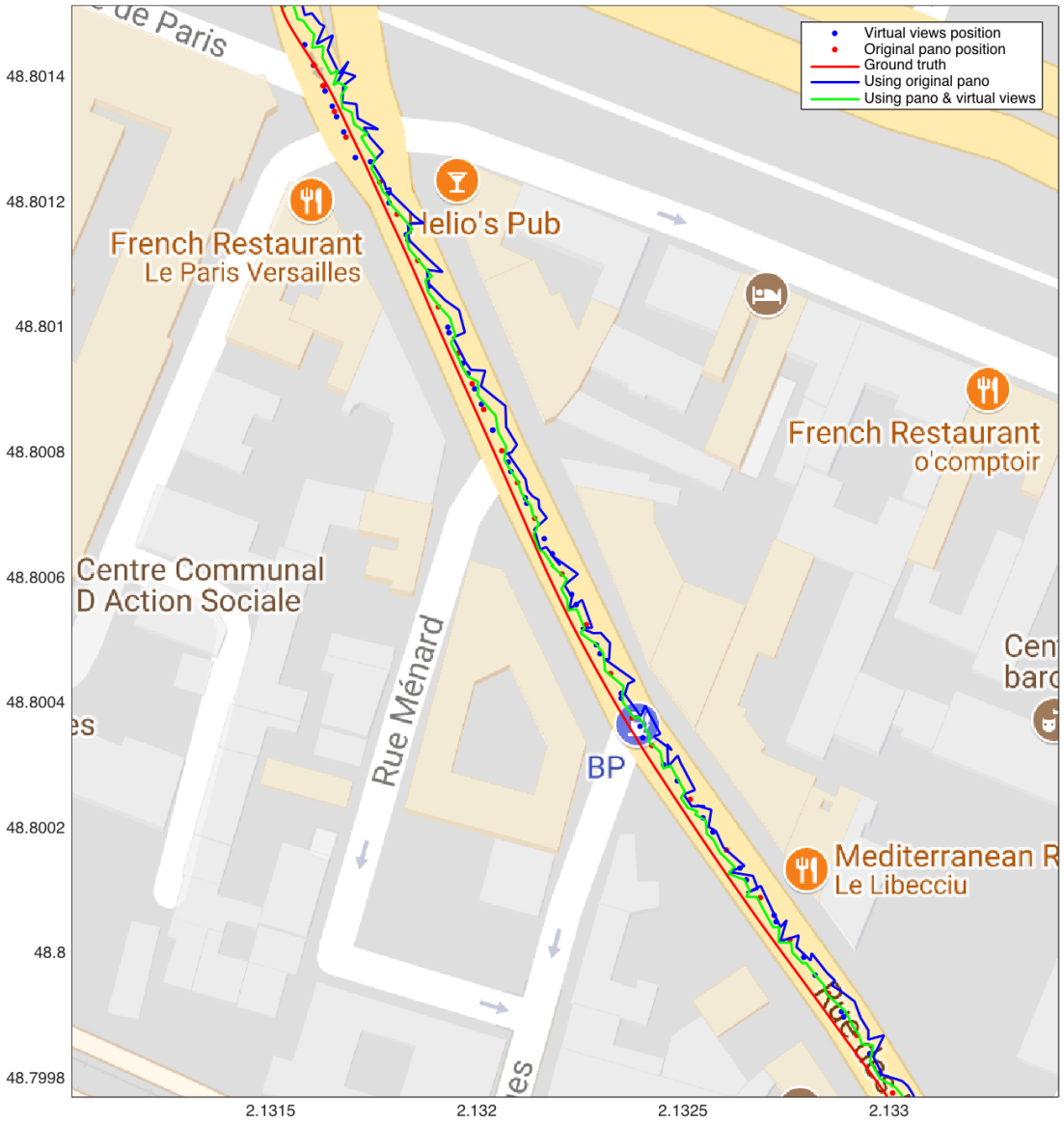


Figure 7.17: The output from a single localization run using original Street Views and synthesizing virtual views: The trajectories obtained with/without virtual views are plotted in green/ blue respectively. The ground truth in red line is recorded by a centimeter-level RTK GPS.

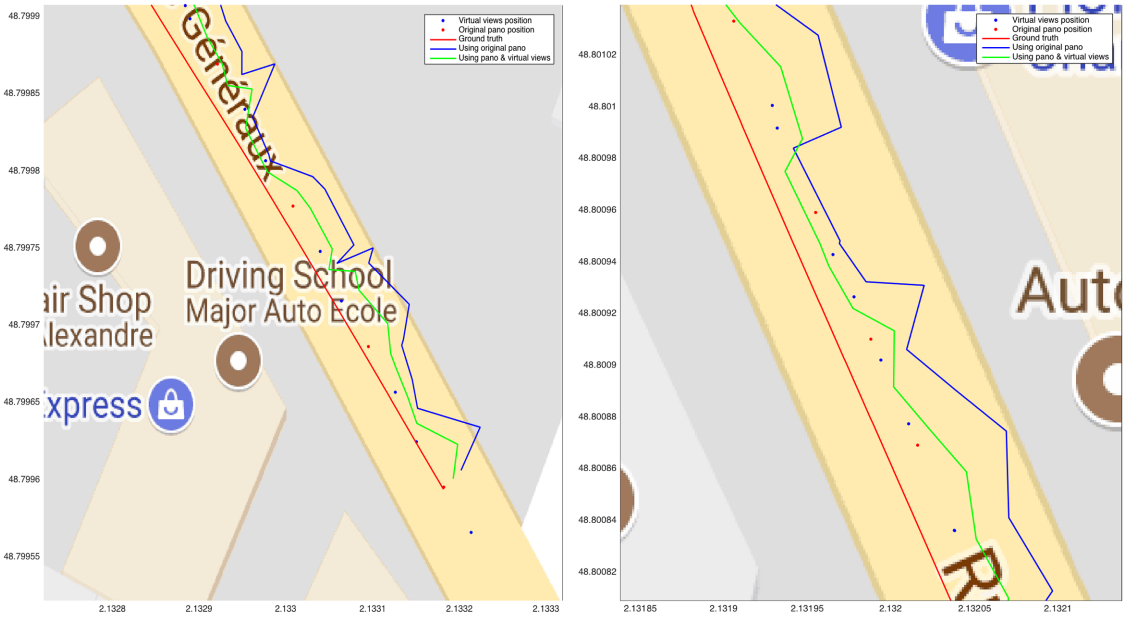


Figure 7.18: The close-up views from the localization result.

are lost. Consequently, our following metric localization will be influenced significantly.

- We use the global yaw angle to determine the segment we are traveling on in the topology. Nevertheless, a long translation distance will generate virtual views out of the current street, especially on narrow crossroads where multiple panoramas meet together. Synthesizing views in such cases is not useful.

- Also, we expect that the final augmented Street View can achieve a uniform distance between every consecutive views' positions. If the translation distance is too small, uniformity can hardly be realized.

We tried several translation distances along the same trajectory in order to find an ideal choice. The geodetic locations were visualized to measure their uniformity. Table 7.1 shows the evaluation results. When the translation was fixed to 4m, as shown in Figure 7.17 and Figure 7.18, we acquired a compact and uniform distribution of geo-tagged virtual views. After discarding virtual cameras located in buildings[4] , we synthesized 53 virtual panoramas in total. Finally, 53×12 synthesized views were added to the Street View database.

Robustness & Accuracy of Localization

Once the augmented database is constructed, the global urban localization can be realized according to our proposed method. Figure 7.17 shows an overview of two localizations estimated respectively by the original and the augmented Street Views, and their close-up views from Figure 7.17 are provided in Figure 7.18. As we can see, the performance of the metric localization depends on the distance between the query image and the Street View retrieved in the topological localization. Along the

[4]It appears seldom only at traffic intersections.

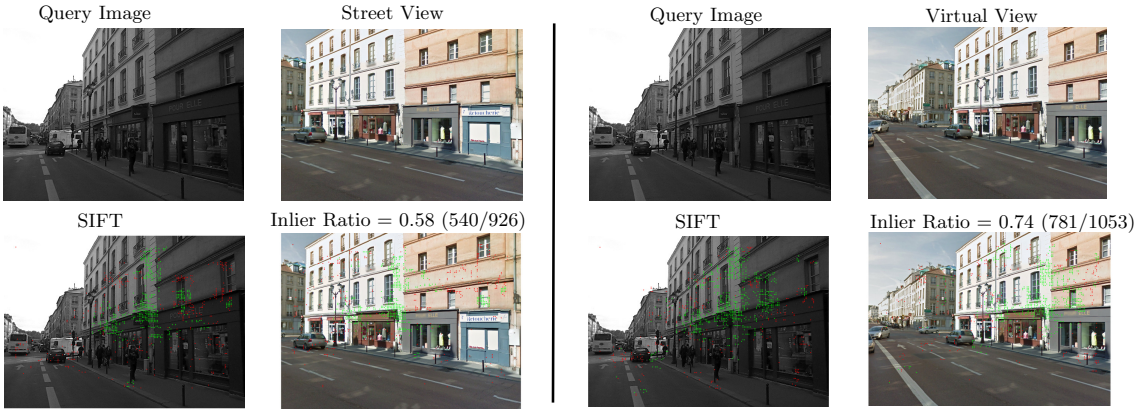


Figure 7.19: The same query image is matched with highly similar Street View retrieved by the BoW and with corresponding virtual view. The FLANN based matches are displayed in red and geometrically verified matches are shown in green. The inlier ratio is measured by proportion of geometrically verified matches.

whole trajectory, we only plotted the localization when the topological localization reached a 4m accuracy.

Translation distance	$2m$	$4m$	$6m$	$8m$
Invalid camera positions	0	4	4	6
Uniform distribution	N	Y	Y	N
Ratio of virtual views with more than 50% null pixels	0.52	0.657	0.78	1

Table 7.1: Evaluation of the translation distance.

	Original Street View	Augmented Street View
Continuity	116/892	272/892
Average Error	$3.96m$	$3.43m$
Ratio in $[0m, 1m[$	19.68%	38.57%
Ratio in $[1m, 2m[$	32.70%	21.82%
Ratio in $[2m, 3m[$	25.13%	22.86%
Ratio in $[3m, 4m]$	22.49%	16.75%

Table 7.2: Evaluation of the localization performance.

As can be seen from Figure 7.17, our approach generally works more smoothly and accurately than the localization without virtual views. The path estimated by augmented Street Views is much closer to the ground truth. Virtual Street Views can reduce the accumulative drifts effectively when the vehicle is far away from the original panorama, prevent that the localization jumps back while the vehicle is moving forward, make the localization path smoother, see Figure 7.18.

We quantify the performance by using several statistic terms as calculated in Table 7.2. First, we define the continuity as a term to evaluate how many query images can be located within a 4m accuracy after the metric localization. We calculated their average error regarding the ground truth during the whole run. The average error means the total error in distance with respect to the number of localized frames.

As seen from the table, the average accuracy of the localization improves considerably and 60.39% of the used query images can reach an error between $[0m, 2m[$. In contrast, most of the localization precision stays between $[2m, 4m[$ using the original Street View.

We also analyzed the 2D-3D matches between these localized query images $w.r.t$ their Street View and virtual views in the work of [220], see Figure 7.19. In the literature, virtual views are often used to improve matching under extreme viewpoint changes. In our former work, we adopted a complex Virtual Line Descriptor (kVLD) to determine the inlier feature point correspondences when the query image was far away from the Street View. We also demonstrated that augmented virtual views can reduce the influence of viewpoint changes and increase inlier matches between query and referenced images as well. After using virtual views, inlier matchings increase significantly. Simpler features, like SIFT, can be used instead of kVLD.

In summary, the main contribution of this algorithm is thus an extension of our former framework. We can compensate the sparsity of Street Views and improve the localization precision with the construction of augmented Street Views database.

7.4 Generalization to the whole Test Area

In the previous sections, we only selected a relatively short urban segment (the sequence No.11 in Table 7.3) to do the experiment. When using this sequence, we verified that there are good urban appearances in this segment, namely with urban buildings without heavy vegetation or moving objects. Moreover, online captured frames are also free of illumination and other dynamic noise. In such an ideal scenario, all these frames are well recognized and localized by their corresponding Street Views. Now, we will test whether the results of our algorithms can be generalized to the whole test area. Before doing that, we add the following modifications to refine our algorithms:

- If the first online frame is not localized by the combined BoW process, our algorithms skip to the next frame until finding a localized frame as a starting point. This strategy contributes to ignoring some initial online frames captured with blur or important illumination. It is common in our online datasets, see Figure 7.20(a).

- In the localization process, if one query frame is considered as a wrongly referenced image by the combined BoW[5], it is ignored.

- During the localization process, if one frame is not localized due to insufficient 2D-3D matchings, the position stays unchanged and our algorithms would skip to the next frame until a localization can be computed. It appears very often when some moving vehicles block the scene to the online cameras, like in Figure 7.20(b).

[5]As we stated before, a referenced image should be checked with the previously detected reference image and be verified if it is a good match according to the distance between them. If the distance is too important, it is a wrong match. In the experiment, we set this threshold to 12m which is also the search radius in the Google Street View extraction. Above this threshold, two panorama would not share a similar scene.

Figure 7.20: Some scenarios during localization: (a) Some big brightness blur frames were often captured in the initial step and they are ignored by our algorithms. (b) Noise from moving objects appears frequently in the urban environment.

Sequence	Vehicle	Number of Frame	Spatial Extent	Number of Street View	Original approach	Extended with Virtual Views
Seq0	Zoé	554	11×265	29	Fail	Fail
Seq1	Scénic	250	11×79	11	$3.76m$	$3.06m$
Seq2	Scénic	898	11×271	29	$3.29m$	$2.63m$
Seq3	Scénic	895	11×222	29	Fail	Fail
Seq4	Scénic	291	11×128	12	Fail	Fail
Seq5	Scénic	841	11×317	33	$3.88m$	$2.54m$
Seq6	Scénic	901	11×216	34	$3.72m$	$2.82m$
Seq7	Scénic	306	11×184	16	Fail	Fail
Seq8	Scénic	141	11×69	8	Fail	Fail
Seq9	Scénic	422	11×198	19	Fail	Fail
Seq10	Scénic	899	11×382	36	Fail	Fail
Seq11	Scénic	892	11×265	29	$3.37m$	$2.85m$
Seq12	Scénic	290	11×56	4	$3.59m$	$2.92m$
Seq13	Scénic	899	11×361	36	$3.84m$	$2.49m$
Seq14	Scénic	348	11×131	13	$3.26m$	$2.67m$
Seq15	Scénic	625	11×196	17	$3.34m$	$2.58m$
Seq16	Scénic	896	11×145	32	$3.06m$	$2.44m$
Seq17	Scénic	172	11×93	15	$2.98m$	$1.56m$
Seq18	Scénic	897	11×234	29	$2.51m$	$2.13m$

Table 7.3: Details of the captured sequences in the test area: the spatial extent is roughly measured by the localizing ruler tool in Google Maps (lane width×length). Average errors are computed respectively from the original approach (using original Street Views) and the extended one (using virtual views).

After these refinements, we tested our algorithms in the other sequences of the test area and the results are illustrated in Table 7.3 and Figure 7.21 to Figure 7.24. According to the table and figures, our algorithms can both achieve submeter level precision and using augmented Street Views can improve the smoothness and accuracy as well. Moreover, we observed that the more panoramas are located in the trajectory, the more accurate the localization is. Taking the sequence No.17 (see Figure 7.24) as an example, in a two-lane trajectory, we can extract more panoramas compared to one lane and it helps to improve the localization. In our algorithms, when a frame is not localized we do not change the position until a subsequent frame is well localized. Another example is that the vehicle can stop while the scene is still being recorded. Thus the same scene can be captured into frames and these repeated frames could be localized by our optimization process and rendered at different positions as shown in the sequence No.12 (see Figure 7.22). There is no doubt that tracking schemes could be used to eliminate these jittered estimates.

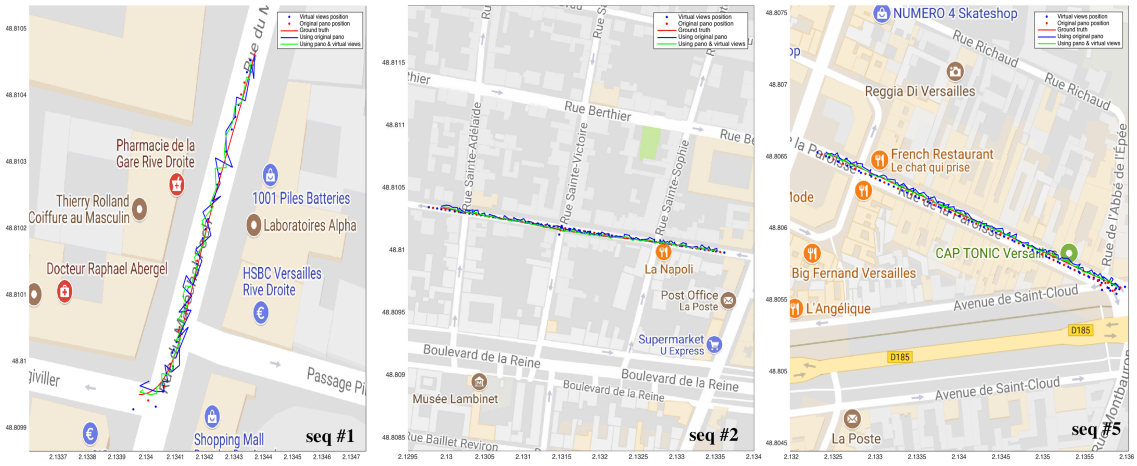


Figure 7.21: The localization output from sequences No.1, 2 and 5 using original Street Views and synthesized virtual views. The positions of original panorama, virtual views and ground truth are noted in red points, blue points and red line respectively. The results from original Street Views and augmented virtual street views are in blue and green lines.

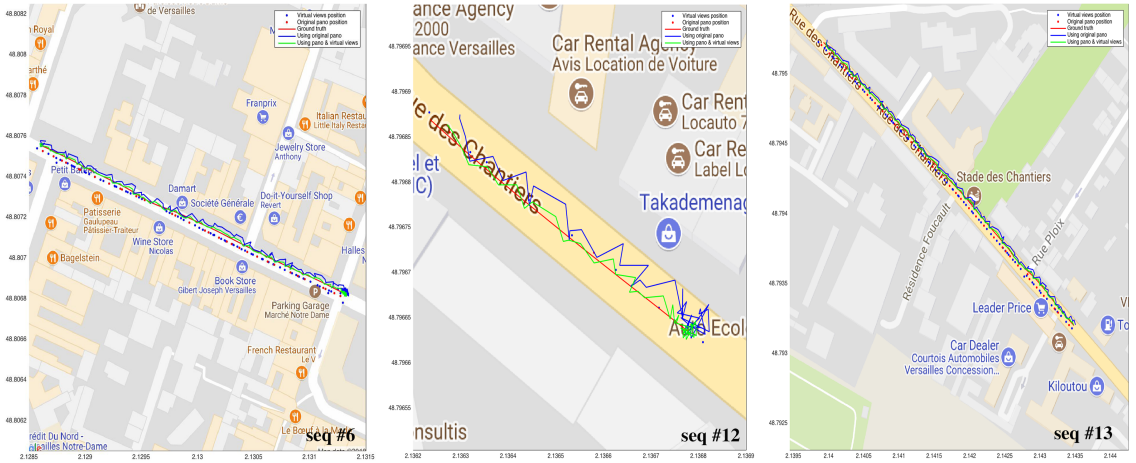


Figure 7.22: The localization output from sequences No.6, 12 and 13 using original Street Views and synthesized virtual views. The positions of original panorama, virtual views and ground truth are noted in red points, blue points and red line respectively. The results from original Street Views and augmented virtual street views are in blue and green lines.

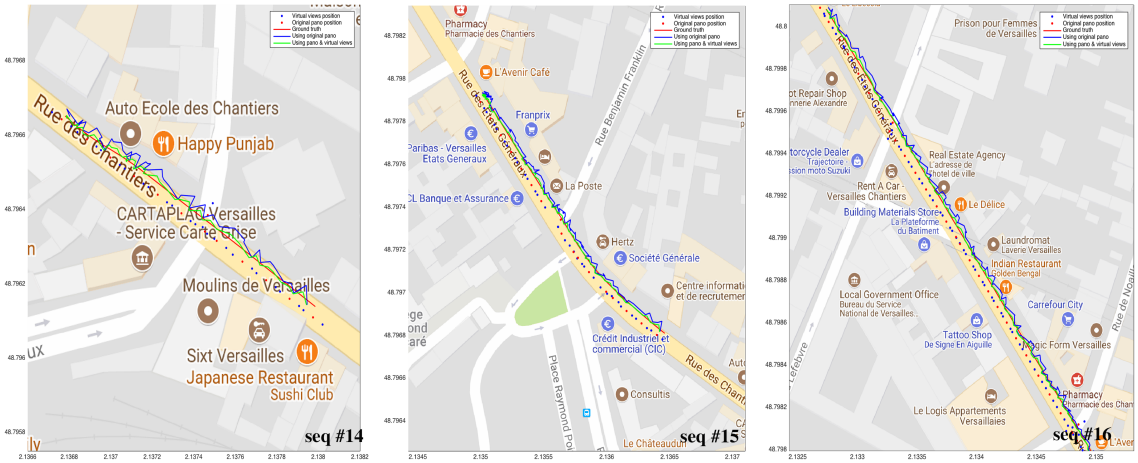


Figure 7.23: The localization output from sequences No.14, 15 and 16 using original Street Views and synthesizedvirtual views. The positions of original panorama, virtual views and ground truth are noted in red points, blue points and red line respectively. The results from original Street Views and augmented virtual street views are in blue and green lines.

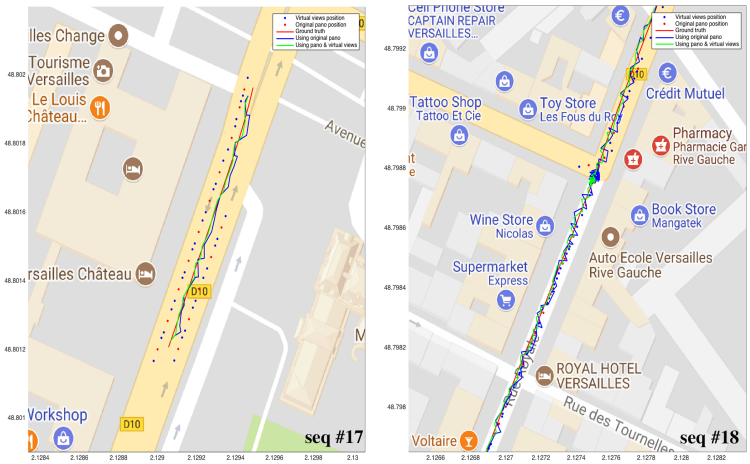


Figure 7.24: The localization output from sequences No.17 and 18 using original Street Views and synthesized virtual views. The positions of original panorama, virtual views and ground truth are noted in red points, blue points and red line respectively. The results from original Street Views and augmented virtual street views are in blue and green lines.

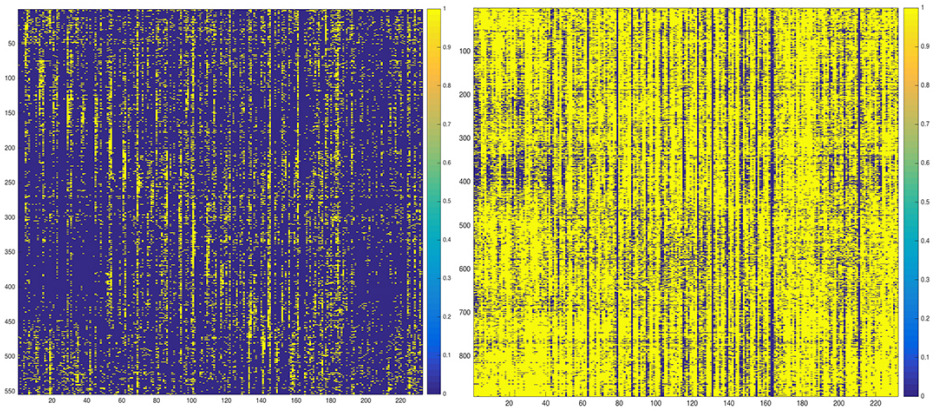


Figure 7.25: Result of the combined BoW working on the sequence No.1 (left) from Zoé and the sequence No.3 (right) from Scénic. Similar to the state-of-the-art evaluation, we find the maximum of the cosine similarity and visualize all values above $0.8 * maximum$.

At the same time we also find there are certain sequences in the test area that are not well localized by our algorithms. The main reason is that we cannot get a proper result in the coarse place recognition process, see Figure 7.25. After the analysis, there are two aspects causing this failure:

- In the online dataset of Zoé, the front camera captures many similar scene at different scales. These frames are separated by 40m but are matched to the same Street View with the combined BoW. The frames from Zoé lack distinctive features. The setting with the camera facing forward is not as favorable as the one of the Scenic with a camera facing the building facades.
- The second cause comes from the disturbance of long distance vegetation. The vegetation is a very negative element for visual localization systems. It covers most of the useful features on the urban facades, and also makes the scenes look similar. In addition, it can also lead to a degradation of the RTK GPS and as such generate a big offset for the ground truth. In the sequences No.3, 4, 7, 8 and 9, most of urban features are covered by the trees, which leads to difficulties to recognize places.

In general, our algorithms are capable of realizing a submeter level localization in most urban environments but they are limited to areas with strong urban appearances. Otherwise, the positioning error increases a lot when these distinctive urban features reduce.

7.5 Conclusion

In this chapter, we have presented two monocular urban localization systems that use a GIS. We pushed the conventional appearance-based localization forward to the metric pose estimation by a graph optimization process. Firstly, we evaluated current state-of-the-art algorithms, such as DBoW and FABMAP, and proposed a more robust place recognition algorithm to our case, called combined BoW. In the online stage, the kVLD descriptor was used to establish 2D-3D correspondences.

Then the pose was estimated by solving the PnP problem and the result was optimized by a standard LBA pipeline. This approach requires neither the construction of a consistent map nor the prior visit of the environment. A simple camera set-up makes this algorithm affordable and easy to be deployed in real urban localization. In order to realize a reliable localization, we thoroughly explore multiple information in the GIS and strictly respect their intra relationships. Our technique demonstrates both a high accuracy *w.r.t* the Street View and robustness in complicated urban environments.

Although a considerable proportion of the former localization system achieves a 2m accuracy, the discontinuity still affect the robustness of the system. Thus, we extended this method by using Google Street View imagery to augment the GIS with synthesized views. Instead of densely sampling the images, we take advantage of the topological information to render useful virtual views in a very sparse way, which enables to lighten the optimization burden and to use simpler descriptors to extract the constraints. These augmented virtual views also allow the system to be more robust to illumination, occlusion and viewpoint changes. We experimentally showed that an augmented Street View based monocular localization system works more accurately, smoothly and compactly than using the original database. In our handcrafted-feature-based localization system, the only input of this approach is an image sequence from a monocular camera and Google Street View. In future works, we can also improve our system by tracking frame to frame correspondences or adding the odometry.

Part IV

Localization using Convolutional Neural Network

Chapter 8

Convolutional Neural Networks

8.1 Introduction

In the previous part, we aimed to develop a coarse to fine metric localization system using handcrafted features. The real-time performance is limited due to the image querying process within thousands of Street Views and the bundle optimization. Since we deal with global scale GIS, it motivates us to leverage deep learning methods to develop an online end-to-end localization system. Deep learning has emerged as a central tool to solve perception problems in recent years [114]. It takes advantage of massively collected data to teach a computer how to execute things only human beings were capable of. The convolutional neural network (convnet), as an active branch of deep learning, plays a dominant role in the current computer vision community [114].

Before diving into the convnet based localization systems in the next chapter, we use this chapter to lay down the foundations of convnets[1]. As we know, the classification problem is a central and basic task in machine learning. We introduce a simple multinomial logistic classifier (also known as "softmax" regressor) and transform it into a neural network step by step. In this phase, some core ideas in deep learning like stochastic gradient descent, the back propagation, overfitting avoidance techniques and activation functions are discussed in details. Then the main building blocks of convnets, namely convolution and pooling layers are reviewed, as well as recently used convnets models and training libraries. We also talk about transfer learning and its applications in the end.

8.2 Deep Learning

8.2.1 From Machine Learning to Deep Learning

In the machine learning field, the classification problem is the foundation for other algorithms, such as regression, reinforcement learning, ranking, etc. Therefore, we take the linear classification as an example and transfer it to a neural network. A

[1]The localization work with convnet was conducted in the very end of the PhD study, therefore instead of providing a comprehensive state-of-the-art, we propose here only a short introduction to principles of deep-learning with Convolutional Neural Network. Readers knowing well deep learning can overlook this chapter.

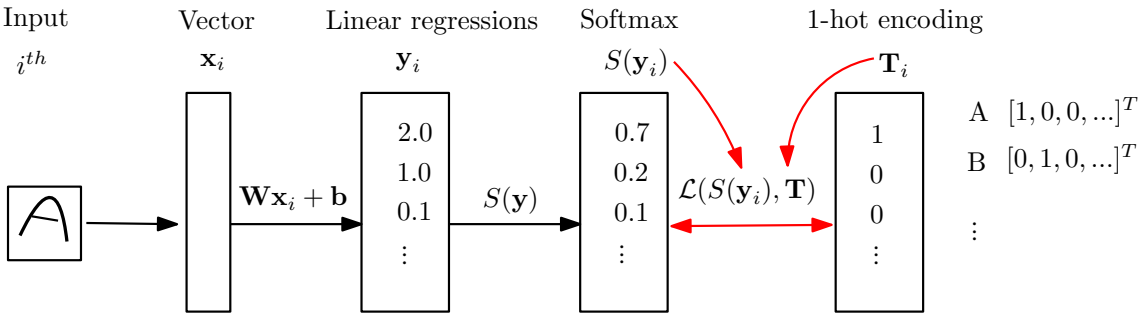


Figure 8.1: A linear classifier to recognize handwriting letters. Image credit of the Udacity Deep Learning online course.

linear classifier makes the classification decision based on the values $\mathbf{Y}$ calculated by a linear combination function f of the input vector $\mathbf{X}$, as stated in Equation 8.1, where $\mathbf{W}$ is the weight matrix and $\mathbf{b}$ is the bias matrix. The *linear function* is actually a giant matrix multiplier. It takes all the inputs as a big vector denoted as $\mathbf{X}$ and multiplies them with a weight matrix to generate its predictions $\mathbf{Y}$.

$$\mathbf{Y} = f(\mathbf{X}, \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{X} + \mathbf{b} \tag{8.1}$$

Assume we have a concrete task to recognize and classify handwritten letter images (*e.g.* MNIST dataset [114]) to their correct letter classes. That is, we need to recognize N examples of the inputs $\mathbf{X}$ (each with a dimensionality D) to K distinct classes. For example, in MNIST dataset, we need to classify $N = 30000$ written letters (each with $D = 20 \times 20 \times 1$ pixels) into $K = 26$ classes (*e.g.* A,B,C,D ...). As shown in Figure 8.1, a handwritten image should be classified as the letter "A". This example has all its pixels flattened out as a single column vector $\mathbf{x}_i$ with $\mathbf{x}_i \in \mathbf{X}, i \in [1, \cdots, N]$. The vector $\mathbf{x}_i$ thus has a shape of $[D \times 1]$. According to Equation 8.1, we apply a linear function $\mathbf{W}\mathbf{x}_i + \mathbf{b}$ to compute the linear regression of input, denoted as $\mathbf{y}_i$. We can find that the weight matrix $\mathbf{W}$ has the size $[K \times D]$ and the bias vector $\mathbf{b}$ has a shape of $[K \times 1]$. The bias vector influences the output $\mathbf{Y}$ but without interacting with the input $\mathbf{x}_i$. With a single matrix multiplication and addition, we realize a linear mapping. The function f maps the input to class scores: it evaluates K separate classifiers in parallel, where each classifier is a row of $\mathbf{W}$. As such, each row of $\mathbf{y}_i$ is a predicted value with respect to the corresponding class.

Training the model consists in tuning the parameters $\mathbf{W}$ and $\mathbf{b}$ so as to obtain, for any input image, an output close enough to the ground truth labels $\mathbf{T}$ (with a shape of $[K \times N]$). In our example, the output of the linear classifier is $\mathbf{y}_i = [2.0, 1.0, 0.1, \cdots]$. For the ground truth $\mathbf{T}(\mathbf{y}_i)$ (with a shape of $[26 \times 1]$), we can set the class score of the letter "A" by a vector $[1.0, 0, 0, \cdots]$, the letter "B" as $[0, 1.0, 0, \cdots]$, until the letter "Z". This method is called *one-hot encoding*. Each label is represented by a vector of dimension equal to the number of classes and in the vector 1 indicates the correct class and 0 for everywhere else.

We find the output of the linear classifier is different from the ground truth. An easy transformation is to provide probability of 1.0 for the maximum value in the vector of $\mathbf{y}_i$ and the rest considered as zero. Thus, a softmax function $S(\mathbf{y}_i)$ (*cf.* Equation 8.2) is applied to $\mathbf{y}_i$. This function allows us to estimate a "probability" of all classes: the prediction probability for the correct class is close to 1 and others

to 0, such as $\mathbf{S}([2.0, 1.0, 0.1, \cdots]) = [0.7, 0.2, 0.1, \cdots]$.

$$S(\mathbf{y}_i) = \frac{[e^{\mathbf{y}_1}, e^{\mathbf{y}_2}, \cdots, e^{\mathbf{y}_K}]^T}{\sum_{k=1}^{K} e^{\mathbf{y}_k}} \tag{8.2}$$

After obtaining $S(\mathbf{y}_i)$, we need a *loss function* to quantify the agreement between the softmax output and the ground truth labels. An optimization is then used to minimize the loss function with respect to the parameters in the linear function, namely $\mathbf{W}$ and $\mathbf{b}$. This optimization process is trained with plenty of training data. The whole process is known as "softmax regression", which is also a simple neural network. Once optimal $\mathbf{W}$ and $\mathbf{b}$ obtained, we can use this classifier to estimate other data (test data).

Then we need to design the loss function $\mathcal{L}$ between the prediction $S(\mathbf{y}_i)$ and its ground truth label $\mathbf{T}_i$. For our example, a natural way to measure the distance between two probability vectors is called the *cross entropy* (*cf.* Equation 8.3). The cross entropy is a sum over all classes, namely through M dimensions. It renders high distance for the incorrect class but low distance for the correct class. In our example, $\mathcal{L}(S(\mathbf{y}), \mathbf{T}) = -(1 * ln(0.7) + 0 * ln(0.2) + 0 * ln(0.1)) = 0.35$.

$$\mathcal{L}(S, \mathbf{T}) = -\sum_{m \in [1, \cdots, M]} \mathbf{T}_m \log(S_m) \tag{8.3}$$

We train the model with all well-labeled training data $\mathbf{X}$ to find the values for the weights and bias which are good at performing predictions. Using this distance representation, the cost function can be defined by the average cross entropy of the entire training set as illustrated in Equation 8.4.

$$\mathcal{C} = \frac{1}{N} \sum_{i \in [1, \cdots, N]} \mathcal{L}(S(\mathbf{W}\mathbf{x}_i + \mathbf{b}), \mathbf{T}_i) \tag{8.4}$$

The cost to minimize is a function of the weights and the biases. In machine learning, this classical numerical optimization problem can be solved by the gradient descent. That is, we compute the gradient (∇) of the cost with respect to the parameters $\mathbf{W}$ and $\mathbf{b}$, denoted as: $\nabla\mathcal{C}(\mathbf{W}, \mathbf{b})$. Then we follow that derivative by taking a proportional step backwards $-\lambda\nabla\mathcal{C}(\mathbf{W}, \mathbf{b})$ and repeat until some stopping criteria are reached (*e.g.* cost below a predefined threshold, or cost not decreasing anymore, etc.). Here λ is a proportional value known as the learning rate. The parameters are updated by Equation 8.5.

$$\begin{aligned} \mathbf{W}_{t+1} &= \mathbf{W}_t - \lambda \frac{\partial \mathcal{C}}{\partial \mathbf{W}} \\ \mathbf{b}_{t+1} &= \mathbf{b}_t - \lambda \frac{\partial \mathcal{C}}{\partial \mathbf{b}} \end{aligned} \tag{8.5}$$

For image data, the pixel values often lie in the $[0, 255]$ range. Feeding these values directly into a learning process may lead to overflows. In the machine learning literature, we have four common forms of data preprocessing [114]: *mean subtraction* refers to subtract the mean across every individual feature in the data and has a geometric interpretation of centering the data around the origin along all dimensions. *Normalization* is to normalize the data dimensions to have a approximately same scale, for example once the data is zero centered, we can divide each dimension

by the standard deviation *PCA (Principle Component Analysis)* can transform the data into a compressed space with less dimensions. The *whitening* takes the data in the eigenbasis and divides every dimension by the eigenvalue to normalize the scale. In practice we often preprocess the input data to have a 0 mean (zero-centered) and equal variances [188].

Moreover, the weights and biases should be initialized at a good starting point for the gradient descent to proceed. There are various initialization techniques and a simple general method is to use small random numbers drawn from a gaussian with 0 mean and 10^{-2} standard deviation [114]. To avoid that the classifier memorizes the training set and fails to generalize the model, we also divide the training dataset into three parts: training (to estimate weights and biases), validation (to fine-tune hyper parameters[2]) and test dataset (to evaluate the final model) as stated in machine learning theory.

When dealing with a large scale of training data, instead of computing the update over the whole training dataset, we often randomly select a small fraction of the training data and use the sample derivative to do gradient descend. This sample gradient is not the right direction to do gradient descent and at times the cost might increase. But if we compensate this by many iterative updates, the training would finally converges to a minimum. This technique is called the stochastic gradient descent (SGD) [114]. The SGD is vastly efficient by computing small samples and also scalable with the data and model.

The technique of SGD is the core of the logistic regression, even in deep learning algorithms. There are many variants originated from SGD, such as Adagrad [53], AdaDelta [226], RMSProp [197], Adam (adaptive moment estimation) [101], etc. We will not dive into these different optimization techniques and it is sufficient to know they mainly improve the SGD update rules from two aspects: one is to take advantage of the momentum (the knowledge accumulated from previous update steps); the other is to use learning rate decay (make learning slower). According to the unit tests for stochastic optimization carried out by [173], Adam can serve as a default SGD scheme in most of neural network training. Adam is computationally efficient and is well adaptive for problems that are large in terms of data and/or parameters.

The softmax regression is a nonlinear model with simple matrix multiplications. These linear operations are very stable and the derivative of a linear model is always constant. However, most problems in reality can only be solved by training more complex models. Therefore, we would like to take advantages of simple linear operations but also require the entire model to be non-linear. Take the previous classifier as an example: instead of having a single matrix multiplication, we can separate the matrix multiplications into two parts (layers) due to the linearity (see in Figure 8.2). Between two parts, we insert some ReLUs (Rectified Linear Units whose equation is denoted in Table 8.1) [105] to add non-linearity. From its equation, ReLU is simply thresholded at zero but it is nonlinear in nature. In fact, any function can be approximated with combinations of ReLUs [114]. After inserting ReLUs, the output would be a simple linear combination of weights and inputs.

The model then becomes a two-layer neural network, where the first layer consists

[2]Hyper parameters refer to all parameters of the learning model (*e.g.* the number of convolution filters in a given convnet layer) or of the training algorithm (*e.g.* the learning rate and the number of gradient iterations).

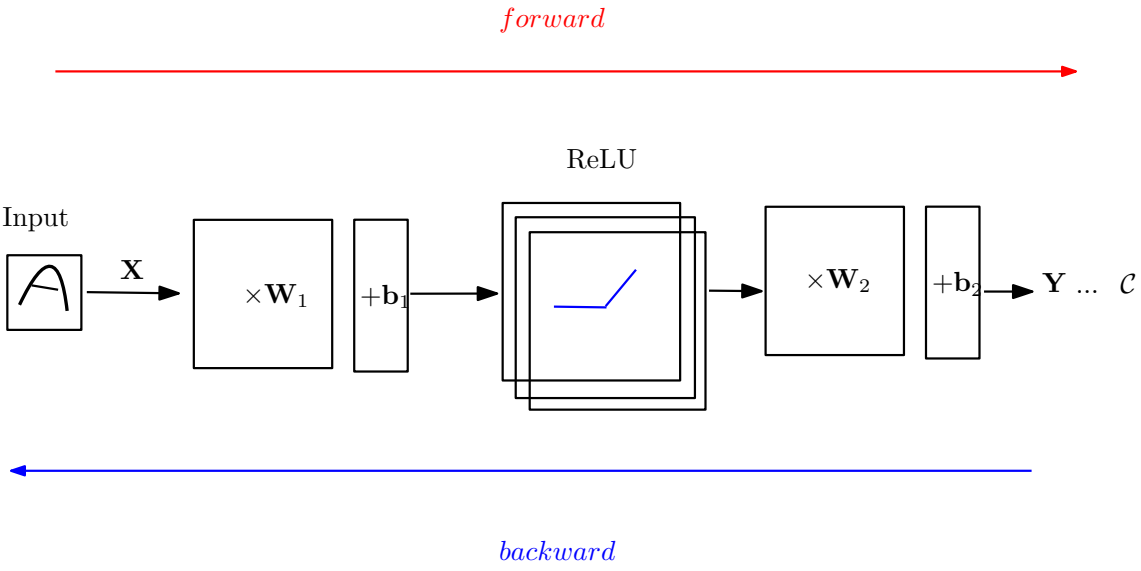


Figure 8.2: A two-layer neural network from the softmax linear classifier of Figure 8.1. The forward process is the linear or non-linear computation among neurons while the backward pass indicates the parameter update (weights and bias) based on the gradient descent.

of the weight multipliers applied to inputs and the *hidden layers*[3] of ReLUs, and the second layer is the multipliers to the intermediate outputs, then followed by the softmax function. We can change this network to be more complex by increasing the size of the activation layers (much wider), or adding more layers to make the model deeper. Usually, training a neural network in a deeper way can be more tricky (due to many non-linearities), but is able to capture a hierarchical structure of the training data [214].

The deep learning network can be very large and it is not possible to write the gradient formula by hand for all parameters. In this way, *back propogation* is introduced to compute the gradients of every inputs and intermediates by recursively applying the chain rule along a computational graph [112]. Similar to the neural network, we can use a graph structure to implement *forward* and *backward* to perform updates. In the forward stage, we compute the result of matrix multiplications and nonlinear activations from layer to layer; in the backward stage, we employ back propagation[4] to compute the gradients of the loss function with respect to the inputs. In deep learning process, there are some conventional terms: one *epoch* is one forward and one backward pass of all the training dataset; the *batch size* is the training number in the small samples of SGD during an epoch; and the number of iterations is the number of the passes.

As we know, the neural network was originally inspired by the biological neural model. Even though the current deep learning goes further than the biological interpretations, we still give a brief description from this point of view. One input from the $\mathbf{X}$ is a basic computational unit called as a *neuron*, and the weights $\mathbf{W}$ and the biases $\mathbf{b}$ serve as the *synapses* around this neuron. Once the sum of the

[3]A hidden layer refers to a layer connected between input and output neurons.

[4]The backward pass begins with the loss and computes the gradients with respect to the output. The gradient with respect to the rest of the model is computed layer-by-layer through the chain rule.

Function	Expression	Pros	Cons	Variants
Sigmoid	$S(x) = 1/(1+e^{-x})$	Active interperation with a range $[0, 1]$	Staturation at tails causes zero gradients Non-zero centered output Computational cost	Softmax
Tanh	$tanh(x) = 2S(2x) - 1$	Zero centered output with the range $[-1, 1]$	Staturation at tails causes zero gradients Computational cost	
ReLU	$relu(x) = max(0, x)$	No saturation when x is positive Computational efficiency 6× fast converging	Non-zero centered ouput Zero gradients when x is negative	Leaky ReLU PReLU ELU
Maxout	$max(w_1^T x + b_1, max(w_2^T x + b_2)$	No saturation and zero gradients	Double paramters	

Table 8.1: A comparison of commonly used activation functions.

synaptic strength arrives a certain threshold, the neuron will fire a spike (output) along its corresponding axon. We model the firing rate of the neuron as an *activation function*. Table 8.1 depicts the commonly used activation functions in the state of the art.

The most common challenge to training a neural network is overfitting [114], where the error on the training data is small but the error on the test data is large. The model has not learned to generalize to new situations. Here we introduce some frequently adopted techniques to prevent overfitting:

- Terminate training early as long as the performance on the validation set stops improving. After recording the model training curves, the gap between the training and validation accuracy indicates the rate of overfitting;
- Add more training dataset or use data augmentation[5][209];
- Reduce the network architecture complexity and use architectures that generalize well;
- Add some penalty terms (artificial constraints) on large weights in the cost function, so as to bound the model complexity, in order to achieve a better generalization (L_1 or L_2 regularization, for instance);
- Use *dropout layer* [185] that deletes a random sample of the activations (makes them zero) in training. For example, we can add a drop out layer with a 50% probability before the hidden layer ReLU in Fig 8.2. It thereby randomly makes 50% of the outputs calculated from the first layer be zero. Dropout forces the networks to learn much more times than before. The final trained parameters would be a convinced consensus from these redundant learnings.

Neural network training is a process full of numerous trials and errors. The design of the network is an open choice, ranging from layer width to model depth and varying from different layer and activation function selection. Moreover, we have a lot of hyper parameters to test, including initializations, learning rate decay, momentum, batch size, etc. However, a popular strategy to train neural networks in the literature is to learning the model in a slower (*i.e.* small learning rate) and deeper (*i.e.* a deep architecture) way.

[5]Data augmentation is mainly used in image datasets. We can generate artificial datasets by simple techniques, such as cropping, rotating, and flipping input images.

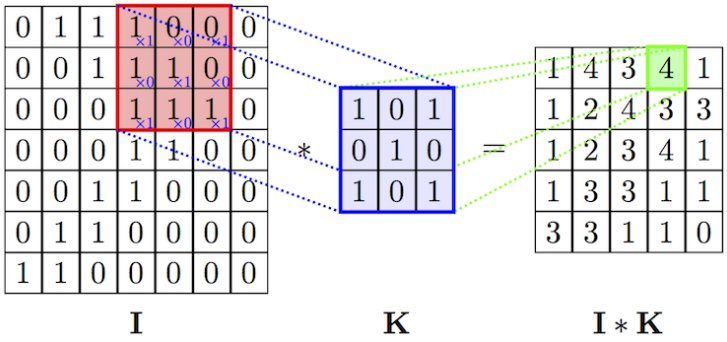


Figure 8.3: Illustration of the convolutional operation: a $[3 \times 3 \times 1]$ filter $\mathbf{K}$ slides on a $[7 \times 7 \times 1]$ image $\mathbf{I}$ and do dot products. The obtained image after the convolution (also called a *feature map*) has a reduced size $[5 \times 5 \times 1]$. How to compute the changed size after the convolutions has been well defined according to [54].

8.2.2 Convolutional Neural Networks

Deep learning is a data driven method. The boom of massive data in recent year and of powerful parallel data processing of the GPU make deep learning possible. Understanding the data structure, such as images or voice sequences, helps a lot to improve the training. Convnets are very similar to ordinary Neural Networks stated in previous sections: they are made up neurons that have learnable weights and biases. One important difference is that convnets only deal with grid-organized inputs such as images (or temporal sequences), which allows us to encodes the structural properties of input data into the architecture. For using ordinary machine-learning on images, input is either a vector of handcrafted features extracted from the image, or the pixel values flattened into a 1D vector (like the previous linear classifier). As shown in previous one-layer softmax regression classifier (applied on flattened 20×20 grayscale images with 26 possible classes), there are already $26 \times 20 \times 20 \times 1 = 10400$ weights to learn. This amount still seems manageable, but clearly it cannot scale to larger images. For example, for RGB images with more respectable size, *e.g.* $[256 \times 256 \times 3]$, it will lead to 196608×26 parameters to learn for a one-layer ordinary neural network.

In the convnet, instead of computing global weighted sums of all pixels, each neuron computes a linear filter applied only on a small rectangular patch the image. In order to analyze the whole image, each filter is applied to every possible patch location. Therefore, for a given filter there is one neuron for each location, and all these neurons should have the same weights because they compute the same filter. This technique is called weight sharing, which reduces the learnable parameters and makes networks easier to train. The weights sharing is realized by convolutional operations of image processing, see in Figure 8.3. The small *filter*, (also known as *kernel*) works as a sharing weight, and the dot products serves as the matrix multiplications in linear regression. In this way, training a model is to learn the parameters in the kernels.

In the convnet, all layers contain 3 dimensions as an image: height, width and depth[6]. Suppose we need to recognize handwritten letters from input images with

[6]Here, the depth means image channels and differs from the network depth, *e.g.* a RGB image has a depth of 3.

size of $[256 \times 256 \times 3]$. For the first layer of the convnet, if we introduce, for instance 16 filters with small spatial size ($[3 \times 3 \times 3]$), we obtain $(256 \times 256) \times 16$ neurons, but only $(3 \times 3 \times 3 + 1) \times 16 = 448$ independent parameters to learn, which is way smaller than for a fully-connected layer used in ordinary Neural Networks. Each filter extends the full depth of the inputs and is shared over the image space. Then after the convolutions, a non-linear activation function (see Table 8.1) is usually applied to the output of each neuron. Furthermore, the convolution layer is generally followed by a pooling layer which reduces dimension by computing the max or average over neighboring neurons. This computation is called the *pooling* layer[7]. After the (convolution+pooling) layer, we finally get 16 feature maps stacked together like pancakes along the new depth, in this example, the output turns to be $[128 \times 128 \times 16]$, see Figure 8.4. Then we can connect the obtained feature maps with a fully connected layer[8] with a size of 26 output (one for each class of letter), and therefore $26 \times (128 \times 128 \times 16 \times 1)$ weights to learn. Regarding the loss function design and the weights update, they follow the same pipeline of the traditional Neural Networks.

However, a one layer convnet is very rare. The popular strategy is to stack several successive layers of (convolution+pooling) to extract more semantic CNN features (*i.e.* the visualization of the output feature maps), as depicted in Figure 8.4. We can see the convolutional and pooling operations reduce the spatial dimensionality layer after layer. On the other hand, it is commonplace to increase the number of filters (hence the depth) from one layer to the next[219]. As we stated in Chapter 3, the low CNN features extract basic geometries, like edges or lines, but the high level features are combinations of these geometries. In summary, convnet aims to represent the training data in semantic representations (in depthwise) instead of spatial representations (hight×width). Once deep and narrow-spatial representations are obtained, we can connect it to the fully connected layers and train our classifier or regressor.

A typical architecture for a convnet is a few layers alternating convolutions and max pooling, followed by a few fully-connected layers at the top. The first famous model to apply this architecture was the LeNet-5 (see Figure 8.6) in 1998 [113]. The successful AlexNet [105] is also an extended form of the LeNet-5.

Nowadays, we have a lot of frameworks to train deep learning models, as listed in Table 8.2. They are all multi-GPU supported and adaptive to the forward /backward computational pass graph of Neural Networks. In our experiment, we mainly use TensorFlow as the primary framework and Keras as an interface with high-level model definition. As we know, training a convnet consists of numerous convolution operations that are computationally heavy for a single core CPU. Leveraging multi-core GPU to train is a common way, for example, by using multi-GPU card under the CUDA framework of NVIDIA. Moreover, some specific cloud services such as Amazon WebService (AWS), Azure or Google Cloud also provide efficient platforms to train convnet models.

[7]The *pooling* layer is very important to reduce the spatial size of the feature maps by using the max or mean function to summarize subregions, namely the maximum (Max pooling) or the average value (Mean pooling).

[8]Recall: fully connected (FC) layers connect every neuron in one layer to every neuron in another layer.

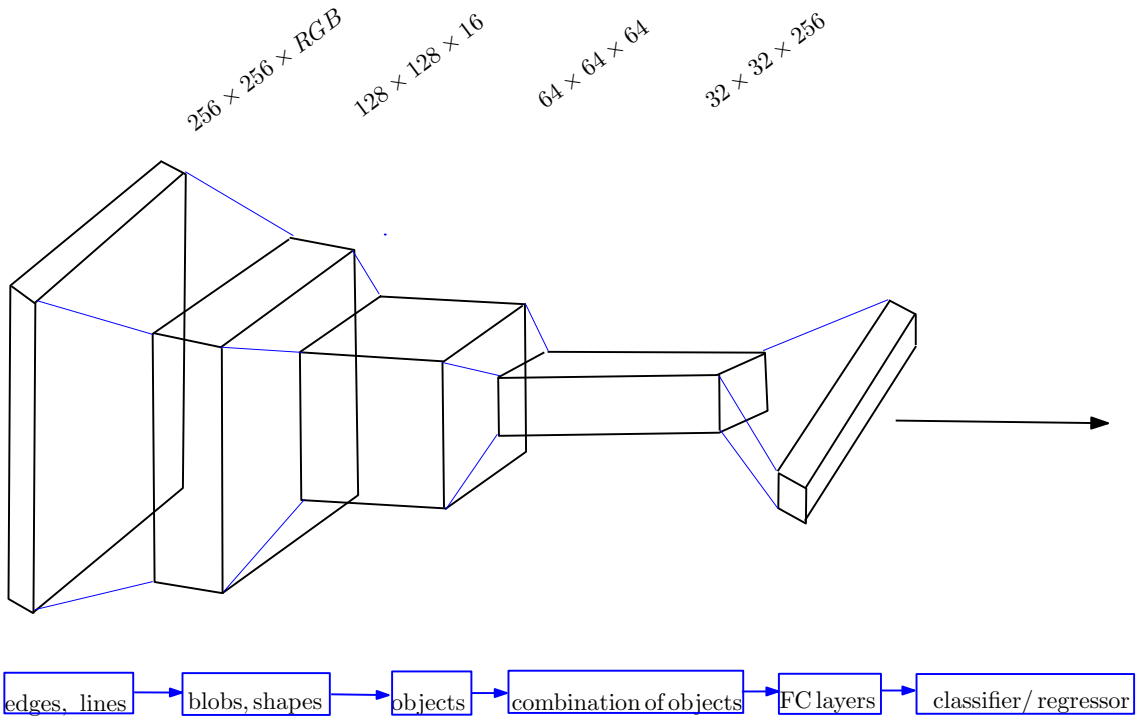


Figure 8.4: Example of a classical convnet architecture with 3 filters and its possible semantic output from every layers.

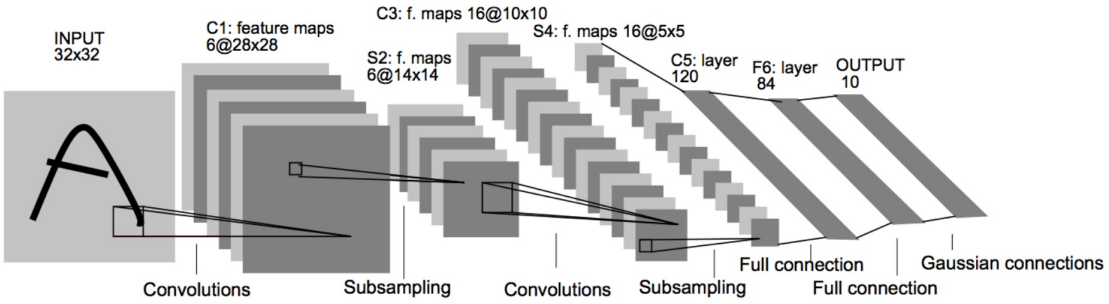


Figure 8.5: Architecture of LeNet-5 as (Conv1-Pool-Conv2-Pool-FC1-FC2), image credit [113].

Framworks	Sponsor	Time	Languages
TensorFlow [1]	Google Brain	2015	C++/Python/Java
Theano [16]	Univ. Montréal	2009	Python
Keras [33]	Google	2015	Python
Torch [41]	Facebook AI, Google DeepMind	2002	Lua/C
Caffe [93]	Berkeley	2013	C++/Python/Matlab

Table 8.2: Some typical frameworks to train deep learning models.

8.3 Transfer Learning

This is actually rare to train an entire convnet with random initializations since it is nearly impossible to have a training dataset of a suitable size. As such, *transfer learning* [18, 29] is proposed to use the gained knowledge in one domain to solve a new problem in another even if the relationships between them are limited. Neural networks often take days or weeks to train, transfer learning can greatly shorten training time.

In practice, we can train a base network on a base dataset by ourselves or take a pretrained convnet directly, and then transfer the learned features to a new target network and train it on a new target dataset. Transfer learning works only when the base features are general enough to both base and target tasks. Modern base convnets are pretrained on large datasets for several weeks, such as ImageNet.

The pretrained or base network usually behaves as a *feature extractor* for the target convnet or as an *initialization* that needs to be fine-tuned. As a feature extractor (namely only extract CNN features), the base network removes the final fully-connected layer to connect the target network but fixes all pretrained weights. When using as an initialization (that is to say, the task is not changed and we benefit from the pretrained weights to speed up a new training.), we only need to fine-tune the pretrained weights during the back propagation. Since the lower layers capture more generic features than the higher layers [219], we can only fine-tune the last layers rather than all the layers of the pretrained convnet.

8.3.1 Application Scenarios

There are four main scenarios when using transfer learning and every scenario has its specific training scheme. The four scenarios depend on the size of the target dataset and the similarity between the target and base dataset. A large dataset is one million images at least while the small can be 2000 images.

- **Target dataset is small and different from the base dataset.**

  When the model is trained with a small dataset, overfitting is the main concern. In this case, the weights of the base convnet remain constant to avoid overfitting. As the base and target datasets are different, they will not share similar high level features (high lever features are more data-specific representations). It is likely to only use the low level features from the base network (*i.e.* feature maps from the start of convnet in Fig 8.6). Therefore, we keep only several layers at the beginning, add a few FC layers with random initializations and train these weights. In such cases, training a simple linear regression computed on the output of the truncated convnet can often be sufficient.

- **Target dataset is small and similar to the base dataset.**

  Rather than retraining the weights of the base network, we keep them unchanged (or *frozen* in Tensorflow) to avoid overfitting as in the first scenario. Since the datasets are similar, both base and target datasets share similar high-level features (*i.e.* feature maps from the end of convnet in Figure 8.6). As such, it is not necessary to change the base network. We just cut off the end of the convnet, add a few FC layers with random initializations and train these weights.

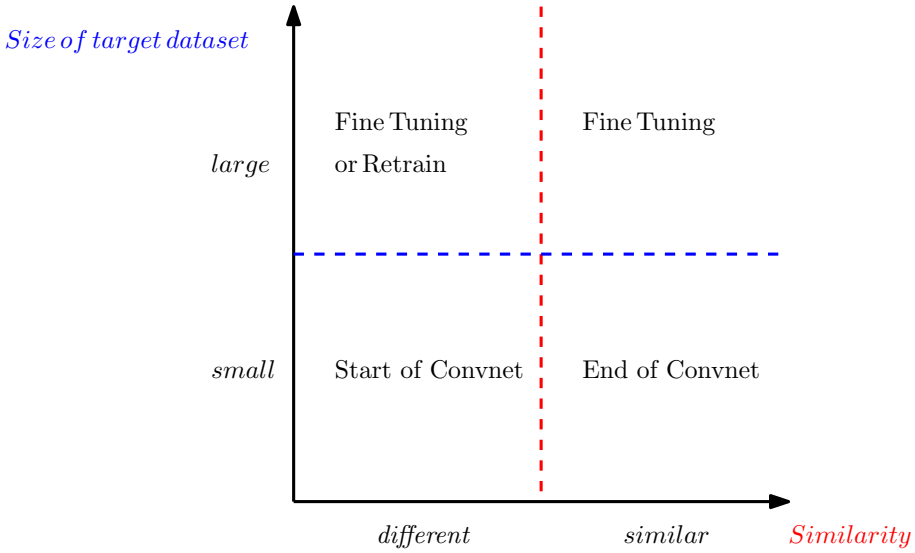


Figure 8.6: The 4 scenarios to use transfer learning.

- **Target dataset is large and different from the base dataset.** When the target dataset is large, overfitting is no longer a dominant issue. Even if two datasets are different, using the pretrained weights to do finetune helps to speed up the training to some degree. Another option is to retrain the whole network if it is possible.

- **Target dataset is large and similar to the base dataset.** When the target dataset is large and also similar to the base dataset, the only work is to replace the last FC layer with an output matching FC layer and fine tune the target network. During the fine tuning, it is important to use a smaller learning rate for the target network.

8.3.2 Pretrained Neural Network

The history of neural network can be traced back to the birth of the perceptron conception in 1957, see Figure 8.7. Then neocognitron in the1980s served as the inspiration for the convnets. After LeNet proposed in 1998, there has not been a big breakthrough until the last years with the availability of massive labeled data and powerful computational processors. The ILSVRC (ImageNet Large Scale Visual Recognition Competition) [105] provided numerous well-labeled training dataset to attract both industry and academia to produce successful networks. As shown in Figure 8.7, deep neural networks can be improved from two aspects: optimizing the layer design and increasing the depth of models. In practical applications, we take into account the accuracy, inference time, power consumption, model complexity, number of operations, etc. We also illustrate some milestone winners of this competition during the last five years in Table 8.3.

AlexNet, as the winner of ILSVRC 2012, was a pioneer in using the ReLU activation functions, dropout layers, maxpooling and image augmentation to avoid the overfitting. It was also first time that massive parallelism of GPUs are used to accelerate convnet training.

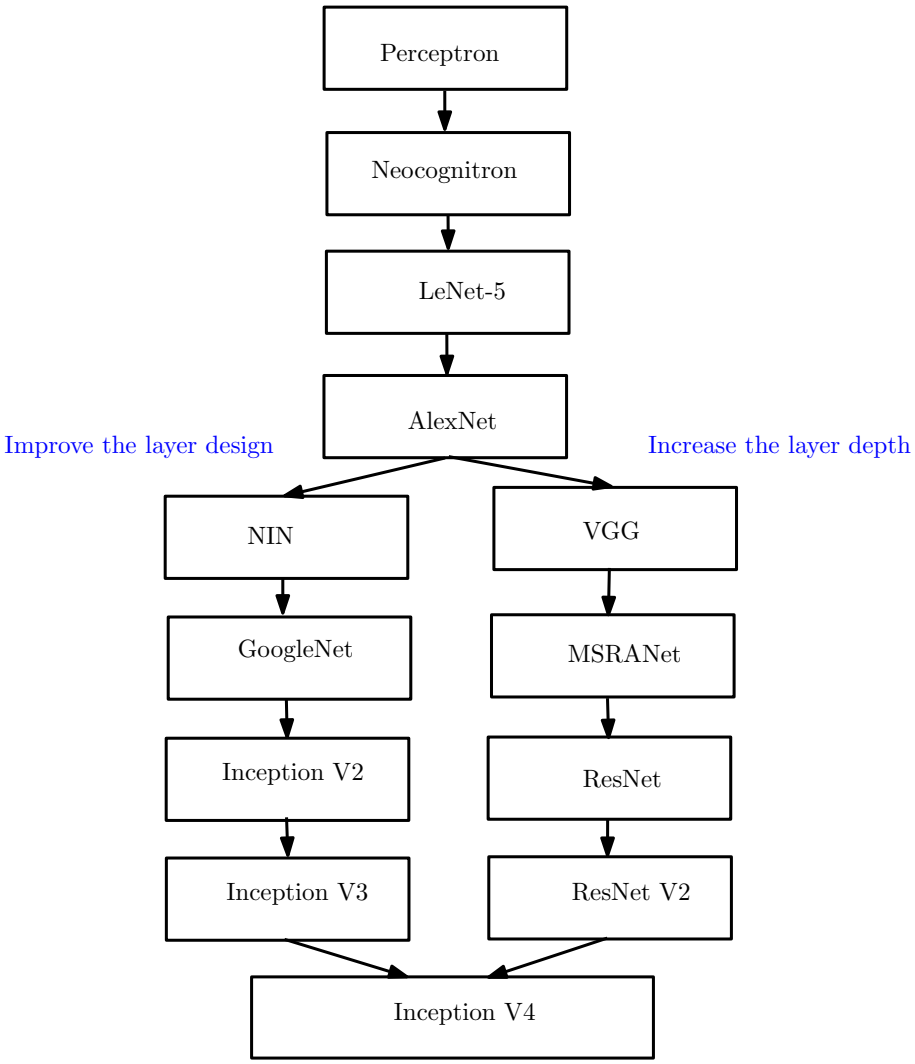


Figure 8.7: History and tendency of the Neural Networks in image recognition field.

Models	Year	Layer	Training Set	Training Parameters	Top-5 error in ILSVRC
AlexNet[105]	2012	5+3	ImageNet	60 million	15%
VGG16/VGG19[178]	2014	13/16+3	ImageNet	138/143 million	7.6/7.3%
GoogleNet[190]	2014	21+1	ImageNet	5 million	6.7%
Inception V2[87]	2015	21+1	ImageNet	5 million	4.8%
Inception V3[191]	2015	21+1	ImageNet	23 million	3.5%
ResNet 152[78]	2015	151+1	ImageNet	78 million	3.6%
Wide ResNet 50[79]	2016	49+1	ImageNet	25 million	5.79%
Inception V4[192]	2016	21+1	ImageNet	55 million	3.08%

Table 8.3: Pre-trained CNN architectures.

VGG, as the second prize of ILSVRC 2014, extended to deeper networks with 16 to 19 layers. The model followed the basic idea of AlexNet but constructs a long sequence of convolutional layers with $[3 \times 3]$ filters and pooling layers. The use of smaller filters can reduce the number of training parameters but increase non-linearities.

GoogleNet (also called Inception V1), as the 1st prize of ILSVRC 2014, was famous for its computational efficiency with 22 layers but only 5 million of parameters. Inspired by NIN (network in network) [122], an inception module was created: rather than making the choice of which size of filter to use in one convolutional layer, the inception module applied parallel filter operations (using filters with sizes of $[1 \times 1]$, $[3 \times 3]$ and $[5 \times 5]$) on the input and concatenate all filter outputs together. There are several versions developed from GoogleNet: Inception V2 took use of batch normalization to reduce internal covariate of output; Inception V3 added factorization into small convolutions on the basis of V2; Inception V4 (also known as Inception ResNet V2) combined the advantages of inception modules with the residual units of ResNet together and achieves the highest accuracy in ILSVRC.

Current work reveals that training a neural network deeper works better than wider. Yet it is not just as simple as to stack more layers. Too deep models will lead to the vanishing gradient problem and the training error degradation [65]. ResNet, winner of ILSVRC 2015, designed residual units to reconstruct the optimization. Instead of fitting the input to a desired mapping, the residual units as in MSRANet [77] make the raw input passing through and overleaping the convolutional layers to obtain two outputs (residuals and identity). The summation of both outputs is our desired mapping but the model only optimizes the residual part. Nowadays, invariants from GoogleNet and ResNet are widely used in the autonomous vehicle area for their high efficiency and accuracy, such as Inception V, wide ResNet and ResNeXt.

8.4 Conclusion

In this chapter, we introduced the main ideas of deep neural networks from a simple softmax regression classifier. Since deep learning is a theory based on practice, we reviewed current developments and some important techniques to avoid overfitting. Then we talked about the convnets and how they handle image datasets. We detailed the architectures of a convnet and the semantic representation of CNN features. Training a convnet from scratch is rare and transfer learning is a primary strategy to design our own network. We discussed the existing pretrained networks as well as their advantages and disadvantages. In the chapter, we only addressed the neural networks in the classification tasks. In following chapter, we will deep into the localization regression with convnets.

Chapter 9

Metric Localization using Deep Learning

Convolutional Neural Network (convnet or CNN) is becoming a powerful and dominant method in computer vision. Recent research shows that the hierarchy of CNN features can be successfully applied in scene classification, semantic representation and visual localization. In the work of Overfeat [175], CNN features trained on ImageNet work better than both global and local visual descriptors. CNN includes a hierarchy of features with some that are low-level enough to represent a lot of simple concepts and other features (those in final layer) that are high-level enough to combine these concepts to be recognized by a classifier [176]. Moreover, pretrained CNN image classifiers can be used as a basis for building and training other CNNs that can successfully perform other tasks, using the transfer learning approach (See Chapter 8). In particular, the classifier can be replaced by a regressor to realize place recognition (*e.g.* VLADNet [9] and PlaceNet [212]) or pose estimation (*e.g.* PoseNet [100] and VidLoc [40]). Rather than establishing reliable frame-to-frame feature correspondences, place labels (pose estimates) can be classified (regressed) from the CNN features.

Since we deal with a global size dataset, we look forward to a scale-adaptive model to localize in real time even if the training itself is very computer-intensive and must therefore be conducted offline. In this chapter, we present a metric and real-time global localization system by training a convnet with Street Views. Our pose regressor convnet is then tested on the online dataset, and continuous poses are regressed in an end-to-end fashion. Labeled training images are also complemented by virtually synthesizing additional images from few Street Views. Starting from a small trajectory in the test area, we design several convnets to evaluate their performances with respect to the quantity/quality of augmented data and the spatial distribution of Street Views. These convnets are built via transfer learning on base networks pretrained on ImageNet image classification dataset like GoogleNet and ResNet.

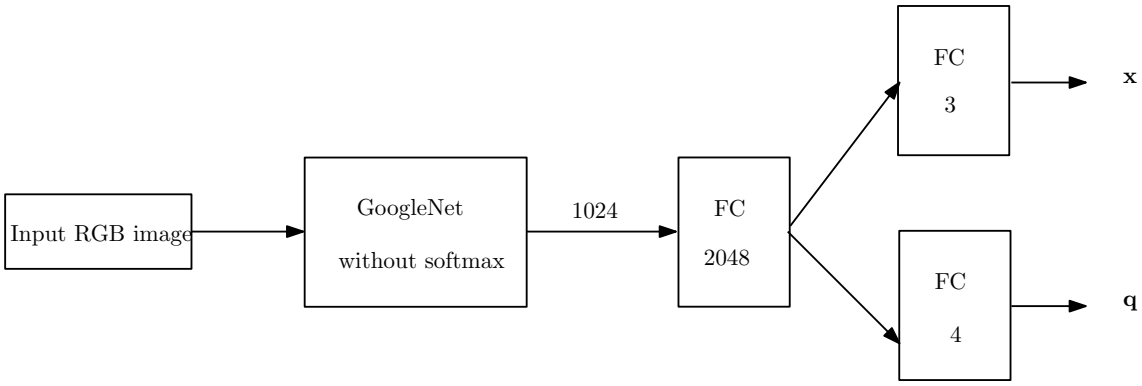


Figure 9.1: A simplified architecture of PoseNet.

9.1 Evaluation of Existing Convnet Localization Model

9.1.1 PoseNet Principle

From metric localization, PoseNet is the first convnet that regresses from a monocular image directly to its 6 DoF pose for both indoor and outdoor scenes. Another impressive work is the VidLoc convnet, which benefits from the temporal continuity of an image sequence by adding LSTM (Long Short-Term Memory) modes to PoseNet. The GIS dataset lacks the temporal constraints in large scale, as such, the RNN (Recurrent Neural Network) based localization models are not involved in our method.

In PoseNet, the training datasets are generated and labeled by a structure from motion reconstruction. Therefore, the test area is covered by many densely distributed images rather than few Street Views. In the training dataset, RGB images and their pose labels are collected. The pose label is composed of the precise camera position $\mathbf{x}$ and orientation (normalized unit quaternion) $\mathbf{q}$ in a 7-dimensional vector as:

$$\mathbf{p} = [\mathbf{x}, \mathbf{q}] = [x, y, z, q_0, q_1, q_2, q_4] \tag{9.1}$$

The quaternion of $\mathbf{q}$ is normalized by a legitimate rotation angle θ along a normal vector $\mathbf{v}$ by:

$$\frac{\mathbf{q}}{\|\mathbf{q}\|} = [\cos\frac{\theta}{2}, \mathbf{v}\sin\frac{\theta}{2}] \tag{9.2}$$

As for the architecture, PoseNet is slightly modified from GoogleNet by replacing all three softmax classifiers by affine regressors (see Figure 9.1). GoogleNet is a 22 layer convnet with 6 inception modules and 2 intermediate classifiers that are only used in training and discarded during the testing. PoseNet slightly modified GoogleNet by replacing all softmax classifiers with their own regressors. A FC layer of size 2048 is added before the final regressors. The SDG (Stochastic Gradient Descent) is adopted to train the convnet by a designed Euclidean loss function

defined in Equation 9.3.

$$\mathcal{L} = \|\hat{\mathbf{x}} - \mathbf{x}\|^2 + \beta \left\| \hat{\mathbf{q}} - \frac{\mathbf{q}}{\|\mathbf{q}\|} \right\|^2 \tag{9.3}$$

In the equation, $(\mathbf{x}, \mathbf{q})$ and $(\hat{\mathbf{x}}, \hat{\mathbf{q}})$ are the ground truth label and pose estimation respectively. All pose labels are defined relative to an arbitrary global reference frame. Since units and error levels may be quite different for the position and orientation estimations, a scale parameter β is used to balance the positon and orientation errors. The optimal β can be observed and fine-tuned by grid search. In the implementation, the scale factor β varies according to the spatial extend of the test area. Naturally β is greater for outdoor scenes as position errors tended to be relatively greater.

9.1.2 The Set-up of Evaluation

As we know, data preparation is an essential part in deep learning. The performance of convnets depends on how the training datasets are organized. In order to well prepare the training datasets from Street View, we first analyzed the original training datasets of PoseNet and draw the inspiration from them. Then, we arranged three types of training datasets from Street View: "*Original Street Views*", "*Augmented Street Views*" and "*Very Augmented Street Views*". "Original Street Views" are the datasets that we used in our first handcrafted feature based method where 8 rectified images are backprojected from each panorama (see Chapter 7.2). "Augmented Street Views" contains "Original Street Views" and the synthesized virtual views 4m forward and backward, which is implemented in our extended handcrafted feature based method (see Chapter 7.3). However, PoseNet performs badly with these two datasets due to overfitting. We therefore also construct "*Very Augmented Street Views*" to test its performance.

Training datasets

As stated before, we explored the original training datasets in PoseNet (*i.e.* King College datasets) to understand the underlying structure of training datasets. We found that the King College datasets contain 1223 RGB images with a 1920×1080 resolution in a 140×40 m^2 space, which means that they have 0.22 training image per square meter. They used central cropping to obtain 224×224 pixel image inputs as required by GoogleNet, as illustrated in Figure 9.2.

Moreover, we visualized the King college dataset labels to discover their data space and structure in Figure 9.3. The pose label has 7 dimensions and a PCA (Principal Component Analysis) method is used to capture the data structure and preserve the pairwise distances in each dimension. In the figure, we can see that various trajectories were captured in various orientations and positions. In addition, the distance between two consecutive training images is about 0.4m, which is much smaller than the distance between two adjacent Street Views.

Compared to the King College datasets, our Street View datasets seems sparsely distributed and considerably insufficient. However, we still applied PoseNet on the "Original Street Views" and "Augmented Street Views" of the sequence No.11 (*cf.* Chapter 7) respectively. We would like to figure out how this pose regression approach performs when trained on a limited Street View dataset and whether the

Figure 9.2: Example of training dataset: an original image of King College dataset on the left and a central-cropped input image for PoseNet on the right.

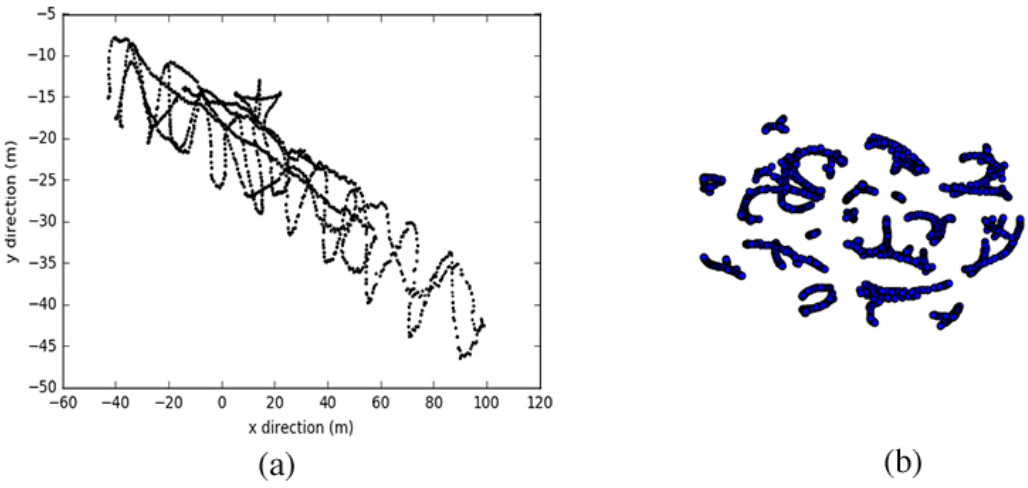


Figure 9.3: Exploration of the King college datasets used in PoseNet: (a) illustrates the spatial distribution of position for 1223 training images; (b) explains the various clusters with regard to orientations using PCA method.

virtual views can contribute to the model training. For the labels in the "Original Street Views" and "Augmented Street Views", we transformed the geometric localizations to UTM form and then normalized into a spatial extend. The orientation of Euler Angles were also converted into a unit quaternion form to meet PoseNet requirements.

PoseNet applied to Original and Augmented Street Views

In the training test, we followed the settings recommended by PoseNet, for example: the convnet was trained by the stochastic gradient descent (*i.e.* Adam mentioned in Chapter 8) with a learning rate of 10^{-5}. The batch size and the number of epochs are fixed to 75 and 500 respectively[1]. All the training images are cropped to 224×224 pixels input as in PoseNet. The scale factor in the loss function and other settings follow the outdoor scene settings of PoseNet. For the pose labels of Street Views, we calculated a mean value of all UTM positions in the training dataset and normalized all the large UTM positions (in both training and test dataset) by subtracting this mean value. This is an effective way to balance the value scale between the translation and orientation. In the test process, the mean value should be added to recover the real predicted value. In PoseNet, GoogleNet is trained by ImageNet and Place datasets, which are RGB image datasets. Our two Street View datasets are RGB images but the online captured dataset from VEDECOM vehicles are grayscale. The idea of transfer learning is to use the well-trained GoogleNet weights as an initialization to speed up the training process. Thereby, we convert the grayscale test images into RGB by the standard OpenCV functions before the prediction process. Actually, we tested several grayscale images on GoogleNet classification task and found there was no degradation for the prediction performance.

We also use 20% of the training datasets for validation purposes so as to observe whether overfitting happens according to the loss function. The training took 8 hours on a AWS EC2 g2.2xlarge instance (with 8 virtual[2] NVIDIA GRID K520 GPUs). The history of the training process is visible on Figure 9.4 (Original Street Views) and Figure 9.5 (Augmented Street Views). As observed from the two figures, the errors in PoseNet models are reduced quickly on both datasets. The validation errors are higher than the training error. In the training on Original Street Views, position is still not converged but it seems there is angle overfitting[3]. In the training on Augmented Street Views, convergence is finished after about 100 epochs where the position and angular error are reduced to about 10m and 0° respectively. Although there is no clear sign of overfitting, the training should be stopped when the validation error is the minimum. This means that the convent can generalize to the unseen data. It appears that with augmented virtual views, the model can converge more quickly. This evaluation help us to make a hypothesis that the Augmented Street Views can improve the model training and reduce the error.

It is clear that there are big gaps between training and validation training curves for both tests. The most common causes of this bad quality include: the low quality

[1]In PoseNet, a good model can be trained after nearly 100 epochs but here we chose 500 for a good observation.

[2]NVIDIA virtual GPU (vGPU) is the industry's most advanced technology for sharing the power of NVIDIA GPUs across virtual machines (VMs) and virtual applications.

[3]The sign of overfitting is here that the training error continue to decrease while the validation error remains constant.

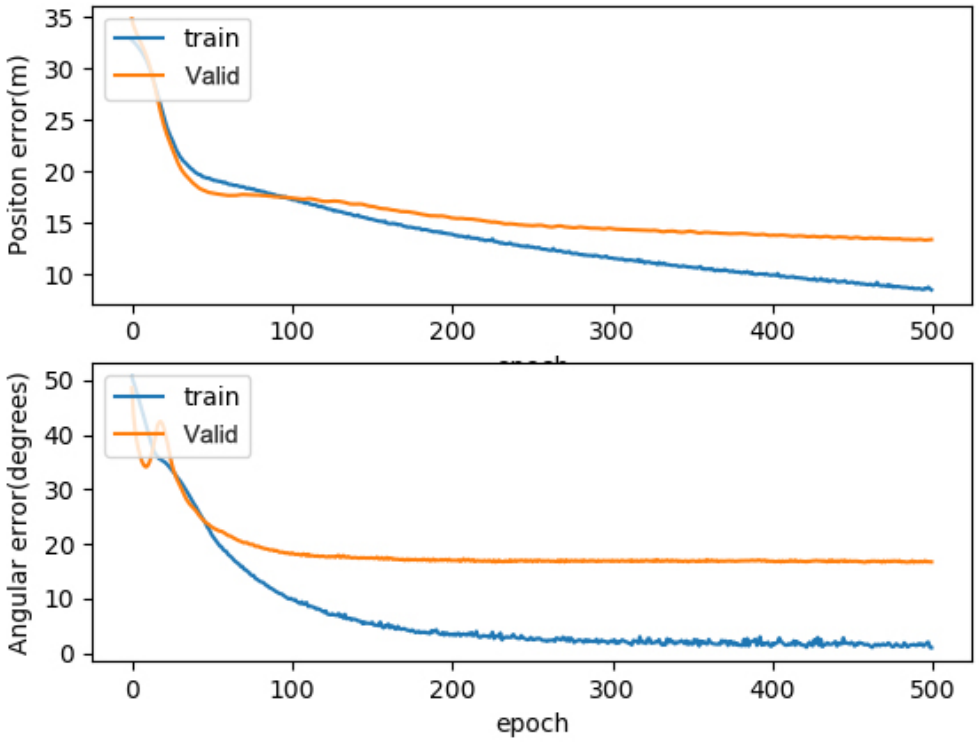


Figure 9.4: PoseNet applied on Original Street Views: the evolution of position and angular average errors on the training and validation datasets with Original Street Views in PoseNet during 500 epochs.

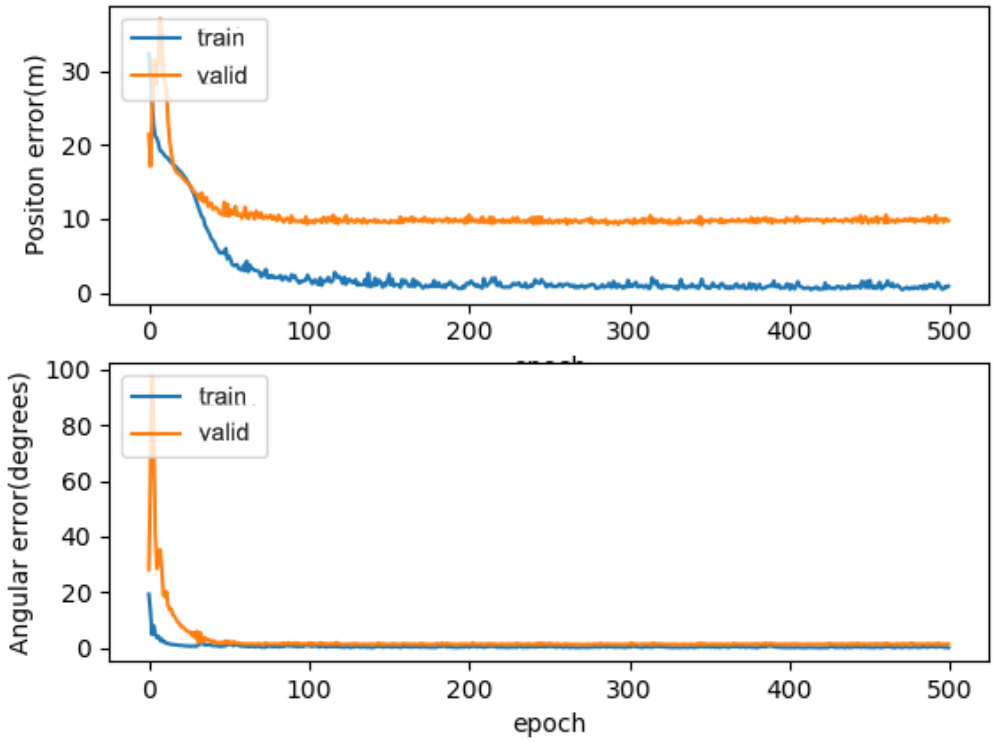


Figure 9.5: PoseNet applied on Augmented Street Views: the evolution of position and angular average errors on the training and validation datasets with Augmented Street Views in PoseNet during 500 epochs.

or the small size of the training data; inadequate model structure and badly setups of hyper parameters. PoseNet has been tested successfully in several urban environments. The hyper parameter β is a balance coefficient to keep the expected value of position and orientation errors to be approximately equal. In our model, we adopted the recommended value according to the spatial extent. Therefore, we believe the main cause of the poor performance on the two datasets is the low quality (a sparse distribution) and the small size of the training data. Intuitively, a training with only 29×8 Street Views in the sequence No.11 is too small to make a regressor work well. Even if transfer learning is implemented in PoseNet, overfitting appears easily when a small training dataset is used. In order to improve the performance, the priority is to augment the training dataset.

Note that we only used a sequence of small spatial extent to train our convnet instead of training a model that works for the whole test area or even for the city. According to PoseNet [100] (see Figure 9.3) and VidLoc [40], the model error increases a lot when the spatial extent is enlarged. They demonstrated that the prediction accuracy is much better in indoor scenes than outdoor.

PoseNet applied to Very Augmented Street Views

As stated before, using panoramic imagery and depth maps of Google Street View, we can place several virtual cameras in the centre of every unit sphere to get rectilinear images. Moreover, we can also render virtual views by moving virtual cameras in other positions. From the results obtained by image based localization methods, we can learn that the vehicle's trajectory is normally around the trajectories of the Street View. It is clear that the camera equipped vehicle usually runs along the street lane without a big lateral bias. Therefore, we only generate various virtual views in forth and back directions along the trajectory of Street Views.

In the data preparation, we should synthesize as many images with high quality as possible. As introduced in the pipeline of the image synthesis, we are able to generate different virtual views, by translating the virtual cameras and changing the local yaw degree (in back projection) of the virtual camera. For example, using the same panorama of Figure 5.11 (Recalled in Figure 9.6) and its corresponding depth map, we test to synthesize virtual views at different locations and with various camera yaw settings, as illustrated in Figure 9.7, 9.8 and 9.9.

As shown in Chapter 5.1, there are no missing pixels when the image is rectified from the panorama without translation. Moreover, more details can be back-projected from panorama when more virtual cameras are created. However, when we set the virtual location far away from the original panorama, skewed objects and spurious artifacts quickly appear in virtual views. Null pixels will be produced if the synthesized views are far away from the panorama. Normally, the farther they are, the more pixels are lost. Consequently, our training convnet model will be significantly influenced because the null pixels can not well initialize weights of filters in the convnet. It even can result in the divergence of the trained model. In order to loosen this influence, two simple but useful approaches are employed:

- We only select useful synthesized images as those having a proportion of null pixels less than half;

Figure 9.6: Recall the Figrure 5.11: a Street View panorama at location $[48.80056, 2.136449]$ is extracted in summer in the test area.

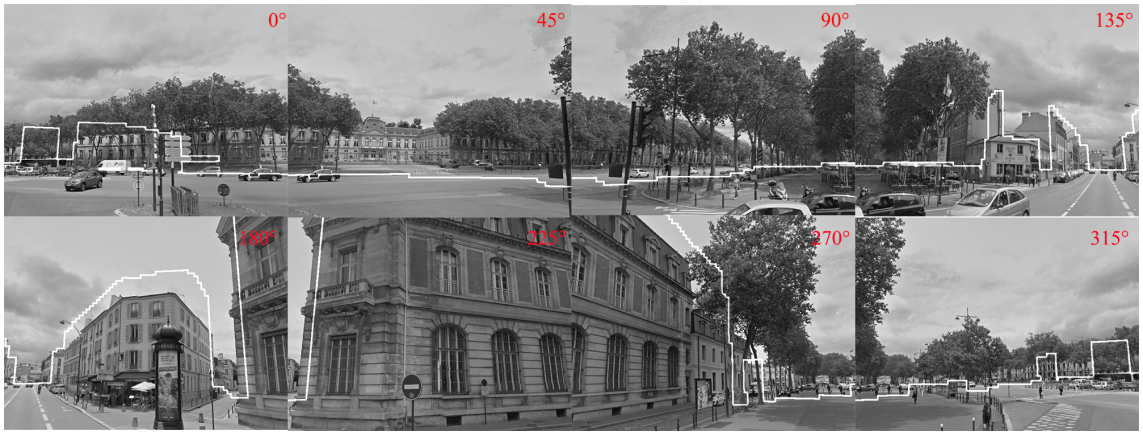


Figure 9.7: Images synthesized from 8 virtual cameras at the same location as the original Street View panorama of Figure 5.11. The white lines reflect the misalignment and poor accuracy of the depth map when projected on perspective images.

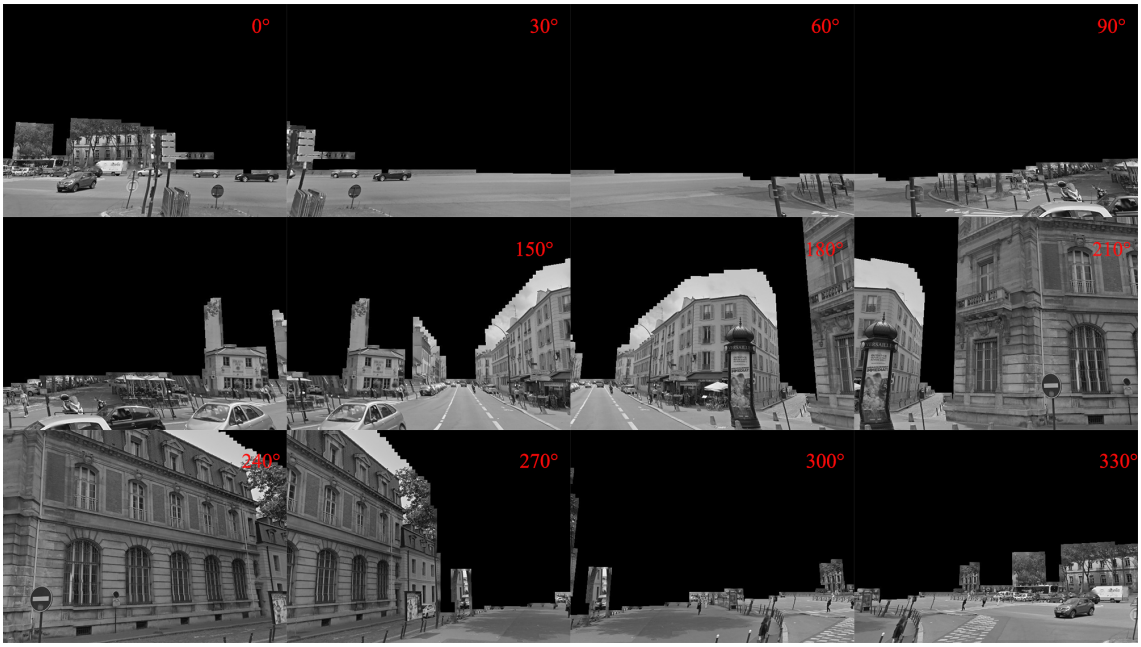


Figure 9.8: Images synthesized from 12 virtual cameras at 1m forward location of Figure 9.6 along the trajectory, namely at [436228.694944, 5405753.37194] UTM coordinates *w.r.t* the original panorama of Figure 5.11.

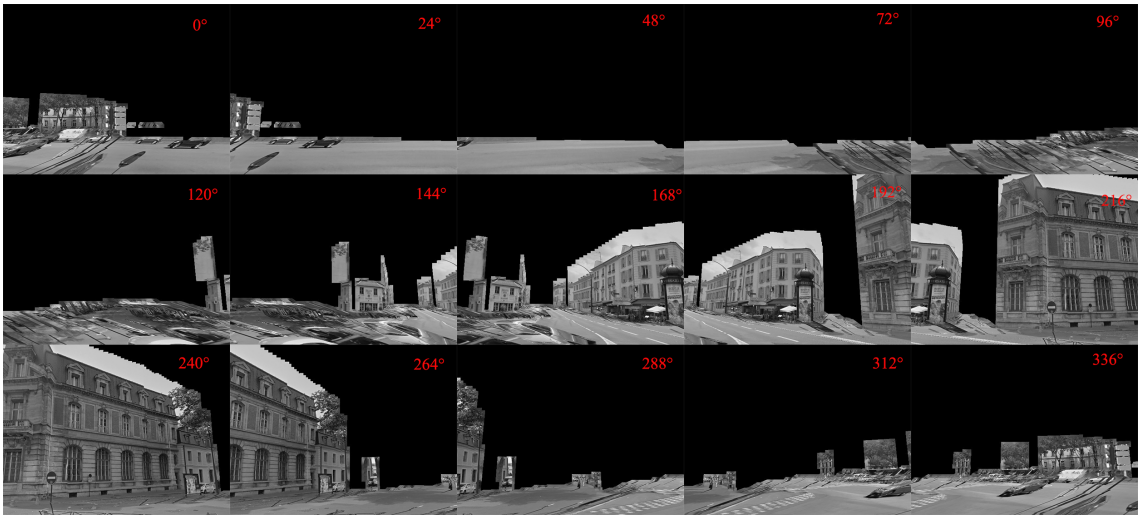


Figure 9.9: Images synthesized from 15 virtual cameras at 4m forward location of Figure 9.6, namely at [436231.656811, 5405745.94043] UTM coordinates *w.r.t* the original panorama of Figure 5.11.

Figure 9.10: For an original Street View image, we resize this image to meet the requirement of PoseNet. It is more suitable to use central-cropping but here we adopted resizing to preserve more scene possible.

- The translation of the virtual camera should be within a maximum 4m[4] forth or back from the original panorama along the trajectories.

In the experiment, we change the virtual camera positions with a 0.2m step within a forward and backward 4m limitation around the input panorama. By setting different yaw angles, we generate $[8, 12, 15, 18, 24, 36]$ virtual cameras respectively in the centre of a same orignal and virtual panorama. We demonstrated that 4m is the maximum distance for an ideal forward and backward synthesis. Then we tested several translation steps within this maximum distance and found out 0.2m is also a good trade-off to render both distinctive and large virtual views.

In order to train a brightness-invariant CNN features, we generate artificial brightness changes as well, see Figure 9.11. Following a classical pipeline in the literature, we convert the color format RGB to HSV and add a small random value to change the V channel value in the training dataset. Moreover, in real urban environment, we also generate random shadows for the training images to simulate the changes of sunlight in 24 hours, see Figure 9.12. It is realized by a random-shape mask whose pixel values equal to 0 but alpha value follows a uniform distribution between 0.4 and 0.6. For each original Street View, we perform 50 random brightness changes and 50 shadow generations, while the pose label remains the same. Please note that the brightness changes are only applied on the original Street Views images as the artificial rendered images have already missing pixels. After this artificial generation, the obtained Very Augmented Street View database is expanded by a nearly 1500 factor.

After this augmentation, we adopted the same training pipelines and setups and Figure 9.13 depicts the history of this training process. As observed, the error gaps between the training and validation datasets decrease considerably when PoseNet was applied to Very Augmented Street Views. In addition, the curves of the training

[4]As we shown in Chapter 7, when the translation is large than 4m, more missing pixels appear.

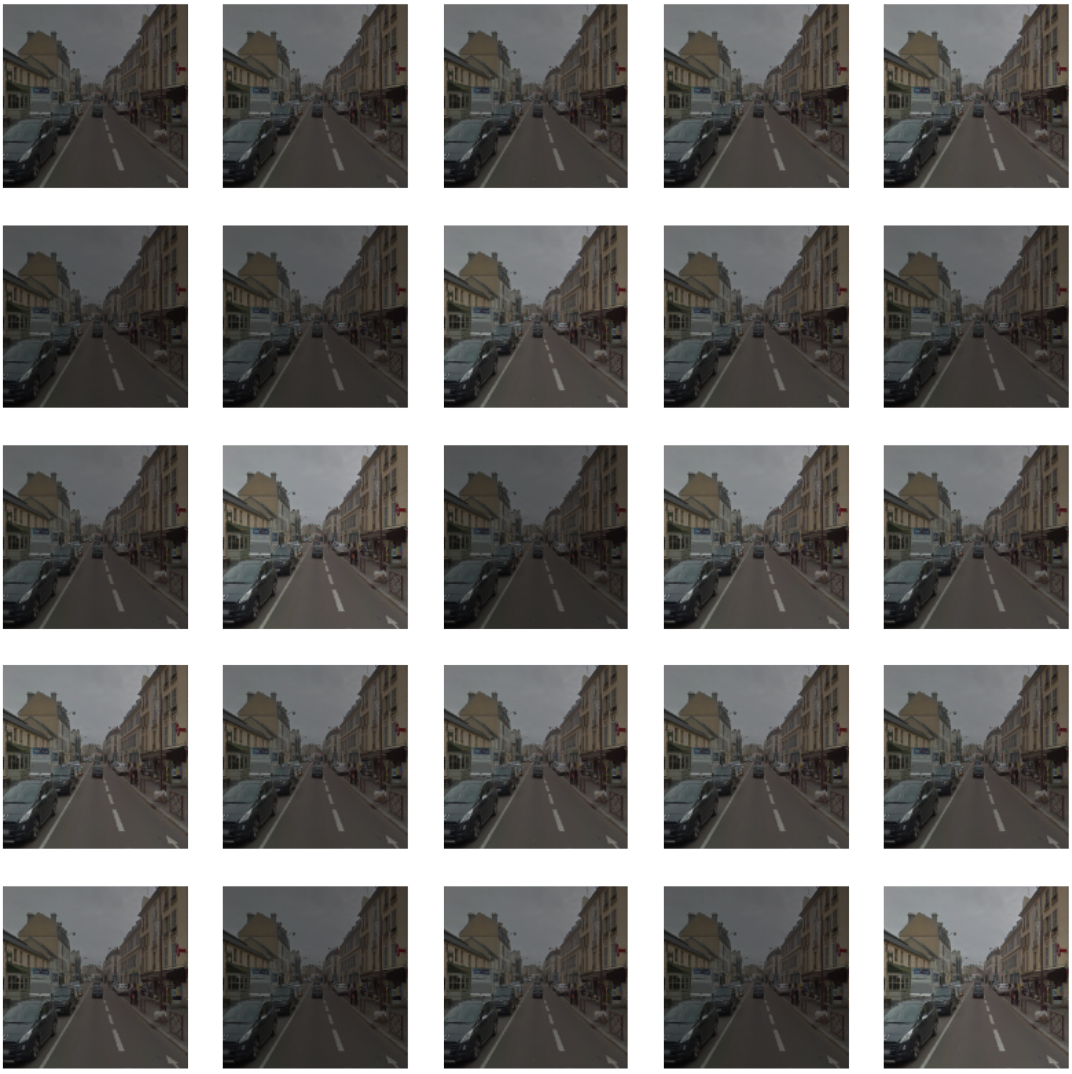


Figure 9.11: Example of 50 augmented images by random brightness from the image of Figure 9.10.

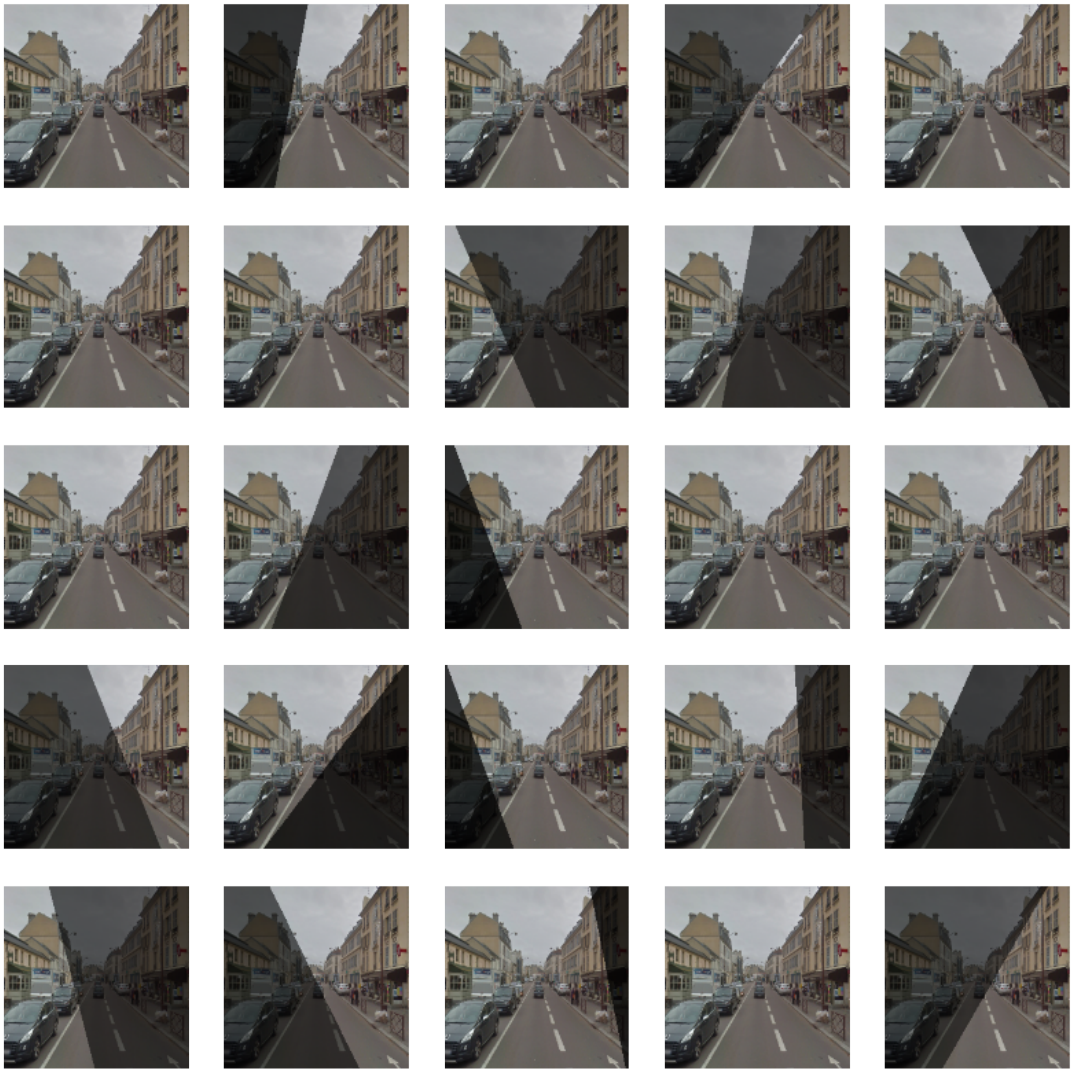


Figure 9.12: Example of 50 augmented images by random shadows from the image of Figure 9.10.

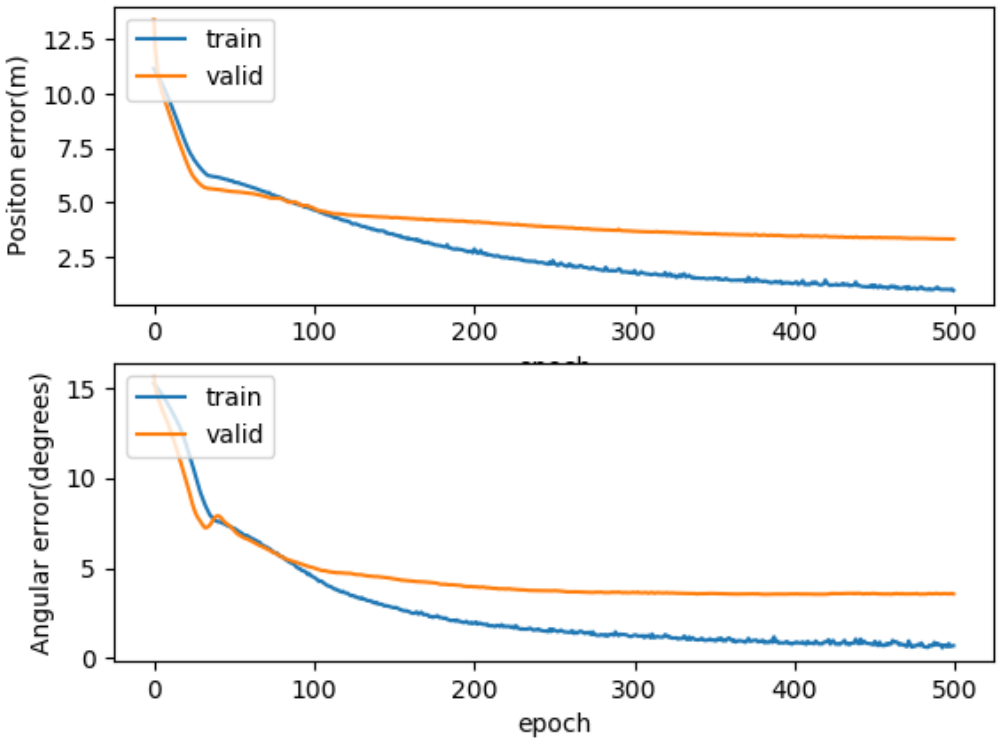


Figure 9.13: PoseNet applied on Very Augmented Street Views: the evolution of position and angular average errors on the training and validation datasets with Very Augmented Street Views in PoseNet during 500 epochs.

error still seem to decrease after the 500th epoch, and the final validation position error is much lower.

9.2 Adapted PoseNet on Very Augmented Street Views

In the previous section, we demonstrated that the insufficient Street View training dataset is the major cause of too large error gap between training and valid datasets in PoseNet. Adding more synthesized images is helpful to improve the performance of the regressor. According to Chapter 8, we can still improve the performance of a pose regressor on the following aspects:

- **Change the convnet architecture from GoogleNet to ResNet.** The choice of the base network is also critical for transfer learning. The inception model in GoogleNet is a big advantage but it is not as deep as ResNet to discover the distinctive features. The residual module in ResNet can explore more distinctive CNN features without model degradation. Also, there are indicators [78] that ResNet is better at training a model without overfitting when dealing with a rather small training dataset.

- **Modify the regressor outputs.** We observe no big changes among the

parameters such as pitch, roll and altitude in both training and test datasets. In our experimental set-up, because the camera is not handheld as PoseNet but fixed on a vehicle, and the ground is essentially flat and horizontal in our test area. Absolute camera position and orientation are significantly varying, and therefore there is no need to regress other pose DoF.

Regressor outputs simplification

As stated before, it is not necessary to render a 6 DoF pose by regressing extra parameters. The training dataset depends mainly on the camera positions and orientations, thereby the convnet model should be able to output a pose vector with only 2D position and camera yaw angle. We labeled all Street Views and their virtual views by geodetic positions $\mathbf{x}$ and global yaw angles θ *w.r.t* the true north. This pose vector $\mathbf{p}$ can be represented by:

$$\mathbf{p} = [\mathbf{x}, \theta] \tag{9.4}$$

Note that the locations $\mathbf{x}$ should be converted in UTM cartesian coordinates during the training process. We also use mean-value subtraction to reduce the ordre of magnitude for geodetic position values. Of course, for each test image the same mean position has to be added to the position output by our convnet in order to obtain an estimated absolute position. The yaw angle θ within $[0, 2\pi]$, which is computed by the panorama's yaw α and the virtual camera's yaw η, is normalized to $[0, 1]$. Therefore, the labels for every image in the training and test dataset are also modified according to the scheme explained above.

In order to train the convnet, we use the sum of the Euclidean error of both the translation and orientation can be seen in Equation. 9.5. In a similar fashion as [100] and [40], a scale factor β is used to balance the error magnitude of the position and angular components. The optimal penalty weight is obtained by grid search during the fine tuning phase. In PoseNet architecture, the recommendation for the scale factor in outdoor scenarios ranges from 200 to 2000. This scale factor is for positions and quaternions. In the test, we set the scale values in a grid as $[10, 100, 150, 200, 300, 400]$ and found that when $\beta = 200$, the loss curve demonstrates the best regression result for the sequence No.11.

$$\mathcal{L} = \|\hat{\mathbf{x}} - \mathbf{x}\|^2 + \beta\|\hat{\theta} - \theta\|^2 \tag{9.5}$$

9.2.1 Architecture Modification

We build our network upon the existing ResNet 50 architecture which is pre-trained on the ImageNet dataset[5]. ResNet takes a 224×224 pixels RGB image as input and in the high level the input is converted to a $7 \times 7 \times 2,048$ feature maps by several convolutions and max-pooling layers. Through the average pooling and a fully-connected layer, the features maps output as a $1,000$-dimensional vector as a classifier in the end. We modified this model as follows: The final classifier is replaced by the pose regressor. Instead of directly adding the final 3-dimension

[5]At the beginning we used ResNet 152 and a memory problem occurred on the AWS platform. Since the accuracy of ResNet 50 and 152 are similar, we use a small ResNet block to train our system.

		PoseNet			Adapted PoseNet
		Original Street Views	Augmented Street Views	Very Augmented Street Views	Very Augmented Street Views
Valid	Position error (m)	13.65	9.41	3.59	1.61
	Angular error (*deg*)	16.79	1.83	3.24	8.57
Train	Position error (m)	8.54	0.56	0.73	0.08
	Angular error (*deg*)	1.28	0.50	1.28	0.94
Test	Position error (m)	48.13	36.72	9.86	7.62
	Angular error (*deg*)	3.34	2.43	3.79	3.55

Table 9.1: Comparison of the average position and angular error obtained by 3 different scales of training datasets in 2 convent architectures. Note that errors in the training and validation process are recorded at the 500th epoch.

pose output layer, we add the 2-dimensional position and 1-dimensional angle layers separately from the final FC layer. As inspired by the PoseNet architecture, this setting is meaningful to improve the accuracy and generalize the model.

Training Procedure

The modified network (called Adapted PoseNet) is trained using the Keras and TensorFlow library. Every input image is resized into 224×224 pixels. The Adam optimizer with a learning rate of 10^{-5} is employed to realize gradient descent. Batch size is 80, and training is conducted during 500 epoch. We also split 20% for the training dataset to form the validation dataset in order to observe whether overfitting happens according to the loss function. All the augmented images are trained by the AWS EC2 g2.2xlarge instance (with 8 virtual NVIDIA GRID K520 GPUs).

The history of training process with Very Augmented Street Views can be seen in Figure 9.14. As illustrated, the loss curves of the training and validation datasets achieve a suitable error range in terms of position and global yaw angle. Actually, our model converges considerably after 100 epochs, which is later than the results to evaluate PoseNet. The reason is that ResNet 50 only provides weights pretrained with ImageNet while PoseNet uses the well-trained weights in Place datasets which would speed up the training process for the pose estimation task. Table 9.1 illustrates a comparison of all the experiments that we have done . From this table, our adapted PoseNet outperforms others in all types of errors.

9.2.2 Result and Discussion

It would be interesting to visualize and interpret the CNN features from this model. To do so, we extract high CNN features from the final convolutional layers. For this test sequence, we have a total of 43268 training images (instances) and the output from the final convolutional layer of ResNet 50 is of in an enormous dimension: $[43268 \times 7 \times 7 \times 2048]$. In deep learning, t-SNE (t-distributed Stochastic Neighbor Embedding) is a very popular tool to visualize the high dimensional vectors, which is able to reduce a high dimensional vector to a low dimensional space while preserving the Euclidean distance among them. Yet it does not scale very well with features that are bigger than a few thousand instances. Thereby, we first use PCA to reduce the dimension to 2000 instances and then apply the t-SNE to project high CNN features to a 2D space. As depicted in Figure 9.15, it indicates that the learnt CNN features have captured some generic information between geometry, scene, camera and topology, regardless of whether it is expressed in the final localization output.

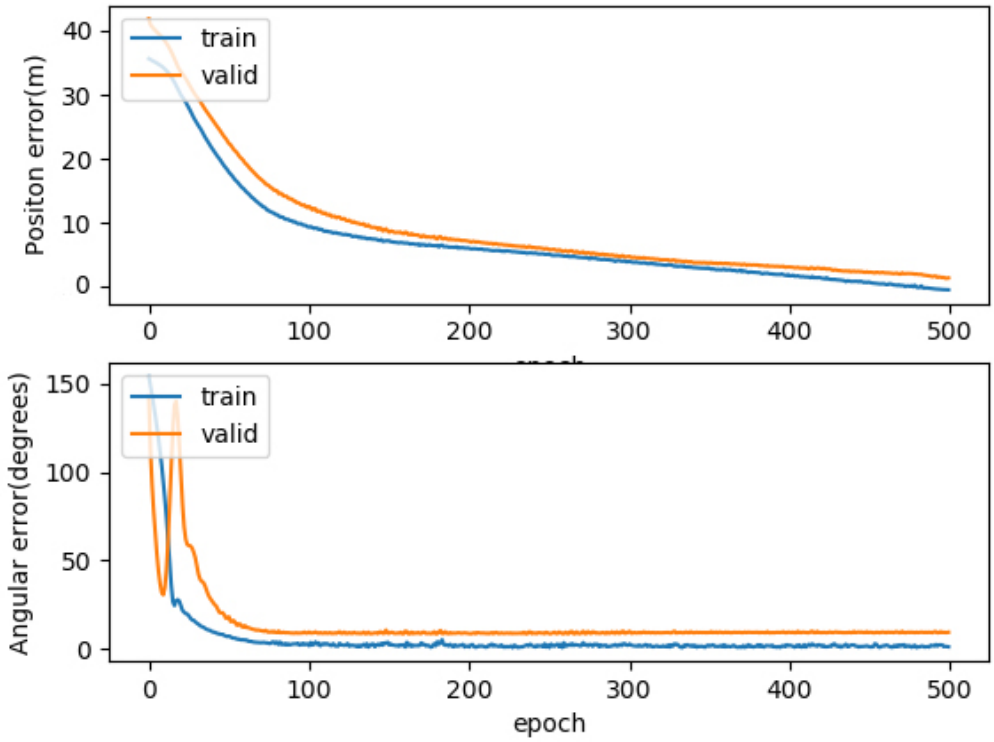


Figure 9.14: Adapted PoseNet applied on Very Augmented Street Views: the evolution of position and angular average errors on the training and validation datasets with Original Street Views in PoseNet during 500 epochs.

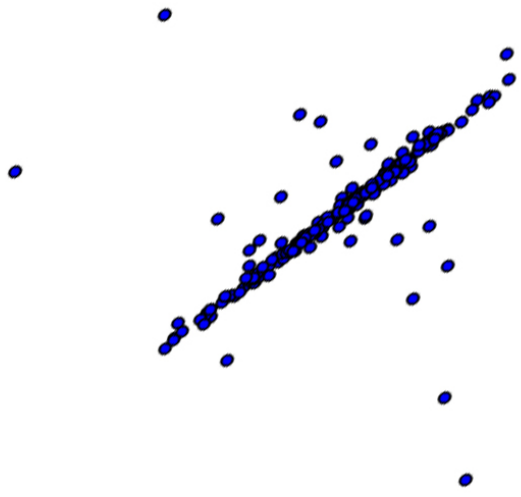


Figure 9.15: The visualization of the final convolutional layer obtained by reducing the high dimension to 2D space. This reduced visualization probably suggests that it is possible to compute the pose information by a non linear function with the higher-level CNN features.

Figure 9.16: Preprocessed test images #1 (left) and #545 (right) in the sequence No.11.

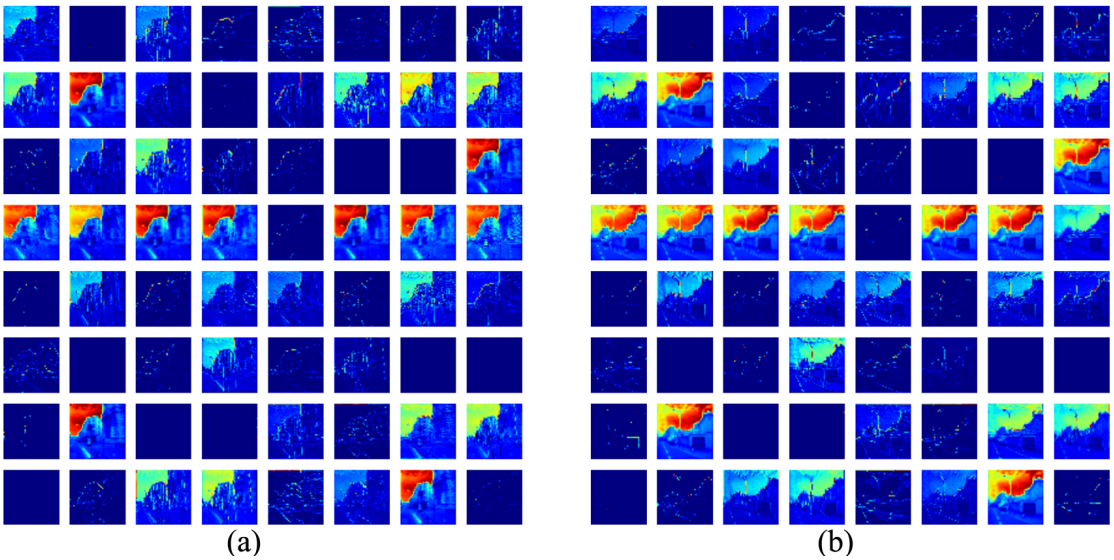


Figure 9.17: Visualization of the feature maps for Figure 9.16 after the first convolutional layer: some edges of building facades are detected.

After verifying this model, we save the customized training weights for the test dataset. The pose regressor should be able to predict the pose for each test image in an end-to-end manner. Taking the test images of Figure 9.16 as an example, we also illustrate the CNN feature learning process by visualizing its high and low level feature maps. In ResNet 50, the feature maps size in the first convolutional layer is $[112 \times 112 \times 64]$ and the final convolutional layer $[7 \times 7 \times 1028]$. Figure 9.17 and Figure 9.18 illustrate the feature learning procedure in our designed model: after the low layer, some geometric elements, such as the building shapes, are detected; yet, after the high level, we only detect some blobs and areas that are useful to the pose estimation. For convenience purposes, we only display the first 128 features maps instead of 1028 for the final layer. If we go through the high level feature maps from the last convolutional layer, we find that most detected "objects" (or CNN features) lie in the horizon of the scene. It is interesting that in the handcrafted feature method, we mostly detect interesting points in the nearer part of the scene while the pose related CNN features seem more generic and focus more on the scene around the point at infinity.

The localization result of the sequence No.11 is illustrated in Figure 9.21. In

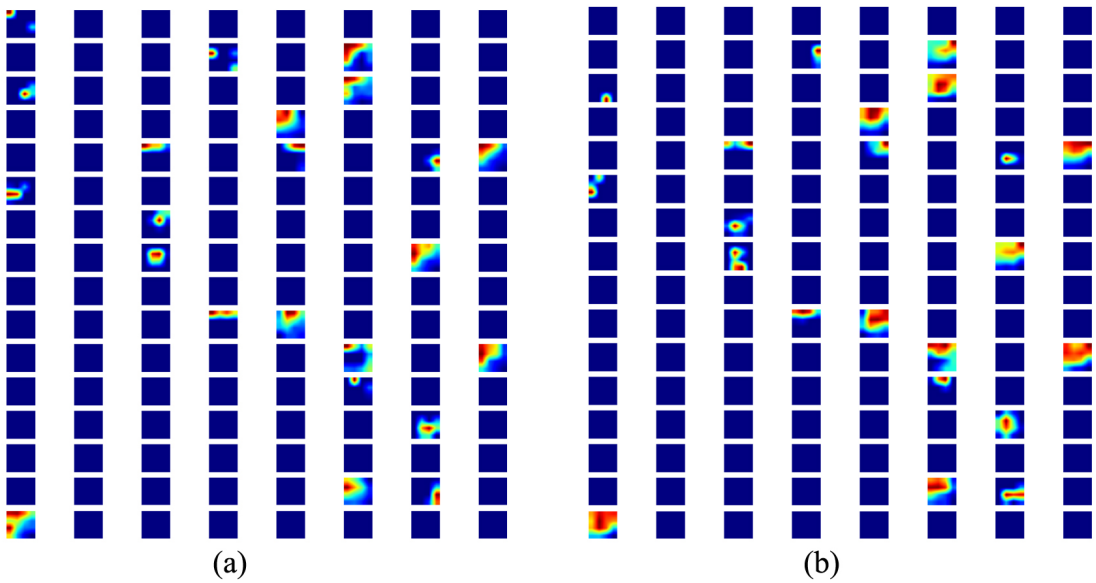


Figure 9.18: Visualization of the feature maps for Figure 9.16 after the last convolutional layer: some high-level objects are detected

the test process, the computation cost is about 75ms for estimation by our trained convnet of the global position and yaw from one monocular image. As shown in Figure 9.21, the convnet can produce all pose estimates along the trajectories from the input monocular image and achieve an average error 7.62m in position accuracy. Compared to the ground truth and the handcrafted feature based methods, the convnet positioning is not very metric and more severe jumps appear occasionally around the trajectory, see more in Figure 9.20. We infer that the lateral offset and jumps are mainly caused by the imbalanced quantity of two-side street images in the training process. Reviewing the augmentation process, we selected potential useful virtual images with few null pixels, as such, in Figure 9.7, 9.8 and 9.9, only the images with a yaw around $[150°, 270°[$ meet our requirements as training data to be used in training process. This imbalance of selection process is probably a reason for the jitters in localization. In essence, this discontinuity come from the lack of training data with good urban appearance. To eliminate these phenomena, our model can be further improved by storing the prior estimates to have a good smoothness. Although the accuracy seems to be unpersuadable to display the metric performance of our model, when compared with PoseNet's $0.48m$ average error on the Microsoft 7-Scenes dataset (captured in a $2.5m \times 1m \times 1m$ area in a dense sequence [178]), our convnet arrives a relatively high precision with few training datasets in a wider urban environment.

9.2.3 Perspectives

The localization work with the convnet was started in the very end of the PhD study. There are still further works to study.

One important refinement should be done in Equation 9.5. Strictly speaking, this loss function is not very suitable, for example, when $\hat{\theta} = 1°$ and $\theta = 350°$, the error would be 349° according to this formula instead of the true value 11°. A good way to improve the function should be use $\sin\theta$ or $\cos\theta$ instead of θ but it will

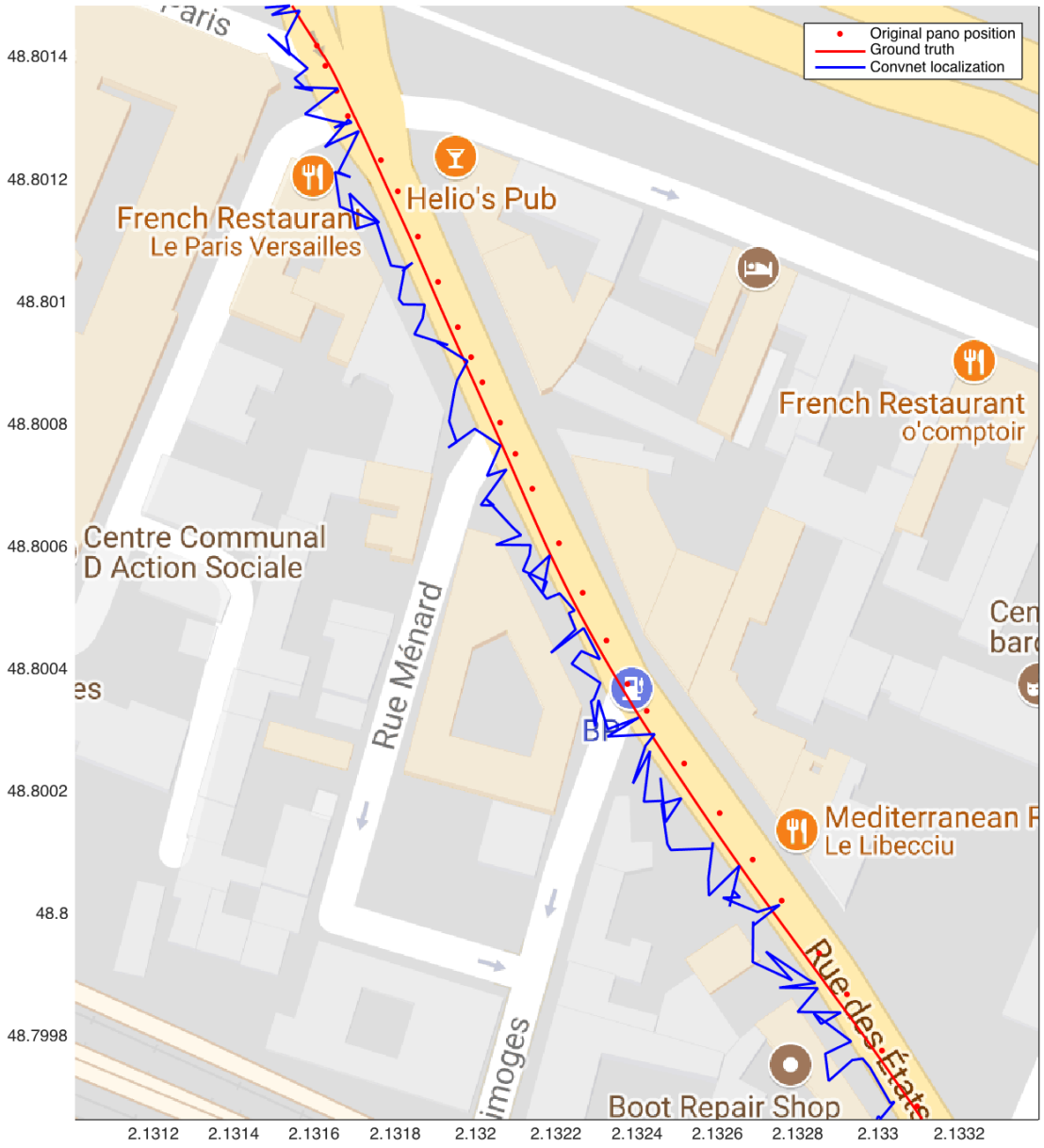


Figure 9.19: The output from a single localization run using the modified convnet. The red points represent the locations of the Street View cameras. The red and blue lines mark RTK-GPS ground truth and the estimated positions of the monocular camera respectively.

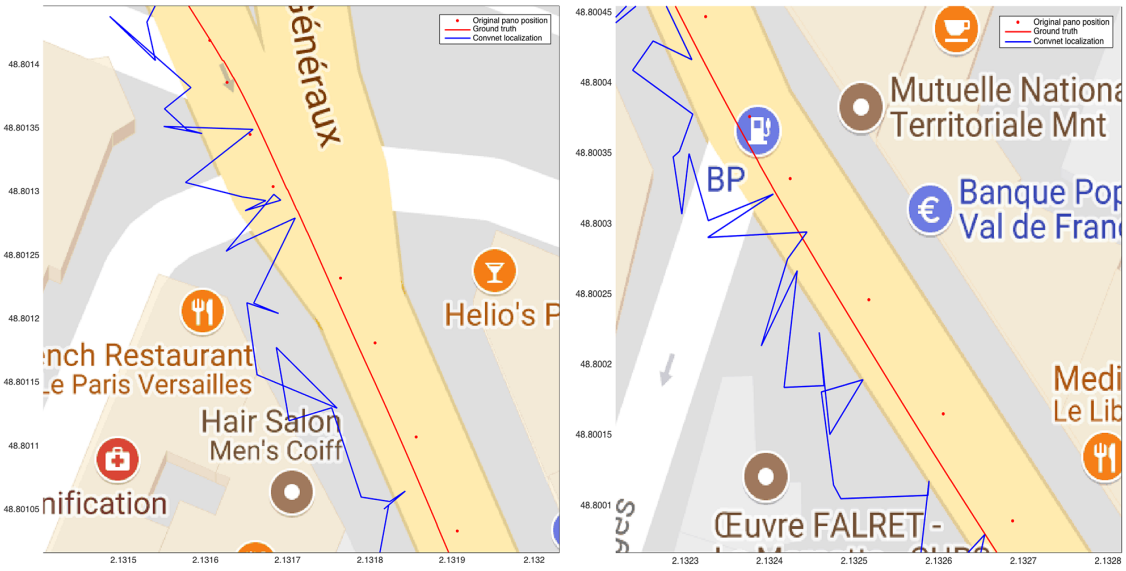


Figure 9.20: 2 close-up views of the localization result in Figure 9.21.

make the gradient descent complicated. After verifying the maximum angular error is always less than 180° in our case, we still adopted this simple formula to train the convnet.

Moreover, it is also very interesting to study the influence of β to the convnet performance as conducted in PoseNet. We believe that β is a core parameter to generalize our convnet to be used in much larger scale urban areas (with more sequences like the sequence No.11). Regarding the metric accuracy and the smoothness of localization, it might be also improved a lot by a simple filter algorithm. All these further works will make the convent be more potential localization in the urban area.

9.3 Generalization to Test Area

In the previous sections, we selected the sequence No.11 in Table 9.2 to do the experiment, for its good urban appearance. Similar to the handcrafted feature method, we test whether this convnet based localization method is capable to be generalized to the whole test area. The convnet training for the sequence No.11 requires nearly 8 hours for each single trial. If we test the grid search to find all best scale factors β for every sequence, it becomes intractable. In fact, PoseNet has already proved that the convnet is a good tool to apply in outdoor localization. Thereby, in our thesis, we will focus on the comparison between the handcrafted and CNN features instead of searching an optimal β value. For saving time, we fix the scale factor $\beta = 200$ (the optimal value for sequence No.11)for all rest sequences. If overfitting or under-fitting occurs in a test segment, we suppose that our method can not be applied in this test segment. We test our algorithms in the rest of the test area and the results are illustrated in Table 9.2 and Figure 9.21 to 9.24.

Observed from the table and figures, the average spatial accuracy is not as precise as the handcrafted feature based method and the metric level is similar to a raw GPS in the urban environment. The position trajectories are not as smooth as the handcrafted feature methods as well but they are close to the locations of original

Sequence	Vehicle	Number of Frame	Spatial Extent	Number of Street View	Number of Training images	Error of Adapted PoseNet	Error of Extended Handcrafted Feature
Seq0	Zoé	554	11×265	29	43577	$7.87m$	Fail
Seq1	Scénic	250	11×79	11	16489	$7.46m$	3.06m
Seq2	Scénic	898	11×271	29	42980	$7.93m$	2.63m
Seq3	Scénic	895	11×222	29	36760	Fail	Fail
Seq4	Scénic	291	11×128	12	17592	Fail	Fail
Seq5	Scénic	841	11×317	33	50556	$8.38m$	2.54m
Seq6	Scénic	901	11×216	34	50864	$7.55m$	2.82m
Seq7	Scénic	306	11×184	16	21120	Fail	Fail
Seq8	Scénic	141	11×69	8	12249	Fail	Fail
Seq9	Scénic	422	11×198	19	26581	Fail	Fail
Seq10	Scénic	899	11×382	36	46455	Fail	Fail
Seq11	Scénic	897	11×234	29	43268	$7.62m$	2.85m
Seq12	Scénic	290	11×56	4	5946	$7.20m$	2.92m
Seq13	Scénic	899	11×361	36	53892	$7.69m$	2.49m
Seq14	Scénic	348	11×131	13	19357	$7.37m$	2.67m
Seq15	Scénic	625	11×196	17	25382	$8.42m$	2.58m
Seq16	Scénic	896	11×145	32	47532	$7.17m$	2.44m
Seq17	Scénic	172	11×93	15	22185	$6.89m$	1.56m
Seq18	Scénic	897	11×234	29	41633	$7.45m$	2.13m

Table 9.2: Localization results for test segments are obtained by the convnet with a fixed β value in the loss function. In order to compare easily, we also display the average spatial errors obtained by the extended handcrafted-feature-based method. Notice that the sequences where the adapted PoseNet fails are also the sequences for which the handcrafted-feature-based method does not work well.

panoramas. There is no obvious relationship between the accuracy and the spatial extent in all these sequences.

One interesting point is that the convnet model does not converge well in the same sequences that are unsuccessful for the handcrafted feature approach. We believe that some distinctive urban appearance is indispensable for both the traditional and CNN features to estimate poses. Long distance vegetation along the trajectory will degrade the extraction for both handcrafted and CNN features.

As stated in Chapter 7.4, the Zoé dataset captures many similar scenes at different scales and it affects a good performance of the place recognition approach while our convnet is able to render good localizations for this dataset, see the sequence No.0. We deduce that the pose-orientated CNN features ignore the scale problem and to some degree the scale helps the model to decide where the vehicle is located. According to our visualization, the high level CNN features pay close attention on the scene near the horizon (or around the point at infinity).

In addition, for the Scénic dataset, we only tested the right camera as in Chapter 7. All the results in the Scénic dataset show a slight right shift compared to the ground truth. We also evaluated the test images from the left camera from the Scénic and a left shift appears. As we mentioned before, we can regard the regressed pose as an output by computing a series of non-linear functions from the high layer CNN features. We infer that this shift mainly depends on the weights generated by the similar training datasets (namely, the original or virtual Street Views in the right direction). It is true that we can get mean positions from both the left and right side (to get a better positioning but this is against our monocular localization goal. Also, the principle behind this technique is not well-founded.

When capturing the sequence No.6 and No.12, the car was stopped at a traffic cross and the camera registered many similar images[6]. Comparing the results from Fig 9.22 and Fig 7.22, we found that our convnet is able to regress the same position for these similar images which is not the case with the handcrafted feature method. In this aspect, the CNN features are more generic than the handcrafted features.

[6]In urban environments, other mobile bodies can still change the scene to some degree, which only influences the traditional feature extraction.

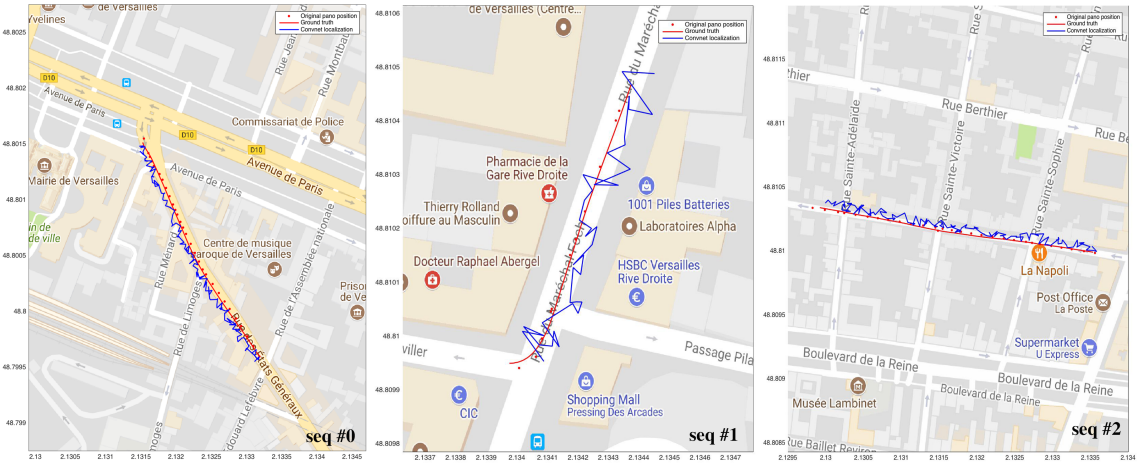


Figure 9.21: The localization output from sequence No. 0, 1 and 2 obtained by the convnet method. The original panorama position, the ground truth and the convnet results are noted in red point, red line and blue line respectively.

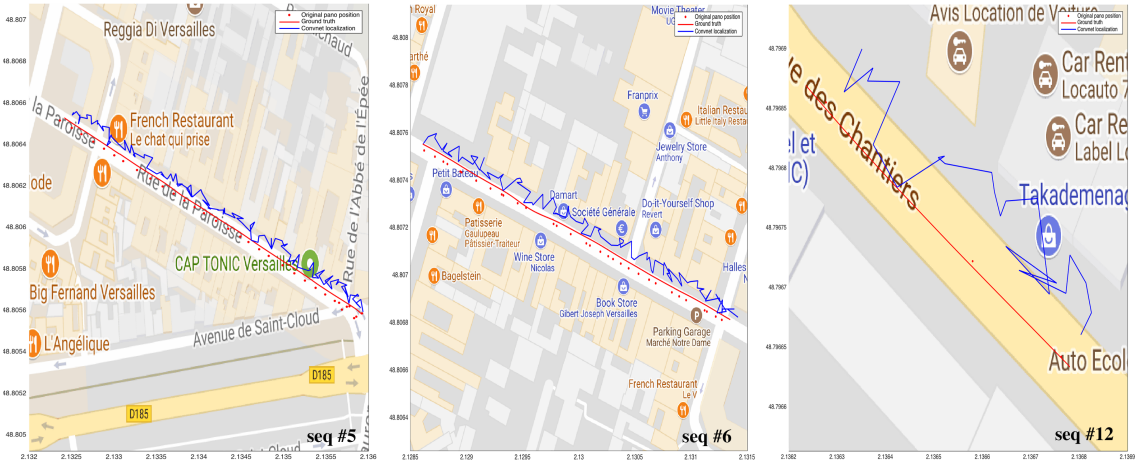


Figure 9.22: The localization output from sequence No. 5, 6 and 12 obtained by the convnet method. The original panorama position, the ground truth and the convnet results are noted in red point, red line and blue line respectively.

Figure 9.23: The localization output from sequence No. 13, 14 and 15 obtained by the convnet method. The original panorama position, the ground truth and the convnet results are noted in red point, red line and blue line respectively.

Figure 9.24: The localization output from sequence No. 16, 17 and 18 obtained by the convnet method. The original panorama position, the ground truth and the convnet results are noted in red point, red line and blue line respectively.

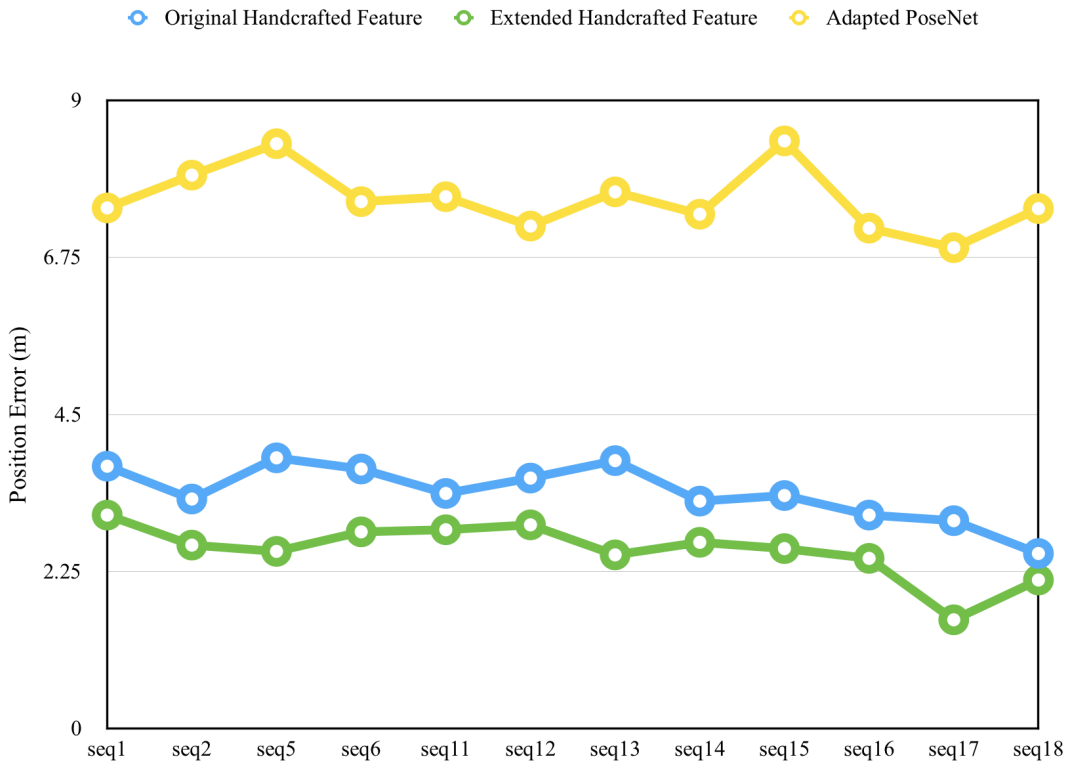


Figure 9.25: Comparison of Handcrafted and CNN feature based methods.

9.4 Comparison of Handcrafted and CNN Feature based Methods

To our best knowledge, there are few state-of-the-art approaches that consider the visual localization problem with only GIS data. Naturally, our two methods provide a way to compare between the traditional handcrafted feature based and CNN feature based methods. According the results from the same test area, it is clear that the handcrafted feature based method works better in accuracy than the CNN feature based, see Fig 9.25. However, it is incautious to make a final conclusion that one method is superior than the other. Currently, learning methods appear to be dominant in the computer vision community but handcrafted features with the sophisticated theory are still advantageous in many aspects. Few years ago, we had the idea that maps are Euclidean and the pose could naturally be presented in 6 DoF within the map. However, the birth of PoseNet opens a new window that the map and the pose might be a low dimensional representations from high dimensional CNN features. In visual localization, we normally have three processes for the handcrafted feature methods, namely feature extraction, feature association and pose estimation. But recently more and more learning oriented researches appear, including learning based local feature selection, learning based outlier rejection and till now learning based pose estimation. Definitely, handcrafted feature methods are the foundation on which learning methods are build upon and they enhance each other as well. Here we give a general discussion only based on the results we obtained.

- **Accuracy**: comparing the results in Table 7.3 and Table 9.2, it is clear that

the handcrafted feature based method achieves more accurate and smooth positions. There is no doubt that kVLD descriptor is able to extract robust features regardless of big changes in viewpoints, illumination and rotations. With the sophisticated design and a RANSAC, this handcrafted descriptor is the propre (specific) choice to our task. Then the pose is estimated on a PnP optimization and then a local bundle adjustment. These two optimization schemes contribute to a high precision. Instead, in the convnet method, the artifacts and missing pixels in the virtual images are the main cause to train a well performed regressor. As we mentioned before, this is because CNN features are more general than the local handcrafted features.

- **Efficiency**: the average time cost to localize one single online image is more than 3s with the handcrafted based methods. Most of the time is spent on the place recognition and optimization process. Compared to the 20 frame-per-second frequency of the MIPSEE camera, it is hard to realize a real time positioning task. However, if we do not consider the training process, the convnet regressor is able to predict the global position for one image within 75ms in average. This end-to-end way is advantageous for the real time implementation with the right hardware.

- **Simplicity**: in the convnet method, we used a hyper parameter β in the loss function to balance the scale magnitude. A grid search method is used to test the loss curves for different values of β. Once this parameter is well selected, we can generalize the same value to other sequences with a similar spatial extent. Yet, in the handcrafted feature based approach, the dictionary sizes in the BoW should be chosen differently according to each sequence by grid search. This is because the different sequences would have various quantities of visual words. Moreover, in the handcrafted feature based approches, we fixed the camera towards the facades to make sure our place recognition would operate properly. Instead, our convnet method does not focus on these beforehanded settings too much and can be employed for both the Zoé and Scénic dataset.

- **Compatibility**: we must consider the camera intrinsic parameters to estimate the pose with handcrafted features. The convnet method can overlook these details and learns high dimensional CNN features which contain the relationship between the camera, geometry and pose. As we can seen, the sequence No.0 and sequence No.11 were captured in the same area but by different VEDECOM vehicles. However, once a regressor is well trained in this test area, no matter what kind of capturing vehicles are used, we can render the global pose directly. To this degree, the convnet method is much easier.

9.5 Conclusion

In this chapter, we have presented a learning based method to use few Google Street View imagery to train an end-to-end convnet pose regressor. We tested the state-of-the-art algorithms and based on the evaluation, we proposed several ways to deal with overfitting due to our limited training dataset. We augmented the training datasets by a thousand factor using a few panoramas. All the synthesized images are derivative from a small number of the original images, which are still the real

training datasets. However, the transfer learning based convnet still demonstrates its power to learn good CNN features to represent the pose. The generic representations are obtained by learning the multiple scenarios, including different geometries, topologies, illuminations and camera settings. Thus they are robust to illumination and important view point changes. Compared to the image-based localization method, there is a precision degradation but this data driven method can be a potential solution in the GIS-based localization.

Part V

Conclusions of the Thesis

Chapter 10

Conclusions and Perspectives

Autonomous driving has become more and more attractive to both industrial and academic sectors in recent years. Determining a precise localization for a self driving car is one of the crucial problems. This thesis investigated the current visual localization systems from handcrafted feature based methods to learning based approaches. Since GIS's have been well studied in this decade, we proposed to rely on the abundant information from available GIS's to simplify and improve the current visual localization systems in two major aspects: rendering a metric global positions directly and avoiding the mapping building process as studied in SLAM.

The first task we conducted is to make the offline commercial GIS data (Google Street View) compatible with our captured data. We explored and analyzed tremendous information in the current GIS, including topological/semantic/metric maps, Street Views, depth maps, 3D cadastral maps and High Definition maps. We selected position-oriented information from them to design a compact topo-metric representation for a good accessibility in a self-driving vehicle. This representation contains topologies, geo-coordinates, panoramic Street Views and associated depth maps. These selected data are organized as an offline dataset. Meanwhile, we captured online datasets with low-cost cameras equipped on the VEDECOM vehicles. In order to make spheric Street Views compatible to the online imagery, we adopted an image warping and interpolation based transformation to render rectilinear images from Street Views.

Based on the prepared online and offline datasets, we developed two localization methods, one is a handcrafted features based computer vision approach, the other is a convolutional neural network based learning technique. In computer vision, extracting points of interest as handcrafted features is a dominant approach to solve the image based positioning. We fully leveraged the topo-metric representation to build a coarse-to-fine localization system, namely a topological place recognition process and then a metric pose estimation by a graph optimization. The only input of this approach is an image sequence from a monocular camera and an offline prepared database. The method was tested in an urban environment and demonstrated both sub-meter accuracy and robustness to viewpoint changes, illumination and occlusion. We also refined this framework by the construction of the augmented Street Views database to compensate the sparse distribution of Street Views and improve the localization precision and smoothness.

The handcrafted feature method obtained a good accuracy but is hard to employ in real time due to the heavy process in the image retrieval and graph optimization.

Since the GIS is a global scale geotagged database, it motivates us to regress global localization from the trained convnet features in an end-to-end manner. We analyzed the current convnet pose regressors with the original offline database. The weak performance was caused by insufficient training data. We therefore augmented the original database by a factor of a thousand and take use of the transfer learning method to avoid overfitting and have a good performance. In the experiment, the regressor can also give a global localization of an input camera image in real time but the accuracy is less than the handcrafted feature based method. Results obtained by the two approaches provide us insights on the comparison and connection between handcrafted feature-based and convnet based methods.

Through the thesis, we proposed a method to deal with the urban localization problem with only a low-cost camera and an offline database from Google Street View. We believe that relying on existing databases to enhance and advance the development of self-driving will be a very hot topic in the future. Our work is a small step for initial exploration and there are many aspects that can be refined. In both proposed methods, we only used a simple image synthesis approach thus missing many information. In fact, a higher quality synthesis could be potentially obtained by combining information from multiple panoramas and depth maps.

Furthermore, we mostly focused on improving the smoothness of the localization by adding virtual views. Actually, fusing some ego information like odometry can solve this problem easily. Our localization accuracy depends on the precision of the GIS data, some more accurate and dense GIS database could help. For example, the depth information in Google Earth 3D model is more trustable than Street View depth maps. It is beneficial for a reliable localization system by matching online captured 3D points with the accurate geo-registrated point clouds from Google Earth 3D model.

In addition, in the test of the learning model, we focused on the overfitting problem and the parameter analysis is overlooked. Some parameter settings, such as the maximum distance $4m$ to synthesize virtual images, were referred from the handcrafted feature methods directly. As such, there are lots of directions to be further studied, for example, to evaluate the method performance with the scale of the environment, to research the influence of various parameter settings in the training model, including the number of required pixels in training images, the optimal spatial density of panoramas for a good training, etc.

Finally, the urban environment is very dynamic and the street appearance often changes due to building construction and destruction. How to update the GIS dataset with our online captured images is also a profound subject to study in the future.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] P. Agarwal, W. Burgard, and L. Spinello. Metric localization using google street view. *arXiv preprint arXiv:1503.04287*, 2015.

[3] M. Agrawal, K. Konolige, and M. R. Blas. Censure: Center surround extremas for realtime feature detection and matching. In *European Conference on Computer Vision*, pages 102–115. Springer, 2008.

[4] A. Alahi, R. Ortiz, and P. Vandergheynst. Freak: Fast retina keypoint. In *Computer vision and pattern recognition (CVPR), 2012 IEEE conference on*, pages 510–517. Ieee, 2012.

[5] A. Angeli, D. Filliat, S. Doncieux, and J.-A. Meyer. Fast and incremental method for loop-closure detection using bags of visual words. *IEEE Transactions on Robotics*, 24(5):1027–1037, 2008.

[6] D. Anguelov, C. Dulong, D. Filip, C. Frueh, S. Lafon, R. Lyon, A. Ogale, L. Vincent, and J. Weaver. Google street view: Capturing the world at street level. *Computer*, 43(6):32–38, 2010.

[7] M. O. Aqel, M. H. Marhaban, M. I. Saripan, and N. B. Ismail. Review of visual odometry: types, approaches, challenges, and applications. *SpringerPlus*, 5(1): 1897, 2016.

[8] R. Arandjelović and A. Zisserman. Three things everyone should know to improve object retrieval. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2911–2918. IEEE, 2012.

[9] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic. Netvlad: Cnn architecture for weakly supervised place recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5297–5307, 2016.

[10] C. Audras, A. Comport, M. Meilland, and P. Rives. Real-time dense appearance-based slam for rgb-d sensors. In *Australasian Conf. on Robotics and Automation*, 2011.

[11] Y. Avrithis and G. Tolias. Hough pyramid matching: Speeded-up geometry re-ranking for large scale image retrieval. *International journal of computer vision*, 107(1):1–19, 2014.

[12] G. Baatz, K. Köser, D. Chen, R. Grzeszczuk, and M. Pollefeys. Leveraging 3d city models for rotation invariant place-of-interest recognition. *International journal of computer vision*, 96(3):315–334, 2012.

[13] A. Babenko and V. Lempitsky. The inverted multi-index. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3069–3076. IEEE, 2012.

[14] T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *IEEE Robotics & Automation Magazine*, 13(3):108–117, 2006.

[15] T. D. Barfoot. State estimation for robotics, 2017.

[16] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[17] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. *Computer vision–ECCV 2006*, pages 404–417, 2006.

[18] Y. Bengio. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 17–36, 2012.

[19] Y. Bengio, I. J. Goodfellow, and A. Courville. Deep learning. *Nature*, 521: 436–444, 2015.

[20] J.-L. Blanco, J.-A. Fernandez-Madrigal, and J. Gonzalez. A new approach for large-scale localization and mapping: Hybrid metric-topological slam. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 2061–2067. IEEE, 2007.

[21] J.-L. Blanco, F.-A. Moreno, and J. González. A collection of outdoor robotic datasets with centimeter-accuracy ground truth. *Autonomous Robots*, 27(4): 327–351, November 2009. ISSN 0929-5593. doi: 10.1007/s10514-009-9138-7. URL `http://www.mrpt.org/Paper:Malaga_Dataset_2009`.

[22] E. Brachmann, A. Krull, S. Nowozin, J. Shotton, F. Michel, S. Gumhold, and C. Rother. Dsac-differentiable ransac for camera localization. *arXiv preprint arXiv:1611.05705*, 2016.

[23] E. Brachmann, F. Michel, A. Krull, M. Ying Yang, S. Gumhold, et al. Uncertainty-driven 6d pose estimation of objects and scenes from a single rgb image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3364–3372, 2016.

[24] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008.

[25] M. A. Brubaker, A. Geiger, and R. Urtasun. Lost! leveraging the crowd for probabilistic visual self-localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3057–3064, 2013.

[26] M. Bujnak, Z. Kukelova, and T. Pajdla. A general solution to the p4p problem for camera with unknown focal length. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[27] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. Brief: Binary robust independent elementary features. *Computer Vision–ECCV 2010*, pages 778–792, 2010.

[28] G. Carrera, J. Savage, and W. Mayol-Cuevas. Robust feature descriptors for efficient vision-based tracking. *Progress in Pattern Recognition, Image Analysis and Applications*, pages 251–260, 2007.

[29] R. Caruana. Learning many related tasks at the same time with backpropagation. In *Advances in neural information processing systems*, pages 657–664, 1995.

[30] S. Ceriani, G. Fontana, A. Giusti, D. Marzorati, M. Matteucci, D. Migliore, D. Rizzi, D. G. Sorrenti, and P. Taddei. Rawseeds ground truth collection systems for indoor self-localization and mapping. *Autonomous Robots*, 27(4): 353, 2009.

[31] K.-T. Chang. *Geographic information system*. Wiley Online Library, 2006.

[32] D. M. Chen, G. Baatz, K. Köser, S. S. Tsai, R. Vedantham, T. Pylvänäinen, K. Roimela, X. Chen, J. Bach, M. Pollefeys, et al. City-scale landmark identification on mobile devices. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 737–744. IEEE, 2011.

[33] F. Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[34] S. Choudhary and P. Narayanan. Visibility probability structure from sfm datasets and applications. In *European conference on computer vision*, pages 130–143. Springer, 2012.

[35] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467, 1968.

[36] O. Chum and J. Matas. Matching with prosac-progressive sample consensus. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 220–226. IEEE, 2005.

[37] O. Chum and J. Matas. Optimal randomized ransac. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(8):1472–1482, 2008.

[38] J. Civera, A. J. Davison, and J. Montiel. Inverse depth parametrization for monocular slam. *Robotics, IEEE Transactions on*, 24(5):932–945, 2008.

[39] J. Civera, O. G. Grasa, A. J. Davison, and J. Montiel. 1-point ransac for extended kalman filtering: Application to real-time structure from motion and visual odometry. *Journal of Field Robotics*, 27(5):609–631, 2010.

[40] R. Clark, S. Wang, A. Markham, N. Trigoni, and H. Wen. Vidloc: 6-dof video-clip relocalization. *arXiv preprint arXiv:1702.06521*, 2017.

[41] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011.

[42] A. I. Comport, M. Meilland, and P. Rives. An asymmetric real-time dense visual localisation and mapping system. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 700–703. IEEE, 2011.

[43] L. Contreras and W. Mayol-Cuevas. Towards cnn map compression for camera relocalisation. *arXiv preprint arXiv:1703.00845*, 2017.

[44] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In *Workshop on statistical learning in computer vision, ECCV*, volume 1, pages 1–2. Prague, 2004.

[45] Y. Cui and S. S. Ge. Autonomous vehicle positioning with gps in urban canyon environments. *IEEE transactions on robotics and automation*, 19(1): 15–25, 2003.

[46] M. Cummins and P. Newman. Fab-map: Probabilistic localization and mapping in the space of appearance. *The International Journal of Robotics Research*, 27(6):647–665, 2008.

[47] M. Cummins and P. Newman. Appearance-only slam at large scale with fab-map 2.0. *The International Journal of Robotics Research*, 30(9):1100–1123, 2011.

[48] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. Monoslam: Real-time single camera slam. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1052–1067, 2007.

[49] J. Delhumeau, P.-H. Gosselin, H. Jégou, and P. Pérez. Revisiting the vlad image representation. In *Proceedings of the 21st ACM international conference on Multimedia*, pages 653–656. ACM, 2013.

[50] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[51] M. Donoser and H. Bischof. Efficient maximally stable extremal region (mser) tracking. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 553–560. IEEE, 2006.

[52] G. Dubbelman and B. Browning. Closed-form online pose-chain slam. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5190–5197. IEEE, 2013.

[53] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[54] V. Dumoulin and F. Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[55] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[56] N. Engelhard, F. Endres, J. Hess, J. Sturm, and W. Burgard. Real-time 3d visual slam with a hand-held rgb-d camera. In *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum, Vasteras, Sweden*, volume 180, pages 1–15, 2011.

[57] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[58] G. Floros, B. van der Zander, and B. Leibe. Openstreetslam: Global vehicle localization using openstreetmaps. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1054–1059. IEEE, 2013.

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

[60] G. Fritz, C. Seifert, M. Kumar, and L. Paletta. Building detection from mobile imagery using informative sift descriptors. In *Image Analysis*, pages 629–638. Springer, 2005.

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

[62] A. Geiger, J. Ziegler, and C. Stiller. Stereoscan: Dense 3d reconstruction in real-time. In *Intelligent Vehicles Symposium (IV)*, 2011.

[63] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[64] A. Gil, O. M. Mozos, M. Ballesta, and O. Reinoso. A comparative evaluation of interest point detectors and local descriptors for visual slam. *Machine Vision and Applications*, 21(6):905–920, 2010.

[65] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[66] A. Glover, W. Maddern, M. Warren, S. Reid, M. Milford, and G. Wyeth. Openfabmap: An open source toolbox for appearance-based loop closure detection. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 4730–4735. IEEE, 2012.

[67] Y. Gong, L. Wang, R. Guo, and S. Lazebnik. Multi-scale orderless pooling of deep convolutional activation features. In *European conference on computer vision*, pages 392–407. Springer, 2014.

[68] A. Gordo, J. Almazán, J. Revaud, and D. Larlus. Deep image retrieval: Learning global representations for image search. In *European Conference on Computer Vision*, pages 241–257. Springer, 2016.

[69] G. Grisetti, R. Kummerle, C. Stachniss, U. Frese, and C. Hertzberg. Hierarchical optimization on manifolds for online 2d and 3d mapping. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 273–278, May 2010. doi: 10.1109/ROBOT.2010.5509407.

[70] J. W. Hager, J. F. Behensky, and B. W. Drew. The universal grids: Universal transverse mercator (utm) and universal polar stereographic (ups). *DMA technical manual*, 8358, 1989.

[71] M. Haklay and P. Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4):12–18, 2008.

[72] B. M. Haralick, C.-N. Lee, K. Ottenberg, and M. Nölle. Review and analysis of solutions of the three point perspective pose estimation problem. *International journal of computer vision*, 13(3):331–356, 1994.

[73] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[74] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[75] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[76] J. Hartmann, J. H. Klussendorff, and E. Maehle. A comparison of feature descriptors for visual slam. In *Mobile Robots (ECMR), 2013 European Conference on*, pages 56–61. IEEE, 2013.

[77] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[78] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[79] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016.

[80] J. Heinly, E. Dunn, and J.-M. Frahm. Comparative evaluation of binary features. In *Computer Vision–ECCV 2012*, pages 759–773. Springer, 2012.

[81] K. L. Ho and P. Newman. Loop closure detection in slam by combining visual and spatial appearance. *Robotics and Autonomous Systems*, 54(9):740–749, 2006.

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

[83] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins. *Global positioning system: theory and practice*. Springer Science & Business Media, 2012.

[84] A. S. Huang, M. Antone, E. Olson, L. Fletcher, D. Moore, S. Teller, and J. Leonard. A high-rate, heterogeneous data set from the darpa urban challenge. *The International Journal of Robotics Research*, 29(13):1595–1601, 2010.

[85] S. Huber and C. Rust. osrmtime: Calculate travel time and distance with openstreetmap data using the open source routing machine (osrm). 2016.

[86] R. A. Hummel. Histogram modification techniques. *Computer Graphics and Image Processing*, 4(3):209–224, 1975.

[87] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[88] A. Irschara, C. Zach, J.-M. Frahm, and H. Bischof. From structure-from-motion point clouds to fast location recognition. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2599–2606. IEEE, 2009.

[89] H. Jegou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. *Computer Vision–ECCV 2008*, pages 304–317, 2008.

[90] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3304–3311. IEEE, 2010.

[91] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33 (1):117–128, 2011.

[92] H. Jegou, F. Perronnin, M. Douze, J. Sánchez, P. Perez, and C. Schmid. Aggregating local image descriptors into compact codes. *IEEE transactions on pattern analysis and machine intelligence*, 34(9):1704–1716, 2012.

[93] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[94] Y. Jiang, Y. Xu, and Y. Liu. Performance evaluation of feature detection and matching in stereo visual odometry. *Neurocomputing*, 120:380–390, 2013.

[95] E. Johns and G.-Z. Yang. Generative methods for long-term place recognition in dynamic scenes. *International Journal of Computer Vision*, 106(3):297–314, 2014.

[96] Y. Kalantidis and Y. Avrithis. Locally optimized product quantization for approximate nearest neighbor search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2321–2328, 2014.

[97] N. Karlsson, E. Di Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian, and M. E. Munich. The vslam algorithm for robust localization and mapping. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 24–29. IEEE, 2005.

[98] A. Kendall and R. Cipolla. Modelling uncertainty in deep learning for camera relocalization. *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2016.

[99] A. Kendall and R. Cipolla. Geometric loss functions for camera pose regression with deep learning. *arXiv preprint arXiv:1704.00390*, 2017.

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

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

[102] L. Kneip, D. Scaramuzza, and R. Siegwart. A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 2969–2976. IEEE, 2011.

[103] J. Knopp, J. Sivic, and T. Pajdla. Avoiding confusing features in place recognition. *Computer Vision–ECCV 2010*, pages 748–761, 2010.

[104] K. Konolige, M. Agrawal, and J. Sola. Large-scale visual odometry for rough terrain. In *Robotics research*, pages 201–212. Springer, 2010.

[105] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[106] Z. Kukelova, M. Bujnak, and T. Pajdla. Real-time solution to the absolute pose problem with unknown radial distortion and focal length. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2816–2823, 2013.

[107] P. Kulkarni, J. Zepeda, F. Jurie, P. Perez, and L. Chevallier. Hybrid multi-layer deep cnn/aggregator feature for image classification. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 1379–1383. IEEE, 2015.

[108] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g 2 o: A general framework for graph optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3607–3613. IEEE, 2011.

[109] R. Kümmerle, B. Steder, C. Dornhege, A. Kleiner, G. Grisetti, and W. Burgard. Large scale graph-based slam using aerial images as prior information. *Autonomous Robots*, 30(1):25–39, 2011.

[110] K. Lai, L. Bo, X. Ren, and D. Fox. A large-scale hierarchical multi-view rgb-d object dataset. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1817–1824. IEEE, 2011.

[111] R. B. Langley. Rtk gps. *GPS World*, 9(9):70–76, 1998.

[112] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

[113] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[114] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[115] V. Lepetit, F. Moreno-Noguer, and P. Fua. Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision*, 81(2):155–166, 2009.

[116] S. Leutenegger, M. Chli, and R. Y. Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555. IEEE, 2011.

[117] J. Levinson, M. Montemerlo, and S. Thrun. Map-based precision vehicle localization in urban environments. In *Robotics: Science and Systems*, volume 4, page 1, 2007.

[118] R. Li, Q. Liu, J. Gui, D. Gu, and H. Hu. Indoor relocalization in challenging environments with dual-stream convolutional neural networks. *IEEE Transactions on Automation Science and Engineering*, 2017.

[119] Y. Li, N. Snavely, and D. P. Huttenlocher. Location recognition using prioritized feature matching. In *European conference on computer vision*, pages 791–804. Springer, 2010.

[120] Y. Li, N. Snavely, D. P. Huttenlocher, and P. Fua. Worldwide pose estimation using 3d point clouds. In *Large-Scale Visual Geo-Localization*, pages 147–163. Springer, 2016.

[121] H. Lim, S. N. Sinha, M. F. Cohen, and M. Uyttendaele. Real-time image-based 6-dof localization in large-scale environments. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1043–1050. IEEE, 2012.

[122] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[123] Z. Liu and R. Marlet. Virtual line descriptor and semi-local matching method for reliable feature correspondence. In *British Machine Vision Conference 2012*, pages 16–1, 2012.

[124] M. Lourakis and X. Zabulis. Model-based pose estimation for rigid objects. In *International Conference on Computer Vision Systems*, pages 83–92. Springer, 2013.

[125] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[126] S. Lynen, T. Sattler, M. Bosse, J. A. Hesch, M. Pollefeys, and R. Siegwart. Get out of my lab: Large-scale, real-time visual-inertial localization. In *Robotics: Science and Systems*, 2015.

[127] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry. *An invitation to 3-d vision: from images to geometric models*, volume 26. Springer Science & Business Media, 2012.

[128] A. Majdik, Y. Albers-Schoenberg, and D. Scaramuzza. Mav urban localization from google street view data. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 3979–3986, Nov 2013. doi: 10.1109/IROS.2013.6696925.

[129] A. L. Majdik, Y. Albers-Schoenberg, and D. Scaramuzza. Mav urban localization from google street view data. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 3979–3986. IEEE, 2013.

[130] D. Marr and E. Hildreth. Theory of edge detection. In *Proc. R. Soc. Lond. B*, volume 207, pages 187–217. The Royal Society, 1980.

[131] D. Massiceti, A. Krull, E. Brachmann, C. Rother, and P. H. Torr. Random forests versus neural networks—what's best for camera localization? In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 5118–5125. IEEE, 2017.

[132] M. Meilland. *Dense RGB-D mapping for real-time localisation and autonomous navigation.* Theses, Ecole Nationale Supérieure des Mines de Paris, Mar. 2012. URL `https://tel.archives-ouvertes.fr/tel-00686803`.

[133] M. Meilland, A. Comport, P. Rives, and I. S. A. Méditerranée. Real-time dense visual tracking under large lighting variations. In *British Machine Vision Conference, University of Dundee*, volume 29, 2011.

[134] F. Michel, A. Kirillov, E. Brachmann, A. Krull, S. Gumhold, B. Savchynskyy, and C. Rother. Global hypothesis generation for 6d object pose estimation. *arXiv preprint arXiv:1612.02287*, 2016.

[135] B. Micusik and J. Kosecka. Piecewise planar city 3d modeling from street view panoramic sequences. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2906–2912. IEEE, 2009.

[136] S. Middelberg, T. Sattler, O. Untzelmann, and L. Kobbelt. Scalable 6-dof localization on mobile devices. In *European conference on computer vision*, pages 268–283. Springer, 2014.

[137] K. Mikolajczyk and C. Schmid. An affine invariant interest point detector. In *European conference on computer vision*, pages 128–142. Springer, 2002.

[138] K. Mikolajczyk and C. Schmid. Comparison of affine-invariant local detectors and descriptors. In *Signal Processing Conference, 2004 12th European*, pages 1729–1732. IEEE, 2004.

[139] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE transactions on pattern analysis and machine intelligence*, 27(10):1615–1630, 2005.

[140] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool. A comparison of affine region detectors. *International journal of computer vision*, 65(1-2):43–72, 2005.

[141] D. Mishkin, J. Matas, and M. Perdoch. Mods: Fast and robust method for two-view matching. *Computer Vision and Image Understanding*, 141:81–93, 2015.

[142] L. Moisan, P. Moulon, and P. Monasse. Automatic homographic registration of a pair of images, with a contrario elimination of outliers. *Image Processing On Line*, 2:56–73, 2012.

[143] Y. Y. Morvan. *Acquisition, compression and rendering of depth and texture for multi-view video*. PhD thesis, Technische Universiteit Eindhoven, 2009.

[144] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2(331-340):2, 2009.

[145] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.

[146] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5): 1147–1163, 2015.

[147] L. Neumann and J. Matas. Real-time scene text localization and recognition. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3538–3545. IEEE, 2012.

[148] D. Nistér, O. Naroditsky, and J. Bergen. Visual odometry. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–652. IEEE, 2004.

[149] M. P. Parsley and S. J. Julier. Towards the exploitation of prior information in slam. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2991–2996. IEEE, 2010.

[150] S. D. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. H. Eng, D. Rus, and M. H. Ang. Perception, planning, control, and coordination for autonomous vehicles. *Machines*, 5(1):6, 2017.

[151] T. Peynot, S. Scheding, and S. Terho. The marulan data sets: Multi-sensor perception in a natural environment with challenging conditions. *The International Journal of Robotics Research*, 29(13):1602–1607, 2010.

[152] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[153] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[154] H. Prashanth, H. Shashidhara, and B. M. KN. Image scaling comparison using universal image quality index. In *Advances in Computing, Control, & Telecommunication Technologies, 2009. ACT'09. International Conference on*, pages 859–863. IEEE, 2009.

[155] K. Prazdny. Egomotion and relative depth map from optical flow. *Biological cybernetics*, 36(2):87–102, 1980.

[156] X. Qu, B. Soheilian, and N. Paparoditis. Vehicle localization using mono-camera and geo-referenced traffic signs. In *Intelligent Vehicles Symposium (IV), 2015 IEEE*, pages 605–610. IEEE, 2015.

[157] F. Radenović, G. Tolias, and O. Chum. Cnn image retrieval learns from bow: Unsupervised fine-tuning with hard examples. In *European Conference on Computer Vision*, pages 3–20. Springer, 2016.

[158] N. Radwan, G. D. Tipaldi, L. Spinello, and W. Burgard. Do you see the bakery? leveraging geo-referenced texts for global localization in public maps. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 4837–4842. IEEE, 2016.

[159] R. Raguram, O. Chum, M. Pollefeys, J. Matas, and J.-M. Frahm. Usac: a universal framework for random sample consensus. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):2022–2038, 2013.

[160] F. Ramm, J. Topf, and S. Chilton. *OpenStreetMap: using and enhancing the free map of the world*. UIT Cambridge Cambridge, 2011.

[161] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1508–1515. IEEE, 2005.

[162] E. Rosten, R. Porter, and T. Drummond. Faster and better: A machine learning approach to corner detection. *IEEE transactions on pattern analysis and machine intelligence*, 32(1):105–119, 2010.

[163] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE, 2011.

[164] A. G. Rundle, M. D. Bader, C. A. Richards, K. M. Neckerman, and J. O. Teitler. Using google street view to audit neighborhood environments. *American journal of preventive medicine*, 40(1):94–100, 2011.

[165] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek. Image classification with the fisher vector: Theory and practice. *International journal of computer vision*, 105(3):222–245, 2013.

[166] T. Sattler, B. Leibe, and L. Kobbelt. Scramsac: Improving ransac's efficiency with a spatial consistency filter. In *Computer vision, 2009 ieee 12th international conference on*, pages 2090–2097. IEEE, 2009.

[167] T. Sattler, B. Leibe, and L. Kobbelt. Fast image-based localization using direct 2d-to-3d matching. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 667–674. IEEE, 2011.

[168] T. Sattler, B. Leibe, and L. Kobbelt. Improving image-based localization by active correspondence search. In *European conference on computer vision*, pages 752–765. Springer, 2012.

[169] T. Sattler, T. Weyand, B. Leibe, and L. Kobbelt. Image retrieval for image-based localization revisited. In *BMVC*, volume 1, page 4, 2012.

[170] T. Sattler, C. Sweeney, and M. Pollefeys. On sampling focal length values to solve the absolute pose problem. In *European Conference on Computer Vision*, pages 828–843. Springer, 2014.

[171] T. Sattler, M. Havlena, F. Radenovic, K. Schindler, and M. Pollefeys. Hyperpoints and fine vocabularies for large-scale location recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2102–2110, 2015.

[172] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1-3):7–42, 2002.

[173] T. Schaul, I. Antonoglou, and D. Silver. Unit tests for stochastic optimization. *arXiv preprint arXiv:1312.6055*, 2013.

[174] G. Schindler, M. Brown, and R. Szeliski. City-scale location recognition. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–7. IEEE, 2007.

[175] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[176] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 806–813, 2014.

[177] J. Shotton, B. Glocker, C. Zach, S. Izadi, A. Criminisi, and A. Fitzgibbon. Scene coordinate regression forests for camera relocalization in rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2930–2937, 2013.

[178] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[179] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *null*, page 1470. IEEE, 2003.

[180] I. Skrypnyk and D. G. Lowe. Scene modelling, recognition and tracking with invariant image features. In *Mixed and Augmented Reality, 2004. ISMAR 2004. Third IEEE and ACM International Symposium on*, pages 110–119. IEEE, 2004.

[181] C. C. Slama, C. Theurer, and S. W. Henriksen. *Manual of photogrammetry*. American Society of photogrammetry, 1980.

[182] M. Smith, I. Baldwin, W. Churchill, R. Paul, and P. Newman. The new college vision and laser data set. *The International Journal of Robotics Research*, 28 (5):595–599, 2009.

[183] R. C. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, 5(4):56–68, 1986.

[184] S. M. Smith and J. M. Brady. Susan—a new approach to low level image processing. *International journal of computer vision*, 23(1):45–78, 1997.

[185] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[186] J. Stillwell. *Naive lie theory*. Springer, 2008.

[187] N. Sunderhauf, S. Shirazi, A. Jacobson, F. Dayoub, E. Pepperell, B. Upcroft, and M. Milford. Place recognition with convnet landmarks: Viewpoint-robust, condition-robust, training-free. *Proceedings of Robotics: Science and Systems XII*, 2015.

[188] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[189] L. Svarm, O. Enqvist, M. Oskarsson, and F. Kahl. Accurate localization and pose estimation for large 3d models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 532–539, 2014.

[190] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[191] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[192] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, pages 4278–4284, 2017.

[193] A. Taneja, L. Ballan, and M. Pollefeys. Never get lost again: Vision based navigation using streetview images. In *Asian Conference on Computer Vision*, pages 99–114. Springer, 2014.

[194] S. Thrun and J. J. Leonard. Simultaneous localization and mapping. In *Springer handbook of robotics*, pages 871–889. Springer, 2008.

[195] S. Thrun, W. Burgard, and D. Fox. *Probabilistic robotics*. MIT press, 2005.

[196] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23(9):661–692, 2006.

[197] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[198] G. Tolias, R. Sicre, and H. Jégou. Particular object retrieval with integral max-pooling of cnn activations. *arXiv preprint arXiv:1511.05879*, 2015.

[199] R. F. Tomlinson. *Thinking about GIS: geographic information system planning for managers*. ESRI, Inc., 2007.

[200] A. Torii, M. Havlena, et al. From google street view to 3d city models. In *Computer vision workshops (ICCV Workshops), 2009 IEEE 12th international conference on*, pages 2188–2195. IEEE, 2009.

[201] A. Torii, J. Sivic, and T. Pajdla. Visual localization by linear combination of image descriptors. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 102–109. IEEE, 2011.

[202] A. Torii, J. Sivic, T. Pajdla, and M. Okutomi. Visual place recognition with repetitive structures. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 883–890, 2013.

[203] A. Torii, R. Arandjelović, J. Sivic, M. Okutomi, and T. Pajdla. 24/7 place recognition by view synthesis. In *CVPR*, 2015.

[204] P. H. Torr and A. Zisserman. Mlesac: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78(1):138–156, 2000.

[205] L. Torresani, M. Szummer, and A. Fitzgibbon. Efficient object category recognition using classemes. *Computer Vision–ECCV 2010*, pages 776–789, 2010.

[206] G. Vaca-Castano, A. R. Zamir, and M. Shah. City scale geo-spatial trajectory estimation of a moving camera. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1186–1193. IEEE, 2012.

[207] A. Vedaldi and B. Fulkerson. Vlfeat: An open and portable library of computer vision algorithms. In *Proceedings of the 18th ACM international conference on Multimedia*, pages 1469–1472. ACM, 2010.

[208] K. Wagstaff, C. Cardie, S. Rogers, S. Schrödl, et al. Constrained k-means clustering with background knowledge. In *ICML*, volume 1, pages 577–584, 2001.

[209] J. Wang and L. Perez. The effectiveness of data augmentation in image classification using deep learning. Technical report, Technical report, 2017.

[210] Z. Wang, B. Fan, and F. Wu. Local intensity order pattern for feature description. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 603–610. IEEE, 2011.

[211] J. Weng, P. Cohen, M. Herniou, et al. Camera calibration with distortion models and accuracy evaluation. *IEEE Transactions on pattern analysis and machine intelligence*, 14(10):965–980, 1992.

[212] T. Weyand, I. Kostrikov, and J. Philbin. Planet-photo geolocation with convolutional neural networks. In *European Conference on Computer Vision*, pages 37–55. Springer, 2016.

[213] M. G. Wing, A. Eklund, and L. D. Kellogg. Consumer-grade global positioning system (gps) accuracy and reliability. *Journal of forestry*, 103(4):169–173, 2005.

[214] Z. Wu, C. Shen, and A. v. d. Hengel. Wider or deeper: Revisiting the resnet model for visual recognition. *arXiv preprint arXiv:1611.10080*, 2016.

[215] J. Xiao and L. Quan. Multiple view semantic segmentation for street view images. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 686–693. IEEE, 2009.

[216] J. Xie, M. Kiefel, M.-T. Sun, and A. Geiger. Semantic instance annotation of street scenes by 3d to 2d label transfer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3688–3697, 2016.

[217] G. Xu and Z. Zhang. *Epipolar geometry in stereo, motion and object recognition: a unified approach*, volume 6. Springer Science & Business Media, 2013.

[218] T. Yeh, K. Tollmar, and T. Darrell. Searching the web with mobile images for location recognition. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–76. IEEE, 2004.

[219] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[220] L. Yu, C. Joly, G. Bresson, and F. Moutarde. Improving robustness of monocular urban localization using augmented street view. In *Intelligent Transportation Systems (ITSC), 2016 IEEE 19th International Conference on*, pages 513–519. IEEE, 2016.

[221] L. Yu, C. Joly, G. Bresson, and F. Moutarde. Monocular urban localization using street view. *arXiv preprint arXiv:1605.05157*, 2016.

[222] J. Yue-Hei Ng, F. Yang, and L. S. Davis. Exploiting local features from deep networks for image retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 53–61, 2015.

[223] A. R. Zamir and M. Shah. Accurate image localization based on google maps street view. In *European Conference on Computer Vision*, pages 255–268. Springer, 2010.

[224] A. R. Zamir, A. Hakeem, L. Van Gool, M. Shah, and R. Szeliski. *Large-scale visual geo-localization*. Springer, 2016.

[225] P. A. Zandbergen. Accuracy of iphone locations: A comparison of assisted gps, wifi and cellular positioning. *Transactions in GIS*, 13(s1):5–25, 2009.

[226] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[227] W. Zhang and J. Kosecka. Image based localization in urban environments. In *3D Data Processing, Visualization, and Transmission, Third International Symposium on*, pages 33–40. IEEE, 2006.

[228] Z. Zhang. Parameter estimation techniques: A tutorial with application to conic fitting. *Image and vision Computing*, 15(1):59–76, 1997.

[229] L. Zhao, S. Huang, L. Yan, and G. Dissanayake. Parallax angle parametrization for monocular slam. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3117–3124. IEEE, 2011.

[230] L. Zheng, Y. Zhao, S. Wang, J. Wang, and Q. Tian. Good practice in cnn feature transfer. *arXiv preprint arXiv:1604.00133*, 2016.

[231] L. Zheng, Y. Yang, and Q. Tian. Sift meets cnn: A decade survey of instance retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[232] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. In *Advances in neural information processing systems*, pages 487–495, 2014.

[233] D. Zou and P. Tan. Coslam: Collaborative visual slam in dynamic environments. *IEEE transactions on pattern analysis and machine intelligence*, 35 (2):354–366, 2013.

Résumé

Dans un travail réalisé au Centre de Robotique et à l'Institut VEDECOM, nous nous sommes intéressés aux systèmes robustes de localisation visuelle en milieu urbain pour la voiture autonome. Obtenir une pose exacte à partir d'une caméra monoculaire est difficile et insuffisant en terme de précision pour la voiture autonome actuelle. Plutôt que d'utiliser des approches comme la navigation par satellites, la Cartographie et Localisation Simultanées (SLAM), et les techniques de fusion de données, nous nous sommes concentrés sur l'utilisation de Systèmes d'Information Géographiques (SIG) pour concevoir une approche fiable, précise et absolue de localisation en milieu urbain.

Le développement de SIG publics nous a apporté un nouvel horizon pour résoudre le problème de la localisation, mais ses informations, telles que les cartes topologiques, sémantiques, métriques, les Street Views, les cartes de profondeur, les cartes cadastrales 3D et les cartes en haute définition, doivent être bien analysées et organisées pour extraire les informations pertinentes pour une voiture autonome. Notre première tâche consistait à concevoir une base de données hors ligne accessible par un robot à partir d'un SIG public dense, à savoir Google Maps, qui a l'avantage d'avoir une couverture mondiale. Nous générons une représentation topométrique compacte de l'environnement urbain dynamique en extrayant quatre données utiles du SIG, y compris : les topologies, les géo-coordonnées, les Street Views panoramiques et les cartes de profondeur associées. Dans le même temps, un ensemble de données en ligne a été acquis par une caméra à bas prix équipée sur les véhicules de VEDECOM. Afin de rendre les Street View sphériques compatibles avec l'imagerie en ligne, une transformation basée sur l'interpolation d'image est introduite pour obtenir des images rectilignes à partir de Street Views.

Nous proposons deux méthodes de localisation : l'une est une approche de vision par ordinateur basée sur l'extraction de caractéristiques, l'autre est une méthode d'apprentissage basée sur les réseaux de neurones convolutionnels (convnet). En vision par ordinateur, l'extraction de caractéristiques est un moyen populaire de résoudre le positionnement à partir d'images. Nous tirons parti de Google Maps et utilisons ses données topo-métriques hors ligne pour construire un positionnement grossier à fin, à savoir un processus de reconnaissance de lieu topologique puis une estimation métrique de pose par optimisation de graphe. La seule entrée de cet algorithme est une séquence d'images provenant d'une caméra monoculaire et la base de données construite à partir de Google Maps. De plus, il n'est pas nécessaire d'établir des correspondances d'image à image, ni d'utiliser l'odométrie. La méthode a été testée en environnement urbain et démontre à la fois une précision sous-métrique et une robustesse aux changements de point de vue, à l'illumination et à l'occlusion. Aussi, les résultats montrent que les emplacements éloignés de Street Views produisent une erreur significative dans la phase d'estimation métrique. Ainsi, nous proposons de synthétiser des Street Views artificielles pour compenser la densité des Street View originales et améliorer la précision.

Cette méthode souffre malheureusement d'un temps de calcul important. Étant donné que le SIG nous offre une base de données géolocalisée à l'échelle mondiale, cela nous motive à régresser des localisations globales directement à partir d'un convnet de bout en bout. La base de données hors ligne précédemment construite est encore insuffisante pour l'apprentissage d'un convnet. Pour compenser cela nous densifions la base d'origine d'un facteur mille et utilisons la méthode d'apprentissage par transfert pour faire converger notre régresseur convnet et avoir une bonne performance. Le régresseur permet également d'obtenir une localisation globale à partir d'une seule image et en temps réel.

Les résultats obtenus par ces deux approches nous fournissent des informations sur la comparaison et la relation entre les méthodes basées sur des caractéristiques et celles basées sur le convnet. Après avoir analysé et comparé les performances de localisation des deux méthodes, nous avons également abordé des perspectives pour améliorer la robustesse et la précision de la localisation face au problème de localisation urbaine assistée par SIG.

Mots Clés

Localisation absolue, Street View, GIS, Handcrafted features, Convnet, Mono-caméra

Abstract

In a work made at Centre de Robotique and Institut VEDECOM, we studied robust visual urban localization systems for self-driving cars. Obtaining an exact pose from a monocular camera is difficult and cannot be applied to the current autonomous cars. Rather than using approaches like Global Navigation Satellite Systems, Simultaneous Localization And Mapping, and data fusion techniques, we mainly focused on fully leveraging Geographical Information Systems (GIS) to achieve a low-cost, robust, accurate and global urban localization.

The development of public GIS's has brought us a new horizon to address the localization problem but their tremendous amount of information, such as topological, semantic, metric maps, Street Views, depth maps, 3D cadastral maps and High Definition maps, has to be well analyzed and organized to extract relevant information for self-driving cars. Our first task was to design a robotic accessible offline database from a dense public GIS, namely Google Maps, which has the advantage to propose a worldwide coverage. We make a compact topometric representation for the dynamic urban environment by extracting four useful data from the GIS, including topologies, geo-coordinates, panoramic Street Views, and associated depth maps. At the same time, an online dataset was acquired with a low-cost camera equipped on VEDECOM vehicles. In order to make spheric Street Views compatible with the online imagery, an image warping and interpolation based transformation is introduced to render rectilinear images from Street Views.

We proposed two localization methods: one is a handcrafted features based computer vision approach, the other is a convolutional neural network (convnet) based learning technique. In computer vision, extracting handcrafted features is a popular way to solve the image based positioning. We take advantages of the abundant sources from Google Maps and benefit from the topo-metric offline data structure to build a coarse-to-fine positioning, namely a topological place recognition process and then a metric pose estimation by a graph optimization. The only input of this approach is an image sequence from a monocular camera and the database constructed from Google Maps. Moreover, it is not necessary to establish frame to frame correspondences, nor odometry estimates. The method is tested on an urban environment and demonstrates both sub-meter accuracy and robustness to viewpoint changes, illumination and occlusion. Moreover, we demonstrate that sparse Street View locations produce a significant error in the metric pose estimation phase. Thus our former framework is refined by synthesizing more artificial Street Views to compensate the sparsity of original Street Views and improve the precision.

The handcrafted feature based framework requires the image retrieval and graph optimization. It is hard to achieve in a real-time application. Since the GIS offers us a global scale geotagged database, it motivates us to regress global localizations from convnet features in an end-to-end manner. The previously constructed offline database is still insufficient for a convnet training. We hereby augment the originally constructed database by a thousand factor and take advantage of the transfer learning method to make our convnet regressor converge and have a good performance. In our test, the regressor can also give a global localization of an input camera image in real time.

The results obtained by the two approaches provide us insights on the comparison and connection between handcrafted feature-based and convnet based methods. After analyzing and comparing the localization performances of both methods, we also talked about some perspectives to improve the localization robustness and precision towards the GIS-aided urban localization problem.

Keywords

Absolute localization, Street Views, GiS, Handcrafted features, Convnet, Mono-camera