Anil Armagan

Image-Based Camera Localization by Leveraging Semantic Information in Urban Environments

DOCTORAL THESIS

to achieve the university degree of
Doktor der technischen Wissenschaften

submitted to

Graz University of Technology

Supervisor

Prof. Dr. Vincent Lepetit
Institute of Computer Graphics and Vision

Graz, Austria, September 2018

Abstract

Accurate camera localization has been key to the success of many applications in different domains, such as augmented reality (AR) and robotics. The problem of predicting the position and orientation of a camera is referred as "localization task". The localization problem needs to be solved wisely where the accurate camera pose should be known in advance, such as for autonomous driving or AR.

Several methods have been proposed to solve the localization problem by using inputs from different sources of modalities such as monocular images, infrared projectors, laser scanners or depth sensors. Using such sensors enables researchers to produce useful representations like ground-level images, aerial images, elevation models or textured 3D models. However, collection of large image datasets or using sensors such as laser scanners are costly and cumbersome to keep them up-to-date to represent the recent changes of the environment. In this thesis, we explore the localization problem by leveraging crowd-sourced and easy to maintain OpenStreetMap (OSM) models with recent advances in semantic segmentation. We focus on accurate localization in urban environments using single images and an initial pose of the camera estimated by simple available sensors, such as a compass and a Global Positioning System (GPS).

This thesis first introduces the localization problem in details and discusses its challenges and applications. Then, we give a literature survey on image-based camera localization where the previous works are mostly dominated by the methods using registered sets of image collections. However, recent developments in deep learning and their success bring new perspectives to the camera localization problem. Image classification and matching based solutions mostly focus on hand-crafted features where occlusions, illumination changes and texture-less areas are usually a difficult problem to cope with. On the other hand, this thesis explores the use of semantic information in urban environments with three frameworks.

For accurate camera localization, we first show how to use recent advances in image segmentation and machine learning to segment buildings and their parts in urban. Then,

we discuss how to use a 3D tracking system to acquire the data required for training the segmentation method. Our first method shows that the semantic information from the buildings in the urban environments is representative enough for accurate localization. To localize an input image, we explore the usage of simple 2.5D maps of buildings and the semantic segmentation. Given an initial pose estimate from the sensors, we first extract the façades and edges of the buildings from the image, and then look for the orientation and location that align rendering of the map with these segments by sampling the pose space around the sensors' estimates. We show that with the exploited semantic information, we can make better pose estimations then the estimates from the sensors. However, the sensor errors tend to be large in the wild and since the pose space is huge, we need a better optimization of the pose search space to have more efficient methods in practice.

Furthermore, this thesis explores the use of the semantic information and 2.5D maps in a learning-based iterative framework for a better pose optimization. Our key contribution in this part is a novel, efficient and robust method to optimize the pose: We train two deep networks to predict the best direction to improve a pose estimate, given a semantic segmentation of the input image and a rendering of the buildings from this estimate. We then iteratively apply the CNNs until converging to a good pose. Our networks can estimate the pose efficiently when the errors from the sensor's are relatively small. In practice, since the sensor error might be very large, we sparsely sample initial poses around the sensors' pose estimates and start the optimization for each initial pose by applying the networks in an iterative fashion. The proposed approach is more accurate and more efficient than the approach described previously which is fully based on uniform sampling of the poses. Since our methods are not using reference images, it can be applied to places unseen during training.

Finally, we propose an approach to bridge the gap between learning-based approaches and geometric approaches. We achieve this by extracting high-level features buildings in the input images and well established minimal solvers. Our approach uses the same semantic information from the buildings and additionally, we exploit the façades' normals in the images. Then, we introduce two minimal solvers to establish a 2D pose given the extracted features and 2.5D map of the environment. The minimal solvers we propose are able to compute the camera pose accurately and robustly. We propose two such minimal solvers: one based on three correspondences of buildings' corners from the image and the 2.5D map and another one based on two corner correspondences plus one façade correspondence. The advantage of our approach is the use of geometric features with robust minimal solvers. Our experiments show that the approach is much more efficient since it relies on more robust features as the high-level features.

Keywords. camera registration, image-based localization, camera relocalization, pose estimation, visual localization.

Kurzfassung

Genaue Lokalisierung der Kameras war der Schlüssel zum Erfolg vieler Anwendungen in verschiedenen Bereichen wie Augmented Reality (AR) und Robotik. Das Problem der Vorhersage der Position und Orientierung einer Kamera wird als "Lokalisierungsaufgabe" bezeichnet. Das Problem bei der Lokalisierung muss mit Bedacht gelöst werden. Die genaue Pose der Kamera sollte wie etwa für autonomes Fahren oder AR im Voraus bekannt sein.

Mehrere Verfahren wurden vorgeschlagen, um das Lokalisierungsproblem zu lösen. Dabei wurden Eingaben von verschiedenen Quellen von Modalitäten wie monokularen Bildern, Infrarotprojektoren, Laserscannern oder Tiefensensoren verwendet. Mithilfe solcher Sensoren können Forscher nützliche Darstellungen wie Bodenbilder, Luftbilder, Höhenmodelle oder texturierte 3D-Modelle erstellen. Die Sammlung großer Bilddatensätze oder die Verwendung von Sensoren wie Laserscannern sind jedoch kostspielig und umständlich, um sie auf dem neuesten Stand zu halten und die jüngsten Änderungen der Umgebung darzustellen. In dieser Arbeit untersuchen wir das Lokalisierungsproblem, indem wir crowdsourcingfähige OpenStreetMap (OSM)-Modelle mit aktuellen Fortschritten in der semantischen Segmentierung nutzen. Wir konzentrieren uns auf die genaue Lokalisierung in den städtischen Umgebungen mit Hilfe von Einzelbildern und einer ersten Pose der Kamera, die durch einfache verfügbare Sensoren, wie einen Kompass und ein GPS (Global Positioning System) geschätzt wird.

Diese Arbeit stellt zunächst das Lokalisierungsproblem im Detail vor und diskutiert deren Herausforderungen und Anwendungen. Anschließend geben wir eine Literaturstudie zur bildbasierten Kameralokalisierung, bei der die bisherigen Arbeiten überwiegend von den Methoden mit registrierten Bildsammlungen dominiert werden. Die jüngsten Entwicklungen im Bereich des Tiefgehendes Lernen und dessen Erfolg bringen jedoch neue Perspektiven für das Kameraproblem. Bildklassifizierungs- und -abgleichbasierte Lösungen konzentrieren sich hauptsächlich auf handgefertigte Merkmale, bei denen Okklusionen,

Beleuchtungsänderungen und strukturlose Bereiche üblicherweise ein schwieriges Problem darstellen. Andererseits untersucht diese Arbeit die Verwendung semantischer Informationen in urbanen Umgebungen mit drei Frameworks.

Für eine genaue Kameralokalisierung zeigen wir zunächst, wie Segmentierung und maschinelles Lernen zu verwenden ist. Dann besprechen wir, wie ein 3D-Tracking-System verwendet werden kann, um die für die Segmentierungsmethode erforderlichen Daten zu erfassen. Unsere erste Methode zeigt, dass die semantischen Informationen aus den Gebäuden in städtischen Umgebungen repräsentativ genug für eine genaue Lokalisierung sind. Um ein Eingabebild zu lokalisieren, untersuchen wir die Verwendung von einfachen 2.5D-Karten von Gebäuden und der semantischen Segmentierung. Bei einer anfänglichen Pose-Schätzung von den Sensoren können wir zunächst die Fassaden und Kanten der Gebäude auf den Bildern extrahieren. Anschliessend suchen wir nach der Ausrichtung und Position, die das Rendern der Karte mit diesen Segmenten angleicht, indem der Pose-Platz um die Schätzungen der Sensoren herum abgetastet wird. Wir zeigen, dass wir mit den ausgenutzten semantischen Informationen bessere Pose-Schätzungen machen können. Allerdings sind die Sensorfehler in der Regel groß und der Pose-Platz ist riesig. Daher brauchen wir eine bessere Optimierung des Suchraums für die Pose, so können wir effizientere Methoden in der Praxis erhalten.

Des Weiteren untersucht diese Arbeit die Verwendung der semantischen Informationen und 2.5D-Karten in einem lernbasierten iterativen Rahmen für eine bessere Optimierung der Pose. Unser Hauptbeitrag in diesem Teil ist eine neuartige, effiziente und robuste Methode, um die Pose zu optimieren. Wir trainieren zwei tiefe Netzwerke, um die beste Richtung für eine bessere Schätzung einer Pose zu bestimmen und geben bei einer semantischen Segmentierung des Eingangsbildes und einer Darstellung der Gebäude diese Schätzung. Wir wenden dann die CNNs iterativ an, bis sie zu einer guten Pose konvergieren. Unsere Netzwerke können die Pose effizient schätzen, wenn die Fehler vom Sensor relativ klein sind. Da der Sensorfehler sehr groß sein kann, nehmen wir in der Praxis die anfänglichen Posen um die Pose-Schätzungen der Sensoren herum und starten die Optimierung für jede Anfangspose, indem wir die Netzwerke iterativ anwenden. Der vorgeschlagene Ansatz ist genauer und effizienter als der zuvor beschriebene Ansatz. Da unsere Methoden nicht für Referenzbilder verwendet werden, können sie auf Orte angewendet werden, die während des Trainings nicht sichtbar sind.

Abschließend schlagen wir einen Ansatz vor, um die Lücke zwischen lernbasierten und geometrischen Ansätzen zu schließen. Dies erreichen wir durch das Extrahieren von High-Level-Features-Gebäuden in den Eingabebildern und gut etablierten Minimallösern. Unser Ansatz verwendet die gleichen semantischen Informationen aus den Gebäuden und zusätzlich nutzen wir die Normalen der Fassaden in den Bildern. Dann führen wir zwei Minimallöser ein, um eine 2D-Pose aufgrund der extrahierten Features und der 2.5D-Karte der Umgebung zu erstellen. Die von uns vorgeschlagenen Minimallöser sind in der Lage, die Kamerapose präzise und robust zu berechnen. Wir schlagen zwei solche Minimallöser vor: einen, der auf drei Übereinstimmungen der Gebäudeecken aus dem Bild

und der 2.5D-Karte basiert und einen zweiten, der auf zwei Korrespondenzecken plus einer Fassadenkorrespondenz basiert. Der Vorteil unseres Ansatzes ist die Verwendung geometrischer Merkmale mit robusten Minimallösern. Unsere Experimente zeigen, dass der Ansatz viel effizienter ist, da er auf höheren Funktionen beruht.

Schlüsselwörter. Kameraregistrierung, bildbasierte Lokalisierung, Kamera Relokalisierung, Pose Schätzung, visuelle Lokalisierung.

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

The text document uploaded to TUGRAZonline is identical to the present doctoral thesis.

Date	Signature

Acknowledgments

First and foremost, I owe my deepest gratitude to my supervisor, Prof. Vincent Lepetit, for his priceless guidance, encouragement, motivation and support throughout the beginning of my work at the Institute of Computer Graphics and Vision.

Special thanks to Assoc. Prof Tae-Kyun Kim and Assoc. Prof. Denis Helic for kindly accepting to be in my committee. I owe them my appreciation for their support and helpful suggestions.

Similar to a chair, we are not complete as long the legs of the chair is complete. For this reason, I would like to present my deepest thank to Carmen, Özlü and Mahdi. None of this would have been possible without their love and support. I am tremendously grateful for all the selflessness and the sacrifices you have made for me.

Finally, I would like to thank all members of CVARLab and the Institute of Computer Graphics and Vision for giving me the chance to be a member of their groups.

This thesis is supported by the Christian Doppler Laboratory for Semantic 3D Computer Vision in collaboration with Qualcomm.

Contents

List of Figures

List of Tables

1 Introduction

Contents

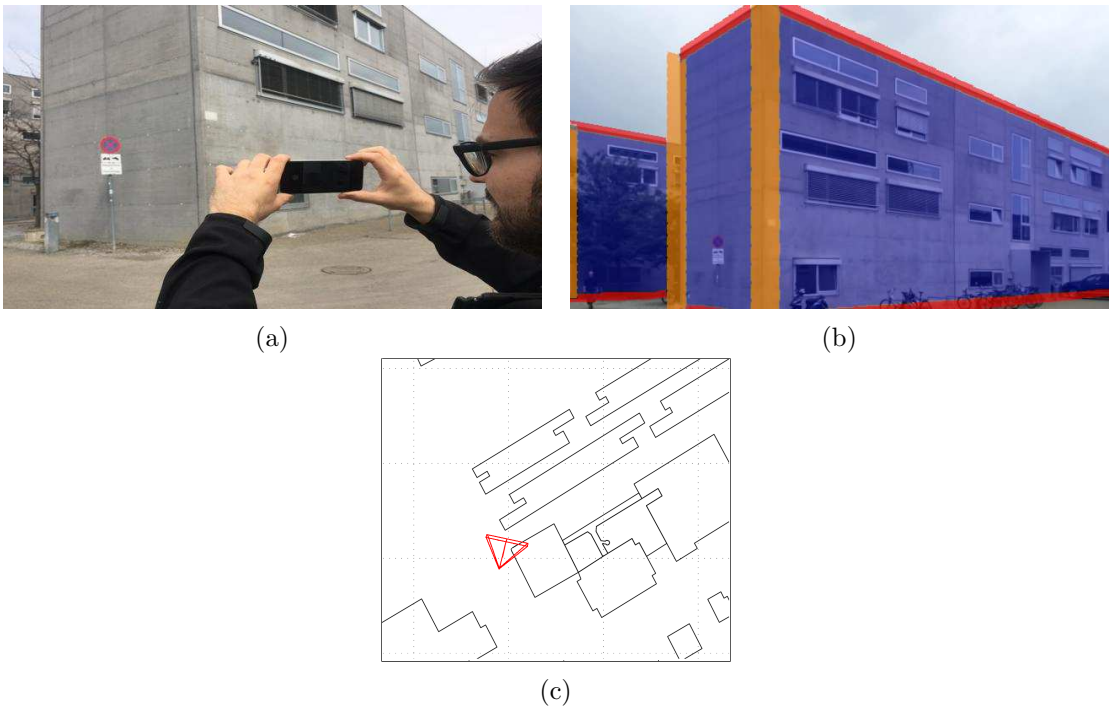
Humans have the ability to successfully localize themselves or other objects in an environment. For example, as soon as we enter a room or we go out of our flat, we build an understanding of the scene, its components and most importantly we know where we are. We have a great visual perception system and most of the times, it is not hard for humans to create a rough mapping of the scene and roughly estimate where each object stands or how they are oriented. It is even easily possible to make localization without depending on if the environment was observed before or not. However, when it comes to machines, it is not straightforward to make them able to localize themselves.

Machines have now the ability to globally localize themselves thanks to the introduction of Global Navigation Satellite Systems (GNSS), where the United States' Global Positioning System (GPS) is an example of GNSS. A GNSS is great to make earth level localization, it uses satellites to localize the device. Together with other sensors such as compass, a machine can estimate its position and orientation. However, their accuracy is not as good as human-level localization system in most cases. Since, such systems can be easily affected by external factors, it makes them inaccurate for localization purposes either in urban or indoor scenarios. For example, a GPS system works less accurate if we are on the street where many cars and buildings surround us. The error of the system might be as high as 30 meters. Similarly, a compass is effected by external metallic structures such as cars and it can make even have errors as high as 40°. Also, a GPS might

not be available all the time either because of signal denied environments or simply the GPS is not available. Therefore, other sources of information such as images are used to improve or make localization possible for more devices.

Image-based localization is a broad topic which has taken attention of different communities e.g. Computer Vision (CV) or Robotics for many years. It has been studied using different methodologies and technologies. Recent advances, including advances in hardware, enabled researches to develop more convenient methods for localization. These advances include equipping devices with cameras that provide us with fine details of a scene, GNSS and variety of sensors e.g. Lidars, compasses or accelerometers. CV, Robotics and Machine Learning (ML) researchers have been developing more efficient and effective algorithms to make devices smarter. Localization techniques started to take more and more attention with the introduction of new sensors and ease of access to them with devices such as our smartphones or tablets that are used in daily life today.

The advances in smart devices together with new research studies have resulted in a wide range of applications in many different fields e.g. logistics, gaming or sports. In 20th century, these applications were considered as science fiction, however; nowadays we are living in this era which was once considered as fiction because of level of autonomy that



(a) (b)

(c)

Figure 1.1: Illustration of image-based camera localization. **(a)** the user captures an input image. **(b)** Once the pose of the camera is estimated, it is possible for example to overlay a 3D model of the environment with the input image. **(c)** The estimated camera pose is shown in red in a 2D map of the environment.

machines have been reached.

In this thesis, we explore the existing methods for image-based localization and propose novel methods to improve upon them. Our approaches make use of object classes that are commonly seen in urban areas to make localization. We leverage the object classes such as buildings in different frameworks to localize a camera that is used to capture an image in urban city environments.

1.1 Problem Statement

Localization, very generally, is a term used for the process of making something e.g. an object, a robot, a satellite, a GSM transmitter, a camera local or estimate its position to a particular place. For example, the problem of estimating the position and orientation of a camera into a global or local coordinates is referred as localization. In this thesis, image-based localization or image-based camera localization techniques where the device is equipped with a camera are considered.

Camera registration or *camera localization* where the task is registering the camera which relates to knowing the accurate position and the orientation of the camera in a relative coordinate system. The coordinate system could be either relative to a map of a known environment or the whole world.

The term *'camera re-localization'* is also frequently seen in the context of image-based localization. Camera re-localization refers to the cases where an approximate estimate of the device's position and orientation or both is known but the task is to re-localize the device to have a better pose estimate. Camera re-localization is usually called when estimate of the camera pose is known with a GPS.

Another term commonly used in the context of image-based localization is *geolocalization* which is mostly referred for computing only the position, not the orientation, of the camera. Therefore, in the rest of the thesis, image-based geolocalization will be referred as a subset of the image-based localization task where only the position is estimated. Plenty of previous studies address the problem as a geolocalization problem, however, some applications like Augmented Reality (AR) require the full pose to be known accurately instead of only the position. The ultimate goal for localization is thus to estimate all degrees of freedom (DoF) that defines a pose. Therefore, this thesis proposes methods to make an estimate of full 6 DoF pose estimation of the camera.

A complete localization of a camera requires to estimate both position and orientation. In the three dimensional world, both position and orientation require three dimensions to be fully defined. Thus, a pose has six degrees of freedom (6 DoF). The 6 DoF pose represents the movements of the camera and these movements in 3D space are defined as follows:

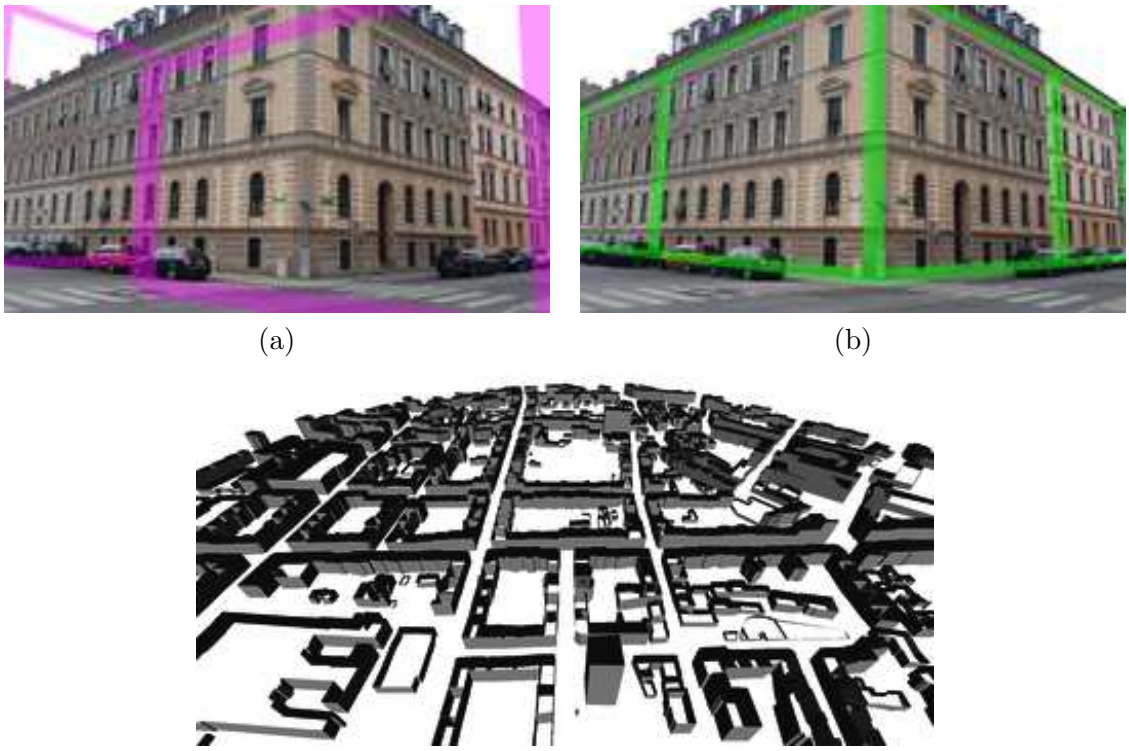


Figure 1.2: Illustration of a coarse pose estimate and an accurate pose estimate. With the help of sensors such as GPS, we can have an initial pose estimate of the camera and use it with the 3D model (c) to overlay the model on the input image. However, the projection of the 3D buildings does not match with the buildings in the image since the coarse estimate of the pose is not accurate (a). By estimating the camera pose accurately, we can successfully overlay the 3D buildings with the buildings in the image.

- **Translation** t**,** moving the camera along the three axes X, Y and Z.
  - t_x**,** moving up and down along the x axis is called heaving.
  - t_y**,** moving forwards and backwards along the y axis is called surging.
  - t_z**,** moving left and right along the z axis is called swaying.
- **Orientation** R**,** rotating the camera in order to face a different axis.
  - $R_x(\gamma)$**,** counterclockwise rotation between x axis is called *roll*.
  - $R_y(\beta)$**,** counterclockwise rotation about y axis is called *pitch*.
  - $R_z(\alpha)$**,** counterclockwise rotation between z axis is called *yaw*.

1.2 Applications

As recent challenges and benchmarks such as [16, 69] show, dealing with urban scenarios is of increasing interest. These challenges show that localization of a camera is necessary for different fields such as CV and Robotics. Even though the accuracy of GPS sensors is sufficient for navigation purposes where the task is directing a user to reach from point A to point B, the GPS information is not accurate enough for many other tasks.

Figure 1.3: An example of Augmented Reality (AR) for navigation purposes. AR is getting more attention with the recent advances in accurate camera pose estimation.

Today, cars have been given the ability to drive themselves. Knowing the accurate pose is crucial for a car to drive autonomously. This has been possible by getting help from many sensors such as inertial measurement unit (IMU), GPS, compass, Light Detection and Ranging (LIDAR) and cameras. However, autonomous driving in the areas like city centers are still challenging due to dynamic factors. It has been essential for the car to know about its environment. The cameras on a car displays high value information by letting the car see the real world to accurately localize itself. One of the keys to success here has been with the successful vision based approaches. An AR system focuses on combining real and virtual objects. The system provides the user with interactive information from the combination of the real world and the virtual world created by inserting virtual objects. The inserted information should visually align with the real world. In this context, correct visual alignment means accurate registration of the real and the virtual objects. To this end, image-based localization has been key to solve the problem.

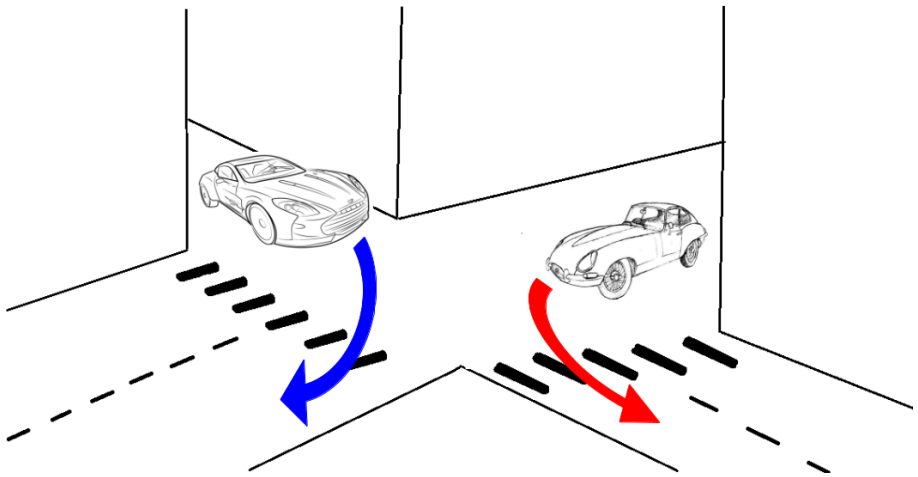


Figure 1.4: A self-driving car uses cameras or information from other sources such as Lidar to make an understanding of its environments and localize itself.

Accurate localization is also essential for robots or autonomous devices. Depending on the task the robot is designed for, in most of the cases it is important for the robot to know its accurate pose to achieve the task. Even if indoor localization techniques are out of scope of this thesis, it is worth to mention that localization techniques are applied even on cleaning robots such as iRobot Roomba 980[1].

1.3 Challenges

The localization problem occurs in both outdoor and indoor environments. However, requirements and challenges for each task are different. In this thesis, we focus on localization in urban environments.

When we consider a generic application for image-based localization, most of the users have access to a smart device such as a smartphone or tablet. However, even if smartphones are equipped with cameras that provide us with relevant information from the scenes, the field-of-view (FoV) of these cameras usually restricted around 60°. The narrow FoV angle of the cameras simply restricts us to get more information about the neighborhood for accurate localization. Panoramic images would be an option to overcome narrow FoV problem, however; most of the devices cannot capture panoramic images. Therefore, panoramic images are not easy to access in practice.

As smart devices are usually equipped with Global Positioning System (GPS), for the

[1] http://www.irobot.at/

outdoor localization task, a device searching for a route from point A to point B could be navigated by using GPS. However, GPS signals are easily affected by many artifacts e.g. long buildings that block GPS signals, metallic structures and cars passing by. For the indoor scenario GPS signal is usually completely blocked by the building, which are called GPS denied environments. Thus, dominant indoor localization methods work from cellular signals, WiFi signals or Bluetooth connectivity. For such methods, the origins of the WiFi access points or Bluetooth beacons should be known for accurate localization. However, this requires to maintain a large number of connection points for the signal transmitters. Therefore, GPS signals or signals from other transmitters are usually not accurate or limited for accurate localization and thus, image-based localization techniques have been developed in order to improve the localization estimates.

When we localize ourselves as humans, we mostly rely on visual cues to define and find our location. We make observations about our neighborhood e.g. man-made structures around us, distinctive buildings, traffic signs, street signs, door signs or logos. Combination of all these visual cues helps us to localize ourselves. This is one of the clues why image-based localization is necessary in the machinery level. However, defining such distinctive features that defines the scene and separates it from the other scenes is not an easy task. There has been many studies on discriminative feature descriptors [53, 88, 97] for many years. However, unless the data includes such discriminative features such as different building façades or logos, it has been a difficult problem to distinguish such man-made structures due to the similarity of them.

Since there has been a drastic increase in the usage of smart devices, the number of images uploaded on the World Wide Web (WWW) has been increased exponentially. The amount of geo-tagged data find on the web is massive. Therefore, another challenge in geolocalization is leveraging large scale geo-referenced image datasets and query processing to localize the query image against large scale pre-registered datasets. Another challenge of pre-registered data collections is that being coarsely tagged. Datasets are usually tagged with only their geographical locations. Therefore, using such collections has limited applications when orientation information is needed as well. It is also hard to maintain such collections to be up to-date.

Detailed 3D models of the environment represent valuable information such as texture for localization. 3D models can be provided with the help of sensors such as laser scanners or with the 3D reconstruction methods using multiple-view of the environment, for example from pre-registered image collections. However, similar to pre-registered images, detailed 3D models are also hard to maintain since the updating the models are costly.

A general problem in computer vision is also valid for image-based localization. Images contain unwanted effects such as blurring, image artifacts, lens distortions or lightning exposures. It is necessary to overcome such artifacts.

Last but not least in urban scenes consists of dynamic objects that cause occlusions e.g. cars, bikes and pedestrians. For example, cars usually occludes horizontal lines of the buildings and they dynamically move in and out from our scene.

1.4 Approaches

Localization of a camera is a well known problem in the community. In this section, we discuss how they have been used in the community for accurate pose estimation. We first describe how Machine Learning (ML) approaches such as regression and classification applies in the context of the localization. Then, we describe the gist of some methods based on what kind of information model they use. For example, while some approaches use large-scale pre-registered image collections with Content Based Image Retrieval (CBIR) techniques, some other approaches use more complex inputs such as 3D models or 2D models of the environment.

Models in ML are mainly divided into two categories based on how prediction is done. First approach is *classification* based prediction models. In image-based localization, classification based approaches approximate a mapping function f from input image to discrete output variables. Simple example of discrete outputs would be geo-tags for image-based localization. Second predictive modeling is called *regression* based prediction modeling. Regression based approaches approximates a mapping function f from input images to a continuous output. In our case, continuous output would be the pose matrix itself.

In the literature of image-based localization, both classification and regression based prediction models are studied. In the rest of the section, general approaches for image-based localization without classifying them as classification or regression based approaches is discussed because both machine learning models can be applied in the approaches described below.

The gist of a well studied approach to image-based geolocalization is using a set of geo-registered images to localize the query image in a CBIR framework. This results in collecting and maintaining large set of geo-referenced image databases to be used to collect visual cues for the scenes of interest. The basic idea to use geo-referenced databases for image-based geolocalization tasks constructs the gist of image-based localization where a query image is identified with the best matching image in the database. Position of the returned image is used as the position estimate. However, this only gives a position for the image since, orientation information is usually not kept in such databases. Another challenge here is that since the number of images in the collection is so large, the search would be very inefficient. Therefore, CBIR [5, 32, 33, 60, 64, 76, 93, 95] methods have been developed to make the searches more efficient with different visual cues and efficient data structures like k-d trees are used to make the search more efficient. Visual cues include defining different kind of feature descriptors such as Scale Invariant Feature Transform (SIFT) [53], Learned Invariant Feature Transform (LIFT) [97], BRIEF [10] or DAISY [88]. It is still worth to mention these descriptors even if nowadays the handcrafted descriptors have been started to replace with descriptors that are learned with training Neural Networks in end-to-end fashion. A descriptor should be invariant to many factors to be effective e.g. . scale, rotation, translation, noise, illumination, blur. However, localization methods based on image matching with geo-referenced databases are

not very practical, since many images need to be collected and registered in the collection. Even collections such as Google Street View[2] are rather sparsely sampled and exhibit only specific illumination and season conditions, making image matching a challenging task.

Even though use of ground-level images dominates the literature, some approaches [51, 64] make use aerial images. However, the problem with the aerial images is not being as widely available as the ground-level images.

On the other hand, some other approaches combine information from different modalities. While registered image collections are frequently used for geolocalization some other studies approach to the problem using detailed 3D models or cadastral maps. 3D models help methods to have a better understanding of the environment. These models usually give us detailed information and nice feature points to make a correspondence matching between the model and the image. However, acquiring such models are quite cumbersome as large geo-referenced image collections. Examples of methods approaching image-based localization with 3D structures includes *Structure-from-Motion* (Sfm) methods [35, 49, 71–73, 81]. Sfm is the problem of recovering the 3D structure of the scene and the camera motion from a set of images and it is mostly used in outdoor environments. In Sfm, camera pose is estimated by repeated operations of expensive *bundle adjustment* (BA) [89] which estimates correspondences between features extracted from the 2D image and features associated with the 3D model. SfM systems are used for learning the 3D structure of the scene, on the other hand *Simultaneous Localization and Mapping* (SLAM) [30, 41, 57] systems can be referred as a broader term for localization that usually brings information from multiple modalities together. SLAM is concerned with the problem of building a map of an unknown environment with the device while at the same time it uses the built map to localize the device. A SLAM system can work in real-time on ordered sequences of images from a fixed camera set-up, whereas Sfm can work on a larger scale with uncalibrated cameras and unordered sequences of images.

In general, all of these geo-tagged image collection approaches and approaches that require detailed 3D models do not scale very well. For both approaches, collection and maintenance of datasets are quite cumbersome and need to be updated quite often. To avoid this problem and the time consuming generation of scene image databases or detailed 3D models, some approaches are built upon low level maps such as untextured 3D models, 2.5D models [3] or simple 2D models. Low level maps are easy to maintain comparing to large scale image datasets or detailed 3D point clouds. From a practical point of view, we want to totally avoid the creation of databases on our own to be more flexible, by not being limited by sparsely sampled areas.

[2] https://www.google.com/streetview/
[3] https://www.openstreetmap.org

1.5 Contributions

In this thesis we focus on camera localization in urban environments. Instead of relying on geo-tagged image collections or detailed 3D models, this thesis makes use of broadly available untextured 2.5D maps together with an initial camera pose estimate that can be estimated with the help of frequently available sensors such as GPS, compass and accelerometer to attack the absolute camera localization problem. Initial estimate found with the help of sensors are usually erroneous due to external factors. As illustrated in Figure 1.2, the pose estimate from the sensors do not let us align a simple 3D model with the image. This shows that the localization of the camera using only the sensors are not enough to make accurate camera localization which is crucial for some applications as discussed in Section 1.2.

On contrary to many other methods using pre-registered images or detailed 3D models, we consider a more practical scenario and we use untextured 2.5D models of the urban environment to localize the images. Given a pose estimate, we leverage information from the buildings in the input images and the 2.5D map of the environment. We achieve this using advanced segmentation methods to exploit discriminative features from the buildings such as façades, edges, corners and façade surface normals.

Learning an accurate segmentation model requires a large amount of training data and manual annotation of the data is costly. We want exploit features such building edges however, the literature does not provide us with a dataset that we can exploit edges of the buildings with a segmentation method. Therefore, we later show how to make use of a 3D tracking algorithm together with the 2.5D maps to easily annotate many images that are required to train an accurate segmentation model.

In order to find the correct pose, we propose different type of algorithms for camera localization. We show accurate camera localization is possible by optimizing the camera pose using the exploited building information and 2.5D maps. We achieve this by evaluating the log-likelihood of the segmented image and the renderings of the 2.5D maps. Our optimization is based on a random pose sampling strategy applied around the initial pose estimate given by the sensors.

Later, we explore the use of the exploited image information and the 2.5D maps in a learning-based localization framework. Given a sensor pose estimate, we achieve this by learning to converge an accurate pose estimate in an iterative framework. We use semantic segmentation of the image and renderings of the 2.5D map and . However, robustly aligning the input image and the map remains challenging: The initial pose from the sensors can be far away from the correct pose, and direct comparisons between the semantic segmentation of the image and the rendering of the model is getting difficult. Therefore, our contribution is a robust method to optimize the camera pose and to estimate a good alignment between the image and the corresponding map. As gradient methods cannot be applied, using the semantic segmentation and a rendering of the map as input, we train two deep networks, one for the translation and one for the rotation, predicting directions that will improve

the current estimate for the pose. By invoking these networks multiple times, we can iteratively improve the estimate of the pose. However, if the initial error is very large, it is not possible to predict a reliable direction. We then run our algorithm from poses sampled around the sensor pose and select the best one. The efficiency and robustness of our approach is demonstrated on a complex real world scenario, where even though starting from a sub-optimal initializations, finally, the correct poses can be estimated.

Finally, we propose a method that combines the reliability of advanced image segmentation methods with the efficiency and accuracy of geometric pose estimation methods. We achieve this by extracting high-level features from the input image by using the segmentation model. We can then compute the camera pose from matches between these high-level image features and their equivalents in the 2.5D map. To do this robustly, we consider minimal solvers to compute camera poses from minimal sets of correspondences. To this end, we propose two well established minimal solvers to compute the camera pose with the extracted high-level features. Our proposed method can make efficient accurate pose estimations because we rely on geometric pose estimation with robust high-level features and efficient minimal solvers.

1.5.1 List of Publications

The contributions of this thesis led to the following peer-reviewed publications:

1. Semantic Segmentation for 3D Localization in Urban Environments
   Anil Armagan, Martin Hirzer and Vincent Lepetit
   In: *Proc. of Joint Urban Remote Sensing Event (JURSE)*
   March 2017, Dubai, UAE
   (Received the best paper award, accepted for oral presentation)

2. 3D Localization in Urban Environments from Single Images
   Anil Armagan, Martin Hirzer, Peter M. Roth and Vincent Lepetit
   In: *Proc. of Workshop of the Austrian Association for Pattern Recognition (OAGM/AAPR)*
   May 2017, Vienna, Austria
   (Accepted for oral presentation)

3. Learning to Align Semantic Segmentation and 2.5D Maps for Geolocalization
   Anil Armagan, Martin Hirzer, Peter M. Roth and Vincent Lepetit
   In: *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR)*
   July 2017, Honolulu, Hawaii, USA
   (Accepted for poster presentation)

4. Accurate Camera Registration in Urban Environments Using High-Level Feature Matching
   Anil Armagan, Martin Hirzer, Peter M. Roth and Vincent Lepetit
   In: *Proc. of British Machine Vision Conference (BMVC)*
   September 2017, London, UK
   (Accepted for spotlight presentation)

1.6 Outline

In the rest of the thesis, first, a comprehensive review of the literature and the background information is given in Chapter 2. More specifically, some background information about a powerful ML tool called Neural Networks, recent advances in image segmentation, minimal solvers will be given and then localization methods which are categorized by the type of information used by each method are discussed.

Urban environments are mostly occupied by buildings. We use advances in semantic segmentation to extract features that are informative for camera localization purposes. More specifically, we consider buildings as an important cue for urban camera localization problem and leverage the advances in semantic image segmentation for extracting features from the buildings in the input images. We, therefore, first show how to use advances in image segmentation to extract such useful features in Chapter 3. However, training a good semantic segmentation model comes with the cost of data collection. We later explain in the same chapter how to efficiently acquire training data with 2.5D maps to train the model. Finally, in Chapter 3 concludes by proposing a method for localization using the extracted information with the segmentation model together with the 2.5D maps to make accurate camera localization. We show that extracted features are discriminative enough to make accurate localization.

Further, in Chapter 4, this thesis improves the usage of 2.5D maps and semantic segmentation of input images by using them in a learning based localization method. In this chapter, we show how to train CNNs to learn to refine a pose estimate in a classification framework by comparing 3D renderings of the 2.5D maps and semantic segmentation of the input image. We later use the trained CNNs to iteratively update initial pose estimate until an accurate pose is found.

Chapter 5 proposes a method that combines the reliability of recent image segmentation methods with the efficiency and accuracy of geometric pose estimation methods. More exactly, our method explores high-level features of the buildings with the efficient and accurate minimal solvers.

Finally, Chapter 6 concludes the thesis by giving an overview and discussion about lessons learned by combining information from untextured 2.5D maps and exploiting semantic and high-level features of the buildings in the input images.

2 Related Work

Contents

This section presents background information about camera pose estimation literature. Section 2.1 discusses some background methods which are required to understand the visual pose estimation methods which are discussed in Section 2.2.

2.1 Background Methods

In this section, background needed for a better understanding of visual localization methods are described.

2.1.1 Neural Networks

This section presents the theoretical background about a powerful machine learning tool called Neural Networks (NNs) and it's variants. First, ideas behind NN is introduced and NNs with more complex structure called Deep Neural Networks (DNNs) are discussed. Variants of DNNs such as CNNs, RNNs, LSTMs and Siamese Networks will be discussed in the later sections of this chapter. Then, training of NNs in Section 2.1.1.5 and regularization of NNs in Section 2.1.1.6 is discussed. Finally, this section will be concluded in Section 2.1.1.7 with the introduction of some of the well known architectures that are used DNNs.

2.1.1.1 Artificial Neural Networks (ANN)

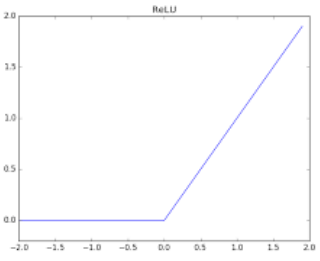
Neural Networks (NNs) or *Artificial Neural Networks* (ANNs) are set of algorithms that are developed with inspiration from the way animal's biological nervous systems work to

process information. NNs and especially, *Deep Neural Networks* (DNNs), which will be explained later, have been very popular in the last decade since computation power of computers increased enough and researchers had more understanding of them especially in terms of regularization of a NN (see Section 2.1.1.6) to back-up this powerful structures.

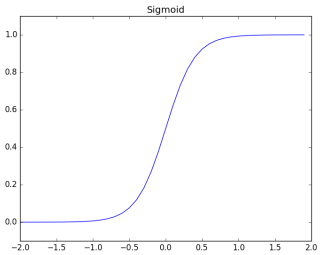
Simplest unit of a NN is called a *neuron* which is used to process the information. An example of a neuron is given in Figure 2.2. A *neuron* represents a non-linear function which is parametrized with its weight w and a bias b. The incoming information is taken as an input and processed by w and b. Then, an activation function σ such as *sigmoid, hyperbolic tangent (tanh), rectified linear unit (relu)* is applied on the result to remove linearity. Examples of activation functions σ are given in Figure 2.1. A neuron can $fire$ depending on if computed result exceeds a threshold T or not. Please note that the activation function is not necessary to define a network but it is usually required for effectiveness of the network.

A network may contain from a few to millions of neurons. A group of neurons can be arranged into a *layer* l. Each of these neurons might be densely connected to nodes in the previous input layer l to get the input data. Today's neural networks are organized into layers of nodes and they process the data mostly by moving it through a single direction which is called *feed-forwarding* the information or its input. Feed-forwarding the data through a neuron means that the neuron is taking the input data from all its connections and multiplies them with w and adds the resulting products together. Finally, the b is added optionally and activation function σ can be applied. Then, the resulting value is given as an output of the neuron. More formally, a neuron with N input values represents a nonlinear function $g : \mathbb{R}^N \rightarrow \mathbb{R}$ with its parameters θ which are the learned parameters w and b. Eq. 2.1 gives more intuition on how a neuron works for a given input x where $x \in \mathbb{R}^N$ dimensions.

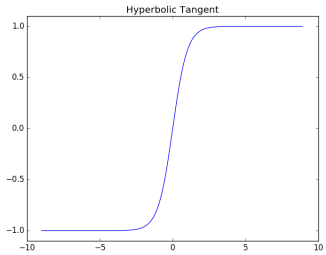
$$g(x) = \sigma\left(\sum_{n=1}^{N} w_n x_n + b\right) \tag{2.1}$$



(a) Rectified Linear Unit (ReLU)



(b) Sigmoid



(c) Hyperbolic Tangent (tanh)

Figure 2.1: Activation functions that are commonly used in neural networks.

If the network is a combination of multiple neurons in a layer l, then each of these neurons feed-forwards its input and the outputs of each *neuron* are combined and fed as input to a neuron in the next layer. A weight matrix W^l is constructed from the weights w of each neuron as well as a bias vector b^l for a layer l. Then, a layer function f^l with input x can be written as in Eq. 2.2 [28].

$$f^l(x) = \sigma^l(W^l x + b^l) \tag{2.2}$$

A network can be constructed with L consecutive layers. The number of neurons in each layer determines number of units in each layer and the number of layers determines the depth of the network. This is where the term Deep Neural Networks (DNNs) is defined. If the number of information processing layers, except the input and the output layers [28], in a network is more than one, then the network is called a *deep network*. For a deep network of L consecutive layers, each layer l_i is fed its input from the previous layer l_{i-1}. Therefore, we can define a deep network with L layers as a function f^L with combination each layer as in Eq. 2.3. Input is defined with x and parameters θ is a combination of each layer parameters θ^{l_i}.

$$f^L(x;\theta) = f^{L-1}(f^{L-2}(\dots f^1(x))) \tag{2.3}$$

f^L maps a K dimensional input x to a P dimensional output, $f^L : \mathbb{R}^K \rightarrow \mathbb{R}^P$. The output of the last layer in the network is equivalent to the output of whole network and therefore; the last layer is called *output layer*. We can also define an *input layer* which just passes the network input to the first layer. The rest of the layers between an input layer and an output layer are called *hidden layers*. An example of a network with two hidden layers is given in Fig. 2.3.

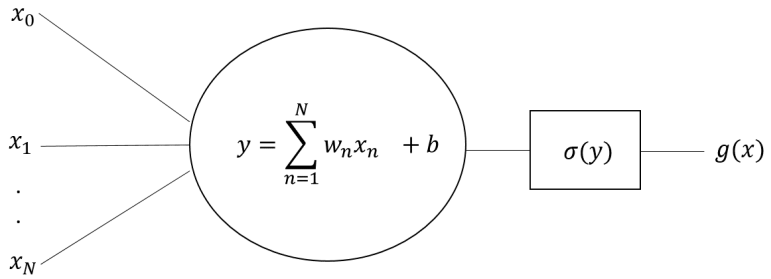


Figure 2.2: Representation of a single neuron. N dimensional input X fed to the neuron and neuron outputs $g(x)$. The neuron multiplies X with its weights W and adds bias on the result of multiplication. Then, activation function σ applied on output y of the neuron.

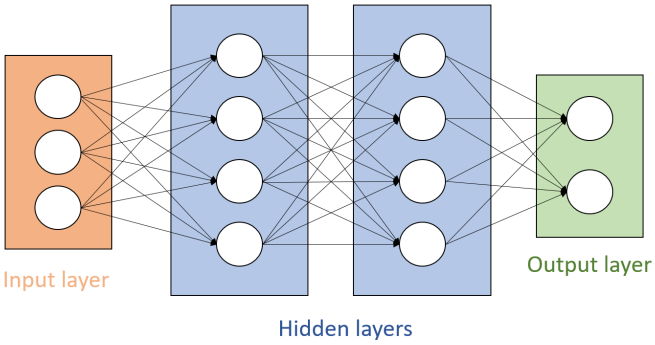


Figure 2.3: A regular 3 layer network with 2 hidden layers and an output layer. Please note that input layer is not considered as a layer when counting the number of layers for a network since there is no computation done in that layer.

2.1.1.2 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a special family of ANNs. CNNs has been investigated in 1980's for classification [28]. They have been very popular for variety of tasks from image-based location classification [94] to object detection [66] in the last decade.

CNNs are similar to ordinary NNs with fully-connected hidden layers defined in Section 2.1.1.1. Differences come from the term *convolution* which is a mathematical operation on two functions to produce a third function, for which the two functions are a *filter* and an *image* for CNNs. A simple example of convolution operation is shown in Fig. 2.4. As ordinary NNs they consist of learned weights and biases. Each neuron in a *convolutional layer* applies convolution operations with learned weights or filters on the 2D input image, sums the biases and optionally an activation function is applied on the output. A CNN may consist of multiple layers as regular networks to transform a volume of activations to another layer through a differentiable function.

Main advantage of CNNs over standard fully connected NNs is CNNs consider spatial structure of the images. This makes them behave near by pixels differently than far away pixels and this is set by size of *receptive fields* of CNN layers. Considering local spatial structure makes them a powerful tool for many tasks e.g. . image classification and object detection by defining less number of parameters comparing to a fully connected layer and therefore; enabling us to create deeper networks with many layers with less training time compared to training of fully connected networks.

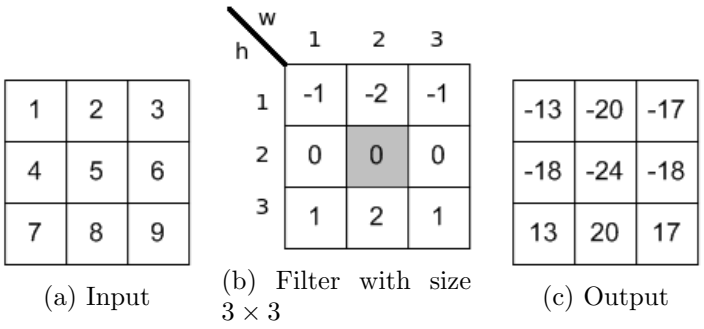
Considering a fully connected network which takes an RGB image of size 28×28 ($width \times height$) as an input would have $28 \times 28 \times 3 = 2352$ neurons in a single hidden layer. This many parameters seem to be manageable, however; when the input image size is larger and we add more number of layers into the architecture, then the number of parameters increases a lot. Networks tend to memorize training samples instead of learning

a general representation from training data when the number of parameters is too much and the number of training samples are limited. This problem is called *overfitting* [28]. CNNs take advantage of images' spatial dimensions and introduce new properties such as size of *local receptive fields*, *feature pooling* and *weight sharing* among different layers into the architecture to decrease the number of learned parameters.

A CNN is usually build upon three type of layers, *convolutional layers*, *pooling layers* and *fully-connected layers*. Fully-connected layers are discussed in Section 2.1.1.1 with the name *hidden layers* where each neuron in this layer will be connected to all the neurons in the previous layer.

Convolutional layers are where the convolutional operation is taking place with learned 2D spatial filters. Depth of each filter is same with the depth of input to that layer. Filters are convolved with the 2D input locally by sliding them through whole input. Size of the sliding is called *stride*. As the filters are slided through the inputs, the network starts to produce a 2D activation map which corresponds to the responses of that filter at every spatial position. Therefore, if the filter size is 5×5 the network does not need to learn weights as big as the input size but only learning $25 \times depth$ parameters would be enough for that layer. If the input is an RGB image with the *depth* would be 3. Since these filters applied locally, the spatial extent of this connectivity is called **local receptive field** and receptive field of a layer is same as the size of the filters for that layer. These filters are applied locally, however; applying many of them results in non-linear filters that become increasingly global. Therefore; CNNs are able to produce low level representations of the input in the lower layers but more high level representations in the later convolutional layers.

Pooling layers perform down-sampling along spatial dimensions of its inputs. Most commonly used pooling operations are *max-pooling* and *average-pooling*. For example, we can define a 2×2 max pooling operator that takes the values of every 2×2 region in a sliding window fashion and keeps the maximum value among the 4 values. A pooling



(a) Input (b) Filter with size 3×3 (c) Output

Figure 2.4: An example of a convolution operation showed on an input of depth one. A filter of size 3×3 applied on an input image of size 3×3 at every pixel of the image by sliding the filter. A valid convolution operation results in an output of same size of the input.

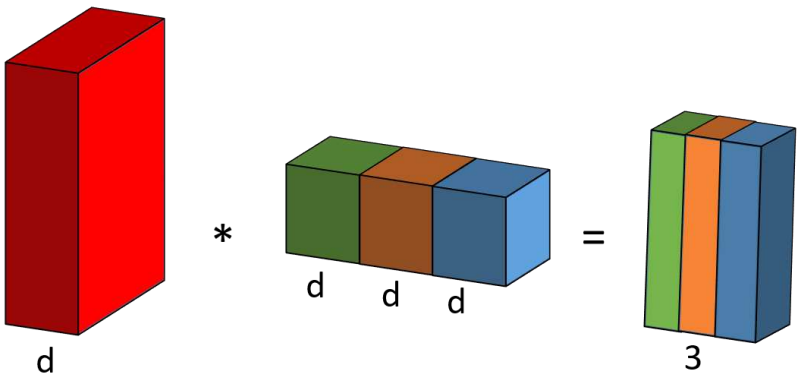


Figure 2.5: A convolutional layer with three filters. Layer's neurons are arranged in three dimensions and an output with depth three is produced because of the number of neurons in the layer. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image.

layer of filter size 2×2 is applied on an output of convolutional layer with size 32×32, to reduce the size to 16×16 and this would reduce the size of input to the next layer, hence, the number of parameters to learn.

Another advantage of CNNs is their *parameter sharing* [28] ability. Parameter sharing comes with a useful assumption that if a feature is useful to compute at a spatial position (x_1, y_1), then it should also be useful to compute another position (x_2, y_2) [28]. With this assumption in hand, we can reduce the number of parameters by sharing the weights and biases of the neurons along the same depth channel of the input. This could simply be used to avoid overfitting and reduce the training time.

Some of the well-known CNN architectures are VGG [79] and GoogLeNet [82]. VGG-16 and VGG-19 are deep networks with 16 and 19 layers, respectively. Both VGG-16 and GoogLeNet with inception module will be discussed in Section 2.1.1.7.

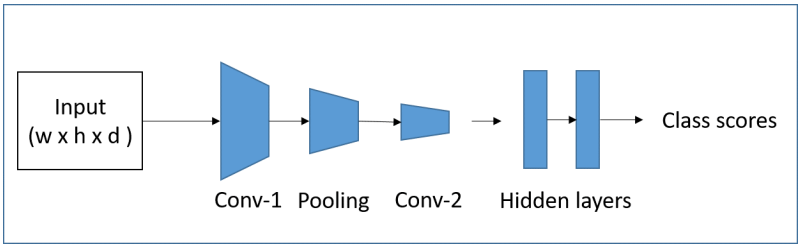


Figure 2.6: A simple CNN architecture is given for a classification task. Network takes an input image of size $(w \times h \times d)$ and feeds it through two convolutional layers, a pooling layer followed by two fully-connected layers.

2.1.1.3 Fully Convolutional Networks (FCNs)

Standard NN architectures have fully connected layers where each neuron is connected to neurons in the previous layer where as a *Fully Convolutional Network (FCN)* consist of layers where all learnable layers consist of convolutions. So FCNs do not have any fully connected layer. FCNs are applied on different tasks from detection [17] to segmentation [52]. To my knowledge, Matan *et al* [55] was first to propose such an architecture while extending GoogLeNet [46] architecture called to recognize strings of digits because the original architecture was designed to have single dimensional input strings.

There are several advantages of FCNs compared to standard CNNs. Since CNNs are using fully connected layers, these layers computes a nonlinear function as explained in Section 2.1.1.1. However, an FCN completely consist of convolutional layers and this makes them compute a nonlinear filter. The advantage of this is being able to operate on any input sizes and producing output of corresponding spatial dimensions while the learned filters are slided through the input.

Another advantage of FCNs is having significantly overlapping receptive fields because there are no hidden layers. This simply makes computation and backpropagation more efficient and also reduces the number of parameters to learn since the hidden layers require much more parameters to learn than the convolutional layers.

Not having fully connected layers do not disable FCNs to make use of pre-trained architectures like VGG-162.1.1.7 with fully connected layers. One can simply replace neurons in fully connected layers with 1×1 convolutions and use the learned weights from pre-trained fully connected layers. Fig. 2.7 shows an example of this conversion from fully connected layers to fully convolutional layers.

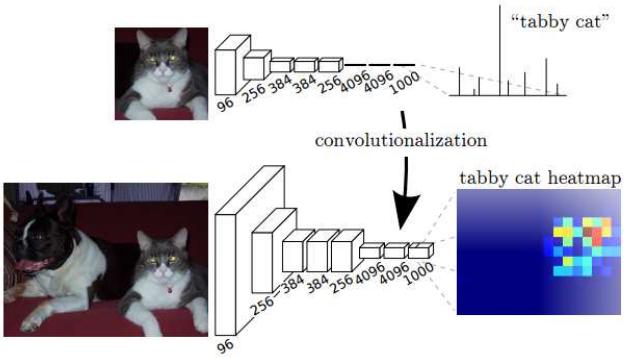


Figure 2.7: A classification network can produce a heatmap by transforming fully connected layers into convolution layers. (Figure is taken from [52].)

Fully connected layers lose most of the spatial information in the image since all neurons in that layer is connected to all layers in the previous layer. However, FCNs are good to preserve the spatial information since they are fully convolutional. Also since FCNs are fully convolutional, they are able to perform on any input size. Therefore, these features

of FCNs make them quite useful for segmentation task. Long *et al* [52] uses pre-trained CNN [79] and converts them to an FCN with replacing pre-trained fully connected layers by 1×1 convolutions. Since CNN architectures like VGG-16[79] uses pooling layers to reduce the number of parameters the spatial dimension of the output of convolutional layers is reduced. However, segmentations tasks requires to assign a class label for each pixel in the image. Reduction of spatial dimensions because of pooling layers is handled with introducing upsampling deconvolutional layers where a backward convolution operation is applied. Another improvement on the results come from introducing *skip* layers [42]. Since there are multiple pooling operations applied in VGG-16 architecture the spatial information is already lost because of the previous pooling layers. Therefore, skip-layers are introduced by passing the information from previous pooling layers to the last layers.

2.1.1.4 Siamese Networks

Siamese neural networks is a class of NN architectures that has two or more identical subnetworks. First siamese architecture is proposed by Bromley and LeCun in 1993 [8] for signature verification, see Fig. 2.8. Identical two networks means that they share exactly same parameters and backpropagation is mirrored for all subnetworks. Siamese networks accepts inputs of different types but after feed-forward step they are joined by an energy function at the top.

Siamese architectures mostly used for finding similarity or a relationship between two images in Computer Vision community. Advantage of such architecture like a Siamese Network is having less parameters since weights are shared across twins.

2.1.1.5 Training of Deep Networks

The procedure used to carry out the learning process in a NN is called training. After deciding the layers and structure of a network, it should be trained on learnable parameters. There are three main learning types in Machine Learning, *supervised, weakly supervised* and *unsupervised learning*. In this section, training of neural networks with supervised learning will be focused.

In supervised learning, a general approach is to divide our data into two sets, a *training set* and a *validation set*. The training set is used to determine learnable parameters θ for network's performance and the validation set is used to validate our network on examples which the network did not see before during the training. Each data sample x is labeled with ground truth information y and these pairs are shown to the network either one by one (batch size equals to one) or having batches of multiple samples. Each time a batch is fed to the network, a *loss* is calculated by comparing the network outputs and y. Then, the network updates θ depending on the magnitude of the loss. A network completes an *epoch* of the training when it sees all training data once and the training may take even hundreds of epochs depending on the loss function. It is important to use validation sets during the training to validate that our networks can generalize the information learned

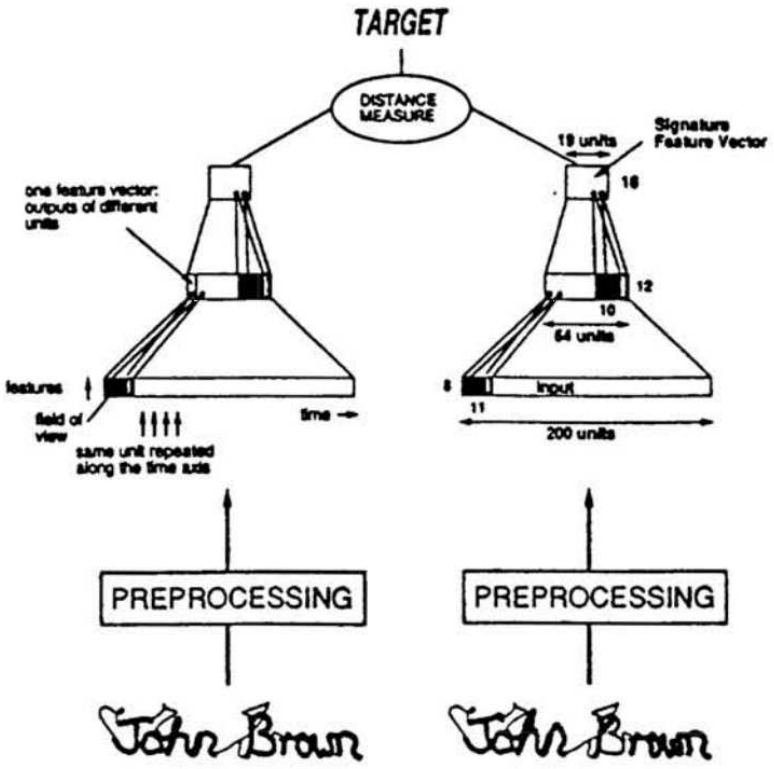


Figure 2.8: First siamese network architecture proposed by Bromley *et al* [8]. It consists of two identical networks each has an input images of dimension 8×200. Figure is taken from [8].

from training samples. If the *loss* from training and validation sets does not have a positive correlation this means that the network *overfits* on the training set and network will not perform well on unseen examples. Overfitting in NN will be described in Section 2.1.1.6.

Training of a network depends on many factors to perform well and the rest of this section will explain some of these factors.

Parameter Initialization. First, all θ should be initialized. There are different types of initialization of parameters e.g. Random, Xavier [27] or initialization from a distribution such as Bilinear. The initialization type has an important role in training by effecting how well the network learns or how long the training takes [27].

Loss Functions. Depending on the task we are trying to solve many different loss functions has been proposed e.g. mean squared, cross entropy, logarithmic, cosine similarity, hinge. Which loss function to use completely depends on the task and the definition and use of these functions are far beyond the scope of this thesis.

Optimization Algorithms. The loss function the network computes has to be minimized. The network parameters are updated by using *backpropagation* [28] algorithm and there are different optimization algorithms such as Gradient Descent (GD), Stochastic

Gradient Descent (SGD), or more advanced ones like RMSProp and ADAM [28]. The type of the optimization algorithm effects the training by deciding how efficient it will the global minimum of the function we are trying to minimize.

Learning Rate. Learning rate is to decide how much to update each parameter in the estimated direction by backpropagation and optimization. Using adaptive learning rates by decreasing or increasing in certain situations depending on the current loss of the network has been also useful trick for training.

2.1.1.6 Overfitting Problem in Neural Networks

A model with a large number of free parameters can describe the underlying data very well. However, such a model can agree well with the data we used to train the model but this doesn't mean that it will be a well generalized model for the task that is trained for. This would mean that the model has enough freedom to represent the data of training but does not generalize the model to other data rather than the data used for training. In Machine Learning, such a problem is called *overfitting*.

One way to reduce overfitting is to increase number of training data to make the model see more real-world examples. This can be done either by collecting more real data or with augmentation of the current training data.

Another approach is to use *early stopping* during training. Early stopping of a training means that the model has learned as much as it can from the current training data. This can be decided by observing the losses of training and validation data. Training of the model is stopped when the loss on the validation data starts to increase while the loss on the training data is still decreasing.

Overfitting is a general problem if the training set is small and the number of parameters to learn is too much. Therefore, we need *regularization* techniques. Regularization is defined as any modification that would reduce the generalization error but not the training error [28]. Some of the *regularization* methods are described in the rest of this section.

Weight Decay. *Weight decay* is one of the most commonly used regularization technique. It makes the assumption that a model with small weights is simpler to learn than a network with large weights. Therefore, it tries to penalize large weights by using either *L1 norm* or *L2 norm*. L1 norm penalizes the absolute value of the weights while L2 norm penalizes the squared value of the weights.

Batch Normalization. *Batch normalization* (BN) normalizes the output of a previous layer by subtracting the batch mean and dividing by the batch standard deviation and introducing two learnable parameters α and β [36]. This is a type of regularization of a network that increases the stability. It can be interpreted as doing a pre-processing before a layer but without breaking the differentiability of the overall function that a network

tries to learn. It is usually added after fully connected layers but also adding it after convolutional layers is an effective regularization.

A problem in Machine Learning algorithm is called *Internal Covariate Shift* and BN is applied to reduce the effect of this problem. Internal Covariate Shift refers to the change in the input distribution in our model between different layers. The parameters of a layer in a network is affected by the parameters of all previous layers. Therefore; small changes in the layers affects all the others. This leads to change in the input distribution.

BN reduces Internal Covariate Shift by adding two learnable parameters to each layer. Normalized output of a layer is multiplied by α and β value is added to the result of multiplication. Then, the result is fed to the activation layers, if any activation function is used.

BN leads us to use higher learning rates because it makes sure that internal covariant shift is reduced and there is no output very high or very low. It allows use of saturating nonlinearities and reduces the dependence of gradients on the scale of parameters. Since the initialization of the parameters are very important for a NN, BN also reduces the importance of parameter initialization.

Dropout. Another effective regularization technique is called *Dropout* and it is commonly assumed as a layer type. This layer randomly deactivates some of the neurons or another way of saying this is the weights of some of the neurons are set to zero. Then, output of the remaining neurons are forwarded to the next layer. This simply reduces the dependency to all neurons in a layer. It is hard to assume all neurons will always give useful information, therefore; dropout reduces to dependency on all the neurons by deactivating them with a probability distribution $p \in (0, 1)$[28]. In this way, the neurons become less sensitive to the other neurons and the model becomes more robust. Dropout is mostly applied on the fully connected or input layers. Fig. 2.9 shows how dropout is applied on the inputs to a layer.

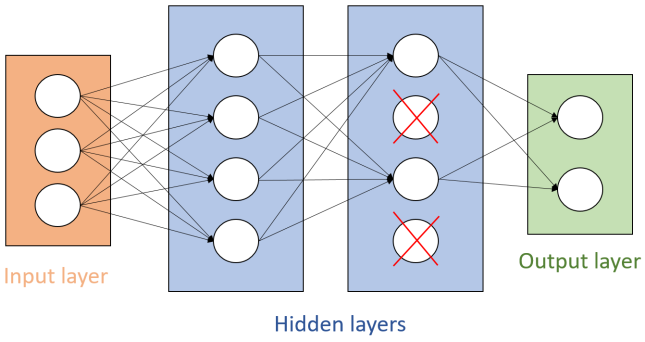


Figure 2.9: Visualization of dropout applied between the two hidden layers (blue). It randomly deactivates some of the neurons, in this case 50% of the neurons in the second hidden layer is deactivated or weights of those neurons are set to zero.

2.1.1.7 Famous Network Architectures

This section mainly discusses two of the well known architectures VGG [79] and GoogLeNet [82] in DNNs. Both of these architectures have shown to be very successful for classification problems but with the help of *transfer learning* the scope of them has extended to different tasks like semantic segmentation [52] and camera localization [40]. Indeed, both VGG and GoogLeNet is introduced after another famous architecture called AlexNet [43]. AlexNet will not be discussed in the later section, however, ideas in that made AlexNet architecture were in the core of many other later architectures. Therefore, it is worth to mention AlexNet. It is composed of five convolutional layers followed by three fully connected layers. AlexNet was first to use activation function ReLU for the non-linearity to decrease the effect of *vanishing gradient* problem and also it was first to apply dropout in a deep architecture.

VGG. VGG network design is one of the first works that took attention to receptive fields of stacked convolutional layers with small kernel sizes such as 3×3. There are two architectures proposed by the Visual Geometry Group (VGG) in Oxford University. Architectures are called VGG-16 with 16 layers (13 convolutional layers, 2 fully connected layers and a softmax layer for the output) and VGG-19 with 19 layers (16 convolutional layers, 2 fully connected layers and a softmax layer for the output) [79]. In this section, I will focus on VGG-16 architecture. It is also notable that only difference of VGG-19 from VGG-16 is having three more convolutional layers to have a larger receptive field.

Figure 2.10 shows design of VGG-16 network, it takes input images of shape $224 \times 224 \times 3$ (width $\times$ height $\times$ number of channels) and it consists of many stacked 3×3 convolutional layers. Advantage of such architecture comes from the receptive field of such stacked layers with small kernel sizes. These 3 stacked layers with 3×3 kernels with stride 1 has same receptive field with a layer with kernel size 7×7 but they have less parameters to learn. Therefore, it is computationally more efficient.

Another advantage of VGG architecture is increased nonlinearity and depth since it has stacked layers with small kernels compared to replacement of a stack with a layer of a larger kernel size. This enables the network to learn more complex features with a lower cost.

Dimension of the inputs to the next convolutional stack is reduced by 2×2 max-pooling layers and then, convolutional layers are followed by three fully connected layers with sizes 4096, 4096 and 1000, respectively. The last fully connected layer applies a softmax classifier and it has dimension 1000 since the network is applied on ImageNet [20] where the number of classes to predict is 1000. An activation function, ReLU, is used after each convolutional and fully connected layer (except the last fully connected layer).

A disadvantage of VGG architecture is its size, number of parameters. The network is so dense and it contains around 160M parameters to learn. Even today, many of the GPUs cannot handle a structure that large.

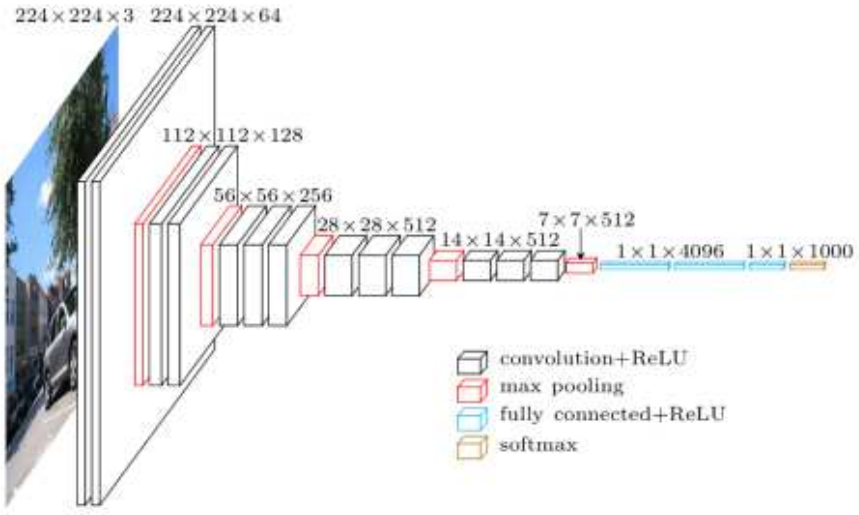


Figure 2.10: Architecture of VGG-16 network which contains around 160M learnable parameters. (Figure is taken from [52].)

GoogLeNet. VGG architecture had a great accuracy on ImageNet challenge but another architecture called GoogLeNet [82] is proposed after VGG and GoogLeNet had great success both in terms of accuracy and efficiency of the model. Szegedy *et al* [82] proposed to use a special module which is called as *inception module* in GoogLeNet. Indeed, this module is what makes GoogLeNet so special and indeed overall architecture of GoogLeNet adapts nine of these inception modules.

An inception module is a combination of multiple convolutional layers as well as a pooling layer applied in parallel. Fig. 2.11 shows how the layers are organized in an inception module. This module gives the flexibility to the designer not to worry about which kernel size is to use and if there should be pooling layer or not. Inception module contains all of them in the same module.

GoogLeNet consists of 22 trainable layers with 9 inception modules. It also contains multiple output layers. This can be assumed as a protection against vanishing gradients problem. Since such deep networks vanishing gradients are a common problem and GoogLeNet approaches this by collecting many outputs if the later layers which are tend to lose information.

ResNet. As we have seen from GoogleNet architecture, increasing the depth increases the accuracy together with the carefully designed inception modules. However, increased depth of the architectures make the model overfit easily before successfully approximating a generic mapping function $H(x)$.

ResNet architecture first proposed by He *et al* [34] for image classification tasks. The gist of the approach relies on the idea of modifying the approximated mapping function

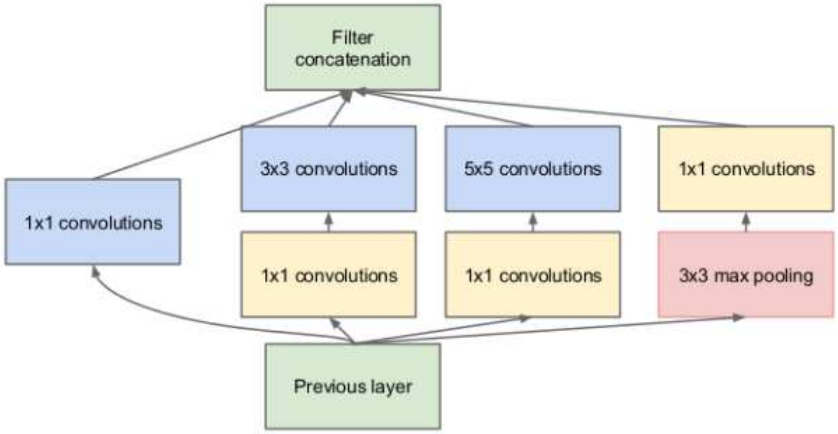


Figure 2.11: Architecture of an inception module. Input is processed in four parallel tracks. In each track convolutional layers with different kernel sizes are used. The important point to make here is to reduce the number of operations required by convolutional layers with large kernel sizes Szegedy *et al* [82] applied convolutional layers with 1×1 kernels, *bottleneck units*, before each convolution layer with larger kernels. Bottlenect units reduces the depth of the input to next layer with larger kernel size to apply less number of operations. Inception module also removes the designer to worry about the used kernel sizes in a layer because an inception module simply applies variety of sizes. (Figure is taken from [82].)

$H(x)$ for inputs x to approximate a residual function $F(x)$. So, instead of approximating $H(x)$ with a stack of non-linear layers by the network, the key is to learn a better defined function such as $F(x) := H(x) - x$. These functions encoded in the architecture as *residual blocks*. An example of the a residual block can be seen in Figure 2.12.

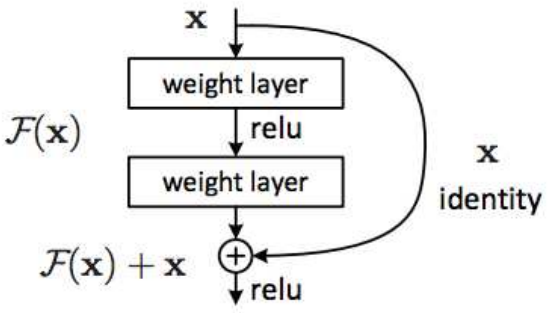


Figure 2.12: Illustration of a residual learning block. $F(x) := H(x) - x$ is learned using the residual blocks. (Figure is taken from [34].)

With the advantage of better defined mapping functions, ResNet has shown to go as deep as 152 layers without the overfitting problem. ResNet achieves better accuracy than both VGG and GoogleNet architectures while it is computationally more efficient than

(a) Input image (b) Segmented image

Figure 2.13: Semantic segmentation example. Each pixel in the **(a)** input image is assigned a label for one of the classes e.g. motorbike, driver or background in **(b)** the segmented image. (Images are taken from PASCAL VOC2012 dataset [23].)

VGG even if the network is much deeper.

2.1.2 Semantic Image Segmentation

Semantic segmentation of images plays a crucial role for scene understanding in our camera localization approaches and it is also a very active research area in CV. Semantic segmentation is understanding of the images at pixel level by assigning each pixel in the image an object class label. Figure 2.13 shows an example of a perfectly segmented input image into three classes, motorbike, driver and background.

Deep Learning (DL) based approaches gained popularity in the last decade and CNNs has shown to have great success on segmentation tasks. One of the first approaches with DL was to classify each pixel based on image patches in a classification framework. This is because CNNs were used with hidden layers and they require the input to be fixed size.

Long *et al* [52] showed how to make use of CNNs to make dense predictions without any hidden layers by replacing the hidden layers with convolutional layers. This approach has few advantages since without hidden layers, CNNs could process images of any size. Another advantage is that since there are no fully connected layers the processing can be done much faster with less memory requirements.

A disadvantage of CNNs for dense image prediction is the use of pooling layers where the spatial information is mostly lost. Pooling layers are dangerous for segmentation problem since pooling layers reduces the dimensionality while discarding the localization information but increasing the field of view of the network. This problem is solved by proposal of architectures called *encoder-decoder* and it is also used by [52]. Encoder part acts like convolutions and pooling layers while reducing the dimension and increasing the field of view. On the other hand, decoder part tries to decode and recover the spatial information lost during encoding part. Adding skip connections from the encoder to the decoder is another trick to help the decoder recover the object details better. Such an encoder-decoder architecture with skip connections is used by Long *et al* [52].

An important factor for the segmentation networks as mentioned with pooling layers

is the size of receptive field of convolutions. If the receptive field is small, it is hard for the network to see whole context in the image. Pooling layers increases the receptive field but reduces the resolution. A solution to this problem is proposed with *dilated (atrous) convolutions* [12, 13, 98]. Dilations are used within the kernels of convolutional layers to increase the receptive field for dense predictions. An advantage of dilated convolutions is that they do not increase number of parameters.

[98] introduces context modules that make use of dilated convolutions for multi scale aggregation. [12, 13] uses atrous spatial pyramid pooling (ASPP) to apply dilated convolutions with different rates and combine them later on with element-wise summation in the pooling layer.

VGG [79] architecture showed promising results in image segmentation tasks and used by [12, 98] but later [13] showed Residual Network (ResNet) [34] architectures performs better at PASCAL VOC challenge [23].

In this thesis. we use recent advances in image segmentation to use it in a pose estimation framework. We consider urban scenarios where man-made structures are dominant and especially the structures like buildings dominate the main part of the scene. Therefore, buildings and its parts such as façades and edges play an important role for our localization method. Our localization methods use [52] to segment the input images into our semantic classes, façades and edges of the buildings.

2.1.3 PnP Problem for Pose Estimation

Determining the orientation and position of a fully calibrated perspective camera from n ($n \geq 3$) 3D points and their corresponding 2D coordinates in the image space is known as the Perspective-n-Point (PnP) problem [31].

A set of N 3D points $X_i, i = 1 \cdots N$ in world coordinate system is projected to the 2D image coordinate points $u_i, i = 1 \cdots N$ with a complete camera projection matrix P as in Equation 2.4.

$$u_i = PX_i \tag{2.4}$$

P is defined with the camera intrinsic parameters K and it's extrinsic parameters $[R|t]$ for camera's position vector and orientation matrix. Then, P is defined as follows:

$$P = K[R|t] \tag{2.5}$$

and where K is defined as in Equation 2.6 with camera's principal points (x_0, y_0), skew parameter s and focal lengths, f_x, f_y.

$$K = \begin{bmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

The full camera projection matrix can be estimated by using at least 6 point correspon-

dences. Then, an homogeneous linear system created for 6 points with the Equation 2.4.

Given K, estimating the extrinsics of the camera $[R|t]$ requires to solve a system of non-linear equations created with 3 correspondences. The minimal case of PnP problem is P3P problem where $n = 3$. P3P has been deeply studied in the literature [31]. In practice, P3P is used together with RANSAC [24] algorithm to remove the outlier correspondences. P3P considers 3 reference points to solve a system of non-linear equations with 4 solutions and uses a last point, called as control point, which used to find a unique solution.

Lepetit *et al* [47] proposes a general solution for $n \geq 4$, called efficient PnP (EPnP). The main idea is to express each reference point as a weighted sum of four virtual control points and the control points become the unknowns of the system. Then the problem reduces to estimating the coordinates of these control points in the camera referential. This is achieved efficiently with a complexity of O(n) by expressing the coordinates as weighted sum of the eigenvectors matrix and solving quadratic equations to pick the right weights for the virtual points.

2.1.4 Minimal Solvers

A general approach to solve a geometrical problem is to use all available image features and solve it using some least squares measure over all features. However, minimal solutions using only minimum number of point correspondences to solve the problem have proven to be noise resistant. Using minimal solvers has been used as an efficient and accurate way to deal with noisy data including outliers. Geometrical problems are frequently attempted to solved by minimal number of image features. Minimal solvers are applied on wide range of applications from pose estimation to camera calibration are being addressed with minimal solvers. A comprehensive list of minimal problems proposed in CV community is kept[1].

Minimal solvers typically rely on geometrical image features for hypotheses generation to find a good fit applying the RANSAC algorithm [24]. RANSAC was introduced to be used with a P3P [25] algorithm, which is a minimal solver. Since then, many minimal solvers have been introduced, for example to compute a camera pose from line correspondences [22], register a camera including internal parameters [9, 45], the essential matrix [59] from point correspondences or relative pose from five points correspondences [59]. Minimal solvers are getting popular and they are applied on various applications. The problems typically involve solving a polynomial system.

In this thesis, we introduce two minimal solvers adapted to our application in Chapter 5. Our solvers' task is to compute a 2D pose (2D translation+rotation). The first solver computes the 2D pose from 3 edge correspondences and the second solver computes it from 2 edge correspondences plus a façade's normal correspondence.

In this chapter, so far, we discussed the background knowledge needed for a better understanding of the localization methods. In the rest of the chapter, approaches for the camera localization are presented and some of these methods are discussed in detail. We

[1] http://cmp.felk.cvut.cz/mini/

categorize the approaches in camera localization based on the source of information used in the gist of the methods.

2.2 Image-based Visual Localization

This section discusses several methods that attack the problem of image-based camera localization. First, Section 2.2.1 discusses the approaches that use registered images as their only source of information. Then, other methods that use and combine information from other modalities such as 3D models are presented in Section 2.2.2.

2.2.1 Localization with Registered Images

This section presents an overview of camera localization methods that are based on ground level registered images. Methods based on registered images rely on images and they do not use any other source of information rather than images. More specifically, a large set of images are collected and they are registered in to a dataset with their geo-location tags or 6 DoF camera poses.

Camera localization using registered images is one of the well studied branch in this task. Therefore, it is important to give an overall overview of the image based localization methods with registered images before going deeper into specific methods. Localization using registered images can be categorized into three sub-problems based on how the task is solved, as a Content-Based Image Retrieval (CBIR) problem [5, 32, 33, 60, 64, 76, 93, 95], 2D-3D correspondence matching problem, a classification problem [37, 94], and recently as a regression problem [40]. These approaches are discussed below.

Image based localization with registered images is mostly studied as an image retrieval task, more specifically, Content-Based Image Retrieval (CBIR). This task is based on understanding contents of the images in our database, in our case a database means registered images with their geographical tags. CBIR retrieves the most similar images in a set when a new image is queried based on its contents. Content means here all important cues such as landmarks, street signs, road markings or architectural details in the image. Defining each image with unique signatures is important for CBIR. A way to define unique global signatures on the images is extracting global descriptors [32] from images. Another way is based on local features such defining bags-of-visual-words (BoVWs) or inverted indices [60, 80]. Local features have been proven to work better than global descriptors [60, 80] but disadvantage of them is that memory requirements because the descriptors are larger to store and they are mostly not invariant to natural scenes. Since natural scenes are lack of nice local features, it is hard to match them against the registered set.

Instead of using local or global image descriptors like SIFT [53] or [61], CNNs has been used as great feature extractors. One of the first successful application as a scene recognition task has been proposed with the SUN database [96] by extracting features with

Overfeat [78] CNN network. Extracting image features with CNNs usually outperforms other local or global descriptors in this task.

When a dense imagery of an environment is available, those images can be used to construct a 3D model with structure-from-motion [72, 81] methods. Then when an image is queried a correspondence set is created between the interest points in the image and 3D points in the model. This correspondence set results in Perspective-n-Point (PnP) problem and a 6 DoF camera pose is found by solving this system [48, 49, 72]. Since some applications require 6 DoF pose of the camera, the advantage of such methods is that when a 3D model is available we can come up with 6 DoF camera poses.

Another way to approach camera geo-localization problem is approaching as a classification task [32, 33, 94]. Approaching a localization system as classification problem is a trivial task when the coverage of the system is whole world. A good example of such a system is shown in PlaNet [94] which will be discussed below.

Recently, localization problem is approached as a regression problem by [40]. Regression task fits into the needs of image localization problem since in regression the number of possible outputs is infinite. An image could be taken from any where on the world without discretizing the output space. Therefore, regression task fits the needs of image localization task very well. PoseNet [40] is a good and, to my knowledge, the first example of image localization as a regression problem. Some variations of PoseNet has been already proposed [38, 39, 92].

In the rest of this section three methods based on registered images will be reviewed in detail. First, a classification based method, (i) PlaNet [94], that uses huge amount of training data where the coverage is whole world and achieves superhuman level of accuracy on urban area will be discussed. Then, two regression based image localization methods will be presented, (ii) PoseNet [40] and its variations which are great examples of transfer learning will be presented, (iii) CNNs+LSTMs for structured feature correlation.

PlaNet [94] approaches the image localization problem as a classification problem with a very large set of registered images. PlaNet divides the world surface into geographical cells and when a new image is queried it tries to assign likelihoods for each cell.

PlaNet trains a CNN with 126M images with Exif geo-locations crawled from the web, 91M for training and 34M for validation. Crawled images has been slightly preprocessed by removing the images with clip-art, diagrams and other unrelated contents while still keeping images of pets, food or products where there is almost no clue about location of those images. Therefore, the training set is pretty noisy. For testing, authors collected 2.3M images by removing test images which are similar to any training image by applying a near duplicate image detection algorithm.

PlaNet uses Google's open source S2 geometry library[2] to adaptively divide the world surface into 26263 cells. S2 cells are created by projecting the surfaces of a cube onto surface of a sphere. Six sides of the cube are subdivided by six quad-trees. Size of the cells

[2]https://code.google.com/archive/p/s2-geometry-library/

are adaptive since PlaNet applies an adaptive subdivision based on the geo-tags of images. Each quad tree of a cell is descended and cell is subdivided until there is no cell left with more than 10000 images and cells with less than 50 images are discarded. Therefore, areas where a photo is very unlikely to be taken are not considered.

Input to the CNN is raw pixel values of the images and the network outputs a one-hot vector encoding the cell containing geo-tag of the image. Advantage of such classification approach is since it has likelihoods for the query image belonging to each cell, it can take advantage of uncertainty about its predictions. The CNN is based on Inception structure which is described in Section 2.1.1.7 and it has around 100M trainable parameters. Training of such a large network with such a large data took around 2.5 months with 200 CPU cores.

The framework uses 2.3M test images and the results claims to have much better accuracy than another large-scale localization method Im2GPS [32] and methods which use other feature descriptors. Further, the classification based approach is extended to take into account temporal dependency of images that are taken approximately within similar times. For this purpose Weyand *et al* adopts LSTMs and applied them on photo albums that are uploaded on *Flickr*. Performance has significantly improved since some images that do not depict any visual cue to make inference about their location can be predicted correctly with the help of other images in the photo album.

PoseNet [40] is a great example of approaching localization problem as a regression problem. In contrast to other approaches where the output space is discretized, outputs of regression based approach are not discretized over the space.

PoseNet proposes the first robust regression based localization system that predicts 6 DoF camera pose from single RGB images without any other source of information like depth. This is achieved by using a 23 layer CNN, more specifically GoogLenet in Section 2.1.1.7, and training it in regression fashion. Training is started with pre-learned weights from ImageNet. This is also a good example of transfer learning since in ImageNet the task is a classification task but PoseNet shows that it can be applied on a different task as regression.

To train this regressor, SfM methods are used to create a 3D model of the environment. First, a training set is collected for the environment of localization and images are used to create a 3D model by using SfM. In this way, training set with 6 DoF poses annotated for each image is created.

The network outputs a 7 dimensional pose vector, $p = [x, q]$ where x is 3D position (3 dimensions) and q is quaternion representation (4 dimensions) for the 3D orientation of the camera. Rotation matrices require orthonormalization during training, therefore; quaternions are used to represent camera rotation since they are easily mapped to legitimate rotations by normalization to unit length [40].

During the training, loss function in Eq. 2.7 is minimized using SGD. Such an objective function as in Equation 2.7 learns position together with orientation. PoseNet claims that learning position and orientation separately performs worse than learning them to-

gether. It is explained as splitting the learning of position and orientation makes them stop learning each information necessary to factor out the orientation from the position, or vice versa [40].

$$loss(I) = \|\hat{x} - x\|_2 + \beta \left\| \hat{q} - \frac{q}{\|q\|} \right\|_2 \tag{2.7}$$

β is a scale factor to arrange the loss value between position and orientation equally because the position error is usually much larger for outdoor scenes. Since PoseNet is applied on both indoor and outdoor scenes, arranging β value is necessary in learning for different type of scenes.

PoseNet achieves approximately $2.40m$ position and $6.76°$ orientation error on large outdoor scenes. Error rates are decreased by predicting multiple crops from a frame and averaging their output poses. When applied on indoor scenes, PoseNet achieves approximately $0.44m$ and $10.40°$ error rates.

A disadvantage of PoseNet is being specific to the trained environment. This means that you cannot apply the trained regressor to other unseen environments. However, it is noted that it is possible to tune the network by providing small set of examples of the new environment. However, this is not well experimented in the paper.

There had been further improvements on the original PoseNet study. Some of these improvements will be briefly described in the rest of this section.

Bayesian PoseNet [38], same as in other tasks like classification, taking uncertainty into account is also important for regression tasks, especially in pose regression. Therefore, [38] considered uncertainty in PoseNet framework. While the original PoseNet produces a deterministic output for an input image, Bayesian PoseNet [38] considers that an ambiguous image could be generated by multiple poses. [38] achieves this making use of dropout layers (Section 2.1.1.6) in training and also during the evaluation to estimate uncertainty of the predicted pose. Bayesian networks improves the error rates of original PoseNet to $1.96m$ and $6.02°$ for outdoor and $0.47m$ and $9.81°$ for indoor scenes.

PoseNet with geometric loss functions [39] is an improvement published by the same authors [39] with the introduction of a new loss function that considers the geometric constraints in the projection, more specifically mean residuals of point projections.

LSTMs for structured feature correlation [92] uses LSTMs on top of a CNN architecture for feature reduction that is suitable for localization. PoseNet [40] showed image-based localization is possible with regression based approaches, however; their results are still not comparable to other feature point based methods like the state of the art Active Search [74] method.

A great improvement after PoseNet came through with a combination of CNNs and LSTMs. Same CNN architecture (GoogLeNet) with the same modifications in PoseNet is used by [92]. Then, LSTMs are used for structured feature correlation by Walch *et al* [92]. While PoseNet carefully uses dropout technique to prevent overfitting caused by the high dimensionality of fully connected (FC) layers, [92] make use of LSTMs on the output of

the last FC layer. LSTMs are used to better correlate features of convolutional and FC layers by reducing dimensionality in a structured way.

Instead of using LSTMs, one could reduce dimensionality of FC layers with some other methods like principal component analysis (PCA), however; [92] shows LSTMs' memory blocks are more effective in this manner.

Dimensionality of last FC layer is 2048 in the architecture and since the feature dimension is too long for LSTM to correlate from beginning to the end, Walch *et al* make use of 4 LSTM networks and apply them in different directions on the previous output. Output of last FC is reshaped to a 32×64 matrix and four LSTMs are applied in different directions, up, down, left and right. Then outputs of these LSTMs are concatenated and fed to two FC layer to make predictions, one for the position and the other one for the orientation.

[92] and their results claim that this technique selects features that are more useful for image localization techniques with an improvement around 30% wrt. PoseNet. However, even if the results improved a lot wrt. PoseNet, still feature point based methods like Active Search [74] are better than regression based approach. However, a big drawback of feature point based methods is they are not able to recover poses from scenes where repetitive structures are too much or there are a lot textureless objects as showed by Walsch *et al* by experimenting on a new large scale indoor dataset.

Experiments show that CNN+LSTM structure achieves better accuracy than the results of PoseNet because of an efficient dimensionality reduction approach, however; there is still a margin to close wrt. feature point based localization methods such as Active Search. Advantage of using method proposed by Walch *et al* is noteworthy when the scene includes textureless or repetitive objects.

2.2.2 Localization with Combining Information from Multiple Modalities

In this section, image-based localization methods which make use of the query image together with other sources of information such as aerial images, orthophoto maps, Digital Elevation Models (DEMs), LIDAR sensors, cadastral maps or cadastral maps with height information (2.5D maps) will be discussed. This section first gives a brief introduction to methods in this domain. Later, it details some of the localization methods.

Aerial images or orthophoto maps are one of the other important input for localization methods. Even if aerial images could be classified in the previous chapter as pre-registered images since they provide methods with different source of information they are better to be discussed in this chapter. Aerial photos are made by using central projection where they represent the earth's surface as observed, however; orthophoto maps are created by orthogonal projection of aerial images and by removing distortions caused by different heights of objects, earth's surface or distortions caused by the camera. Aerial images or orthophoto maps are considered in many studies [50, 51, 91, 95]. Key idea in these works

is to make cross view matching between the query image and such maps or aerial images. For example, [50] uses a database made of triplets of ground images, an orthophoto map and an attribute map which includes attributes like water, grassland, vegetation to make cross view matching when an image is queried. [51, 86, 95] uses deep representations to make cross-view matching. [95] proposed a method where image features are extracted by a CNN from both ground level and orthophoto maps. Then localization is completed by finding nearest neighbors from the extracted features. [51] also used CNNs by adopting a Siamese architecture to make cross view matching from Google Street View[3] and tilted aerial images. Similar to [51], Tian *et al* [86] also uses Google Street View images but this method separates each building with an RCNN and apply Siamese architecture on each building to make the matching with reference aerial images. These studies show that methods using orthophoto maps or aerial images can be applied both in urban and rural environments.

Simultaneous Localization and Mapping (SLAM) is also considered as a localization system where inputs from different modalities such as LIDAR sensors, depth cameras, stereo images or GPS are usually combined. SLAM considers to construct a map of the environment while simultaneously localizing the agent. Deep learning based SLAM systems started to be considered in some recent studies [21, 85]. These methods will not be discussed further since SLAM systems require much more explanation than scope of this thesis.

Another source of information frequently used in image geolocalization is Digital Elevation Map (DEM). DEMs are 3D representation of a terrain's surface. DEMs can be also called as Digital Terrain Models (DTM) or Digital Surface Models (DSM) and they represent very informative features of terrains by giving height information. Elevation models have been key inputs to many of the geolocalization methods [58, 64, 75, 83, 90] but using them mostly for rural areas like mountains or deserts. One of the standard method to use DEM for localization was to render horizon lines from the map and to match them with horizon lines extracted from query image [83]. [58] used a more novel technique to extract horizon lines in the query image by a Multi Layer Perceptron (MLP) and used local peaks as feature points to make a correspondence set between the input image and the DEM. A MLP is a basic neural network where each neuron is made of perceptrons. [75] proposed a large scale algorithm by considering contours of mountains and geometric constraints like consistent orientation to localize images of Alps in Switzerland. All previously mentioned studies used DEMs for rural areas but Ramalingam *et al* [64] considers localization for urban canyons (where street is covered by long buildings on bot sides) using an upward facing omni-skyline image and a DEM but this method was applicable only for streets dominated by skyscrapers. All these studies can make use of elevation models only if horizon line is visible.

DEMs have been used for natural environments in previously mentioned studies. Less

[3] http://www.google.com/streetview/

detailed maps like cadastral maps or 2.5D maps which are height augmented cadastral maps have been more exploited urban areas in [1–4, 6, 11, 14, 19, 56, 65, 84]. Building outlines are important clues for localization in urban. Baatz *et al* [6] used panoramic street-view images and a database of textureless 3D buildings extracted from floor plans to make urban localization. [6] approaches the problem by applying rectification based on vanishing points to remove the perspective and reducing the problem to a 2D homothety problem. Other work [14, 19] also considers panoramic images to make use of their large field-of-view to make the matching of building fragments easier with a 2D map.

Other methods [4, 56, 65, 84] make use of cadastral maps with height information such as 2.5D maps. [65] registers an image with respect to a 2.5D model by matching 3D and 2D lines and points. However, in this case a second image, which has already to be registered, is required to establish the 3D-2D correspondences. Consequently, the first image of a sequence needs to be manually annotated.

Similarly, [56] establishes line correspondences between the input image and a 2.5D map of the scene. However, due to insufficient accuracy regarding the image orientation, additionally, some kind of user interaction is required.

Similar to our approaches presented in this thesis, [84] make use of image segmentation for localization to find alignment between the 2.5D maps and the input images. [84] estimates the 3D pose by segmenting the façades in the input image and aligning them to a 2.5D map by requiring an optimization in the 6D pose space and the authors have to introduce a swarm-based optimization. The approach relies on a detailed 2.5D model than the ones used in our approaches and in addition, high resolution panoramas from Google Street View in order to provide an accurate initial geolocation. The use of panoramic images also helps the optimization as discussed above. With regular cameras, considering only the façades is often ambiguous: For example, if a street has only aligned with the buildings with similar structures, the translation along the direction of the street is not constrained.

[4] uses the same setting as we approach the camera localization. [4] uses only an untextured 2.5D map and a pose estimate as reference information. However, [4] relies heavily on the extraction of straight line segments, in particular to find the reprojections of the corners of the buildings. This step is specifically fragile, as buildings' edges do not necessarily appear as line segments in images, and additionally, also spurious segments can be extracted without corresponding to the corner of a building.

In the rest of this section, some of the methods in image-based localization will be detailed. More specifically, first, [51] will be discussed in detail to show how aerial images and ground level input images are used in a CNN to make cross-view matching. Then, a more recent approach [86] which exploits many buildings in the query image to make cross-view matching with dominant sets will be discussed.

Ground to Aerial Cross-View Matching. [51] shows a good example of usage of Siamese architectures, see Section 2.1.1.4, for making cross-view image matching between ground level Street View[3] images and 45° tilted aerial images. Figure 2.14 shows an

example of a ground level, an aerial image and aligned image pairs used by [51]. Image alignment for making pairs of ground and aerial images are done by using publicly available depth estimates.

[51] creates their dataset of size 78K image pairs by sampling in a region of $15x15$ meters from panoramic street-view images by not using more than two cross-view pairs from each street-view image. This data is used to train a network with a Siamese architecture, in Fig. 2.15 with a contrastive loss function as Equation 2.8 [29] where x and y are the input pairs, $l \in 0, 1$ is the label indicating pair is matched pair or not, $m > 0$ is the margin for unmatched pairs and D is penalization function based on the Euclidean distance between learned features for x and y. During testing of their Siamese architecture k-nearest-neighbor matching is used on the extracted feature descriptors.

$$L(x, y, l) = \frac{1}{2} l D^2 + \frac{1}{2}(1 - l) \max(0, (m - D^2)) \tag{2.8}$$

(a) (b)

Figure 2.14: Illustration of camera localization with aerial images [51]. [51] uses a Siamese architecture to match the ground-level query image with the aerial image collection. (a) Example of a query image from Google Street View[3] and matching aerial image database. (b) Example of aligned image pairs. (Figures are taken from [51] for illustration purposes.)

.

Lin *et al* [51] shows extensive performance evaluations of their system by giving precision/recall curves and comparing the proposed method with other feature descriptors such as simple CNNs or Histogram Oriented Gradients (HOG) [18].

Cross-View Image Matching with Dominant Sets. Since Siamese architectures are promising as shown by [51], a recent work by Tian *et al* [86] also makes use of them. However, they apply the Siamese network to extract features of each building in the query image where buildings are extracted by a R-CNN [67] and later they use the extracted features to create a graph of dominant sets. Fig. 2.14 shows pipeline of their system.

[86] employs Faster R-CNN [67] network to make region proposals for buildings in the query image. After completing building detection, they search for the matching buildings in the reference database with known geo-locations. The goal here is to find good feature representations and as [51], [86] adopts Siamese networks for this purpose with same contrastive loss function as in Eq. 2.8. Then, K-Nearest-Neighbors of each building are retrieved based on matching scores from the Siamese network. However, since k-nearest-

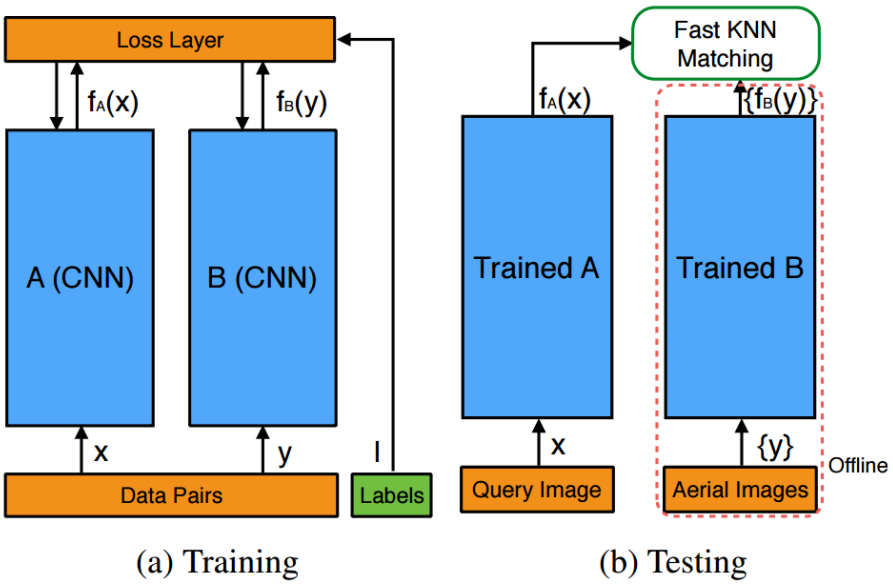


Figure 2.15: Siamese network architecture is used to train a network for cross-view image matching. (Figure is taken from [51] for illustration purposes.)

neighbors does not correspond to the correct matches most of the times, Tian *et al* [86] makes us of them to form a cluster. This is because of the assumption of near buildings in a query also have close GPS coordinates. The nearest neighbors of each query building forms a cluster and from all the selected reference buildings an undirected edge-weighted graph is created. After constructing the graph, dominant sets algorithm [62, 63] is applied to find a reference image from each cluster and the final geolocalization prediction is done by taking the mean GPS coordinates of the selected reference images from each cluster.

For experimenting their method, [86] uses their own collected data around Pittsburg, Orlando and Manhattan for aerial images and they crawl Google-Street-Map[3] for query images as in [51]. An extension of the method is also applied with the help of street-view by considering a query image with 4 different views (north, south, east, west). This extension helps a lot compared to full image matching and building matching.

In the rest of the thesis, we introduce three methods to tackle with the problem of absolute camera localization in urban environments. Our methods leverages semantic segmentation of the buildings and their parts from the input images. To this end, Chapter 3 presents how to benefit from advances in semantic segmentation to train a segmentation model for the buildings and efficiently acquire the training data needed to train the model. Later, the same chapter presents a localization method that optimizes the likelihood of the poses by leveraging the learned segmentation model and 2.5D maps. Chapter 4 further improves the optimization step of our algorithm and explore learning-based methods to optimize the current pose of the camera in an efficient way by using CNNs to converge iteratively to the ground truth pose. Finally, in Chapter 5, we exploit high-level features

such as 2D building corners and façade surface normals together with efficient minimal solvers. Our approach bridges the gap between learning-based approaches and geometric approaches.

3

Camera Localization with Semantic Segmentation and 2.5D Maps

Contents

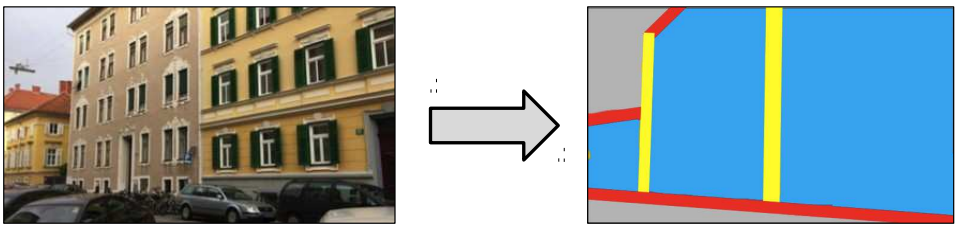
In this chapter, we discuss how to make use of semantic segmentation methods to extract discriminative features for accurate camera localization. Thus, we first show how to train a semantic segmentation model for camera localization using advances in image segmentation methods. Then, we present how to efficiently collect the training data required to train the segmentation model. Later, we show how to use the learned segmentation model in a simple but effective framework for localization.

3.1 Semantic Image Segmentation for Urban Environments

Semantic image segmentation has been proven to be useful for quite many tasks in the community. Pixel-wise segmented images help us in a better scene understanding. In our case, we leverage semantic segmentation to make inferences about the camera position and the orientation. In the rest of the thesis, semantically segmented input images and the output of the segmentation method as input to our different methods e.g. learning-based or geometrical-based. In this chapter, the semantic segmentation method that we explore to accurately localize the camera is presented.

Buildings play a crucial role in our scenario since we are interested in localization in urban environments. Our scenes mostly consists of buildings because of the nature of

urban areas. Therefore, we consider the buildings and their parts such as façades or edges as crucial cues for our localization frameworks.



(a) Urban scene (b) Rendering of the 2.5D map

Figure 3.1: Building parts define the structure of the scene. We show the structure depicted by the building in the scene by giving an example of perfect buildings' part segmentation of the input image (a) by rendering a model under the ground truth pose of the camera (b). In this case, parts are façades (blue), vertical building edges (yellow) and horizontal building edges (red).

Given a color input image, our goal is to learn a semantic segmentation model to segment building parts. For this purpose, we use a learning model called convolutional networks (CNNs). However, a drawback of CNNs for dense image prediction is the use of pooling layers where the spatial information is mostly lost. Pooling layers are dangerous for segmentation problem since they reduce the dimensionality while discarding the localization information but increasing the receptive field of the network.

This problem is attacked with the introduction of architectures called *encoder-decoder* and it is also used by [52]. The encoder part acts like convolutions and pooling layers while reducing the dimension and increasing the receptive field. On the other hand, the decoder part tries to decode and recover the spatial information lost during the encoding part. Adding skip connections from the encoder to the decoder is another trick to help the decoder recover the object details better. An example of such an encoder-decoder architecture with skip connections is used by Long *et al* [52] and the architecture can be seen in Figure 3.2.

An important factor for the segmentation networks is the size of receptive field of the convolutions. If the receptive field is small, it is hard for the network to see whole context in the image. Pooling layers increases the receptive field but reduces the resolution. A solution to this problem is proposed with *dilated (atrous) convolutions* [12, 13, 98]. Dilations are used within the kernels of convolutional layers to increase the receptive field for dense predictions. An advantage of dilated convolutions is that they do not increase number of parameters.

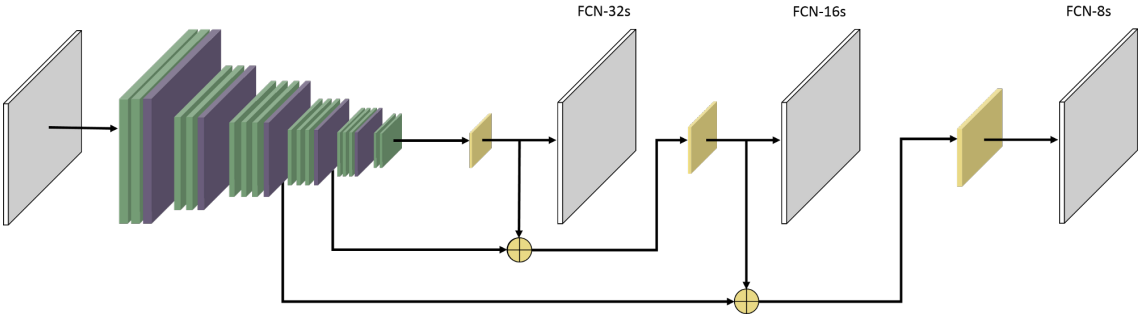


Figure 3.2: Architecture of the fully convolutional network (FCN) used in Long *et al* [52]. Input image is fed to the VGG-16 [79] layers (green and purple). Then, upsampling layers with different stride sizes with skip connections are applied. (Figure is taken from [52])

3.1.1 Exploiting Fully Convolutional Network (FCN) for Building Segmentation and Surface Normal Estimation

We choose one of the well designed off the shelf architectures called fully convolutional network (FCN) [52] as our segmentation method. Given a color input image I, we train a FCN [52] to perform semantic segmentation of the input images. FCN applies a series of convolutional and pooling layers to the input image, followed by deconvolution layers to produce a segmentation map of the whole image at the original resolution. Other recent works have a similar architecture and performance [7] and [68].

To our knowledge, FCN [52] is one of the first methods that applied conversion of hidden layers to fully convolutional layers for semantic segmentation. It also takes advantage from transfer learning and uses pre-trained VGG-16 [79] weights for initialization. VGG-16 includes fully connected layers which is a restriction for semantic segmentation task since we want to be fully independent of input image size. FCN handles this by converting fully connected layers of VGG-16 to convolutional layers by reshaping operations. Therefore, FCN can process input images of any size since the network does not include any fully connected layers.

FCN feeds forward the input through the network and uses learnable deconvolution layers to upsample the final prediction of the network. First upsampling layer of FCN upsamples the prediction with a stride size of 32 since VGG-16 applies five pooling layers and outputs a prediction that is 1/32 scale of the input image. However, the predictions done by learned upsampling layer with 32 stride size is very coarse. To be able to give more fine predictions FCN introduces skip layers to the architecture.

Another advantage of FCN is the use of skip layers to learn more about the context before decreasing the resolution by many pooling layers. FCN introduces 2 such skip layers. First one is applied after the 4^{th} pooling layer and the output of 4^{th} pooling layer is concatenated with the output of last convolution layer. The output of last convolution layer is also upsampled by a scale of 2 to make the dimensions match for the concatenation

operation. The resulted output is given to learn a new upsampling layer with stride size of 16 to have the same input size. Another skip layer is introduced to combine information after 3^{rd} pooling layer, 4^{th} pooling layer and the last convolutional layer. However, since the scale factor after the 3^{rd} pooling layer is 1/8, FCN uses an upsampling layer with a stride size of 8 in this case.

(a) (b) (c)

Figure 3.3: Vertical edges are important for camera localization in urban environments. Each row shows an example of (a) input images and their segmentations into only (b) façades and both (c) façades and vertical edges. For localization purposes vertical edges are important for scene understanding.

In our case, we aim at segmenting the façades and the vertical edges at building corners or between different façades. Vertical edges play a crucial role in our method for localization. Figure 3.3 shows the importance of the vertical edges for localization purposes. It is clear that without the vertical edges we can construct a similar structure with only façade segmentation, however, this structure created by the semantic segmentation method is not useful enough to accurately localize the camera. Therefore, we use FCN [52] to segment input images into façades, vertical edges and horizontal edges. Everything else is referred to as 'background'. We therefore consider four classes: façade, vertical edge, horizontal edges and background.

For training, we use a stage-wise procedure as [52], where we start with a coarse network (FCN-32s) initialized from VGG-16 [79], fine-tune it on our data, and then use the thus generated model to initialize the weights of a more fine-grained network (FCN-16s). This process is repeated in order to compute the final segmentation network having an 8 pixels prediction stride (FCN-8s).

The output of the trained segmentation method for a given color image I is a set of probability maps having the same resolution as I, one for each of our classes:

$$S(I) = \{P_{\text{facade}}, P_{\text{vertical\_edge}}, P_{\text{horizontal\_edge}}, P_{\text{background}}\} . \tag{3.1}$$

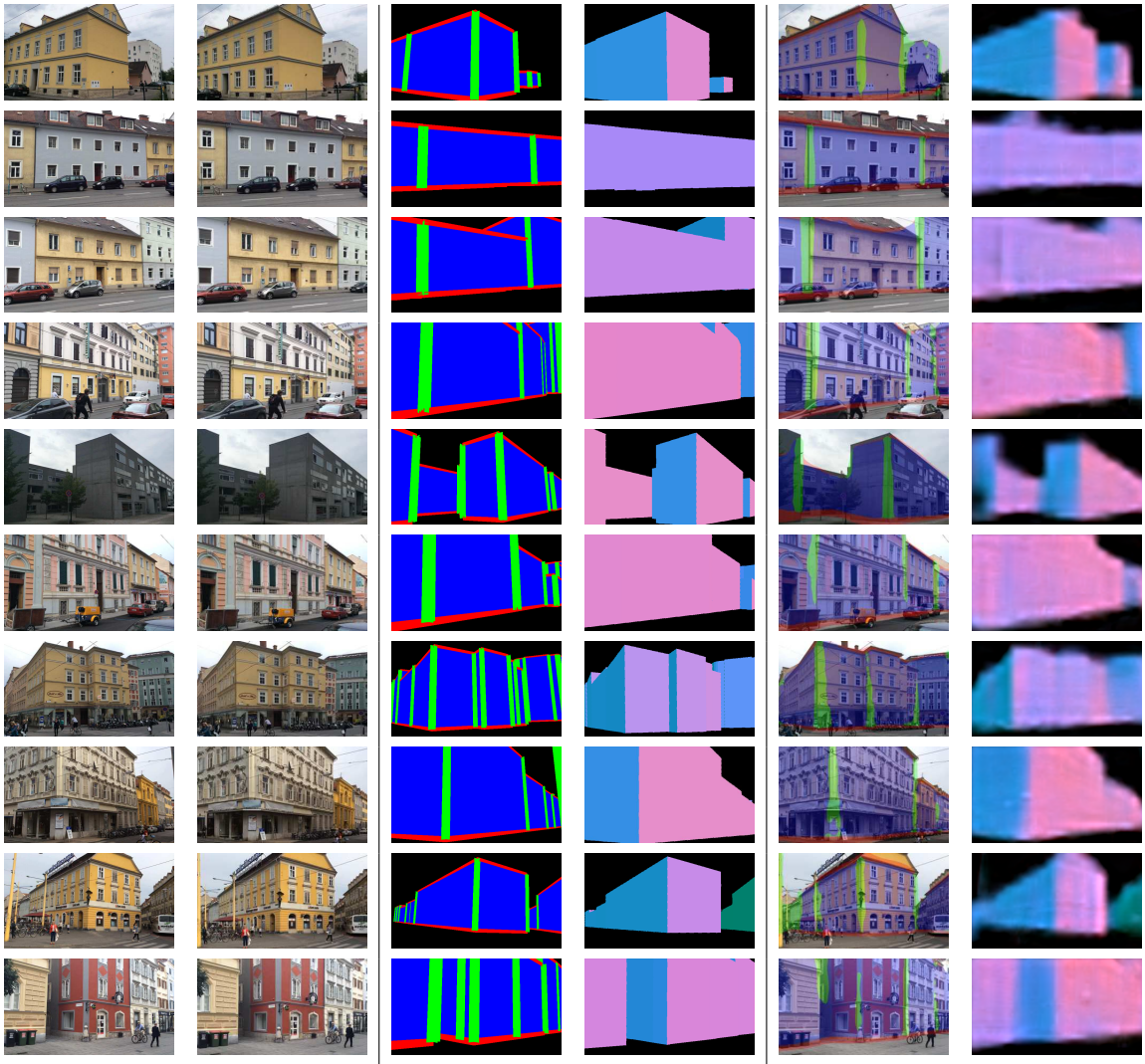


Figure 3.4: Examples of outputs for the two FCNs. First, input images (a) are vertically rectified (b) to feed as an input to our FCNs. Our FCNs are trained using the ground truth segmented images (c) and the surface normals (d). Given the input image, the first FCN outputs the buildings' part predictions (e) and the second FCN predicts the façades' surface normals (f) for each façade pixel in the image.

3.1.1.1 Using FCN for Façades' Normal Estimation

We further make use of FCN to learn more geometrical features from the buildings. These features are used in our approach in Chapter 5 to exploit high-level features. More specif-

ically, we train a second FCN to predict the buildings' orientations. We achieve this by encoding the 1D façades' surface normals $N(I_{\text{input}})$ using the registered training images and training another FCN by slightly modifying the last layer for regressing the surface normals.

At run-time, the FCN gives us a normal estimate for each pixel of the input image in the form of an angle in the range $[-90°; +90°]$. The surface normals are discretized over the color space with a linear transformation to represent the normal of a pixel in RGB color space where each predicted normal n for each pixel is transformed to color space with $(n + 1) \times 0.5 \times 255$.

We show the outputs of the new trained FCN for façades' normals and previously trained FCN for semantic segmentation of the buildings' parts in Figure 3.4. Please note that, learned representation for the façade normals will be only used for our approach in Chapter 5.

3.1.1.2 Rectification of Input Images

Vertical edges of the buildings do not appear vertical in the images due to orientation around the gravity vector. We observed that rectifying input images vertically to make the vertical edges appear vertical in the image coordinate system makes the FCN converge slightly faster. Furthermore, this step is important for our approach in Chapter 5 where we exploit building corner information and to extract accurately we need straight edges in the images. Figure 3.4 shows examples of the rectified input images.

In practice, we use the orientation of the device with respect to the gravity vector as given by the accelerometer. These two vectors are typically accurate enough in practice. Then, the input images are warped using these vectors. For the rectification of the training images, we already have a ground truth pose from the manual registration of the first frame and therefore, we can use the vertical orientation to rectify the input images.

3.2 Data Acquisition for the Semantic Segmentation Model

In the previous chapter, we explained how to train the semantic segmentation model for the buildings' parts if we have enough number of annotated training images. In this chapter, we discuss how to acquire annotated data for buildings' parts. More specifically, we want to acquire pixel-wise annotations for the following semantic classes: building façades, buildings' vertical and horizontal edges.

3.2.1 Manual Annotation of the Buildings' Parts

We first, collect a small set of images and attempt to manually annotate our image set for the corresponding object classes. However, annotating even a couple of hundred images

is costly. We use LabelMe[1] to annotate each image for our semantic classes. An example of an annotated image via LabelMe annotation toolbox is shown in Figure 3.5.

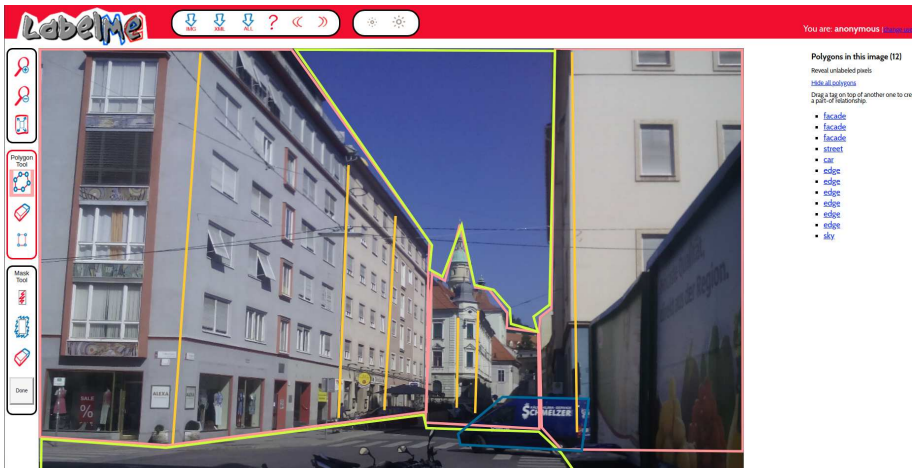


Figure 3.5: Manual pixel-wise image annotation. Each building's part in the image is annotated by drawing polygons via LabelMe [70]. The annotation for the corresponding classes in many images is required to train the networks. However, manual annotation of many images that are required to train the FCN is costly.

Manual annotation of an image usually takes around a couple minutes to ten minutes. Deep-learning segmentation methods require a large number of training images to generalize well. Therefore, we need to collect large number of images if we want to have a decent segmentation model. This process is quite costly in terms of time by considering the number of images we need to annotate and the time to annotate each image. Another, disadvantage of the manual annotation of the training images is inconsistency between the annotations due to different annotators or pixels that are missed to be annotated during the process. An example of wrong annotations for the vertical edge pixels and façade pixels are shown in red and blue circles, respectively in Figure 3.6.

3.2.2 Consistent and Fast Annotation of Buildings' Parts

Since manual annotation of building' parts is costly, we show how to collect training data more efficiently. Rather than collecting a single image for each scene, we collect video sequences to represent a scene. More formally, we record sequence of frames while translating and rotating the camera. We want to annotate each frame in an efficient manner without too much supervision as in the manual annotation of each pixel. We achieve this by using simple untextured 2.5D models and a key-point-based 3D tracking system.

[1] http://labelme2.csail.mit.edu

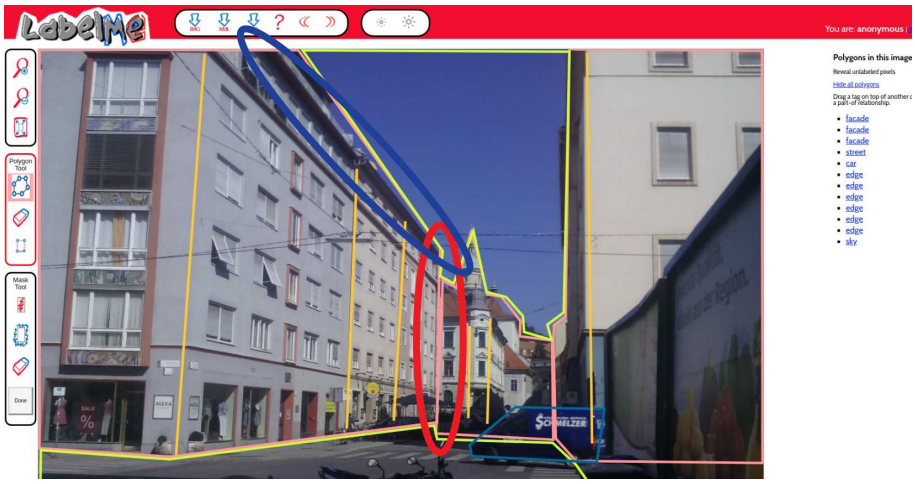


Figure 3.6: Consistent annotations for the training data is important. Manual annotations heavily depend on the annotator. The annotations might include false positives or missing annotations, e.g. false positive sky pixels which are annotated as façades or missed vertical edge pixel annotations.

3.2.2.1 2.5D OpenStreetMap Models

Accurate geographical data is quite valuable for many applications. OpenStreetMap (OSM) is a project to create such a crowd-sourced editable map of the world. The key term for the OSM is crowd-sourcing which encourages the OSM volunteers worldwide to contribute through the collection of geographical data.

Accurate geographical mapping has many usages from autonomous navigation systems to city planning or even in level of country management. Such data has been collected over the years with the help of GIS or scanning systems. However, as soon the coverage of the mapping increases accuracy of the data is most likely to be reduced. Since the human-made structures are rapidly changing in the city scale areas.

OSM contributes to the community by freely releasing their data with an open content license and by constantly updating the data. OSM community succeeds to have updated data with the help of crowd-sourced volunteers who can update the maps and add interest points as their wish.

The growth at the number of GPS enabled satellite navigation devices, such as iOS or Android devices, helps volunteers to contribute to the community. The volunteers do the mapping by using GPS traces from the devices and their experiences by overlaying their GPS traces with the bird-eye area maps and setting the required attributes such as buildings and their locations.

OSM consists of more than a 1 million contributors[2]. As it has been shown by Ciepluch *et al* [15] not all the annotators contributes to the data on the same level. Both in terms

[2]http://wiki.openstreetmap.org/wiki/Stats

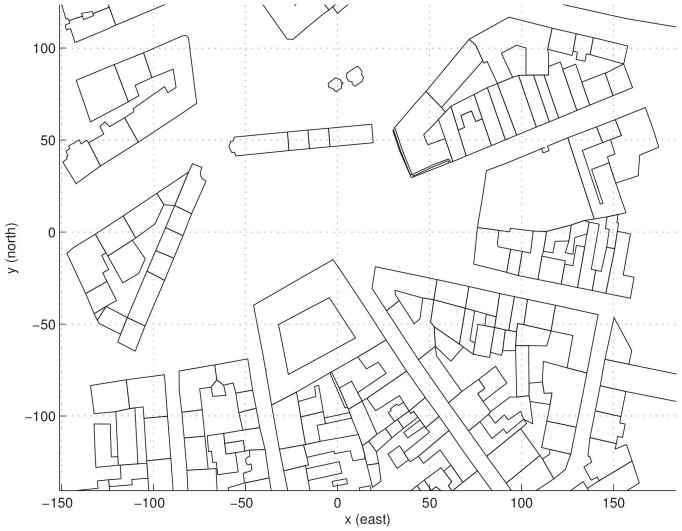


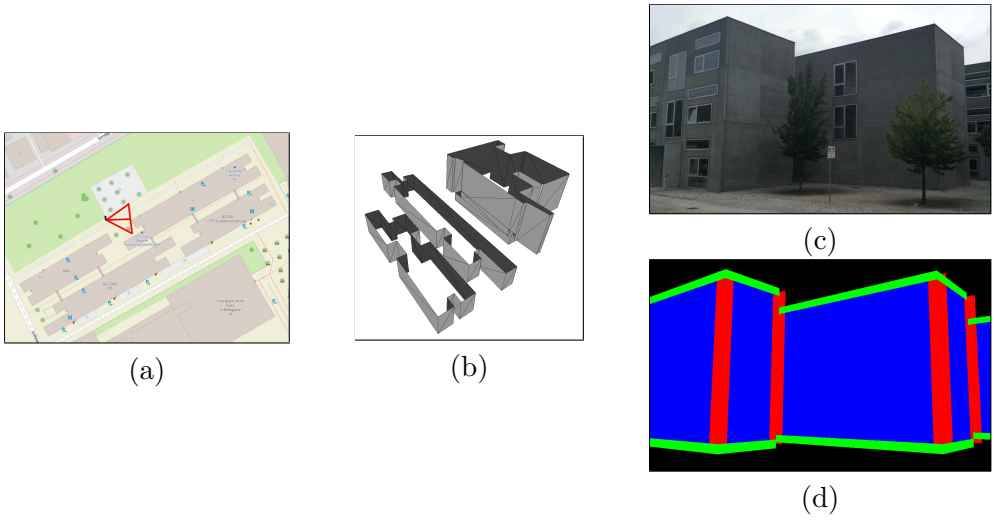
Figure 3.7: Example of building outlines from an OSM. We use light 2.5D models from OSM where the building outlines and their heights are defined. 2.5D maps has no color or texture information. Therefore, they are easy to maintain and use within fast rendering pipelines.

of amount and quality. The problem of having many non-commercialized annotators from different level of annotation experiences reduces the quality of data by having inconsistent or wrong annotations. Therefore, some studies have been conducted [26, 54, 77] to evaluate the quality of the OSM data. These studies claims that the quality of the OSM data is still far from a mapping data collected with sensing devices. However, over the years the gap between the quality of commercialized mapping dataset and the OSM dataset is closing [77].

OSM models provide us with rough information about the buildings in the urban, e.g. height of the buildings and 2D coordinates of the buildings' corners in the image or more detailed information like the roof shapes. However, the details might not be available for all the buildings depending on the annotations. We are interested in the OSM models which let us use the buildings' height information and 2D corner positions in real world. These models represent the buildings without any texture or color. These models are simple but sufficient enough to represent 3D information, therefore, they are called as 2.5D maps. We convert the 2.5D models to 3D models by elevating the buildings according to the 2D corner positions and the heights.

3.2.2.2 Key-Point-Based 3D Tracking

Given the initial pose $\tilde{\mathbf{p}}$ of the camera for the first frame and 3D model $\mathcal{M}$ of the environment, we can track the poses for the rest of the frames by using a 3D tracking framework. Please note that the limitations of the 3D tracking system is not in our focus here. We



(a) (b) (c) (d)

Figure 3.8: Manually registering the first frame into the 2.5D map. Initial pose $\tilde{\mathbf{p}}$ (a) found by registering the first frame (b) of the sequence into the 2.5D map (c) by aligning the rendering (d) of the 2.5D map with the first frame.

want to use the 3D tracking system as a tool only to track the poses for the rest of the frames. For this purpose, we adapt the 3D tracking system proposed by Arth *et al* [4].

3D trackers need models $\mathcal{M}$ to track, we gather such models from OpenStreetMap[3] (OSM). First, we create simple 3D models from the 2.5D maps. These models are not very detailed but sufficient for tracking. Then, for each sequence, we initialize the pose $\tilde{\mathbf{p}}$ for the first frame manually, and the tracker estimates the poses for the remaining frames. This allows us to label façades and their vertical edges very efficiently. Since there is a chance that 3D tracking system might fail to track the poses we apply semi-supervision during tracking and re-initialize the tracking from the frame where the tracking failed to accurately find the pose.

3.2.2.3 Training Dataset for the Semantic Segmentation Method

We recorded 95 video sequences using a smart device (an *Apple iPhone 6s*) with an average length of about 10 seconds and acquired the corresponding 2.5D maps from OSM. The first frame of the each sequence is used to manually register into the corresponding 2.5D map. Using the 3D tracker described above, we end up with a ground truth pose for each frame in the sequence. In order to ensure an accurate labeling, in particular for the vertical edges, we exploit our model rendering pipeline: We only keep frames in which the reprojection of the 3D model is well aligned with the real image, and remove those frames that suffer from tracking errors or drift. The frames with small relative displacement compared to previous frames are also removed to have a balanced set of images. In this

[3] https://www.openstreetmap.org

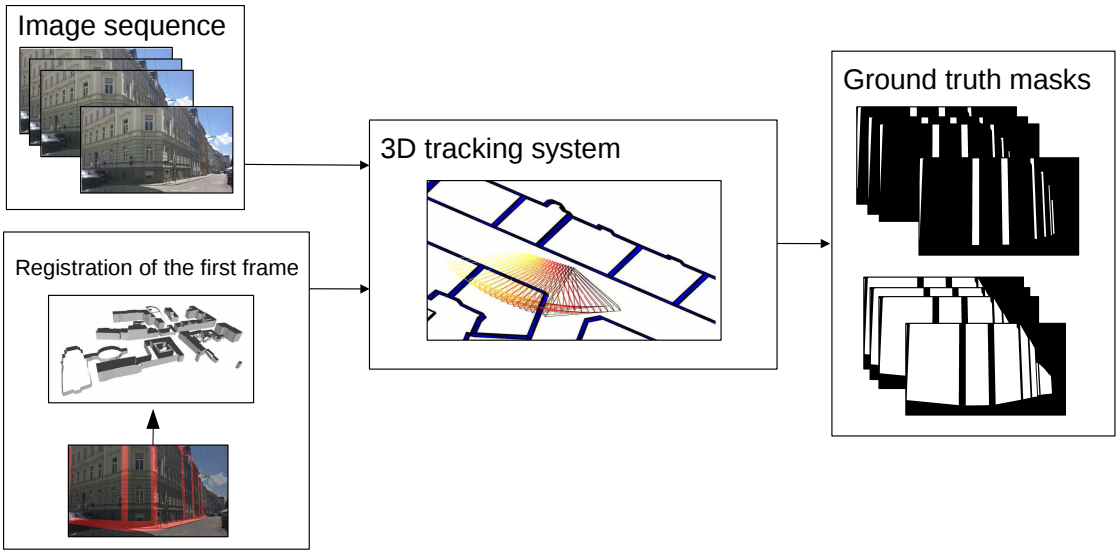


Figure 3.9: Efficiently labeling images for training data generation. We manually register the first frames of several image sequences into the 2.5D models. Then we use a key-point based 3D tracker to track the initial pose over the sequences. Given the tracked poses from the 3D tracker we can easily generate renderings of the models under the poses. This gives us the labels for the façades and their edges for all the frames.

way, we obtain a training set of around 12000 images with minimal manual effort, except for the registration of the 95 initial frames and our dataset also provides the registered poses. We further augment our dataset with horizontal mirroring of the images.

3.2.2.4 Test Dataset for Camera Localization

To evaluate our methods, we use the extended dataset of [4]. This dataset is made of 40 images captured with an *Apple iPad Air* providing an orientation and location estimate that we use as a pose prior $\tilde{\mathbf{p}}$. The images are taken in urban and suburban environments of Graz, Austria. The images were registered from manual correspondences to get ground-truth pose to calculate relative errors for the orientation and the position. Orientation error between the estimated orientation and the orientation of the ground-truth pose is calculated as sum of squared angular differences for all axes. Position error is calculated similarly to the orientation error. Mean orientation error of the sensors over all images varies from 0.25° to 49° with an average of 11.3° and the location error from $0.25m$ to $23m$ with an average of $13.4m$. Standard deviations of the orientation and the position sensor errors are 9.97° and $5.73m$, respectively.

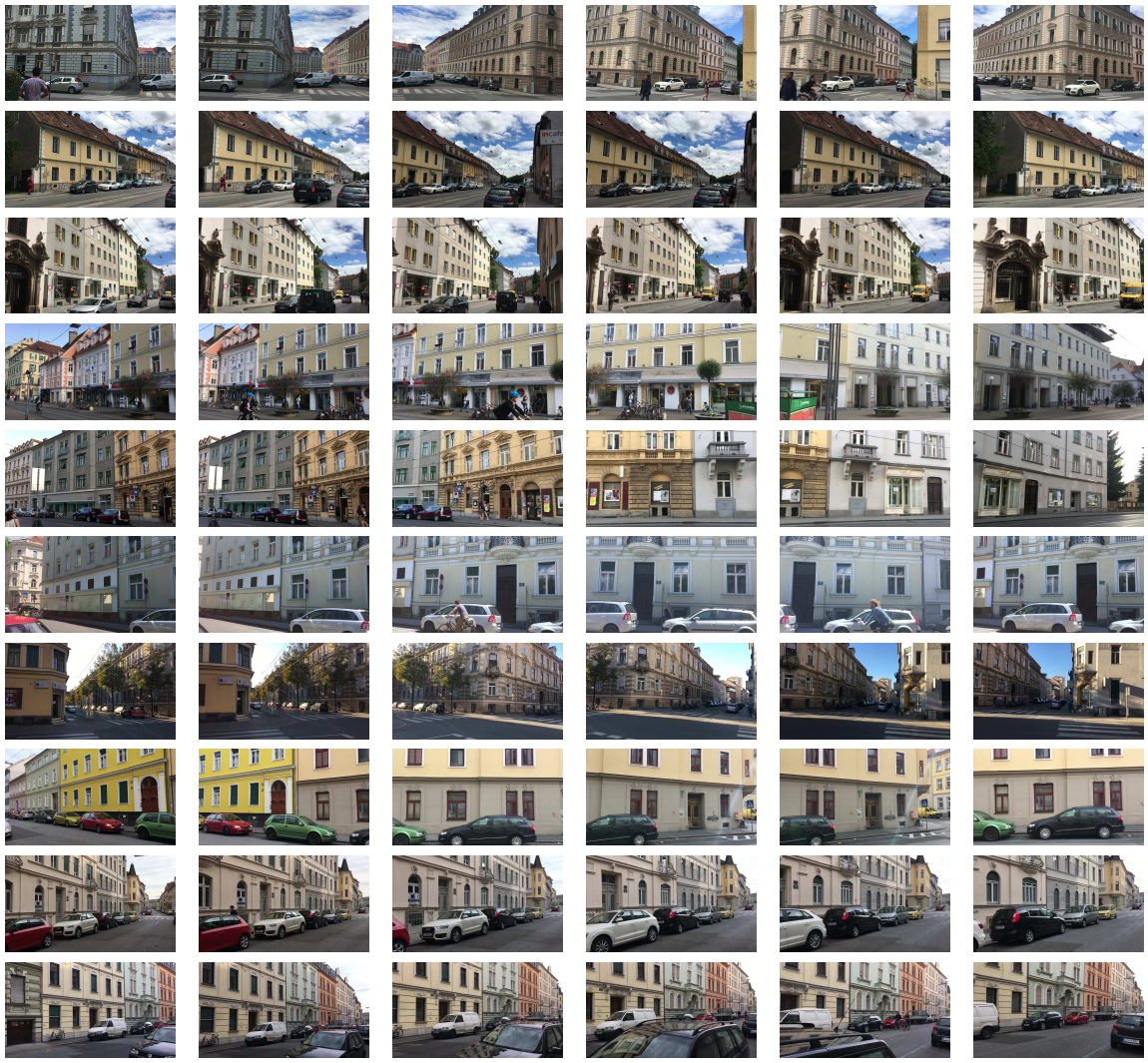
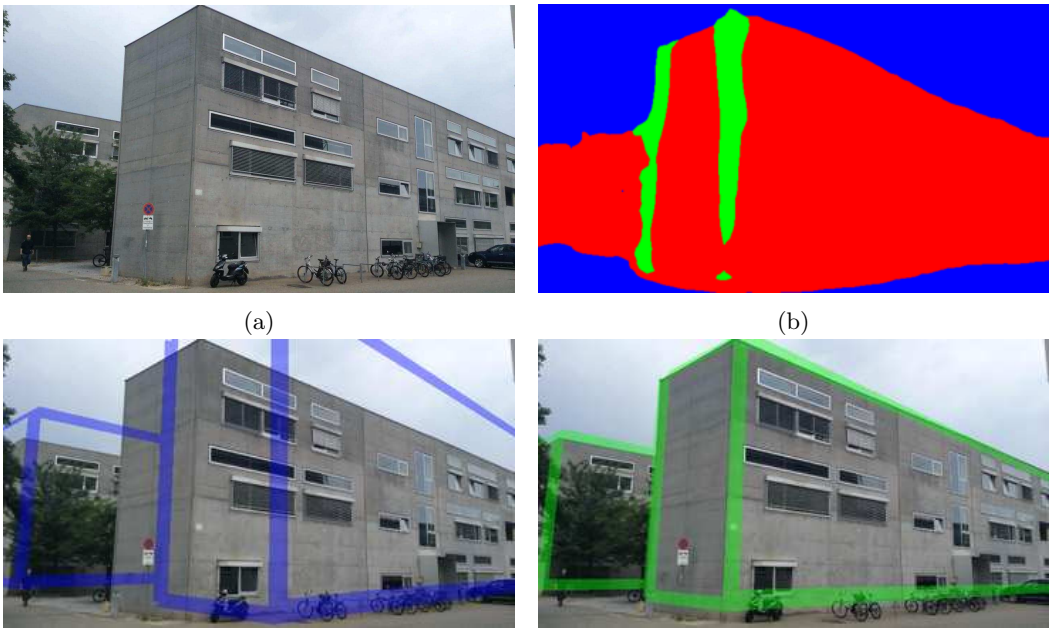


Figure 3.10: Sample frames from our data collection. Each row shows the samples from sequences of frames that is recorded for each scene. Our dataset consists of 95 such frame sequences recorded while changing the orientation and the translation of the camera. From these sequences, we end up with 11000 images in our dataset. Each of these frames are registered into the world coordinate system by using the tracker described in Section 3.2.2.2.

3.3 Combining Semantic Segmentation and 2.5D Maps for 3D Localization



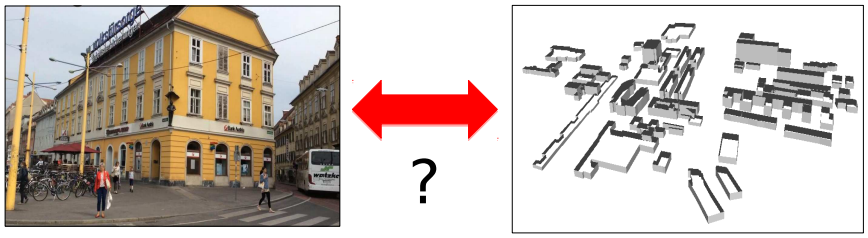
(a) (b) (c) (d)

Figure 3.11: Overview of our approach: Given an input image (a), we segment the façades and their vertical edges (b). We sample poses around the pose provided by the sensors (c), and keep the one that aligns the 2.5D map and the segmentation (d).

We have information from two different modalities, pixel values and 2.5D model of the environment. Figure 3.12 shows an example from both of the sources. We need to construct a relation between two sources to reach our goal of accurate localization.

2.5D models give us specific information about structures of the buildings, buildings' corner locations, façades' heights, *etc.* We can visualize this information by rendering the model under some pose. Figure 3.1 gives an example for rendering of the model under the ground truth pose for the given input image by coloring façades, vertical edges and horizontal edges of the buildings. If the same information can be extracted from the input image we could construct the first step of our algorithm. We use the semantic segmentation method described previously in this chapter to semantically segment the input images to help us optimize the pose.

Given a color input image I, we train a fully convolutional network (FCN) [52] to perform a semantic segmentation. FCN applies a series of convolutional and pooling layers to the input image, followed by deconvolution layers to produce a segmentation map of the whole image at the original resolution. Other recent works have a similar architecture



(a) Urban scene (b) 2.5D map

Figure 3.12: Examples of available inputs to our method. Our method uses RGB input images and a 2.5D model of the environment that's acquired given a sensor pose estimate. Our goal is to make accurate localization of the camera using exploited information from different modalities such as images and 2.5D maps. The problem is to construct the relation between the input image and the 2.5D map.

and performance [7] and [68]. Please note that the segmentation model in this chapter is trained on a smaller set of our dataset collected and it does not use horizontal edges.

We use the probability maps $S(I)$ to geo-localize an input color image I, starting from an initial estimate $\tilde{\mathbf{p}}$ of the pose provided by the sensors of the device, and a 2.5D map of the surrounding. In practice, $\tilde{\mathbf{p}}$ can be far away from the correct pose, but it still gives us a coarse estimate of the correct pose. We then look for the pose around $\tilde{\mathbf{p}}$ with the largest log-likelihood given the input image:

$$\hat{\mathbf{p}} = \arg\max_{\mathbf{p}} \mathcal{L}(\mathbf{p}) \,, \tag{3.2}$$

where $\mathcal{L}(\mathbf{p})$ is the log-likelihood:

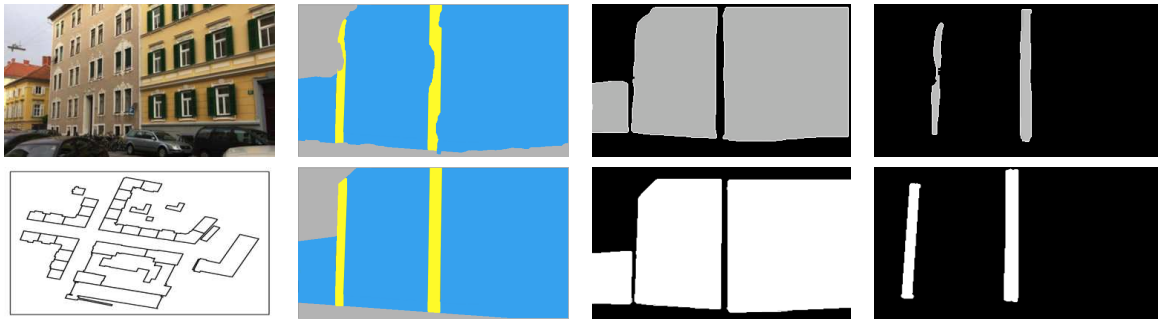
$$\mathcal{L}(\mathbf{p}) = \sum_{\mathbf{x}} \log P_{c(\mathbf{p},\mathbf{x})}(\mathbf{x}) \,. \tag{3.3}$$

The sum runs over all image locations $\mathbf{x}$; $c(\mathbf{p}, \mathbf{x})$ is the class at location $\mathbf{x}$ when rendering the model under pose $\mathbf{p}$, and $P_c(\mathbf{x})$ is the probability for class c at location $\mathbf{x}$ where P_c is one of the probability maps predicted by the semantic segmentation step in Equation (3.1).

As the log-likelihood function in Equation (3.3) is not differentiable and may have many local maximums, we sample poses around the sensor pose $\tilde{\mathbf{p}}$ on a regular grid, and keep the one with the largest log-likelihood.

3.4 Evaluation

We evaluated our approach on the test dataset described in Section 3.2.2.4, which contains images taken by an *Apple iPad Air* in urban and suburban environments of Graz, Austria.



(a) (b) (c) (d)

Figure 3.13: Illustration of similarity between the semantic meaning we depict from the input image and the semantic meaning from the renderings of the model under the ground truth pose. (a) represents the input source, (b) segmentation represented in color and (c) & (d) shows the probability maps for façades and vertical edges. Please note that in this section we do not use the horizontal edges to show the reliability of vertical edges and façade segmentations on the localization task.

The segmentation model is trained for the classes $P_{\text{facade}}, P_{\text{vertical\_edge}}, P_{\text{background}}$.

The sensor positioning errors of the dataset range from about 0.4 m to about 16.5 m, with an average error of about 8 m. The rotational errors of the gyroscopes are small, however, the orientation error around the up direction, given by the compass, can be as large as 30°. We sample the location in a squared region of 20 m×20 m with a step size of one meter in each direction. We also sample the rotation of the camera around the up direction every 3° over a range of $[-30°; +30°]$ centered on the orientation provided by the compass. At the end this step, our sampling space consists of 8400 poses to be evaluated.

Figure 3.14 shows qualitative results obtained with our method and a failure case due to a segmentation error. We quantitatively evaluated our method for both position and orientation errors. Our method decreases the mean error to 4.5 m; 56% of the images have an error below 2 m, and 78% below 5 m. Our method performs well on decreasing the orientation errors as well: After applying our method, 62% of the images have an orientation error below 2°, 75% are below 5° and the mean orientation error decreases to 4.3°.

Table 3.1 gives the time spent by the significant steps of our method. Note that the input image is only segmented once and this segmentation is used for each pose evaluation. The overall time depends on the number of poses evaluated. This number is a meta-parameter of our method: Increasing it will improve the accuracy of the final pose estimate, but the computation time will also increase linearly.

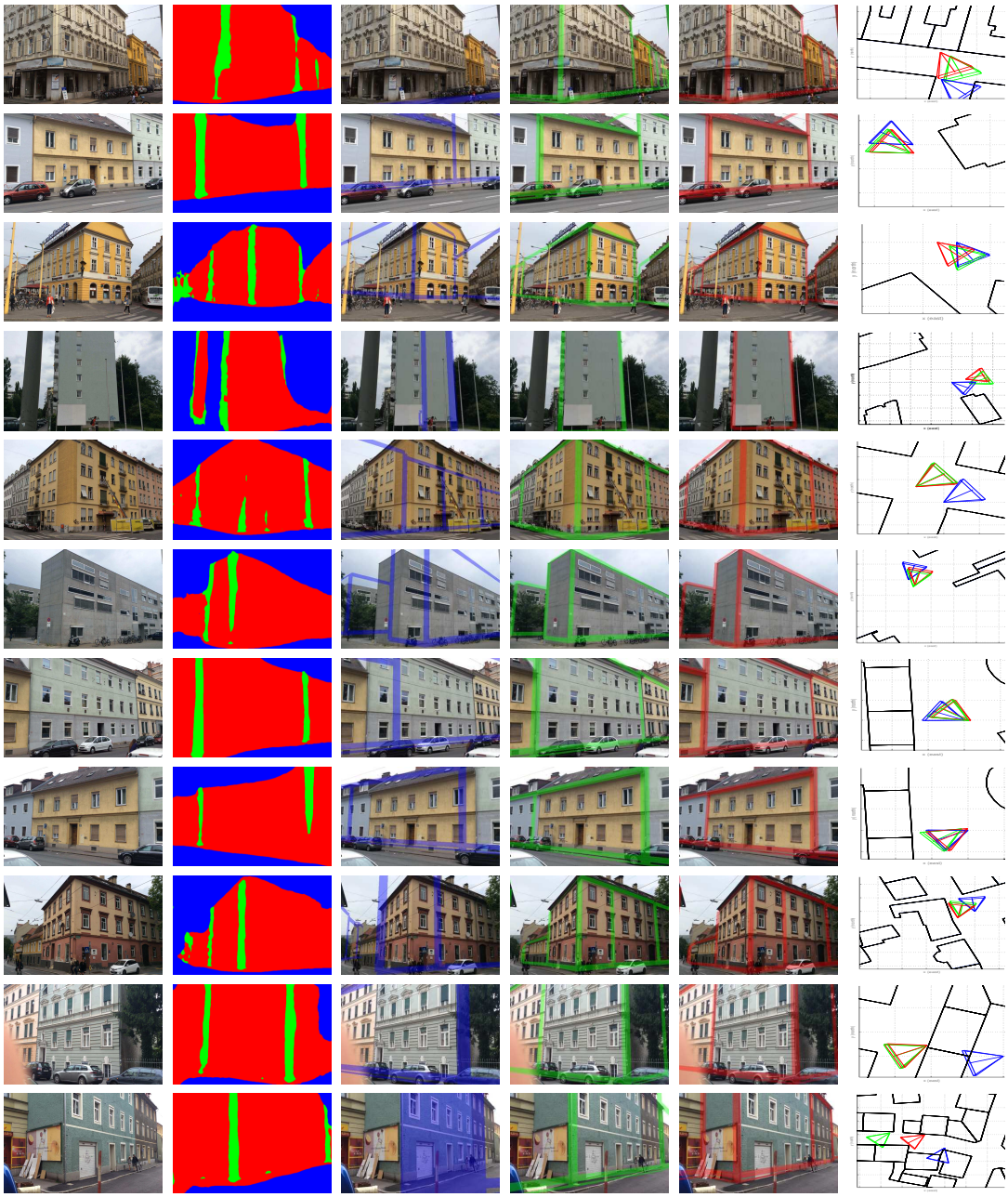


Figure 3.14: Visual comparison of poses. Each row shows the results for a test image. The last row shows a failure case. (a) Test image, (b) segmented image using our network (red: façade, green: vertical edges, blue: background), (c) rendering of the 2.5D map using the sensor pose, (d) rendering using the best pose found with our method, (e) rendering using the ground truth pose, (f) the different camera poses shown on a map (blue: sensor pose, green: pose found with our method, red: ground truth pose). The last row shows that the segmentation fails to find one of the edges, resulting in an incorrect evaluation of the model-image alignment quality.

Step	Computation Time per Frame (ms)
Semantic segmentation	120
Rendering	5
Log-Likelihood	10
Total	135

Table 3.1: Computation time for each step of our method.

3.5 Summary

In this chapter, we described an approach for image-based localization using the semantic segmentation of the input images where we optimize the pose likelihood based on a uniform pose sampling strategy.

We use the segmentation method in [52] to train our model to segment façades, vertical edges and horizontal edges. The segmentation allows us to accurately evaluate the alignments using a simple 2.5D model of the surroundings with the image. We later, showed how to collect the training data necessary to train the segmentation model efficiently by combining key-point based 3D tracking systems and 2.5D models of the environment.

Further, we exploit the 2.5D maps together with the semantic segmentation of the input image to evaluate quality of a pose by calculating the log-likelihood of the alignment between the renderings of the model and the input image. Our results show that the semantic information we exploit from the buildings are representative enough to describe a pose.

Our approach makes accurate pose estimations with an error range of a few meters and a couple of degrees, even though the model is not textured and lacks many details and might have some errors. Our approach is general and could exploit other classes such as windows or roofs if available. However, in the wild sensor errors might be high and in this case, our approach needs to evaluate many samples to accurately find the best alignment. Instead of relying on the uniform sampling strategy to optimize the pose, in the next chapter, we propose a method to optimize the pose in a more efficient way by using learning-based strategies.

4 Learning to Refine a Pose Estimate

Contents

Previously, we showed how to train the semantic segmentation model and use it for image-based camera localization. We achieved this by evaluating likelihoods of poses that are uniformly sampled around the sensor's pose estimate $\tilde{\mathbf{p}}$. Our approach is easily tunable for accurately recovering the large sensor errors with more samples from a larger pose space around $\tilde{\mathbf{p}}$. However, this strategy makes the approach inefficient when tuned for recovering large $\tilde{\mathbf{p}}$ errors. Thus, in this chapter, we explore optimization of the pose in a learning-based framework. More specifically, we design an iterative learning-based framework where our goal is to update the camera pose in each iteration to converge to the ground truth pose. We achieve this by training Deep Networks to predict the best direction to improve a pose estimate, given a semantic segmentation of the input image and a rendering of the buildings from this estimate. We then iteratively apply this CNN until converging to a good pose.

We follow the same setting as our previous approach in Chapterr̃efchap:chap00. We consider an input image I_{input} and a sensor pose estimate $\tilde{\mathbf{p}}$ is given and corresponding 2.5D map is acquired freely from OSM. Then, our goal is to geolocalize a camera in an urban outdoor scene, starting from the provided pose estimate $\tilde{\mathbf{p}}$ in a learning-based framework. More specifically, we propose an approach to refine a camera pose starting from a coarse pose estimate in an iterative fashion.

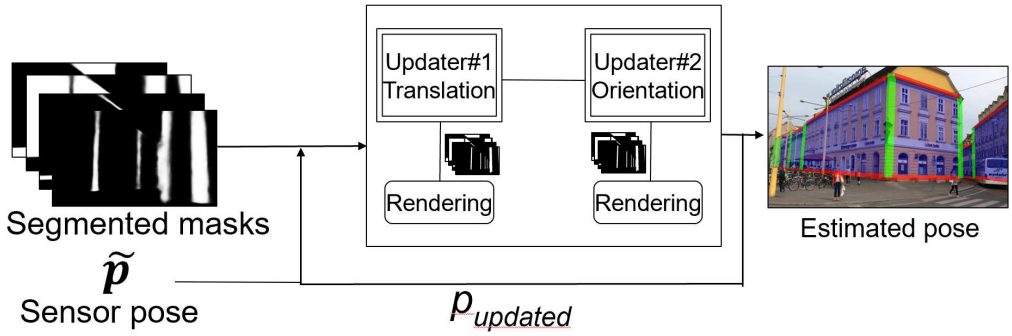


Figure 4.1: Overview of our approach for learning to refine a pose estimate. We trained two CNNs to refine the pose for the position and orientation errors in an iterative fashion. We use the probability maps predicted by FCN and a sensor pose estimate as inputs to our approach. Then, our CNNs takes the renderings of the 2.5D map under the current pose and the probability maps from the FCN to converge to a better pose and outputs a direction to update the pose at each step. The magnitude of the update is found by a line search algorithm. Our approach is applied iteratively and the pose estimate is updated at each step.

4.1 Learning to Predict a Direction for Pose Update

The initial sensor pose $\tilde{\mathbf{p}}$ gives us a coarse estimate of the pose. In practice, the angles with respect to the gravity are well defined via the sensors, giving us two angles of the camera orientation, namely the roll and pitch. As we are using a hand held device, we can also assume a fixed camera height (we use 1.6 m in practice). Thus, only three degrees-of-freedom (along the ground plane) are remaining, two for the location and one for the orientation. However, as these estimates can be very far away from the ground truth, correcting them is challenging.

To deal with this problem, we train two networks to predict directions to improve the pose estimates. The first network predicts a direction for the location. We initially tried to predict a 2D vector pointing to the correct location. This, however, did not succeed, as this problem was too difficult to learn. In fact, the length of the vector would depend on the distances to the buildings, which are lost at least to some extent because of the perspective projection.

Instead, we relax the task and solve a simpler classification problem: We discretize the directions along the ground plane into 8 possible directions, defined in the camera coordinate system. Then, given the semantic segmentation of the image and a rendering of the 2.5D map from the current estimate, we train a network CNN_t to predict the direction that improves the estimated location. We also add a class indicating that the location is already correct and should not be changed. The network CNN_t thus returns a

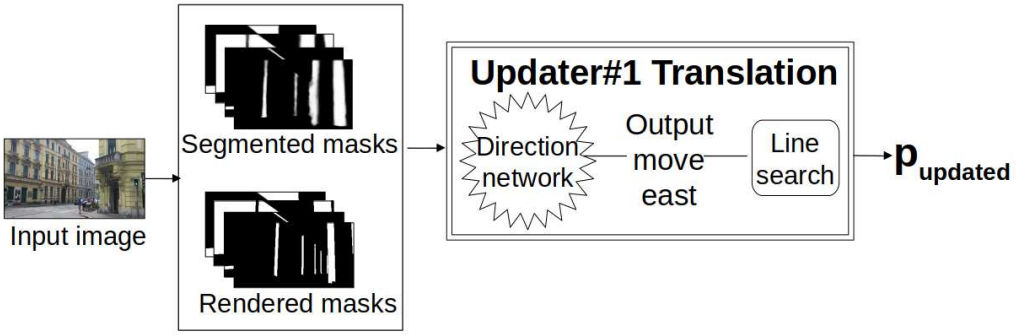


Figure 4.2: Example of our pose update step taken by our model for translation correction. Our method iteratively updates the pose both for translation and orientation correction until the networks are not confident with their predictions. Then the network outputs a prediction not to move.

9-dimensional vector:

$$\mathbf{d}_t = \mathrm{CNN}_t(R_\mathrm{F}, R_\mathrm{HE}, R_\mathrm{VE}, R_\mathrm{BG}, S_\mathrm{F}, S_\mathrm{HE}, S_\mathrm{VE}, S_\mathrm{BG}) \,, \tag{4.1}$$

where S_F, S_HE, S_VE, S_BG are the probability maps computed by the semantic segmentation for the input image I_input for the classes façade, horizontal edge, vertical edge, and background, respectively. R_F, R_HE, R_VE, R_BG are binary maps for the same classes, created by rendering the 2.5D map for the current pose estimate. Examples of these probability and binary maps are shown in Figure 4.3. The direction corresponding to the largest value in the output $\mathbf{d}_t$ is the direction predicted by the network.

In addition, we train a second network CNN_o to estimate an update for the orientation:

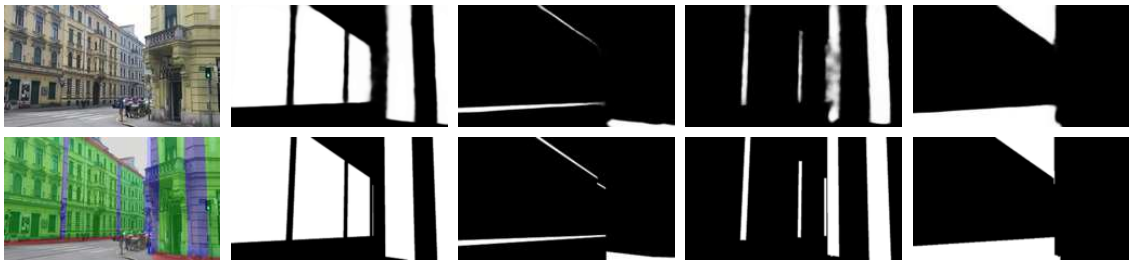
$$\mathbf{d}_o = \mathrm{CNN}_o(R_\mathrm{F}, R_\mathrm{HE}, R_\mathrm{VE}, R_\mathrm{BG}, S_\mathrm{F}, S_\mathrm{HE}, S_\mathrm{VE}, S_\mathrm{BG}) \,, \tag{4.2}$$

where the three values of $\mathbf{d}_o$ indicate if it is best to rotate the camera to the right, to the left, or do not rotate at all.

We use the same architecture for both networks, CNN_t and CNN_o. Each pair made of a probability map and a rendering for a class is fed to the network along a separate stream. Each stream consists of 2 convolutional layers with 64 and 128 filters, respectively. The sizes of the filters are 5×5 and 3×3. The outputs from the streams are concatenated and fed to fully connected layers: We use three fully connected layers with 1024, 512 and 128 units. The last layer implements a linear logistic regressor. We optimize both networks using the RMSprob [87] algorithm.

How these two networks are applied for pose estimation is described in Section 4.2 in more detail. Applying two networks solving separated problems has two main advantages: (1) We do not need to balance between translation and orientation. (2) The resulting optimization problem is easier and can be also solved on computationally less powerful

devices.



(a) (b) (c) (d) (e)

Figure 4.3: Illustrative example of the inputs to our localization networks: (a) Input image (top) and its segmentation (bottom). (b)–(e): Probability maps S_{F}, S_{HE}, S_{VE}, S_{BG} (top row) and the binary masks R_{F}, R_{HE}, R_{VE}, R_{BG} (bottom row).

4.2 Pose Estimation Algorithm

Starting from the initial estimate $\tilde{\mathbf{p}}$, we iteratively apply CNN_t and CNN_o and update the current pose after each iteration. In practice, $\tilde{\mathbf{p}}$ can be far away from the correct pose. Moreover, the networks CNN_t and CNN_o introduced above are able to predict a good direction in practice, but do not provide a magnitude.

We therefore use a line search strategy to decide the magnitude of the update. To evaluate the quality of a pose as in [4], we use the maximum log-likelihood:

$$s_{\mathbf{p}} = \sum_{c \in \{\mathrm{F,HE,VE,BG}\}} \sum_{i \in R_c} \log S_c^i \,, \tag{4.3}$$

where S_c^i is the probability at location i for class c from the semantic segmentation, and $\{i \in R_c\}$ is the set of locations that are set to 1 in the rendered binary mask R_c.

Given one direction by one of the two networks, we evaluate several poses along this direction, and keep the one that maximizes the log-likelihood in Equation (4.3). We then switch to the other network. These steps are iterated, and we stop when the two networks predict not to move any more. The overall procedure is summarized in Algorithm 1.

Algorithm 1

procedure OPTIMIZEPOSE(I_{input}, $\tilde{\mathbf{p}}$, $\mathcal{M}$)
  $S = (S_{\text{F}}, S_{\text{VE}}, S_{\text{HE}}, S_{\text{BG}}) \leftarrow FCN(I_{\text{input}})$
  $\mathbf{p} \leftarrow \tilde{\mathbf{p}}$
  repeat
    $R = (R_{\text{F}}, R_{\text{VE}}, R_{\text{HE}}, R_{\text{BG}}) \leftarrow \text{render}(\mathbf{p}, \mathcal{M})$
    $d_t \leftarrow \arg\max_i \text{CNN}_t(S, R)[i]$
    if $d_t \neq$ 'do not move' **then**
      $\mathbf{p} \leftarrow \text{lineSearch}_t(\mathbf{p}, d_t, S, \mathcal{M})$
    end if
    $d_o \leftarrow \arg\max_i \text{CNN}_o(S, R)[i]$
    if $d_o \neq$ 'do not move' **then**
      $\mathbf{p} \leftarrow \text{lineSearch}_o(\mathbf{p}, d_o, S, \mathcal{M})$
    end if
  until $d_t =$ 'do not move' and $d_o =$ 'do not move'
end procedure

procedure LINESEARCH$_t$($\mathbf{p}$, d, S, $\mathcal{M}$)
  steps $\leftarrow$ fixed set of step sizes
  for $step_j$ in *steps* **do**
    $\mathbf{p}_j \leftarrow \text{updatePose}(\mathbf{p}, d, step_j)$
    $(R_{\text{F}}, R_{\text{VE}}, R_{\text{HE}}, R_{\text{BG}}) \leftarrow \text{render}(\mathbf{p}_j, \mathcal{M})$
    $score_j = \sum_{c \in \{\text{F,HE,VE,BG}\}} \sum_{i \in R_c} \log S_c^i$
  end for
  $j \leftarrow \arg\max_j score_j$
  return $\mathbf{p}_j$
end procedure

Illustrative examples showing the progress over time from the initial sensor pose to the finally obtained pose are given in Figure 4.4.

4.3 Evaluation

To demonstrate the benefits of our approach, we first give an overview of the used benchmark and training data and then give results for artificial and real world scenarios.

4.3.1 Training and Evaluation Data

For training of these deep networks we used 50000 samples virtually generated from 95 images with known ground truth poses (see Chapter 3) and computed the semantic segmentation for each image. To generate these samples, we added random noise on the ground truth poses, sampled from a uniform distribution: We sample the location noise in the interval $[-10m; +10m]$ and the rotation noise in the interval $[-5°; +5°]$. If the dis-

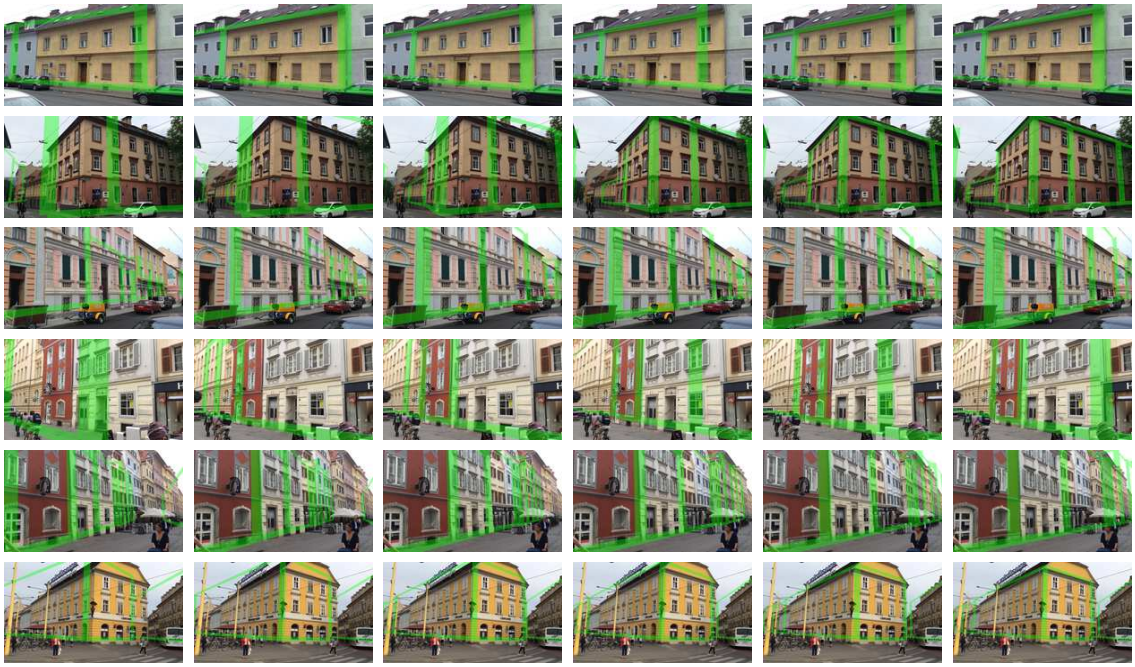


Figure 4.4: Visualization of iteration steps taken by our algorithm for several scenes. Starting from the initial pose (first column) our method keeps iterating until it reaches the final pose (last column).

tance between the ground truth pose and the random pose was smaller than a threshold the desired output was set to the 'do not move' class; otherwise, we set it to the discretized direction closest to the direction between the ground truth and the random pose.

Similar to our method presented in the previous chapter, our approach tested on the dataset described in Section 3.2.

Converging from a Close Initial Estimate Figure 4.5 shows several examples of applying our algorithm for an initial pose that is within a radius of $5m$ from the ground truth location, and an orientation in the range of $[-5°; +5°]$.

Converging from an Estimate Provided by Real Sensors Real sensors can provide measurements with very large errors, in the order of 25 meters and 50 degrees. This makes convergence hard, since comparing the rendering from such a noisy pose and the input image cannot provide meaningful information for a pose update. In such a case, our strategy is to sparsely sample initial poses around the pose predicted by the sensors. We then run our iterative algorithm from each of these initial poses and keep the best final pose according to the log-likelihood. The initial poses are sampled uniformly for each $15m$ within a radius of $30m$ from the sensor's position and for each $10°$ in the range of

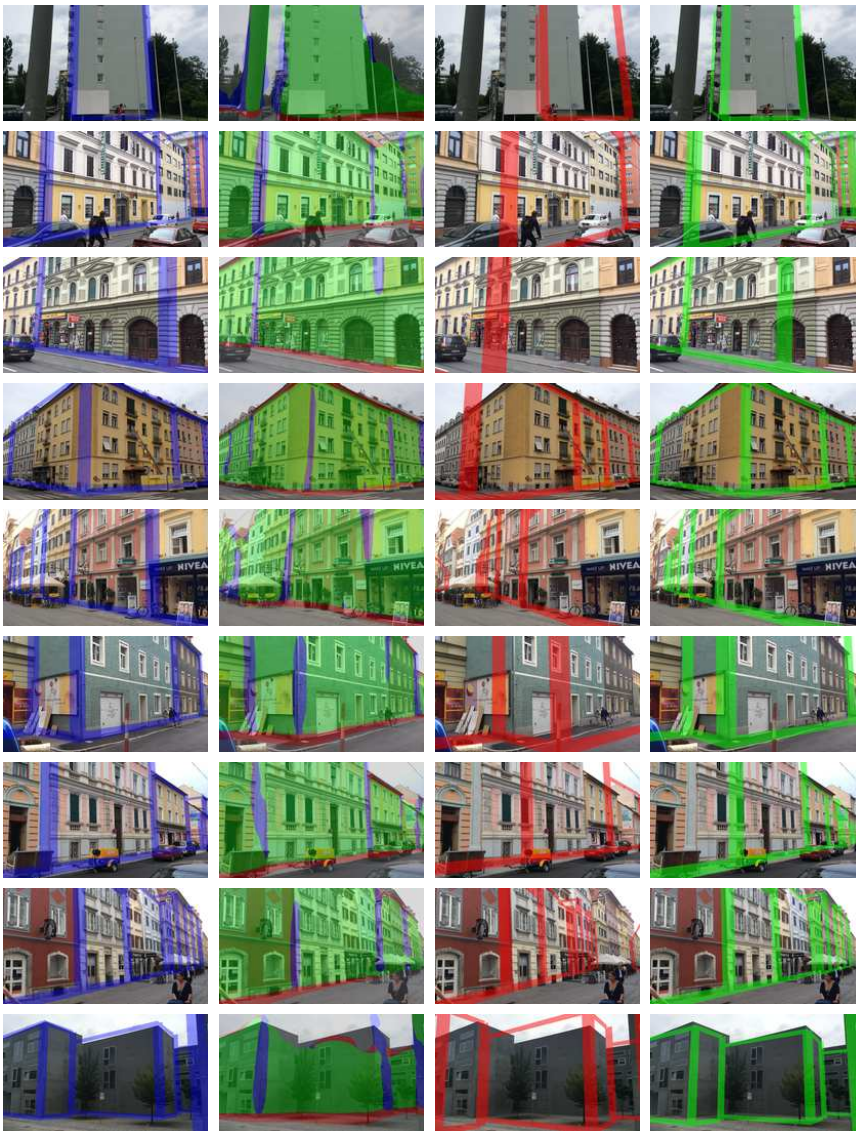


Figure 4.5: Converging from a close initial estimate. (a) Test image with the ground truth pose overlaid, (b) segmented image, (c) noisy pose rendering used to initialize our algorithm, (d) pose found with our method.

$-30°; +30°$ from the sensor's yaw angle. Therefore, uniform sampling gives us 175 initial poses to apply our localization networks and we apply four distances in our line search evaluation.

Figure 4.6 shows examples of sensor poses and our finally estimated poses. Starting from the sensor data, our method decreases the average orientation error from 11.3° to 3.2°. At the same time, the positioning error decreases from $13.4m$ to $3.1m$. Figure 4.7 shows the orientation and the position errors for each image for both, real sensor poses and poses obtained by our method.

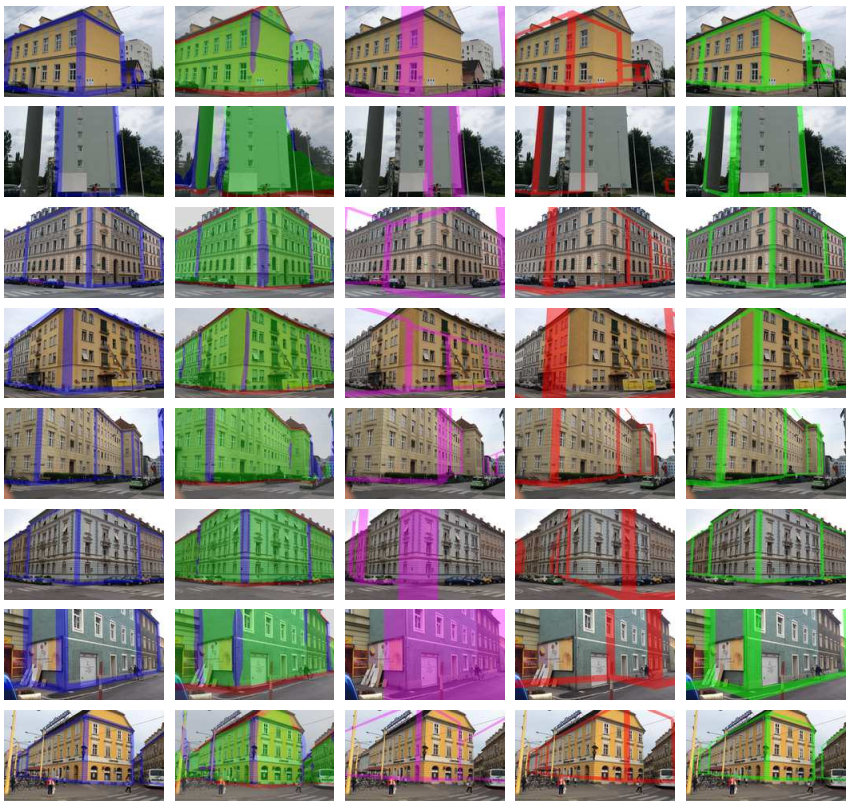


Figure 4.6: Converging from an estimate provided by real sensors. (a) Test image with the ground truth pose overlayed, (b) segmented image, (c) real sensor pose, (d) pose where the optimization started the search to find the best estimated pose, (e) final pose found by our method.

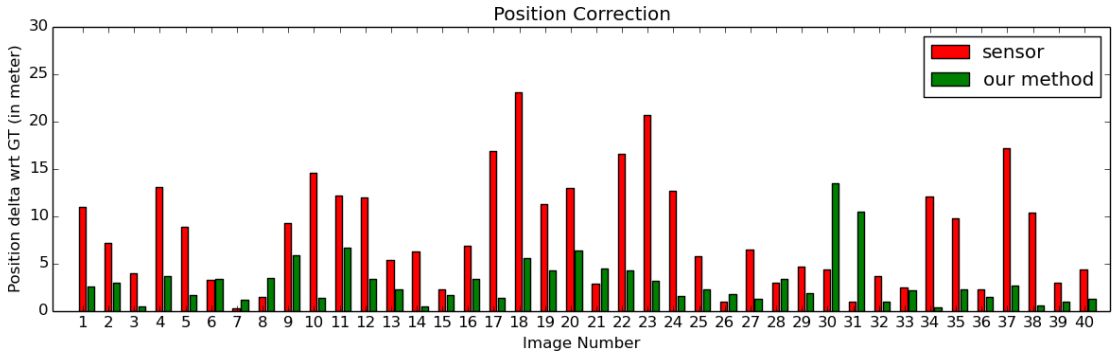


Figure 4.7: Position errors for sensor poses and poses obtained by our method.

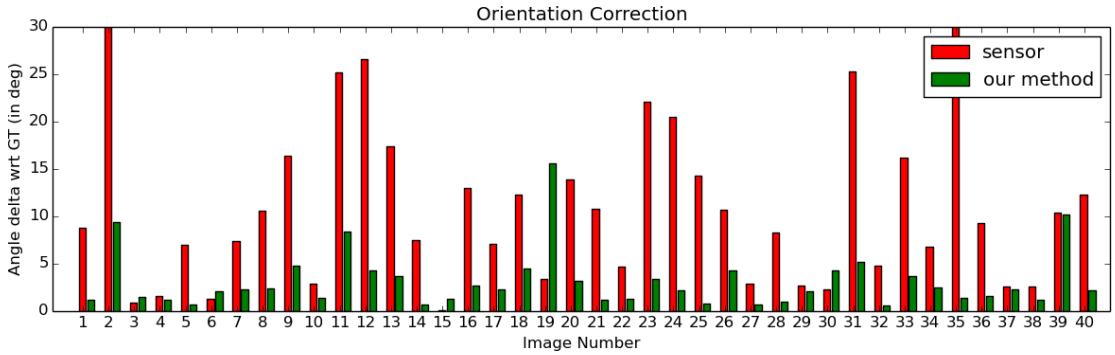


Figure 4.8: Orientation errors for sensor poses and poses obtained by our method.

4.4 Summary

In this chapter, we showed that it is possible to learn to predict good directions to refine the poses to a more accurate pose. We believe that such an approach is general: It is useful when it is not possible to differentiate an objective function as it is the case for our problem with the image likelihood, or when it is not clear which objective function should be optimized to reach a desired goal.

Another advantage of our approach is that the training set can be augmented very easily, by generating estimates around real data: In our case, we could easily sample poses around the sensor poses, but this sampling strategy will also work for other problems.

We show that our networks can efficiently correct the pose estimates where the error is relatively small. In practice, we use the uniform pose sampling strategy to accurately correct the sensor poses that have large errors. Each sampled pose is used to start our optimization for pose corrections. To correct the translation and orientation we train two different networks. We use these networks to iteratively update the pose until we find a good alignment with the semantic segmentation and the rendering of the 2.5D map under the updated pose.

Our experiments showed that due to the large sensor errors, sparse pose sampling strategy can be applied to iteratively apply our CNNs starting from each sampled pose. This strategy helped us to recover large errors and not converge through similar buildings due to similar structures. However, for practical reasons, we want to avoid large number of iterations. To this end, the next chapter focuses on direct pose estimation rather than iterative. We achieve this by defining robust geometrical high-level features for camera localization in urban environments together with robust minimal solvers to make direct estimations. Our upcoming approach bridges the gap between the learning-based and geometric approaches.

5

High-Level Feature Matching for Camera Registration

Contents

In Chapter 3, we showed how to exploit semantic segmentation of buildings and use them to make localization by evaluating the log-posteriors of uniformly sampled poses around the sensors' estimate. Further, we improved our approach using a learning-based framework in Chapter 4 to align simple untextured 2.5D maps of the environment with the input image. However, the convergence of this method can become slow since it uses uniform pose sampling strategy to recover sensor poses that have large errors. Similarly, our first approach stands on a uniform sampling of the pose space. Therefore, when the sensor error is large the previous proposed needs to be tuned to cover a large pose space and this makes them less efficient.

In this chapter, we propose a method that combines the reliability of recent advanced image segmentation methods with the efficiency and accuracy of geometric pose estimation methods. More exactly, we use Deep Learning-based segmentation [7, 52, 68] to extract the buildings' edges as in the previous chapters and in addition also their façades' normals.

We can then compute the camera pose from matches between these image features and their equivalents in the 2.5D map. To do this robustly, we consider minimal solvers [24, 25, 44, 45] to compute camera poses from minimal sets of correspondences. Approaches based on minimal solvers were introduced to work on image features including feature points to compute geometric data such as the essential matrix between two images or a

3D pose between an object model and an image. They are typically accurate, fast, and very robust, since they can be used in a RANSAC [24] loop.

Here we show that we can use this strategy with high-level features extracted using very recent methods to compute the camera pose. We introduce two minimal solvers adapted to our application: computing a 2D pose (2D translation + rotation) from 3 edge correspondences or from 2 edge correspondences plus a façade's normal correspondence. Note that we use a 2D projection model and as in Chapter 4, we can then compute a 3D pose from these 2D poses and information from the sensors. In practice, using 3 edges tends to be more accurate than using 2 edges and a normal, but it is also more likely that only 2 edges are visible rather than 3. We therefore use both minimal solvers in a RANSAC loop and keep the camera pose that provides the best likelihood computed as in [4]. Our experiments show that this is faster and more accurate than our approaches proposed in Chapter 3 and Chapter 4.

Systems for absolute pose estimation needs to be dynamic and reliably operate on unseen scenes. The 2.5D maps provides us the corners of the buildings it is possible to make correspondences with corners in the map and corners in the image if detected. However, it is not always easy to detect buildings' corners in the images due to narrow field-of-view, natural structure of the buildings or objects occluding the buildings. Vertical edges of a building is more likely to be visible in the image than the corners of the building. Therefore, we do not use 2D-3D corner correspondences but we use 1D-2D correspondences between horizontal positions of the vertical edges appear in the image and the 2D corners in the 2.5D map. Sensors provide accurate information for the orientation of the camera wrt. gravity (2 DoF) and we find fixing the camera altitude (a DoF) is a safe assumption when hand-held devices or autonomous driving scenarios are considered. We can estimate a full pose in 2D with 3 point correspondences since the problem is reduced to estimating only 3DoF. However, having 3 edges as a minimal solution is not a practical assumption. Therefore, we also approach the problem by estimating 2D rotation from a correspondence between façade normal in 2.5D model and an estimated façade normal from the image. Then, 2D translation can be estimated from two 1D-2D point correspondences. To find the correspondences from the image, we adopt recent advances in Deep Learning [7, 52, 68] to segment the image and estimate façade normals from the image.

In the remainder of this chapter, we first give an overview of our method in the next section, then Section 5.1 describes how we extract high-level features such as 2D building corners and façade normals. Later in Section 5.3 and how to use minimal solvers to finally estimate the camera pose. Then, we compare our method on the same test set used in Chapter3 and Chapter4 and, finally, we discuss our findings.

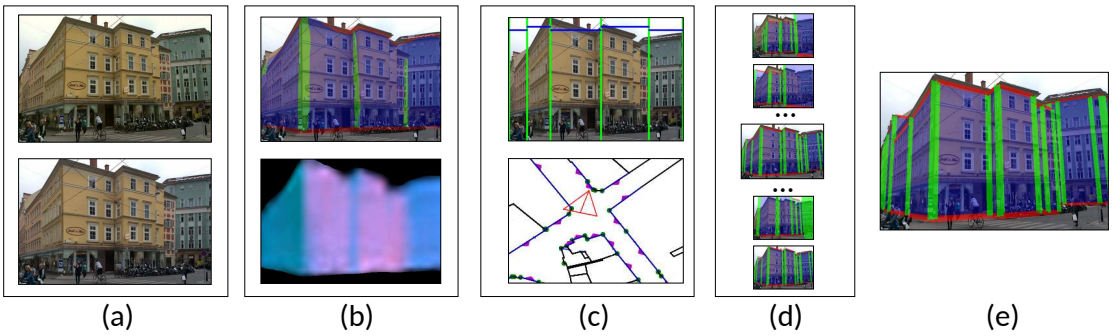


Figure 5.1: Overview of our approach: Input image (a, top) is rectified w.r.t plumb line (a, bottom). From the rectified input image, we extract façades and buildings' edges segmentation (b, top) and façades' normals (b, bottom). From the segmentation, we detect the buildings' corners and façades in the image (c, top), which are matched with corners and façades in the 2.5D map (c, bottom). From these matches, we generate possible pose hypotheses (d) using minimal solvers, and keep the pose which is most consistent with the segmentation and the normals (e).

5.1 Method Overview

Our input is a color input image I_{input} of an urban area, a prior on the camera pose, and a 2.5D map of the surrounding buildings. In practice, this prior can come from the sensors such as compass and GPS of the device that captured the image. Our goal is then to find an accurate estimate of the camera pose.

We propose a method that uses high level feature correspondences that can be constructed between I_{input} and a 2.5D map $\mathcal{M}$ of the surrounding. Unfortunately, these models come without a texture but the advantage of them is they clearly define the structure of the scene by providing information about outline and height of the buildings.

Bottom and top corners of the buildings in the image could be matched with the corners of the buildings in the $\mathcal{M}$. However, parts of the buildings that are visible in the I_{input} varies a lot depending on the camera itself, the camera pose, occlusion, size and structure of the buildings. Corners of the buildings might be completely invisible or only few might be visible in the image. It is not always possible to create a correspondence set between corners of the buildings in I_{input} and $\mathcal{M}$ in 3D, therefore we are using edges of the buildings instead of corners. Since there is no guarantee that buildings should be fully visible from top to down we are approaching the problem in 2D.

We want to estimate yaw angle θ, translation in x axis t_x and in y axis t_y. We can find the pose with three point correspondences but three correspondences might not be visible all together. We constraint the problem with having two edges and a façade visible in the I_{input}. If we could estimate the normal direction of the façades in I_{input}, then we could easily estimate the rotation by matching the normals in I_{input} and $\mathcal{M}$. Then, two edges can be used to solve for translation in two axes with a known rotation. We propose

two minimal problems. First, the problem is constrained with a facade and two edges. Please note that, we need neither the façades nor the edges to be fully visible. Secondly, we make correspondences from three edges. Even if two edges and normal of a façade is enough to estimate a pose, it is desirable to use larger point sets to reduce redundancy to noise. We find it useful to estimate the pose by using three edge correspondences in case of visibility and availability of the edges.

The rest of the chapter first describes how we extract high-level information from I_{input} in order to match I_{input} with the 2.5D map. We use simple methods to extract the buildings' corners and normals using semantic segmentation. More sophisticated methods could be developed, however, these methods were sufficient to show the effectiveness of our general approach. We then explain how to compute a camera pose from a minimal set of correspondences using minimal solvers and, finally, how we use these solvers in a RANSAC loop to robustly estimate the camera pose.

5.2 Extracting High-Level Features from the Input Image

We first vertically rectify the input images I_{input}, as previously in Section 3.1.1.2, to make the vertical edges appear as vertical in the input images. We then apply the FCN [52] (see Section 3.1) for semantic segmentation to obtain probability maps S_{F}, S_{VE}, S_{HE}, S_{BG}, for the façades, vertical and horizontal edges, and background (sky and ground plane), respectively.

Extracting the Buildings' corners Our minimal solvers rely on the coordinates along the image horizontal axis of the buildings' corners. We obtain these coordinates from S_{VE}, the probability map for vertical edges. As shown in Figure 5.2, we first compute an accumulator A_{VE} for each column of S_{VE}:

$$A_{\text{VE}}[u] = \sum_{v=1}^{H} S_{\text{VE}}[u, v] \,, \tag{5.1}$$

where H is the number of rows of the probability map and $S_{\text{VE}}[u, v]$ is the probability of image location $[u, v]^\top$ to be on a vertical edge. We then obtain the coordinates of potential corners by extracting the local extrema of A_{VE} after Gaussian smoothing. This gives us a set of column indices $U = \{u^{(i)}\}_{i=1..N_u}$ that are likely to contain buildings' corners.

Extracting the Façades' Normals Our second minimal solver also relies on the façades' normals. To estimate the façades' normals $N(I_{\text{input}})$, we use the FCN trained and described in Chapter 3. At run-time, this gives us a normal estimate in the form of an angle in the range $[-90°; +90°]$ for each pixel of the input image.

As shown in Figure 5.3, we also identify the façades in the input image as intervals along the image horizontal axis between two consecutive possible $u^{(i)}$ and $u^{(i+1)}$, extracted

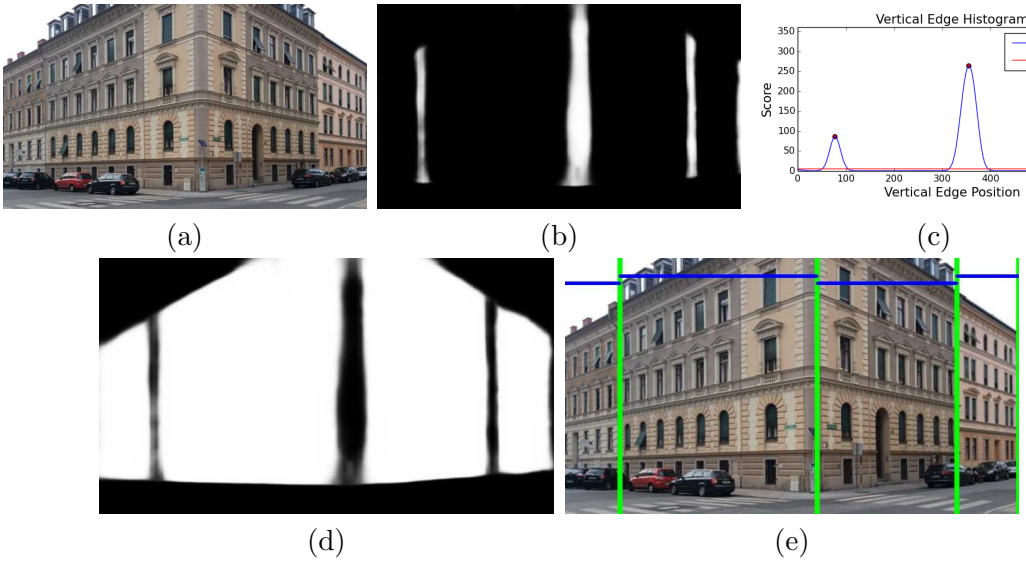


Figure 5.2: Extracting building corners and façades. (a) The input image after rectification; (b) probability map S_{VE} for the vertical edges and (c) histogram of vertical edge probabilities and its local extrema (red) we use as the locations for the buildings' corner; (e) found corners (in green) and defined façades (in blue) shown over the rectified input image.

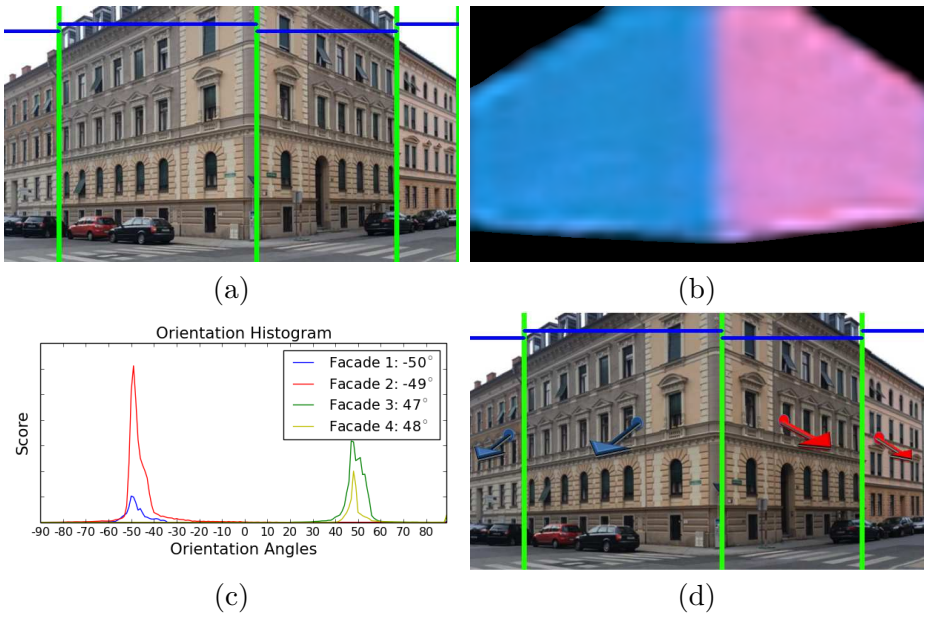


Figure 5.3: Extracting façade normal orientations. (a) Corners and façades extracted as shown in Figure 5.2, (b) surface normal estimation for the rectified input image, (c) orientation histograms for each façade, (d) final orientations assigned to each façade.

as explained previously in this section. We also consider the interval between the first column on the left of the image and the first detected corner $u^{(1)}$ and the interval between the last detected corner $u^{(N_u)}$ and the last column W on the right of the image. To summarize, the extracted façades are therefore denoted $f^{(0)} = [1; u^{(1)}]$, $f^{(i)} = [u^{(i)}; u^{(i+1)}]$ for $i \in [1..(N_u - 1)]$, and $f^{(N_u)} = [u^{(N_u)}; W]$. We denote by F the set $\{f^{(i)}\}_{i=0..N_u}$.

For each façade $f^{(i)}$ $i \in [0; N_u]$, we estimate its normal using a method inspired by the SIFT descriptor to compute a dominant gradient orientation [53]: We quantize the normal angles into bins, and each pixel between columns $u^{(i)}$ and $u^{(i+1)}$ votes for the bin corresponding to its predicted normal. The votes are weighted by the probability of the pixel to lie on a façade, as predicted in S_{F}. We finally take the normal orientation $n^{(i)}$ for façade $f^{(i)}$ as the orientation corresponding to the bin with the largest score.

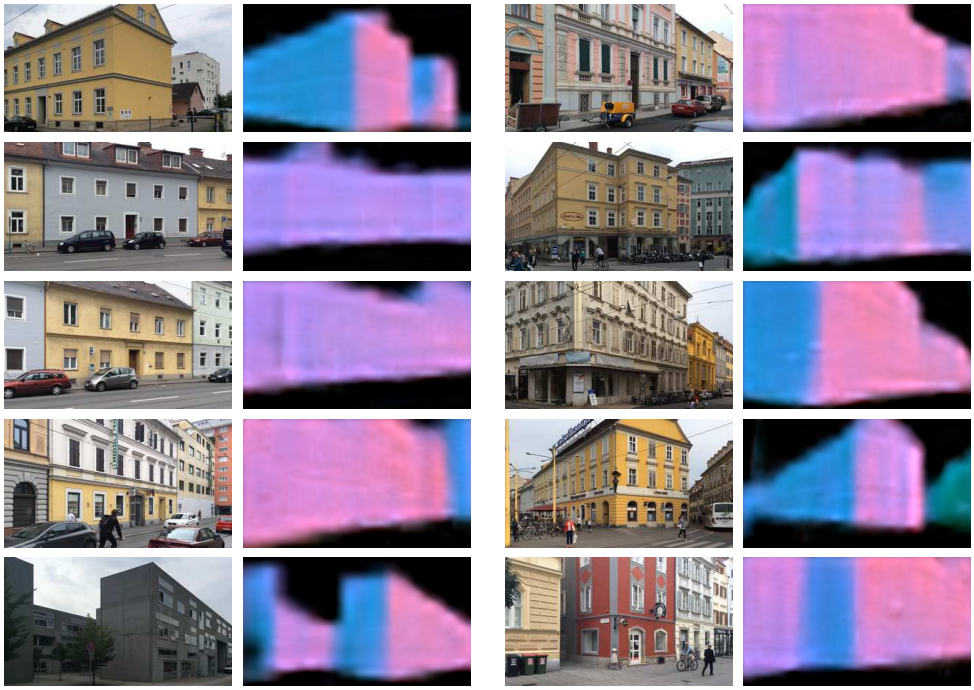


Figure 5.4: Surface normal estimation examples of building façades. Color space representing the surface normals is discretized for visualization.

5.3 Minimal Solvers

We consider two minimal solvers to compute the camera pose. These minimal solvers are relatively simple because we consider high-level features that can be extracted and match from the image and the 2.5D map.

5.3.1 Using Three Corner Correspondences

Let us consider three correspondences between buildings' corners extracted from the image and buildings' corners extracted from the 2.5D map:

$$u_1 \leftrightarrow [x_1, y_1]^\top , \quad u_2 \leftrightarrow [x_2, y_2]^\top , \quad u_3 \leftrightarrow [x_3, y_3]^\top ,$$

where $u_i \in U$ and $[x_i, y_i]^\top$ lie on the ground plane. We want to estimate the 2D location $[t_x, t_y]^\top$ of the camera on the ground plane and its orientation θ. Let us denote by $\mathrm{P}(\mathbf{t}, \theta; \mathbf{m})$ the projection of a 2D point $\mathbf{m}$ on a column of the rectified image:

$$K = \begin{bmatrix} f_u & u_0 \\ 0 & 1 \end{bmatrix} , \tag{5.2}$$

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{5.3}$$

$$\mathrm{P}(t, \theta; m) = \frac{(K(R_\theta m + t))_0}{(K(R_\theta m + t))_1} , \tag{5.4}$$

where K is the intrinsic parameters for the rectified image for the horizontal axis, R_θ is a 2D rotation matrix, and $(.)_0$ and $(.)_1$ denote the first and second coordinates of a vector, respectively. From the 3 correspondences, we get three equations of form $u_i = \mathrm{P}([t_x, t_y]^\top, \theta; [x_i, y_i]^\top)$ with $i = 1, 2, 3$. Introducing $c = \cos\theta$ and $s = \sin\theta$, we can transform these 3 equations into 3 linear equations and one quadratic equation since $c^2 + s^2 = 1$. After applying Gauss-Jordan elimination to the 3 linear equations, we get:

$$\begin{bmatrix} . & . & . & . \\ 0 & . & . & . \\ 0 & 0 & . & . \end{bmatrix} \begin{bmatrix} t_x \\ t_y \\ c \\ s \end{bmatrix} = b . \tag{5.5}$$

The last equation has the form $ac + bs = d$. We can then replace c in $c^2 + s^2 = 1$ by $c = (-bs + d)/a$, and solve for s. Once we know s, we can easily compute c, then t_x and t_y. Since there are two possible values for s, this gives two possible camera poses, but the one that is in the opposite direction of $\tilde{\mathbf{p}}$ can be discarded. For readers who is interested in more details about our solvers can find more details in Appendix A.1.

5.3.2 Using Two Corner Correspondences and One Façade Correspondence

Let us now consider two correspondences between buildings' corners extracted from the image and buildings' corners extracted from the 2.5D map and one correspondence between

a façade extracted from the image and a façade extracted from the 2.5D map:

$$u_1 \leftrightarrow [x_1, y_1]^\top , \quad u_2 \leftrightarrow [x_2, y_2]^\top , \quad f \leftrightarrow \mathcal{F} ,$$

where $f \in F$ and $\mathcal{F}$ is a façade in the 2.5D map. For the following derivations none of the extremities of $\mathcal{F}$ has to be visible in the image, which makes this solver interesting when only two edges are visible. Unfortunately, this solver tends to be less accurate than the previous one because of noise in the normal estimation.

θ is the angle between $n(f)$, the normal of the façade observed in the image as explained in Section 5.2, and $n(\mathcal{F})$, the normal of the façade in the 2.5D map model. It can thus be computed as the following:

$$\theta = \arccos(n(f) \cdot n(\mathcal{F})) . \tag{5.6}$$

Once θ is known, it is easy to compute t_x and t_y by solving a system of two linear equations.

5.3.3 Creating Pose Hypotheses

At test time, we do not know the correct correspondences between the input image and the 2.5D map. We therefore need to consider all possible correspondence hypotheses between the features extracted from the image and the buildings in the surrounding of the pose provided by the sensors. For example, the exhaustive set of correspondences for the three corner solver has the size $N_u N_\mathcal{V}(N_u - 1)(N_\mathcal{V} - 1)(N_u - 2)(N_\mathcal{V} - 2)/3!$, where N_u is the number of corners extracted from the image and $N_\mathcal{V}$ is the number of corners in the 2.5D map; typical values for N_u and $N_\mathcal{V}$ are 4 and 100 respectively.

We, however, do not have to consider all hypotheses from the exhaustive set, since the corners, and the façade in the case of the second solver, need to be visible to estimate a possible solution. We therefore pre-process the 2.5D map and create for each corner $\mathcal{V}$ a list $\mathcal{L}_\mathcal{V}$ containing all corners that can be visible simultaneously. At run-time, we create a list $\mathcal{W}$ of potentially visible corners given the pose prior provided by the sensors, made of the corners that are in or close to the field of view of this pose prior.

The possible hypotheses we need to consider can therefore be written as $u_1 \leftrightarrow \mathcal{V}_1$, $u_2 \leftrightarrow \mathcal{V}_2$, $u_3 \leftrightarrow \mathcal{V}_3$, with $(u_1, u_2, u_3) \in U^3$ and $u_1 \neq u_2$, $u_1 \neq u_3$, $u_2 \neq u_3$, and

$$\mathcal{V}_1 \in \mathcal{W}, \quad \mathcal{V}_2 \in \mathcal{W} \cap \mathcal{L}_{\mathcal{V}_1}, \quad \mathcal{V}_3 \in \mathcal{W} \cap \mathcal{L}_{\mathcal{V}_1} \cap \mathcal{L}_{\mathcal{V}_2} .$$

These constraints on $\mathcal{V}_1$, $\mathcal{V}_2$, $\mathcal{V}_3$ allow us to consider only of 1.5% the hypotheses in the exhaustive set on average in our experiments. The same approach can of course be used for the second minimal solver by considering a list, for each corner, of the façades that can be visible simultaneously.

Moreover, some poses computed by the solvers are clearly erroneous because they are very far for the prior pose. We therefore discard them without evaluating them. This step allows us to finally consider only 0.03% of the hypotheses in the exhaustive set on average

in our experiments. To select the best hypothesis among the remaining ones, we rely on their log-posterior, as explained below.

The methods presented in Section 5.3.2 and Section 5.3.1 can be applied if the correct correspondence of the vertical edges and the normals between I_{input} and $\mathcal{M}$ are given. However, we don't know the correct correspondences in practice and we; therefore create a hypotheses space $\mathcal{H}$ to represent all possible correspondences. $\mathcal{H}$ is used in a RANSAC loop to estimate an accurate pose. The hypotheses space we construct should consider both methods presented in Section 5.3 but please note that, hypotheses that use three edges can be used only for the images where we detect at least three vertical edges.

We are searching for the possible number of selections of 2 edges and a façade normal from the image and also from the map. We constraint the possible normal selections to reduce the hypotheses space. A normal from the image could only be selected if the normal's façade uses one of the selected vertical edges. Same is applicable for the selection of façade normals from the map. Let n be the number of vertical edge positions in I_{input} as detected in Section 3.1 and m be the number of visible vertices in 2.5D map.

Then, $\frac{n!}{2!(n-2)!} n_{F_I} \frac{m!}{(m-2)!} n_{F_\mathcal{M}}$ hypotheses can be constructed by considering the 2 edges and a normal selection where n_{F_I} is the number of possible normal selections for the current pair of vertical edges $n_{F_\mathcal{M}}$ is the possible normal selections for the current pair of vertices. For the selection of three edges, $\frac{n!}{3!(n-3)!} \frac{m!}{(m-3)!}$ number of pairs could be selected. All created hypotheses are combined to estimate a $\mathbf{p}$ as in Section 5.3 and evaluate the estimation in the next session.

5.4 Evaluating Hypotheses

Our methods use the log-likelihood of the image segmentation $S(I_{\text{input}})$ given the pose to evaluate the quality of the pose, considering the four classes defined in Section 5.2. Here, we also want to consider the façades' normals $N(I_{\text{input}})$ to obtain a better evaluation. We also take into account the pose prior provided by the sensors: Some images are potentially ambiguous, such as in the last row in Figure 5.8, where several buildings of similar lengths are aligned with each other. Thus, only from the image segmentation, it is not possible to decide which building the camera is facing. In such case, we would like to keep among the possible camera poses the one that is closest to the pose prior. We therefore look for the pose $\mathbf{p}$ that maximizes pose posterior:

$$P(\mathbf{p} \mid I_{\text{input}}) = P(\mathbf{p} \mid S(I_{\text{input}}), N(I_{\text{input}})) \propto P(S(I_{\text{input}}), N(I_{\text{input}}) \mid \mathbf{p}) P(\mathbf{p}) \,. \tag{5.7}$$

Taking the pixels to be independent given a pose, we have

$$P(S(I_{\text{input}}), N(\text{input}) \mid \mathbf{p}) = \prod_{\mathbf{x}} P(S(I_{\text{input}})[\mathbf{x}] \mid \mathbf{p}) P(N(\text{input})[\mathbf{x}] \mid \mathbf{p}) \,,$$

where $\mathbf{x}$ takes all the possible pixel location values in the image. Let's consider R_{F}, R_{HE}, R_{VE}, R_{BG}, the binary maps for the classes façade, horizontal edge, vertical edge, and background which are created by rendering the 2.5D map under the pose $\mathbf{p}$, and R_N the rendering of the façade normals. We assume that $P(N(I_{\mathrm{input}})[\mathbf{x}] \mid \mathbf{p})$ follows a Gaussian distribution centered on $R_N[\mathbf{x}]$ if $\mathbf{x}$ lies on a façade according to the rendering (i.e. $\mathbf{x} \in R_{\mathrm{F}}$). If $\mathbf{x} \notin R_{\mathrm{F}}$, the normal estimation does not provide a reliable estimate as it was trained only for façades and we use a constant value for $P(N(I_{\mathrm{input}})[\mathbf{x}] \mid \mathbf{p})$. We also consider that $P(\mathbf{p})$ follows an isotropic Gaussian distribution centered on $\tilde{\mathbf{p}}$. Taking the logarithm of Equation (5.7), we obtain the log-posterior:

$$s_{\mathbf{p}} = \sum_{c \in \{\mathrm{F,HE,VE,BG}\}} \sum_{\mathbf{x} \in R_c} \log S_c[\mathbf{x}] - \lambda_N \sum_{\mathbf{x} \in R_{\mathrm{F}}} (N(I_{\mathrm{input}})[\mathbf{x}] - R_{\mathrm{F}}[\mathbf{x}])^2 - \lambda_p \|\mathbf{p} - \tilde{\mathbf{p}}\|_2^2 \quad (5.8)$$

plus terms that do not depend on $\mathbf{p}$, where λ_N and λ_p are constant we manually tune to match the noise in the normal prediction and the pose prediction from the sensors. We finally keep the pose estimate provided by the minimal sensors described in Section 5.3 on the hypotheses generated as explained in Section 5.3.3 that maximizes this log-posterior. From this 2D pose estimate, we obtain a 3D pose by using the angles w.r.t. gravity from the sensors and fixing the altitude of the camera to $1.6m$ as in the previous chapter.

5.5 Evaluation

For our experiments, we use the same training and testing data as described in Section 3.2. In this section, we compare our quantitative results and the number of posterior evaluations with those obtained by our previous methods. We use the same error metric as in previous chapters (see Section 3.4) to measure the pose error. We take the $L2$ distance of the estimated position and orientation to the ground truth position and the orientation.

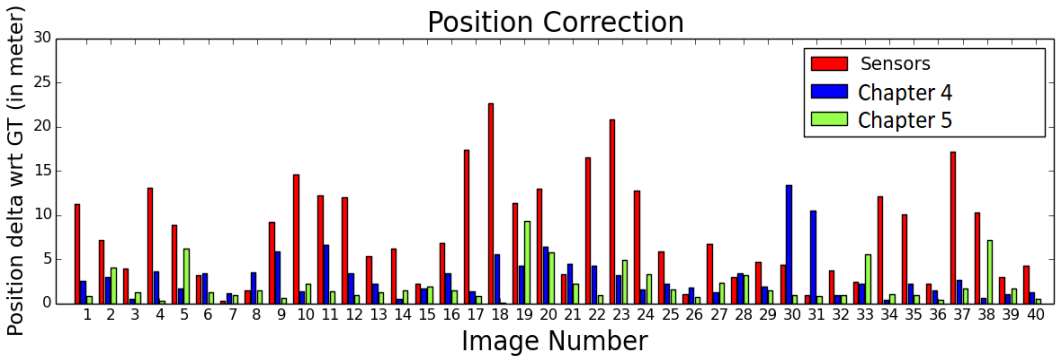


Figure 5.5: Location errors for the poses from the sensors, the poses obtained by our method presented in Chapter 4, and the poses obtained by our method.

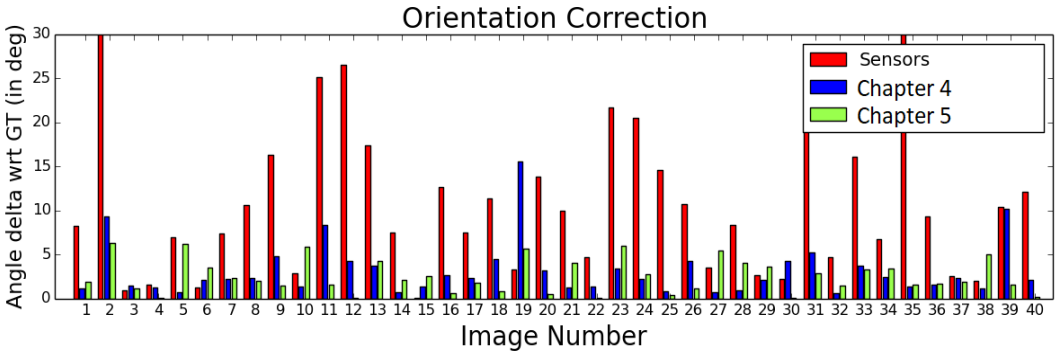


Figure 5.6: Orientation errors for the poses from the sensors, the poses obtained by our method presented in Chapter 4, and the poses obtained by our method.

	Mean Orientation Error (°)	Mean Position Error (m)
Sensor pose - $\tilde{\mathbf{p}}$	11.3	13.4
Chapter 3 - Armagan *et al* [1]	4.3	4.5
Chapter 4 - Armagan *et al* [3]	3.2	3.1
Chapter 5 - Armagan *et al* [2]	2.5	2.1

Table 5.1: Mean position and orientation errors of the poses found by our methods and sensors. Errors are wrt. the ground truth positions and orientations.

Accuracy. Figure 5.5 and Figure 5.6 compare orientation and the position estimates provided by the sensors, estimated by our methods in Chapter 3 and Chapter 4, and recovered by our current method. We apply the method on 40 test images. Our method decreases the average location error from $13.4m$ to $2.1m \pm 2.05$ and the orientation error from $11.3°$ to $2.51° \pm 1.8$, which is significantly better than the one obtained by the previous learning-based approach: $3.1m \pm 2.62$ for the location error and $3.2° \pm 3$, which represents a 30% improvement. Figure 5.8 shows some qualitative results.

Computation times. The exhaustive number of possible matches in our approach can potentially become very large: For example, it is equal to 4 million if N_u the number of corners in the image is 4 and $N_{\mathcal{V}}$, the number of corners in the 2.5D map is 100 in the case of the first solver. However, our heuristics that are described in Section 5.3.3 keeps 59722 hypotheses which we run the solvers for. The solvers are very fast as they require only $10\mu s$ for each hypothesis: Running the solvers therefore takes about $0.6s$. Out of these 59722 poses, we obtain on average 1255 poses that are close enough to the pose prior to be evaluated using the log-posterior of Equation 5.8. This 1255 pose evaluations should be compared to the 6000 pose evaluations that we need to perform on average in Chapter 4. This gives a speedup factor of about 4 for our approach.

Additionally to show efficiency of our method, we plot the number of posterior evaluations needs to be considered by the algorithm to find the final pose estimate. Numbers for all 40 test images are summarized in Figure 5.7. This number is often very small, but can become large for the most complex scenes depending on the number of vertices found in the map for the sensors' position and orientation.

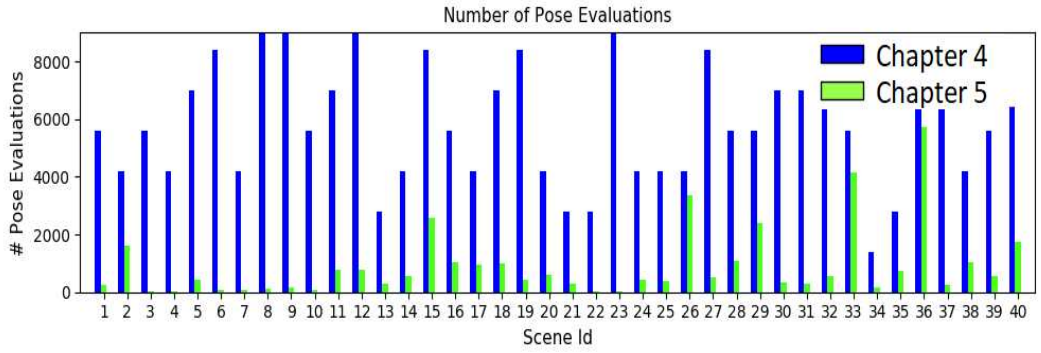


Figure 5.7: Number of posterior pose evaluations done by our methods. Our method based on high-level feature matching (green) needs less posterior evaluations compared our learning-based method (blue). Our method presented in this chapter requires less posterior evaluations than our method in Chapter 4. Please note that, the method in Chapter 4 uses sparse pose sampling strategy to start the optimization to be able to make accurate estimations when the sensor error is large. More specifically, it samples 175 poses and evaluate four different update magnitudes in the line search algorithm. Our method presented in Chapter 3 uses a dense pose sampling strategy and samples 8400 poses. Therefore, the number of posterior evaluations remains constant for all images for our approach in Chapter 3.

5.6 Summary

In this chapter, we presented an approach for camera registration where we combine the reliability of high-level feature extraction and the efficiency of well established minimal solvers.

The high-level features of buildings are exploited using semantic segmentation and the normal estimation models following the same architecture in [52]. The proposed approach introduces two accurate and efficient minimal solvers that estimates 2D camera pose by combining the high-level features, 2D building corners and a façade normal, estimated from the input image and similar features extracted from the 2.5D map to create a large hypothesis space. Our efficient solvers solve these hypotheses accurately and efficiently because they are simple and they use reliable high-level features. We discard the poses that are invalid (that lies inside a building) or far from the pose prior of the sensors. Remaining pose solutions are evaluated using the semantic information of the buildings together with the normal estimations and renderings of the 2.5D map to make posterior pose evaluations.

Urban scenes tend to have similar shaped buildings next to each other. Therefore, to prevent our method having false positive that are very similar to true positives, here we rely on the sensors and we give higher weights to the pose scores that are closer to the sensors' positions.

Our evaluations show both quantitatively and qualitatively this method is more accurate and efficient compared to the methods in the previous chapters. We believe that this is a very general and promising direction for future research, where other high-level features are simultaneously extracted from images, matched, and used to compute a pose with novel minimal solvers.

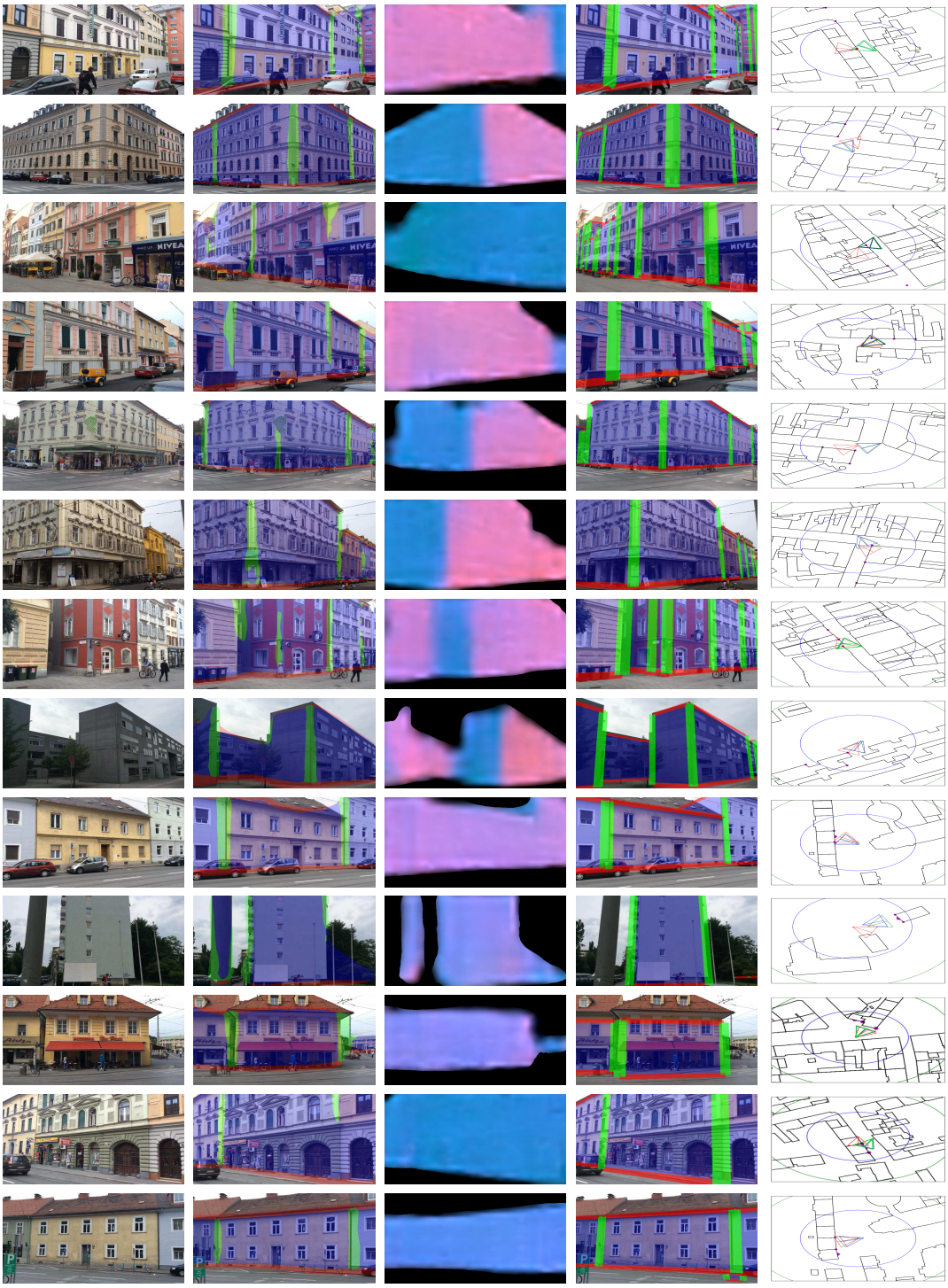
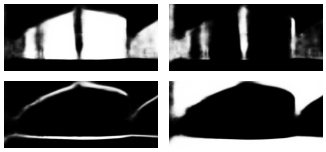


Figure 5.8: Some poses obtained using our method. (a) Input image, (b) segmentation, (c) normals, (d) rendering of the map from the pose estimated with our method, and (e) 2.5D maps with blue: ground truth pose, red: sensor pose, and green: the pose obtained with our method. The poses for the two first examples were found by the first minimal solver, the poses for the two last examples by the second mininal solver. More qualitative results are provided in appendix section.

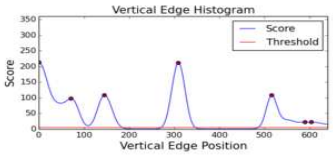
(a) Input image.

(b) Rectified input image.

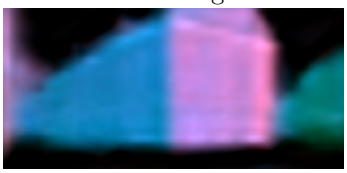
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



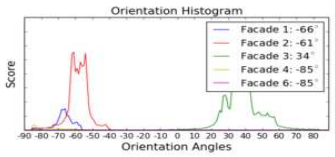
(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.



(f) Facade normal estimation.

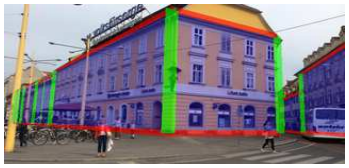


(g) Orientation histograms for detected facades.

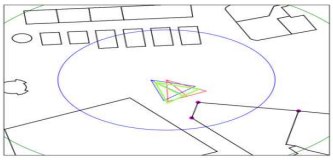
(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.



(k) Model reprojection using our method's estimate.



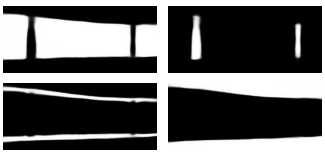
(l) Map (blue:ground truth, red:sensors, green:our method).

Figure 5.9: Intermediate results for Scene #25. Best solution was found using 3 corner correspondences.

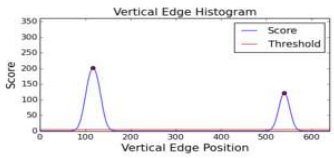
(a) Input image.

(b) Rectified input image.

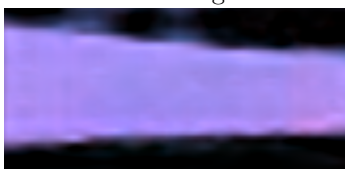
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



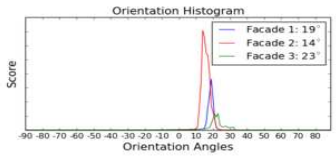
(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.



(f) Facade normal estimation.



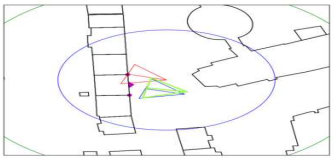
(g) Orientation histograms for detected facades.

(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



(l) Map (blue:ground truth, red:sensors, green:our method).

Figure 5.10: Intermediate results for Scene #4. Best solution was found using 2 corner correspondences and one façade normal.

6

Conclusion & Discussion

In this thesis, we proposed different strategies to tackle the problem of camera localization. Camera localization has a large context since accurate camera pose estimation is essential for recent popular applications such as AR or Autonomous Driving. We focus on one of the generic problems in this context where we build on image-based camera localization in urban environments. Our strategies follow a common setting where a single image and an initial pose estimate of the camera is available. Using the given estimate from the sensors such as a GPS, we focused on absolute camera localization problem.

The literature in image-based localization is mostly dominated by the methods based on pre-registered image collections. However, these method don't scale very well since most of the time the image set is not up-to-date or costly to do so. On the other hand, some approaches use inputs from multiple modalities like detailed 3D models of the environment. Similar to pre-registered images, detailed models of the environments are costly to maintain. In this thesis, we proposed methods exploring the use of semantic and geometrical features with the state-of-the-art segmentation methods in different frameworks. Compared to other camera localization approaches that dominate the literature, we achieve this with the help of 2.5D maps that are freely acquired from OpenStreetMap (OSM) and easy to maintain. The advantage of OSM maps is being simple therefore, they are easy to update and being crowd sourced with millions of volunteers. Therefore, the OSM maps stays widely available and up-to-date with the help of the volunteer annotators. This makes our approaches more practical and robust to changes.

We focused on urban environments where the area is mostly covered with buildings that prevents the sensors to make accurate estimations together with the other factors. Our approaches make use of the information exploited from the buildings. More specifically, we extract the façades, vertical edges, horizontal edges, corners and the façades' normal estimations. We use these features because they are robust and reliable in the urban. Relying on the robust and commonly available features is necessary to make accurate camera localization. In Chapter 3, we showed how to train state-of-the-art segmentation models to exploit such features to use in our localization approaches. Since the pixel-wise

acquisition of the training data to train such models is costly, in Section 3.2, we showed how to efficiently collect the data using 2.5D models and the 3D trackers.

Later, Chapter 3 presented an approach to show how to use the exploited information for localization purposes. Our proposed method searches for the best alignment between the semantic information from images and the semantic information acquired from the renderings of the 2.5D maps. Our localization framework remains simple but efficient enough to show such semantic information from the buildings are reliable features to make accurate localization. However, our method densely samples the pose space to search for the best alignment and this makes it inefficient when the algorithm is tuned to recover for large sensor errors.

In Chapter 4, we explored a learning-based strategy to align the semantic segmentation of the input images with the renderings of the 2.5D map. To achieve this, we model to learn a pose update direction that makes the renderings of the maps align better with the segmentations of the image. In practice, we use two such pose updaters to correct the errors for translation and the orientation errors. We found separating the update model for orientation and position made our algorithm converge faster. To be robust to larger sensor errors, we introduced sparse pose space sampling. We showed that this method is more accurate and more efficient than dense the dense sampling of the pose space. We believe such an approach is useful when the objective function is not differentiable or it is not clear which objective function should be optimized to reach a desired goal. Another possible direction is to extend our approach to work on image sequences instead of single images. We believe then LSTMs can be explored to learn a pose update direction together with the update magnitude.

In Chapter 5, we explored a more traditional technique using minimal solvers. In contrast to traditional approaches relying on the low level features, we make use of more robust high-level features with semantic and geometrical information. More specifically, we extract building corners and façade normal orientations from input images and 2.5D maps. These features are later used to combine with minimal solvers. We proposed two well established minimal solvers. The first one uses three building corner correspondences and the second one uses two corner correspondences and a façade normal correspondence between the image the map. Since, in practice, finding two building edges is a more relaxed problem we find the second solver useful for our problem. Our experiments showed that using both solvers made our system more accurate and reliable for some scenes. Moreover, this method is more accurate and efficient than our previous proposed methods and more robust to large sensor errors. The strength of our approach is relying on robust high-level features and efficient minimal solvers. Also our feature space consists of both semantic and geometrical features and this let's us introduce a more complex posterior function using the façade normals and the building corners. Our proposed method can work more efficient with a more complex hypothesis discarding strategy to reduce the hypothesis space. We believe building edges have more to contribute for imaged-based camera localization in urban environments. For example, a possible research direction is exploiting subclasses

Figure 6.1: Renderings of the convex, concave and flat vertical edges of the 2.5D map under the ground truth pose are overlayed on the input image. We show an example of the different vertical edge types such as convex (orange), concave (magenta) and flat (green) that can be further exploited for urban camera localization.

of buildings' vertical edges. We can consider vertical edges appear as 3 different types in general such as convex, concave and flat edges. Exploring more semantic and geometric features by using convex and concave edge structures of the buildings as illustrated in Figure 6.1 is a promising direction as well as exploring the appearance order of the edges. For example, probability of having 3 arbitrary edges with convex, flat and flat types is much higher than having concave, convex and convex edges. This idea can be used to increase the accuracy and reduce the hypothesis space of our approach presented in the last chapter. In practice, it is important to have efficient algorithms and therefore, we believe exploiting untextured simple maps with more high-level features such as edge types is a good direction for image-based localization.

A Appendix

In the following, we provide additional results and information for our approach mentioned in Chapter 5. We first give, in Appendix A.1, more details about the projection formulas that our minimal solvers use. We illustrate the whole process by showing intermediate steps and results for 38 additional images in Appendix A.2. Please note that our test set consists of 40 images in total and 2 of the results are showed in Section 5.5.

A.1 Minimal Solvers

In Chapter 5, we introduced two minimal solvers. The first one uses three 1D building corner form the image and three 2D model corners. The second one uses a façade surface orientation around the yaw axis and two corner correspondences. Here we detail the minimal solvers by detailing the solved equation system.

A 2D point set $U_i = [X_i, Y_i]^T, i = 1 \cdot N$ with N points is projected into 1D points $u_i, i = 1 \cdot N$ as follows:

$$u_i = K(RU_i + t) \tag{A.1}$$

where K is the 2D camera intrinsics matrix with focal length α_u and principal point u_0 (see Equation A.2), R is the 2D rotation matrix (see Equation A.3) and t is the 2D translation vector of the camera (see Equation A.4).

$$K = \begin{bmatrix} \alpha_\mathrm{u} & u_0 \\ 0 & 1 \end{bmatrix} \tag{A.2}$$

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{A.3}$$

$$t = \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{A.4}$$

Expanding the Equation A.1, we can write the projection formula for the points u_i as follows:

$$u_i = u_0 + \alpha_{\mathrm{u}} \frac{\cos(\theta) X_i - \sin(\theta) Y_i + t_x}{\sin(\theta) X_i + \cos(\theta) Y_i + t_y} \tag{A.5}$$

To simplify the problem, Equation A.5 can be written as:

$$A_i \cos(\theta) + B_i \sin(\theta) + C_i t_x + D_i t_y + E_i = 0 \tag{A.6}$$

where E is a constant and:

$$A_i = (u_0 - u_i) Y_i + \alpha_{\mathrm{u}} X_i,$$

$$B_i = (u_0 - u_i) X_i - \alpha_{\mathrm{u}} Y_i,$$

$$C_i = \alpha_{\mathrm{u}}$$

$$D_i = u_0 - u_i$$

We are using three 1D reference points u_1, u_2, u_3 and their correspondences $(X_1, Y_1), (X_2, Y_2), (X_3, Y_3)$ from the model. These points are used to create system of equations by using the Equation A.6 as follows:

$$\begin{aligned} A_1 \cos(\theta) + B_1 \sin(\theta) + C_1 t_x + D_1 t_y + E_1 = 0, \\ A_2 \cos(\theta) + B_2 \sin(\theta) + C_2 t_x + D_2 t_y + E_2 = 0, \\ A_3 \cos(\theta) + B_3 \sin(\theta) + C_3 t_x + D_3 t_y + E_3 = 0, \end{aligned} \tag{A.7}$$

Introducing $c = \cos(\theta)$ and $s = \sin(\theta)$, we can transform these 3 equations into 3 linear equations and one quadratic equation since $c^2 + s^2 = 1$. We apply Gauss-Jordan elimination to the 3 equations and get the following linear equation system:

$$\begin{bmatrix} . & . & . & . \\ 0 & . & . & . \\ 0 & 0 & . & . \end{bmatrix} \begin{bmatrix} t_x \\ t_y \\ c \\ s \end{bmatrix} = \mathbf{b}\,. \tag{A.8}$$

The last equation has the form $s = (-A'c + E')/B'$ (see Equation A.11). Then, we can use the s to replace within the equation $c^2 + s^2 = 1$ as in Equation A.12 to solve the system and find 2 solutions for c. One solution is in the opposite direction of compass orientation. We keep the solution that is in the same direction with compass. After solving the Equation A.12 for s and the Equation A.11 for c, we solve the Equation A.10 and the Equation A.9 to solve for t_y and t_x, respectively. In the rest of the Appendix, we give the full equations after applying the Gauss-Jordan elimination.

$$
\begin{aligned}
& c * (\alpha_{\mathrm{u}} * x_1 + u_0 * y_1 - u_1 * y_1) + \\
& \alpha_{\mathrm{u}} * t_x + \\
& s * (-\alpha_{\mathrm{u}} * y_1 + u_0 * x_1 - u_1 * x_1) + \\
& t_y * (u_0 - u_1) = 0 \,,
\end{aligned} \tag{A.9}
$$

$$
\begin{aligned}
& c * (-\alpha_{\mathrm{u}} * x_1 + \alpha_{\mathrm{u}} * x_2 - u_0 * y_1 + u_0 * y_2 + u_1 * y_1 - u_2 * y_2) + \\
& s * (\alpha_{\mathrm{u}} * y_1 - \alpha_{\mathrm{u}} * y_2 - u_0 * x_1 + u_0 * x_2 + u_1 * x_1 - u_2 * x_2) + \\
& t_y * (u_1 - u_2) = 0 \,,
\end{aligned} \tag{A.10}
$$

$$
\begin{aligned}
& c * (-\alpha_{\mathrm{u}} * u_1 * x_2 + \alpha_{\mathrm{u}} * u_1 * x_3 + \alpha_{\mathrm{u}} * u_2 * x_1 - \alpha_{\mathrm{u}} * u_2 * x_3 \\
& - \alpha_{\mathrm{u}} * u_3 * x_1 + \alpha_{\mathrm{u}} * u_3 * x_2 - u_0 * u_1 * y_2 + u_0 * u_1 * y_3 + \\
& u_0 * u_2 * y_1 - u_0 * u_2 * y_3 - u_0 * u_3 * y_1 + u_0 * u_3 * y_2 - \\
& u_1 * u_2 * y_1 + u_1 * u_2 * y_2 + u_1 * u_3 * y_1 - u_1 * u_3 * y_3 - \\
& u_2 * u_3 * y_2 + u_2 * u_3 * y_3) + \\
& s * (\alpha_{\mathrm{u}} * u_1 * y_2 - \alpha_{\mathrm{u}} * u_1 * y_3 - \alpha_{\mathrm{u}} * u_2 * y_1 + \alpha_{\mathrm{u}} * u_2 * y_3 + \\
& \alpha_{\mathrm{u}} * u_3 * y_1 - \alpha_{\mathrm{u}} * u_3 * y_2 - u_0 * u_1 * x_2 + u_0 * u_1 * x_3 + \\
& u_0 * u_2 * x_1 - u_0 * u_2 * x_3 - u_0 * u_3 * x_1 + u_0 * u_3 * x_2 - \\
& u_1 * u_2 * x_1 + u_1 * u_2 * x_2 + u_1 * u_3 * x_1 - u_1 * u_3 * x_3 \\
& u_2 * u_3 * x_2 + u_2 * u_3 * x_3) = 0 \,,
\end{aligned} \tag{A.11}
$$

$$
\begin{aligned}
& s^2 + s^2 * (\alpha_{\mathrm{u}} * u_1 * y_2 - \alpha_{\mathrm{u}} * u_1 * y_3 - \alpha_{\mathrm{u}} * u_2 * y_1 + \\
& \alpha_{\mathrm{u}} * u_2 * y_3 + \alpha_{\mathrm{u}} * u_3 * y_1 - \alpha_{\mathrm{u}} * u_3 * y_2 - u_0 * u_1 * x_2 + \\
& u_0 * u_1 * x_3 + u_0 * u_2 * x_1 - u_0 * u_2 * x_3 - u_0 * u_3 * x_1 + \\
& u_0 * u_3 * x_2 - u_1 * u_2 * x_1 + u_1 * u_2 * x_2 + u_1 * u_3 * x_1 - \\
& u_1 * u_3 * x_3 - u_2 * u_3 * x_2 + u_2 * u_3 * x_3)^2 / \\
& (-\alpha_{\mathrm{u}} * u_1 * x_2 + \alpha_{\mathrm{u}} * u_1 * x_3 + \alpha_{\mathrm{u}} * u_2 * x_1 - \alpha_{\mathrm{u}} * u_2 * x_3 - \\
& \alpha_{\mathrm{u}} * u_3 * x_1 + \alpha_{\mathrm{u}} * u_3 * x_2 - u_0 * u_1 * y_2 + u_0 * u_1 * y_3 + \\
& u_0 * u_2 * y_1 - u_0 * u_2 * y_3 - u_0 * u_3 * y_1 + u_0 * u_3 * y_2 - \\
& u_1 * u_2 * y_1 + u_1 * u_2 * y_2 + u_1 * u_3 * y_1 - u_1 * u_3 * y_3 - \\
& u_2 * u_3 * y_2 + u_2 * u_3 * y_3)^2 = 1 \,,
\end{aligned} \tag{A.12}
$$

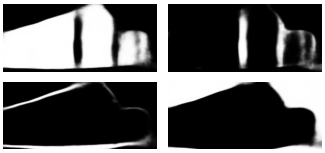
A.2 Intermediate Results

In this section, we show the intermediate results on our test set found by our method in Chapter 5.

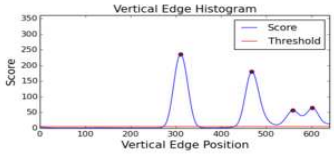
(a) Input image.

(b) Rectified input image.

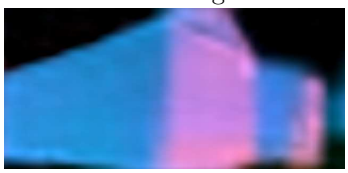
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



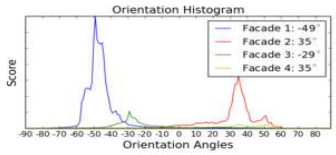
(d) Probability maps (facade, vertical edge, horizontal edge, background).



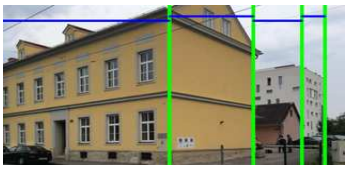
(e) Vertical edge histogram.



(f) Facade normal estimation.



(g) Orientation histograms for detected facades.

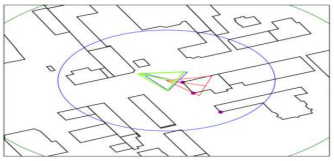


(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



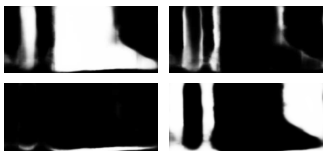
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #1 - Best solution was found using 3 corner correspondences.

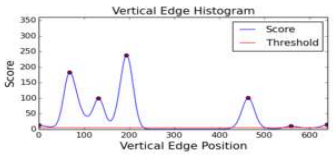
(a) Input image.

(b) Rectified input image.

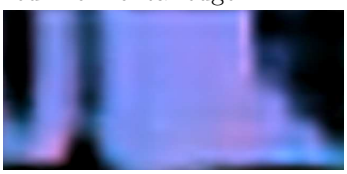
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



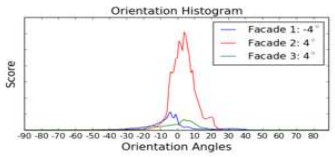
(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.

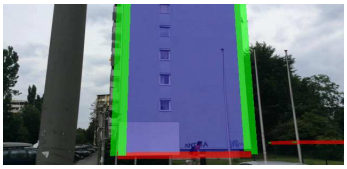


(f) Facade normal estimation.



(g) Orientation histograms for detected facades.

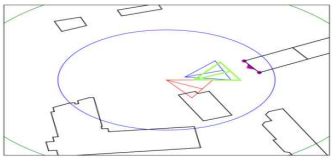
(h) Found corners (green) and defined facades (blue) shown over the rectified input image.



(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



(l) Map (blue:ground truth, red:sensors, green:our method).

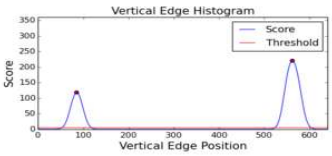
Scene #2 - Best solution was found using 2 corner correspondences and one façade normal.

(a) Input image.

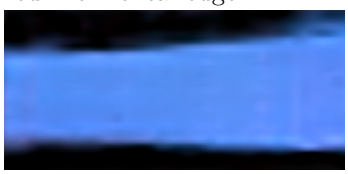
(b) Rectified input image.

(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge

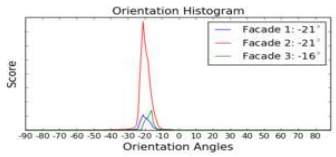
(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.



(f) Facade normal estimation.



(g) Orientation histograms for detected facades.

(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.

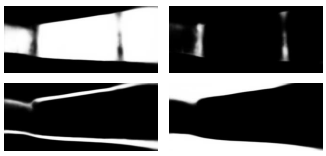
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #3 - Best solution was found using 2 corner correspondences and one façade normal.

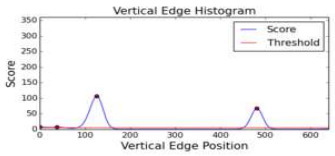
(a) Input image.

(b) Rectified input image.

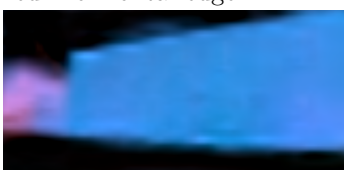
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



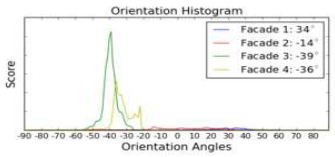
(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.



(f) Facade normal estimation.



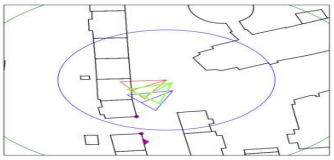
(g) Orientation histograms for detected facades.

(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



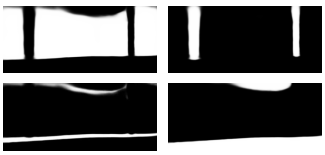
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #5 - Best solution was found using 2 corner correspondences and one façade normal.

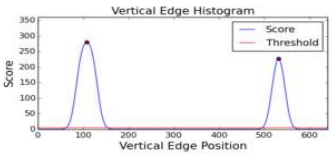
(a) Input image.

(b) Rectified input image.

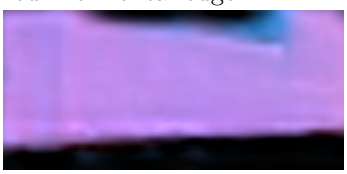
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



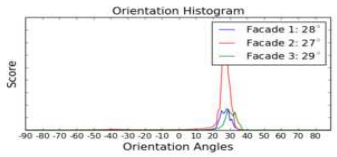
(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.



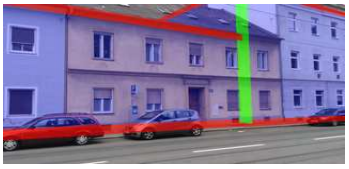
(f) Facade normal estimation.



(g) Orientation histograms for detected facades.

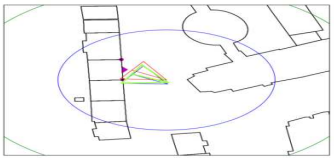
(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.



(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



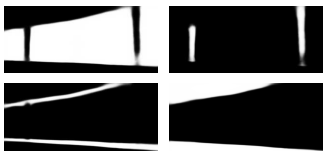
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #6 - Best solution was found using 2 corner correspondences and one façade normal.

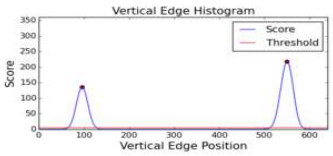
(a) Input image.

(b) Rectified input image.

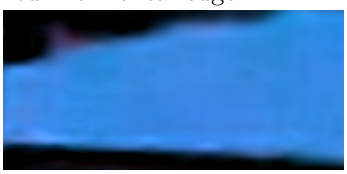
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



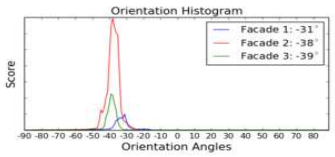
(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.

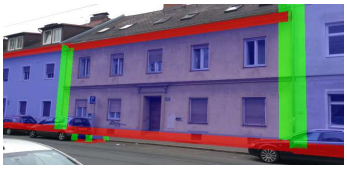


(f) Facade normal estimation.

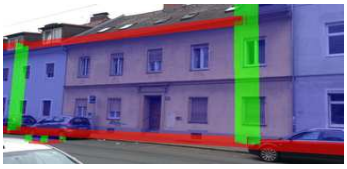


(g) Orientation histograms for detected facades.

(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

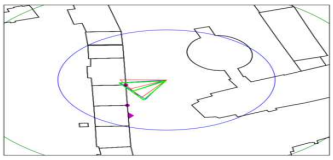


(i) Model reprojection using ground truth pose.



(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



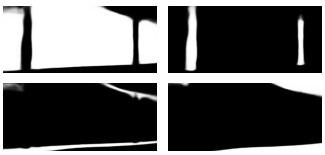
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #7 - Best solution was found using 2 corner correspondences and one façade normal.

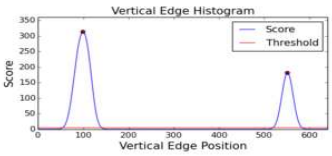
(a) Input image.

(b) Rectified input image.

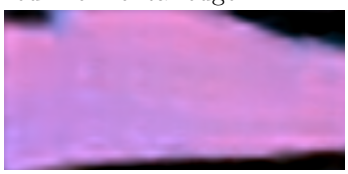
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



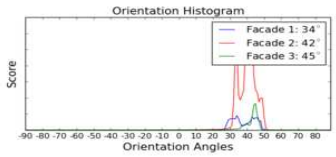
(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.



(f) Facade normal estimation.



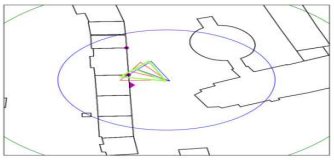
(g) Orientation histograms for detected facades.

(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



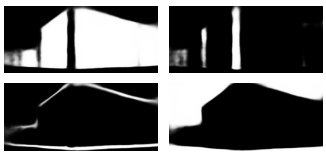
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #8 - Best solution was found using 2 corner correspondences and one façade normal.

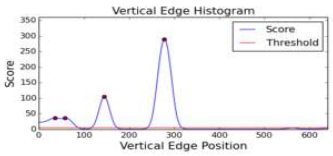
(a) Input image.

(b) Rectified input image.

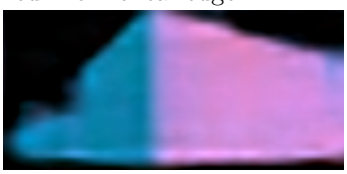
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



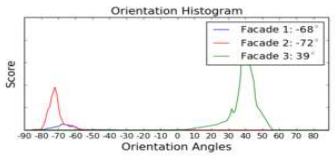
(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.



(f) Facade normal estimation.



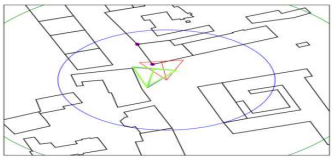
(g) Orientation histograms for detected facades.

(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



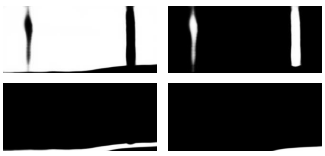
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #9 - Best solution was found using 3 corner correspondences.

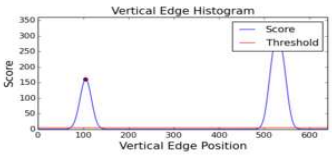
(a) Input image.

(b) Rectified input image.

(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge

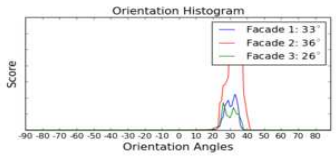


(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.

(f) Facade normal estimation.



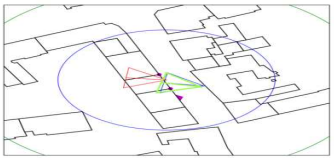
(g) Orientation histograms for detected facades.

(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



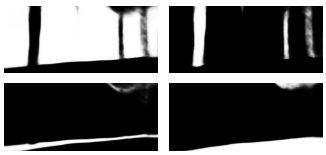
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #10 - Best solution was found using 2 corner correspondences and one façade normal.

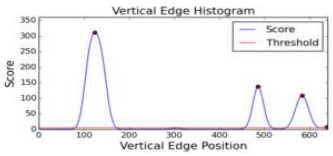
(a) Input image.

(b) Rectified input image.

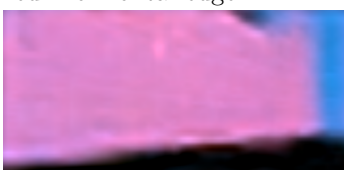
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



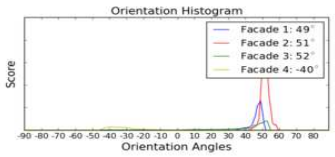
(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.



(f) Facade normal estimation.



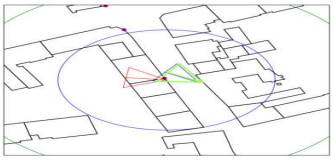
(g) Orientation histograms for detected facades.

(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



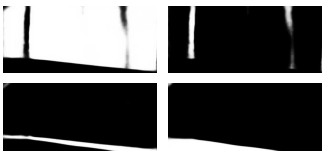
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #11 - Best solution was found using 3 corner correspondences.

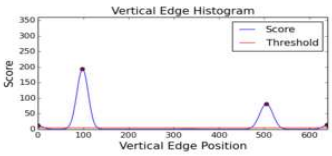
(a) Input image.

(b) Rectified input image.

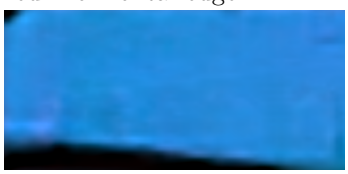
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



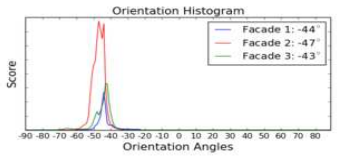
(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.



(f) Facade normal estimation.



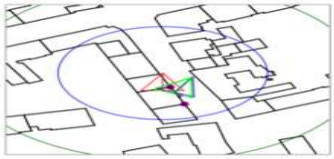
(g) Orientation histograms for detected facades.

(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



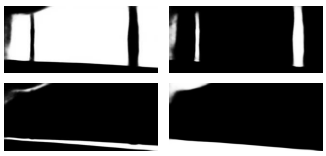
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #12 - Best solution was found using 2 corner correspondences and one façade normal.

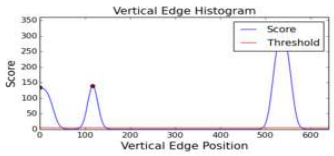
(a) Input image.

(b) Rectified input image.

(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge

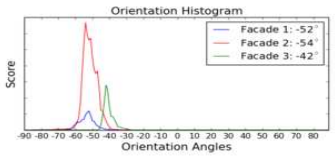


(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.

(f) Facade normal estimation.



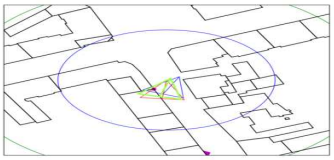
(g) Orientation histograms for detected facades.

(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



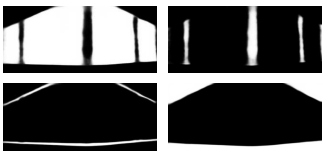
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #13 - Best solution was found using 2 corner correspondences and one façade normal.

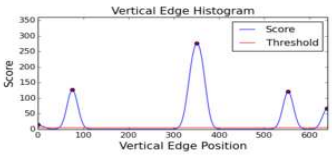
(a) Input image.

(b) Rectified input image.

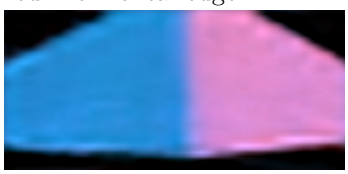
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



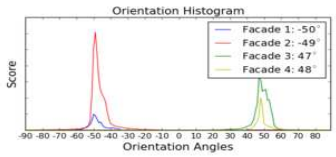
(d) Probability maps (facade, vertical edge, horizontal edge, background).



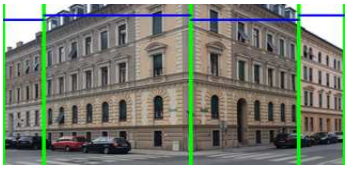
(e) Vertical edge histogram.



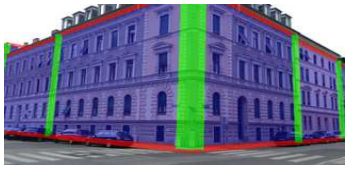
(f) Facade normal estimation.



(g) Orientation histograms for detected facades.

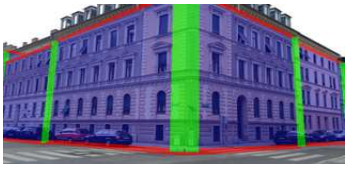


(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

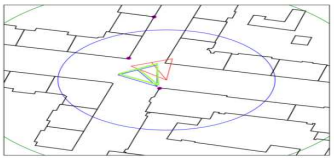


(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.



(k) Model reprojection using our method's estimate.



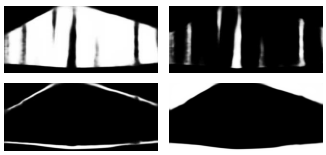
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #14 - Best solution was found using 3 corner correspondences.

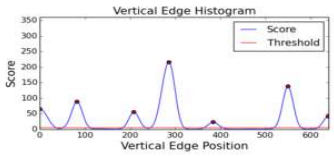
(a) Input image.

(b) Rectified input image.

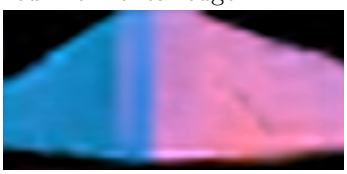
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



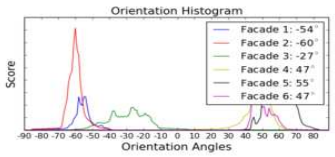
(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.



(f) Facade normal estimation.

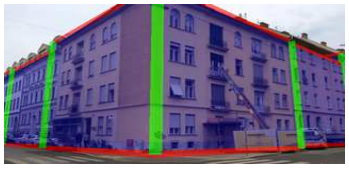


(g) Orientation histograms for detected facades.

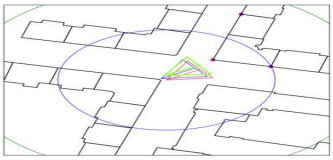
(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.



(k) Model reprojection using our method's estimate.



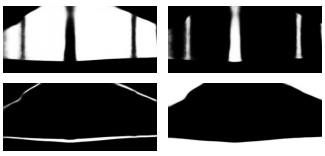
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #15 - Best solution was found using 3 corner correspondences.

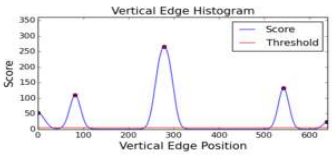
(a) Input image.

(b) Rectified input image.

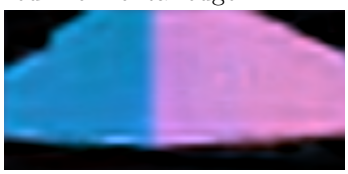
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



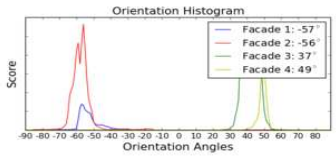
(d) Probability maps (facade, vertical edge, horizontal edge, background).



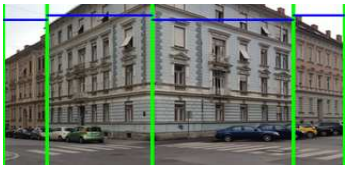
(e) Vertical edge histogram.



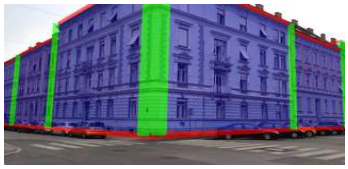
(f) Facade normal estimation.



(g) Orientation histograms for detected facades.

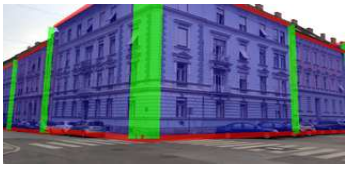


(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

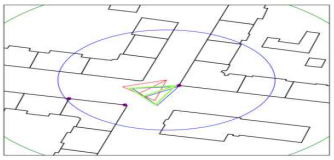


(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.



(k) Model reprojection using our method's estimate.



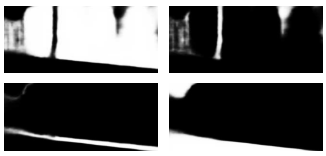
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #16 - Best solution was found using 3 corner correspondences.

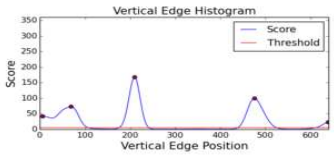
(a) Input image.

(b) Rectified input image.

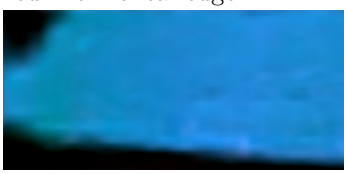
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



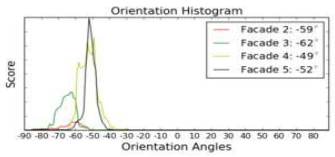
(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.



(f) Facade normal estimation.



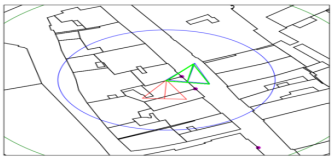
(g) Orientation histograms for detected facades.

(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



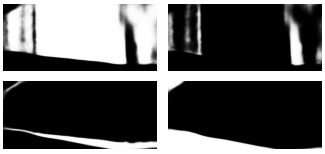
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #17 - Best solution was found using 2 corner correspondences and one façade normal.

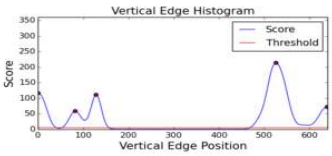
(a) Input image.

(b) Rectified input image.

(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge

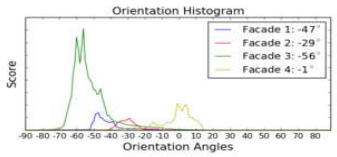


(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.

(f) Facade normal estimation.



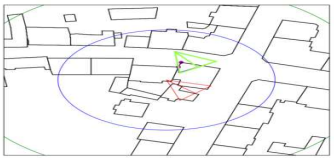
(g) Orientation histograms for detected facades.

(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



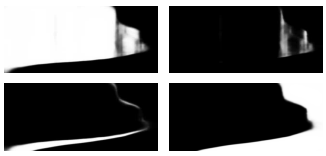
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #18 - Best solution was found using 3 corner correspondences.

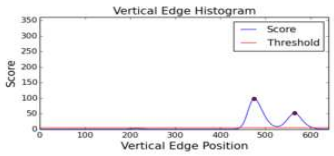
(a) Input image.

(b) Rectified input image.

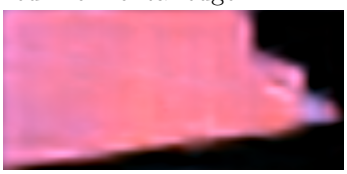
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



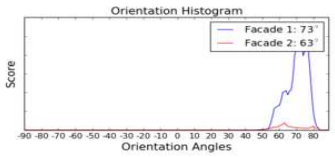
(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.



(f) Facade normal estimation.



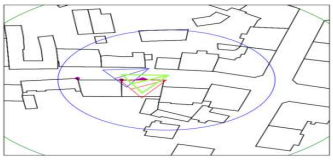
(g) Orientation histograms for detected facades.

(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

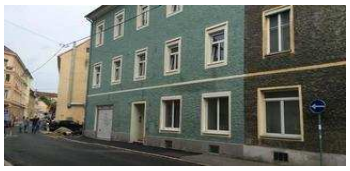
(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



(l) Map (blue:ground truth, red:sensors, green:our method).

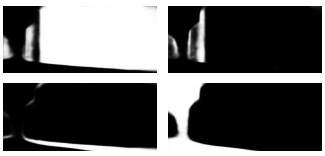
Scene #19 - Best solution was found using 2 corner correspondences and one façade normal.



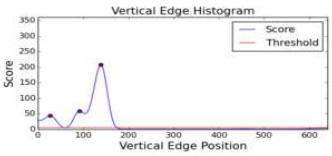
(a) Input image.

(b) Rectified input image.

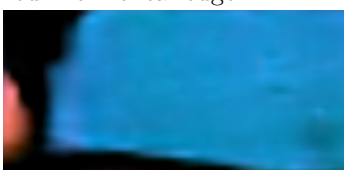
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



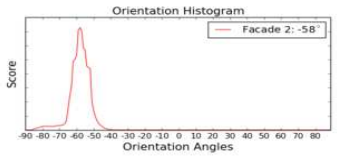
(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.



(f) Facade normal estimation.



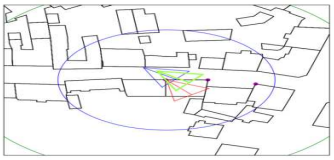
(g) Orientation histograms for detected facades.

(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



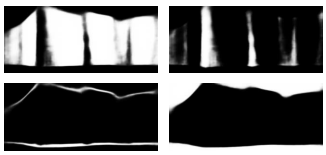
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #20 - Best solution was found using 3 corner correspondences.

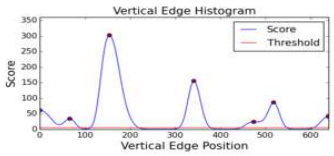
(a) Input image.

(b) Rectified input image.

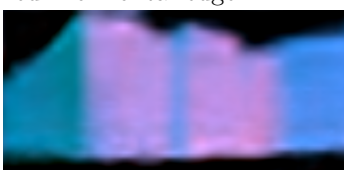
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



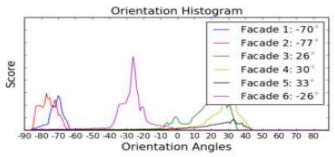
(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.



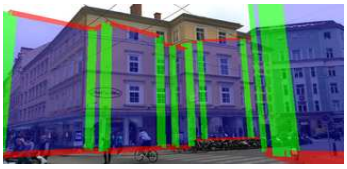
(f) Facade normal estimation.



(g) Orientation histograms for detected facades.

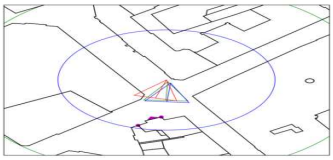
(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.



(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



(l) Map (blue:ground truth, red:sensors, green:our method).

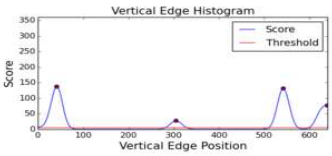
Scene #21 - Best solution was found using 3 corner correspondences.

(a) Input image.

(b) Rectified input image.

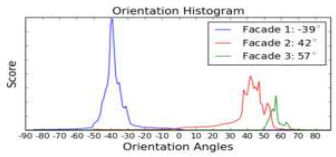
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge

(d) Probability maps (facade, vertical edge, horizontal edge, background).

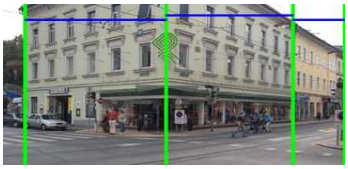


(e) Vertical edge histogram.

(f) Facade normal estimation.



(g) Orientation histograms for detected facades.



(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.

(l) Map (blue:ground truth, red:sensors, green:our method).

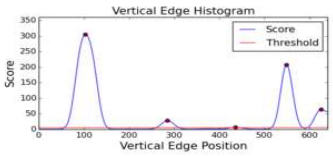
Scene #22 - Best solution was found using 3 corner correspondences.

(a) Input image.

(b) Rectified input image.

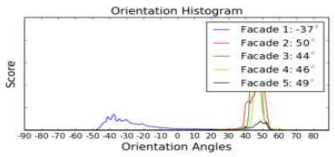
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge

(d) Probability maps (facade, vertical edge, horizontal edge, background).

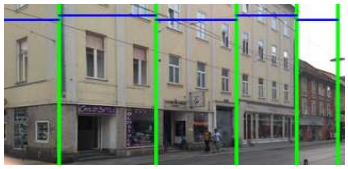


(e) Vertical edge histogram.

(f) Facade normal estimation.



(g) Orientation histograms for detected facades.



(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.

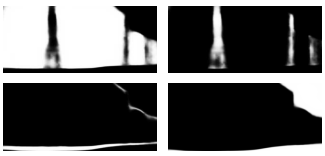
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #23 - Best solution was found using 3 corner correspondences.

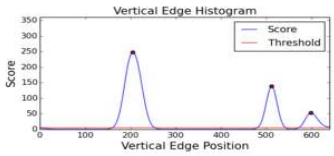
(a) Input image.

(b) Rectified input image.

(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge

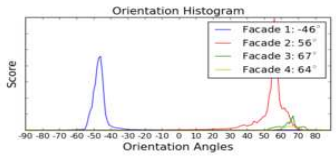


(d) Probability maps (facade, vertical edge, horizontal edge, background).

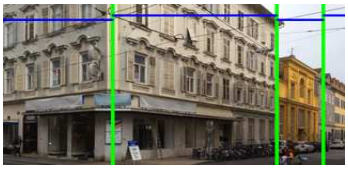


(e) Vertical edge histogram.

(f) Facade normal estimation.



(g) Orientation histograms for detected facades.

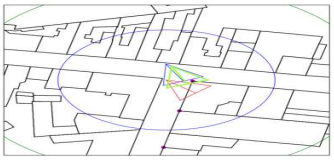


(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



(l) Map (blue:ground truth, red:sensors, green:our method).

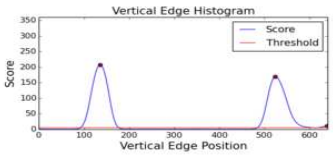
Scene #24 - Best solution was found using 2 corner correspondences and one façade normal.

(a) Input image.

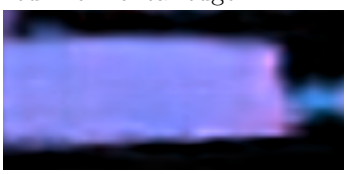
(b) Rectified input image.

(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge

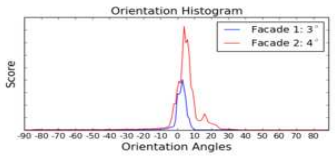
(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.



(f) Facade normal estimation.

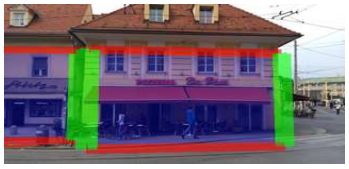


(g) Orientation histograms for detected facades.

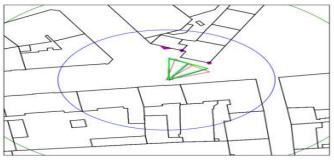
(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.



(k) Model reprojection using our method's estimate.



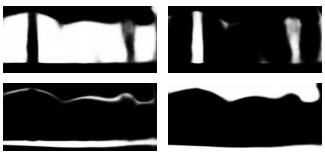
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #26 - Best solution was found using 2 corner correspondences and one façade normal.

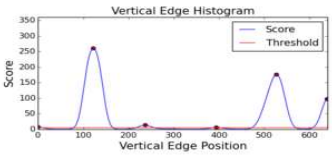
(a) Input image.

(b) Rectified input image.

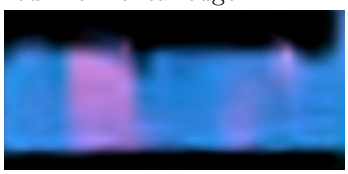
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



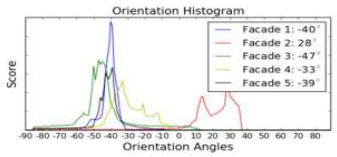
(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.



(f) Facade normal estimation.



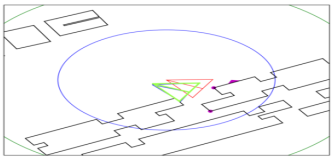
(g) Orientation histograms for detected facades.

(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



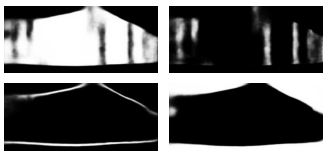
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #27 - Best solution was found using 2 corner correspondences and one façade normal.

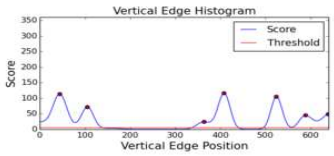
(a) Input image.

(b) Rectified input image.

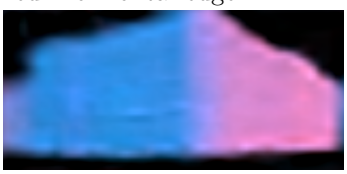
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



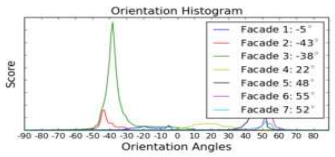
(d) Probability maps (facade, vertical edge, horizontal edge, background).



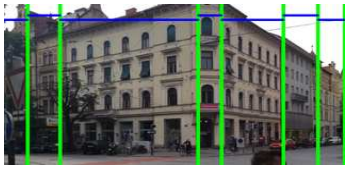
(e) Vertical edge histogram.



(f) Facade normal estimation.



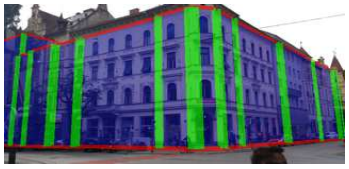
(g) Orientation histograms for detected facades.



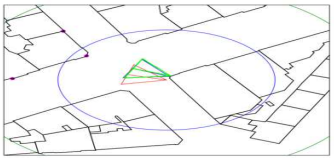
(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.



(k) Model reprojection using our method's estimate.



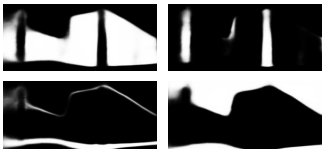
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #28 - Best solution was found using 3 corner correspondences.

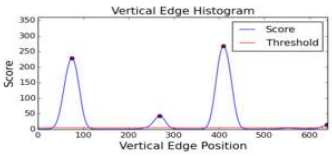
(a) Input image.

(b) Rectified input image.

(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge

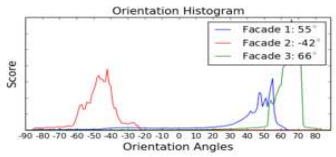


(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.

(f) Facade normal estimation.



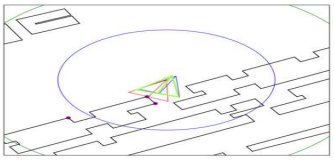
(g) Orientation histograms for detected facades.

(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



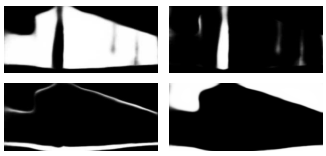
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #29 - Best solution was found using 3 corner correspondences.

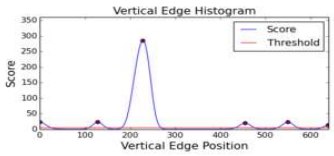
(a) Input image.

(b) Rectified input image.

(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge

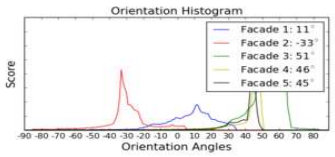


(d) Probability maps (facade, vertical edge, horizontal edge, background).

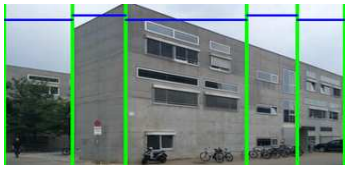


(e) Vertical edge histogram.

(f) Facade normal estimation.



(g) Orientation histograms for detected facades.

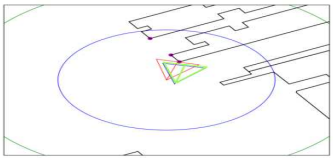


(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



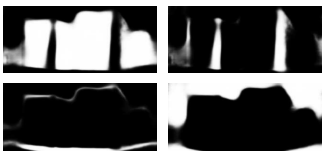
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #30 - Best solution was found using 3 corner correspondences.

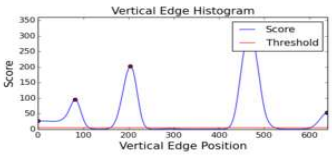
(a) Input image.

(b) Rectified input image.

(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge

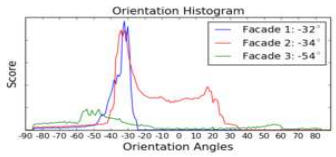


(d) Probability maps (facade, vertical edge, horizontal edge, background).

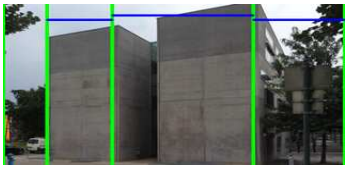


(e) Vertical edge histogram.

(f) Facade normal estimation.

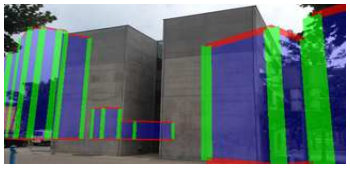


(g) Orientation histograms for detected facades.



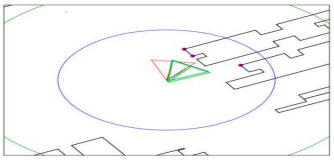
(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.



(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



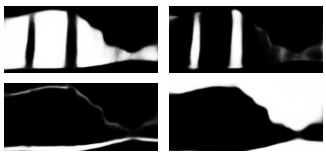
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #31 - Best solution was found using 3 corner correspondences.

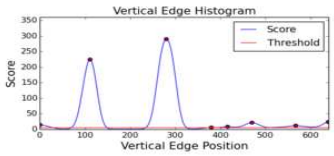
(a) Input image.

(b) Rectified input image.

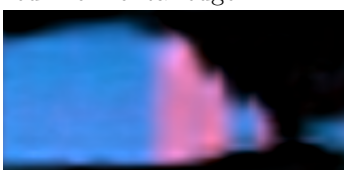
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



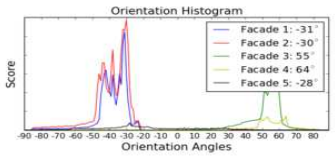
(d) Probability maps (facade, vertical edge, horizontal edge, background).



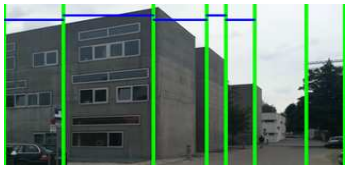
(e) Vertical edge histogram.



(f) Facade normal estimation.

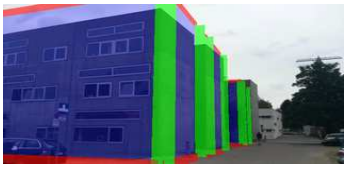


(g) Orientation histograms for detected facades.

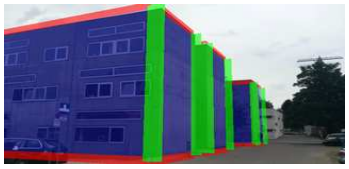


(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

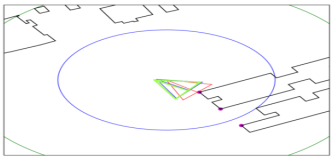
(i) Model reprojection using ground truth pose.



(j) Model reprojection using sensors estimation.



(k) Model reprojection using our method's estimate.



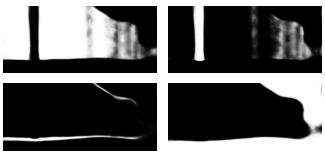
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #32 - Best solution was found using 3 corner correspondences.

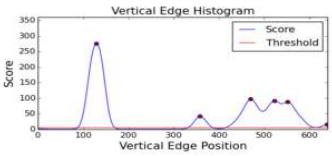
(a) Input image.

(b) Rectified input image.

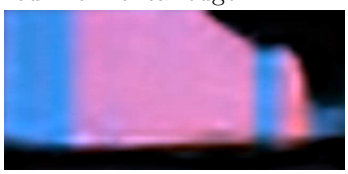
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



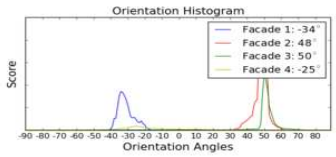
(d) Probability maps (facade, vertical edge, horizontal edge, background).



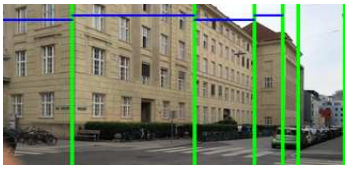
(e) Vertical edge histogram.



(f) Facade normal estimation.

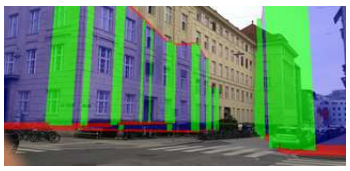


(g) Orientation histograms for detected facades.

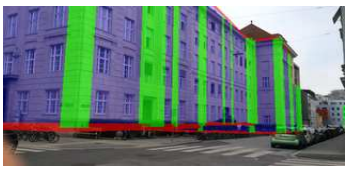


(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

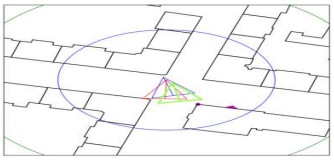
(i) Model reprojection using ground truth pose.



(j) Model reprojection using sensors estimation.



(k) Model reprojection using our method's estimate.



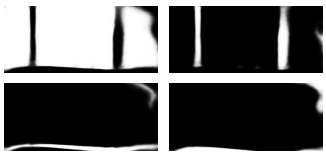
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #33 - Best solution was found using 2 corner correspondences and one façade normal.

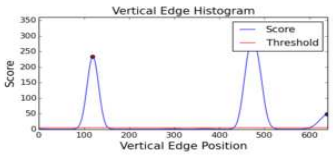
(a) Input image.

(b) Rectified input image.

(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge

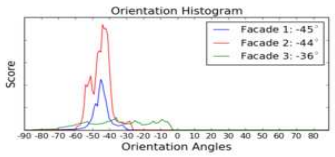


(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.

(f) Facade normal estimation.



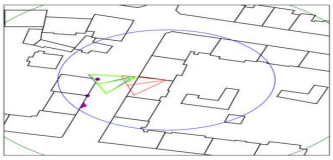
(g) Orientation histograms for detected facades.

(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



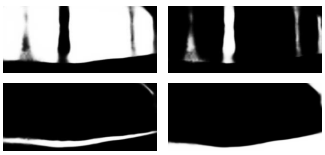
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #34 - Best solution was found using 2 corner correspondences and one façade normal.

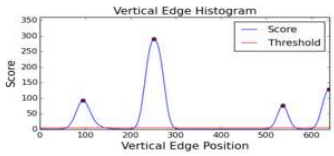
(a) Input image.

(b) Rectified input image.

(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge

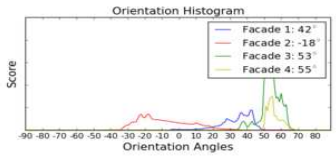


(d) Probability maps (facade, vertical edge, horizontal edge, background).

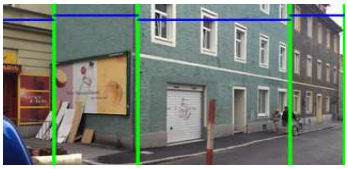


(e) Vertical edge histogram.

(f) Facade normal estimation.



(g) Orientation histograms for detected facades.

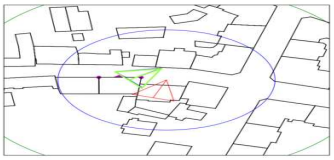


(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



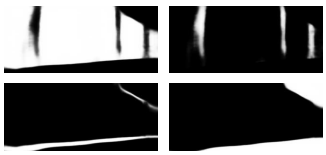
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #35 - Best solution was found using 2 corner correspondences and one façade normal.

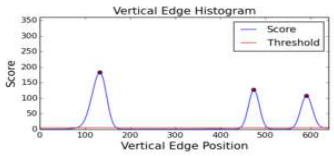
(a) Input image.

(b) Rectified input image.

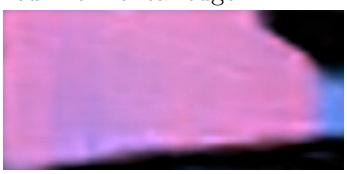
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



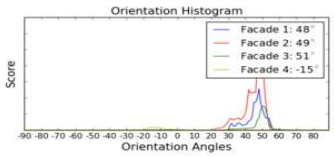
(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.



(f) Facade normal estimation.

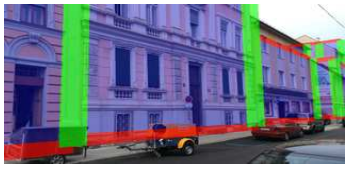


(g) Orientation histograms for detected facades.

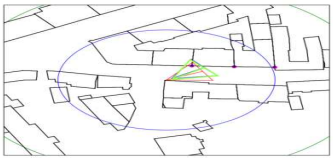
(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.



(k) Model reprojection using our method's estimate.



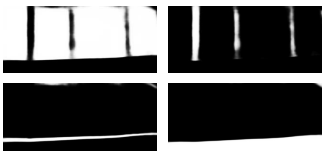
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #36 - Best solution was found using 2 corner correspondences and one façade normal.

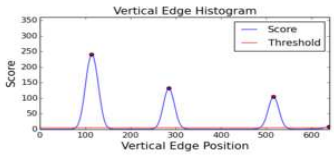
(a) Input image.

(b) Rectified input image.

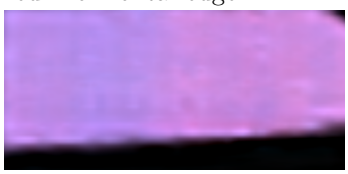
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



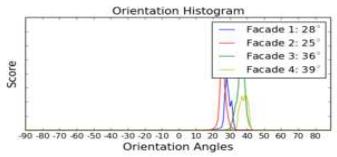
(d) Probability maps (facade, vertical edge, horizontal edge, background).



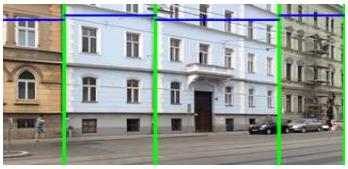
(e) Vertical edge histogram.



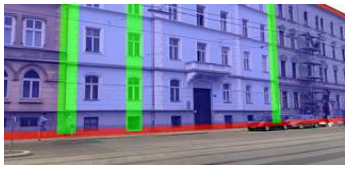
(f) Facade normal estimation.



(g) Orientation histograms for detected facades.

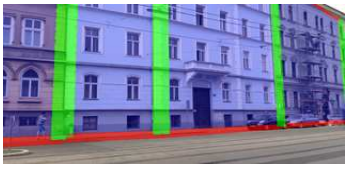


(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

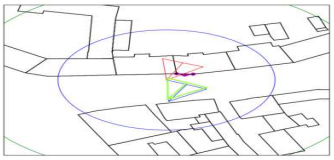


(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.



(k) Model reprojection using our method's estimate.



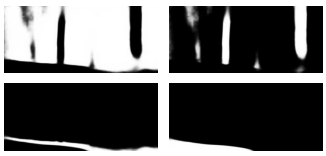
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #37 - Best solution was found using 2 corner correspondences and one façade normal.

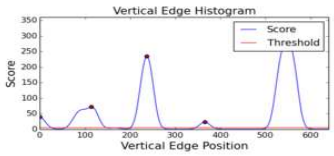
(a) Input image.

(b) Rectified input image.

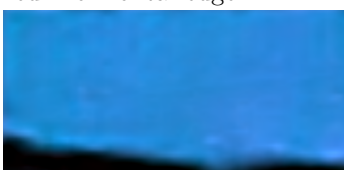
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



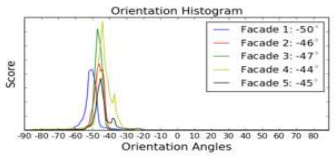
(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.



(f) Facade normal estimation.



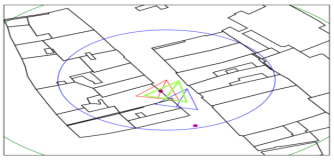
(g) Orientation histograms for detected facades.

(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



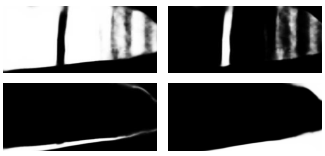
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #38 - Best solution was found using 2 corner correspondences and one façade normal.

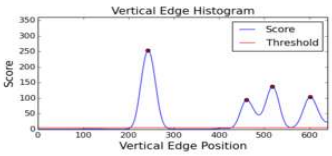
(a) Input image.

(b) Rectified input image.

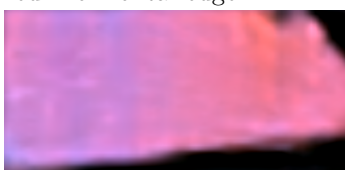
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



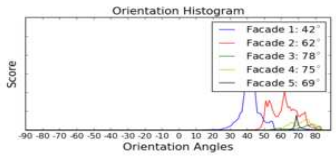
(d) Probability maps (facade, vertical edge, horizontal edge, background).



(e) Vertical edge histogram.

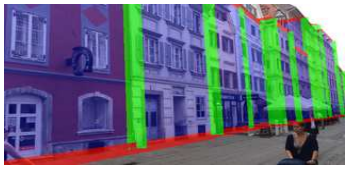


(f) Facade normal estimation.



(g) Orientation histograms for detected facades.

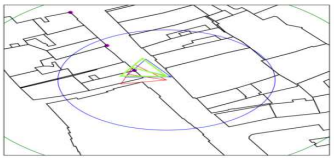
(h) Found corners (green) and defined facades (blue) shown over the rectified input image.



(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



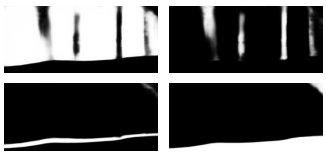
(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #39 - Best solution was found using 3 corner correspondences.

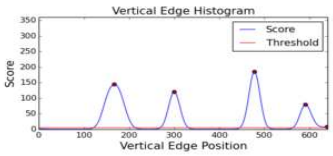
(a) Input image.

(b) Rectified input image.

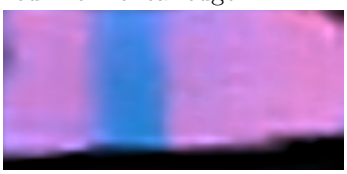
(c) Segmentation Result (blue: , green: vertical edge, red: horizontal edge



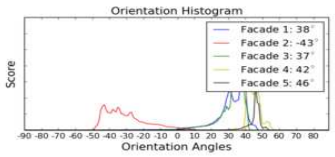
(d) Probability maps (facade, vertical edge, horizontal edge, background).



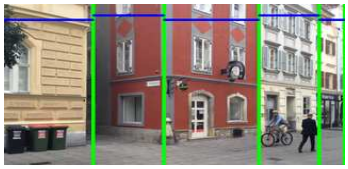
(e) Vertical edge histogram.



(f) Facade normal estimation.



(g) Orientation histograms for detected facades.

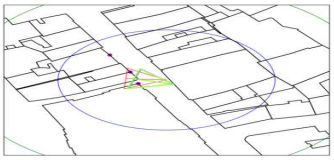


(h) Found corners (green) and defined facades (blue) shown over the rectified input image.

(i) Model reprojection using ground truth pose.

(j) Model reprojection using sensors estimation.

(k) Model reprojection using our method's estimate.



(l) Map (blue:ground truth, red:sensors, green:our method).

Scene #40 - Best solution was found using 3 corner correspondences.

Bibliography

[1] Armagan, A., Hirzer, M., and Lepetit, V. (2017a). Semantic segmentation for 3D localization in urban environments. In *Joint Urban Remote Sensing Event*. (page 38, 81)

[2] Armagan, A., Hirzer, M., Roth, P. M., and Lepetit, V. (2017b). Accurate camera registration in urban environments using high-level feature matching. In *British Machine Vision Conference*. (page 81)

[3] Armagan, A., Hirzer, M., Roth, P. M., and Lepetit, V. (2017c). Learning to align semantic segmentation and 2.5D maps for geolocalization. In *Conference on Computer Vision and Pattern Recognition*. (page 81)

[4] Arth, C., Pirchheim, C., Ventura, J., Schmalstieg, D., and Lepetit, V. (2015). Instant outdoor localization and slam initialization from 2.5d maps. In *International Symposium on Mixed and Augmented Reality*. (page 38, 52, 53, 64, 72)

[5] Avrithis, Y., Kalantidis, Y., Tolias, G., and Spyrou, E. (2010). Retrieving landmark and non-landmark images from community photo collections. In *Association for Computing Machinery Multimedia Conference*. (page 8, 32)

[6] Baatz, G., Köser, K., Chen, D., Grzeszczuk, R., and Pollefeys, M. (2010). Handling urban location recognition as a 2D homothetic problem. In *European Conference on Computer Vision*. (page 38)

[7] Badrinarayanan, V., Kendall, A., and Cipolla, R. (2015). SegNet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*. (page 45, 56, 71, 72)

[8] Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R. (1994). Signature verification using a "Siamese" time delay neural network. In *Advances in Neural Information Processing Systems*. (page 22, 23)

[9] Bujnak, M., Kukelova, Z., and Pajdla, T. (2008). A general solution to the p4p problem for camera with unknown focal length. In *Conference on Computer Vision and Pattern Recognition*. (page 31)

[10] Calonder, M., Lepetit, V., Strecha, C., and Fua, P. (2010). Brief: Binary robust independent elementary features. In *European Conference on Computer Vision*. (page 8)

[11] Cham, T., Ciptadi, A., Tan, W., Pham, M., and Chia, L. (2010). Estimating camera pose from a single urban ground-view omnidirectional image and a 2D building outline map. In *Conference on Computer Vision and Pattern Recognition*. (page 38)

[12] Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2015). Semantic image segmentation with deep convolutional nets and fully connected CRFs. In *International Conference for Learning Representations*. (page 30, 44)

[13] Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L. (2016). DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *arXiv preprint arXiv:1606.00915*. (page 30, 44)

[14] Chu, H., Gallagher, A., and Chen, T. (2014). GPS refinement and camera orientation estimation from a single image and a 2D map. In *Conference on Computer Vision and Pattern Recognition Workshops*. (page 38)

[15] Ciepluch, B., Mooney, P., Jacob, R., and Winstanley, A. (2011). Sketches of generic framework for quality assessmenft of volunteered geographical data. In *IEEE Geoscience and Remote Sensing Society*. (page 50)

[16] Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. (2016). The Cityscapes dataset for semantic urban scene understanding. In *Conference on Computer Vision and Pattern Recognition*. (page 5)

[17] Dai, J., Li, Y., He, K., and Sun, J. (2016). R-FCN: Object detection via region-based fully convolutional networks. In *Advances in Neural Information Processing Systems*. (page 21)

[18] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Conference on Computer Vision and Pattern Recognition*. (page 39)

[19] David, P. and Ho, S. (2011). Orientation descriptors for localization in urban environments. In *International Conference on Intelligent Robots and Systems*. (page 38)

[20] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., and Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition*. (page 26)

[21] DeTone, D., Malisiewicz, T., and Rabinovich, A. (2017). Toward geometric deep SLAM. *arXiv preprint arXiv:1707.07410*. (page 37)

[22] Dhome, M., Richetin, M., Lapreste, J.-T., and Rives, G. (1989). Determination of the attitude of 3d objects from a single perspective view. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(12):1265–1278. (page 31)

[23] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338. (page 29, 30)

[24] Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the Association for Computing Machinery*, 24(6):381–395. (page 31, 71, 72)

[25] Gao, X.-S., Hou, X.-R., Tang, J., and Cheng, H.-F. (2003). Complete solution classification for the perspective-three-point problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(8):930–943. (page 31, 71)

[26] Girres, J. F. and Touya, G. (2010). Quality assessment of the French OpenStreetMap dataset. *Transactions in Geographic Information Systems*, 14(4):435–459. (page 51)

[27] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*. (page 23)

[28] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT press Cambridge. (page 17, 18, 19, 20, 23, 24, 25)

[29] Hadsell, R., Chopra, S., and LeCun, Y. (2006). Dimensionality reduction by learning an invariant mapping. In *Conference on Computer Vision and Pattern Recognition*. (page 39)

[30] Hakeem, A., Vezzani, R., Shah, M., and Cucchiara, R. (2006). Estimating geospatial trajectory of a moving camera. In *International Conference on Pattern Recognition*. (page 9)

[31] Hartley, R. and Zisserman, A. (2003). *Multiple View Geometry in Computer Vision*. Cambridge University Press. (page 30, 31)

[32] Hays, J. and Efros, A. (2008). IM2GPS: estimating geographic information from a single image. In *Conference on Computer Vision and Pattern Recognition*. (page 8, 32, 33, 34)

[33] Hays, J. and Efros, A. (2015). Large-scale image geolocalization. In *Multimodal Location Estimation of Videos and Images*. Springer. (page 8, 32, 33)

[34] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*. (page 27, 28, 30)

[35] Heinly, J., Schonberger, J. L., Dunn, E., and Frahm, J. M. (2015). Reconstructing the world* in six days *(as captured by the yahoo 100 million image dataset). In *Conference on Computer Vision and Pattern Recognition*. (page 9)

[36] Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*. (page 24)

[37] Kalogerakis, E., Vesselova, O., Hays, J., Efros, A., and Hertzmann, A. (2009). Image sequence geolocation with human travel priors. In *International Conference on Computer Vision*. (page 32)

[38] Kendall, A. and Cipolla, R. (2016). Modelling uncertainty in deep learning for camera relocalization. In *International Conference on Robotics and Automation*. (page 33, 35)

[39] Kendall, A. and Cipolla, R. (2017). Geometric loss functions for camera pose regression with deep learning. In *Conference on Computer Vision and Pattern Recognition*. (page 33, 35)

[40] Kendall, A., Grimes, M., and Cipolla, R. (2015). PoseNet: A convolutional network for real-time 6-DOF camera relocalization. In *International Conference on Computer Vision*. (page 26, 32, 33, 34, 35)

[41] Klein, G. and Murray, D. (2007). Parallel tracking and mapping for small AR workspaces. In *International Symposium on Mixed and Augmented Reality*. (page 9)

[42] Koenderink, J. J. and van Doorn, A. J. (1987). Representation of local geometry in the visual system. *Biological Cybernetics*, 55(6):367–375. (page 22)

[43] Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. (page 26)

[44] Kukelova, Z., Bujnak, M., and Pajdla, T. (2008). Automatic generator of minimal problem solvers. In *European Conference on Computer Vision*. (page 71)

[45] Kukelova, Z., Bujnak, M., and Pajdla, T. (2013). Real-time solution to the absolute pose problem with unknown radial distortion and focal length. In *International Conference on Computer Vision*. (page 31, 71)

[46] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551. (page 21)

[47] Lepetit, V., Moreno-Noguer, F., and Fua, P. (2009). EPnP: An accurate O(n) solution to the PnP problem. *International Journal of Computer Vision*, 81(2):155. (page 31)

[48] Li, Y., Snavely, N., Huttenlocher, D., and Fua, P. (2012). Worldwide pose estimation using 3D point clouds. In *European Conference on Computer Vision*. (page 33)

[49] Li, Y., Snavely, N., and Huttenlocher, D. P. (2010). Location recognition using prioritized feature matching. In *European Conference on Computer Vision*. (page 9, 33)

[50] Lin, T. Y., Belongie, S., and Hays, J. (2013). Cross-view image geolocalization. In *Conference on Computer Vision and Pattern Recognition*. (page 36, 37)

[51] Lin, T. Y., Cui, Y., Belongie, S., and Hays, J. (2015). Learning deep representations for ground-to-aerial geolocalization. In *Conference on Computer Vision and Pattern Recognition*. (page 9, 36, 37, 38, 39, 40)

[52] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Conference on Computer Vision and Pattern Recognition*. (page 21, 22, 26, 27, 29, 30, 44, 45, 46, 55, 59, 71, 72, 74, 83)

[53] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110. (page 7, 8, 32, 76)

[54] Ludwig, I., Voss, A., and Krause-Traudes, M. (2011). A comparison of the street networks of Navteq and OSM in Germany. In *Advancing Geoinformation Science for a Changing World*, volume 1, pages 65–84. Springer. (page 51)

[55] Matan, O., Burges, C. J., LeCun, Y., and Denker, J. S. (1992). Multi-digit recognition using a space displacement neural network. In *Advances in Neural Information Processing Systems*. (page 21)

[56] Meierhold, N. and Schmich, A. (2009). Referencing of images to laser scanner data using linear features extracted from digital images and range images. *International Society for Photogrammetry and Remote Sensing*, 38(3/W8):164–170. (page 38)

[57] Middelberg, S., Sattler, T., Untzelmann, O., and Kobbelt, L. (2014). Scalable 6-DOF localization on mobile devices. In *European Conference on Computer Vision*. (page 9)

[58] Naval Jr, P. C., Mukunoki, M., Minoh, M., and Ikeda, K. (1997). Estimating camera position and orientation from geographical map and mountain image. In *38th Research Meeting of the Pattern Sensing Group, Society of Instrument and Control Engineers*. (page 37)

[59] Nister, D. (2004). An efficient solution to the five-point relative pose problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):756–770. (page 31)

[60] Nister, D. and Stewenius, H. (2006). Scalable recognition with a vocabulary tree. In *Conference on Computer Vision and Pattern Recognition*. (page 8, 32)

[61] Oliva, A. and Torralba, A. (2006). Building the gist of a scene: the role of global image features in recognition. *Progress in Brain Research*, 155:23–36. (page 32)

[62] Pavan, M. and Pelillo, M. (2003). A new graph-theoretic approach to clustering and segmentation. In *Conference on Computer Vision and Pattern Recognition*. (page 40)

[63] Pavan, M. and Pelillo, M. (2007). Dominant sets and pairwise clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(1):167–172. (page 40)

[64] Ramalingam, S., Bouaziz, S., Sturm, P., and Brand, M. (2010). SKYLINE2GPS: Localization in urban canyons using omni-skylines. In *International Conference on Intelligent Robots and Systems*. (page 8, 9, 32, 37)

[65] Ramalingam, S., Bouaziz, S., and Sturm, P. F. (2011). Pose estimation using both points and lines for geo-localization. In *International Conference on Robotics and Automation*. (page 38)

[66] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Conference on Computer Vision and Pattern Recognition*. (page 18)

[67] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*. (page 39)

[68] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation. In *Conference on Medical Image Computing and Computer Assisted Intervention*. (page 45, 56, 71, 72)

[69] Ros, G., Sellart, L., Materzynska, J., Vazquez, D., and Lopez, A. (2016). The SYNTHIA Dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Conference on Computer Vision and Pattern Recognition*. (page 5)

[70] Russell, B. C., Torralba, A., Murphy, K. P., and Freeman, W. T. (2008). LabelMe: a database and web-based tool for image annotation. *International Journal of Computer Vision*, 77:157–173. (page 49)

[71] Sattler, T., Havlena, M., Radenovic, F., Schindler, K., and Pollefeys, M. (2015). Hyperpoints and fine vocabularies for large-scale location recognition. In *International Conference on Computer Vision*. (page 9)

[72] Sattler, T., Leibe, B., and Kobbelt, L. (2011). Fast image-based localization using direct 2D-to-3D matching. In *International Conference on Computer Vision*. (page 33)

[73] Sattler, T., Leibe, B., and Kobbelt, L. (2012). Improving image-based localization by active correspondence search. In *European Conference on Computer Vision*. (page 9)

[74] Sattler, T., Leibe, B., and Kobbelt, L. (2017). Efficient & effective prioritized matching for large-scale image-based localization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(9):1744–1756. (page 35, 36)

[75] Saurer, O., Baatz, G., Köser, K., Pollefeys, M., et al. (2016). Image based geo-localization in the alps. *International Journal of Computer Vision*, 116(3):213–225. (page 37)

[76] Schindler, G., Brown, M. A., and Szeliski, R. (2007). City-scale location recognition. In *Conference on Computer Vision and Pattern Recognition*. (page 8, 32)

[77] Sehra, S. S., Singh, J., and Rai, H. S. (2014). A systematic study of OpenStreetMap data quality assessment. In *International Conference on Information Technology: New Generations*, pages 377–381. (page 51)

[78] Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. (2013). Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*. (page 33)

[79] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*. (page 20, 22, 26, 30, 45, 46)

[80] Sivic, J. and Zisserman, A. (2003). Video Google: A text retrieval approach to object matching in videos. In *International Conference on Computer Vision*. (page 32)

[81] Svarm, L., Enqvist, O., Oskarsson, M., and Kahl, F. (2014). Accurate localization and pose estimation for large 3D models. In *Conference on Computer Vision and Pattern Recognition*. (page 9, 33)

[82] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Conference on Computer Vision and Pattern Recognition*. (page 20, 26, 27, 28)

[83] Talluri, R. and Aggarwal, J. K. (1993). Image map correspondence for mobile robot self-location using computer graphics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(6):597–601. (page 37)

[84] Taneja, A., Ballan, L., and Pollefeys, M. (2012). Registration of spherical panoramic images with cadastral 3d models. In *3D Imaging, Modeling, Processing, Visualization and Transmission*. (page 38)

[85] Tateno, K., Tombari, F., Laina, I., and Navab, N. (2017). Cnn-slam: Real-time dense monocular SLAM with learned depth prediction. *arXiv preprint arXiv:1704.03489*. (page 37)

[86] Tian, Y., Chen, C., and Shah, M. (2017). Cross-view image matching for geo-localization in urban environments. *arXiv preprint arXiv:1703.07815*. (page 37, 38, 39, 40)

[87] Tieleman, T. and Hinton, G. (2012). Lecture 6.5-Rmsprop: Divide the gradient by a running average of its recent magnitude. *Coursera: Neural Networks for Machine Learning*. (page 63)

[88] Tola, E., Lepetit, V., and Fua, P. (2010). Daisy: An efficient dense descriptor applied to wide-baseline stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(5):815–830. (page 7, 8)

[89] Triggs, B., McLauchlan, P. F., Hartley, R. I., and Fitzgibbon, A. W. (1999). Bundle adjustment - a modern synthesis. In *International Workshop on Vision algorithms*. (page 9)

[90] Tzeng, E., Zhai, A., Clements, M., Townshend, R., and Zakhor, A. (2013). User-driven geolocation of untagged desert imagery using digital elevation models. In *Conference on Computer Vision and Pattern Recognition Workshops*. (page 37)

[91] Viswanathan, A., Pires, B. R., and Huber, D. (2014). Vision based robot localization by ground to satellite matching in GPS-denied situations. In *International Conference on Intelligent Robots and Systems*. (page 36)

[92] Walch, F., Hazirbas, C., Leal-Taixe, L., Sattler, T., Hilsenbeck, S., and Cremers, D. (2017). Image-based localization using LSTMs for structured feature correlation. In *International Conference on Computer Vision*. (page 33, 35, 36)

[93] Wang, J., Leung, T., Rosenberg, C., Wang, J., Philbin, J., Chen, B., and Wu, Y. (2014). Learning fine-grained image similarity with deep ranking. *arXiv preprint arXiv:1404.4661*. (page 8, 32)

[94] Weyand, T., Kostrikov, I., and Philbin, J. (2016). PlaNet-photo geolocation with convolutional neural networks. In *European Conference on Computer Vision*. (page 18, 32, 33)

[95] Workman, S., Souvenir, R., and Jacobs, N. (2015). Wide-area image geolocalization with aerial reference imagery. In *International Conference on Computer Vision*. (page 8, 32, 36, 37)

[96] Xiao, J., Ehinger, K. A., Hays, J., Torralba, A., and Oliva, A. (2016). Sun Database: Exploring a large collection of scene categories. *International Journal of Computer Vision*, 119(1):3–22. (page 32)

[97] Yi, K., Trulls, E., Lepetit, V., and Fua, P. (2016). LIFT: Learned invariant feature transform. In *European Conference on Computer Vision*. (page 7, 8)

[98] Yu, F. and Koltun, V. (2015). Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*. (page 30, 44)