Christian Poglitsch

Outdoor Localization using a Particle Filter

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme
Software Development and Business Administration

submitted to

Graz University of Technology

Supervisor

Prof. Dr. Dieter Schmaltstieg
Institute for Computer Graphics and Vision

Graz, Austria, July 2015

Abstract

We propose an outdoor localization system using a particle filter. In our approach, a textured and geo-registered model of the outdoor environment is used as a reference to estimate the pose of a smartphone. The device's position and the orientation, obtained from a Global Positioning System (GPS) receiver and an inertial measurement unit (IMU), are used as a first estimation of the true pose. Then, based on the sensor data, multiple pose hypotheses are randomly distributed and used to produce renderings of the geo-referenced virtual model. With vision-based methods, the rendered images are compared to the image received from a smartphone, and the matching scores are used to update the particle filter. The outcome of our system improves the camera pose estimate in real time without user assistance. In contrast to previous methods, it is not necessary to move around until a baseline is found or to rotate the smartphone. Experimental evaluation shows that the method significantly improves real-virtual alignment in the augmented camera image.

Kurzfassung

In dieser Arbeit stellen wir eine Methode zur Lokalisation von mobilen Computern, wie Smartphones oder Tablets, im Freien vor. Als Technik setzen wir einen Partikelfilter ein. Unser Ansatz nutzt ein texturiertes und geographisch registriertes Model der Umgebung als Referenz um die Lage eines Smartphones oder Tablets zu bestimmen. Als erste grobe Schätzung der Position und der Orientierung des Computers dienen die vorhandenen Sensoren, wie Inertialsensor und GPS. Ausgehend von den Sensordaten werden zufallsgeneriert Hypothesen über die Lage der Kamera erstellt und anhand des virtuellen Models Bilder dieser Positionen erzeugt. Mittels Computer Vision Methoden werden die virtuell generierten Bilder mit dem importierten Bild des mobilen Gerätes verglichen und gewichtet. Das Resultat unseres Ansatzes optimiert die Position und Orientierung des mobilen Gerätes in Echtzeit ohne Unterstützung des Benutzers. Im Vergleich zu früheren Methoden ist es nicht notwendig, dass der Benutzer die Kamera bewegt, bis eine Baseline vorhanden ist, oder dass die Kamera um die eigene Achse gedreht werden muss. Testergebnisse zeigen eine signifikante Verbesserung der Position und Orientierung in Echtzeit.

Statutory Declaration

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 marked all material which has been quoted either literally or by content from the used sources.

The text document uploaded to TUGRAZonline is identical to the presented master's thesis dissertation.

Place	Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Ort	Datum	Unterschrift

Acknowledgments

First and foremost I would like to thank Jonathan Ventura for the time he invested in this project. His valuable input and comments helped me to design, program and write about this very interesting topic.

Furthermore, I would like to thank Dieter Schmalstieg and Clemens Arth for their support and the opportunity to write a paper that hopefully gets accepted at the International Symposium on Mixed and Augmented Reality (ISMAR) 2015.

Contents

List of Figures

List of Tables

1 Introduction

This work addresses the problem of image-based 3D localization. The task is to estimate the camera pose given a query image. For Augmented Reality (AR) applications accurate position and orientation information is crucial. Displaying virtual content in the real environment requires knowledge of the exact pose of the display device.

Two items of information must be known to provide seamlessly augmented visuals over the real world: (a) the current pose in the real world that needs to be augmented and (b) the virtual object geometry and its accurate registration with the real world [38]. If these conditions are fulfilled, the augmented reality experience becomes more convincing and useful for the user, when virtual content respects and responds to the real environment. Therefore, virtual content needs to be rendered seamlessly into real world imagery.

Outdoor AR applications today rely mostly on the built-in sensors to estimate the pose of the camera. Self-localization in large environments is a vital task for accurately registered information visualization in location-based AR [2]. Self-localization is the task of autonomously determining the pose of a device with respect to a reference coordinate system. For high-quality AR, the current camera pose with respect to the environment must be estimated with full six degrees of freedom (6DoF) at real-time update rates.

Today's smartphones estimate the pose by triangulation from satellites using GPS and Wireless Local Area Network (WLAN) transmitters, along with a digital compass and inertial orientation sensors like gyroscope or accelerometer. Accuracy is limited with these approaches, e.g., civilian GPS exhibits an accuracy of 7.8 meters at a 95 % confidence level[1]. To improve accuracy, augmentations like differential GPS can be used, but these extensions are expensive and typically not part of consumer hardware. These sensors are further limited by the availability of a signal, e.g., in urban environments, building walls interfere with satellite visibility, WLAN transmission, and magnetometer measurements. The lack of accuracy is a major problem, because the pose estimation is insufficient for high quality AR [53]. Therefore, AR applications cannot rely solely on the sensors of smart phones. As a consequence, alternative methods have to replace or assist the sensors

[1] `http://www.gps.gov/`

of the smart phones to estimate their pose.

A promising approach to assist self-localization with sensors is computer vision. Vision-based methods provide high precision in estimating the camera pose and the ability to run in real time on modern hardware [52]. To combine information and content of the real and virtual worlds efficiently and effectively, they need some model of the real as well as the virtual environment. The camera pose can be estimated relative to a virtual model of the environment by matching features between the camera image and model. Such techniques can provide a high level of positional accuracy, within 5-25 cm. There is one main challenge to implementation of wide-area visual localization: how to achieve accurate real-time localization on a smartphone or tablet.

In this work, we solve the outdoor localization problem with a particle filter, otherwise known as Monte Carlo Localization [49]. In our approach, a textured and geo-referenced model of the outdoor environment is used as a reference to estimate the pose of a smartphone. The GPS position and the orientation obtained by IMU are used as a first estimation of the real pose. Using a motion model, multiple poses are calculated and rendered in the virtual environment. The result is a collection of images and their corresponding camera poses. With vision-based methods, the rendered images are compared to the image or video received from the smartphone. In this process, every particle gets a weight value. The weight of a particle can be thought of as the probability that the particle's state corresponds to the true state of the system [49]. The particle with the highest weight represents the corrected 6DoF camera pose.

The outcome of our system estimates the camera pose in real time without user assistance. In contrast to previous methods, it is not necessary to move around, until a baseline is found [52], or to rotate the smart device [2]. Our approach works for images as well as for video streams obtained from a smartphone or tablet. The result of our work can be used in multiple applications, because an accurate camera pose estimation in real time is crucial for location based AR applications in fields like tourist navigation, orientation, infrastructure maintenance or computer games.

2 Related Work

Contents

The goal of this project is to design a sophisticated localization and tracking system that works accurate in real time. The most common approach for global registration on a smartphone is to rely on IMU and GPS to determine the device position and orientation. Augmented reality applications often use the built-in sensors to determine the pose but cannot rely on the accuracy and precision of the sensors. Therefore, it is not possible to compute pixel-accurate graphical overlays, which provide spatial relationship between virtual and real world.

The problem of computing the position and orientation of a camera with respect to a virtual representation of the scene, which is referred to as image-based localization [28], has received a lot of attention in the computer vision community. It has important applications in location recognition [21, 27, 39, 44], autonomous robot navigation [1, 32, 41], AR and Mixed Reality (MR) [14, 24, 54]. Accurate self-localization enables AR applications in fields like tourist navigation, orientation, infrastructure maintenance or computer games.

There are two approaches to image-based localization. The first addresses localization and requires knowledge of created map or 3D model [14, 20, 27, 42] in advance. Vision-based localization is solved by (a) using a collection of images or (b) a 3D model as reference. The second approach addresses the problem of SLAM, where the camera is localized within an unknown scene. This approach is recording a virtual model of the environment with methods like Structure from Motion (SfM) and is processing localization at the same time.

In this project, we focus on localization, because our goal is a localization and tracking framework that works on a smartphone in real time with no user support. SLAM

approaches deliver promising results, but require the user to take a series of images, until a reconstruction can be produced. Therefore, in this project, we will use the results from SfM methods to improve user experience for AR applications.

2.1 Localization

The current state-of-the-art in image-based localization can mainly be divided into two different approaches: (a) Image Retrieval (IR) techniques and (b) Brute Force (BF) nearest neighbor matching methods [15]. As reference for the query image, a collection of geo-referenced images or 3D models constructed with SfM is possible. IR-based methods first identify a subset of images of the database that are most similar to the query image, using efficient techniques such as the vocabulary tree [34]. The BF localization strategy, which bypasses the image retrieval step by matching directly between the query features and the entire point cloud, consistently leads to improved localization quality [43].

Many approaches exist for localization from a single image, in some cases only with accuracy comparable to consumer GPS [6, 44, 48, 50, 51, 56, 57]. These systems typically use large-scale mapping efforts with thousands of photos [47] or use a real-time tracking and mapping approach, which is challenging to operate in a large space.

Arth et al. [2] propose a system for self-localization on smartphones using a GPS position and panoramic view of the user's environment. In order to run directly on mobile devices, their approach requires only moderate computational and storage resources. It is assumed that a 3D reconstruction of the environment is ready to be used for model-based tracking. An issue is the limited Field of View (FOV) of mobile device cameras. This fact makes it problematic to compute accurate localization, since, often, too few high-quality interest points are contained in a single image. At startup, the user is required to explore the environment through the camera's viewfinder. It is, therefore, necessary to expect that the user is standing still for a moment, while orienting the device. Their system requires this behavior for initializing the tracking and assumes the user's cooperation.

SfM approaches enable the creation of large scale 3D models of urban scenes. These compact scene representations can be used for accurate image-based localization. Image-based localization deals with the problem of how to precisely recover the 3D camera pose from a query image within a known 3D world. One possible approach is to localize the pose with respect to a 3D point cloud obtained by a SfM pipeline.

Real-time global localization using a 3D model of the environment requires a pre-made point cloud [53] or edge [36] model of the environment, which could be registered to a useful global reference frame, such as the floor plan of a building or a geographic coordinate system such as a UTM zone [52]. There are several problems with directly tracking the globally aligned point cloud. Typically, the point cloud needs to be stored on the device to achieve real-time performance, which introduces problems in data transfer, storage, and maintenance. In terms of the interface experience, the user may have to physically search with the camera for some time until a proper viewpoint is found, from

which the system can localize. Another issue is that the global point cloud is made in an offline process, and thus becomes outdated when the environment geometry, appearance or illumination changes.

An important bottleneck is the computation of 2D-to-3D correspondences required for pose estimation [42]. Despite the scalability of SfM approaches [20, 27, 42], real-time image-based localization in large environments remains a challenging problem. As the scene gets larger, recognizing unique landmarks becomes more challenging. This difficulty may be overcome by using sophisticated image features [20, 27, 42] such as Scale-Invariant Feature Transform (SIFT) [30], but these are too expensive to compute in real-time.

Reitmayr and Drummond [36] present an edge-based tracking system using textured 3D building models for the input query of a mobile phone. They make use of the video image as well as gyroscope and measurements of gravity and magnetic field. This work mainly addresses frame-to-frame tracking, but also includes an approach for re-localization using a set of corner-like features extracted using the FAST corner detector [40]. The features are compared using the SSD of the feature vectors, and the best match for each feature is stored. Because the 3D models created from photographs do not exactly fit the corresponding map points, the registration still contains errors of up to 0.25 meter between points in the models and the map data. For future work, Reitmayr and Drummond suggest to use GPS for an initial guess to limit the search range for possible locations.

In a follow-up work, Reitmayr and Drummond discuss the extension of their hybrid tracking system with a GPS sensor to add initialization and robustness against failures of the tracker [37]. The GPS-based measurements provide initial 2D position information at start-up that allows the vision-based component to initialize correctly. To overcome the inaccuracy in the GPS position, a local search in the neighborhood of the data provided by the sensors is performed, in a manner similar to the approach by Coors et al. [9]. Our approach also uses renderings and their corresponding poses of a textured model; however, in contrast to Reitmayr and Drummond, we use pixel-wise cost functions instead of edge search, and our algorithm processes continuous particle filtering, rather than one-shot localization. As result, our approach unifies localization and continuous tracking in a single method.

Self-localization is a deeply investigated field in mobile robotics, and many effective solutions have been proposed. In this context, Monte Carlo Localization (MCL) is one of the most popular approaches, and represents a good trade-off between robustness and accuracy. The basic underlying principle of this family of approaches is using a Particle Filter for tracking a probability distribution of the possible robot poses [31]. The common approach of algorithms proposed to solve position tracking is Kalman Filter localization [26], while global positioning encloses common frameworks like Multi Hypotheses Localization [22], Histogram Filters [5] and Particle Filters [18].

Klein and Murray [23] demonstrate a real-time, full-3D edge tracker based on a particle filter. Since the early work of Harris [19] this task has often been accomplished by detecting edges in the image. Using a Computer Aided Design (CAD) model of the object to be

tracked as reference, the camera pose which best aligns best to the rendered model is determined. Edges are easy to detect in images, offer a large degree of invariance to pose and illumination changes and have some resilience to difficult imaging conditions like noise and blur. Therefore, this approach remains an active field of computer vision. However, edge models lack detail in comparison to fully textured models as used in this work. The primary disadvantage of edges as features to track is that edge images look similar. Whereas a rich selection of description techniques exist for matching point features, edges are often matched simply by image proximity to a prior. Without a valid prior pose estimate, most edge-based systems will break and not recover tracking [23]. Unimodal methods calculate one Gaussian posterior pose for each frame, which then provides a single prior pose for next frame. If the estimate is sufficiently incorrect, tracking will fail. Particle filters provide an alternative approach to propagating pose estimates. Prior distributions are no longer limited to single Gaussian, but can adopt truly non-Gaussian, multi-modal forms.

Klein and Murray show that it is possible to measure each particle's likelihood directly on the graphics card. Depending on model complexity, this can be done at rates in excess of 10,000 pose hypotheses per second, and allows their system to track objects of a complexity comparable to that supported by state-of-the-art unimodal systems.

Aubry et al. [4] demonstrate a technique that can reliably align images of architectural sites, like drawings, paintings or historical photographs, to a 3D model of the site. It is a challenging task to align images, as a query can be very different from the 3D model due to rendering style, drawing errors, lighting or change of seasons. Furthermore, the space of possible alignments of the input image to a large 3D model is huge. To align different query inputs like drawings, paintings to a 3D model local feature matching based on interest points, e.g., SIFT, often fails to find correspondings across paintings and photographs. In their approach they summarize a 3D model as a collection of discriminative visual elements. They define these elements to be mid-level patches with respect to a given viewpoint. They describe their method as a combination of multi-view geometry and part-based object recognition.

State-of-the-art image-based localization uses 3D point clouds obtained from SfM techniques to align query images. Sibbing et al. [46] explore how point cloud rendering techniques can be used to create virtual views to extract features that match real image-based features as closely as possible. These features are used to establish correspondences for camera pose estimation. In their approach, they use a database to compute query images. To estimate the camera pose, they find matches between 2D features and 3D points in the model by using a 3-point-pose algorithm inside a Random Sample Consensus (RANSAC) loop. To localize a query image, their database creates a sample of rendered views of the point cloud. The computed viewpoints are similar enough to the real image to enable feature matching. In our work, we use a similar approach where rendered images are created as an input for our cost function. However, we use the whole image instead of only feature observations. Additionally, our approach is not only a localization, but also

a camera tracking technique.

2.2 SLAM

SLAM has received much attention in the Augmented reality community in the last years. The approach refers to a set of methods to solve the pose estimation and 3D reconstruction problem simultaneously, while a system is moving through the environment [38].

Initial work by Davison et al. [11, 24] demonstrated that a system using a single camera is able to build a 3D model of its environment while also tracking the camera pose. While SLAM provides an inherent tracking solution, it does not provide any reference to a known, global location. Therefore, information that is referenced to such a real location, for example through a GPS position, cannot easily be rendered in a purely SLAM-based system.

Visual SLAM systems use the camera itself to determine device position, by tracking and mapping detectable features in the surrounding environment [52]. Davison et al. [11, 12] were the first to propose monocular SLAM using a filtering approach. Klein and Murray proposed a keyframe-based SLAM [24]. In this approach, keyframes are sampled from the camera and processed in a background thread to produce a point cloud reconstruction (the map). In general, monocular SLAM provides high accuracy camera tracking in real-time, and is even capable of running on mobile phone platforms [25]. However, the camera pose is only given in a local reference system, defined with respect to the first camera frame or an initialization target [10]. Lothe et al. register a SLAM map captured from a moving vehicle to a polygonal 3D model, but require an initialization provided by GPS or manual input [29].

The Parallel Tracking and Mapping (PTAM) system by Klein and Murray [24] and its extension to multiple maps [7] is efficient enough to work on smartphones [25, 53]. In this approach, the user moves a handheld camera around the scene, whereas the 3D map is built from scratch and used for tracking and annotation. While it is a powerful approach for small indoor scenes, this approach has limitations for larger outdoor scenes. The main issue is that the system requires careful movement of the camera to ensure enough baseline for 3D point triangulation, especially in the outdoor case.

A few previous works also use some combination of mapping and tracking with global localization. Arth et al. use visual orientation tracking on a client and perform 6DoF localization from the resulting panorama as a background task [2, 3]. At startup, the user is required to explore the environment through the camera's viewfinder.

SLAM [8, 55] systems are real-time, but their performance degrades in larger scenes, where map maintenance becomes progressively expensive. These techniques are also fragile, if the camera moves too quickly, which makes them less attractive for persistently computing a precise camera pose over longer durations.

Newcombe et al. demonstrated in their work a real time camera tracking and reconstruction framework based on detailed depth maps [33]. They use hundreds of images from

a video stream captured with a hand-held RGB color model (RGB) camera to produce a set of surfaces. In their approach they immediately use the created model to track the camera's 6DoF pose by image alignment against the dense model. Their approach works in real time on the graphics processing unit (GPU) and demonstrates that a dense model provides reliable tracking under rapid motion.

SfM methods (alternatively referred as Monocular SLAM) primarily work with feature-based models of the world. In a first step, feature-based methods observe a set of features from a query image. In a second step, the camera position and scene geometry is computed as a function of these feature observations only. Therefore, feature based-methods discard all other information which is present in an image. Research show that using a feature-less dense method provide a more complete, accurate and robust reconstructing and tracking.

Engel et al. demonstrate a feature-less monocular SLAM framework [16]. Their approach locally tracks the motion of a camera to build a large-scale map of the environment. Sensors like depth or stereo cameras have limited range at which their measurements are realizable. Therefore, they are not as flexible as a SLAM approach. Schoeps et al. use a direct monocular visual odometry system which runs in real-time on a smartphone [45]. As a direct method, it tracks and maps on the images themselves instead of extracting features from the images. Images are tracked using direct image alignment, while geometry is represented as semi-dense depth map. Similar to Newcombe et al., both papers use a semi-dense SLAM system to increase the robustness of the camera pose estimates compared to sparse feature-based tracking systems.

In a nutshell, there are two different approaches to process SLAM:

- Panoramic SLAM
  - This approach only provides rotation, and, as result, the user must stay in one place
  - Works instantly
  - Localization success rate is strongly correlated to aperture angle
- Full 6DoF SLAM
  - In this case, the user can move freely
  - As disadvantage, a baseline is required, therefore, the user has to walk several meters

2.3 Contribution

The main question is: What do users expect from AR applications? Schmalstieg provides a list of requirements for location based AR applications[1]:

- An application that works on smartphones, tablets or eyeglasses
- A localization in real time with 30Hz or less than 100ms latency
- The accuracy is expected around one cm and less than one degree
- The system works indoor and outdoor

State-of-the-Art computer vision based approaches are assisted by inertial sensor data and provide prune search space with sensor priors:

- GPS: only search near position prior
- Compass: only search in approximate heading
- Accelerometer/Gravity: Only consider features with right orientation

Our outdoor localization system provides an accurate camera pose in real time with no assistance from the user. Pruning the search space with sensor priors provides a first estimation of the camera pose. The user is not required to stay in one position, to move several meter to get a baseline or to rotate around a position. The current version of our application works on a desktop computer and imports a recorded video stream and sensor data from a smart device. Performance measurements indicate that a mobile version is feasible.

As reference for our vision-based computations, our localization framework requires a geo-registered 3D model with textures as reference model. For this project, a model of the Hauptplatz in Graz, Austria, the Tummelplatz in Graz, Austria, and geo-referenced images are provided[2]. In the test areas, we recorded multiple videos and their respective sensor data with a smartphone. Tests are applied under different lighting conditions and with more or less objects occluding the background to test the robustness of our approach. The captured videos and its corresponding sensor data are transfered to a desktop computer where our algorithm is executed. With our testcases, we demonstrate the localization and tracking ability of our approach.

In our opinion, a framework which demonstrates the current state of the art in particle filter based localizations is a valuable contribution to the current research in this field. We can demonstrate that, given today's hardware it is possible to compute the pose of

[1] UMIC Day 2014, `https://www.umic.rwth-aachen.de/uploads/media/UMICDay2014_Schmalstieg.pdf`

[2] Christian Doppler Laboratory for Handheld Augmented Reality at Graz University of Technology, `https://handheldar.icg.tugraz.at/index_detailed.php`

a smartphone with a particle filter that uses hundreds of particles each second. Today's GPUs and CPUs are powerful enough to compute a brute force method. Another benefit of our approach is that the approach requires no support from the user. The size of our virtual reference model is about 3 MB. This is another advantage, because this is an amount of data a consumer mobile phone can download easily from a server. Furthermore, our approach requires little memory space, because our system only saves the parameters required for the camera poses.

3 System overview

Contents

Our outdoor localization system provides accurate localization in real time with no assistance from the user. Pruning the search space with sensor data provides a first estimation of the camera pose. The user is not required to rotate around a fixed position or move several meters in order to get a baseline initialization. The current version of our application works on a desktop computer, but was tested on video and sensor data from a smartphone. Our desktop application performs the localization process in real time with 30Hz.

In our approach, a sophisticated system is designed to accomplish an accurate localization process in real time. The framework imports a textured model of the environment and data from a smartphone. These data include video stream and the corresponding camera pose of each frame. The pose consists of position received from the GPS unit and orientation obtained from the gyroscope. Our approach uses a particle filter where each particle consists of a camera pose and an importance value describing the likelihood of each particle.

3.1 Sensor Data

The system imports the input data and displays the video stream, the camera pose in the virtual environment according to the data from the smartphone and the corrected results from the particle filter.

In our framework, we import two types of items:

1. The rendering of the virtual environment requires a geo-referenced model of the environment and geo-referenced panorama images for texturing the model. Our test

areas are the Hauptplatz in Graz, Austria, shown in Figure 3.1 and the Tummelplatz in Graz, Austria. Required memory for the geometric model and the panorama images is about 3MB. The images used for texturing are handpicked in a way such that the whole area around the user is textured, e.g., eight textures are used for the virtual model of the Hauptplatz.

2. Data collected from the smartphone includes the video stream and corresponding device poses for each frame. The device delivers intrinsic camera data for the focal length f, center of projection along the x-axis (c_x) and along the y-axis (c_y). For extrinsic parameters, the device delivers data for latitude and longitude from the GPS unit in the WGS84 format and the attitude matrix A (equation 3.2) for yaw, pitch and roll angles from the IMU as rotation matrix. We map the GPS position to UTM coordinates.

In our system, we represent the device pose c using OpenGL's gluLookAt format, which consists of

- $eye = (x_{eye}, y_{eye}, z_{eye})$
- $center = (x_{center}, y_{center}, z_{center})$
- $up = (x_{up}, y_{up}, z_{up})$

These vectors represent the camera center, camera viewing target, and camera up direction, respectively. This representation is redundant, since a camera pose has only 6DoF. However, as will become evident below, this representation is very suitable for our purposes, since it allows us to concentrate sampling on a subspace of the 9DoF used in this notation.

Our frameworks uses the UTM coordinate system, and it is necessary to pre-process the input data and to project the data into UTM. For data capturing, we used an Apple iPhone 4.

The attitude output from Apple iOS is: +X is north, +Y is west, +Z is up, whereas UTM uses: +X is east, +Y is north, +Z is up.

Therefore, we need to apply 90 degree turn (equation 3.1) to Apple iOS attitude.

UTM conversion:

$$U = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{3.1}$$

The attitude A received from the smartphone:

$$A = \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{32} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix} \tag{3.2}$$

To get the *center* and *up* vector results we compute the up and center vector in iPhone axes (equation 3.3) and world coordinates (equation 3.4):

$$\begin{aligned} up_{world} &= A^{-1} * \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \\ center_{world} &= A^{-1} * \begin{pmatrix} 0 & 0 & -1 \end{pmatrix} \end{aligned} \tag{3.3}$$

$$\begin{aligned} up_{UTM} &= U * up_{world} \\ center_{UTM} &= U * center_{world} \end{aligned} \tag{3.4}$$

The current version of our application works on a desktop computer and is based on OpenSceneGraph (OSG)[1] and Open Source Computer Vision (OpenCV)[2]. OSG is responsible for rendering the virtual environment which is processed on the GPU. To get the importance of each particle, vision based methods are used to compare the rendered image to the real world image for each frame. This is computed on the central processing unit (CPU) using the OpenCV library.

3.2 Virtual City Model

For our vision-based localization approach, a virtual model of the environment is required. To render the model, our framework imports a geo-referenced model obtained from a SfM pipeline and geo-referenced panorama images for texturing the model. One of the test areas is Hauptplatz in Graz, Austria, shown in Figure 3.1. The other test area is Tummelplatz in Graz, Austria. Data is provided by the Christian Doppler Laboratory for Handheld Augmented Reality, Graz University of Technology[3]. The provided data includes a geo-referenced untextured model of the first district in Graz and geo-referenced images with their respective camera pose. Images are recorded from a street side view. The 3D model was constructed using large-scale SfM from panoramas and alignment with a LiDAR city model.

[1] `http://www.openscenegraph.org/`

[2] `http://opencv.org/`

[3] Christian Doppler Laboratory for Handheld Augmented Reality at Graz University of Technology, `https://handheldar.icg.tugraz.at/index_detailed.php`

Figure 3.1: The system uses a solid geo-referenced model (left) and panorama images (middle) as input. Using projective texture mapping, both input data are combined to a virtual model of the environment (right).

To compute the virtual environment, our framework requires a geo-referenced model and a set of i geo-referenced images to compute the virtual model (table 3.1). The format of the imported data is as follow, where we denote f as the focal length of the camera, I_w as the image width and I_h as the image height:

Untextured model [.ply format]			
f	I_w	I_h	
filename$_i$	$\mathbf{eye}(\mathbf{x},\mathbf{y},\mathbf{z})_i$	$\mathbf{center}(\mathbf{x},\mathbf{y},\mathbf{z})_i$	$\mathbf{up}(\mathbf{x},\mathbf{y},\mathbf{z})_i$

Table 3.1: Input data to compute virtual city model. This includes an untextured geo referenced model of the environment, parameters of the camera and geo referenced textures used for projection.

The imported virtual model is stored in the .ply format. For projective texture mapping, we use the OSG class osg::TexGen[4].

To construct this class, a viewing matrix and a projection matrix is required. The viewing matrix is computed by Open Graphics Library (OpenGL) function gluLookAt[5] and uses *eye*, *center* and *up* as parameters (see Table 3.1). For the projection matrix, the framework uses a symmetrical perspective projection provided by the OpenGL function gluPerspective[6]. The field of view angle, in radian, in the y direction (fovy) parameter used for gluPerspective is computed according to equation 3.5.

$$\text{fovy} = 2 * \tan(I_h/(2 * f)) \tag{3.5}$$

Next to the virtual model, data from a smartphone is imported (table 3.2). The sensor data include position and orientation to calculate the viewing matrix. The parameters for the projection matrix include the fovy parameter, which is set according to the camera data and the aspect ratio according to the image size. zNear is set to 0.1 and zFar is set to 100.

[4] `http://trac.openscenegraph.org/documentation/OpenSceneGraphReferenceDocs/a00868.html`
[5] `https://www.opengl.org/sdk/docs/man2/xhtml/gluLookAt.xml`
[6] `https://www.opengl.org/sdk/docs/man2/xhtml/gluPerspective.xml`

Input data from smartphone include a set of images j and their respective camera poses.

The format of the imported data is as follow:

focal		I_w	I_h	
filename_j	time index_j	$\mathbf{eye}(\mathbf{x},\mathbf{y},\mathbf{z})_j$	$\mathbf{center}(\mathbf{x},\mathbf{y},\mathbf{z})_j$	$\mathbf{up}(\mathbf{x},\mathbf{y},\mathbf{z})_j$

Table 3.2: Input data from the smartphone includes images and information from GPS and gyroscope to compute the pose and parameters of the camera. Our system supports the processing of images or videos.

Our model uses geo-referenced images for projective texture mapping to provide a reference model for the image queries of the particle filter. Our projection model works very well for facades of buildings as seen in Figure 3.2, if the query image has a similar pose compared to the geo-referenced input image. But the computed model has issues caused by projecting textures, if the angle of the projection is too big. Further issues are that our testcases are limited to the area around the position of the input images, e.g., if the user is too close to the buildings, images lack details. But not only the images restrict our possible testcases, the model itself has few polygons, which is an advantage regarding performance, but a disadvantage in regard to details. As demonstrated in Figure 3.2 (right, top), the fountain is hard to recognize.

Figure 3.2: Our projection model works very well for facades of buildings (left, top). None of the used geo-referenced images is recorded close the a building, therefore, the image quality is reduced near buildings (left, bottom). Other issues are the low polygon count, which makes it hard to recognize the fountain (right, top) and problems caused by projective texture mapping (right, bottom).

4 Particle Filter

Contents

We use a particle filter to solve the localization problem. The algorithm processes the live video stream and produces a corrected pose estimate in real-time.

A particle filter is a simulation-based method for tracking a system with partially observable state. We briefly review particle filters in this section. The reader is referred to Thrun et al. [49] for a more detailed review. A particle filter maintains a weighted (and normalized) set of sampled states, $S_t = \{\langle x_t^i, w_t^i \rangle \; i = 1, ..., N\}$, called particles p_t^i, where x_t^i is the state of the i^{th} particle, and w_t^i is its weight. The weight of a particle can be thought of as the probability that the particle's state corresponds to the true state of the system. In our case, the state is the configuration of the viewing matrix using the gluLookAt format described in section 3.1. The steps of one particle filter iteration are as follows:

1. Each particle is propagated according to our propagation model
2. Their weights are computed by comparing rendered and real world frames
3. The top ranked particles are re-sampled to obtain a new set of equally weighted particles, which approximates the new posterior distribution

Re-sampling causes more particles to be generated in areas of the state space that are likely to correspond to the true state of the system. In our system, one particle filter iteration is processed for every new frame when it arrives.

Each particle consists of a camera pose, shown in Figure 4.1, and the associated weight describing the likelihood of each particle to be the correct pose of the camera. The state

space has the nine dimensions, consisting of the components of the vectors *eye*, *center* and *up*.

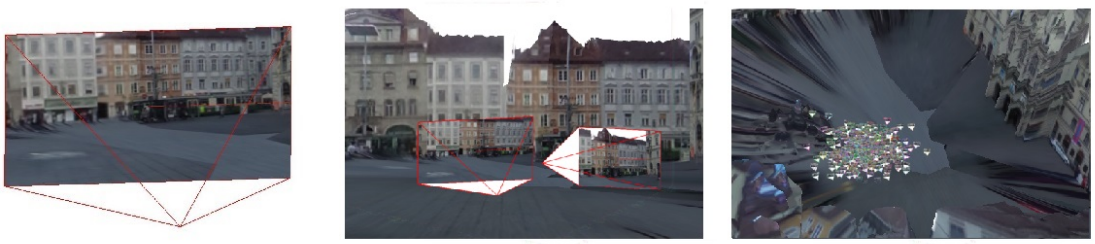

Figure 4.1: Each particle has a camera pose (left) and its weight. To set up the particle filter, multiple particles are computed (middle, right).

4.1 Particle Dimensions

The initial sensor data from the smartphone are error-prone and, therefore, correction is necessary. One approach to correct the position is to search in the local neighborhood to estimate the real camera pose. To process this step, the particle filter requires a distribution to propagate the particles and the quantity N of particles propagated for each frame. One disadvantage of the particle filter algorithm is the poor performance, if the dimensions of the observed state are high. In our case, we have nine dimensions. To reach real time performance, we use a particular distribution model with the following properties:

- We assume a constant altitude for value z_{eye}, because the GPS reading is noisy and the phone is typically held at roughly the same height off the ground at all times. In comparison to the distance to the buildings, the variation in camera height is negligible, so constant camera height was chosen to reduce the number of particles needed.
- It is expected that the smartphone is not rolled, and, therefore, *up* barely changes.
- We further expect the user to look at the facade of the buildings and not to the ground or sky; therefore, z_{center} barely changes.

As result, the values of $x_{eye}, y_{eye}, x_{center}$ and y_{center} have the most impact on the algorithm.

4.2 Motion Model

In this discussion, we refer to the device pose delivered by the internal GPS/IMU sensors as the 'sensor pose'.

Each particle p_t^i is formed by perturbing a prior pose by a sample from a statistical motion model. Similar to the work of Klein and Murray [23], we sample these motions from a Gaussian noise model. For the initial step, we propagate particles as follows:

$$p_{t,d}^i = c_{i,d} + \mathcal{N}(0, \sigma_d) \tag{4.1}$$

where p_t^i is the particle, $c_{i,d}$ the d-th dimension of the sensor pose c_i of the i^{th} frame and σ_d the standard deviation of the noise added for that dimension. For the following re-sampling steps, c is replaced by the state of one of the most probable particles from the previous iteration.

$$p_{t,d}^{i+} = p_{t,d}^{i-} + \mathcal{N}(0, \sigma_d) \tag{4.2}$$

where p_t^{i+} is a particle for the current frame, p_t^{i-} is a probable particle of the previous frame, d the dimension and σ_d the standard deviation of the noise added for that dimension.

For each frame, changes in the sensor data are measured. The most probable particles p_t^{i-} are duplicated and modified with the sensor changes between the current and the previous frame.

$$p_t = p_t^{i-} + (c_i^+ - c_i^-) \tag{4.3}$$

where c_i^- is the sensor pose of the previous frame and c_i^+ the sensor pose of the current frame. The unmodified sensor pose is also added as a particle.

In summary, for maximum robustness, our particle re-sampling includes (a) the most likely particles from the previous iteration (b) the most likely particles, modified with changes between the current and last frame according to the sensors and (c) the unmodified sensor pose.

4.3 Calculating Particle Weight

To compute the weight of a particle, we compare an image rendered at the particle's pose to the current image from the device camera.

We use two different weighting approaches in order to capture color, texture and edge information. The first approach uses the sum of squared distances (SSD) measure to calculate the pixelwise difference in color between a rendered image and a frame from the smart device, shown in Figure 4.2. The second approach uses SSD to compare the image gradient magnitudes, shown in Figure 4.3.

The image SSD method, as shown in Figure 4.2, compares the rendered image to the image captured from the smart device, with a mask applied. We mask out the part of the rendering corresponding to the sky (pixels not belonging to any scene geometry), specified as a binary mask $m_j \in \{0, 1\}$. We compute the SSD for each particle p_i as follows:

$$e_I^i = \frac{\sum_j m_j \cdot ||I_j - V_{i,j}||^2}{\sum_j m_j} \tag{4.4}$$

where e_I^i is the error, I is the camera image, and V_j is the rendered image for particle i.

The gradient magnitude SSD method, as shown in Figure 4.3, computes the average difference between the gradient magnitudes of the rendered image and the masked gradient magnitudes of the camera image as follows:

$$e_G^i = \frac{\sum_j m'_j \cdot (||G_j^I|| - ||G_{i,j}^V||)^2}{\sum_j m'_j} \tag{4.5}$$

where e_I^i is the error, G^I contains the gradient magnitudes of the camera image, G_i^V contains the gradient magnitudes of the rendered image for particle i, and m' is the binary mask after applying a dilation operator. The dilation helps to avoid removing important edges on the contours of the buildings. The gradients are computed using the Sobel operator applied to the rendered and the real world image.

The results of these operations are two measures of the average error between the rendered and the real world image. The error value for each method is normalized so that

$$\sum_{i=0}^{N} e_I^i = 1, \quad \sum_{i=0}^{N} e_G^i = 1 \tag{4.6}$$

The weights are then calculated as

$$w_I^i = \exp(-e_I^i), \quad w_G^i = \exp(-e_G^i) \tag{4.7}$$

and normalized so that

$$\sum_{i=0}^{N} w_I^i = 1, \quad \sum_{i=0}^{N} w_G^i = 1 \tag{4.8}$$

The final weight is calculated simply as the sum of the weights from the two methods:

$$w^i = w_I^i + w_G^i \tag{4.9}$$

The particle with the highest combined weight w^i is considered the best estimate of the device pose, i.e., the pose to be used for AR rendering.

Before computing the error functions and weights, all images under consideration are downsampled. This helps to avoid or reduce artifacts caused by the differences between the video stream from the smart device and the images used to create the 3D environment model. Real world objects like people, cars, differences in the lighting condition or camera artifacts can cause artifacts in the images. Another advantage of small images is the improved performance, because the particle filter algorithm requires hundreds of images to estimate the correct camera pose.

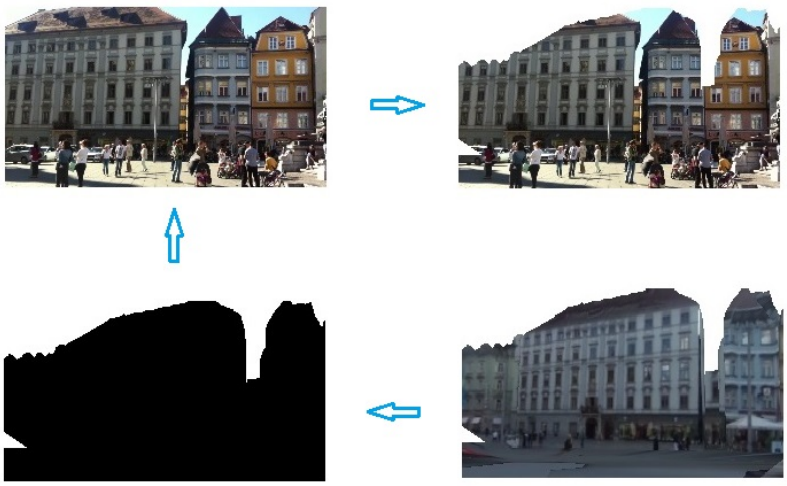

Figure 4.2: The SSD method compares a rendered image to a masked image captured from the real world video stream (right, bottom). In a first step, a masked image is created that discards the sky from the virtual image (left, bottom). White pixels are not used for comparison. The real world image (left, top) is combined with the mask, and the result (right, top) is compared to our rendered image. The similarity between the masked, rendered image and the input camera image contributes to the weight of the particle.

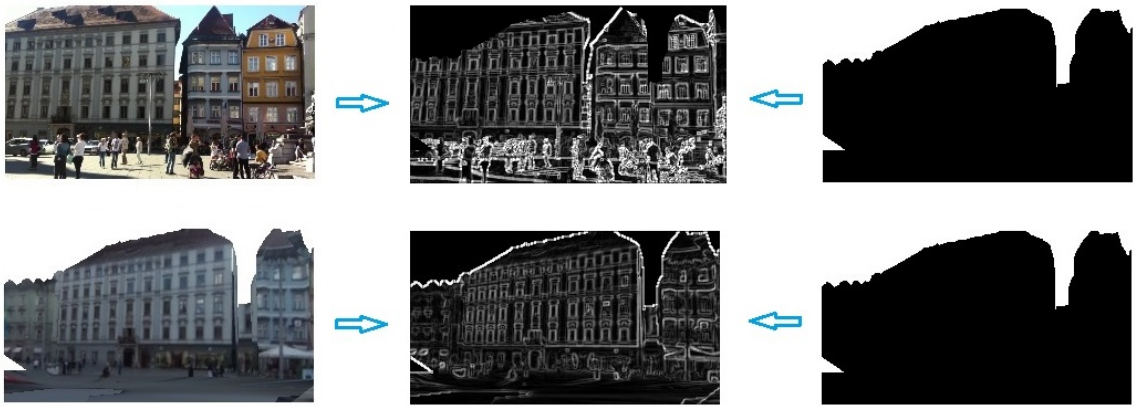

Figure 4.3: The gradient magnitude SSD method compares image gradients. A Sobel operator is applied to the rendered image and the real world image (left). Pixels in the sky in the rendered image (right) are masked out in both images. The similarity between the masked gradient images (middle) contributes to the weight of each particle.

5 Processing Pipeline

This section discusses the processing pipeline and the required computation time for each step. Our system is proposed to work at 30 Hz, therefore, it is necessary to identify the computation time of every step.

The particle filter process can be divided into two separated parts. In our approach, two threads are responsible for computing. In a main thread, images are rendered for each particle. A working thread is executing the computation of the errors for both methods. The first step is to render all particles for one frame and to compute their respective errors. In a second step, it is necessary to compute the weight of each particle to get the most likely result.

To measure the performance test are made with our reference system (Intel Core i7-4770, 16 GB RAM and an AMD Radeon R9 280X), and the images for the particles have a width of 100 pixels and a height of 56 pixels.

For the first step, our system renders the images required for initial and update step on the GPU and computes the error of each particle according to equations 4.4 and 4.5 on the CPU.

Required computation time for the first step on our reference system is about one millisecond for each particle (table 5.1). Although the CPU is responsible for image based computation, rendering each particle requires more time, as shown in table 5.1. Performance measurements show that computing e_G^i requires significantly more time than computing e_I^i. The reason is that we use the CPU to execute the Sobel operator.

The second step for each frame is to compute the particle weight on the CPU according to equation 4.7. The required computations times for this step with our reference computer are shown in table 5.2. The step rendering viewports is not part of the particle filter algorithm, but it is included in the framework to present the results.

Our reference system is able to compute nearly one thousand particles each second with parallel execution, as shown in Figure 5.1. Our execution pipeline performs with two parallel threads. The main thread is responsible for rendering and computing the

Process	Time in ms
Main thread	
Rendering particle	≈ 1.01
Copy texture from GPU to CPU memory	≈ 0.1
Working thread	
Compute e_I^i	≈ 0.26
Compute e_G^i	≈ 0.61

Table 5.1: Several steps are necessary to execute the the particle filter algorithm. In our approach two threads are responsible for computing. The main thread is responsible for rendering for each particle. A working thread is executing the computation of the errors for both methods in a parallel process.

Process	Time in ms
Main thread	
Update step	≈ 2.10
Rendering viewports	≈ 3.30

Table 5.2: In a final step the particles weights are computed and presented. The third process is not part of the particle filter algorithm but it is included in the framework to present the results. There are three viewports: The first view shows the rendering of the virtual scene with the pose from the particle filter. The second viewport presents the scene with the pose received from the smartphone, whereas the third viewport shows the video stream from the smartphone.

weights. A working thread is computing the Sobel operator required for image gradient magnitudes, e_I^i and e_G^i according to equations 4.4 and 4.5 for each image.

Performance measurements show that the GPU load is approximately 66%, which indicated that there is space left for performance improvements. The required memory is about 100 MB.

To render the virtual model, we use projective texture mapping at runtime. For future improvements we recommend to import a textured model, which will increase rendering speed up to 10%, because currently the projective textures are computed at each frame.

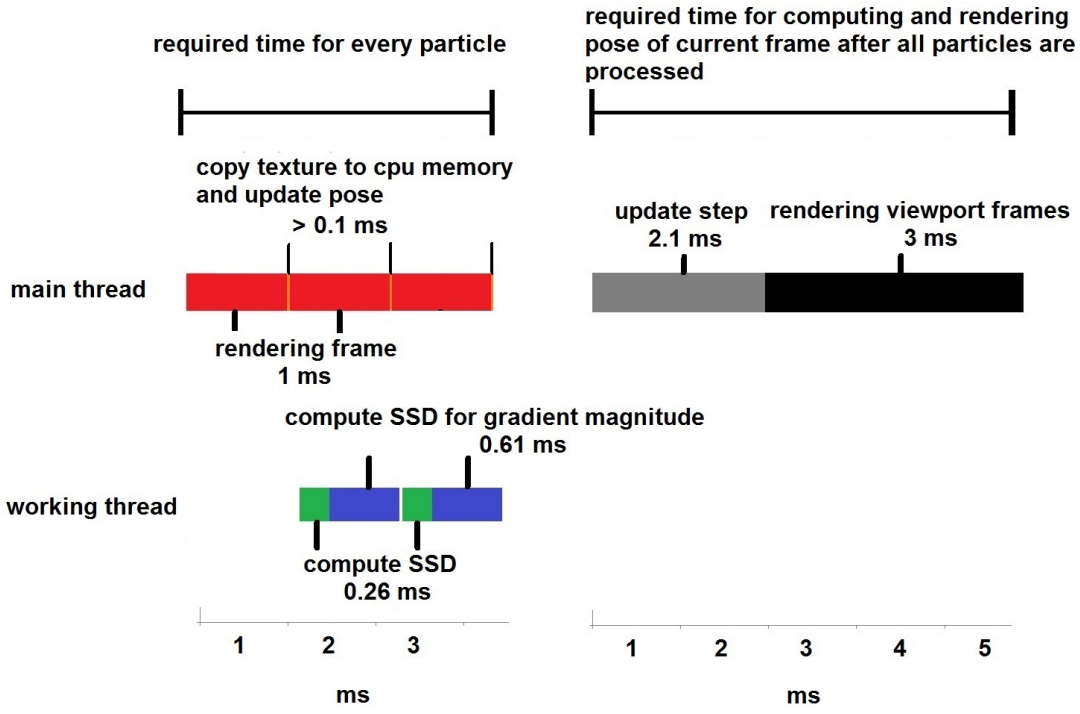

Figure 5.1: Our execution pipeline performs with two parallel threads. The main thread is responsible for rendering and computing the weights. A working thread is computing the error e_I^i and e_G^i according to equations 4.4 and 4.5 for each image.

6 Experiments

Contents

Our particle filter approach provides accurate localization in real time with no assistance from the user. Pruning the search space with sensor information provides a first estimation of the camera pose. The user is not required to rotate around a fixed position or move several meters in order to get a baseline initialization.

In this chapter, we test the capabilities of our system in two different test areas. The test areas for our particle filter system are the Hauptplatz and Tummelplatz in Graz, Austria. Video, GPS and IMU data from a smartphone are recorded under weather and environment conditions which differ from the provided panorama images used for our virtual reference model. In our tests, the operator moved freely in the area, but avoided pointing the camera toward the ground or the sky. Rolling of the camera (rotation around the optical axis) was also avoided. In total, we recorded sixteen videos, where four videos were discarded due to very bad sensor data. This is the case when the position is far more than ten meters off the real position.

The hardware and software for recording data and processing our particle filter are as follows:

For recording GPS, IMU data and the video stream we used an Apple iPhone 4. To process the particle filter, we used a desktop computer with an Intel Core i7-4770, 16 GB RAM and an AMD Radeon R9 280X. Furthermore, we used a laptop with an Intel Core i5-3117U, 8 GM RAM and a NVidia Geforce 620M.

Dimension	σ_d	Approx. range
x_{eye}, y_{eye}	5.5	up to 9 meters
x_{center}, y_{center}	0.15	up to 9 degrees
$z_{center}, x_{up}, y_{up}, z_{up}$	0.085	up to 5 degrees

Table 6.1: Noise parameters for the initialization step

The current version of our application uses OSG for rendering and OpenCV for image processing.

6.1 Implementation Notes

This section provides some details about the particle filter configuration for the initial step and the update step. The initial step is responsible for an accurate localization. Therefore, the noise parameters should be higher and the initial step requires more particles to compute an accurate first estimation of the pose. The update steps compute with lower noise parameters and less particles to provide real-time tracking. This approach works well, if the initial step finds the correct pose.

For each particle, the rendered frame from the virtual environment is compared to the image from the smartphone. Each frame is scaled to a width of 100 pixels, whereas the height depends on the aspect ratio of the smartphone image and is, in the case of an Apple iPhone 4, 56 pixels. Our system is initialized with one thousand particles. The chosen noise parameters for the initial particle distribution governed by Equation 4.1 are given in Table 6.1. The initial values are based on empirical data[1].

This setup provides a lot of freedom for the dimensions $x_{eye}, y_{eye}, x_{center}, y_{center}$. In our test cases, the user can walk and look around freely. The only restrictions are to avoid rolling the smartphone and not to direct it towards the ground or into the air. The restrictions help to reduce the dimensions of our particle filter system. As a result, fewer particles are necessary, which reduces computation time.

In the subsequent update iterations, the three most probable particles p_t^1, p_t^2, p_t^3 and the sensor pose c_i are used for re-sampling. The chosen noise parameters for the particle motion model governed by Equation 4.2 are given in in Table 6.2. To reach a real-time performance, only the initial step has more freedom in each dimension, because this step is responsible for finding the correct camera pose.

In the update step, additional particles are generated according to equation 4.3 and modified with the motion detected by the inertial sensors. In total, for our testcases the update step uses only 13 particles. As a result, the update frames require significantly less computation than the first initialization frame. This enables real time tracking even on slower computer systems. Our test videos have 450 frames each, so altogether 6850

[1] `http://www.nstb.tc.faa.gov/reports/PAN86_0714.pdf`

Dimension	σ_d	Approx. range
x_{eye}, y_{eye}	0.36	up to 60 centimeters
x_{center}, y_{center}	0.01	up to 0.5 degrees
$z_{center}, x_{up}, y_{up}, z_{up}$	0.01	up to 0.5 degrees

Table 6.2: Noise parameters for the update steps

particles are evaluated for a 15 second long video at 30 Hz. To work in real time the device has to compute approximately 450 particles each second.

The system configuration is based on our twelve test videos. To ensure that our system works with untested videos, we recommend to apply supervised learning to configure the parameters. Considering that the sensor pose and the real position according to the video stream is available, supervised machine learning algorithms should be able to optimize our parameters.

6.2 Image-based Results

We tested our system with twelve videos recorded in our test areas. Figure 6.1 shows sampled frames from the test videos alongside renderings of the virtual environment, comparing the best pose from the particle filter to the uncorrected sensor pose. As can be clearly seen in these examples, our approach significantly improves the camera pose and offers much better real-virtual alignment than the raw camera pose delivered by the GPS and IMU sensors.

We also found that our system is robust to image artifacts like shadows and the presence of objects that are not in the virtual environment, like cars or people. Examples of occluding objects are shown in the first three rows of Figure 6.1, where occluding people, street cars, and street carts are present in either the real image or the virtual image. Rows four and five show examples of severe shadowing in the real image, which is correctly handled by our system and does not affect the camera pose estimate.

In most cases, the rendering of the best particle closely matches the input camera image. However, we found that, in some cases, the estimated camera pose exhibits an error of up to the sensor pose. We found that, because the 3D model is textured with projective texture mapping, the localization result is negatively affected, if the camera's viewing angle differs greatly from the viewing angle of the images used to make the 3D model. Another issue is that the initial step does not guarantee to provide a good first estimation of the pose due to random sampling. The update step relies on a accurate result from the initial step. But, even with a bad initial start pose, results show that over time, the particle filter tends to converge. The nature of the random sampling process in the particle filter can also cause jittering in stationary scenes, for example, when two particles have similar weights, but slightly different poses.

Especially in the context of AR, this jitter will result in frequency movement of virtual

Figure 6.1: The input data for a pose query is an image from the smartphone (left) and sensor data from GPS and gyroscope. The device sensors are used as an initial guess (right). Our system provides an improvement of the initially guessed camera pose (middle). Note the robustness to issues such as occluding objects (first three rows) and shadows (fourth and fifth rows).

objects in the view of a user and will impact the user experience. Using more particles could reduce the jitter, but will not remove it entirely, as it is caused by the random nature of the particle filtering process. However, it should be possible to combine the localization method with some frame-to-frame tracking system that produces a smooth camera trajectory.

Depending on the characteristics of the sensors and the environment conditions, the particular noise parameters we chose might need to be increased or decreased. For example, when near building walls, GPS will produce worse location estimates, because of restricted satellite visibility and signal reflection. The magnetometer (used to produce a heading estimate) also is affected by nearby metal structures. If the accuracy of the pose estimate from the sensors decreases, the system might need to be modified to have a larger search space (by increasing the standard deviations in the propagation model) and to use more particles to represent the larger search space.

Although our system, as tested, is implemented on a desktop computer, we are confident that the approach could easily achieve real-time rates on mobile hardware. This is because most of the computation consists of either rendering of textured models with low polygon count, or simple image processing operations with small filter size. These kinds of operations are perfectly suited for the GPU currently available on smartphones or tablets.

6.3 Localization Accuracy

Image-based results look promising, but the remaining question is if the good image-based results are reflected in an accurate camera pose. This section investigates the accuracy of our particle filter algorithm. To measure the accuracy, we use different testcases.

The first and simple test uses geo-referenced images provided by the Christian Doppler Laboratory for Handheld Augmented Reality, Graz University of Technology. Other tests use data imported from a smartphone, where the position is looked up on Google Maps[2] and Bing Maps[3].

In the first testcase, see table 6.3, we evaluate the accuracy of the UTM position with one of our geo-referenced images. These images are exactly the same images used for texturing our city model. To simulate a smartphone, the sensor data is modified with a randomly generated noise factor. The results show that the particle filter algorithm finds a position very close to the true position. This testcase has twenty frames where every frame shows the same image. The required computation time on our desktop reference system is about three seconds. To evaluate the accuracy of our approach, this testcase uses one hundred particles for the update step.

One reason that the particle filter algorithm does not find the correct position is the low polygon count of the model, e.g., the roof of the buildings are more flat in the model

[2] `http://www.geoplaner.com/`
[3] `https://www.bing.com/maps/`

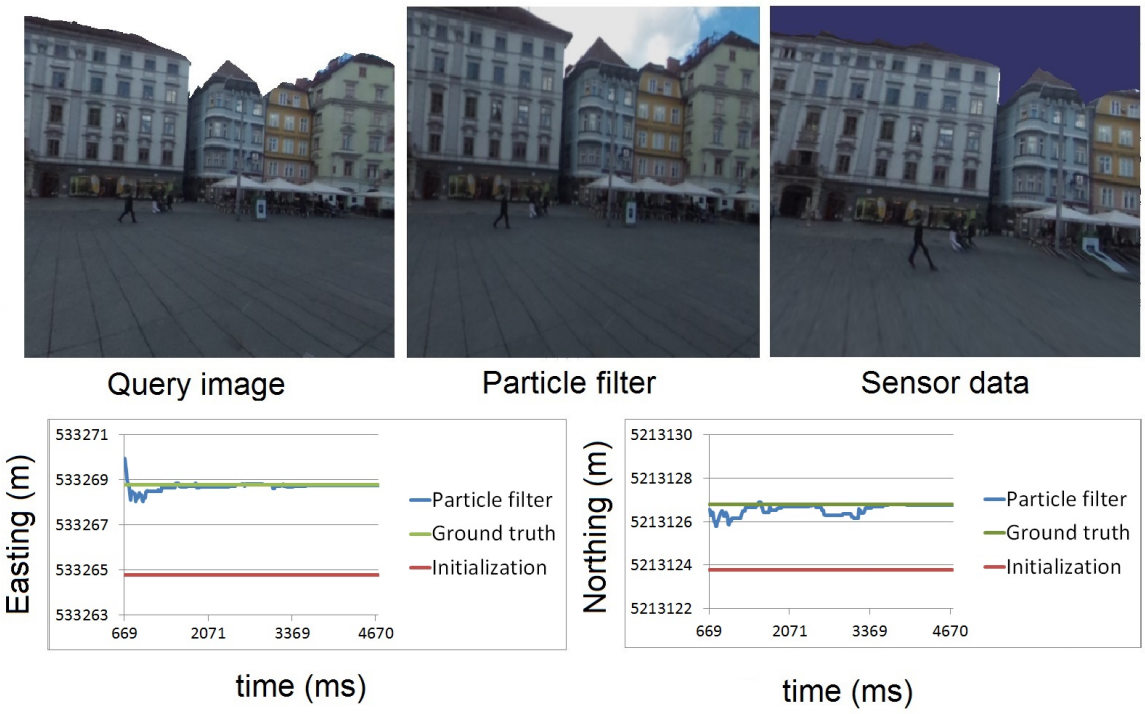

Table 6.3: In this testcase, we evaluate the accuracy of our algorithm. The input query (left image) is one of our geo-referenced images. To simulate a smartphone, the sensor data is modified with a randomly generated noise factor. The image on the left is our query image, our randomly generated noisy camera pose (right image) and the result of the particle filter (middle image).

compared to the real world. Another issue is that projective texture mapping projects parts of the sky onto the roof.

In the next testcase, we evaluated the accuracy of a video stream from our smartphone (table 6.4). For this test, we only used the first frame of the query video. For this testcase, the update step uses 13 particles for each frame, and the test duration was ten seconds on our desktop reference system. Results show how long it takes our algorithm to find the correct position. In most cases, our algorithm finds a pose very close to the real pose immediately or after a couple of frames. In rare cases, our algorithm is stuck near the sensor position. The reason is that the variables for the update step are very restricted and constant; therefore, it can happen that our algorithm is trapped near the initial position.

The results for the position of the whole video sequence is shown in table 6.5. In this case, the particle filter starts with a bad position. About three seconds are required to find a position close to the true position. It is noteworthy that the position is noisy, although the video was captured without moving. Our algorithm considers changes in the GPS position and the IMU according to equation 4.3. For the update steps it is necessary to distribute new particles near the pose, because it might take time to find the correct position. The disadvantage of this approach is that it causes jittering, which could be

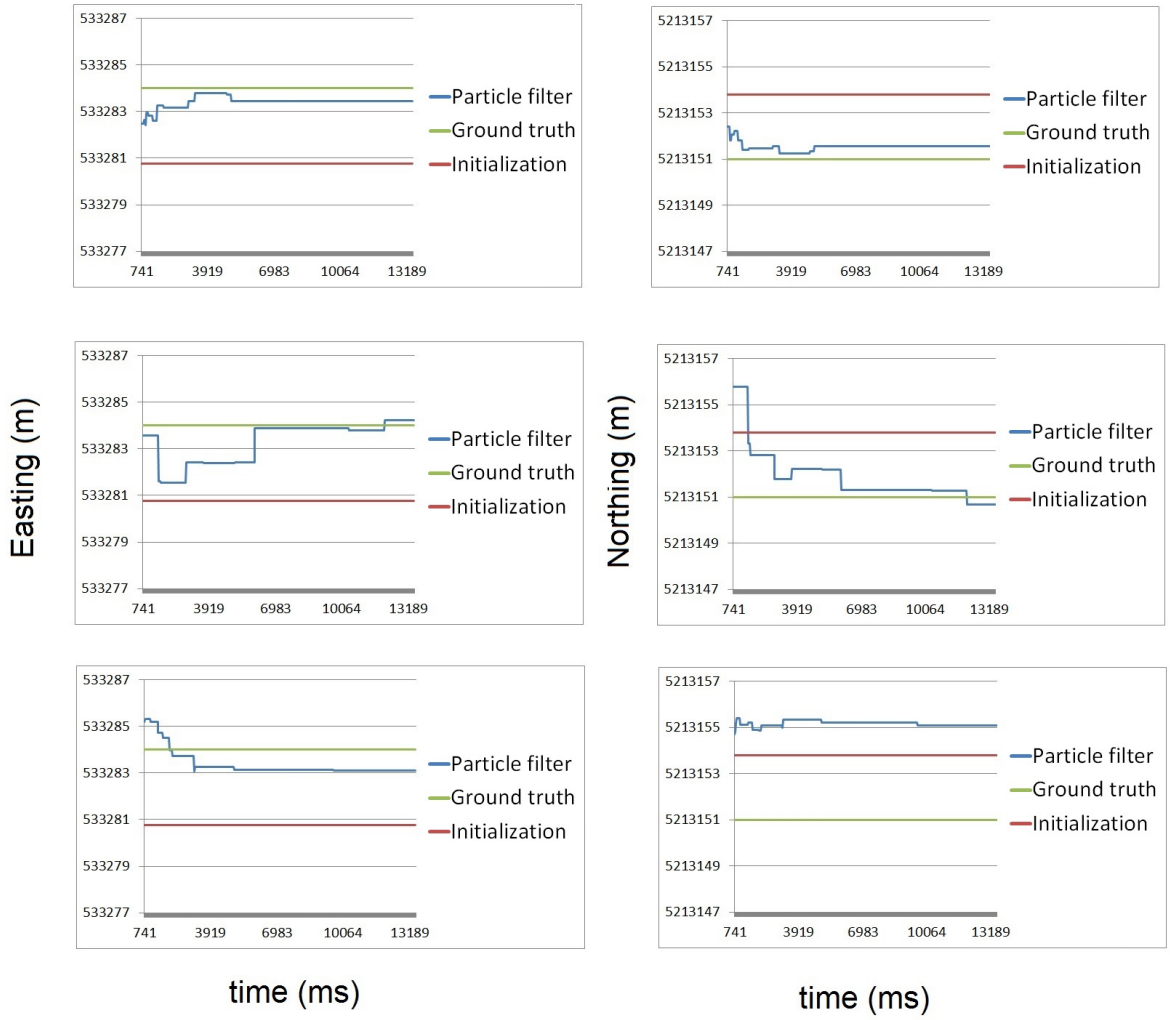

Table 6.4: Image-based results look promising. But in our approach, we use low detail polygon model and low resolution images to compute the weight of each particle. In this test, we compared the UTM position for x and y for one of our test videos. For the first test (row 1-3), we replaced every frame of the test video with the first frame. The test shows how long it takes our algorithm to find the correct position. Results show that the correct position is found immediately (first row). Rare case where the particle filter algorithm requires several iterations until a good position is found (second row). Rare case where the algorithm does not find the correct position (third row). The variables for the update steps have less freedom; therefore, it can happen that our algorithm is trapped near the initial position.

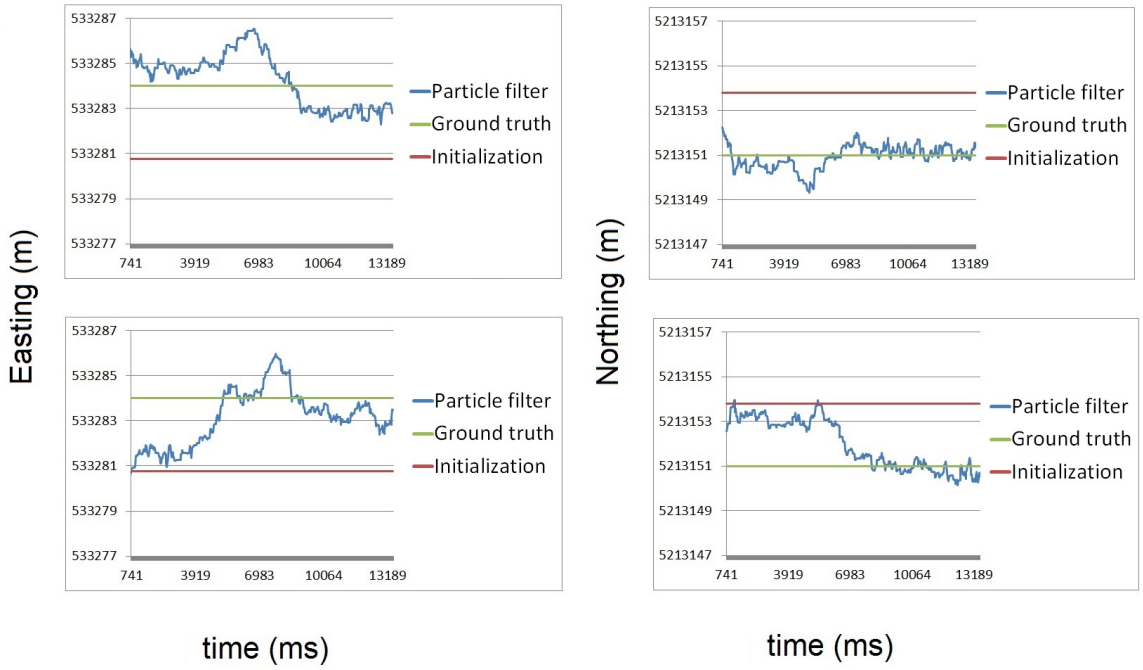

Table 6.5: The results for the position of the whole video sequence with a good initial guess (first row) and a bad initial guess (second row). It is noteworthy that the position is noisy, although the video was captured without moving.

prevented if the sensors suggest that there is no change.

Our approach considers the most likely position as the correct position. The most probably particle has the highest combined weight according to equation 4.9. Table 6.6 shows the particle weights for one of our test videos. The initial step is computed with 1000 particles. Results show that there are few particles with high and few particles with low weight. There is no significant difference between the weight of the particles. The update steps have 13 particles, and results show that the most likely particle has a weight only slightly above the average value for all particles. The particle weight for one of our test videos show that the weight values are very noisy (bottom). This result reflects the jittering effect visible in stationary scenes. Our results show that the jittering effect decreases over time, until no better pose can be found.

A disadvantage of our algorithm is that there is no reference value that determines a computed pose is correct. In our approach, a pre-determined number of particles is computed without adapting the number of samples over time, because there is no break condition. A reference value could be compared to the cost functions 4.4, 4.5 and the computed weight value for each particle according to equation 4.7. This would enable a break condition where the algorithm stops searching the state space for a more likely camera pose. Currently, there is no break condition, because results show that the weight

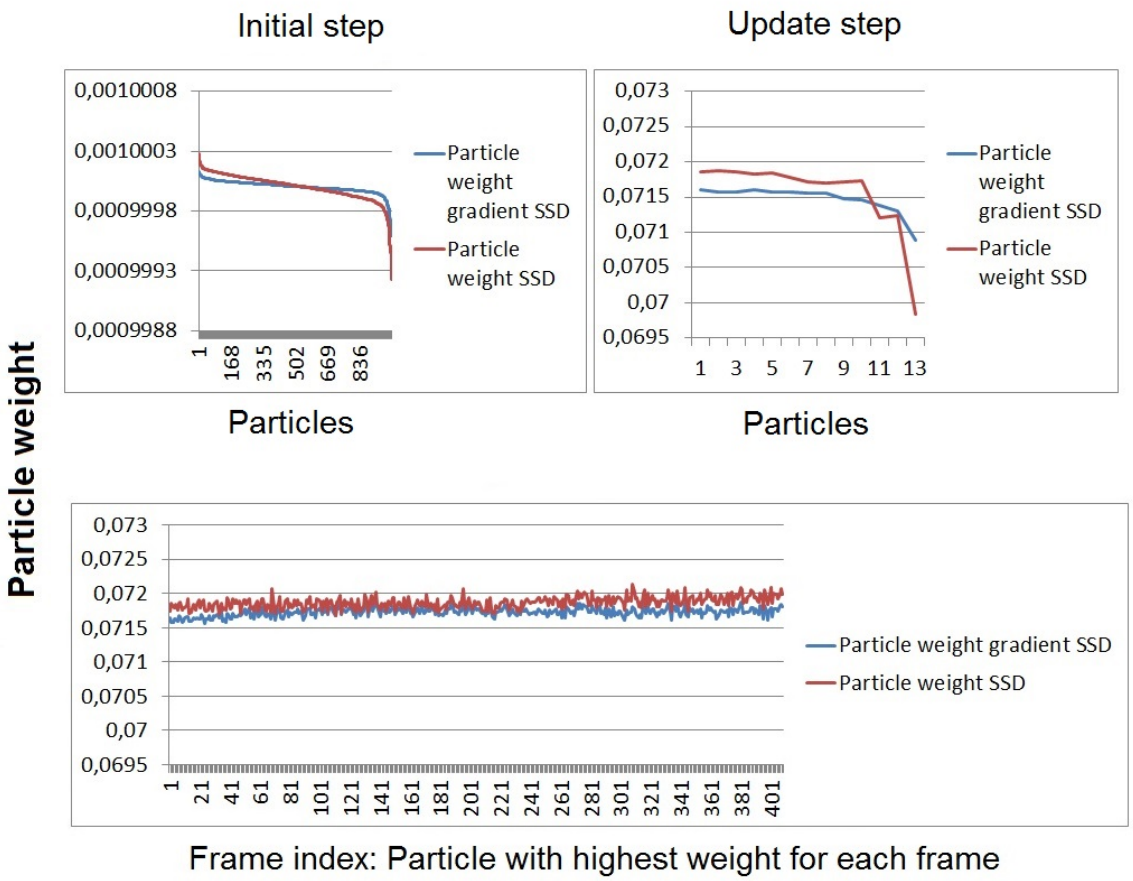

Table 6.6: The weight of particles according to our calculation (equation 4.7). The initial step is computed with 1000 particles (left, top). Results show that there are few particles with high and few particles with low weight. There is no significant difference between the weight of the particles. The update steps (right, top) are computed with 13 particles for each frame.
The particle weight for one of our test videos with 420 frames show that the weight values are very noisy (bottom). These result reflects the jittering effect visible in stationary scenes, where a more probably pose is found in the next update step.

values are close together; therefore, it is hard to estimate if a position is correct.

Our results show that, due to the random nature of the approach, there is no guarantee to find the correct position, because, without a reference value, the algorithm cannot estimate if the most probable pose is the true pose. Furthermore, a reference value would also improve performance. If the correct pose is found, the next update step would require less particles.

6.4 Comparison of Methods

To compute the weight of each particle, we use two different vision-based cost functions, as discussed in section 4.3. As shown in chapter 5, a second cost function requires little additionally computation time. In this section, we discuss the benefit of a second method. Figure 6.2 shows some failure cases for each method. The rendered images on the right have a bad pose estimate; however, both particles have similarly high weight when compared to the input image on the right using our cost functions. In future work we would like to find ways to mitigate this problem.

Figure 6.2 shows the input query from the smartphone (first column), the result for the particle with the highest combined weight (second column), the result for gradient magnitude SSD (third column) and the result for SSD (fourth column).

Figure 6.2: To compute the final weight of a particle according to equation 4.9 we use two methods (SSD and gradient magnitude SSD). The input query from the smartphone (first column). The result for the particle with the highest combined weight (second column). The result for gradient magnitude SSD (third column) and the result for SSD (fourth column).

SSD is less effective if a significant amount of sky is visible, causing the number of pixels under consideration to decrease and leading to a high weight. Gradient magnitude SSD work best when parts of the building and parts of the sky are visible. This is because the highest gradients are at the contours of the buildings. The result of the combined weight provides a very good result.

Both methods work with low resolution images. We use low resolution images to reduce artifacts and to increase performance. Results show that an image width between 50 to 200 pixels and its corresponding camera dependent height deliver a good performance with an accurate pose result.

Figure 6.3 shows a testcase where our algorithm fails to find the correct pose. According to equations 4.4 and 4.5, the average pixel-wise error for both of our cost functions is computed. For both testcases, the most likely pose is computed. Results show that the error for both cost functions is smaller for the more likely camera pose. Results indicate that not enough particles are distributed to find the correct pose. In the current

configuration, only the initial step has a lot of freedom to distribute the particles according to our motion model. For future improvements, we suggest to compute multiple update steps with more freedom and more particles to provide a better accuracy of our algorithm.

Figure 6.3: The left image is the query from a smartphone. The image in the middle shows a case where the initial steps provides a bad result. Compared to a good result (right image), the error for both cost functions is higher. This indicates that not enough particles are distributed to find the correct pose.

6.5 Performance

AR applications are expected to work in real time. Therefore, we tested our approach with different computer systems as shown in table 6.7. The results show that a current desktop computer (first row) can process the video faster than required.

All of our test videos work with 13 particles for each frame, and our desktop reference system is powerful enough to compute this configuration in real time. With seven particles the laptop computer system (second row) works very fast. The problem is that with a low number of particles, testcases with fast motion fail. 25 particles provide slightly better results at much higher costs. Results indicate that a smartphone version is feasible.

Despite the computation time required, computational resources needed for our particle filter algorithm are low. The input data require is about 3 MB for the virtual model and the geo-referenced textures for an area of approximately 100x100 meters. This is an amount of data a smartphone can download easily. Required memory for the program is about 100 MB.

The advantage of the particle filter algorithm is that the number of particles can be changed easily. Therefore, it is possible to adjust the number of particles depending on the used computer system. Our system uses a fixed number of samples during the estimations process which can be very inefficient. One possible solution is to use Kullback-Leibler distance (KLD) sampling. This approach adjusts the number of samples based on the likelihood of observations [17]. In their adaption, a small number of samples is chosen, if the density of the state space is focused on a small part, whereas, if the uncertainty is high, a large number of samples is chosen.

Frames per Second (FPS)	# particles	System
70.1	7	Intel Core i7-4770
		AMD Radeon R9 280X
50.5	13	
37.8	25	
33.3	7	Intel Core i5-3117U
		NVidia Geforce 620M
25.4	13	
20.7	25	

Table 6.7: AR applications are expected to work in real time. Therefore, we tested our approach with different computer systems. For every test, the computer system has to perform the video sequence as fast as possible. The results show that a current desktop computer (first row) can perform the video faster than required. All of our test videos work with 13 particles for each frame. With seven particles, the slow laptop computer system (second row) works very fast, but with a low number of particles testcases with fast motion fail. 25 particles provide slightly better results at much higher costs.

6.6 Environment and Occlusion

A major problem for AR applications is that the real world environment is changing rapidly and dynamically. Therefore, it is necessary for AR to handle dynamic changes in the scene. Our localization approach uses the smartphone sensors as initial data. This is an advantage for difficult cases, where vision-based approach can easily fail. Objects in the foreground occluding the background or different lighting conditions are considered hard tasks. In this section, we discuss the behavior of our approach under these difficult conditions.

For a testcase with lighting conditions significantly different than in our geo-referenced image, we captured a video at early night. The results demonstrates robustness to lighting conditions not similar to the input data (Figure 6.4). The sensors from the smartphone determine the search space for the particle filter. The most likely pose within this search space provides a very good result for this testcase.

In most of our testcases, only small parts of the background are occluded. Major parts of the facades and roofs of the buildings are directly comparable between the images and the virtual model. Figure 6.5 shows an example where objects in the foreground are occluding parts of the virtual model. The query image occludes parts of the background and is far off the correct position. The result demonstrates robustness of our approach when objects are occluding the background.

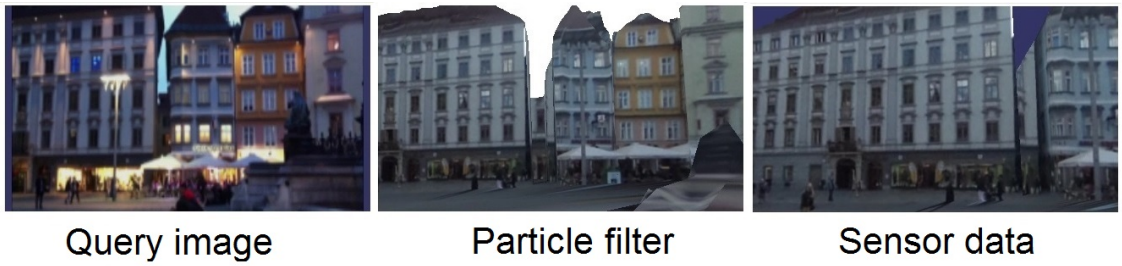

Figure 6.4: Result for video captured early night. The results demonstrates robustness to lighting conditions not similar to the input data. The input data for a pose query is an image from the smartphone (left) and sensor data from GPS and gyroscope. The device sensors are used as an initial guess (right). Our system provides an improvement of the initially guessed camera pose (middle).

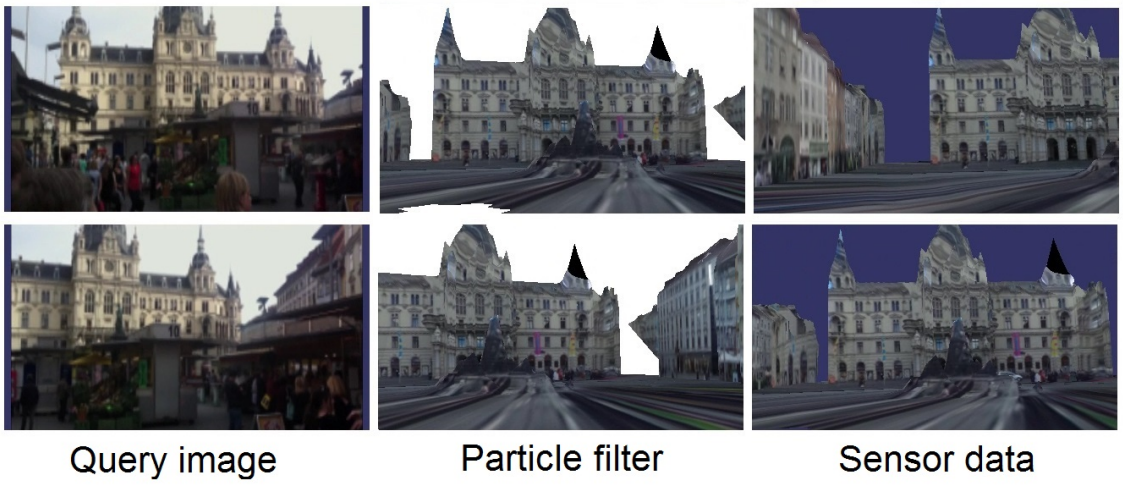

Figure 6.5: In this example, objects in the foreground are occluding parts of the virtual model. The result demonstrates robustness of our approach when objects are occluding the background. The input data for a pose query is an image from the smartphone (left) and sensor data from GPS and gyroscope. The device sensors are used as an initial guess (right). Our system provides an improvement of the initially guessed camera pose (middle).

7 Conclusions and Future work

Our work has demonstrated an accurate outdoor localization system that runs in real time. The outcome of our system improves the camera pose without user assistance. In contrast to previous methods, it is not necessary to move until a baseline is found or to rotate the smartphone. The particle filter provides robustness to device motion, even when exposed to rapid, unpredictable accelerations. Furthermore, the particle filter is flexible in its ability to adapt to different computation environments and different desired levels of performance, accuracy and robustness, by simply adjusting the number of particles and propagation model used.

Another interesting property of our system is that both localization and continuous tracking are achieved with the same method. In addition, we do not make any use of slow-to-compute feature descriptors or large descriptor databases and, instead, only require a simple textured model of the target environment.

A disadvantage of the proposed system is slightly jitter in stationary scenes. This condition occurs when two particles with a slightly different pose have a similar weight. A possible remedy would be to increase the image resolution used when the particle filter converges, to increase the precision of the pose estimate. Another issue is that it is not guaranteed that the initial step finds an accurate first estimation of the pose.

Our results are promising, and a mobile version of the application may be feasible in the near future. Achieving this requires further improvements to the performance of our application. While our current implementation runs the vision algorithms on the CPU, these algorithms can be easily be moved to the GPU. We believe a smart device could be capable of computing approximately 400 particles per second needed for our system to work at 30 Hz.

One avenue of future work is to improve the particle propagation model. A more accurate distribution function could provide even further performance and accuracy, because fewer particles would be necessary. For example, a pedestrian motion model could be used [35]. Another improvement would be view-dependent rendering [13] to increase the accuracy of the image-based rendering and the range of the localization.

Our work hints the possibility that image-based localization based on sparse descriptors may in the long run be replaced by dense GPU-based methods. In particular, the increasing availability of high-quality urban models created with structure-from-motion methods, supported by large geo-data providers such as Google or Microsoft, can be leveraged in GPU-based "tracking by synthesis" approaches such as the one explored in this project. The performance of such an approach is largely independent of the image resolution used and can incorporate many effects such as occlusion and shading, which are difficult to consider in conventional image matching data structures.

In our opinion, the particle filter approach presents an interesting alternative to existing image-based localization approaches and demonstrates that this method can work in practice. We believe that the proposed method is of interest to researchers working on the image-based localization problem and that it could lead to simple but powerful localization pipelines.

A

List of Acronyms

AR Augmented Reality . . . 1

6DoF six degrees of freedom . . . 1

SLAM Simultaneous Localization and Mapping . . . xi

OSG OpenSceneGraph . . . 13

OpenGL Open Graphics Library . . . 14

GPS Global Positioning System . . . iii

WLAN Wireless Local Area Network . . . 1

SfM Structure from Motion . . . 3

SIFT Scale-Invariant Feature Transform . . . 5

FOV Field of View . . . 4

UTM Universal Transverse Mercator . . . xv

MCL Monte Carlo Localization . . . 5

IR Image Retrieval . . . 4

CAD Computer Aided Design . . . 5

BF Brute Force . . . 4

SSD sum of squared distances . . . xiii

PTAM Parallel Tracking and Mapping . . . 7

OpenCV Open Source Computer Vision . . . 13

GPU graphics processing unit . . . 8

CPU central processing unit . . . 13

fovy field of view angle, in radian, in the y direction . . . 14

IMU inertial measurement unit . . . iii

MR Mixed Reality 3

FPS Frames per Second 38

RANSAC Random Sample Consensus 6

RGB RGB color model 8

KLD Kullback-Leibler distance 37

ISMAR International Symposium on Mixed and Augmented Reality ix

Bibliography

[1] Achtelik, M., Achtelik, M., Weiss, S., and Siegwart, R. (2011). Onboard imu and monocular vision based control for mavs in unknown in- and outdoor environments. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3056–3063. (page 3)

[2] Arth, C., Klopschitz, M., Reitmayr, G., and Schmalstieg, D. (2011). Real-time self-localization from panoramic images on mobile devices. In *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*, pages 37–46. (page 1, 2, 4, 7)

[3] Arth, C., Mulloni, A., and Schmalstieg, D. (2012). Exploiting sensors on mobile phones to improve wide-area localization. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 2152–2156. (page 7)

[4] Aubry, M., Russell, B. C., and Sivic, J. (2014). Painting-to-3d model alignment via discriminative visual elements. *ACM Trans. Graph.*, 33(2):14:1–14:14. (page 6)

[5] Burgard, W., Fox, D., Hennig, D., and Schmidt, T. (1996). Position tracking with position probability grids. In *Advanced Mobile Robot, 1996., Proceedings of the First Euromicro Workshop on*, pages 2–9. (page 5)

[6] Cao, S. and Snavely, N. (2014). Graph-based discriminative learning for location recognition. *International Journal of Computer Vision*, pages 1–16. (page 4)

[7] Castle, R., Klein, G., and Murray, D. (2008). Video-rate localization in multiple maps for wearable augmented reality. In *Wearable Computers, 2008. ISWC 2008. 12th IEEE International Symposium on*, pages 15–22. (page 7)

[8] Castle, R. O., Klein, G., and Murray, D. W. (2011). Wide-area augmented reality using camera tracking and mapping in multiple regions. *Comput. Vis. Image Underst.*, 115(6):854–867. (page 7)

[9] Coors, V., Huch, T., and Kretschmer, U. (2000). Matching buildings: pose estimation in an urban environment. In *Augmented Reality, 2000. (ISAR 2000). Proceedings. IEEE and ACM International Symposium on*, pages 89–92. (page 5)

[10] Cummins, M. and Newman, P. (2010). Appearance-only slam at large scale with fab-map 2.0. *The International Journal of Robotics Research*. (page 7)

[11] Davison, A. (2003). Real-time simultaneous localisation and mapping with a single camera. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1403–1410 vol.2. (page 7)

[12] Davison, A., Mayol, W., and Murray, D. (2003). Real-time localization and mapping with wearable active vision. In *Mixed and Augmented Reality, 2003. Proceedings. The Second IEEE and ACM International Symposium on*, pages 18–27. (page 7)

[13] Debevec, P. E., Yu, Y., and Borshukov, G. (1998). Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping. *Rendering Techniques*, pages 105–116. (page 41)

[14] Dong, Z., Zhang, G., Jia, J., and Bao, H. (2009). Keyframe-based real-time camera tracking. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1538–1545. (page 3)

[15] Donoser, M. and Schmalstieg, D. (2014). Discriminative feature-to-point matching in image-based localization. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 516–523. (page 4)

[16] Engel, J., Schops, T., and Cremers, D. (2014). LSD-SLAM: Large-Scale Direct Monocular SLAM. In *European Conference on Computer Vision*, Zurich, Switzerland. (page 8)

[17] Fox, D. (2001). Kld-sampling: Adaptive particle filters. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *NIPS*, pages 713–720. MIT Press. (page 37)

[18] Fox, D., Burgard, W., Dellaert, F., and Thrun, S. (1999). Monte carlo localization: Efficient position estimation for mobile robots. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, pages 343–349, Menlo Park, CA, USA. American Association for Artificial Intelligence. (page 5)

[19] Harris, C. (1993). Active vision. chapter Tracking with Rigid Models, pages 59–73. MIT Press, Cambridge, MA, USA. (page 5)

[20] Irschara, A., Zach, C., Frahm, J., and Bischof, H. (2009a). From structure-from-motion point clouds to fast location recognition. In *Proceedings of Computer Vision and Pattern Recognition (CVPR)*. (page 3, 5)

[21] Irschara, A., Zach, C., Frahm, J.-M., and Bischof, H. (2009b). From structure-from-motion point clouds to fast location recognition. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2599–2606. (page 3)

[22] Jensfelt, P. and Kristensen, S. (2001). Active global localization for a mobile robot using multiple hypothesis tracking. *Robotics and Automation, IEEE Transactions on*, 17(5):748–760. (page 5)

[23] Klein, G. and Murray, D. (2006). Full-3d edge tracking with a particle filter. In *British Machine Vision Conference Proc 17th*. (page 5, 6, 19)

[24] Klein, G. and Murray, D. (2007). Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. (page 3, 7)

[25] Klein, G. and Murray, D. (2009). Parallel tracking and mapping on a camera phone. In *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*, pages 83–86. (page 7)

[26] Leonard, J. and Durrant-Whyte, H. (1991). Mobile robot localization by tracking geometric beacons. *Robotics and Automation, IEEE Transactions on*, 7(3):376–382. (page 5)

[27] Li, Y., Snavely, N., and Huttenlocher, D. P. (2010). Location recognition using prioritized feature matching. In Daniilidis, K., Maragos, P., and Paragios, N., editors, *ECCV (2)*, volume 6312 of *Lecture Notes in Computer Science*, pages 791–804. Springer. (page 3, 5)

[28] Lim, H., Sinha, S. N., Cohen, M. F., and Uyttendaele, M. (2012). Real-time image-based 6-dof localization in large-scale environments. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2012)*. (page 3)

[29] Lothe, P., Bourgeois, S., Dekeyser, F., Royer, E., and Dhome, M. (2009). Towards geographical referencing of monocular slam reconstruction using 3d city models: Application to real-time accurate vision-based localization. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2882–2889. (page 7)

[30] Lowe, D. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110. (page 5)

[31] Marchetti, L., Grisetti, G., and Iocchi, L. (2007). A comparative analysis of particle filter based localization methods. In Lakemeyer, G., Sklar, E., Sorrenti, D., and Takahashi, T., editors, *RoboCup 2006: Robot Soccer World Cup X*, volume 4434 of *Lecture Notes in Computer Science*, pages 442–449. Springer Berlin Heidelberg. (page 5)

[32] Meier, L., Tanskanen, P., Fraundorfer, F., and Pollefeys, M. (2011). Pixhawk: A system for autonomous flight using onboard computer vision. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2992–2997. (page 3)

[33] Newcombe, R. A., Lovegrove, S., and Davison, A. (2011). Dtam: Dense tracking and mapping in real-time. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2320–2327. (page 7)

[34] Nister, D. and Stewenius, H. (2006). Scalable recognition with a vocabulary tree. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, CVPR '06, pages 2161–2168, Washington, DC, USA. IEEE Computer Society. (page 4)

[35] Quigley, M., Stavens, D., Coates, A., and Thrun, S. (2010). Sub-meter indoor localization in unmodified environments with inexpensive sensors. In *IROS*, pages 2039–2046. IEEE. (page 41)

[36] Reitmayr, G. and Drummond, T. (2006). Going out: robust model-based tracking for outdoor augmented reality. In *Mixed and Augmented Reality, 2006. ISMAR 2006. IEEE/ACM International Symposium on*, pages 109–118. (page 4, 5)

[37] Reitmayr, G. and Drummond, T. (2007). Initialisation for visual tracking in urban environments. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 161–172. (page 5)

[38] Reitmayr, G., Langlotz, T., Wagner, D., Mulloni, A., Schall, G., Schmalstieg, D., and Pan, Q. (2010). Simultaneous localization and mapping for augmented reality. In *Ubiquitous Virtual Reality (ISUVR), 2010 International Symposium on*, pages 5–8. (page 1, 7)

[39] Robertsone, D. and Cipolla, R. (2004). An image-based system for urban navigation. In *Proc. BMVC*, pages 84.1–84.10. doi:10.5244/C.18.84. (page 3)

[40] Rosten, E. and Drummond, T. (2005). Fusing points and lines for high performance tracking. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1508–1515 Vol. 2. (page 5)

[41] Royer, E., Lhuillier, M., Dhome, M., and Lavest, J.-M. (2007). Monocular vision for mobile robot localization and autonomous navigation. *Int. J. Comput. Vision*, 74(3):237–260. (page 3)

[42] Sattler, T., Leibe, B., and Kobbelt, L. (2011a). Fast image-based localization using direct 2d-to-3d matching. In *Proceedings of International Conference on Computer Vision (ICCV)*. (page 3, 5)

[43] Sattler, T., Leibe, B., and Kobbelt, L. (2011b). Fast image-based localization using direct 2d-to-3d matching. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 667–674. (page 4)

[44] Schindler, G., Brown, M., and Szeliski, R. (2007). City-scale location recognition. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–7. (page 3, 4)

[45] Schoeps, T., Engel, J., and Cremers, D. (2014). Semi-dense visual odometry for ar on a smartphone. In *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*, pages 145–150. (page 8)

[46] Sibbing, D., Sattler, T., Leibe, B., and Kobbelt, L. (2013). Sift-realistic rendering. In *3D Vision - 3DV 2013, 2013 International Conference on*, pages 56–63. (page 6)

[47] Snavely, N., Seitz, S., and Szeliski, R. (2006). Photo tourism: exploring photo collections in 3d. *ACM Transactions on Graphics (TOG), 25(3):835-846.* (page 4)

[48] Takacs, G., Xiong, Y., Grzeszczuk, R., Chandrasekhar, V., chao Chen, W., Pulli, K., Gelfand, N., Bismpigiannis, T., and Girod, B. (2008). Outdoors augmented reality on mobile phone using loxel-based visual feature organization. In *In Proceeding of ACM international conference on Multimedia Information Retrieval*, pages 427–434. (page 4)

[49] Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic robotics.* MIT press. (page 2, 17)

[50] Torii, A., Sivic, J., and Pajdla, T. (2011). Visual localization by linear combination of image descriptors. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 102–109. (page 4)

[51] Vaca-Castano, G., Zamir, A., and Shah, M. (2012). City scale geo-spatial trajectory estimation of a moving camera. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 1186–1193. (page 4)

[52] Ventura, J., Arth, C., Reitmayr, G., and Schmalstieg, D. (2014). Global localization from monocular slam on a mobile phone. *IEEE Transactions on Visualization and Computer Graphics*, 20(4):531–539. (page 2, 4, 7)

[53] Ventura, J. and Hollerer, T. (2012). Wide-area scene mapping for mobile visual tracking. In *Mixed and Augmented Reality (ISMAR), 2012 IEEE International Symposium on*, pages 3–12. (page 1, 4, 7)

[54] Wagner, D., Reitmayr, G., Mulloni, A., Drummond, T., and Schmalstieg, D. (2010). Real-time detection and tracking for augmented reality on mobile phones. *Visualization and Computer Graphics, IEEE Transactions on*, 16(3):355–368. (page 3)

[55] Williams, B., Klein, G., and Reid, I. (2007). Real-time slam relocalisation. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. (page 7)

[56] Zamir, A. and Shah, M. (2010). Accurate image localization based on google maps street view. In *Computer Vision - ECCV 2010*, volume 6314, pages 255–268. Springer Berlin Heidelberg. (page 4)

[57] Zhang, W. and Kosecka, J. (2006). Image based localization in urban environments. In *3D Data Processing, Visualization, and Transmission, Third International Symposium on*, pages 33–40. (page 4)